PROGRAMOZÁS TEHETSÉGGONDOZÁS FELFEDEZTETŐ MÓDSZERTANNAL

DOKTORI ÉRTEKEZÉS

NIKHÁZY LÁSZLÓ

2025

PROGRAMOZÁS TEHETSÉGGONDOZÁS FELFEDEZTETŐ MÓDSZERTANNAL

NIKHÁZY LÁSZLÓ

EÖTVÖS LORÁND TUDOMÁNYEGYETEM, INFORMATIKA DOKTORI ISKOLA

INFORMATIKA SZAKMÓDSZERTAN DOKTORI PROGRAM

ISKOLAVEZETŐ: DR. HORVÁTH ZOLTÁN

EGYETEMI TANÁR, INTÉZETVEZETŐ

PROGRAMVEZETŐ: DR. ILLÉS ZOLTÁN

HABILITÁLT EGYETEMI DOCENS

TÉMAVEZETŐ: DR. ZSAKÓ LÁSZLÓ †

NYUGALMAZOTT TANSZÉKVEZETŐ EGYETEMI DOCENS

MÉDIA- ÉS OKTATÁSINFORMATIKAI TANSZÉK

DOI: 10.15476/ELTE.2025.325

BUDAPEST, 2025

Köszönetnyilvánítás

Köszönetemet szeretném kifejezni témavezetőmnek, Dr. Zsakó László tanár úrnak, aki mindenben segített, és végig kísérte a doktori kutatásomat. Mélyen megrendített engem és mindannyiunkat 2024 májusában bekövetkezett halála. Nagyon sajnálom, hogy ezt az értekezést teljes terjedelmében már nem láthatta.

Hálás vagyok a teljes családomnak a sok támogatásért, különösen feleségemnek, Lisethnek a türelméért és tanácsaiért, valamint Anyukámnak a disszertációm átolvasásáért és sok apró nyelvhelyességi hiba javításáért.

Köszönetet mondok Erdősné dr. Németh Ágnesnek a sok éves együttműködés mellett az értekezés szakmai szemmel történő áttekintéséért és módosítási javaslataiért is.

Mindig hálás leszek Pósa Lajosnak, aki inspirálta a kutatásomat. Rengeteget köszönhetek a hétvégi matematikatáboroknak, amelyekben diákként, segítőként és táborvezetőként is részt vettem. A táborvezetői munkámban mindvégig segített és mentorált, aminek köszönhetően kész lettem arra, hogy saját tehetséggondozó táborokat indítsak programozás terén.

Köszönettel tartozom a sok fiatal segítőmnek, akik táboraimban aktívan közreműködnek a tanításban és szabadidős programokban is, nélkülük nem jöhettek volna létre a ProgTáborok.

Végül köszönöm a tanítványaimnak a sok együtt töltött időt és közös munkát, amely számomra is rengeteg örömet hozott és hoz a jövőben is, ez hatalmas motivációt ad a munkámban.

Tartalomjegyzék

1. Bevezetés

1.1. Motiváció

Diákként nagy hatással voltak rám Pósa Lajos matematikatáborai. A táborokat a felfedeztető módszertannal való tanítás teszi különlegessé, amely a matematika tehetséggondozásban kiválóan működik (Juhász, 2019). Felnőttként bekapcsolódtam a matematikatáborok vezetésébe, de közben mindig foglalkoztatott, hogy informatikából lehetne-e hasonló tehetséggondozó programot létrehozni, vagyis algoritmusokat, adatszerkezeteket és programozást tanítani felfedeztető módszertannal. Doktori kutatásom céljaként azt tűztem ki, hogy a matematikatanításban alkalmazott Pósa-módszert adaptálva kifejlesszek olyan felfedeztető módszertant és tananyagot programozás tehetséggondozásra, ami a gyakorlatban is működik. E sorok írásakor büszkén elmondhatom, hogy már 7 különböző évfolyamon több, mint száz tehetséges diáknak volt hasonló élményekben része a programozástáboraimban, mint nekem diákként a matematikatáborokban.

1.2. A dolgozat felépítése

A 2. fejezetben a téziseket fogalmazom meg, ezt követően öt fő részben mutatom be a kutatási eredményeimet. A 3. fejezetben áttekintem a matematika tanításban alkalmazott Pósa-módszer legfontosabb elemeit, és ezeket adaptálva, kiegészítve bemutatom azt a felfedeztető módszertant algoritmikus programozás tanítására, amellyel a táboraimban kísérleteztem.

A 4. fejezetben ismertetem a programozás tehetséggondozó táborokhoz fejlesztett tantervet és tananyagot, amelyet a Pósa-módszer tananyagtervezési keretrendszerével készítettem (Katona, 2022), azaz programozási versenyfeladatokat felhasználva hoztam létre úgynevezett problémaszálak hálózatát. A terjedelmi keretek miatt a több, mint 600 feladatot tartalmazó feladatgyűjteményt az 1. számú mellékletben adom meg.

Az 5. fejezetben bemutatom a táborokat, amelyekben a gyakorlatban is használtam a kidolgozott szisztémát és tananyagokat. Részletesen beszámolok a felméréseim eredményeiről, amelyekkel a módszerem sikerességét mértem.

A 6. fejezetben elsőként a felfedeztető módszertanú algoritmikus programozás tanítás online, virtuális térben történő megvalósítását, illetve a hozzá használt eszköztárat mutatom be. Másrészt, az online oktatás révén kivitelezhető, heti-kétheti rendszerességű országos

tehetséggondozó szakkörök előnyeiről és hátrányairól írok, az erről készített felmérésemet ismertetem. A szakköri rendszer nem szerepelt az eredeti célkitűzéseim között, de egy rendkívül hasznos új lehetőség.

Végül, a 7. fejezetben bemutatok egy egyetemi kurzus két csoportjával készített összehasonlító elemzést, amelyben Kirschner és társainak (Kirschner, Sweller, & Clark, 2006) a felfedeztető oktatásról megfogalmazott kritikáira reflektálok.

A 8. fejezetben röviden összefoglalom az eredményeimet és visszatérek a tézisek igazolására, majd a felhasznált irodalmak jegyzéke, weboldalak elérhetőségeinek felsorolása, publikációs lista és a mellékletek következnek.

1.3. Az algoritmikus programozás fogalma

Egyre több szakirodalom használja az „algoritmikus programozás” kifejezést, viszont pontos meghatározást nem találtam egyikben sem. Mivel az értekezésemnek központi témája, ezért fontosnak tartom leírni, hogy mit értünk alatta.

Az *algoritmikus programozás* a hatékony algoritmusok tervezésének és megvalósításának gyakorlata jól definiált számítási problémák megoldására. Magában foglalja a matematikai alapok, algoritmustervezési technikák és adatszerkezetek alkalmazását olyan programok kidolgozására, amelyek optimalizálják a tár- és műveletigényt.

Az algoritmuselmélettel összevetésben egy kulcsfontosságú eleme az, hogy a kigondolt megoldásokat implementáljuk is valamilyen programozási nyelven, és teszteljük a program helyességét és hatékonyságát. Ezzel kapcsolatot teremtünk az elmélet és gyakorlat között. A tesztelés általában feketedoboz tesztelés bemenet-kimenet párokkal, meghatározott memória- és időlimitek mellett, amivel meg lehet győződni az algoritmus megfelelő tár- és időkomplexitásáról. Az algoritmikus programozás területe szorosan kapcsolódik a versenyprogramozáshoz, viszont fontosnak tartom kiemelni, hogy versenyzés nélkül is lehet művelni. Másfelől pedig vannak nem algoritmikus jellegű programozási versenyek is.

Az algoritmikus programozás célja tehát egy-egy matematikailag jól meghatározott feladatot megoldó algoritmus implementálása, amelynek segítségével meg lehet győződni a megoldás helyességéről. Így általában rövid, néhányszor tíz, de legfeljebb párszáz soros programok születnek, amelyekben nem célkitűzés a továbbfejleszthetőség és újrafelhasználhatóság – ellentétben például az objektum-orientált programozással. A programok bemenete és kimenete általában meghatározott formátumot követő karakteres adatsor, tehát nem elsődleges a

felhasználói interakció, és nem készítünk felhasználói felületet – szemben mondjuk az alkalmazásfejlesztéssel és webprogramozással. Ugyanakkor természetesen előfordulhat, hogy egy-egy nagyobb rendszer részeként adódik olyan feladat, amire algoritmikus programozási alapelvekkel kell megoldást adni, majd integrálni a rendszerbe, tehát előfordulhat más programozási paradigmákkal történő együttes alkalmazása.

1.4. Kutatási módszerek

1.4.1. Akciókutatás

Mivel az általam kidolgozott szisztémát és tananyagot a gyakorlatba ültetve saját magam teszteltem és lépésről lépésre fejlesztettem tovább, ezért az akciókutatás módszerét tartottam a legalkalmasabbnak a kutatásomhoz. Az akciókutatás egy participatív és ciklikus kutatási módszer, amelynek célja a problémák megoldása és a gyakorlat javítása a kutatásban részt vevő személyek közvetlen bevonásával (Reason & Bradbury, 2001).

A kutatásom kifejezetten participatív jellegű volt abból a szempontból, hogy kísérleti táboraim diákjai aktívan segítették a kutatómunkámat, minden tábor után kértem visszajelzéseket tőlük, és sokszor közösen reflektáltunk arra, hogy szakmai és szervezési szempontból mit lehetne másképpen, jobban csinálni.

Kétféle ciklikusság jellemezte a munkámat. Egyrészt ugyanazzal a csoporttal újra és újra találkoztunk több táboron keresztül, így a didaktikai módszereket táborról táborra tudtam csiszolni, a változásokat a diákok is érzékelhették. Másrészt ugyanazokat a tananyagokat később a többi, fiatalabb csoporttal is megcsináltam, immár továbbfejlesztett tartalommal. Így a szakmai részt, a tantervet és feladatsorokat is tudtam egyre jobbá és komplettebbé tenni.

A táborok révén mindvégig gyakorlati fókuszú volt a kutatásom, ami az akciókutatás további gyakori ismertetőjele. Természetesen sok háttérmunkát kellett végeznem ahhoz, hogy elindíthassam a táborokat, számos forrást áttekintve meg kellett terveznem a tananyagot és feltérképeznem az összefüggéseket. Számtalan feladatot át kellett néznem és elemeznem, hogy megalkothassam a Pósa-módszerre jellemző feladathálózatot programozásból, de amint lefektettem a rendszer alapjait, elkezdtem gyerekekkel foglalkozni. Ez a gyakorlati fókusz és gyors visszacsatolás lehetőséget adott arra, hogy igazoljam a módszerek működését, és javítsam a hiányosságokat bennük.

1.4.2. Kérdőívek

A táboraimban, szakköreimen és még az egyetemi foglalkozásaim alatt is legtöbbször kérdőíven gyűjtöttem visszajelzéseket. A kérdőívekkel az volt a célom, hogy a teljes csoportra vonatkozóan képet kapjak arról, hogy a diákokra milyen hatással vannak a módszereim, és mi a véleményük róluk. Felhasználtam a kérdőíven kapott válaszokat arra is, hogy az alkalmak között javítsak a tanítási gyakorlatomon, és arra is, hogy a válaszok elemzésével tendenciákat figyeljek meg és hipotéziseket igazoljak.

A kérdőívek előnye, hogy lehetővé teszik az anonimitást, ami csökkentheti a válaszok torzítását, és bátrabban fogalmazhatnak meg kritikákat bennük. Az adatok strukturált formátumban gyűlnek, amelynek segítségével összehasonlíthatóak lesznek a válaszok. Egyaránt alkalmaztam a kérdőíveken zárt és nyitott kérdéseket is. A zárt kérdések korlátozhatják a válaszadók kifejezőképességét, ugyanakkor jól kiértékelhető és jól ábrázolható adatokat kapunk. A nyitott kérdések jóval kevésbé strukturált adatokat eredményeznek, viszont általuk sokkal több olyan visszajelzést kaphatunk, amelyek alapján lehet fejleszteni a gyakorlatunkon. A nyitott kérdésekre adott válaszokat is elemeztem összesítve néhány esetben, mégpedig úgy, hogy sokszor előforduló egyező motívumokat kerestem a válaszokban, és ez alapján csoportosítottam őket.

A kérdőívekkel azt kívántam elérni, hogy nagy mennyiségű mérési adattal tudjam igazolni, hogy a céljaim megvalósulnak. Nem tartottam megvalósíthatónak azt, hogy a módszeremet klasszikus társadalomtudományi kísérleti metodikával, azaz kontrollcsoport és kísérleti csoport vizsgálatával összemérjem más tanítási módszerrel, több okból sem. A kiemelkedően tehetséges gyerekekből kevés van és olyan különbségek vannak közöttük, hogy nem lehetne igazán reprezentatív kutatást végezni. Másrészt, nem szeretnék egy gyereket sem kihagyni abból az élményből, amit a felfedező tanulás tud nyújtani nekik, nem tartottam volna helyénvalónak, hogy egy kontrollcsoportot kevésbé élvezetes módszerrel tanítsak. Harmadrészt, az oktatás egyes hatásai csak nagyon hosszú távon mutatkoznak meg és az én tevékenységem mellett annyi más effektus éri a diákokat, hogy szinte lehetetlen ezeket a hosszú távú eredményeket egyértelműen valamely módszertannak vagy konkrét foglalkozásnak tulajdonítani. Egy kisebb volumenű összehasonlító tanulmányt végeztem egyetemistákkal, amit a 7. fejezetben be is mutatok, viszont azzal csak egy aspektusát vizsgáltam a felfedeztető oktatásnak, nem a teljes metodikámat alkalmaztam és nem is az általam kidolgozott tananyagon.

2. Tézisek

I. Az algoritmikus programozás tanítása tehetséges diákoknak megvalósítható felfedeztető módszertannal, a matematika oktatásban sikeres Pósa-módszer alapvető szemléletének megtartásával, a módszertan kis módosításával. Megtarthatók a Pósa-módszernek olyan fontos elemei, mint például a nyitott kérdésfeltevés, a csoportmunka, az oktató segítő szerepe, és a feladatok egymásra épülése, összefonódása. A tanulási folyamat kiegészül egy kivitelezési lépéssel, ugyanakkor a csatlakozó kérdéseknek nehezebb teret adni. A számítógépes problémamegoldás és a matematika különbségeiből adódóan kevésbé jelenik meg a kutatás alapú tanulás, és erősebben a probléma alapú tanulás.

II. A programozási versenyeken szereplő feladatok alkalmasak arra, hogy problémaszálak hálózatát (web of problem threads) alkossunk belőlük, és ezzel megteremtsük a Pósa-módszer alkalmazásában központi szerepet játszó tananyagot a módszertan elméleti kereteinek megfelelő rendszerben. Ennek igazolására egy teljes tantervet hoztam létre a szükséges témák és kapcsolódási pontok azonosításával, valamint témakörönkénti feladatszálak megadásával. A tantervben 57 témakör és több, mint 600 feladat szerepel, és az irányított gráf formátumnak köszönhetően rugalmasan alakítható a tanítási sorrend.

III. A hétvégi programozástáborokban a felfedeztető módszertannal, csoportos problémamegoldással megvalósuló tanulás egyszerre nyújt élvezetet és fejlődést a tehetséges diákoknak. A táborokon átívelő tananyaggal és homogén életkorú csoportokkal a hosszú távú tanulás és a közösségépítés is megvalósítható.

IV. Online oktatásban is megvalósítható a kollaborációra és önálló munkára épülő felfedeztető tanítás a megfelelő szoftvereszközök és platformok használatával. Az online szervezés túlmutat a kényszermegoldáson, a hátrányok mellett számos előnye is van. A legfontosabb ezek közül az, hogy több diák számára válik elérhetővé ugyanaz a szakkör a földrajzi elhelyezkedéstől függetlenül, ezáltal homogénebb tudásszintű csoportok jöhetnek létre és a diákok a képességeiknek megfelelő oktatást kaphatják.

V. Megfelelő nehézségű feladatok esetén a diákok több fejlődést tapasztalhatnak az önálló problémamegoldás útján, mint a tanár által vezetett, mintaszerű közös megoldás figyelemmel kísérése során, még akkor is, ha az utóbbi módszerrel több feladat feldolgozása történik meg ugyanannyi idő alatt.

3. A Pósa-módszer alkalmazása a programozás oktatásban

3.1. A felfedeztető tanítás

A felfedeztető tanítás kifejezés olyan pedagógiai módszerekre utal, amelyek során a diákok egy adott téma felfedezésén keresztül tanulnak. Három fő előnyét lehet azonosítani: mélyebb ismeretek megszerzése, kognitív készségek fejlődése, a diákok érdeklődésének fokozása (Bruner, 1961). A mély tudás ebben az értelemben azt jelenti, hogy a megtanult információk, fogalmak és módszerek nagyon erősen gyökereznek a hosszú távú memóriában, így az illető magasabb szinten érti a témát, és sikeresebben tudja alkalmazni ezt a tudást új helyzetekben. A tanulási folyamat során a tanulók az idő nagy részét aktív egyéni vagy csoportos munkával töltik, tudásukat ezeken a tevékenységeken keresztül saját maguk építik fel. Ez hatalmas, változatos formában használt agyi kapacitásokat igényel, amelyek révén a kognitív készségek nagymértékben fejlődnek. A tanulók öröme kulcsfontosságú érték az egész folyamatban, hiszen motiválja őket a további részvételre és növeli kitartásukat.

A diákok felfedező tanulását leggyakrabban olyan helyzetekben tapasztaljuk, amikor egy érdekes problémát kell megoldaniuk, miközben korábbi tapasztalataikra és előzetes tudásukra támaszkodhatnak. Vagyis a tanulási folyamat teljes egésze problémamegoldások sorozataként értelmezhető, amelyben a tanulók rendre különböző kihívásokkal néznek szembe. A problémamegoldás iránti erőfeszítéseik mögött számos kognitív folyamat zajlik. Eközben felfedezik azokat az alapvető igazságokat és elveket, amelyeket tanítani kívánunk nekik. Ezáltal a diákok saját tudásukat építik fel, míg az oktató elsődleges szerepe ennek a folyamatnak az irányítása és támogatása. Továbbá fontos szerepe van a tanárnak abban is, hogy kialakítsa a tanulókban azt a készséget, hogy gondolkodásukat problémamegoldási célokra használják, amely kulcsfontosságú az életben való sikerekhez.

A felfedeztető tanítás gyökerei Rousseauhoz, Pestalozzihoz és Deweyhez köthetők. John Dewey, aki az amerikai pedagógiai reform egyik legjelentősebb alakja volt, hangsúlyozta: „Szoros és nélkülözhetetlen kapcsolat van a konkrét tapasztalás folyamata és az oktatás között.” (Dewey, 1933). Neves pszichológusok és tanuláskutatók szintén támogatták ezt az oktatási megközelítést, például Seymour Papert így fogalmazott: „Nem tudsz megtanítani másoknak mindent, amit tudniuk kell. A legtöbb, amit tehetsz, hogy felkészíted őket arra, hogy amikor szükségük lesz valamilyen tudásra, akkor majd tudják, hogy honnan és hogyan szerezzék meg azt.” (Papert, 1996).

A felfedeztető tanítás erősen kapcsolódik a konstruktivista tanuláselmélethez, amely arra a feltételezésre támaszkodik, hogy az emberek a mentális tevékenységek során építik fel tudásukat. A tanulókat értelmet kereső organizmusoknak tekintik, és tapasztalataikra reflektálva alakítják ki a körülöttük lévő világ értelmezését. A modern felfedeztető tanítási módszerek a konstruktivizmusra épülnek, és sokan Brunert tekintik a felfedeztető tanítás atyjának, aki a könyvében (Bruner, 1979) így ír erről: „A felfedeztető tanításra hangsúlyt fektetve pont azt a hatást érjük el, hogy a tanuló maga is konstrukcionista lesz, a saját tapasztalatait rendszerezi nemcsak oly módon, hogy szabályosságokat és kapcsolatokat fedez fel, hanem képes kiválasztani azokat az információkat, amelyek az adott helyzetben hasznosak."

Tehát mindannyian saját szabályokat és mentális modelleket alkotunk, melyek segítségével értelmet adunk annak, amit érzékelünk. A konstruktivista elmélet szerint a tanulás folyamata során ezeket a mentális modelleket igazítjuk az új tapasztalatokhoz, hogy azokkal összhangban legyenek. Bár ezek az értelmi modellek kezdetben lehet, hogy nem teljesen tükrözik a valóságot (például a kisgyermekek naiv elméletei), de idővel egyre összetettebbek és valósághűbbek lesznek. A konstruktivista ismeretelméletnek négy alapfeltevése van (Piaget, 1971):

- A tudást nem átadjuk, hanem létrehozzuk.
- Az előzetes tudás hatással van a tanulási folyamatra.
- A megértés lokális, nem globális.
- A hasznos tudás megszerzéséhez erőfeszítésekre van szükség, meghatározott céllal.

3.2. A felfedeztető tanítás típusai

Neves oktatók a 20. század második felétől kezdve alakították ki a konstruktivista elmélet gyakorlatba ültetésének több formáját a felfedeztető tanítás területén.

Az egyik legkézenfekvőbb alkalmazás a természettudományok területén a *kísérlet alapú tanulás*, amelyben a diákokat kísérletek sora elé állítjuk, hogy saját tapasztalataikon keresztül megismerjék és értelmezzék a természeti jelenségeket. A kísérletek során a diákok nemcsak megfigyelik a jelenségeket, hanem részt vesznek azok előidézésében és megértésében is, ami segíti őket abban, hogy saját tudásukat konstruálják. Példaként, van Joolingen így értelmezi a felfedező tanulást (van Joolingen, 1999): „A felfedező tanulás egy olyan tanulási forma, amelyben a diákok hozzák létre a tudást egy adott területen való kísérletezéssel, és szabályokat

alkotnak meg a kísérletek eredményei alapján. Ennek a fajta tanulásnak az alapelve, hogy mivel a tanulók maguk tervezik meg a kísérleteiket és ők következtetik ki az adott tudományterület szabályait, ezért tulajdonképpen ők építik, konstruálják a saját tudásukat. A konstruktív tevékenységek miatt feltételezhetjük, hogy magasabb szinten fogják érteni az adott területet, mintha az információkat csak előadná egy tanár."

A *kutatás alapú tanulásnak* öt fő lépése van Bishop és társai szerint (Bishop, Bertram, & Lunsford, 2004): a kérdésfeltevés, vizsgálat, alkotás, megbeszélés és visszatekintés. A folyamat diákközpontú, gyakran a diákok irányítása alatt áll, és egy kör vagy spirál modell segítségével jellemezhető. Célja minden esetben az aktív tanulás elősegítése, különösen az egyén saját érdeklődési körére építve. Minden kérdés új ötleteket vagy további kérdéseket generálhat, így folyamatosan bővül a tanulás és a kutatás területe.

Számunkra különösen érdekes a *probléma alapú tanulás*, amely lehetővé teszi a diákok számára, hogy valóságos környezetben alkalmazzák a tanult ismereteket és készségeket, miközben fejlesztik a problémamegoldó képességeiket és a kollaboratív munkavégzést, hiszen a diákok csoportokban dolgoznak, való életből vett problémákon. Finkle és Torp így határozta meg a módszerüket (Finkle & Torp, 1995): „Egy olyan tantervfejlesztési és oktatási rendszer, amely párhuzamosan fejleszti a problémamegoldási stratégiákat, diszciplináris tudást és képességeket, a tanulókat aktív problémamegoldó szerepbe helyezéssel, amelyben egy való világbeli problémát tükröző nem jól strukturált feladattal néznek szembe." A programozás tehetséggondozásban alkalmazott módszerünk erősen kapcsolódik ehhez, viszont nagyon fontos és hatalmas különbség, hogy jól definiált feladatokkal dolgozunk.

A matematikatáborokban alkalmazott Pósa-módszer erősen kötődik az *irányított felfedező tanuláshoz*, amely meghatározó alakjának Dr. Charles E. Wales-t, a West Virginia University professzorát tekintik. Ebben a tanulási módszerben a diákokat egy oktató vagy oktató szoftver vezeti végig a folyamaton. Bibergall szerint ezt a tanulási formát a konvergens gondolkodás jellemzi (Bibergall, 1966): „A pedagógus állítások vagy kérdések sorozatát dolgozza ki, amelyek lépésről lépésre irányítják a tanulót, aki felfedezések sorát teszi, amelyek egy előre meghatározott célhoz vezetnek". Az oktató tehát arra ösztönzi a tanulót, hogy aktívan vizsgálja a témát, és saját maga jöjjön rá a megfelelő válaszra. Mosston (Mosston, 1972) tíz kognitív műveletet határoz meg, amelyek a tanuló aktív vizsgálata során történhetnek: elemzés, felismerés, szintézis, összehasonlítás és ellentétezés, következtetések levonása, hipotézisek kialakítása, memorizálás, kérdések megfogalmazása, feltalálás, felfedezés. Azáltal, hogy a diák

aktívan részt vesz a folyamatban, mélyebben megérti és jobban megjegyzi a tananyagot. Mosston azonban figyelmeztet arra, hogy „a felfedező tanulás nem lehetséges, ha megadjuk a válaszokat”, és rámutat e tanítási módszer bizonyos korlátjaira, például inkább egyéni, nem pedig csoportos felhasználásra tervezték, illetve könnyen lehet félrevezető, mivel az oktató befolyásolhatja a tanulói magatartást.

Borthick és Jones módszere, a *kollaboratív felfedező tanulás* csoportmunkával ötvözi a felfedeztető tanítást (Borthick & Jones, 2000): „A felfedező tanulásban a résztvevők megtanulnak felismerni egy problémát, jellemzik, hogy egy megoldás milyen lenne, releváns információkat keresnek, kialakítanak egy megoldási módszert, és kivitelezik azt. A kollaboratív felfedező tanulásban a résztvevők egy közösségbe épülve oldanak meg problémákat együtt.” A tehetséggondozó táborainknak nagyon fontos eleme a csoportmunka. Pósa kezdeti táboraiban még egyénileg dolgoztak a diákok, és az ő elmondása alapján ekkor valami nagyon hiányzott a matematika foglalkozásokon. Majd kísérletezett a csoportmunkával, és sokkal jobban élvezték a diákok, és jobban tudtak haladni is, így hamar teljesen általános lett a táborokban a csoportmunka.

3.3. Matematika tehetséggondozás Pósa-módszerrel

A matematikai tehetséggondozásnak komoly hagyományai vannak Magyarországon, és számos matektábort rendeznek. Itt a Pósa Lajos által alapított A Gondolkodás Öröme Alapítvány által szervezett táborokkal (Juhász, 2019) való kapcsolódási pontot mutatom be. Ezekben a hétvégi matematikatáborokban a nemzetközileg is elismert Pósa-módszert alkalmazzuk, amely a felfedeztető tanítás egy formája. Pósa Lajos tanítványaként belülről tapasztalhattam meg a táborok által nyújtott különleges élményt, majd egyetemista éveim alatt segítőként vettem részt bennük. Később, 2014-től 2020-ig két matematikatábori csoportot vezettem, 7. osztályos koruktól egészen 12. osztályos korukig. Jól ismerem tehát Pósa módszertanát, és komoly inspirációt adott a programozás tanítási módszertanom kidolgozásához. A híres matematikus és matematika tanár által alkotott tehetséggondozási módszertan az irányított, kollaboratív és kutatás alapú felfedeztető tanítás vonásait tartalmazza.

Irányított felfedeztető tanítás, ugyanis a diákok a tanár által megtervezett feladatokon gondolkodva fedezik fel azokat az összefüggéseket, amiket tanítani szeretnénk. A feladatok felszín alatt erős összefüggésekkel rendelkeznek. Ezt a bonyolult, szövevényes rendszert

Katona és Szűcs problémaszálak hálójaként (Web of Problem Threads) írja le (Katona & Szűcs, 2017).

Kollaboratív felfedeztető tanítás, hiszen a diákok csoportmunkában dolgoznak, ami központi eleme a módszertannak. Nem csupán az a célja ennek, hogy a kollaborációs készségeik fejlődnek. A közös gondolkodás motiválja a diákokat, közösségi élményt nyújt nekik, miközben szerteágazóbb ötleteket tudnak hozni, mint külön-külön, és tudják segíteni egymást, ha elakadtak.

Kutatás alapú tanítás, mivel a matematikatáborokban a diákok bevezetést kapnak a kutatói szerepbe. Átvezetjük őket a matematika néhány jelentős történeti pillanatán, és a kor matematikusainak helyébe képzelve magukat alkothatnak fogalmakat és tehetnek felfedezéseket. Ezek közül számos helyzetben a kísérletezésnek fontos szerepe van, tehát megtapasztalhatják a kísérletezés szerepét ebben a tudományágban is, ami az iskolai matematika órákon sokkal kevésbé jelenik meg, mint például természettudományos tantárgyakból. A feladatokon gondolkodás közben is ösztönözzük őket arra, hogy kísérletezés segítségével alkossanak sejtéseket, melyeket viszont minden alkalommal bizonyítaniuk is kell. Emellett arra tanítjuk őket, hogy egy-egy probléma megoldása után ne elégedjenek meg, hanem tegyenek fel további kapcsolódó kérdéseket, ami a kutatói szemlélet egyik alapvető jellemzője. Az érdekes kérdéseket különösen értékeljük, hiszen a matematikusok közössége is így tesz a kutatási kérdésekkel, és ezeket integráljuk a tananyagba.

3.3.1. Feladatszálak a Pósa-módszerben

Pósa pedagógiájában a felfedeztető tanítás arra épül, hogy a diákok számára egy-egy mélyebb tudás vagy módszer elsajátítása sok egymásra épülő feladat segítségével történik. Juhász Péter szálaknak nevezi ezeket az egymáshoz kapcsolódó feladatokat (Győri & Juhász, 2017). Ezek a feladatok lehetnek azonos témához kötődőek (például gráfelmélet), ugyanabba a típusba tartoznak (mint például stratégiás játékok), de legtöbbször az alkalmazott ötletek és módszerek köré épülnek (például indukció). A feladatok tehát egymásra épülnek, vagyis az egyik feladat megoldásához szükség van bizonyos ötletekre, módszerekre, amelyek a tanulók számára könnyebben elérhetőek, ha egy korábbi feladatot megoldottak. Ezt a közös gondolkodási elemet, amely egy szálon belül összeköti a feladatokat, Katona és Szűcs a szál magjának (kernel) nevezik (Katona & Szűcs, 2017), és ez általában nem nyilvánvaló módon, hanem elrejtve jelenik meg a feladatokban. A módszer lényege, hogy újabb és újabb váratlan helyzetekben alkalmazzunk hasonló gondolkodási elveket, mint korábban.

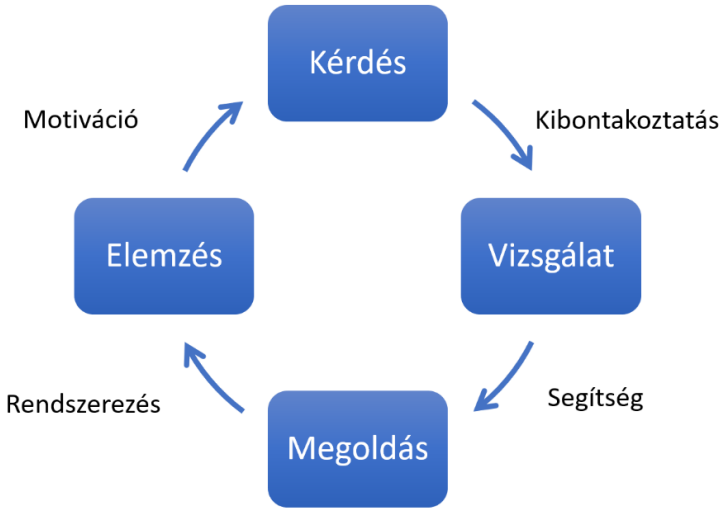
A fent említett mag lehet például az indukció, amit Pósa láncreakciónak nevez a fiatal diákok körében, és egy egyszerű logikai feladvánnyal kezdődik, amelyben egymásra hivatkozó vallomások alapján kell megtalálni három bankrablót. Évekkel később a diákok az indukció segítségével olyan összetett tételek bizonyításáig jutnak el, mint például az, hogy „minden irányított teljes gráfban van Hamilton-út”. Itt szeretném megemlíteni, hogy az utóbbi állítást nem ilyen száraz formában, hanem egy nyitott kérdésként mutatjuk be, ahol sárkányok szállítják az embereket a szigetek között, hogy a gyerekek számára szórakoztatóbb legyen. További példákat mutat be feladatszálakra Bóra Eszter, amelyek magja az algoritmikus gondolkodás (Bóra, 2020), illetve a mozgás elvének alkalmazása a problémamegoldásban (Bóra & Juhász, 2023), valamint Katona Dániel, amelyben a központi elem a megmaradó tulajdonság (Katona, 2021).

3.3.2. Feladatszálak hálózata

A táborok matematikai foglalkozásai során mindig több feladatszál fut párhuzamosan, ami azt jelenti, hogy a résztvevők a csoportmunka során egyszerre több problémán is dolgozhatnak, melyek különböző szálakhoz kapcsolódnak. Ezek a szálak azonban nem teljesen függetlenek egymástól; vannak találkozási pontjaik, ahol az egyik segíthet a másikon, sőt, egy feladat akár több szálhoz is kapcsolódhat. Így a szálak egymásba fonódnak, elágaznak, és új szálak is keletkezhetnek az egyes csatlakozó kérdésekkel. Katona Dániel a doktori értekezésében (Katona, 2022) részletesen bemutatja a problémaszálak hálózatát, mint a Pósa-módszer tananyagtervezésének egyik leglényegesebb vonását. A feladatok hálózata tulajdonképpen olyan tanterv, amely elősegíti az oktatási céljainkat képező ismeretek és készségek elsajátítását. Tehát a tanár fő kihívása az, hogy olyan tantervet alakítson ki, amely illeszkedik a tanítási céljaihoz, és ez magában foglalja a szükséges kompetenciák azonosítását, azok rendszerbe foglalását, valamint olyan feladatok összegyűjtését vagy kidolgozását, amelyek elősegítik és irányítják a tanulási folyamatot.

3.3.3. A tanulás folyamata

A Pósa-féle felfedeztető tanítás folyamatának szemléltetésére készítettem egy körmodellt, Kolb Experiential Learning Cycle nevű körmodelljének mintájára (Kolb, 1984). A téglalap alakú szektorok jelölik a folyamat egyes részeit, míg a nyilak mellett szereplő szavak az oktató szerepét mutatják a folyamat katalizálásában.



3.1. ábra: A Pósa-módszer tanulási folyamatának körmodellje.

A tanítási folyamat egy *kérdésfeltevéssel* vagy problémával kezdődik, amely lehet az oktatótól, diáktársaktól vagy akár a saját kútfőből is származó. A matematikai táborokban ez legtöbbször mindenki jelenlétében zajlik. Pósa mindig hangsúlyozza, hogy ne egy állítást adjunk meg bizonyításra, hanem helyette tegyünk fel nyílt kérdést, vagy még jobb, ha csak bemutatunk egy szituációt, és hagyjuk, hogy a diákok kérdezzenek (Pósa, 2001). Például a korábban említett sárkányos világban egy olyan helyzetet vázolunk fel, ahol a sárkány-járatok pontosan egy irányban repülnek bármely két sziget között. Ezután a feltett kérdésekért jutalmazzuk a gyerekeket (nem pedig a megoldásokért – hiszen maga a kérdezés is érték), így legtöbbször előkerül többek között az a kérdés is, amelyet mi, tanárok is fel szeretnénk adni. Célunk, hogy a gyerekeket megismertessük a kutatással, és különösen a matematikában egy érdekes kérdés nagyon értékes a tudományos közösség számára (kutatásalapú tanulás motívuma).

A diákok divergens gondolkodással indulnak el egy feladat megoldása során, Pósa pedagógiájában a matematika területén a kísérletezésnek jelentős szerepe van. Ehhez megfelelő időt és szabadságot biztosítunk a tanulóknak, hogy csoportokban (különböző helyiségekben) kutassanak, próbálkozzanak, hibázzanak, saját megfigyeléseket tegyenek, és közelebb kerüljenek, akár el is juthassanak a megoldáshoz vagy bizonyításhoz. Ezt a szakaszt nevezzük *vizsgálatnak*. Fontos megjegyezni, hogy a közoktatásban gyakran nem szentelnek kellő időt erre a szakaszra; a tipikus tanári elvárás az, hogy a diákok azonnal megoldást találjanak a feladatra. Ideális esetben az oktató feladata csupán az, hogy bátorítsa és támogassa a diákokat az önálló gondolkodásban, amit az ábrán *kibontakoztatásként* jelöltem.

Az ábrán jelölt *megoldás* lépés a matematikában egy rövid időszakot jelent, amikor – akár sok próbálkozás után – a diákok egy gondolatmenete célba ér, és ezt felismerik. Ennek eléréséhez a tanár megfelelően tervezett *segítséget* nyújthat. Tehát fontos felkészülni az oktatóknak és segítőiknek arra, hogy mit tegyenek, ha a diákok a csoportmunka során elakadnak valamilyen ponton. Pósa szerint az is sokat ér, ha segítséggel oldják meg a feladatot, ezért tanárként előre tervezhetjük, hogy milyen helyzetben és milyen kérdéssel, tanáccsal segítsük előre a diákok munkáját (irányított felfedeztető tanítás motívuma).

A következő fontos lépés a táborokban plenárisan zajlik: a megoldások újbóli áttekintése, *elemzése*, valamint a gondolkodási folyamatra visszatekintés, illetve a kapcsolatok feltárása más feladatok és módszerek között. Az ide vezető úton az oktató szerepe a tudatosítás, az önálló gondolkodás során tanult vagy felfedezett dolgok *rendszerezése*. Az utolsó lépés – amivel be is zárul a kör és újraindul a folyamat – az, hogy minden alkalommal, amikor megbeszélünk egy probléma megoldását, lehetőséget biztosítunk és *motiváljuk* a diákokat kapcsolódó kérdések feltevésére.

3.4. Felfedeztető oktatási módszertan algoritmusok és programozás tanítására

3.4.1. Motiváció, célok

A 3.3. fejezetben röviden bemutatott matematikai tehetséggondozó módszert szeretnénk alkalmazni az algoritmikus programozás tanítására. A Gondolkodás Öröme Alapítványban a vezetésemmel elindítottunk egy új programozás tehetséggondozó programot, a ProgTáborokat (Nikházy, 2020).

Juhász Péter a felfedeztető matematika tanítás alapelveként fogalmazza meg, hogy „Legfontosabb célunk, hogy megpróbáljuk gondolkodni tanítani a gyerekeket. Vagyis azt próbáljuk elérni, hogy ne képletek, bemagolt módszerek, alkalmazható tételek után kutassanak a fejükben, hanem engedjék szabadjára a fantáziájukat, és bátran ötleteljenek, kreatívan gondolkodjanak.” Ez a megközelítés összhangban van a programozás tehetséggondozó programunk céljaival, amelynek középpontjában az algoritmikus gondolkodás és a problémamegoldás áll. Célunk, hogy megmutassuk a gyerekeknek az érdekes, számítógéppel megoldható problémákon való fejtörés örömét, és egy-egy működő program elkészítésével járó

sikerélményt. Emellett lehetőséget kínálunk az érdeklődőknek a programozási versenyek világának megismerésére.

A hangsúlyt nem a versenyre, hanem a versenyeken szereplő problémák szépségére fektetjük. Ugyanakkor a versenyfeladatok ideálisan illeszkednek a terveinkbe, hiszen a résztvevők algoritmizálási és problémamegoldási képességeit mérik. A feladatokat tapasztalt programozók tűzik ki részben azzal a céllal, hogy más emberek is örömüket leljék a rajtuk való gondolkodásban. A problémák szépsége nehezen definiálható fogalom, de a versenyeken résztvevők közössége nagyra értékeli. A szépség származhat akár egy érdekes kérdésfeltevésből, egy elegáns megoldásból, egy ismert módszer váratlan szituációban való alkalmazásából, távoli témakörök összekapcsolódásából stb. Tehát bizonyos értelemben a programozási versenyek világa is hasonlóan öncélú, mint a matematika: azok az emberek tartják fenn, akik örömmel vesznek részt benne.

3.4.2. A matematikai módszertan módosítása, eltérések és hasonlóságok

A számítógépes problémamegoldás természete különböző a matematikától. Rengeteg sztenderd algoritmus és adatszerkezet van, amelyeket testre kell szabni, kombinálni és számos különböző forgatókönyvben alkalmazni. Ezeket feladatok sorozatán keresztül próbáljuk megtanítani úgy, hogy a diákok lehetőség szerint saját maguk fedezzék fel az algoritmusok lényegét. Ugyanakkor még nagyobb hangsúlyt fektetünk ezeknek a módszereknek a különböző problémákban való alkalmazására. Ezért megközelítésünket problémaalapú tanításnak tekinthetjük. A feladatok között hasonló egymásra épülések és összefüggések vannak, mint a Pósa-féle matematikatáborokban. Az általunk tanított algoritmusokhoz és adatszerkezetekhez feladatszálakat hoztam létre, és igyekeztem ezeket összekapcsolni, így alakítva ki az algoritmikus programozás tanítására használt probléma hálózatot.

A programozás egy fontos sajátossága, amely megkülönbözteti a matematikától, az a kigondolt algoritmusok implementálása, ami egy teljesen új lépést hoz be a tanulás folyamatába is. A cél helyesen működő programok létrehozása, és erre az alkotómunkára nagy hangsúlyt fektetünk a tanítás során is. Programozóként először az algoritmusokat fejben megtervezzük, és bizonyítjuk azok helyességét, a megoldás viszont csak akkor lesz kész, ha implementáltuk az algoritmust egy programban és ez helyesen is működik. Ennek érdekében kimerítően teszteljük a programot, számos bemenetre ellenőrizzük, hogy helyes és eléggé gyors-e. A versenyeken a megoldásokat ez alapján értékelik, és manapság már az ehhez használt automatikus értékelő rendszerek később is hozzáférhetők maradnak.

Az algoritmikus problémamegoldás tanításának legelején csak arra fókuszálunk, hogy helyes megoldást írjunk (például egyszerű mohó és dinamikus programozásos feladatok esetében), de hamar olyan témákat veszünk, amik kifejezetten arról szólnak, hogy nem egy helyes megoldás kitalálása a kihívás, hanem az, hogy hatékony algoritmussal csináljuk. Ilyen téma például a prefix összegek, két mutató technika és bináris keresés. Tehát az algoritmikus programozásban folyamatosan jelen van ez az újfajta kérdés: mennyire hatékony a kitalált megoldás, lehet-e hatékonyabban csinálni? Ezen kérdések vizsgálatakor is nagyon hasonló módszerekre van szükség, mint matematika feladatok megoldásakor. A gyakorlatban, a feladatmegoldás során a hatékonyság ellenőrzése úgy valósul meg, hogy egy megadott időlimiten belül le kell futnia a programnak a legnagyobb lehetséges tesztadatokra is, megadott maximális memóriahasználat mellett. A cél mindig az algoritmus aszimptotikus viselkedésének a tesztelése, vagyis, hogy a bemenet méretének függvényében hogyan alakul a futási idő- és tárhelyigény. A bemenetre megadott korlátokból pedig ki lehet következtetni, hogy milyen komplexitású (például lineáris, négyzetes, stb.) algoritmus elvárt a feladat megoldásához. A diákoknak megtanítjuk, hogy az algoritmusok komplexitásán múlik főként a programok gyorsasága, de emellett odafigyelünk arra is, hogy a kódoláskor ne csússzon be hiba, ami ezt elrontaná.

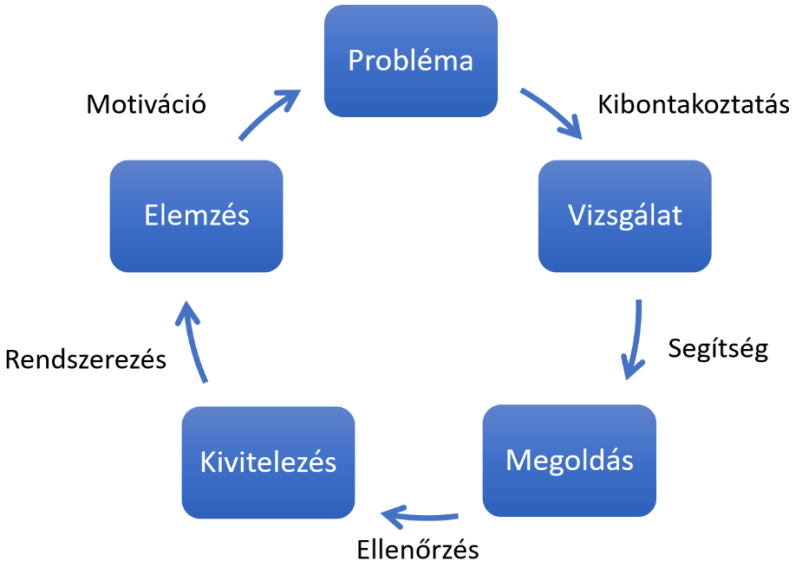
A Pósa-módszerre jellemző csoportmunkát kiválóan lehet alkalmazni a programozás tanításban is. A matematikatáborokban 2-4 fős csoportok vannak, a programozásban 2-3 fős csoportokat javaslok, mivel egy-egy megoldás elkészítése hosszadalmasabb összebeszélést igényel, és 4 fő esetén nagyobb veszélye van annak, hogy bizonyos csoporttagok kimaradnak valamiből vagy két együttműködő párra szakad a csoport. Az 5. fejezetben bemutatott méréseim alapján is azt látom, hogy a diákokat motiválja a közös munka, de még fontosabb, hogy egymást inspirálhatják, egymás ötleteiből táplálkozhatnak és segíthetik egymást az implementáció közben is. Ez utóbbi, a közös programozás egy extra elem, ami miatt talán kijelenthetjük, hogy a programozásban még fontosabb lehet a csoportmunka, mint a matematikában.

A matematikatáborokban központi szerepe van annak, hogy egy feladat megoldásának megbeszélése után a diákok tesznek fel csatlakozó kérdéseket. Ez modellezi a kutatói munkát, hiszen a kutatások mindig érdekes kérdésfeltevéssel kezdődnek. A programozástáborokban is lehetne teret biztosítani a csatlakozó kérdéseknek, hiszen általában érdekes új problémákat kaphatunk, ha módosítunk egy-egy feladat peremfeltételein, vagy csavarunk valamit a történeten. Egyelőre ezt az elemet kevésbé alkalmazzuk a táborokban, mert az effajta kalandozások nélkül is szűk a táborok időkerete, másrészt nem is lenne könnyű a diákok

kérdéseiből online beadható feladatot készíteni. A jövőben ezt az aspektusát szeretném még továbbfejleszteni a tehetséggondozó táboroknak.

3.4.3. A tanulási folyamat

A fentiek alapján a problémaközpontú algoritmikus programozás tanítás folyamatát a matematikatáborokra adott körmodellemhez képest kiegészítettem egy kivitelezési lépéssel, amely magába foglalja az implementációt, tesztelést és hibakeresést, hiszen ezek szorosan összefonódó folyamatok. Az oktató feladata ebben a folyamatban az, hogy visszajelzést adjon a megoldásról, és figyelmeztesse a tanulókat, ha például a megoldás nem helyes, vagy nem elég hatékony, konkrét ellenpéldákkal segítve. Ezért ez az ellenőrzési lépés komoly szellemi energiát igényel, hiszen át kell gondolni, hogy a tanuló által kigondolt algoritmus helyes-e, a program pedig megfelelően működik-e, valamint segíteni kell a hibák kijavításában.



3.2. ábra: A problémaközpontú programozás tanulási folyamatának körmodellje.

A kivitelezés folyamán általában egyéni munka zajlik, bár lehetőség van a párban történő programozásra is (pairwise programming). A folyamat többi részében a matematika módszertani elemek alkalmazhatóak: a vizsgálat és megoldás során ideális, ha a diákok csoportban dolgoznak, míg az elemzés és problémafelvetés esetében a közös megbeszélés a legjobb megközelítés.

3.4.4. Feladatszálak hálózata

A Pósa-módszer tananyagtervezési keretrendszerét, azaz a problémaszálak hálózatát jól lehet alkalmazni az algoritmikus programozás tanítására is. A következő fejezetben azt mutatom be,

hogyan alakítottam ki a feladatszálakat és gyűjtöttem hozzájuk feladatokat. Szerencsére a korábbi programozási versenyek feladatai hatalmas mennyiségben állnak rendelkezésre online, és általában valamilyen jópofa történettel fedik a mögöttes kérdést, ahogy a matematikatáborok feladataihoz mesék is tartoznak, így első ránézésre nem egyértelmű, hogy melyik szálhoz tartoznak. Kívánatosnak tartom, hogy a tanításhoz használt problémáinkhoz legyen elérhető ilyen tesztelő rendszer. Ezért igyekeztem a céljaimnak megfelelő, a problémaszálak hálózatába beilleszthető versenyfeladatokat keresni. Emellett néhány saját feladathoz én készítettem kiértékelő programot.

4. Problémaközpontú tananyag és tanterv az algoritmikus programozás tanítására

4.1. Célkitűzés

A felfedeztető módszertanú programozás tehetséggondozó programunk létrehozása szempontjából kulcsfontosságú az, hogy a tananyagot a Pósa-módszerre jellemző problémaszálak hálózataként építsük fel. Csak akkor valósulhatnak meg a pedagógiai céljaink, ha izgalmas feladatokat tudunk adni a diákoknak a megfelelő egymásra épülési rendszerben és kapcsolódási pontokkal, amelyek által a tanítani kívánt alapvető algoritmusokat, adatszerkezeteket és problémamegoldási módszereket elsajátíthatják a tanulók. Kutatásom egyik legfontosabb célja volt az, hogy létrehozzam ezt a tantervet és gyakorlati alkalmazás közben folyamatosan továbbfejlesszem.

A legtöbb könyv, online kurzus és tutorial a programozás elemeire összpontosít. Mi más célokat tűztünk ki magunk elé, ezért ezek nem tartoznak a tanterv hatáskörébe, de szeretnénk a programozási alapokra építeni. Algoritmusok tanításakor feltételezzük a nyelvi elemek ismeretét, mint például a változók, típusok, operátorok, feltételes utasítások, ciklusok stb. A nagyon alapvető algoritmikus struktúráktól indulunk, és olyan komplex módszerekig jutunk, amelyeket a Nemzetközi Informatikai Diákolimpián használnak. Ennek a tantervnek elsődleges felhasználási területe a ProgTábor tehetséggondozó program, amely 12 éves kortól kezdődik és 18 éves korig tart. Pósa matematikai táborai megmutatták számunkra, hogy lehetséges évekig együtt tartani a tábori csoportokat, rendszeresen fejleszteni tudásukat és olyan témákat tanítani, amelyek egymásra épülnek.

Az interneten számos kiváló forrás érhető el, amelyek elősegítik az algoritmikus programozás haladó szintű elsajátítását, például Halim könyve (Halim, Halim, & Effendy, 2018), vagy a korábbi Codeforces versenyek hatalmas problématára (Codeforces, 2024). Ahhoz azonban, hogy ezeket oktatási céljaink érdekében felhasználhassuk, ezeket az anyagokat és feladatokat úgy kell rendszerbe szerveznünk, hogy a tanulás a problémamegoldás sorozatán keresztül történjen. Olyan helyzeteket szeretnénk létrehozni, amelyekben a diákok egy problémával szembesülnek, és a kívánt algoritmushoz vezető kulcsötleteket már korábban, különböző feladatok megoldása során látták. Ezért a tanár számára a fő kihívás a tananyag megtervezése és a feladatok megfelelő struktúrába rendezése, amely lehetővé teszi az adott téma megfelelő időben történő bevezetését.

4.2. Felépítés

A módszerünk esetén a tanterv egy hatalmas feladathálózat, amelyen keresztül a diákok felfedezik, gyakorolják és bővítik ismereteiket. Először egy áttekintésben, magasabb szinten mutatom be a hálózatot, mégpedig úgy, hogy rendszerezem a feladatok elméleti hátterét, és azok közötti kapcsolatokat azonosítok. Az algoritmusokról szóló könyvek és weboldalak általában a témakörök (pl. adatszerkezetek, gráfelmélet, geometria) szerint rendezve haladnak a tananyaggal, ami nagyszerű, ha már ismerünk valamit, de nem feltétlenül ideális sorrend a tanuláshoz. Nemcsak a nehézségi sorrendről beszélek, hanem a célom az is, hogy megtaláljam az ideális tervet, ahol minimalizáljuk a szükséges nagy ötletek számát az elvárt algoritmusok kidolgozásához. Például a Bellman–Ford és Floyd–Warshall algoritmusok tanítása esetén jó, ha már gyakorlottak a diákok a dinamikus programozásban.

A témakörök a közöttük megtalálható kapcsolatok által tehát már maguk is egy gráfot alkotnak, amelynek az általam feltérképezett változatát meg is mutatom a 4.3. fejezetben. A tervezést ezen a magasabb szinten kezdtem, ezt követően folytattam azzal, hogy az egyes témakörökhöz feladatokat gyűjtöttem. A témaköröknek alapvetően négy típusa van: problémamegoldó módszerek, algoritmusok vagy algoritmus-sablonok, adatszerkezetek, elméleti fogalmak. A Pósa-módszer elemzői a problémaszálak magjának vagy kerneljének nevezik azt a közös gondolati elemet, aminek elsajátítása érdekében kitűzzük a feladatokat (Katona, 2021), és ez legtöbbször egy matematikai gondolkodásmód (például engedékenység), de lehet akár közös fogalom (például maradékok) is. Amit tehát témakörnek hívok, az tulajdonképpen egy-egy feladatszál magja.

Minden témakörhöz feladatszálat alakítottam ki, amelyben a feladatoknak megvan a maga célja a témakör tanítási folyamatában. Ez a feladatgyűjtemény nem teljes, és sosem lesz az, hiszen a folyamatos bővítés és változtatás része a filozófiámnak, viszont a jelenleg bemutatott állapotában is alkalmas már arra, hogy egy 7-12. osztályon átívelő táborsorozatot felépítsünk belőle. Az egyes feladatszálakon belül a felfedeztető tanítás szempontjából a következő három fontos feladattípust határoztam meg:

- Bevezetés: ezek a témával kapcsolatos legelső feladatok, amelyek a felfedezést elősegíthetik.
- Elmélyítés: a korábban megismert módszerek alkalmazását jelentik új helyzetekben, vagy gyakorlófeladatok is lehetnek, a céljuk az, hogy megerősítsék a diákok tudását.

- Szintézis: a terminológiámban olyan feladatokat jelentenek, amelyek több tanult módszert kötnek össze, vagy magas szintű megértést igényelnek az érintett koncepciókról.

A feladatok természetesen gyakran kapcsolódnak más témakörökhöz és azok feladataihoz. A feladatok szintjén történő kapcsolódások, kölcsönhatások egy hatalmas hálózatot teremtenek, azonban ezt a hálózatot ábrázolni nem tudom áttekinthető módon, praktikus keretek között. Így a feladatokat táblázatos formában adom meg, ami a terjedelmére tekintettel is az *1. számú mellékletben* kapott helyet. A táblázatban a feladatok szerepénél a fenti három mellett még két jelzőt használok gyakorlati szempontok miatt: az „alap" olyan bevezető típusú feladatot jelöl, amiben pont egy tankönyvi algoritmus implementációját kell megcsinálni, a „lekötő" feladatok pedig nehéz feladatok, amelyek nem kritikus elemei a feladatszálaknak, és fakultatív feladatként fel lehet őket adni azoknak a diákoknak, akik mások előtt járnak (típusukat tekintve elmélyítő vagy szintézis feladatok lehetnek).

4.3. Témák és feladatok

A tanterv megtervezése a tanítani kívánt témák és módszerek meghatározásával kezdődik. A programozási versenyek elég jól tükrözik, hogy az informatikusok közössége mit tart fontos tudásnak és készségeknek az algoritmikus programozás területén. A tanterv elemeit a versenyek anyagai és a versenyekre való felkészülést segítő, naprakész szakirodalom alapján választottam ki.

4.3.1. Források

Legfontosabb forrásom a Nemzetközi Informatikai Olimpia (IOI) sillabusza (Verhoeff, et al., 2024), amely kiváló összefoglalót nyújt az elvárt ismeretekről. A Halim és társai által írt Competitive Programming 4 (Halim, Halim, & Effendy, 2018) tartalmazza az International Collegiate Programming Contest egyetemi hallgatóknak szóló versenyén előforduló témák többségét. Megvizsgáltam Laaksonen Competitive Programmer's Handbook című könyvének tartalmát is (Laaksonen, 2017), hiszen az általuk fejlesztett CSES weboldal (Laaksonen, Salmi, & Talvitie, Code Submission Evaluation System, 2024) számomra az egyik leghasznosabb feladatbankot foglalja magában. Van néhány kiváló online forrás és oktatóanyag, amely összegyűjti a fontos algoritmusokat és adatszerkezeteket, például az USACO Guide (Competitive Programming Initiative, 2024) és CP-Algorithms weboldalak (CP-Algorithms,

2024), a Geeks for Geeks (GeeksforGeeks, 2024) és a Codeforces (Codeforces, 2024) különböző blogbejegyzései.

Az egyik cikkemben (Nikházy, 2019) fiatal diákjaim segítségével áttekintettem a Nemes Tihamér Programozási Verseny és az Informatika OKTV Programozás kategóriájában a 2015-2019 években kitűzött feladatokat (több, mint 150 feladatot), és az ezekben előforduló témakörök azonosítása alapján készítettem a hazai országos versenyekhez is sillabuszt. Így átfogó kép alakult ki bennem a magyar versenyek anyagáról, valamint hasznomra vált a sok éves versenyzői és felkészítő tanári tapasztalatom, ezt a fenti forrásokkal integrálva állítottam össze a tananyag tartalmát, amely meglátásom szerint a középiskolások számára a legfontosabb ismereteket tartalmazza.

Számos hasonló témájú munka született már korábban, különösen arról, hogyan készítsük fel a diákokat a versenyekre. Király Sándor a programozás tanításának teljes tervét írja le a kezdetektől az IOI-ra való felkészülésig (Király, 2011), megfelelő pedagógiai irányelvekkel és hasznos tanácsokkal együtt. Erdősné dr. Németh Ágnes doktori disszertációjában részletes betekintést nyújt a hazai és nemzetközi informatikai tehetséggondozásba (Erdősné, 2019), felvázolva egy tervet is haladó programozás oktatására középiskolásoknak. A konkrét témáknál, ahol feladatokat tartalmaznak, mindkét dolgozat az általam választott feladatokhoz nagyon hasonló feladatokat mutat be. Munkájukhoz és a fent említett szakirodalmakhoz képest a tanterv újdonsága abban rejlik, hogy a feladatrendszert Pósa felfedeztető tanítási módszeréhez igazítom. Mivel ebben a didaktikában a feladatok kiválasztása és a kapcsolódási pontok döntő szerepet játszanak, ezért részletesebb és teljesebb feladatlistát közlök, a fent ismertetett struktúrába rendezve.

A feladatokat különféle online feladatbankokból gyűjtöttem össze, amelyek nyilvánosan elérhetők, és rendelkeznek kiértékelő rendszerrel a megoldások ellenőrzéséhez. Ez számunkra fontos, mert szeretnénk megmutatni a diákoknak, hogy nemcsak az elmélet, hanem a megvalósítás is lényeges része az informatikának. Erdősné kiváló áttekintést nyújt az online kiértékelő rendszerekről és jellemzőikről (Erdősné, 2018), amely magában foglalja szinte az összes alább felsorolt weboldalt. Ahogyan azt a következő fejezetekben és a mellékletekben lehet látni, a problémagyűjteményemben a feladatok nagy része a Codeforces, a CSES és a Mester feladatbankokból van, mert ezekben a gyakorlási módban a diáknak (vagy a tanárnak) hozzáférése van a kis tesztesetekhez és azok várt válaszaihoz, amely nagyon hasznos a hibakeresés szempontjából. A CSES az értékeléshez használt teszteseteket teljes egészében

elérhetővé teszi, a Codeforces a be- és kimenetek első 500 karakterét mutatja, a Mester pedig 4 KB méretig engedi letölteni a teszteket tanári fiókkal. A korlátozások nem jelentenek nagy hátrányt, mivel a túl nagy méretű adatokat mi úgysem tudnánk áttekinteni.

A feladatok forrásaiként az alábbi weboldalakat használom (gyakoriság szerinti sorrendben). Az oldalak webcíme a dokumentum végén, a 10. fejezetben van felsorolva.

- **Codeforces**: egy átfogó platformot nyújt a programozók számára, ahol versenyek, gyakorló feladatok és közösségi fórumok segítségével fejleszthetik algoritmikus képességeiket. Az oldalon megtalálhatóak ranglisták, részletes megoldási útmutatók, valamint lehetőség van a közösségen belüli interakcióra és tudásmegosztásra is. Az összes korábbi versenyfeladat megoldható gyakorló üzemmódban, ahol a tesztesetek egy része is látszik, ezen kívül a Codeforces Gym-ben számos gyakorlóverseny van, illetve hagyományos versenyek feladatai is fel vannak töltve. Ki szeretném emelni a Mashup funkciót, amivel könnyedén össze lehet állítani egy versenyt az összes korábbi feladatból válogatva. Sőt, a Polygon rendszerben saját feladatokat is ki lehet dolgozni.
- **Code Submission Evaluation System (CSES)**: elsősorban tanuláshoz kifejlesztett algoritmikus programozási feladatbank, amely rengeteg szép és klasszikus feladatot tartalmaz, és letölthetők a tesztesetek is, ami nagyban könnyíti a hibakeresést.
- **Mester**: magyar feladatbank, ahol a hazai programozási versenyfeladatok mellett még számos gyakorlófeladat elérhető témakörökbe rendezve. Sajnos a rendszeren belül nincs olyan link, amely egy feladatra mutatna, ezért a feladatok megosztására egy GitHub repositoryt használunk, ahol a feladatok leírásai megtalálhatók.
- **AtCoder**: szintén egy teljes körű versenyplatformot biztosít, ahol rendszeres versenyek, gyakorló feladatok és ranglisták segítségével fejleszthetik algoritmikus programozási készségeiket a felhasználók, különös hangsúlyt fektetve a japán és nemzetközi programozó közösség összekapcsolására. Az összes korábbi verseny feladatai elérhetőek és megoldhatók.
- **Sphere Online Judge (SPOJ)**: egy online kiértékelőrendszer, amely hatalmas feladatbankkal rendelkezik, több mint 20 000 programozási problémát kínál.
- **OJ.uz**: a nemzetközi diákolimpiák és regionális programozási versenyek korábbi feladatsorai nagy mennyiségben lelhetők fel itt, és a beadott megoldások nyilvánosak mindenki számára.

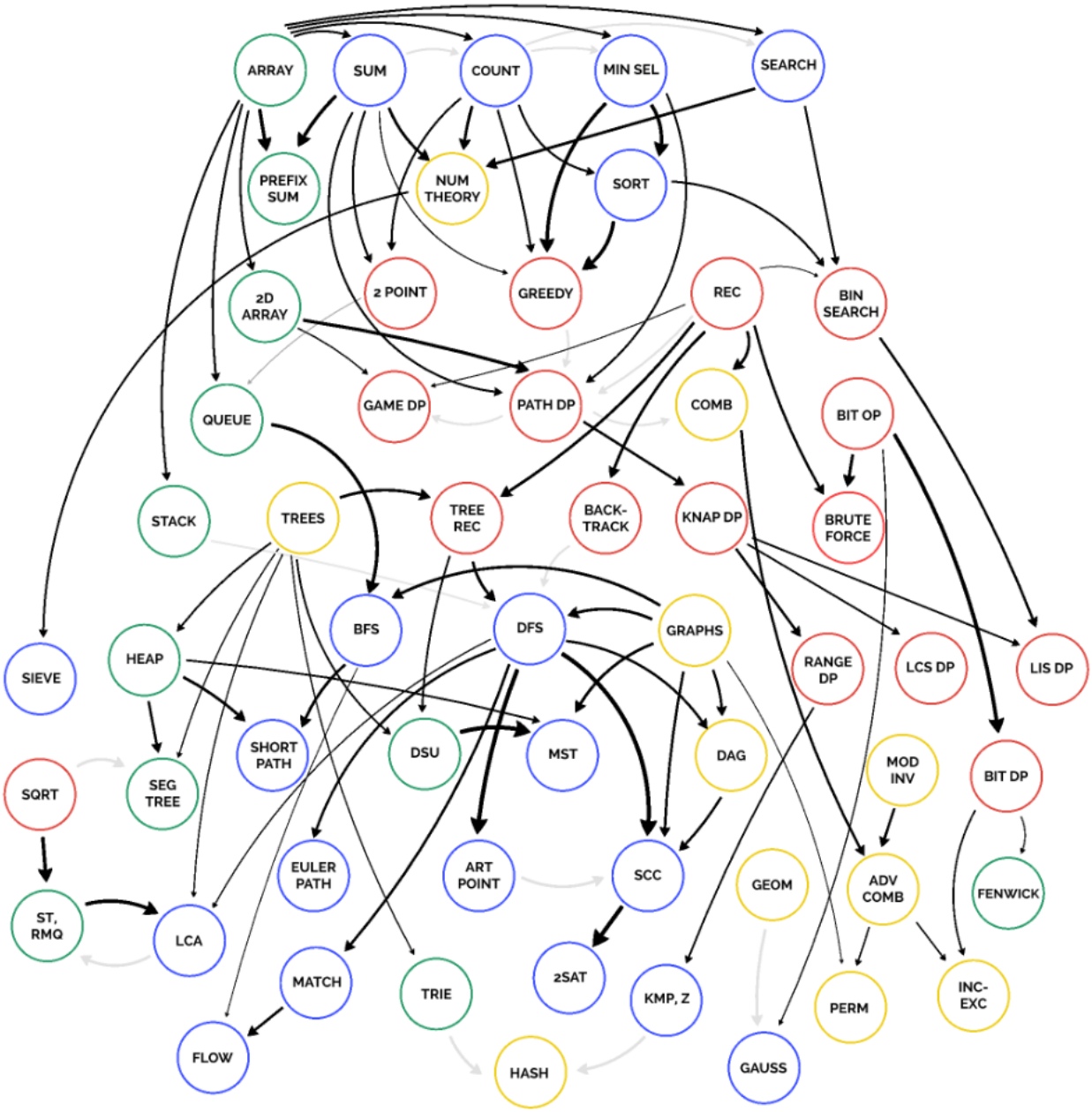
- **HackerRank**: egy oktatásra és különösképpen céges interjú felkészítésre fókuszáló oldal, ahol különféle szakmai készségeket fejlesztő feladatok várnak megoldásra, témák szerint csoportosítva.
- **NJudge**: Noszály Áron által fejlesztett magyar nyelvű kiértékelő-rendszer és feladatbank, amelyben az elmúlt évek hazai országos programozási versenyeinek a feladatai megoldhatók.
- **UVa Online Judge**: a korábbi ICPC versenyfeladatok legnagyobb gyűjteménye, az online kiértékelőrendszer használata elég nehézkes, mivel régi rendszer.
- **CodeChef**: egy átfogó versenyplatformként indult, amely hasonlított a Codeforces-hoz és az AtCoder-hez, viszont az utóbbi időben sajnos átalakulóban van, a szoftverfejlesztői munkapiacra felkészítő weboldalhoz kezd hasonlítani, és egyre kevesebb tartalom érhető el rajta nyilvánosan.
- **Timus Online Judge**: az egyik legnagyobb feladatgyűjtemény korábbi orosz versenyek és felkészítő táborok feladataiból.

4.3.2. Témák

A 4.1. ábra a tanterv témaköreinek grafikus ábrázolását mutatja, mindegyiket néhány betűs kóddal rövidítettem a könnyebb olvashatóság érdekében. A témák teljes nevei és leírásai a 4.1. táblázatban találhatók. Négy típusba soroltam őket, amelyek különböző színekkel vannak jelölve az ábrán:

- problémamegoldó módszerek (piros),
- algoritmusok vagy algoritmus-sablonok (kék),
- adatszerkezetek (zöld),
- elméleti háttér, fogalmak (sárga).

A témák közötti kapcsolatokat nyilak ábrázolják, minél vastagabb a nyíl, annál erősebb a függőség közöttük. Emellett világos szürke nyilakat is használok, amelyek nem függőséget, hanem inkább hasonlóságot jelölnek, amikor egy módszer ismerete segít a másik módszer elsajátításában.



4.1. ábra: A tanterv témaköreinek gráfja

Kód	Angol név	Magyar név	Leírás
ARRAY	Array	Tömb	Tömb használata elemek sorozatának tárolására.
SUM	Sum	Összegzés	Számsorozat összegének kiszámítása.
COUNT	Count	Megszámlálás	Egy sorozat bizonyos tulajdonsággal rendelkező elemeinek megszámlálása.
SEARCH	Search	Keresés	Egy adott tulajdonsággal rendelkező elem(ek) keresése egy sorozatban.
MIN SEL	Minimum Selection	Minimumkiválasztás	Egy sorozat legkisebb/legnagyobb értékének kiválasztása.
SORT	Sort	Rendezés	Sorozat rendezése egyszerű rendezési algoritmusokkal és a beépített rendezés (sort() függvény) használata.
NUM THEORY	Basic Number Theory	Egyszerű számelméleti algoritmusok	Egy egész szám osztóinak meghatározása, egyszerű prímteszt, Euklideszi algoritmus.
2D ARRAY	Two-dimensional array, Matrix	Két dimenziós tömb, Mátrix	Két dimenziós tömb használata táblázat, mátrix tárolására.
GREEDY	Greedy Algorithms	Mohó algoritmusok	Problémák megoldása mohó döntésekkel, annak felismerése, hogy ez az optimumhoz vezet-e.
PREFIX SUM	Prefix Sum	Prefix összegek	Egy sorozat prefix összegei, mint adatszerkezet (más néven kumulatív összegek).
2 POINT	Two Pointers Method	Két mutató technika	A két mutató technika egyes optimalizálási feladatok felgyorsítására.
REC	Recursion	Rekurzió	Problémamegoldás a rekurzió erejével, egyszerű rekurzív összefüggések.
PATH DP	DP for Best Path	Lépegetős DP	Bevezetés a dinamikus programozásba: útvonal optimalizálása mezők sorozatán vagy négyzetrácson.
GAME DP	DP for Simple Games	Játék DP	A győztes stratégia megtalálása egyszerű két fős játékokban dinamikus programozással.
COMB	Combinatorics	Kombinatorika	Kombinatorikai alapfeladatok, mint például permutációk, kombinációk, Fibonacci-típusú sorozatok stb.
BIN SEARCH	Binary search	Bináris keresés	Bináris keresés egy rendezett sorozatban egy elem megtalálására, valamint optimalizálási feladatokban a szélsőérték megtalálására.
BITOP	Bitwise operations	Bitműveletek	Kettes számrendszerbeli alak és bitműveletek: és, vagy, xor, shiftelés, stb.
BRUTEFORCE	Brute force, exhaustive search	Kimerítő keresés	Az összes lehetőség kipróbálása, technikák az összes részhalmaz és összes permutáció vizsgálatára.
BACKTRACK	Backtrack	Visszalépéses keresés	Kimerítő kereséses megoldások gyorsítása visszalépéses kereséssel, 8 királynő probléma és rokon problémák.
KNAP DP	DP in Knapsack problem	Hátizsák DP	Néhány klasszikus DP probléma: hátizsák, pénzváltás és hasonlók.

4.1. táblázat: A tanterv témakörei

Kód	Angol név	Magyar név	Leírás
TREES	Trees, Binary Trees	Fagráfok, bináris fák	Különböző helyzetekben megjelenő fa struktúra, pl. családfa, vállalati struktúra.
TREE REC	Recursion on trees	Fa rekurzió	Fákról szóló problémák megoldása rekurzió segítségével, hierarchikus struktúrát tartalmazó feladatok.
STACK	Stack	Verem	A verem (LIFO) adatszerkezet megértése és használata.
QUEUE	Queue, Deque	Sor, kétvégű sor	A sor (FIFO) és a kétvégű sor adatszerkezetek megértése és használata.
GRAPHS	Graphs	Gráfok	A gráfok fogalmi bevezetése a különböző problémák hátttereként.
BFS	Breadth-First Search	Szélességi bejárás	Szélességi bejárás alkalmazása a legrövidebb utak meghatározására és más problémák megoldására.
DFS	Depth First Search	Mélységi bejárás	A mélységi gráfbejárás rekurzív algoritmusa és alapvető alkalmazásai.
DAG	Directed Acyclic Graphs	Irányított körmentes gráf	DAG-ot tartalmazó problémák, mint például a topológikus rendezés, kritikus út.
SHORT PATH	Shortest Path Algorithms	Legrövidebb utak	Legrövidebb utak keresése egy súlyozott gráfban. Bellman–Ford, Floyd–Warshall, Dijkstra algoritmusok.
HEAP	Heap, Priority Queue	Kupac adatszerkezet, prioritási sor	A kupac adatszerkezet és alkalmazása a prioritási sor megvalósítására, kupacrendezésre.
DSU	Disjoint Set Union / Union-find	Unió-holvan	Az unió-holvan adatszerkezet, és alkalmazásai különböző problémákban.
MST	Minimum Spanning Tree	Minimális feszítőfa	A minimális feszítőfa meghatározása súlyozott gráfokban. Kruskal és Prim algoritmusok.
RANGE DP	DP on ranges	DP intervallumokon	Dinamikus programozás intervallumokon, vagy kétváltozós részproblémákkal.
LCS DP	Longest Common Subsequence	Leghosszabb közös részsorozat	Két sorozat leghosszabb közös részsorozata és rokon problémák.
LIS DP	Longest Increasing Subsequence	Leghosszabb növekvő részsorozat	A leghosszabb növekvő részsorozat probléma: a dinamikus programozást bináris kereséssel gyorsítjuk.
BIT DP	Bitmask DP, DP over subsets	Bitmaszk DP	Dinamikus programozás részhalmazokra, a halmazok kettes számrendszerbeli ábrázolásának felhasználásával.
SCC	Strongly Connected Components	Erősen összefüggő komponensek	Kosaraju és Tarjan algoritmusa és alkalmazásai.
ART POINT	Articulation Points, Bridges	Elvágó pontok, hídélek	Kétszeresen összefüggő gráfok, Tarjan algoritmusa és az L-érték az elvágó csúcsok és élek megtalálására.
EULER PATH	Eulerian Path	Euler-séta	Az Euler-út vagy -kör feltétele és megtalálása irányított és irányítatlan gráfokban.
MATCH	Maximal Matching	Maximális párosítás (páros gráfokban)	Magyar algoritmus a maximális párosításra páros gráfban.

4.1. táblázat: A tanterv témakörei

Kód	Angol név	Magyar név	Leírás
GEOM	Geometry	Geometriai algoritmusok	Síkbeli geometriai problémák rácspontokon.
MOD INV	Modular Inverse, Fast exponentiation	Modulo inverz és gyorshatványozás	Hatékony algoritmus hatványozásra és a inverz számításra maradékosztályokon.
SIEVE	Sieve of Eratosthenes	Eratosztenészi szita	Prímek listájának meghatározása szita módszerrel.
ADV COMB	Advanced Combinatorics	Haladó kombinatorika	Különböző nehéz kombinatorikai problémák.
PERM	Permutations	Permutációk	Permutációkkal kapcsolatos feladatok, ciklusfelbontás.
INC-EXC	Inclusion-Exclusion Principle	Logikai szita	A logikai szita módszer alkalmazása kombinatorikai kérdések megválaszolására.
SEG TREE	Segment Tree	Szegmensfa	A statikusan kiegyensúlyozott bináris fák (szegmensfák) használata lekérdezések feldolgozására.
FENWICK	Binary Indexed / Fenwick Tree	Fenwick-fa	A Fenwick-fa adatszerkezet a szegmensfák alternatívájaként.
HASH	Hashing	Hashelés	Hashelés alkalmazása programozási versenyfeladatokban, hashtábla alapelve.
TRIE	Trie, Suffix Tree, Suffix Array	Trie	Adatszerkezetek szavak tárolására és keresésére: Trie, Suffix Tree, Suffix Array.
KMP, Z	Knuth-Morris-Pratt, Z-algorithm	Knuth-Morris-Pratt és Z-algoritmus	Fejlett string mintaillesztési algoritmusok: KMP, Z-algoritmus.
SQRT	SQRT Decomposition	Gyökös felbontás	A gyökös felbontás, mint problémamegoldó technika, Mo algoritmusa.
LCA	Lowest Common Ancestor	Legalacsonyabb közös ős	Egy fában két csúcs legalacsonyabb közös ősének megtalálása és alkalmazásai.
ST, RMQ	Sparse Table, Range Minimum Query	Sparse Table, Range Minimum Query	Sparse Table használata a Range Minimum Query probléma megoldására, valamint további alkalmazások.
FLOW	Network Flows	Hálózati folyamok	Feladatok modellezése hálózati folyamokkal, maximális folyam és minimális vágás meghatározása.
GAUSS	Gaussian Elimination	Gauss-elimináció	Lineáris egyenletrendszer megoldása.
2SAT	2-satisfiability problem	2-SAT probléma	A 2-SAT probléma megoldása gráfalgoritmusokkal, és alkalmazása más feladatokban.

4.1. táblázat: A tanterv témakörei

4.3.3. Feladatok

A fenti 57 témakör problémaszálaihoz több, mint 600 feladatot gyűjtöttem. Ezeket az 1. számú mellékletben listázom táblázatos formában. A feladatok azonosításához a címük mellett a feladatbankon belüli azonosítójukat is megadom, vagy a Mester esetén az elérési útvonalat. Minden feladatnál feltüntetem a témakört és a problémaszálon belüli szerepét is. A feladatok kiválasztásánál mindig az játszotta az elsődleges szerepet, hogy milyen pedagógiai célt szeretnék elérni vele.

- Bevezető feladatok esetében az a fontos, hogy jól elő tudja idézni az adott módszer felfedezését.
- Elmélyítő feladatoknál a megfelelő nehézségi szint és a feladat szépsége a fő szempont.
- Szintézis feladatoknál pedig az elsődleges, hogy milyen témakörök, és milyen ötletek jelennek meg benne.

Fontos megjegyezni, hogy ugyan a tantervben minden feladat egy-egy témakörhöz tartozik, amely a fő oktatási cél a feladat kitűzésekor, viszont minden feladatot a legjobb nyitott kérdésként feladni úgy, hogy ne tudhassák a diákok, hogy melyik témaköröket kell benne alkalmazni. Például a Mesterből vett feladatok esetén érdemes lehet leszedni a szöveg fejlécében található témamegjelölést. Így számos feladat megoldásakor több megközelítést is megvizsgálunk, például dinamikus programozásos feladatok esetén mohó ötleteket, vagy fordítva. Tehát lépten-nyomon előjön, hogy a megfelelő megoldási módszert is keressük, mint ahogy ez bármely új probléma megoldása során is történik.

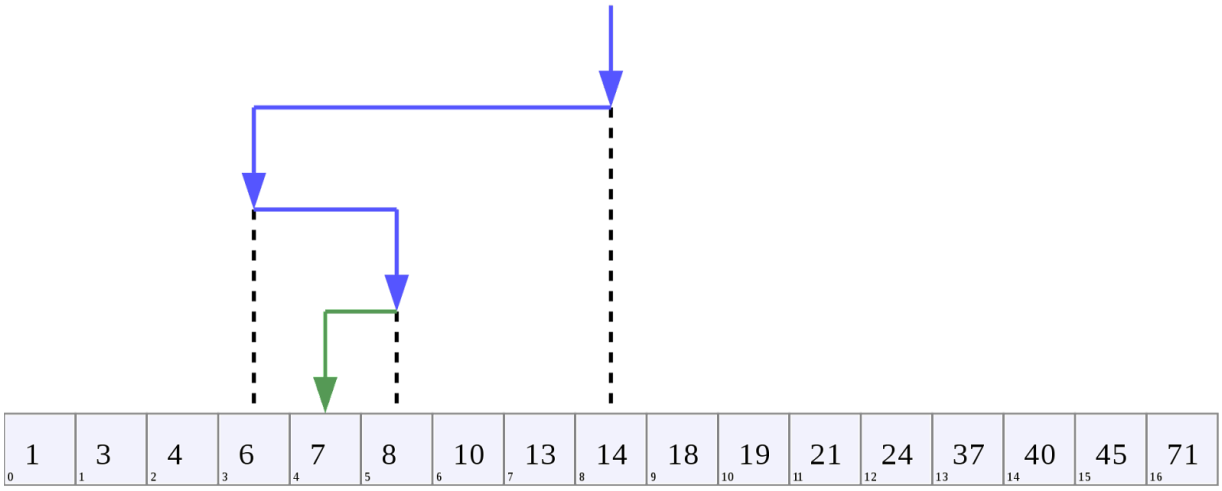
A következő fejezetben néhány feladatszálat bemutatok részletesen, ezzel szemléltetem kézzelfogható példákon a felfedeztető tanterv kidolgozását. Az összes feladat hasonló bemutatásához többszáz oldal lenne szükséges, így túlmutat e dokumentum keretein.

4.4. Részletes példák

4.4.1. Bináris keresés

A bináris keresés, más néven logaritmikus keresés algoritmusát rendezett sorozatokban alkalmazzuk. Ez gyakran felezéses módszernek is nevezhető (és igazából Pósa is így hívja), mivel lényege, hogy a keresett elemet tartalmazó halmazt minden lépésben felezzük (páratlan sok elem esetén nem pontos felezés lesz, de a lehető legközelebb hozzá). Ennek következtében a műveletigénye az elemek számának logaritmusa, ami rendkívül hatékony algoritmussá teszi.

Az alábbi ábra szemlélteti az algoritmus lépéseit egy konkrét rendezett tömbön, ahol a keresett elemet tartalmazó halmaz mindig a tömb egy intervalluma.



4.2. ábra: A bináris keresés vizualizációja egy példán (Forrás: Wikipedia)

A bináris keresés megvalósítása során érdemes nyilvántartani a keresési intervallum két szélét (ezt a fenti ábra nem szemlélteti). Az algoritmus pszeudo-kódja az alábbi (Python szintaxissal):

```
def binary_search(array, value):
    lower = 0
    upper = len(array)
    while lower + 1 < upper:
        middle = (lower + upper) // 2
        if array[middle] <= value:
            lower = middle
        else:
            upper = middle
    return lower
```

Az algoritmus során azt az invariáns tulajdonságot tartjuk fenn, hogy a keresett elem nagyobb vagy egyenlő az intervallum első eleménél, és kisebb az utolsó eleménél. Tehát a keresési intervallum ebben a megvalósításban az egyik végén zárt, a másik végén pedig nyílt, ami rendkívül egyszerűvé teszi az egyes eseteket: ha a középen vizsgált elem jó, akkor a zárt végét állítjuk oda, ha nem jó, akkor pedig a nyílt végét. Ezt addig kell ismételnünk, amíg a két határ egymás mellé nem kerül, hiszen akkor már csak egy elem jöhet szóba. A fenti függvény csak akkor adja vissza a keresett elem sorszámát, ha az szerepel a sorozatban. Ellenkező esetben az utolsó elem sorszámát adja vissza, amely kisebb vagy egyenlő a keresett értéknél (kivéve, ha a

sorozat minden eleme nagyobb nála). A kapott sorszám alapján ellenőrizhető, hogy a keresett elem található-e az adott pozícióban.

Az algoritmus nemcsak rendezett sorozatokban történő kereséshez alkalmazható, hanem szélsőérték-keresési feladatokra is. Olyan típusú kérdések megválaszolására használható, hogy mi a legkisebb vagy legnagyobb értéke egy változónak, amely esetén egy adott tulajdonság teljesül. Ahhoz, hogy helyes legyen az algoritmus, a vizsgált tulajdonságnak monotonnak kell lennie, vagyis egy adott érték alatt igaz, felette pedig hamis (vagy fordítva). Ezt az elképzelést az alábbi ábra szemlélteti.

4.3. ábra: A bináris keresés működésének feltétele, hogy a kívánt tulajdonság egy darabig teljesüljön (piros szín), majd pedig ne teljesüljön (kék szín).

Ebben az általánosított módszerben bevezetünk egy predikátumot, vagyis egy igaz-hamis függvényt, amely eldönti, hogy az éppen adott érték még megfelelő-e nekünk, vagy már nem. Valójában ugyanezt a módszert alkalmazzák a numerikus analízisben is, például folytonos függvények zérushelyeinek keresésére. Az angol szakirodalomban erre a módszerre a „bisection method” név használatos, és a programozásban is elterjedt ez a kifejezés, bár mellette ugyanannyira gyakori a „binary search” elnevezés is, annak ellenére, hogy az a sorozatban való keresésre utal. Magyarul a „bináris keresés” és a „logaritmikus keresés” elnevezéseket használjuk. Az általánosított bináris keresés algoritmusát így mutathatjuk be:

```
def binary_search(min_value, max_value):
    lower = min_value
    upper = max_value
    while lower + 1 < upper:
        middle = (lower + upper) // 2
        if good(middle):
            lower = middle
        else:
            upper = middle
    return lower
```

A fenti pszeudokódban látható, hogy a `good` függvény egy predikátum, ami eldönti, hogy az adott érték megfelel-e a követelményeknek. A keresést két érték között végezzük, a `min_value` és `max_value` között. Az algoritmus akkor helyes, ha kezdetben az alsó

határnál a predikátum igaz, míg a felső határnál már nem (zárt-nyílt tulajdonsága van a keresési intervallumnak).

Az alábbiakban bemutatom, hogyan segíthetünk a diákoknak felfedezni ezt a módszert, és hogy lehet számos haladó példán keresztül fejleszteni az alkalmazását.

4.4.1.1. A témakör bevezetése

A bináris keresés felezéses módszeréhez nagyon jó előzményeket tudunk biztosítani, mert egy egyszerű számkitalálós barkochba játék során remekül előhozható ez az ötlet. Az alábbi feladatot még a while ciklus tanításakor is feladathatjuk, és ez egybevág a felfedeztető tanítás azon jellemzőjével, hogy az alapozást jóval a témakör tanítása előtt kezdjük.

B1. *Írj barkochba programot, amely minél kevesebb lépéssel talál ki egy 1 és 100 közötti számot, amelyre a felhasználó gondol! Csak eldöntendő kérdéseket tehet fel a programod. Általánosítsd úgy, hogy a 100 helyére bármilyen $N \leq 10^9$ számot lehessen megadni felső határnak!*

A feladat nyitott, tehát nem csak olyan típusú kérdésekkel dolgozhatnak a diákok, hogy a gondolt szám egy adott számnál kisebb vagy nagyobb. Viszont másfajta kérdésekkel csak bonyolultabb vagy több kérdést használó megoldás jön ki, így nagy eséllyel a diákok a legkézenfekvőbb, felezéses módszert találják meg.

A témakörünk következő felvonásában a bináris keresést egy való életbeli helyzetben alkalmazzuk, amelynek modellezésére az alábbi feladatot adhatjuk:

B2. *Írj programot egy szótárban való keresésre! A szótárban szereplő N darab szó ábécé sorrendben adott. Ezt követően K szó mindegyikére ki kell írnod, hogy hányadik pozícióban szerepel a szótárban! (Minden kérdéses szó benne van a szótárban. $1 \leq K, N \leq 10^5$)*

A feladat során tehát egy rendezett sorozatban történik a keresés, ami számok helyett szavakat tartalmaz. Ez azonban nem okoz problémát, mivel a programozási nyelvekben rendelkezésre állnak operátorok vagy függvények a szövegek közötti összehasonlításhoz. A diákok a való életben előjövő helyzetben találják magukat, ahol a megoldási ötlet természetesen adódik, mivel egy szótárt sem érdemes az elejétől végiglapozni, hanem inkább a közepén kinyitva kell kezdeni a keresést. A megoldás során eljutunk az előző pontban bemutatott kereső algoritmushoz. Teljesen elfogadható, ha a diákok kezdetben lineáris kereséssel próbálkoznak, hiszen ezt tanulták korábban. Ekkor felmerülhet a kérdés, hogy hogyan lehetne ezt a folyamatot

gyorsítani. Ha maguktól nem jutnak el a felezéses módszerhez, akkor segítő kérdésekkel terelhetjük őket, például: „Képzeld el, hogy egy szótárban keresel, mit tennél?” Vagy további segítségként: „Mi lenne, ha valaki másnak lenne a szótár a kezében, és csak azt mondaná neked, hogy egy adott pozícióban melyik szó van? Hogyan tudnád kitalálni a keresett pozíciót?”

A következő feladat, ahol már nem magától értetődő a bináris keresés alkalmazása, a *Codeforces 706B, Interesting Drink* című feladat lehet. A feladat szövege röviden:

B3. *Vaszilij kedvenc üdítőitalát N boltban lehet kapni, más-más árakon. K napon egymás után szeretné ezt megvásárolni, minden napra adott, hogy mennyi pénze van. Add meg, hogy az egyes napokon hány különböző boltban tudná megvásárolni az italát! ($1 \leq K, N \leq 10^5$)*

A megoldás első lépése az, hogy a boltok árait növekvő sorrendbe rendezzük. Ezt követően minden egyes napi pénzösszeget bináris kereséssel kereshetünk ebben a sorozatban, hogy meghatározzuk, hány olyan bolt van, ahol az ár kisebb vagy egyenlő a megadott pénzösszeg. Egyáltalán nem nyilvánvaló helyzetben jelenik meg itt a bináris keresés. Két fontos lépés van a megoldásban, ami túlmutat a korábbiakon: fel kell ismerni, hogy a rendezés segít, és azután nem egy konkrét elemet keresünk a sorozatban, hanem azt akarjuk megtudni, hogy hány elem van benne, ami egy bizonyos korlát alatt van. Ez a helyzet hasonló Pósa matematikatáborainak stílusához, ahol gyakran más szituációkban kell alkalmazni az előzőleg tanult elveket, ezzel fejlesztve a diákok kreatív gondolkodását. A következő fejezet pedig egy még nagyobb lépésről szól.

4.4.1.2. Egy új helyzet: szélsőérték keresés

A problémaközpontú tanulás elveit követve, ebben az esetben is egy probléma felvetésével kezdünk, erre megfelelő lehet például a *Codeforces 760B, Frodo and Pillows* című feladat:

B4. *Frodó házában N hobbit alszik egy sorban egymás mellett lévő ágyakon. Frodónak M párnát kell kiosztania úgy, hogy minden hobbit kapjon párnát, és legfeljebb eggyel kevesebbet, mint a szomszédja. Legfeljebb hány párnát tud osztani magának, ha a K-adik ágyon alszik? ($1 \leq K \leq N \leq M \leq 10^9$)*

Nem túl nehéz kiszámítani azt, hogy ha Frodó magának X párnát adna, akkor összesen hány párnára lenne szüksége ahhoz, hogy mindenki elégedett legyen. Ha ez a szám nem nagyobb, mint M, akkor Frodó meg tudja oldani a problémát X párnával; egyébként nem. Tehát létrehozunk egy predikátumot, ami eldönti, hogy Frodó tud-e X párnát adni magának. Az is világos, hogy ha Frodó nem tud X párnát adni magának, akkor nem tud többet sem. Így a

predikátum segítségével a korábban bemutatott bináris keresés segítségével $O(\log N)$ idő alatt megtalálhatjuk a maximális X párnamennyiséget, amit Frodó megengedhet magának.

A megoldáshoz egy komoly ötlet kell: fel kell fedezni, hogy ha van egy predikátumunk, ami eldönti, hogy egy adott érték megfelelő-e még, akkor a felezéses módszerrel gyorsan tudunk szélsőértéket keresni. A felfedeztető módszertan része, hogy ilyen helyzetekben megfelelő segítséget tudjunk adni. Ezért a diákoknak feladatként adnám, hogy írják meg ezt a predikátumot, ami eldönti, hogy Frodó X párnát tud-e magának adni. Ezután próbáljanak rájönni, hogyan kell ezt használni. Az első ötlet lehet, hogy egyesével növelik az X-et, amíg a predikátum nem vált hamisra. Ez remek kiindulási pont, mert így könnyen át lehet látni, hogyan cserélhető le ez a megoldás bináris keresésre.

A *Codeforces 670D2 Magic powder* egy olyan feladat, amelyet a módszer elmélyítésére adhatunk fel valamivel később:

B5. *Sütiket szeretnénk sütni, amelyekhez N összetevő (liszt, tojás, cukor, só stb.) kell. Egy sütihez szükséges mennyiségek adottak (a_1, a_2, …, a_N egész számok), és azt is tudjuk mennyi van nekünk ezekből (b_1, b_2, …, b_N). Valamint van P egységnyi varázsporunk is, ami minden összetevőt tud helyettesíteni. Legfeljebb hány darab sütit tudunk sütni? ($1 \leq N \leq 10^5$, $1 \leq a_i$, b_i $P \leq 10^9$)*

Ebben a feladatban a korábban látott trükköt egy kicsit bonyolultabb helyzetben kell alkalmazni. A predikátumunk most azt dönti el, hogy tudunk-e X darab sütit sütni. Ehhez kiszámoljuk, hogy X sütihez mennyi varázspor kell, majd megvizsgáljuk, hogy van-e elég belőle. Végig kell mennünk az összetevőkön, és össze kell számolnunk a hiányzó mennyiséget, ami $O(N)$ lépés, tehát egy jóval műveletigényesebb predikátumról beszélünk itt. A teljes megoldás tehát $O(N \cdot \log P)$ lépésben fut le, ami gond nélkül belefér az időlimitbe. Ezen a ponton tudatosíthatjuk, hogy ha találunk egy algoritmust a megoldhatóság eldöntésére, akkor annak segítségével szélsőértéket is tudunk keresni a komplexitás szempontjából nagyon kis szorzófaktorral.

Ez egy megfelelő alkalom arra, hogy reflektáljunk a tanultakra, és a 4.4.1. pontban szereplő általánosított bináris keresést részletesen megbeszéljük a diákokkal. Ettől a ponttól kezdve az eszköztárunk része lesz a szélsőérték feladatok hasonló módszerrel való megoldása.

4.4.1.3. Haladó feladatok, kombináció más módszerekkel

Bemutatunk még két összetettebb példát, amelyek szerepe szintézis, a bináris keresés más módszerekkel együttesen alkalmazva jelenik meg. Ezen a szinten a bináris keresés már tulajdonképpen egy apró segédeszköz, amire a szélsőérték-keresési feladatoknál gondolni kell. A *2017/18. évi Informatika OKTV döntő Csatornák* című feladata így szól:

B6. *Egy város csomópontjait csatornahálózat köti össze. Ismerjük az egyes csatornák szélességét. Egy vállalkozás A csomópontból B csomópontba szeretne hajóval árut szállítani. Készíts programot, amely megadja a legszélesebb hajó szélességét, amely alkalmas a feladatra és az ehhez minimálisan bejárandó csatornák számát!*

A bináris keresést itt egy gráfelméleti feladatban alkalmazzuk, szélességi bejárással kombinálva. A predikátum segítségével azt szeretnénk eldönteni, hogy egy X szélességű hajó el tud-e jutni az A pontból a B pontba. Ezt úgy tehetjük meg, hogy csak a legalább X szélességű csatornákon hajtjuk végre a szélességi bejárást a két pont között. A bináris keresés ezen predikátum használatával megadja a hajó szélességének maximumát, és a szélességi bejárás révén a legrövidebb út hosszát is meghatározzuk.

A következő feladat pedig egy szép példa a *2010. évi Nemzetközi Informatikai Diákolimpiáról*, melynek címe *Quality of Living*:

B7. *Adott egy R×C méretű téglalap, amely minden természetes számot pontosan egyszer tartalmaz 1-től R·C-ig. Meg kell határoznunk azt a minimális P számot, amelyre igaz, hogy mediánja lesz valamelyik H×W méretű téglalapban lévő számoknak.*

Milyen predikátumot kell megvalósítanunk ahhoz, hogy alkalmazhassuk a bináris keresést? Arra kell tudnunk választ adni, hogy egy adott X értéknél lehet-e kisebb medián. Ezt meg tudjuk állapítani, ha minden H×W méretű téglalapra meg tudjuk mondani, hogy hány X-nél kisebb elem van benne, mert ha egy téglalapban az elemek többsége kisebb, mint X, akkor a medián is kisebb nála. A számokból 0-1 mátrixot képezünk aszerint, hogy X-nél kisebbek vagy sem. Majd erre a mátrixra 2D prefix összegeket számolunk: minden bal felső saroktéglalapra kiszámoljuk a számok összegét. Ezzel lehetőségünk nyílik arra, hogy minden egyes téglalapban konstans műveletben megállapítsuk, hány 1-es található. Összességében tehát $O(R \cdot C)$ művelettel megvalósítható az eldöntés.

Ebben a feladatban a kihívást nem maga a bináris keresés elve jelenti, hanem az, hogy hogyan alkalmazzuk azt sikeresen. Olyan módszereket használunk, amelyeket a diákok más tanulási

szálak mentén sajátítanak el, így ez a feladat is szintetizáló típusú, hasonlóan az előzőhöz, amely összekapcsolja a különböző feladatszálakat. Ehhez hasonló helyzeteket próbálunk teremteni a problémák megfelelő megtervezésével.

4.4.2. Dinamikus programozás

Sok oktatási anyag a dinamikus programozást (DP) a rekurziót követően vezeti be, mint egy módszert a többszörös rekurzió használata során fellépő műveletigénybeli problémák leküzdésére. A tantervemben egy alternatív felépítést javaslok, amiről nem állítom, hogy feltétlenül jobb, mint másoké, de adott helyzetben, például a táboraimban jól működik, és úgy gondolom, hogy jobban illeszkedik a táborba járó diákok előképzettségéhez és a tábor céljaihoz. Erdősné (Erdősné & Zsakó, 2016) és Forišek (Forišek, 2015) kiváló áttekintést adnak a DP helyéről a népszerű algoritmusokat bemutató tankönyvekben, és jogosan érvelnek amellett, hogy ezek a könyvek nem nyújtanak jó megközelítést a DP paradigma bemutatására a középiskolás diákoknak. Mindkét cikk azt javasolja, hogy a DP-t a rekurziót követően tanítsák, Forišek még stratégiát is ad arra, hogyan alakítsunk át egy felülről lefelé rekurzív megoldást egy alulról felfelé DP megoldássá. Király azt javasolja, hogy a DP-t csak a rekurzió és a visszalépéses keresés, valamint a mohó algoritmusok után tanítsuk haladó programozási ismeretek szintjén (Király, 2011). Függetlenül a három említett szerzőtől, nagyon hasonló feladatokat választottam, hasonló sorrendben. Van egy kulcsfontosságú különbség: én alulról felfelé (bottom-up) stratégiával kezdek a felülről lefelé (top-down) helyett. Egyetértek Erdősné álláspontjával, hogy a Logo nyelv nagyon szilárd alapot nyújthat a rekurzióhoz már fiatalon, de sajnos az utóbbi években sok olyan gyerekkel találkozom, akik elkezdenek algoritmikus programozást tanulni anélkül, hogy valaha is foglalkoztak volna Logo-val. Ez az egyik oka annak, hogy nem szeretnék a rekurzióra támaszkodni.

Számos olyan fiatal gyereket tanítottam, akiknek könnyen érthetőek a tömbök, és már rutinosan használták őket jóval a függvények tanulása előtt, nem beszélve a rekurzív függvényekről. Sokszor a programozást és az algoritmusokat együtt tanítom, és a programozási nyelv elemeit eszköznek tekintem az algoritmusok tanításához. A DP fogalmát akár a függvények tanítása előtt is be lehet vezetni. Ebben a megközelítésben a DP egy tömb vagy táblázat kitöltésének felel meg, amelyben részeredményeket számolunk ki a korábban kiszámított részeredmények felhasználásával. A felfedeztető módszertanhoz jól illeszkedik ez a megközelítés, mivel számos feladat természetesen adódó megoldási elve ilyen, és ezek a feladatok, amelyek közül néhányat alább bemutatok, remekül használhatók a DP tanításához. Továbbá a „táblázat

kitöltéses” módszer sokkal ártalmatlanabbnak hangzik, mint a „részproblémákra bontás”, bár a felszín alatt az történik. Az alábbiakban a beillesztett képletek áttekintésével nyomon is követhető, hogyan válnak egyre bonyolultabbá a DP feladatok, ahogy haladunk előre a tananyagban.

A hagyományos, nem felfedeztető módszertanú tanításban, mint például Forišek megközelítésében is jól működik az, hogy egy egyszerű, tipikus mintapéldán bemutatjuk az általános problémamegoldási módszert. DP esetén Forišek a Fibonacci számokat használja ehhez, amelynek során a bemutatott rekurzív képletet átalakítja alulról felfelé számolható DP megoldássá, majd stratégiát ad arra, hogy összetettebb feladatoknál hogyan lehet ezt az átalakítást csinálni. A felfedeztető oktatásban azonban ez kevésbé járható út, mivel, ha például a Fibonacci számok kiszámítását feladnánk a diákoknak, akkor a legtöbben rögtön tömb kitöltésével, alulról felfelé számolva oldanák meg, a rekurzív függvény alak kihagyásával. Így nem kerülne elő a rekurzív képlet átalakításának problémája sem. Emiatt más feladatokkal ideális felépíteni a témakör tanítását, az alábbiakban bemutatok egy lehetséges felépítést, amely a saját tanítványaimnál bevált.

4.4.2.1. Bevezetés: Lépegetős DP, Játék DP, Kombinatorika

A dinamikus programozás bevezetéséhez használt feladatoknak közös vonása, hogy valamilyen helyzetben egy optimális útvonalat, legjobb lépéssorozatot keresünk. Ezért ezt a feladattípust „lépegetős” DP-nek nevezzük. Egy rendkívül egyszerű bevezetés, kiváló első feladat a témakörben az *AtCoder DP_A: Frog 1*, amely így szól:

P1. *Van N kő, az i-edik kő magassága h_i. Egy béka az 1. kőről akar eljutni az N. kőre minimális költséggel. Egy ugrással az i-edik kőről az i+1-edik vagy i+2-edik kőre ugorhat, ennek a költsége $|h_i - h_j|$, ahol j az a kő, amelyikre érkezett. Add meg a minimális költséget az N. kőre eljutáshoz!*

A megoldást így írhatjuk fel képlettel:

```
DP[i] = min(DP[i-1] + |h[i] - h[i-1], DP[i-2] + |h[i] - h[i-2]|)
DP[1] = 0, DP[2] = |h[2] - h[1]|
```

A feladat nagyon jó lehetőséget biztosít arra, hogy ellenpéldákkal cáfoljuk meg az egyes mohó algoritmusok helyességét, hiszen egy rövid számsorozat megfelelő lehet erre. Mivel a felépítésünkben ez a legelső DP feladat, teljesen természetes, hogy a diákok először mohó ötleteket mondanak, például „a következő két kő közül mindig ugorjunk arra, amelyiknél az

ugrás költsége kisebb", vagy „a lehetőségek közül mindig válasszuk azt, ami az N. kő magasságához közelebb áll" stb. Minden ehhez hasonló ötlet esetén érdemes megkérni őket, hogy próbáljanak olyan példát találni, amikor nem optimális megoldást ad a módszer. Kis idő múltával felmerül, hogy valójában nem jó eldönteni, hogy melyik lehetőséget válasszuk, meg kellene vizsgálni minden esetet ügyesen. Viszont ahhoz túl sokféle útvonal van, hogy legeneráljuk az összeset, és megvizsgáljuk őket, ehelyett jobb megoldás kellene.

Tanárként azt a kérdést tehetjük fel a diákoknak, hogy ha gép nélkül, kézzel kellene egy viszonylag hosszabb (pl. N=20) példát megoldani, akkor mit csinálnának? Érdemes is eljátszani egy konkrét példával, hiszen amikor egy nem triviális feladványt kell megoldani, elkezdenek módszerekben gondolkodni. És ekkor már nagyon gyakran jön az ötlet, hogy sorban számoljuk ki minden kőre az odajutás minimális költségét. Ezt lehet még további kérdéssel segíteni: mi lenne, ha először nem a legutolsó kőre próbálnánk meghatározni a minimális költséget? Egy példán sorban leírhatjuk a számokat, amik az egyes kövekre való eljutási költségek, és ezzel már természetesen adódik, hogy programmal ki lehet tölteni egy tömböt, amelyben minden elemet a már kiszámolt két megelőző elemből tudunk származtatni. A képlet olyan értelemben rekurzív lesz, hogy az előzőleg kiszámolt DP értékekre hivatkozunk, de nem szükséges rekurzióról és rekurzív függvényről beszélni, maradhatunk annál, hogy egy tömb értékeit sorban egymás után kiszámoljuk, és eközben felhasználjuk a korábbi eredményeket.

A feladat általánosítása az *AtCoder DP_B: Frog 2*, ahol már nemcsak a két következő kő valamelyikére ugorhat a béka, hanem a következő K kő valamelyikére:

P2. *Van N kő, az i-edik kő magassága h_i. Egy béka az 1. kőről akar eljutni az N. kőre minimális költséggel. Egy ugrással az i-edik kőről az i+1, i+2, …, i+K-adik kövek valamelyikére ugorhat, ennek a költsége $|h_i - h_j|$, ahol j az a kő, amelyikre érkezett. Add meg a minimális költséget az N. kőre eljutáshoz!*

A megoldást így írhatjuk fel képlettel:

```
DP[i] = min(DP[i-j] + |h[i] - h[j]|, j=1..min(K,i-1))
DP[1] = 0
```

Ez egy nagyon jó feladat a tanultak elmélyítésére, mert ha egy diák megértette az előző feladatban alkalmazott alapelvet, akkor ennek a feladatnak is menni kell neki, hiszen csak még egy ciklust be kell tenni a minimum-kiválasztásra.

Számos lépegetős DP feladatban egy kétdimenziós négyzetrács mezőin kell útvonalat keresnünk. A kétdimenziós tömbök (mátrixok) ismeretével ezek a feladatok is nagyon jó bevezetést tudnak adni a dinamikus programozáshoz. Egy könnyű feladat, ami kezdésnek jó a témakörben a *Mester / Haladó / Dinamikus programozás / 107. Kincsek a hegyoldalon*:

***P3**. „Egy jobbra-lefelé lejtő hegyoldalon kincseket helyeztünk el, amelyekből a lehető legtöbbet egyetlen szánkóval szeretnénk összegyűjteni. A szánkóval a bal felső sarokból indulhatunk, és lejtő irányba (azaz vagy jobbra, vagy lefelé) haladhatunk. Amelyik mezőn átmegyünk, az ott levő kincset felvesszük. A hegyoldalon kijelöltek néhány gyűjtőpontot, a szánkóval valamelyikhez el kell jutnunk (és onnan tovább nem mehetünk). Készíts programot, amely megadja azt a gyűjtőpontot, ahova a legtöbb kincset vihetjük, valamint az oda vihető kincsek maximális számát!”*

A cellákba vihető kincsek maximális mennyiségének kiszámítása természetesen adódik, kiváltképp mivel a békás feladatok előkészítik ezt. Ha mégsem, akkor tudunk segíteni a diákoknak, például úgy, hogy mutatunk egy bonyolult példát papíron, és megkérjük a diákokat, hogy oldják meg kézzel, ennek hatására nagyon jó eséllyel kitalálják a módszert. Ha ez nem megy, akkor is jó először a példán, konkrét számokkal végigbeszélni, aztán az általános megoldást formalizálni.

0	❖ 1	1	1	1	1
0	1	❖ 2	2	2	❖ 3
❖ 1	1	2	❖ 3	❖ 4	**4**
1	1	❖ 3	❖ 4	4	❖ 5
❖ 2	2	**3**	❖ 5	5	5
❖ 3	❖ 4	❖ 5	**5**	❖ 6	**6**

4.4. ábra: A Kincsek a hegyoldalon feladat megoldása egy konkrét példára, táblázat kitöltéssel

A DP megoldás képletét így írhatjuk fel:

```
DP[i,j] = max(DP[i-1,j], DP[i,j-1]) + T[i,j]
DP[i,0] = 0, DP[0,j] = 0
```

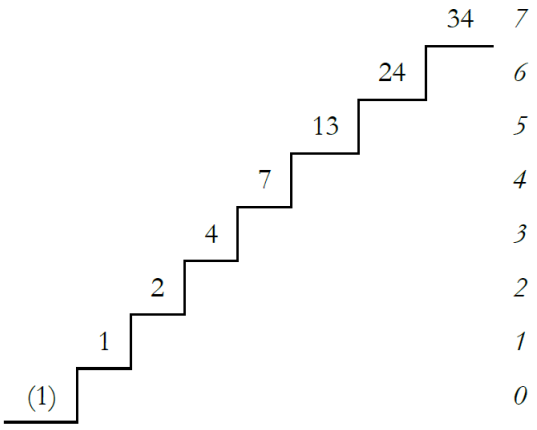
Elmélyítésként egy másik, négyzetrácson pontgyűjtögetős típusú feladatot javaslok, ahol a lehetséges lépések más irányúak: *Mester / Haladó / Dinamikus programozás / 116. Pontgyűjtő*

verseny. További hasonló, de bonyolultabb feladat a *Mester / Haladó / Dinamikus programozás / 7. Benzin*, ahol már az útvonalat is elő kell állítani. Ugyanebben a négyzetrácson lépegetős DP kategóriában egy nehéz feladat a *Codeforces 429B Working out*, amit használhatunk később, a tanultak felelevenítésére.

A lépegetős DP témakörrel párhuzamosan általában fut egy kombinatorika feladatokból álló szál is. Egy nagyon fontos kapcsolódási pont a két szál között, hogy a kombinatorika feladatok megoldását gyakran dinamikus programozással számolhatjuk ki. A Pascal-háromszög elemeinek kiszámítása is tekinthető DP feladatnak. Egy kiváló bevezető feladat a DP alkalmazására kombinatorika feladatokban a *Mester / Haladó / Kombinatorika / 11. Lépcsők:*

C1. *„Az iskola bejáratánál N lépcsőfok van. Egyszerre maximum K fokot tudunk lépni, ugrani fölfelé. Minden nap egyszer megyünk be az iskolába. Készíts programot, amely megadja, hogy hány napig tudunk más és más módon feljutni a lépcsőkön!”*

Javasolom, hogy ezt a feladatot hamar adjuk fel, rögtön az első néhány lépegetős DP feladat után. Nagyon jól tetten érhető a DP alulról felfelé építkező jellege, ahogy az egyes lépcsőfokokra jutás különböző módjainak számát kiszámoljuk sorban.



4.5. ábra: Az egyes lépcsőfokokra hányféleképpen juthatunk fel, ha a maximális lépés mérete K=3.

A Lépcsők feladat megoldásának képlete:

```
DP[i] = DP[i-1] + DP[i-2] + … + DP[i-K]
DP[0] = 1, a DP értékeket negatív indexeken 0-nak tekintjük
```

Elmélyítő feladatnak alkalmazhatjuk a feladat nehezebb variánsát: *Mester / Haladó / Kombinatorikai algoritmusok / 8. Hibás lépcsők*.

A lépegetős DP – kombinatorika témakörök kapcsolódási pontjaként még egy klasszikus és fontos feladat a *CSES 1638 Grid Paths*:

***C2**. Képzeljünk el egy N×N méretű rácsot, amelynek négyzeteiben csapdák lehetnek. Nem szabad olyan négyzetre lépni, ahol csapda van. A feladatod az, hogy kiszámítsd, hány út vezet a bal felső négyzetből a jobb alsó négyzetbe. Csak jobbra vagy lefelé mozoghatsz.*

A megoldás során lényegében ötvözzük a *Lépcsők* és a *Kincsek a hegyoldalon* megoldásában látott módszereket:

```
DP[i,j] = DP[i-1,j] + DP[i,j-1], ha (i,j) nem csapda, egyébként 0
DP[i,0] = 0, DP[0,j] = 0
DP[0,1] = 1
```

Egy harmadik szál kicsit később kezdve, de az eddigiekkel párhuzamosan fut: egyszerű két fős kombinatorikus játékok elemzése és optimális stratégiák kialakítása a nyerő és vesztes pozíciók meghatározásával. Ez egy remek lehetőség arra is, hogy offline is elkezdjük ennek a világnak a felfedezését azzal, hogy ténylegesen játszunk néhány egyszerű játékot és számítógép nélkül megtaláljuk a nyerő stratégiájukat.

Egy nagyon egyszerű feladat a *HackerRank Game of Stones*:

***G1.** Kezdetben van N kő egy halomban. Két játékos felváltva vehet el 2, 3, vagy 5 követ egy lépésben. Aki nem tud elvenni, az veszít. Ki fog nyerni, ha mindketten optimálisan játszanak?*

Ebben az esetben is a megoldáshoz vezető út az, hogy nem kapásból N kőre próbáljuk megválaszolni, hanem ugyanazt a kérdést megoldjuk 1, 2, ..., N-1 kőre is. Így máris megérkeztünk a DP módszeréhez, az alulról felfelé kiszámítási módon. Mivel a feladat meglehetősen egyszerű, ezen a ponton jó bevezetni a nyerő és vesztő játékállapotok fogalmát. A DP értékek jelentése ebben az esetben az, hogy az adott számú kőről indulva a kezdő játékos megnyeri-e a játékot. A kiszámítási képlet:

```
DP[i] = nem(DP[i-2] vagy DP[i-3] vagy DP[i-5])
DP értékek a negatív indexeken hamisnak tekintendők
```

A feladat általánosítását rögtön ezt követően fel tudjuk adni, ez az *AtCoder DP_K Stones*:

G2. *Egy halomban kezdetben van K kő. Két játékos felváltva vehet el a_1 vagy a_2 vagy ... vagy a_N darab követ egy lépésben. Aki nem tud elvenni, az veszít. Ki fog nyerni, ha mindketten optimálisan játszanak?*

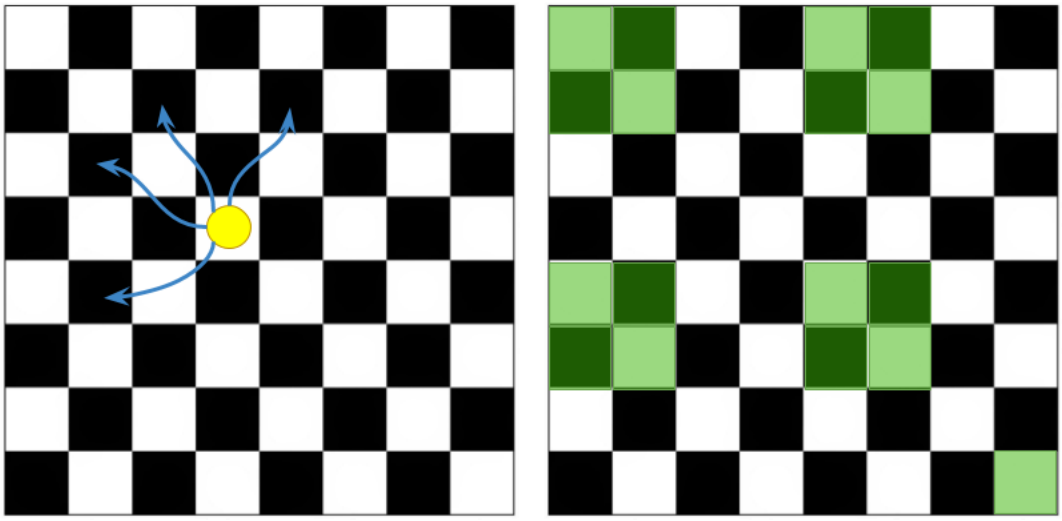
Itt tehát annyi a változás, hogy nem konkrétan 2, 3 vagy 5 követ lehet elvenni, hanem előre megadott N-féle lépés közül lehet választani. A megoldási ötlet ugyanaz, viszont technikailag

egy kicsit összetettebb kivitelezni, mert ciklus kell a lépési lehetőségek megvizsgálásához. A képlet:

```
DP[i] = nem VAGY(DP[i-a[i]], i=1..N)
DP értékek a negatív indexeken hamisnak tekintendők
```

A témához kapcsolódó megerősítő feladatunk egy fogalmilag egyszerű, de programozás technikailag bonyolult, kétdimenziós játék, amelyet sakktáblán játszanak: *HackerRank A Chessboard Game*.

G3. *Két játékos egy bábut mozgat egy sakktáblán. A bábut egy lépéssel az (x, y) mezőről az (x-2, y+1), (x-2, y-1), (x+1, y-2) vagy (x-1, y-2) mezők valamelyikére lehet mozgatni. A játékosok felváltva lépnek egy adott mezőről indulva, aki nem tud lépni veszít. Határozd meg egy (a, b) kezdőmezőre, hogy ki fog nyerni, ha mindketten optimálisan játszanak!*



4.6. ábra: A megengedett lépések és a nyerő mezők a sakktáblás játékban.

Az előzmények ismeretével világos, hogy a nyerő és vesztő mezők meghatározásával ki tudjuk elemezni a játékot. Azonban a fenti bal oldali ábrán szemléltetett lehetséges lépések alapján jól látszik, hogy akár soronként, akár oszloponként haladva nem tudjuk sorban eldönteni minden mezőre, hogy nyerő vagy vesztő, hiszen hivatkoznánk egy még ki nem számolt értékre. Akkor hogyan lehet csinálni? Mi garantálja egyáltalán, hogy véget ér a játék? Észrevehetjük, hogy a bábu mindig egy olyan másik átlóra lép át, ami a bal felső sarokhoz közelebb van (az x+y érték csökken minden lépéskor). Ez garantálja a játék végességét, és egy kiszámítási sorrendet is ad: haladjunk átlós sorrendben. A képlet:

```
DP[i,j] = nem(DP[i-2,j+1] vagy DP[i-2,j-1] vagy
              DP[i-1,j-2] vagy DP[i+1,j-2])
DP értékek a táblán kívül hamisnak tekintendők
```

Ezen a ponton, amennyiben a diákoknak már stabil tudása van a rekurzióról, megmutathatjuk nekik a rekurzív megoldást memorizálással kombinálva, és szemléltethetjük a rekurzió erejét: nem kell nekünk kitalálni a kiszámítási sorrendet, a számítógép automatikusan kitalálja helyettünk. A programozástáborokban majdnem a DP-vel együtt, esetleg némileg eltolva vezetjük be a rekurziót. A két problémamegoldási stratégia közötti szoros kapcsolatot léptennyomon megvilágíthatjuk. A két téma párhuzamosan futása segíti a megalapozásukat. A sakktáblás játéknál láthattuk, hogy a rekurzív megoldás (memorizálással) egyes esetekben elegánsabb megoldáshoz vezet, ugyanakkor számos korábbi feladatban (pl. *Frog 1-2*, *Lépcsők*, *Grid Paths*, *Game of Stones*) csak túlbonyolítás lenne az alulról felfele történő tömbkitöltéshez képest.

4.4.2.2. A dinamikus programozás erejének felfedezése

Miután a tanulók megérezték a DP ízét olyan problémákon keresztül, ahol az átmenetek lépések egy játékszerű forgatókönyvben, kevésbé nyilvánvaló DP problémákat fogunk eléjük tárni különböző mesékkel tálalva. Ezt a szálat Hátizsák DP-nek neveztük el a nagyon reprezentatív Hátizsák-probléma után. Egy hasonló, de sokkal egyszerűbb feladattal, a Pénzváltás problémával kezdünk, ahol a kérdés az, hogy milyen pénzösszegeket lehet kifizetni valamilyen érmekészlettel (az érmék minimális számát még nem kérdezzük). Online beadható formában a feladat megjelenik például itt: *Mester / Haladó / Dinamikus programozás / 69. Nem kifizethető címlet*.

H1. *„Egy országban N-féle címlet használatos. Készíts programot, amely megadja, hogy mely 1 és M közötti pénzösszegeket nem lehet a használható címletekkel pontosan kifizetni!”*

A megoldás egyetlen igaz-hamis tömbbel modellezhető, és a kifizethetőség eldöntése minden egyes egész számú összegre egyre inkább intuitív ötlet (programozás technikailag nagyon hasonló az *AtCoder DP_K Stones* feladathoz). Ha az érmék értékeit C[1..N] jelöli, akkor a következőképpen fogalmazhatjuk meg a megoldást:

```
DP[i] = VAGY(DP[i-C[j]], j=1..N)
DP[0] = igaz
DP értékek a negatív indexeken hamisnak tekintendők
```

A kifizethetőség után a következő lépés az érmék számának minimalizálása, a *CSES 1634 Minimizing Coins* feladat. Ez nagyon tanulságos példa a mohó megoldás helytelenségére is. Egy másik variációja a pénzváltás problémának a *CSES 1745 Money Sums*, ahol nem áll rendelkezésre akármennyi az egyes címletű pénzérmékből, tehát a bemenetben szereplő érmék csak egyszer használhatók. Hogy tudjuk kezelni ezt a problémát? Két módszerrel is lehet, bevezethetünk még egy változót (DP[i,j] jelentése az, hogy az első i pénzérmével kifizethető-e j összeg), vagy pedig megfordítjuk a kiszámítási sorrendet, a fenti DP képletet i=M..1 -ig számoljuk.

A Hátizsák-probléma (*AtCoder DP_D Knapsack 1*) különleges szerepet tölt be a DP-tananyagunkban, hiszen ez az első alkalom, amikor egy egyáltalán nem nyilvánvaló részproblémákra bontást alkalmazunk két változóval.

H2. Van N tárgy, mindegyiknek adott a súlya és az értéke. Ki szeretnénk választani közülük néhányat úgy, hogy a súlyaik összege ne haladja meg a hátizsákunk K kapacitását, és az összértékük a lehető legnagyobb lehet. Mi a legnagyobb elérhető összérték?

Nagyon csábítóak a különböző mohó módszerek a feladatnál, és természetesen itt is remek alkalom nyílik arra, hogy a különböző mohó megoldásokhoz ellenpéldákat keressünk közösen, de erre érdemes készülni is tanárként. Még azután sem egyszerű megtalálni a megfelelő részproblémákat, miután megsúgtuk a diákoknak, hogy gondolkozzanak DP megoldáson. DP[i,j] jelentése itt az, hogy az első i tárgy felhasználásával egy j méretű hátizsákba legfeljebb mennyi összértéket pakolhatunk. Előzményként a *CSES 1745 Money Sums* feladat segít abban, hogy az i index bevezetésével tudjuk biztosítani, hogy minden tárgyat csak egyszer használjunk. A kiszámítási képlet, amennyiben S[i] az i-edik tárgy súlya és V[i] az értéke:

```
DP[i,j]= max(DP[i-1,j], DP[i-1,j-S[i]] + V[i])
DP[i,0] = 0, DP[0,j] = 0
DP értékek a negatív indexeken -∞ -nek tekintendők
```

Számos való életből vett probléma is megoldható ezzel a módszerrel. Például a *Mester / Haladó / Dinamikus programozás / 66. Munkagépek* feladatban taszkok két gép közötti elosztása visszavezethető a pénzváltás problémára. A *Mester / NT, OKTV, IOI válogató / OKTV 2018/19 2. forduló / 3. Vásár* feladatban egy kereskedő profitját kell maximalizálnunk, és ez egy Hátizsák-probléma megoldására redukálható.

4.4.2.3. DP alkalmazása gráfalgoritmusokban

A dinamikus programozás ismerete nélkülözhetetlen a későbbiek során néhány gráf algoritmusban. Két nagyon gyakori legrövidebb út algoritmus, a Bellman–Ford és a Floyd–Warshall algoritmus két különböző DP megoldás erre a problémára. Nagyon jól ki tudjuk használni ezt a tényt a tehetséggondozó programunkban, hiszen azok a diákok, akik nagyon szilárd DP alapokkal rendelkeznek, képesek lehetnek önállóan vagy kis segítséggel felfedezni a Bellman–Ford és a Floyd–Warshall algoritmusokat. Ennek elősegítése érdekében tanácsolhatjuk nekik, hogy próbálják meg a legrövidebb utakat keresni DP-vel. Ha szükséges, pontosíthatjuk is, a Bellman–Ford esetében végezzenek DP-t az úton lévő élek száma szerint. A Floyd–Warshall algoritmus sokkal trükkösebb ötletet igényel, a DP-t az úton lévő közbülső csúcsok szerint végezzük. Egy kiváló feladat, amely segíthet ebben a felfedezésben, a *Codeforces 295B. Greg and Graph*. Továbbá számos probléma megoldásakor használunk gráfalgoritmusokat és dinamikus programozást együtt, például topologikus rendezés és DP szintézise a *CSES 1680 Longest Flight* Route feladat, és egy még komplexebb szintetizáló feladat kombinatorika, DP, körkeresés és topologikus rendezés alkalmazására a *HackerRank Kingdom Connectivity* feladat.

4.4.2.4. Haladó klasszikus DP problémák

Mivel a dinamikus programozásnak oly sok alkalmazása van, folyamatosan visszatérünk hozzá nehezebb és nehezebb feladatokkal. Kiemelünk két klasszikus és markáns DP-vel megoldható problémát, ami a leghosszabb közös részsorozat (longest common subsequence, LCS), és a leghosszabb növekvő részsorozat (longest increasing subsequence, LIS). Ezek köré egy-egy problémaszálat építünk.

A leghosszabb közös részsorozat (*AtCoder DP_F LCS*) probléma megoldásához kétdimenziós DP tömböt használunk, ahol a DP[i, j] érték annak a részproblémának a megoldása, amelyben az első i és j elemet vesszük a két sorozatból. Ha A és B a bemeneti sorozatok, a DP képlet a következőképpen alakul:

```
DP[i,j] = max(DP[i-1,j], DP[i,j-1],
              DP[i-1,j-1]+1 ha A[i]=B[j])
DP[i,0] = 0, DP[0,j] = 0
```

Egy következő, elmélyítő feladat az LCS DP szálon egy szintén klasszikusnak mondható rokon probléma a *CSES 1639 Edit Distance*, amelyben két string egymásba transzformálásához

szükséges minimális lépésszámot kell megadni. Szép alkalmazása az LCS megoldásának a *Mester / NT, OKTV, IOI Válogató / Nemes Tihamér 2. 2018 / 19 3. forduló / 4. Jelek* feladat, ami lényegében a leghosszabb ismétlődő részsorozat problémája, amelyre szintén $O(N^2)$ műveletigényű dinamikus programozásos megoldás adható, viszont a memóriával a Jelek feladat korlátai mellett már spórolni kell.

Nagyon jól fel lehet építeni a leghosszabb növekvő részsorozat (LIS) probléma köré épülő szálat, hiszen számos szép feladat van ebben a témakörben. Az $O(N^2)$ DP megoldás egy jó első lépés, de aztán a cél, hogy eljussunk az $O(N \cdot \log N)$ algoritmushoz, amelyben bináris keresést is használunk. A *Mester / Haladó / Mohó algoritmusok / 19. Konténeroszlopok* feladat, ami lényegében megegyezik a *CSES 1073 Towers* feladattal, egy mohó módszerrel megoldható feladat, ami hihetetlen sokat tud segíteni a hatékony LIS algoritmus felfedezésében, és valamilyen értelemben a LIS duálisa is: azt kéri, hogy osszuk fel a sorozatot minimális számú csökkenő részsorozatra. A mohó döntések folyamán kialakuló megoldásban megfigyelhető a részsorozatok végének rendezett tulajdonsága, és kézenfekvő, hogy bináris kereséssel lehet gyorsítani. Ami nem kézenfekvő, hogy eközben gyakorlatilag a LIS hosszát is kiszámítjuk, és az egyben alsó korlátot is ad a megoldásra, mert ha találunk egy *k* hosszú növekvő részsorozatot, és felbontjuk a sorozatot *k* darab csökkenő részsorozatra, akkor látható, hogy nincs hosszabb növekvő részsorozat, és csökkenő részsorozatokra bontásnál sem keletkezhet kevesebb. De ezt csak akkor kezdjük elemezni, miután feladtuk a *CSES 1145 Increasing Subsequence* feladatot, ami a LIS kiszámítását kéri, és a kiemelkedő diákok ilyen előzmények után maguk is rájönnek a hatékony $O(N \cdot \log N)$ algoritmusra.

Megjegyzem, hogy a *Konténeroszlopok* vagy *Towers* feladatot a LIS DP problémaszálhoz sorolom előzmény feladatként, annak ellenére, hogy legmarkánsabban a mohó módszer fedezhető fel benne, a bináris keresés alkalmazásával. Tehát lehetne akár a GREEDY és a BINSEARCH szál találkozási pontja egy szintézis feladatként, és valójában a feladat feladásakor ez mind előjön, viszont tanári szemmel a tervezés szempontjából fontosabb az előkészítő szerepe. További érdekesség, hogy ugyanazt az algoritmust, ami a leghosszabb növekvő részsorozatot számolja ki, egyszer mohó algoritmusnak nevezzük, máskor pedig dinamikus programozásos módszernek hívjuk. DP módszer, ha úgy tekintünk rá, hogy a *DP[i]* érték azt jelenti, hogy egy *i* hosszú növekvő részsorozatnak mi a minimális végződése, és ezeket az értékeket frissítjük ahogy haladunk végig a sorozaton. Mohó algoritmus, ha azt a döntést helyezzük előtérbe, hogy a következő elemet mindig annak a részsorozatnak a végére

illesztjük, ami a legkisebb nála nagyobb vagy egyenlő elemre végződik. Ez a kettősség véleményem szerint nem jelent ellentmondást, hanem az algoritmus szépségét növeli.

Ragyogó alkalmazásai vannak a LIS-nek további feladatokban, amit a szálunk tartalmaz, például a *Mester / Haladó / Dinamikus programozás / 47. Kockákból legmagasabb torony*, amelyben tornyot kell építeni a lehető legtöbb kockából úgy, hogy adott a kockák mérete és súlya, és csak egy kisebb és könnyebb kockát helyezhetünk egy másik tetejére. Ugyanez a feladat más mesével a *Codeforces 4D Mysterious Present*. Ebben az egyik tulajdonság szerinti rendezés az első ötlet, és azt követően alkalmazhatjuk a LIS algoritmust. Még egy nevezetes alkalmazása a LIS-nek az a kérdés, hogy legalább hány mozgatással lehet növekvő sorrendbe rendezni egy sorozatot (ahol a mozgatás azt jelenti, hogy egy elemet kiveszünk, majd tetszőleges másik kettő közé beszúrjuk). A *Codeforces 269B Greenhouse Effect* feladat gyakorlatilag erre kérdez rá. A kritikus ötlet, hogy a helyben maradó elemek számát akarjuk maximalizálni, és máris látható a LIS alkalmazása. Nagyon nehéz feladat van ebben a témakörben, a *Codeforces 650D Zip-line*, amelyben az a kérdés, hogy elemek változtatásával hogyan változik a LIS hossza. Ez haladó adatszerkezetek nélkül is megoldható, csupán a LIS algoritmust kell nagyon mélyen és jól érteni. Még egy nehéz feladat, amit a hazai diákolimpiai válogatóra én javasoltam, a *Mester / NT, OKTV, IOI Válogató / IOI / CEOI Válogató 2020 / 3. Hadjárat*, viszont annak a megoldásához már szükséges az algoritmust adatszerkezetekkel is kombinálni.

4.4.2.5. DP intervallumokon és részhalmazokon

Egy külön szálat építhetünk olyan feladatokból, amikor a DP részproblémák egy-egy intervallumnak felelnek meg, például egy string esetén annak minden substringjére számoljuk ki a megoldást DP-vel. Angol nyelven szoktak rá „DP on substrings” vagy „DP on ranges” névvel is hivatkozni, a szálat én RANGE DP-nek nevezem. Szép feladat a *Mester / Haladó / Dinamikus programozás / 80. Rúd felvágása*, melyben a legolcsóbb megoldást keressük egy rúd darabokra vágására, amennyiben egy vágás költsége a kettévágott rész hosszával arányos. A DP megoldás során olyan értékeket számolunk ki, hogy mi a minimális költsége az i-ediktől a j-edik vágásig elvégezni a darabolást. A játék DP (GAME DP) szál és ennek a szálnak a találkozása a *Mester / Haladó / Dinamikus programozás / 22. Fehér és fekete korongok*, hiszen abban egy játékra kell optimális stratégiát számolni, és játékállapotok a DP részfeladatok, amelyek történetesen a kezdeti korongsor egy-egy intervallumának felelnek meg. A *Codeforces 607B Zuma* feladat palindrom substringekről szól, a *Codeforces 1132F Clear the*

String pedig azonos karakterből álló substringekről, ezek mind tipikus példák az effajta DP feladatokra. Egy cseppet meglepőbb alkalmazás, szintén nagyon szép feladat ezen a szálon a *Codeforces 1509C The Sports Festival*.

Még egy szál, amelyet BITMASK DP-nek nevezünk, olyan DP feladatokat foglal magába, ahol a megoldott részproblémák megfelelnek egy halmaz összes részhalmazának. Egy részhalmazt egy 0-1 sorozattal reprezentálhatunk, amelyet egyszerűen megfeleltetünk egy egész számnak kettes számrendszerben. Az átmenetek a DP ezen formájában általában a részhalmaz egy elemének hozzáadásával vagy eltávolításával járnak, amit bitmanipulációkkal, bitenkénti operátorokkal tehetünk meg. Ezt a szálat azután lehet jól bevezetni, amikor a részhalmazok egész számokkal való reprezentálását már használtuk olyan feladatokban, ahol kimerítő keresést végeztünk az összes részhalmazon (BRUTEFORCE szál). Egy jó példafeladat a *Mester / NT, OKTV, IOI Válogató / IOI / CEOI Válogató 2018 / 10. Vásárlások*, amelyben adott K termék ára N napon keresztül, az árak naponta különbözhetnek és az N nap alatt mind a K terméket meg szeretnénk vásárolni, de minden napon legfeljebb 1 terméket vehetünk meg. A megoldásban DP[i,H] értékeket számolunk ki, amelynek jelentése a minimális költség, ha az első i nap alatt a termékek H részhalmazát vásároltuk meg. Ezt követően remek gyakorló feladat a *Codeforces 580D. Kefa and Dishes*, valamint az *AtCoder DP_O Matching*. Szép alkalmazása a módszernek a *CSES 1653 Elevator Rides* feladat megoldása. A *CSES 1690 Hamiltonian Flights* feladat gráfelmélettel szintézise a módszernek, míg a *CSES 2181 Counting Tilings* kombinatorikával. A BITMASK DP jól kapcsolható a logikai szita módszeréhez is (inclusion-exclusion principle, INC-EXC szál), amivel még számos kombinatorika feladat megoldható.

A fentiekben a dinamikus programozás számos helyzetben való alkalmazását tárgyaltam, például bináris kereséssel együtt, gráfalgoritmusokban, egy sorozat intervallumain, vagy részhalmazokon bitmanipulációkkal, és további forgatókönyvek is léteznek, amelyeket itt nem emeltem ki. A DP témaköre nagyon mély, lehetnek rendkívül nehéz problémák, amelyek „tisztán” DP feladatok. Később, összetett adatszerkezetek által vezérelt szálakon lépten-nyomon előfordul DP, például számos adatszerkezet konstrukciós lépésében (RMQ Sparse Table, LCA, Fenwick-fa, LPS-tábla a KMP-ben), tehát a dinamikus programozás az egész tananyagunkban központi fogalom. Ebben a fejezetben áttekintést adtam róla, külön fókuszálva a bevezetésére, és bemutatva a mély tudás fejlesztésének lehetőségét sok feladat segítségével, hosszú időn átívelve.

5. Programozástáborok

5.1. A táborok bemutatása

5.1.1. Motiváció, célok

Pósa Lajos matematika tehetséggondozó táborai nagy hatással voltak rám diákként és segítőként is. Ez nem csak rám igaz, számos kollégám ugyanígy vélekedik, és manapság A Gondolkodás Alapítvány (továbbiakban Alapítvány) kereteiben sokkal nagyobb közösség végez matematikai tehetséggondozó tevékenységet Pósa módszereivel. Erről a metodikáról azóta számos cikk és doktori értekezés is született (Katona, 2022). Emellett diákként azt éreztem, hogy matematikából rengeteg lehetőségem van tehetséggondozó programokban részt venni (Pósa táborai mellett említhetjük az Erdős Iskolát, a Medve matektáborokat, a korábbi Zalamat által szervezett nyári táborokat, és a számos többnapos országos matematikaverseny döntőket), informatikában viszont nagyon kevés. Amikor elkezdtem az Alapítványban matematikatáborokat vezetni, megfogalmazódott bennem célkitűzésként, hogy programozás és algoritmusok tanítására is lehetne hasonló táborokat szervezni. A doktori kutatásom fő motivációja, hogy a táborok szakmai hátterét, tananyagát és pedagógiai módszertanát kidolgozzam. A kezdeményezésnek a ProgTábor nevet adtam.

A ProgTábor fő célja, hogy megmutassuk az erre fogékony diákoknak, hogy a gondolkodtató, algoritmikus jellegű feladatok megoldása számítógép segítségével egy rendkívül élvezetes tevékenység, miközben nagyon sokat tudnak tanulni, fejlődni. Kiemelt cél továbbá, hogy az algoritmikus programozási ismeretek tanítására felfedeztető tanítási módszert alkalmazzunk, amely minél közelebb áll a Pósa-féle matematika tanítási módszertanhoz. A tábor nagyívű célja pedig az, hogy a programozás iránt érdeklődő diákokból összetartó közösséget építsünk, ami hosszú távon lehetővé teszi a program kiterjesztését is. Nem elsődleges cél a versenyekre felkészítés, de mivel a problémák, amikkel foglalkozunk, mindenekelőtt programozási versenyeken fordulnak elő, ezért azt is várjuk, hogy a résztvevők körében nő a programozási versenyek népszerűsége, és a versenyeken nyújtott teljesítményük, így a magyar diákok szélesebb köre lesz képes kiemelkedő nemzetközi versenyeredmények elérésére is.

5.1.2. Résztvevők, kiválasztás, csoportok

A tábor résztvevői 6-12. osztályos tehetséges diákok, akik érdeklődnek a programozás és az algoritmusok iránt, illetve motiváltak a tanulásra. A diákokból 6-7. osztályos korukban választunk ki országos szinten 25-30 főt, akikből egy csoportot alakítunk. A csoport együtt marad 5 éven keresztül, viszont van lehetőség közben új diákok belépésére, de természetesen kilépésre is.

Egy tábori csoport indulásakor széles körben meghirdetjük a jelentkezést, és a jelentkezőkből a résztvevőket egy online feladatsor segítségével választjuk ki. A feladatsor megoldásához alapszintű programozás tudásra van szükség (változók, típusok, elágazás, ciklusok, tömbök, beolvasás-kiírás), tehát ez sajnos feltétele a jelentkezésnek. Viszont a feladatokat úgy válogatjuk össze, hogy a megoldásukban döntően a számítógép nélküli gondolkodás számítson, az implementáció ne legyen összetett. Szinte bármilyen programozási nyelv ismeretével lehet jelentkezni, mivel az online feladatsor számos különböző nyelven enged megoldásokat beküldeni (például C++, C#, Java, JavaScript, Pascal, Python, Rust, sok egyéb mellett). A programunknak része az, hogy megtanítjuk a C++ nyelv legfontosabb elemeit azoknak, akik más nyelven tanultak programozni, mivel a versenyfeladatok megoldásakor legtöbbször előnyt jelent a C++ használata más nyelvekhez képest. A résztvevők kiválasztásánál arra törekszünk, hogy legalább 20% lány legyen köztük, ezért előfordulhat, hogy különböző alsó határt szabunk meg fiúknak és lányoknak.

A csoportok tekintetében egy nagyon fontos kérdés, amit fel kell tennünk magunknak, hogy jó-e, ha 5 éven keresztül együtt marad a csoport, és az elsődleges tényező a csoportok kialakításakor az életkor, a következő a tudás. Nem lenne-e jobb időnként átalakítani a csoportokat, és homogénebb tudásszintű, de heterogénabb életkorú csoportokat képezni? A közösségépítés szempontjából az azonos életkorú és hosszú távon együtt maradó csoportok ideálisak, viszont a tudásbeli eltérések lassítják a haladást és nagyobb kihívást jelentenek oktatói nézőpontból. A ProgTáborok mintájaként szolgáló matematikatáborokban ez a jelenség sokkal kevésbé jelenik meg, mert a matektáborok tananyaga az iskolai tantervet kiegészíti, arra épít (de aztán jóval túlmutat rajta). A programozástáborokban lényegében nem létezik ilyen iskolai tanterv, amire építünk, minden ismeret extra ahhoz képest, amit az iskolában tanulnak, és emiatt több új ismeretet kell elsajátítaniuk a diákoknak a táborban, és jelentősen csökkent az élményen, ha ezeket máshonnan ismerik. A táborok indulásakor alig voltak olyan szakkörök, amiken a tábor anyagával átfedő algoritmikus programozást tanulhattak a diákok, viszont azóta

elindult számos online szakkör (amelyekről a 6. fejezetben írok), és emiatt nőttek a diákok közötti tudásbeli különbségek. Ez új kihívást jelent, és a jövőre nézve megfontolandó, hogy változtassunk a csoportösszeállítási elveken, vagy a tananyag fókuszán.

A tehetséggondozó program kezdete óta négy tábori csoportot indítottam, amelyek közül egy már végzett, három jelen pillanatban is futó csoport van. A legidősebbek ebben az évben végeznek, és ősszel egy új csoportot indítok. Az alábbi táblázat foglalja össze a csoportok legfontosabb jellemzőit. A létszám az évek során változik, hiszen új diákok csatlakozhatnak és a tagok közül néhányan ki is léphetnek a csoportból.

Csoport neve	Indulás - végzés (várható) éve	Évfolyam induláskor	Jelenlegi évfolyam	Létszám	Eddigi táborok száma
Pilot	2020 – 2022.	10.	egyetem	15-20	6
Alpaka	2021 – 2024.	8-9.	11-12.	26-30	9
Graphisoft	2022 – 2026.	7-8.	9-10.	23-26	8
Realogic	2023 – 2028.	6-7.	7-8.	25-28	4

5.1. táblázat: ProgTábor csoportok

5.1.3. Módszertan, tananyag

A 3. és 4. fejezetben részletesen bemutattam a módszertant és tananyagot, amelyet a ProgTáborokban használunk. Itt a táborra jellemző gyakorlati megvalósítást részletezem.

5.1.3.1. Csoportmunka és megbeszélések

A módszertannak fontos eleme a csoportmunka. A táborokban 2-3, esetleg 4 fős csoportok alakítását kérjük a diákoktól, tehát alapvetően ők választhatják meg, hogy kikkel szeretnének együtt dolgozni, de természetesen segítünk nekik egymásra találni. A csoportmunka alatt külön helyiségekben dolgoznak a diákok, közben az oktatók járnak körbe, meglátogatják őket, és beszélgetnek a gondolataikról, megoldásaikról, valamint segítenek az implementációban is. A tábor vezető tanára mellett van néhány segítő oktató, akiknek a csoportmunka alatti támogatás az elsődleges feladata. Tehát a 3.4.3. pontban bemutatott körmodell vizsgálat és kivitelezés lépése mindenképp a csoportmunka alatt történik, sőt rendszerint a megoldás lépése is, hiszen a diákok ideális esetben maguktól vagy kis segítséggel jönnek rá a megoldásra, amit külön meg is beszélnek egy oktatóval a csoportban.

A csoportmunkában a diákoktól azt várjuk el, hogy minden feladat esetében először gondolkodjanak külön-külön egy rövid ideig, majd csak utána beszéljék meg a gondolataikat, és ebben a diskurzusban ne vegyen részt az, aki ismerte korábbról a feladatot. Azt is kérjük, hogy aki rájött a megoldásra, ne „lője le” azt kapásból a csoport többi tagjának, hanem hagyja a gondolatmenetüket kibontakozni. Ideális esetben olyan csoporttagok beszéljenek egymással, akik azonos szinten vannak a megoldással, vagyis mindannyian megoldották, vagy még egyikük sem tudja a megoldást. Egyéb esetekben a csoportot meglátogató oktatóval beszéljenek a feladatról, például, ha egy két tagú csoport egyik tagja ismerte a feladatot, akkor a másikat egyénileg vezetjük rá a megoldásra.

A csoportmunka és a plenáris, frontális megbeszélések váltakozva következnek a tábor szakmai programjában. A megbeszéléseken a 3. fejezetben leírtaknak megfelelően átbeszéljük és letisztázzuk a megoldásokat, amiket a legtöbben már a csoportmunka alatt felfedeztek, másrészt elemezzük azokat és rendszerezzük is az alkalmazott módszerek tekintetében. Tehát a tanulás folyamatának körmodelljében az elemzés lépés a közös megbeszéléseken történik. Elvétve előfordul, hogy egy-két csoportban nem jönnek rá a megoldásra, és a közös megbeszélés során hallják azt, ilyenkor tehát a megoldás lépés számukra akkor valósul meg. Ebben az esetben az elemzés előtt még egy csoportmunkás foglalkozás alkalmával biztosítunk lehetőséget a megoldás implementálására, ami nagyban erősíti a megértést. Szintén a megbeszéléseken kerülnek elő új problémák, új feladatok. A plenáris foglalkozásokról felvételt is szoktunk készíteni annak érdekében, hogy a hiányzók pótolni tudják az anyagot, hiszen a táborok egymásra épülnek, a korábban tanultak ismerete szükséges ahhoz, hogy fejlődni tudjanak a diákok és élvezzék a táborokat.

5.1.3.2. Csapatvetélkedők

Van egy különleges csoportmunkás program minden táborban, ez pedig a csapatvetélkedő. A vetélkedő elsődleges célja az, hogy szombat késő délután, amikor már fáradtak lennének a diákok, egy adrenalinnal növelt extra programban felüdülve dolgozzanak újra a feladatokon. Ez Pósa matematikatáboraiban is így van. Szintén nagyon jó alkalom a csapatvetélkedő arra, hogy visszajelzést kapjunk arról, hogy a táborban tanult módszereket mennyire sikerült elsajátítaniuk, mennyire megy egy-egy új helyzetben az alkalmazásuk.

Két fajta csapatvetélkedőt szoktunk tartani. A gyakrabban alkalmazott módban egy versenyfeladatokból összeállított feladatsort kell megoldaniuk a csapatoknak, általában a Codeforces rendszerben. Viszont fontos változtatás a versenyekhez képest, hogy a résztvevők

limitált mennyiségben kérhetnek segítségeket a szervezőktől. A segítségeknek három típusát biztosítjuk: teszteset kérése, elméleti megoldáshoz segítség („hint”) kérése, és hibakeresési segítség. Világos, hogy így nem szavatolható az, hogy minden csapat teljesen ugyanabban a bánásmódban részesül, de egyrészt szervezőként igyekszünk így tenni, másrészt tudatosítjuk a diákokban, hogy a vetélkedő lényege, hogy élvezzék, nem pedig az, hogy milyen eredmények születnek a végén. Ezt tükrözi a díjazás is, csokoládét kapnak díjként, és minimális különbséget teszünk a teljesítményük alapján: vannak a nagyon jól dolgozó csapatok, és a még annál is jobban dolgozó csapatok, akik kicsit több csokit kapnak.

Alkalmanként csinálunk rendhagyóbb csapatvetélkedőt, amelyet bot vetélkedőnek nevezünk. Ennek során egy általunk kidolgozott egyszerű stratégiás játékra kell olyan programot fejleszteni, amely képes azzal a játékkal játszani, azaz egy számítógépes bot játékost írni. A diákok által írt botok aztán egymás ellen játszanak egy bajnokságot a játékban. Ennek lebonyolítására legtöbbször egy saját fejlesztésű, AI Arena elnevezésű rendszert használunk (AI Arena, 2024). A bot vetélkedők enyhén összetettebb programozás tudást igényelnek, bár a rendszer fejlesztésekor törekedtünk arra, hogy minél alacsonyabb legyen a belépési szint. A diákok komplex helyzetben, sokkal közvetettebben alkalmazhatják az elsajátított ismereteket, ahhoz képest, mint amikor egy jól körülhatárolt versenyfeladaton dolgoznak. További célja ennek a feladattípusnak, hogy színesítse a táborok szakmai anyagát.

5.1.3.3. Egy-egy tábor tananyaga, feladatok

Minden táborban van két-három fő téma, amelyek feladatszálain haladunk, amelyek közül legalább egy teljesen új, és emellett korábbi táborok témáiból is vannak feladatok, tehát más szálakat is folytatunk. A feladatokat négy kategóriába oszthatjuk, attól függően, hogy milyen helyzetben használjuk őket:

- **Normál tábori feladatok**: a csoportmunkás foglalkozásokon feladott feladatok, a tábor fő anyagát képezik, a feladatszálakon való haladást biztosítják. A 4.2. pontban meghatározott feladattípusokból a bevezető típusú feladatok szinte kivétel nélkül mindig normál tábori feladatok, továbbá lehetnek elmélyítő és szintézis feladatok is köztük.
- **Lekötő feladatok**: bonyolult feladatok a tábor anyagához tartozó feladatszálakból, amelyek nem szükségesek a téma elsajátításához, hanem haladó alkalmazásokról szólnak. Arra az esetre szolgálnak, ha az erős diákok végeznek a normál feladatokkal, akkor tudnak ezeken a feladatokon dolgozni. Alapelv a táborokban, hogy mindig legyen

feladat, amin gondolkodni lehet, sose unatkozzon senki. Típusukat tekintve elmélyítő vagy szintézis feladatok.

- **Csapatverseny feladatok**: Codeforces feladatok a szombat délutáni csapatversenyre. A nagy részük illeszkedik az aktuális tábor és korábbi táborok feladatszálaiba, de lehet köztük aranyos, önbizalomnövelő feladat és nehéz lekötő feladat is. Gyakori köztük a szintézis típusú feladat, amikor több témát kell együtt alkalmazni, de bőven vannak elmélyítő feladatok is a csapatversenyeken.
- **Házi feladatok**: a táborok közötti otthoni munkára adott feladatok, szintén a Codeforces rendszerben kitűzve. A céljuk a táborokban tanult ismeretek átismétlése, egy-egy új helyzetben való alkalmazása. A feladatszálak szerves részét képezik, a legtöbbjük elmélyítő feladat, de előfordult már, hogy egy következő tábori témához adtam fel előkészítésként bevezető típusú feladatot.

Egy táborban általában 10-12 normál feladat van, 4-6 lekötő feladat, 7-9 csapatverseny feladat és 8-10 házi feladat a következő táborra. A tábor tananyagát úgy érdemes összeállítani, hogy kiválasztjuk azt a két-három fő témát, amelyeken haladunk és áttekintjük azok feladatszálait, bevesszük a szükséges feladatokat belőlük. Másrészt a korábbi táborokban kezdett feladatszálakból a soron következő feladatokat is beillesztjük a tábor anyagába. További fontos lépés az, hogy az új szálak kapcsolódási pontjait keressük meg a korábbi szálakkal, hiszen egy-egy ilyen feladattal több témát tudunk egyszerre erősíteni. A csapatverseny összeállítása nagyon jó támpont tud lenni, hiszen sokféle nehézségű feladatot kell összeállítani a megadott témákból, így elég jól lehet előre látni, hogy mely feladatok fognak passzolni abba a problémakörbe. A normál tábori feladatok időbeli tervezését is meghatározza, hiszen a bevezető feladatokkal elő kell készítenünk a csapatversenyen feladott elmélyítő feladatokat.

Példaként az alábbiakban felvázolom a Realogic csoport 4. táborának anyagát, amelyet 2024. május 3-5-ig tartottunk, hiszen a fenti elvek kontextusba helyezve sokkal jobban bemutathatók. A tábort Leitereg András kollégámmal közösen terveztük meg és vezettük, számos fiatal segítő közreműködésével. Három fő szál volt, amelyet ezen az alkalmon indítottunk, a BACKTRACK (visszalépéses keresés), BRUTEFORCE (kimerítő keresés) és BINSEARCH (bináris keresés). Folytattuk a korábbi táborokban elkezdett BFS (szélességi bejárás), 2POINT (két mutató technika) és KNAP DP (hátizsák DP) szálakat, és a csapatversenyen előkerült még a PREFSUM (prefix összegek) és DFS (mélységi bejárás) téma is.

A tábori normál feladatok listája:

Sorszám	Feladatcím	Feladatbank és azonosító / Szöveg	Téma
1	Apple Division	CSES 1623	BRUTEFORCE, BITOP
2	Magic Powder - 1	Codeforces 670D1	BRUTEFORCE
3	Barkochba	Írj barkochba programot, amely minél kevesebb lépéssel talál ki egy 1 és 100 közötti számot, amelyre a felhasználó gondol! Csak eldöntendő kérdéseket tehet fel a programod. Általánosítsd úgy, hogy a 100 helyére bármilyen $N \leq 10^9$ számot lehessen megadni felső határnak!	BINSEARCH
4	N Bástya	Mester / Haladó / Visszalépéses keresés / 23.	BACKTRACK
5	Futár	Mester / Haladó / Gráfok, szélességi bejárás / 17.	BFS
6	Magic Powder - 2	Codeforces 670D2	BINSEARCH
7	Sum of Two Values	CSES 1640	2POINT
8	Money Sums	CSES 1745	KNAP DP
9	Chessboard and Queens	CSES 1624	BACKTRACK
10	Interesting drink	Codeforces 706B	BINSEARCH
11	BFS-DFS	CSAcademy BFS-DFS	BFS, DFS

5.2. táblázat: A Realogic / 4. tábor normál tábori feladatai

A lekötő feladatok listája:

Sorszám	Feladatcím	Feladatbank és azonosító / Szöveg	Téma
L1	Kötélhúzás	Mester / Haladó / Dinamikus programozás / 51.	KNAP DP
L2	Cycle In Maze	Codeforces 769C	BFS
L3	Lámpák	Mester / NT, OKTV, IOI Válogató / IOI/CEOI Válogató 2019 / 4.	BRUTEFORCE, BITOP
L4	Petya kérdése	Adott n ($1 \leq n \leq 100\ 000$) szám, lehetnek negatívak is közöttük ($-10^9 \leq a_i \leq 10^9$), valamint egy k szám. Maximum hány szomszédosat lehet kiválasztani közülük úgy, hogy az összegük legfeljebb k legyen?	PREFIX SUM, BINSEARCH
L5	Graph Cutting	Codeforces 405E	DFS
L6	Relay Race	Codeforces 213C	PATH DP

5.3. táblázat: A Realogic / 4. tábor lekötő feladatai

A csapatverseny feladatok:

Sorszám	Feladatcím	Feladatbank és azonosító / Szöveg	Téma
A	XOR Mixup	Codeforces 1698A	BITOP
B	Prefix Sum Addicts	Codeforces 1738B	PREFSUM
C	Burning Midnight Oil	Codeforces 165B	BINSEARCH
D	PFAST Inc.	Codeforces 114B	BRUTEFORCE, BITOP
E	Jumping on Walls	Codeforces 198B	BFS
F	Slime Escape	Codeforces 1734D	2POINT
G	Checkout Assistant	Codeforces 19B	KNAP DP
H	Graph Cutting	Codeforces 405E	DFS

5.4. táblázat: A Realogic / 4. tábor csapatverseny feladatai

A következő táborra a házi feladatok:

Sorszám	Feladatcím	Feladatbank és azonosító / Szöveg	Téma
A	XOR Mixup	Codeforces 1698A	BITOP
B	Worms	Codeforces 474B	BINSEARCH, PREFIX SUM
C	Preparing Olympiad	Codeforces 550B	BRUTEFORCE
D	Hamburgers	Codeforces 371C	BINSEARCH
E	Learning Languages	Codeforces 277A	DFS
F	Kefa and Company	Codeforces 580B	2POINT
G	Permutation Game	Codeforces 1033C	GAME DP
H	Smallest number	Codeforces 55B	BRUTEFORCE
I	Good Subarrays	Codeforces 1398C	PREFIX SUM
J	Police Stations	Codeforces 796D	BFS

5.5. táblázat: A Realogic / 4. tábor utáni házi feladatok

A tábor elején még nem végleges a feladatok listája, csak áttekintjük és kigyűjtjük az egyes szálak feladatait és a csapatversenyre jelölt feladatokat is, viszont fontos, hogy rugalmasak maradjunk és a csoport haladásához alkalmazkodjunk a tábor közben. A pénteki első foglalkozáson kitűzött feladatokat kell kiválasztani, ezek ebben a táborban az 1-5. feladatok és

az L1-L3. feladatok voltak. Az 1-4. feladatokkal elindítottuk mindhárom új szálat, és az 5. feladat egy nehezebb szélességi bejárásos feladat volt, amely egy fontos ötletet tartalmaz. A lekötő feladatok közül az első kettő közepesen nehéz feladat a korábbi szálakból, a harmadik pedig egy nagyon nehéz és szép feladat az egyik új szálhoz kapcsolódóan.

Szombaton délelőtt feladtuk a 6-8. feladatokat, amelyek közül a 6. volt az első olyan helyzet, amikor bináris keresést alkalmaztunk szélsőérték-keresésre, míg a 7. és 8. korábbi szálakat folytattak. Ezzel elő is készítettük a szombat délutáni csapatversenyt minden témakörében, amelyek változatos témákból, 8 különböző szálból voltak, vegyes, egyre növekvő nehézségben. Törekedtünk arra, hogy az új témakörök elmélyítésére legyenek viszonylag könnyen oldható Codeforces feladatok, a nagyobb kihívást jelentőek pedig korábbi táborok anyagához kapcsolódjanak.

Vasárnap délelőttre még bőven maradt feladat korábbról, ahogy ez szokott is lenni, hiszen célunk, hogy adjunk időt a gyerekeknek gondolkodásra, ne túl hamar beszéljük meg a megoldásokat. Emellett még a 9-11. feladatokat tűztük ki, amelyek összetettebb feladatok voltak a BACKTRACK, BINSEARCH és BFS szálakból. Feladtunk új lekötő feladatokat is (L4-L6), amelyek közül az L4 egy olyan kérdés volt, amelyet az egyik diák vetett fel csatlakozó kérdésként a 7. feladat megoldása után.

A házi feladatokat a tábor után egy héttel állítottuk össze, és tetten érhető, hogy igyekszünk minél több szálon elmélyítő feladatokat adni, különösen a táborban indított szálakon.

5.1.4. Megvalósítás, helyszín, időbeosztás

A ProgTáborok alapvetően hétvégi táborok, egy tábor általában péntek délután 5 órától vasárnap délután 4 óráig tart. Hosszú hétvégék esetében lehet egy nappal hosszabb. Másrészt, minden csoportnak nyáron is szoktam egy tábort tartani, amelyek lehetnek hétköznapokon is, és általában fél nappal hosszabbak, például kedd délelőtt 10 órától csütörtök délután 5 óráig.

A táborok helyszíne az AIT Budapest Campus, egy magánegyetem Budapesten, ahova hétközben amerikai egyetemisták járnak, hétvégén pedig használhatjuk az épületet mi. Az épület beosztása ideális a tábornak olyan szempontból, hogy a közös megbeszéléseknek van nagy terem, és számos kisebb terem és tanulószoba van, ahol a diákok tudnak csoportban dolgozni külön helyiségekben. Minden teremben és tanulószobában van whiteboard, ami elősegíti a közös gondolkodást. A 20-30 diákból 7-11 csoport szokott kialakulni, tehát ennyi külön helyiségre szükség van.

Egy hétvégi tábor ideális időbeosztása:

Péntek

- 16:30 - Érkezés
- 17:00 - 18:30 - Házi feladatok megbeszélése, kérdésekre válasz
- 18:00 - 19:30 - Új feladatokon gondolkodás és programozás csoportmunkában
- 19:30 - 22:30 - Vacsora, esti játék

Szombat

- 9:00 - 13:00 - Csoportmunkás foglalkozások és közös megbeszélések váltakoznak, új témaköröket tanulunk, közben 30 perc szünet
- 13:00 - 14:00 - Ebédszünet
- 14:00 - 16:00 - Sport, szabadidős tevékenységek (gépek nélküli program)
- 16:00 - 17:00 - Korábbi feladatok megbeszélése
- 17:00 - 19:30 - Csapatverseny
- 19:30 - 22:30 - Vacsora, esti játék

Vasárnap

- 9:00 - 13:00 - Csoportmunka, megbeszélések, közben 30 perc szünet
- 13:00 - 14:00 - Ebédszünet, szabad program
- 14:00 - 15:00 - Csoportmunka
- 15:00 - 16:00 - Végső megbeszélés, táborzárás

A ProgTábor program indításakor 2020-ban a járványügyi korlátozások miatt az első tábort csak online tudtuk megtartani. A táborok online megvalósításáról részletesebben írok a 6. fejezetben. Azóta is, minden induló csoport első, bevezető táborát online tartjuk, mivel sokkal kisebb kötöttségekkel és költségekkel jár.

5.2. Alkalmazott eszközök

5.2.1. Discord

A táborokban a Discord kommunikációs platformot használjuk a következő célokra:

- Foglalkozások közben linkek, fájlok gyors megosztása
- Feladatok kitűzése (legtöbbször egy link egy online feladatbankban)
- Megoldások beküldése
- Csoportmunkában egy privát csatornában linkek, fájlok megosztása egymással

- Segítők hívása a csoport szobájába
- Csapatverseny alatt a megengedett segítségek kérése
- Táborok közötti kapcsolattartás, házi feladatok küldése, visszajelzések gyűjtése
- Felvételek megosztása
- Online tábor tartása

Négy fő szempont miatt esett a választásunk a Discord platformra:

- Online tábor esetén nagyon könnyű a csoportmunkás foglalkozásokat megtartani a külön hangszobák segítségével, amelyek között egyszerű az átjárás és jól menedzselhetők.
- A szöveges csatornák arra is eszközt nyújtanak, hogy a feladatokat, tananyagokat visszanézhető és áttekinthető formában publikáljuk a diákoknak.
- A rangok segítségével könnyedén menedzselhetőek a különböző csatornákhoz való hozzáférési jogok.
- Saját Discord chatbot írásával számos folyamat jól automatizálható.

A Discord tehát kommunikációs csatorna, közösségi tér és tananyagok gyűjtőhelye is egyben. Minden tábori csoportnak van egy saját szervere (a Discordban egy csoport közös virtuális terét szervernek hívják). Fontosnak tartjuk, hogy rendszerezzük a megosztott tartalmakat, ennek érdekében minden szerveren megtalálhatók az alábbi szöveges csatornák:

- #feladatok-anyagok: kizárólag a táborban kitűzött feladatok és hozzájuk tartozó tananyagok vannak ebben a csatornában.
- #megoldások: minden feladatra egy-két kiválasztott szép megoldás van itt, amelyeket a megbeszéléseken elemzünk.
- #mit-tanultunk: rövid lista arról, hogy az egyes táborokban mik voltak a fő témakörök, mire kell emlékezni a jövőben.
- #felvételek: a táborban a megbeszélésekről készült felvételeket tesszük közzé itt.
- #general: általános kommunikációra használt csatorna, legtöbbször logisztikai, szervezési üzeneteket küldünk itt.
- #off-topic: vicceknek, mémeknek, a táborhoz kevésbé kapcsolódó tartalmaknak fenntartott csatorna.
- #mentorok-privat: csak az oktatók számára látható csatorna, belső kommunikációra.
- #csop1, #csop2, ..., #csop12: az egyes csoportok belső kommunikációjára fenntartott privát csatorna, amihez a csoporttagok és a mentorok férnek hozzá.

5.2.2. ProgTábor Bot

Fejlesztettünk a táborokhoz egy saját chatbotot, a neve ProgTábor Bot. A chatbotot a megoldások beadására, segítségek automatikus kiosztására, és a csapatvetélkedő alatt kért segítségek könyvelésére használjuk jelenleg. Sokat könnyít a szervezők dolgán, és a gyerekek szeretik használni. A következő parancsokat ismeri:

- !done <feladat sorszám> <megoldási link>: megoldás beadása egy adott feladathoz egy link formájában, ami lehet a kiértékelő rendszerben a beadásra mutató hivatkozás (submission link), egy online programozási felületen (mint az online IDE) a programfájl linkje, vagy egy olyan oldalra feltöltött szöveg linkje, mint a Pastebin, ahol a kódot meg lehet osztani. A megoldási linkeket egy Google táblázatba gyűjti nekünk a bot, ahol táborvezetőként áttekinthető formában látjuk, hogy az egyes csoportok hogyan haladnak a feladatokkal, melyik feladatot beszélhetjük meg, és gyorsan át tudjuk nézni a megoldásaikat, kiválasztani a legszebbeket.
- !hint <feladat sorszám>: egy előre megírt segítség, alapötlet kérése a feladathoz – ezt elsősorban a csapatverseny alatt használják a diákok, egyébként a normál csoportmunka alatt személyre szabottabb segítségeket adunk.
- !test <feladat sorszám>: az adott feladathoz olyan teszteset kérése, amire hibás a programuk – a Codeforces csapatverseny alatt mi látjuk a teszteseteket, de ők nem, ezzel a paranccsal lehet kérni tőlünk.
- !debug <feladat sorszám>: debuggolási segítség kérése az adott feladathoz – a bot értesíti a mentorokat és közli, hogy hova kell menniük, miközben feljegyzi a segítségkérés tényét (ez a csapatverseny alatt érdekes).
- !vicc: az off-topic csatornában küld a bot egy random viccet.

5.2.3. Codeforces, CSES, Mester és más feladatbankok

A táborokban a 4. fejezetben leírt tantervet és annak feladatait használjuk, így elengedhetetlen eszközeink a különböző online feladatbankok és hozzájuk tartozó online kiértékelő rendszerek. A 4.3.1. pontban felsorolt mind a 11 rendszert használjuk, itt kiemelném közülük a három legfontosabbat (a rendszerek webcímei a 10. pontban vannak felsorolva).

A Codeforces egy professzionális programozási verseny platform, amiben utólag az összes korábbi versenyfeladat elérhető, és a gyakorló üzemmódban nagyon barátságosan használható, hiszen mutatja a kis teszteseteket és más felhasználók beadásait. Számunkra nagyon fontos a

Mashup funkció, amit a csapatversenyekhez és a házi feladatokhoz is használunk. A Mashup funkcióban pillanatok alatt össze lehet állítani egy privát versenyt a korábbi Codeforces feladatokból (vagy akár a Polygon rendszerben saját feladatot is készíthetünk hozzá). A Mashup versenyre csapatokban is tudnak regisztrálni a felhasználók. A tananyagunkban külön jelentőséggel bírnak a feladatszálainkba illeszkedő szép Codeforces feladatok, amelyeket a csapatversenyekre tartogatunk.

A CSES weboldal egy kifejezetten oktatási célú feladatbankot és kiértékelő rendszert foglal magában, és azért szeretjük nagyon használni, mert sok klasszikus feladat és olyan feladat van benne, amiben „csak” egy tankönyvi alap algoritmust vagy adatszerkezetet kell megvalósítani. Ilyen feladatokat a versenyeken nem szokás kitűzni, ezért a versenyplatformokon csak nagyon ritkán elérhetőek, viszont nekünk ezek az alapfeladatok nagyon fontosak a tantervünkben, és a CSES remek lehetőséget biztosít arra, hogy tesztelni tudjuk az algoritmusok helyességét. A beadás után minden tesztesetet mutat, ami szintén ritkaságszámba megy az online értékelőrendszerek esetén, ezzel jelentően megkönnyíti a hibakeresést.

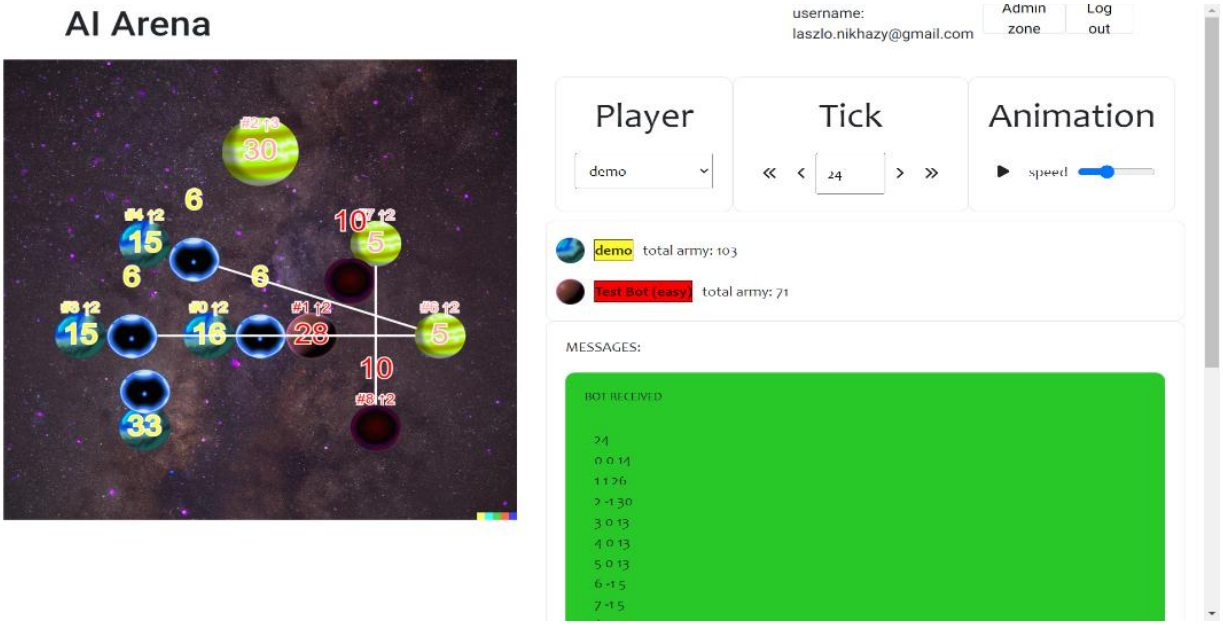
A Mester feladatbank nagy előnye, hogy magyarul vannak a feladatok szövegei, és rengeteg szép korábbi magyar versenyfeladat elérhető benne, automatikus kiértékeléssel együtt. Tanári fiókkal a kisméretű teszteseteket is le tudjuk tölteni a rendszerből. Nagy hátrány viszont nekünk, hogy a feladatok témakörök szerint kategorizálva vannak, és így a megoldási módszer szerepel a feladat elérési útvonalában és a szöveg fejlécében is, ami pont keresztbe tesz a felfedeztető tanítási módszernek. Ezt a problémát úgy szoktuk kiküszöbölni, hogy a feladatszöveget kivágjuk és fejléc nélkül megosztjuk a gyerekekkel, majd, amikor megoldották elméletben a feladatot, eláruljuk nekik, hogy hol találják meg a rendszerben a feladatot, hogy tudják beadni a programjukat. Szintén hátrány, hogy a feladatokra nem lehet linket küldeni, valamint mások beadásait is csak tanári fiókkal, külön engedéllyel lehet látni.

5.2.4. AI Arena

A bot vetélkedőhöz egy saját fejlesztésű rendszert használunk, amelynek a neve AI Arena (AI Arena, 2024). Az AI Arena egy webalkalmazás, amelynek segítségével a diákok által írt egyszerű stratégiai játékokat játszó programok egymás ellen versenyeznek. A bot vetélkedő során egy általunk kiválasztott, lehetőleg egyszerű játékban versenyeznek a csapatok. A játékok általában körökre osztottak. Például, a Planet War játékban néhány bolygó elfoglalásáért versenyez két játékos. Kezdetben egy-egy bolygón vannak saját egységeik, a többi bolygó semleges. A játékosok által birtokolt bolygók minden körben új egységeket

hoznak létre, a számuk a bolygó termelésével egyenlő, amely egy előre meghatározott konstans minden bolygóra. Új bolygó elfoglalásához a játékosok más bolygókra küldhetnek egységeket, amelyek a két bolygó távolsága által meghatározott körön át utaznak, majd megérkezéskor harcolnak, és a bolygó annak az irányítása alá kerül, akinek a legtöbb egysége van ott.

Minden játékhoz tartozik a rendszerben egy beépített vizualizáció, amellyel lépésről lépésre és animálva is meg tudják nézni a felhasználók, hogy hogyan zajlott egy meccs, azokkal az információkkal együtt, hogy milyen parancsokat adtak a játékban résztvevő bot játékosok az egyes körökben. Ez nemcsak szórakoztató, hanem tanulságos is, hiszen a diákok így jobban megérthetik a stratégiák működését. A Planet War játék vizualizációja látható az alábbi képen. A bot programja a parancsokat a standard kimenetre írja ki, és a játékállapotot a standard bemenetről kapja meg, mivel a diákok alapvetően ahhoz vannak szokva, hogy a versenyfeladatoknál a standard input-outputon keresztül kommunikálnak az értékelőrendszerrel. A felhasználók tehát feltölthetik a weboldalon a programjaikat, amelyeket a szerver gépen lefordítunk és egy játékszerver lebonyolítja a programok között a meccset.



5.1. ábra: Az AI Arena rendszer felhasználói felülete, a Planet War játék vizualizációja

A vetélkedők során az AI Arena le tudja bonyolítani a bajnokságot a csapatok botjai között úgy, hogy mindegyik bot játszik egy meccset mindegyik másik ellen (egy az egyben). Lehetséges több különböző pályát is bevenni egy bajnokságba. A diákok a bajnokság előtt a rendszerben a saját programjaikat tudják futtatni egymás ellen, és egy-két beépített tesztbot ellen is, és a vizualizációban elemezhetik a botjuk működését. A vetélkedő közepén mindig csinálunk egy tesztbajnokságot, ami még nem számít az eredménybe, csak kipróbálhatják az

aktuális programjaikat a csapatok egymás ellen. A végén ki kell jelölniük a legjobb programbotjukat, amit beregisztrálnak az éles bajnokságra. A csapatvetélkedő általában egy 3 órás program, ami épphogy elég arra, hogy egy egyszerű játék esetén már valami értelmes stratégiát megvalósító botot írjanak a résztvevők. Szokott panasz lenni arra a diákok részéről, hogy kevés rá az idő, azonban egy hétvégi tábor programjába több nem fér bele. A jövőben azt tervezzük, hogy több táboron keresztül visszatérünk egy-egy játékra, és így otthon is tovább tudják fejleszteni a stratégiájukat a csapatok.

5.3. Felmérések

A ProgTáborokról folyamatosan kértem visszajelzéseket a diákoktól. A visszajelzéseket elsődlegesen online űrlapok (Google Forms) formájában gyűjtöttem. Az így összegyűlt adatok segítségével az alábbi kérdéseket vizsgálom:

- Alkalmas-e a Pósa-módszer adaptálásaként kialakított felfedeztető tanítási módszer algoritmikus programozás tanítására?
- Tud-e egyszerre élvezetet és fejlődést nyújtani a táborokban alkalmazott felfedeztető tanítási módszer?
- Mely tevékenységek közben érzik a legtöbb fejlődést a diákok, és melyeket élvezik a legjobban?
- Milyen tényezők segítenek a diákok motivációjának fenntartásában?

Kétféle kérdőívet használtam a táborokban, egy nagyon rövid visszajelzés kitöltését kértem az egyes foglalkozásokról külön-külön még a táborban, és egy hosszabb űrlap kitöltését a tábort követő egy hétben.

5.3.1. Rövid visszajelző kérdőív az egyes foglalkozásokról

A rövid kérdőívvel az volt a célom, hogy számszerűsítve tudjam mérni, hogy a diákok saját érzése szerint az egyes foglalkozásokat mennyire élvezték és mennyit fejlődtek rajtuk. Természetesen a fejlődés mértékét a diákoknak nagyon nehéz megítélniük, de azt képesek érezni, hogy az egyes foglalkozásokon egymáshoz képesti összehasonlításban mennyi új dolgot sajátítottak el. Az alábbi kérdéseket tartalmazta a rövid kérdőív:

1) Mennyire élvezted a foglalkozást? (1=Unatkoztam – 5=Nagyon élveztem)
2) Mennyit tanultál/fejlődtél a foglalkozáson? (1=Semmit – 5=Sokat)
3) Bármilyen rövid szöveges visszajelzés

5.3.2. Bővebb visszajelző kérdőív egy teljes táborról

A táborok utáni bővebb visszajelző kérdőívben olyan kérdéseket tettem fel, amelyek segítségével megerősítést kaphattam a rövid visszajelzések eredményeivel kapcsolatban, és részletes képet kaphattam arról, hogy mennyire sikerül elérni a pedagógiai céljaimat az alkalmazott módszerekkel. A kérdőívben egyes kérdések a konkrét tananyagra és feladatokra vonatkoztak, annak érdekében, hogy a szakmai anyagot is tudjam fejleszteni, valamint kíváncsi voltam arra is, hogy általában mit szeretnek és nem szeretnek a diákok a táborban, hogy lássam, mivel lehet hosszú távon fenntartani a motivációjukat és érdeklődésüket. Továbbá konkrét változtatási javaslatokat is kértem tőlük. Az alábbi kérdések szerepeltek a kérdőíven:

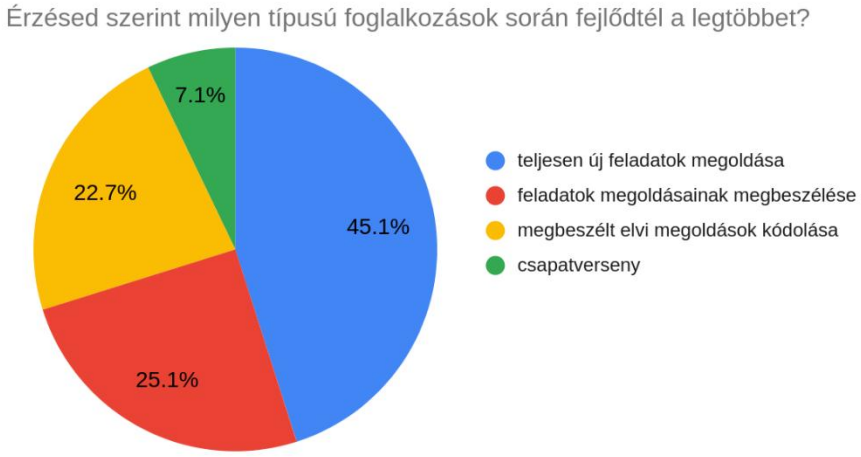
1) Érzésed szerint milyen típusú foglalkozások során fejlődtél a legtöbbet?
2) A tábornak melyik része tetszett a legjobban neked?
3) Melyik feladat tetszett a legjobban az egész táborban?
4) Mikor volt olyan helyzet, amikor egy feladat megoldása során saját magatok által kitalált módszert használtatok?
5) Milyen új dolgokat tanultál a tábor alatt?
6) Hogy működött a csoportmunka nálatok?
7) Mennyire voltak érdekesek a feladatok? (1-5)
8) Mennyire volt sok a feladat (a csapatversenyeket leszámítva)? (1-5)
9) Mennyire volt gyors a tempó? (1-5)
10) Mennyire voltak érthetőek a megoldások, magyarázatok? (1-5)
11) Milyen pozitívumokat emelnél ki?
12) Milyen negatívumokat tapasztaltál?
13) Milyen változtatási javaslataid vannak?

5.4. Eredmények értékelése

Az 5.1. pontban bemutatott négy tábori csoportnak összesen 27 tábort tartottam az elmúlt négy évben. A táborokról szóló visszajelző kérdőívek kitöltése önkéntes alapon működött. A bővebb kérdőívre összesen 296 kitöltés érkezett, míg az egyes foglalkozásokra vonatkozó rövid kérdőívek válaszaiból 1901 kitöltés elemzésével vonhattam le következtetéseket.

5.4.1. A felfedeztető tanítás sikeressége az algoritmusok tanításában

A kutatás központi kérdése, hogy működik-e a Pósa-módszer alkalmazása algoritmikus programozás tanítására. A bővebb visszajelző kérdőív (1), (4) és (6) kérdésére kapott válaszok a legerősebb indikátorok erre. Rendkívül érdekes ebből a szempontból az, hogy a diákok mely típusú foglalkozások alatt érezték a legtöbb fejlődést. Az alábbi grafikon szemlélteti a kérdésre kapott válaszok eloszlását.



5.2. ábra: A diákok melyik típusú foglalkozások alatt érezték a legtöbb fejlődést

Az összes kitöltés mintegy 45%-ában nyilatkoztak úgy a diákok, hogy a teljesen új feladatok megoldásával fejlődtek a legtöbbet, tehát a tanulás elsődleges színterének a csoportmunkás foglalkozásokat, azon belül is a feladatokon gondolkodást tekintették, ami a tanulási folyamatban a vizsgálat és megoldás lépése. Ezzel szemben például a közös megbeszéléseket nagyjából 25%-ban érezték a leghasznosabbnak a fejlődés szempontjából. Ez a statisztika még a várakozásaimat is felülmúlja, és erőteljesen mutatja, hogy az algoritmikus programozás tanításában is működik a felfedeztetés, hiszen a diákok képesek a saját tudásukat építeni.

A csoportmunka minősége is mérvadó mutató a felfedeztető tanítási módszer sikerességét tekintve, mivel ott kell működnie a tanítani kívánt ismeretek felfedezésének. A „Hogy működött a csoportmunka nálatok?” kérdésre adott válaszok elemzésével részletes képet kaphatunk erről. Bőven vannak olyan válaszok, amelyek a céljainknak megfelelő csoportmunkát írnak le, ezeket pozitív válaszként kategorizáltam. Például:

- *„Nálunk kiválóan működött a csapatmunka. Hagytuk, hogy mindenki saját magától jöjjön rá a megoldásokra és ha kellett segítettünk egymásnak, de nem egymás helyett oldottuk meg. Ami a lekódolást illeti ott is jó volt a feladatmegosztás és mindenki mindig csinált valamit.”*

- *„Rövid gondolkodás után megbeszéltük az ötleteinket, elvi hibákat kerestünk benne. Mikor konszenzus volt a helyességről, valaki elkezdte lekódolni. Debuggingban is segítettünk egymásnak.”*
- *„Együtt gondolkodtunk és kódoltunk, de a fontosabb alapfeladatokat (pl.: Dijkstra) külön kódoltuk.”*
- *„szerintem jól, a csapattársak nagyon kedvesek voltak, jól tudtunk együtt működni”*
- *„Remekül, csoporttársaimtól tanultam új dolgokat és a közös munka szerintem hatékony volt”*
- *„Jól működött, ha valamit nem tudtunk, együtt rájöttünk”*
- *„főként együtt gondolkodtunk, kódolni vegyesen külön és közösen”*

Nem lehet mindig tökéletes a csoportmunka sem, érkeztek negatív válaszok is, mint:

- *„Többnyire inkább külön dolgoztunk a feladatokon”*
- *„Szerintem nem igazán, mert a csoporttársaimmal eltérő szinten voltunk”*
- *„Én a könnyebbeken dolgoztam, szobatársaim a nehezeken”*
- *„külön gondolkodtunk, amikor valakinek nagyon nem ment, akkor lekódoltam én helyette”*
- *„Ha valaki meg tudott oldani egy feladatot olyankor azt leprogramozta és utána elmagyarázta a többieknek, hogyha működött”*
- *„Kicsit úgy éreztem, hogy főleg én oldottam meg a feladatokat, és ez kicsit zavart.”*

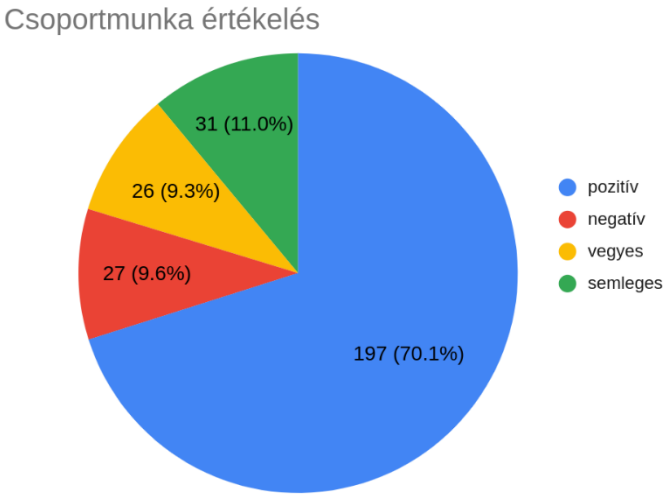
Néhány visszajelzés vegyesen tartalmaz pozitív és negatív elemeket is, például:

- *„Voltak olyan feladatok, amiken közösen tudtunk gondolkodni, de voltak olyanok is amelyet egyikünk ismerte, másikunk pedig nem, ezért néha kicsit nehézkes volt.”*
- *„Szerintem relatíve jól, bár a Bálint végezte a munka oroszlánrészét”*
- *„Nálunk Levivel jól gondolkodtunk együtt de kódolni 1szer sem kódoltam közösen senkivel ezt kicsit hiányoltam”*
- *„Alapvetően jól, bár voltak olyan alkalmak, sok feladat megvolt elméletben, és a kódolással haladás a csoportban elcsúszott”*

És bizonyos visszajelzéseket semlegesként kategorizáltam, amelyek tárgyilagosan nyilatkoznak a csoportmunkáról, és valamennyire sikerült csoportban dolgozniuk, de nem egészen úgy, ahogy ideális lett volna, például:

- *„Néha közösen ötleteltünk, néha külön gondolkodtunk”*
- *„Fentről haladtunk lefelé a feladatokkal, párhuzamosan többen dolgozva, legalább 2-en átbeszélve egy feladatot.”*
- *„Gondolkodtunk és ha valakinek volt valamilyen ötlete akkor azt elmondta”*
- *„általában én programoztam a dp-ket de általában együtt beszéltük meg a megoldást”*
- *„Először önálló gondolkodás, igény szerint módszer helyességének megbeszélése. Ezután általában mindenki implementálta a megbeszélt algoritmust.”*

Az alábbi grafikon mutatja az összesített eloszlását a pozitív, negatív, semleges és vegyes válaszoknak.



5.3. ábra: A csoportmunka értékelése

A pozitív válaszok aránya 70%, míg a negatív válaszok aránya 10% alatt van. Az is látható a válaszokat vizsgálva, hogy a negatív válaszok nagy része abból adódik, hogy túl nagy tudásbeli különbségek vannak a csoport tagjai között. Így egyértelműen kijelenthetjük, hogy megfelelően kialakított csoportmunkával sikeresen lehet felfedeztető módon algoritmusokat tanítani a tehetséges diákoknak.

A bővebb kérdőív (4) kérdését (*Mikor volt olyan helyzet, amikor egy feladat megoldása során saját magatok által kitalált módszert használtatok?*) azért tettem fel, hogy képet kapjak arról, hogy történik-e új ismeretek felfedezése a csoportmunka alatt, és fel tudják-e idézni ezeket a helyzeteket a diákok. Úgy gondoltam, hogy ha meg tudnak említeni legalább egy konkrét helyzetet, akkor az azt jelenti, hogy működött a felfedezés, akár még több helyzetben is, tehát alapvetően az ilyen típusú válaszok számomra nagyon pozitívak, de természetesen, ha valaki általánosságban beszél arról, hogy sokszor volt ilyen, az is jót jelent. A válaszokat kategorizáltam a típusuk szerint.

Számos olyan válasz érkezett, amikor egy konkrét feladatot említettek, például:

- *„A hátizsák problémás feladatok közül az elsőnél, amikor még nem beszéltük meg a módszert.”*
- *„A Dijkstrát elvben mi találtuk ki, és a kódoláshoz kaptunk segítséget.”*
- *„Road Reparationben kitaláltuk a Prim algoritmust”*
- *„A lajháros feladatnál”*
- *„Az színes épület feladatnál, amikor rájöttünk a nem rekurzív képletre”*
- *„Gráfok, körmentes gráfok - 15.Utak száma - feladatnál mesteren”*

Emellett voltak bőven még jobb válaszok, amelyek több konkrét feladatot említettek, ezek közül néhány példa:

- *„Több esetben is volt, road reparation, largest rectangle, investigation, consecutive letters”*
- *„5. feladatban a BFS és Dijkstra közötti algoritmus, 9. feladatban a nehezebbek, 10. feladat”*
- *„Például a lajhár és a worms”*
- *„A feladatok többségénél. Például a forgatásos feladatban, ezenkívül a csapatverseny feladatainál és a kígyó programozásánál.”*
- *„A csapatversenynél több feladatnál, illetve a munkasorrend, csatornák (bár nem lett lekódolva végül), nearest smaller number”*

Sokszor írták a diákok konkrétumok említése nélkül azt, hogy több ilyen helyzet is volt, például:

- *„A szombati csapatmunkára kapott feladatokra legtöbbször mi jöttünk rá a megoldásra.”*
- *„Amikor új anyag volt és sikerült kitalálni a működő algoritmust”*
- *„többször is volt ilyen, de általában szinte ilyen volt, vagyis a segítők rávezettek minket a módszerre”*
- *„Sokszor”*
- *„a legelső csoportmunkánál és a csapatversenynél”*

Nagyon örültem annak, hogy egyes diákok azt jelezték vissza, hogy szinte mindig ez történt, néhány ilyen válasz:

- *Szinte mindig, mivel nem igazán ismertük a csoportommal ezeket az alapvető elveket, módszereket.*
- *„A legtöbb feladatnak saját magunk jöttünk rá a megoldására, nem mindig sikerült a feladatok megbeszéléséig leprogramozni őket, de utána az összeset sikerült önerőből és a saját ötlet alapján lekódolni.”*
- *„kb. mindig, néha kaptunk kis lökést”*
- *„A feladatok nagyrésze ilyen volt.”*
- *„Algoritmusokat egy két nehezebb feladat kivételével mindig mi találtuk ki. Implementációban nem láttam hogy bárki valami különlegeset használt volna.”*

Sajnos a válaszok nem elhanyagolható hányada állítja, hogy nem volt ilyen helyzet, ami annak tudható be, hogy néhány diák számára a táborok egy részében nincs új algoritmus. Bár azt megjegyezhetjük, hogy még ebben az esetben is az egyes feladatoknál az algoritmusok testre szabásakor szinte mindenkinek előjön egy új ötlet. Ebbe a kategóriába soroltam például az alábbi válaszokat:

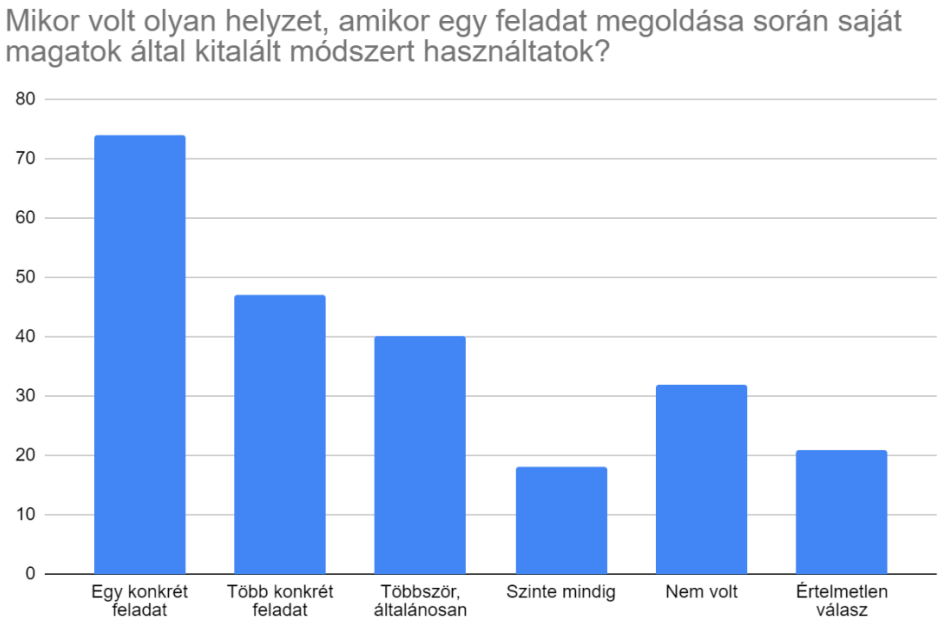
- *„Nem nagyon volt, mert ezeket az algoritmusokat már régebbről ismertem.”*
- *„Nem volt ilyen.”*
- *„inkább ismert algoritmusok használata volt, de ezek "felbontása", és részfeladataik megoldása segített a működésük megértésükben”*

- *„A feladattípusokat ismertük régebbről, emiatt ritkán fordult elő, hogy új módszert kellett kitalálnunk.”*
- *„Talán az előző másodikjához (SPOJ/INVESORT) kellett a legtöbb ötlet, de a nagy része inkább eddig ismert algoritmusok alkalmazása volt (ez nem negatív, a feladatok kihívások, és szórakoztatóak voltak, csak nem feltétlen éreztem, hogy kitaláltam egy új módszert, vagy hasonló)”*

Létre kellett hozni egy olyan kategóriát is, amibe a vizsgált kérdés szempontjából nem értelmezhető válaszok kerültek, mert néhányan nem megfelelően válaszoltak, vagy egyszerűen félreértették a kérdést, és azt gondolták, hogy olyan megoldásokról érdeklődünk, amelyek eltérnek a megbeszélttől. Néhány ilyen válasz:

- *„Szerintem mindent nagyjából úgy oldottunk meg ahogyan megbeszéltük”*
- *„Többször is más irányba indultunk el a -később- megbeszélthez képest, de ha jól emlékszem a kódolásnál már az elhangzottak alapján kódoltunk”*
- *„nem hinném, hogy a megoldásaink a feladatokra eltérnének a megbeszéltektől”*
- *„Nem tudom”*
- *„Igen”*

Az alábbi, 5.4. ábrán látható az egyes kategóriák között a válaszok eloszlása. A felfedeztetés működése szempontjából pozitív válaszból (első négy kategória) összesen 179 darab van. Ezzel szemben negatív válaszból 32 darab, melyek nagy része egyébként abból adódik, hogy egy csoporton belül jelentősen különböző tudásszintű diákok is lehetnek, amely problémáról az 5.1.2. pontban írtam bővebben. Összességében megállapíthatjuk, hogy a felfedeztető tanítás megvalósul a táborokban.



5.4. ábra: A felfedezésre vonatkozó válaszok eloszlása a kategóriák között

5.4.2. Élvezet és fejlődés

A felmérésekkel vizsgáltam azt is, hogy az egyes foglalkozások alatt mennyi fejlődést tapasztalnak a diákok, és mennyire élvezik az adott foglalkozást. Az 5.3.1. pontban részletezett rövid visszajelzésekben 1-től 5-ig terjedő skálán értékelték minden foglalkozást ebből a két szempontból, összesen 1901 válasz érkezett. Az összes választ tekintve a két változó együttes eloszlását mutatja az alábbi táblázat (alulról az i-edik sor balról j-edik oszlopában azon válaszok száma van, amiben az élvezet i, a fejlődés j pontot kapott).

élvezet						
5	4	16	110	257	580	
4	1	35	132	289	212	
3	0	31	88	72	19	
2	6	22	10	6	2	
1	5	3	0	1	0	
	1	2	3	4	5	fejlődés

5.6. táblázat: Élvezet és fejlődés együttes eloszlása a rövid visszajelzéseken

A számok arról tanúskodnak, hogy alapvetően nagyon pozitívan értékelik a foglalkozásokat a diákok mind élvezet, mind fejlődés szempontjából, hiszen a visszajelzések kb. 70%-ában mindkét érték 4 vagy 5. Az is látszik, hogy a két mennyiség között van korreláció, hiszen a főátló körül sűrűsödnek az adatok, továbbá több adatpont van a főátló felett, mint alatt, tehát az élvezet gyakrabban kapott magasabb értékelést a fejlődésnél, mint fordítva. Számszerűsítve, az élvezeti értékek átlaga 4,34, a fejlődési értékeké 4,11, míg a két mennyiség korrelációja 0,48. A kölcsönhatás a két mennyiség között meglátásom szerint kétirányú, tehát nincs egyértelmű ok-okozati kapcsolat, hiszen lehetséges, hogy a diák egy foglalkozást azért élvez, mert érezhetően fejlődik, új dolgokat tanul, ugyanakkor az is igaz, hogy könnyebben tud új dolgokat tanulni egy olyan foglalkozáson, amit élvez, mint egy olyanon, amin unatkozik. Viszont az egyértelmű, hogy a tanulás élvezete hozzájárul a hosszútávú motiváció fenntartásához, ezért kulcsfontosságú számomra, hogy élvezetes foglalkozásokkal tudjam megvalósítani a diákok tudásának fejlesztését, ami az eredményeket tekintve kétségkívül megvalósult a táborokban.

Érdekes megvizsgálni a foglalkozások típusa szerint csoportosítva a válaszok eloszlását. Öt típusra bontottam szét a foglalkozásokat. Érkeztek értékelések tisztán csoportmunkás és megbeszéléses foglalkozásokról, de olyanokról is, ahol vegyesen volt csoportmunka és megbeszélés. Továbbá különvettem a csapatvetélkedőket, amelyeknek a korábban említett két típusa van, a Codeforces csapatverseny és a bot vetélkedő. Az alábbi táblázatokban látható a válaszok eloszlása, a bal alsó sarokban van feltüntetve a foglalkozás típusa. A számolt statisztikákat (átlag, szórás, korreláció) a lentebb található 5.12. számú táblázatban foglaltam össze.

élvezet						
5	1	3	20	64	164	
4	0	4	35	73	50	
3	0	6	19	13	2	
2	0	3	1	0	0	
1	0	0	0	0	0	
csoport-munka	1	2	3	4	5	fejlődés

5.7. táblázat: Csoportmunkás foglalkozásokon élvezet és fejlődés együttes eloszlása

élvezet						
5	0	0	4	24	110	
4	0	3	27	95	87	
3	0	12	45	35	10	
2	4	12	5	4	2	
1	3	3	0	1	0	
megbe-szélés	1	2	3	4	5	fejlődés

5.8. táblázat: Megbeszéléseken élvezet és fejlődés együttes eloszlása

élvezet						
5	1	4	26	67	126	
4	0	21	41	81	57	
3	0	6	9	10	5	
2	1	5	1	1	0	
1	0	0	0	0	0	
vegyes	1	2	3	4	5	fejlődés

5.9. táblázat: Vegyes foglalkozásokon az élvezet és fejlődés együttes eloszlása

A legérdekesebb összehasonlítás számunkra a csoportmunka és megbeszélések szembeállítása. Mindkét esetben az eloszlás a főátló jobb felső szakasza körül összpontosul, de a csoportmunkák esetében sokkal erősebb az adatok sűrűsödése a jobb felső sarokban, tehát

alapvetően jobb a megítélése ezeknek a foglalkozásoknak mind élvezet, mind fejlődés szempontjából, amit az átlagok is mutatnak, a csoportmunkában az élvezet és fejlődés átlaga 4,45 és 4,23, a megbeszéléseken 3,92 és 4,10. Az is jól látszik az eloszlásokon, hogy a csoportmunkák esetében a főátló felett van több adatpont, tehát az élvezet erősebb, míg a megbeszéléseken a főátló alatt, tehát a fejlődés erősebb, ez az átlagokból is egyértelmű. Viszont felettébb érdekes, hogy általánosságban a csoportmunkában magasabbak a fejlődési értékek, tehát úgy érzik a diákok, hogy azokon a foglalkozásokon több új ismeretet sajátítottak el, mint a megbeszéléseken. Ez jelzi, hogy a céljaim megvalósultak, hiszen a csoportmunka a tanítás felfedeztető szakasza, ezzel szemben a megbeszéléseken az algoritmusokat és módszereket részletesen bemutatjuk, elemezzük, ami a hagyományos tanítási módszereknek felel meg.

élvezet						
5	2	4	43	80	145	
4	0	2	24	34	17	
3	0	4	10	10	1	
2	0	2	3	1	0	
1	0	0	0	0	0	
csapat-verseny	1	2	3	4	5	fejlődés

5.10. táblázat: Codeforces csapatversenyeken élvezet és fejlődés együttes eloszlása

élvezet						
5	0	5	17	22	35	
4	1	5	5	6	1	
3	0	3	5	4	1	
2	1	0	0	0	0	
1	2	0	0	0	0	
vetél-kedő	1	2	3	4	5	fejlődés

5.11. táblázat: Bot vetélkedőkön élvezet és fejlődés együttes eloszlása

A vetélkedőkre és csapatversenyekre érkezett válaszokban sokkal erősebb az élvezet, mint a fejlődés, de ez pont a várakozásoknak megfelelő, mivel kevés új ismeretet foglalnak magukban a feladatok azokon a foglalkozásokon, szinte kizárólag a már ismert dolgok új helyzetekben való alkalmazásáról és egymással való kombinálásáról szólnak. Továbbá az is igaz, hogy jellemzően ezeket a foglalkozásokat élvezték a legjobban a diákok, ez leolvasható az átlagokból is, és a külön kérdés alapján is, amit alább mutatok be. Így a csapatvetélkedős foglalkozások is elérték a céljukat.

A vegyes foglalkozások értékelésében az az érdekesség figyelhető meg, hogy az élvezetre vonatkozó válaszok a csoportmunkás foglalkozások mintázatát mutatják, míg a fejlődésre adott válaszok a megbeszéléses foglalkozások eloszlásához vannak közel, összességében viszont elég hasonló a kép az összes válasz átlagához.

	Csoportmunka	Megbeszélés	Vegyes	Csapatverseny	Vetélkedő	Összes
Élvezet átlag	4.45	3.92	4.39	4.62	4.51	**4.34**
Élvezet szórás	0.69	0.92	0.70	0.68	0.76	**0.80**
Fejlődés átlag	4.23	4.10	4.07	4.14	3.75	**4.11**
Fejlődés szórás	0.86	0.98	0.96	0.89	1.11	**0.95**
Korreláció	0.48	0.67	0.37	0.37	0.49	**0.48**
Darabszám	458	486	462	382	113	**1901**

5.12. táblázat: Élvezet és fejlődés visszajelzések statisztikái a különböző típusú foglalkozások esetén

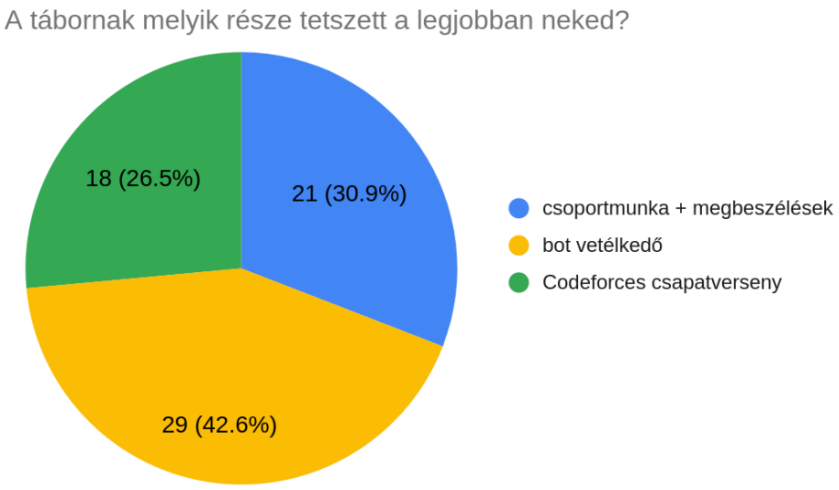
A rövid kérdőívekre adott válaszok megerősítésére a bővebb kérdőívben feltettem két kérdést:

- Érzésed szerint milyen típusú foglalkozások során fejlődtél a legtöbbet?
- A tábornak melyik része tetszett a legjobban neked?

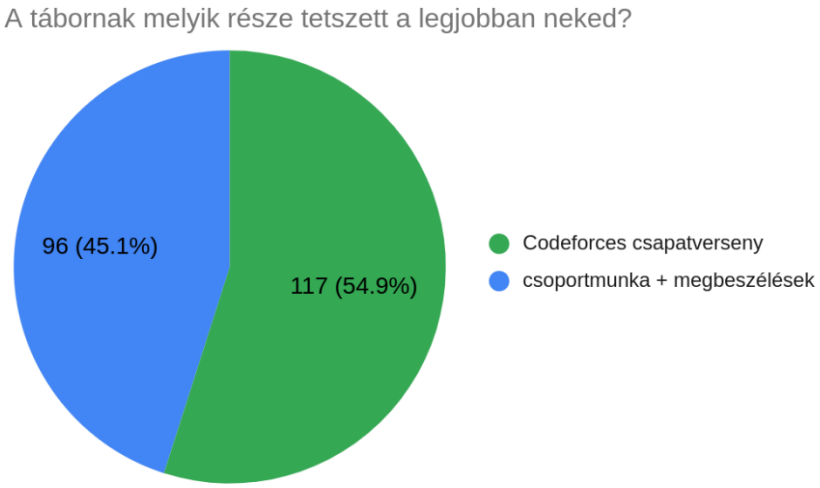
A második kérdés válaszlehetőségei a tábor egyes szakaszaira vonatkoztak, például:

- Péntek esti házi feladat megbeszélés és csoportmunka
- Szombat délelőtti csoportmunka és megbeszélés
- Szombat délutáni csapatvetélkedő
- Vasárnapi csoportmunka és megbeszélések

Így ezekben az értékelésekben a csoportmunka és megbeszélés nem választható ketté, továbbá a különböző csoportmunka és megbeszéléses szekciók szétválasztásának csak olyan szempontból volt értelme, hogy a konkrét tananyagrész tanítására vonatkozó visszajelzéseket gyűjtsek és ezek alapján tudjam továbbfejleszteni a szakmai részt. Itt viszont azt elemzem, hogy milyen típusú foglalkozásokat mennyire élveztek a diákok, és ennek érdekében ezeket célszerű együttesen kezelni. Továbbá, a csapatvetélkedő lehet Codeforces csapatverseny és bot vetélkedő is, sőt a legtöbb alkalommal amikor bot vetélkedőt szerveztünk, tartottunk Codeforces csapatversenyt is, mert sok diák nagyon hiányolta volna. Az alábbi grafikonokon tehát a táborban zajlott csapatvetélkedő(k) típusa szerint szétválasztva mutatom a válaszok eloszlását (máshogy nincs értelme, mert sokkal kevesebb táborban volt bot vetélkedő, mint Codeforces csapatverseny).



5.5. ábra: A legjobban tetszett foglalkozások aránya olyan táborokban, ahol mindkét fajta csapatvetélkedő volt



5.6. ábra: A legjobban tetszett foglalkozások aránya olyan táborokban, ahol csak Codeforces csapatverseny volt

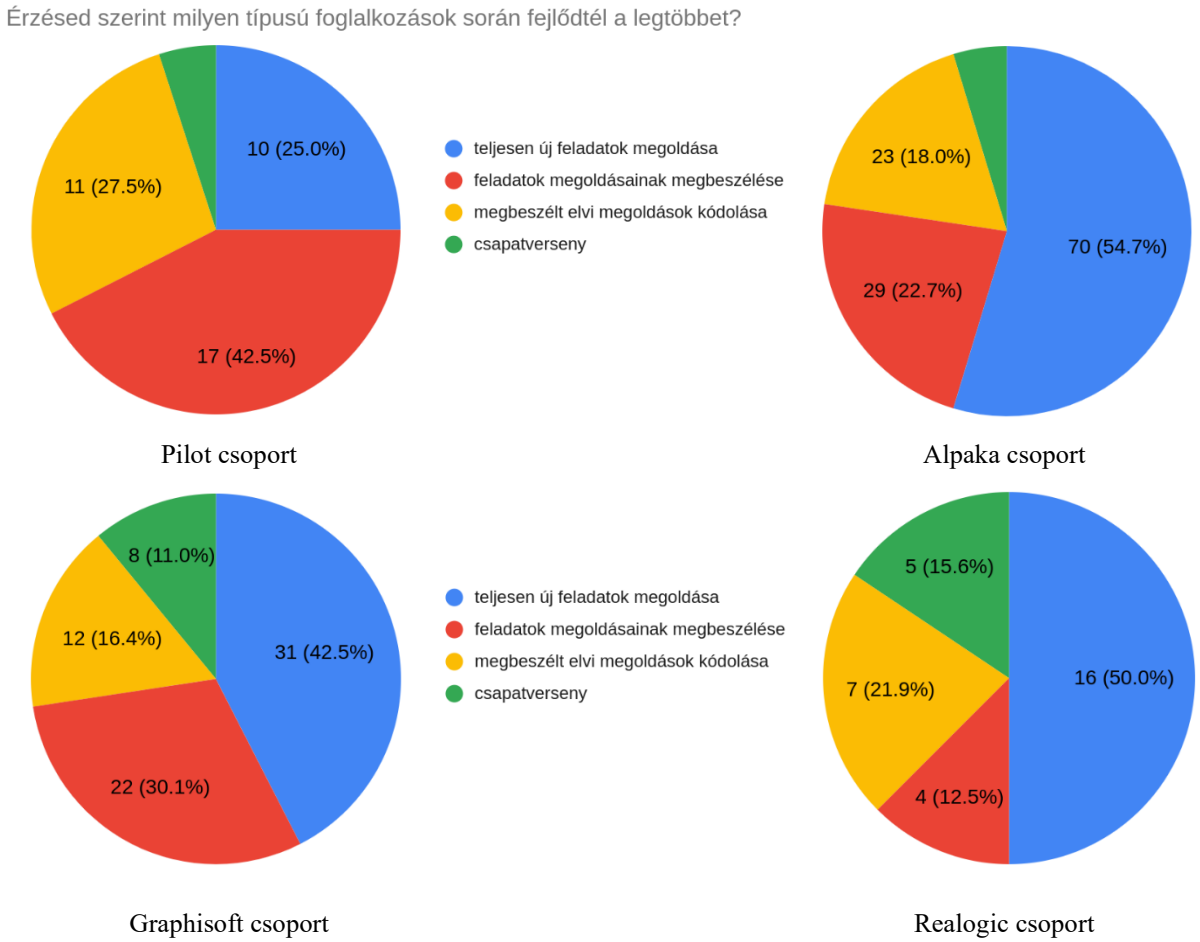
Az az érdekesség figyelhető meg a statisztikákban, hogy alapvetően a csapatvetélkedőket nagyon szeretik a diákok, hiszen amikor csak egy Codeforces csapatverseny van és mellette három csoportmunka/megbeszéléses foglalkozás, akkor is a csapatverseny kapja a szavazatok többségét (54,9%), és amikor bot vetélkedőt és csapatversenyt is szerveztünk, akkor a kettő együtt a szavazatok majdnem háromnegyedét kapta (73,5%), ezen belül is a bot vetélkedő volt a legnépszerűbb. Azonban abban az egy táborban, amikor csak bot vetélkedő volt, fordított lett a statisztika, 73,3%-ban a csoportmunka és megbeszéléses foglalkozások győztek. Ennek kettős indoka lehet, egyrészt a diákok feltűnően hiányolták a Codeforces csapatversenyt (az

egyéb válaszokból ez kiderül), másrészt egy bonyolultabb stratégiás játék volt a bot vetélkedőn, amiben kevesebbeknek volt sikerélménye. Mivel ezt csak egy táborban szerveztük így, messzemenő következtetéseket nem vonok le belőle.

5.7. ábra: A legjobban tetszett foglalkozások aránya abban a táborban, ahol csak bot vetélkedő volt

Összességében kijelenthetjük, hogy a csapatvetélkedős foglalkozások nagyban hozzájárulnak a táborok pozitív megítéléséhez és így sokat segítenek a diákok motivációjának fenntartásában. A bot vetélkedős foglalkozások pedig színesítik a tábor szakmai anyagát, és népszerűek a diákok körében, mivel jelentősen különböznek a hagyományos programozási versenyektől is.

A bővebb kérdőív első kérdésére („Érzésed szerint milyen típusú foglalkozások során fejlődtél a legtöbbet?”) az összesített statisztikát bemutattam az 5.4.1. pontban, amiből kiderült, hogy a csoportmunkás foglalkozásokat, azon belül is a teljesen új feladatok megoldását tekintik a leghasznosabbnak a diákok. A válaszok között a négy csoport összes táborát figyelembe vettem. Érdemes szétválasztani csoportonként a visszajelzéseket, hiszen egy csoport több táborában ugyanazok a diákok jeleztek vissza, így a csoportra jellemző preferenciákat láthatjuk külön-külön, ami részletesebb betekintést ad a mérési adatokba.



5.8. ábra: A diákok melyik típusú foglalkozások alatt érezték a legtöbb fejlődést az egyes csoportokban

A tendenciák hasonlóak az egyes csoportokban, három csoportban (Alpaka, Graphisoft, Realogic) a teljesen új feladatok megoldása alatt érezték a legtöbb fejlődést a résztvevők, de ebből a szempontból erős kivételt jelent a Pilot csoport, amelyiknél gyakorlatilag felcserélt helyzetben van a feladatok megoldásainak megbeszélésével. Ennek az eltérésnek az indoka az lehet, hogy a csoport idősebb életkorban indult, amikor nagyobb különbségek voltak már a diákok között, és kezdettől fogva összetettebb témák voltak, amelyekről a kevésbé kiemelkedő képességű diákok a megbeszélések során tudtak új ismereteket szerezni. Egy másik indok lehet az, hogy az esetükben a válaszok fele még a karanténidőszak alatti online táborokból érkezett, és személyesen jóval hatékonyabb tud lenni a csoportmunka.

Az Alpaka és Graphisoft csoportokban a fejlődés megítélésében a foglalkozások sorrendje azonos: 1. új feladatok megoldása, 2. megoldások megbeszélése, 3. implementáció, 4. csapatverseny. Az összesített statisztikában is ez a sorrend, és ez meg is felel annak, amit a felfedeztető módszertanunktól elvárunk. A Realogic csoportban érdekes módon a megbeszélés

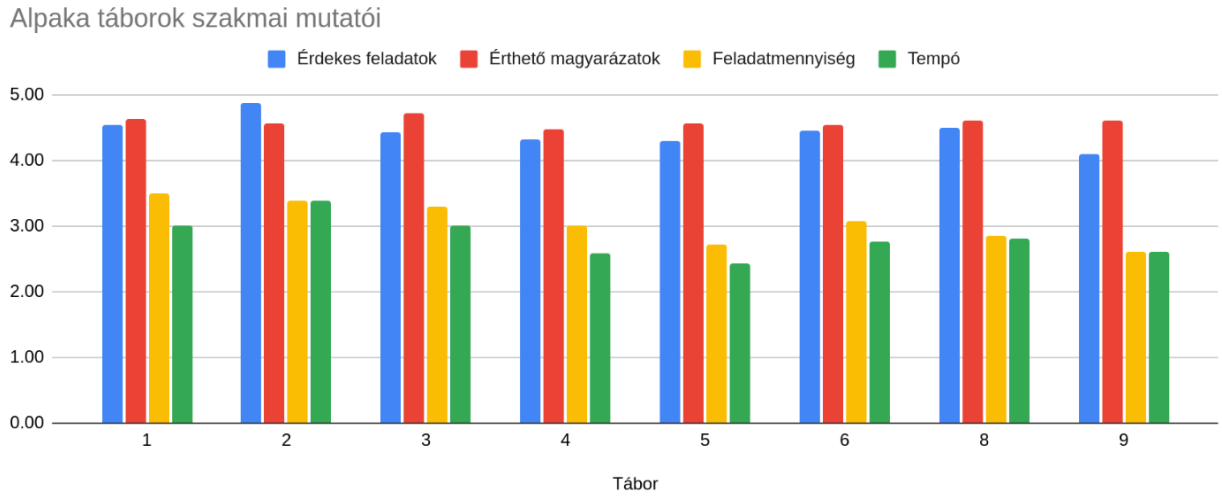
visszacsúszott a 4. helyre, viszont ott még jóval kevesebb visszajelzés alapján alakult ki ez a sorrend, arra számítok, hogy a táborok előrehaladtával ez meg fog változni, és a másik két táborhoz hasonló statisztika jön létre.

5.4.3. Tananyag és haladás értékelése

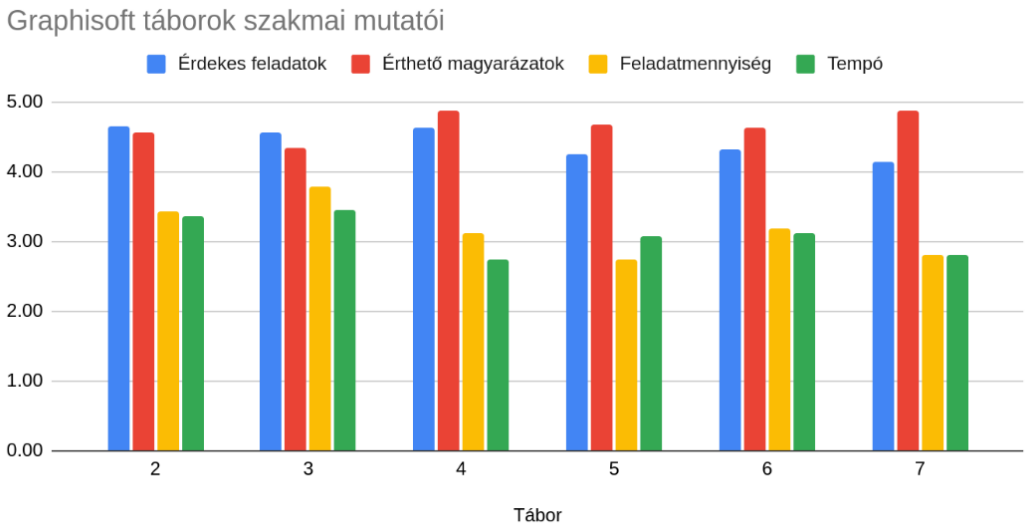
A bővebb visszajelző kérdőív 7-10. kérdésével számszerű visszajelzéseket kértem a tábor szakmai részéről:

- Mennyire voltak érdekesek a feladatok? (1-5)
- Mennyire volt sok a feladat (a csapatversenyeket leszámítva)? (1-5)
- Mennyire volt gyors a tempó? (1-5)
- Mennyire voltak érthetőek a megoldások, magyarázatok? (1-5)

Az alábbi diagramokon az egyes csoportokra külön, táboronként átlagolva ábrázolom az eredményeket. A feladatok érdekessége és a magyarázatok érthetőségénél az 5 a legjobb válasz, míg a tempó és a feladatmennyiség megítélésénél az ideális válasz a 3, hiszen akkor pont jó a haladás sebessége és a feladatok száma. Sajnos egyes táborokról nincsenek ilyen típusú visszajelzéseim, mert nem küldtem el a diákoknak a kérdőívet, vagy pedig nem voltak rajta ezek a kérdések (Alpaka / 7, Graphisoft / 1, Pilot / 2).



5.9. ábra: Az Alpaka csoport táborainak szakmai mutatói



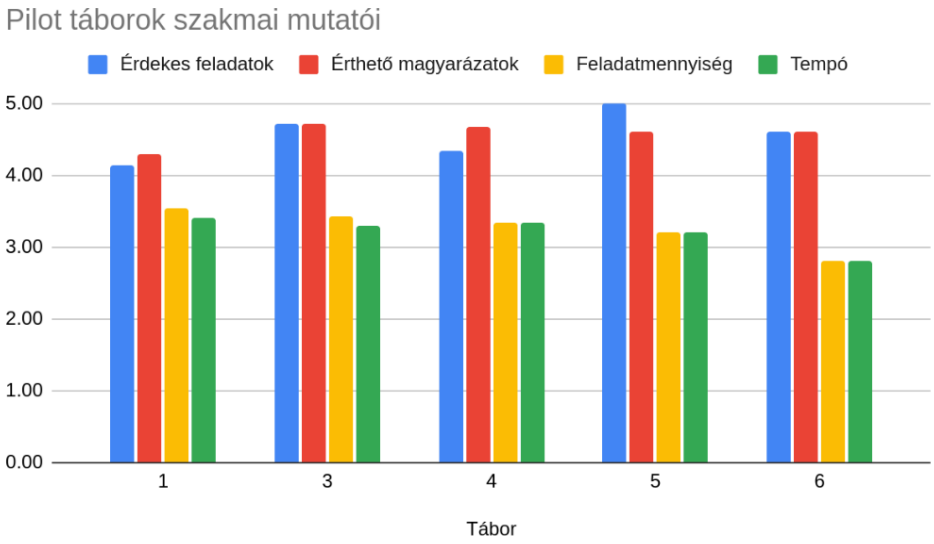
5.10. ábra: A Graphisoft csoport táborainak szakmai mutatói

A magyarázatok érthetősége meglehetősen stabilan 4,5 körüli átlagot mutat minden táborban, egyetlen kivétel a Realogic / 3, ahol igazából két kirívóan alacsony értékelés húzza le az átlagot. Tehát a megbeszélések ilyen szempontból felettébb jól sikerültek.

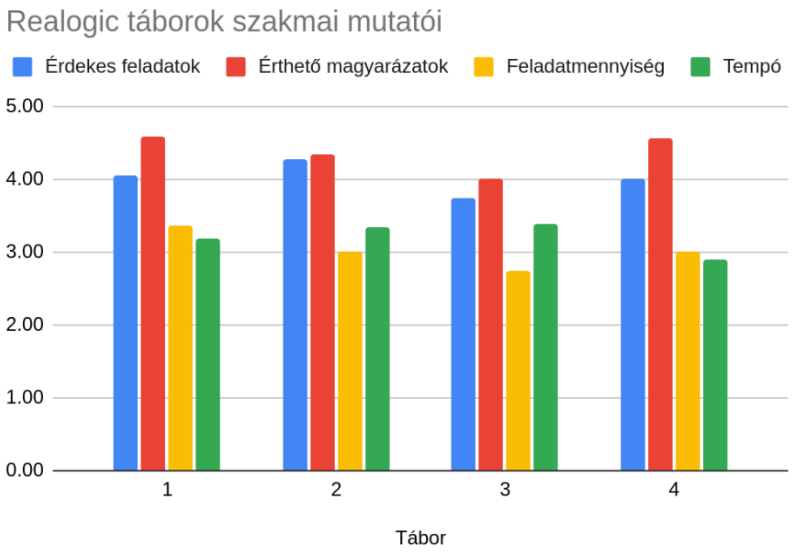
A feladatok érdekessége az Alpaka és Graphisoft csoportokban stabilan 4 felett, jellemzően 4,3 és 4,6 között alakul, viszont enyhe csökkenő tendenciát mutat a táborok előrehaladtával. Ennek indoka lehet az, hogy a haladóbb algoritmusok tanításakor már kevesebb érdekes alkalmazás fér bele a tábor idejébe, de ezen lehet alakítani és a jövőben megpróbálok valamennyit változtatni a feladatok összeállításában. A Pilot csoport esetében még nem volt ilyen tendencia.

A legújabb, Realogic csoport esetében viszont a feladatok érdekessége csak 4,0 körül mozog, a harmadik táborban annál kevesebb is volt. A magyarázat erre az lehet, hogy az online szakkörök térnyerésével a diákok egy része meglehetősen sok dolgot ismer a tábor anyagából. A jelenségről az 5.1.2. pontban már írtam, és jelenleg a táborok tekintetében ez egy fontos megoldandó probléma, ami miatt a közeljövőben a táborok anyagán vagy a csoportok összeállításán némileg változtatni fogunk.

Összességében azonban a 4,0 és az afeletti átlagos értékek egyértelműen azt jelzik, hogy a diákoknak tetszenek a feladatok és élményt nyújt nekik a rajtuk való gondolkodás.



5.11. ábra: A Pilot csoport táborainak szakmai mutatói

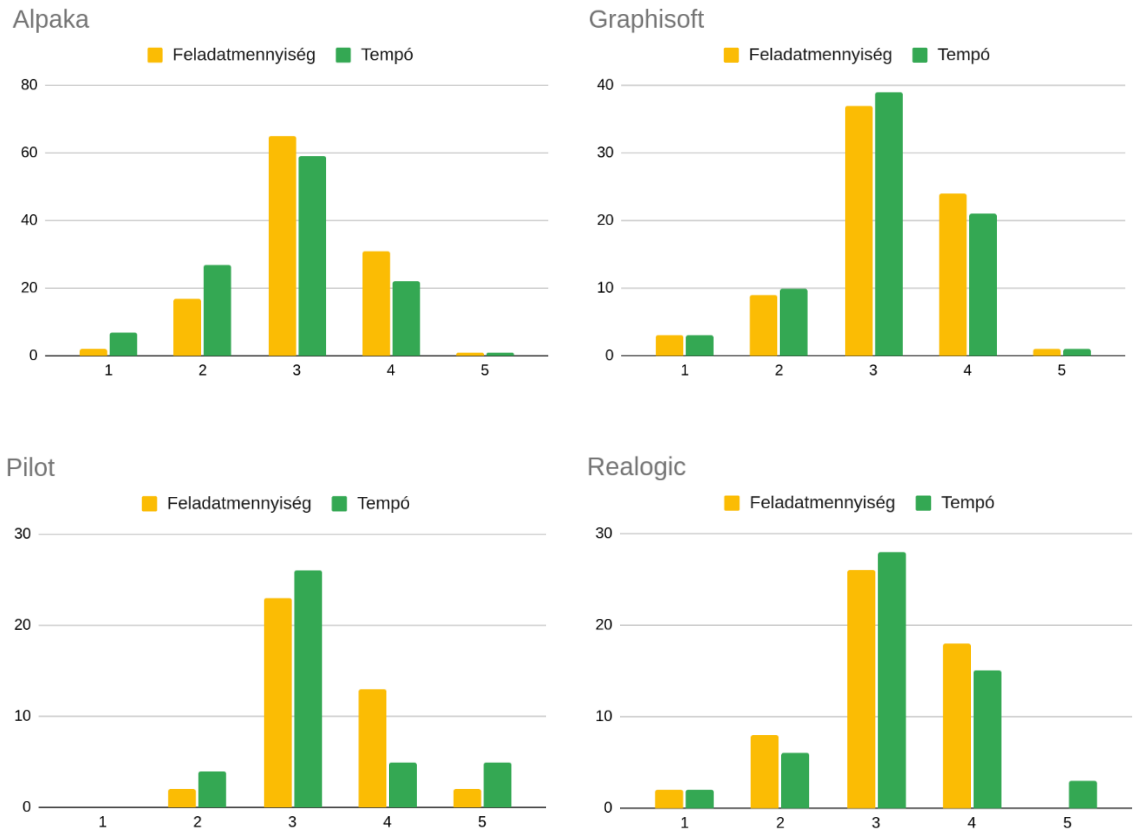


5.12. ábra: A Realogic csoport táborainak szakmai mutatói

A feladatok mennyisége és a haladás tempójának átlaga elég szépen a 3-as átlag körül mozog, ami azt jelenti, hogy a diákok átlagos szintjéhez képest jó ütemben sikerült haladni és megfelelő mennyiségű feladatot összeállítani. Az egyes csoportokon belül a táborok előrehaladtával enyhe csökkenő tendencia figyelhető meg mindegyik mennyiségben, ennek két külön indoka lehet. A tempót illetően valószínűleg az elején, amíg nem építhetünk a korábbi táborok anyagára, meglehetősen más hozott ismeretekkel és képességekkel jönnek a diákok, és egyeseknek hihetetlenül gyors a haladás, viszont néhány tábor után tudnak támaszkodni a korábban tanultakra és eleget fejlődnek ahhoz, hogy ne érezzék ezt. A feladatok száma pedig a nehezebb feladatok és munkaigényesebb implementációk miatt egyre kevesebb lesz a

táborokban, és ezáltal nagyon ritkán fordul elő, hogy valakiknek nem lenne ideje egy feladattal foglalkozni legalább valamennyit.

Az átlag nem mutat teljes képet egyik mérőszámban sem, és mindenképp fontos áttekintenünk, hogy esetleg szélsőségek átlagaként jön ki egy jó középérték, vagy pedig közel normális eloszlást követnek a válaszok. Az alábbi diagramok az egyes csoportok összes válaszának eloszlását mutatják, ami alapján lehet látni, hogy a válaszok a normális eloszlás haranggörbéjéhez hasonló alakú eloszlást mutatnak az 5-fokozatú skálán, tehát a tempó és feladatmennyiség szempontjából a diákok többségének jó a tábor.



5.13. ábra: A feladatmennyiségre és tempóra vonatkozó válaszok eloszlása csoportonként

5.4.4. Közösség, segítők és más motivációs tényezők

A bővebb kérdőív utolsó előtti két kérdése általánosságban a pozitívumokra és negatívumokra vonatkozott. A szabad szöveges válaszokban a gyakran megjelenő elemeket vizsgáltam.
A pozitívumok között sokszor említették a feladatokat, például:

- *„Nagyon jók voltak a feladatok és nagyon jól fel volt építve!”*
- *„A feladatok sokszínűsége, a megbeszélések, a jó környezet”*
- *„Segítők nagyon rendesek, király lekötő feladatok”*

- *„A feladatok egyre nehezedőek voltak, így mindenki találhatott egy számára izgalmas problémát. A megoldások átbeszélése segített a tempó megtartásában, a témakörök többnyire egymásra épültek. A csapatverseny további izgalmakat hozott a táborba, s egy plusz motivációként hatott."*

A mentorok (akik tapasztalt, de fiatal segítők) is rengeteg pozitív visszajelzésben szerepeltek, például:

- *„Kedves, segítőkész, bölcs mentorok, jó/érdekes feladatok"*
- *„Nekem az tetszett, hogy a segítők gyakran bejöttek a szobákba (vagy ha hívtuk őket) és a tudás szintjüknek megfelelően tudtak segíteni a gondolkodásban, ötletelésben és a programozásban."*
- *„A segítők mindig segítettek, ha egy feladat megoldásánál nem jöttünk rá, hogy mi az adott feladat megoldásának elmélete, vagy ha elírtunk valamit a programban, ami miatt nem futott le vagy nem megfelelően működött. Tetszett, hogy csapatokban dolgoztunk."*
- *„A mentor állandó jelenléte és tanácsai sokkal aktívabbá, hasznosabbá tette számomra a tábort"*
- *„szuperek a segítők, finom a kaja"*

A csoportmunka is gyakran ismétlődő elem:

- *„Jó volt a csapattal együtt gondolkodni, programozni és a megbeszéléseket is nagyon élveztem."*
- *„Az hogy közösen oldottuk a feladatokat és sokat segítettünk egymásnak"*
- *„[...] A csoportmunka hasznosságát is kiemelném, hiszen nemcsak hatékonyabban, hanem jobb kedvvel és motiváltabban oldjuk meg a feladatokat."*
- *„Habár nem szeretek csapatban dolgozni, a táborokban mindig élvezem"*
- *„Jó volt ez a csoportos dolog, hogy ki lehetett próbálni milyen csapatban, másokkal együtt programozni."*

Sokan kiemelik a csapatvetélkedőt vagy csapatversenyt:

- *„Szerintem a csapatverseny nagy jól meg volt szervezve és a helyszínen található felszerelés és a táblák sokat segítettek a gondolkodásban"*
- *„Jó volt most ez a Codeforces csapatverseny, a többi feladat is jól volt válogatva"*
- *„Jobb étkezés, érdekes új csapatvetélkedő, remek hangulat"*
- *„Planet War játék nagyon jó volt."*

Számos visszajelzés szólt a megbeszélésekről, magyarázatokról is:

- *„Jó volt, hogy mindig megbeszéltük a megoldásokat, ha valaki nem értette, nyugodtan szólhatott, és tetszettek az esti játékok."*
- *„Egy-egy feladathoz több megoldást is bemutattak, ezáltal jobb/több megoldási lehetőséget tudtunk meg"*
- *„A társaság, a feladatok érthető megoldása/magyarázata, hogy nem lett elmondva a megoldás csak rávezettek a mentorok"*
- *„Az általad adott magyarázatok teljesen érthetőek voltak, nem voltam úgy vele, hogy nem értem, mint például néhány másik résztvevő magyarázatánál, és a prezentációk is elég jók voltak"*
- *„Minden elsőre érthető volt"*

Többen említik az újonnan tanult témákat:

- *„Kifejezetten tetszettek az új algoritmusok, sokat tudtam fejlődni”*
- *„A flow számomra érdekes és új dolog, sportok nagyon jók”*
- *„Kreatívak voltak a témák és az új programok is.”*
- *„A geometriai feladatok nagyon hasznosak voltak, mert erről a témáról nem tudtam a tábor előtt olyan sokat. Szünetekben sokat beszélgettem a mentorokkal és a diákokkal feladatokról, tanácsokat is kaptam tőlük azzal kapcsolatban, hogy hogyan gyakoroljak/fejlődjek. Csoportmunka során hasznos segítségeket kaptunk, ha elakadtunk. Az új anyagok a megbeszéléseken teljesen érthetően lettek elmagyarázva.”*
- *„Tetszett, hogy egy tematika körül oldottunk meg feladatokat, csak kicsit más alaphelyzettel”*
- *„Sok hasznos dolgot tanultam a tábor alatt”*

Továbbá egy rendkívül gyakori elem a társaság, közösség:

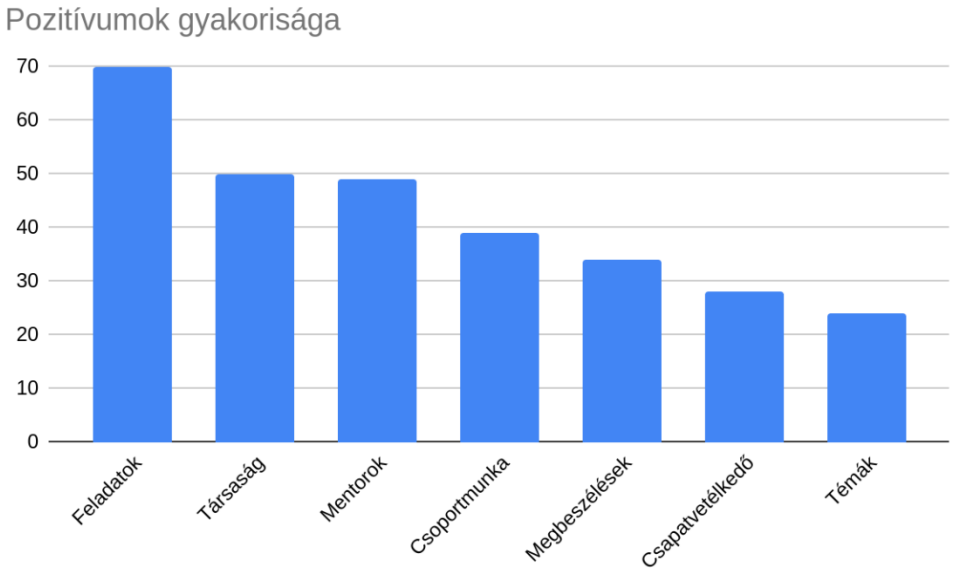
- *„Nagyon jó a társaság és hogy mindenki érdeklődik és hajt a témák iránt. Ez egy óriási motivációs tényező.”*
- *„Nagyon jó volt a társaság, nagyon jól lehetett társasozni”*
- *„Jó volt hozzám hasonlókkal lenni”*
- *„Új embereket volt lehetőségem megismerni.”*
- *„A hely, változatos feladatok, remek közösség és segítőkész emberek”*
- *„Nagyon sok új dolgot tanultam és nagyon jó volt a csoport, mindenki értelmes volt és normálisan beszélt, viselkedett az órákon.”*
- *„Érdekes dolgokat tanultam és nagyon jó a közösség”*
- *„Jó volt a hangulat, társaság, a feladatok érdekesek voltak, szervezők közvetlenek”*

Az alábbi grafikonon szemléltetem a fent felsorolt elemek gyakoriságát a 296 kitöltésen belül. Örvendetes, hogy a feladatokat annyira gyakran említik, hogy az vezeti a statisztikákat, viszont sokkal fontosabb extra információt jelent, hogy a következő két legjelentősebb pozitívum a társaság és a mentorok.

A fiatal mentorok hidat teremtenek a táborvezető és a gyerekek között. A gyerekeknek ők egyszerre játszótársak és nagy tudású instruktorok, ezáltal egyszerre alakulhat ki baráti és példaképi kapcsolat köztük. Maguk a mentorok is nagyon élvezik a tábort, számukra ez munka és szórakozás együtt, a lelkesedésük pedig roppantul motiválja a diákokat is. A táborok sikerességében kulcsfontosságú szerepük van.

A közösségépítést rendkívül fontos célnak tekintettem, és ez a válaszok alapján visszaigazolást nyert. Hihetetlenül erős motivációs tényező tud lenni, ha a diákok találkoznak hozzájuk hasonló beállítottságú társakkal, és a közös tanulás mellett együtt játszanak, beszélgetnek, kapcsolatokat teremtenek. A kortárs kapcsolatok tinédzser korban fontosabbak a felnőttekkel való kapcsolatoknál, így, ha vannak olyan kortársak, akik ugyanúgy érdeklődnek az

algoritmusok és a programozás iránt, akkor egymás elkötelezettségét drasztikusan tudják növelni. Ráadásul néhány diák olyan közegből jön, hogy a ProgTáborban találkozik először ilyen kortársakkal, és ez sorsdöntő tényező lehet. A közösségi szempont amellett szól, hogy azonos életkorú diákokból alakítsunk csoportokat, amelyek több éven keresztül együtt maradnak. Márpedig vitathatatlannak mutatkozik a közösség motivációs szerepe a tanulásra nézve.



5.14. ábra: A pozitívumok között a gyakori motívumok előfordulásainak száma

A negatívumokat is áttekintettem hasonló rendszerrel. A legtöbb válasz egyszerűen azt fejezte ki, hogy nincs negatívum, például:

- *„Nem tapasztaltam negatívumokat”*
- *„Semmit”*
- *„Nem volt semmi ami különösebben negatív, tetszett a tábor”*

Visszatérő negatív elem az időbeosztás, például kevés idő a kódolásra, hosszú megbeszélések, kevés szünet stb.:

- *„Kicsit sok volt a feladat, így sokra nem volt időnk lekódolni”*
- *„Megbeszélés unalmas, néha nincs elég idő az implementációra.”*
- *„Nem tudom, hogy lehetne-e ezen egyáltalán jól javítani, de a sok résztvevő és sokféle feladat miatt néhány megbeszélésére, megoldásainak elmagyarázására nem jutott idő.”*
- *„sajnos ereztem hogy a szunetek ellenere is elegge sok ideig ultunk a gep elott, ami miatt egy ido utan faradtnak ereztem magamat.”*
- *„Néha mikor rájöttünk egy feladatra/elkezdtük programozni utána volt szünet, ami félbeszakította a gondolatmenetünk, utána pedig már megbeszélés volt, ahol megbeszéltük a megoldást”*
- *„Neha nem volt eleg ido a csapatban megbeszelni es le is kodolni a feladatot.”*
- *„Talán kicsit kevés volt a feladatok száma a megbeszélések sűrűségéhez képest.”*

Ehhez hozzá kell tenni, hogy egyrészt szándékosan van sok feladat, hogy senki ne unatkozzon, és arra figyelünk, hogy a fontos feladatokra legyen ideje mindenkinek. Azt nem lehet sajnos elkerülni, hogy egyes csoportoknak nincs ideje minden feladatra, hiszen mások jóval gyorsabban haladnak. Azt is próbáljuk tudatosítani a diákokban, hogy nem baj, ha nem jut mindenre idő, de sajnos nagyon erős bennük az a beidegződés, hogy nekik minden feladatot meg kell csinálniuk. A tábor rövidsége miatt fel kell vállalnunk a kompromisszumokat.

Külön gyakori elem a megbeszélések hosszúsága, unalmassága:

- *„Túl sok időt töltünk el néhány feladat megbeszélésével."*
- *„Az elhúzódó megbeszélések néha olyan témákról szóltak ami számomra nem volt érdekes"*
- *„A feladatok megbeszélése egy kicsit lassú, mintha sokszor a tervezettnél több ideig tartottak volna"*
- *„Megbeszélések, amikor hosszúak, unalmasak."*
- *„tudásszintbeli különbségek, elhúzódó megbeszélések"*
- *„Szerintem kevés volt a szünet, mert a megbeszélések engem nagyon elfárasztottak és rögtön utána csapat versenyezni (vagy csak gondolkozni is) sok volt nekem."*

Ezen jó lenne javítani, de nem egyszerű, mert nyilván azoknak a diákoknak unalmasak a megbeszélések, akiknek könnyen mentek a feladatok, viszont egyes diákoknak nagyon fontos, hogy lassan, jól érthető formában átbeszéljük a megoldásokat még egyszer, és rendszerezzük a tanultakat.

Többször előfordul, hogy a tábori anyagra, témákra panaszkodnak:

- *„Voltak olyan témák, amiknek az alapjaival kapcsolatban már a tanultak nagy részét ismertem"*
- *„Nem sok új elméletet tanultam"*
- *„Kicsit túl gráfosnak éreztem a témát. Voltak más feladatok is (bitmask, backtrack), de azokon nem kellett sokat gondolkodnom."*
- *„Kicsit sok volt a kombinatorikai feladat és kicsit unalmasak is voltak."*
- *„Lehetnének komolyabb módszerek"*

A feladatok is visszatérő elem:

- *„Ha ismered a feladatokat, nehéz motiváltnak maradni"*
- *„Kevés feladat volt, amin gyakorolhattunk volna, emiatt nem tudom gyorsan lekódolni őket."*
- *„egy kicsit sok volt a feladat"*
- *„Kicsit nehezek voltak számomra a feladatok"*
- *„A végére már nem volt nagyon olyan típusú feladat, amibe bele tudtunk volna kezdeni, mert vagy túl nehéz volt, vagy már kész volt"*

Néhányszor említik a tempót:

- *„Nekem kicsit lassú volt a tempó"*
- *„A gyors tempó elég sok nyomást helyezett ránk, sokszor nem volt idő egy jó ötletet leprogramozni."*

- *„Volt ami nekem kicsit lassú volt, az online megoldás miatt néha lemaradtam 1-2 mondatról ami zavaró volt.”*
- *„Nagyon tömény volt az egész, és a végére úgy éreztem, kezdtem elveszteni a fonalat...”*

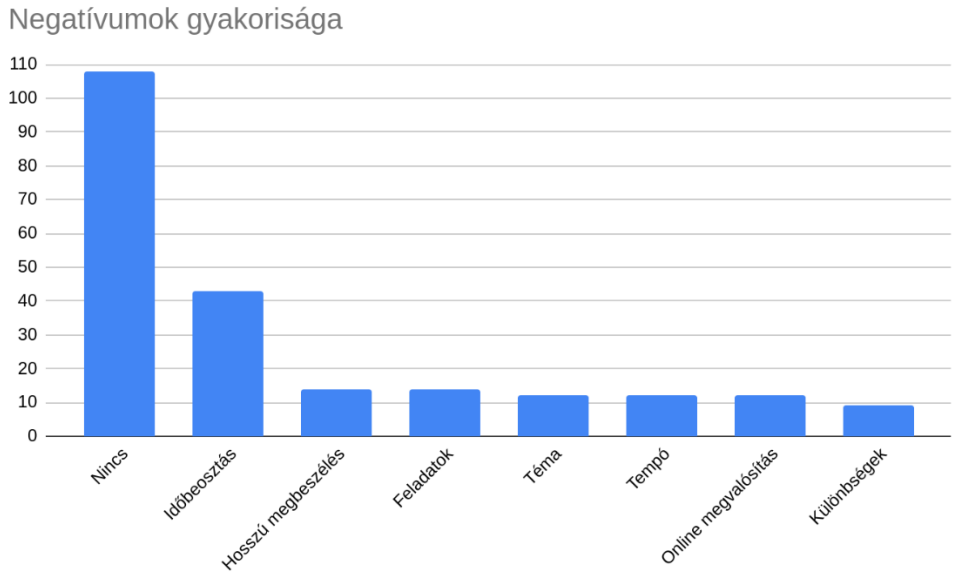
Azt is látják a gyerekek, hogy a tudásbeli különbségek nehézséget okoznak a táborban:

- *„Nagy szintbeli különbségek vannak a résztvevők között.”*
- *„Talán az egy kicsit negatívum, hogy viszonylag vannak nagy különbségek az emberek tudása között, és ezért nem sok újat tanultam. De remélem, idővel ez a különbség csökken.”*
- *„különböző dolgokat ismerünk, és ezért nehéz mindenkinek újat mondani”*
- *„Volt egy kis előismeretekbeli különbség a csoporton belül, így nem tudtunk az összes feladaton közösen gondolkozni.”*

Végül, az online táborok esetén gyakran említik a személyes találkozás hiányát, a technikai problémákat, és azt, hogy akkor végig a gépnél kell ülni:

- *„csak a helyzet nem tetszik, hogy nem egy helyen vagyunk sajnos”*
- *„Az internet egy elég nagy probléma volt, mert nem otthonról vettem részt a foglalkozáson, és egyszerre 2 óránál többet nehéz volt gép előtt ülni.”*
- *„Valamikor nehéz online felületeken beszélni.”*
- *„Fárasztó volt 3,5-7 órát ülni a gép előtt naponta”*

Az egyes negatív motívumok gyakoriságát mutatja a 296 válasz között az alábbi diagram.



5.15. ábra: A negatívumok között a gyakori motívumok előfordulásainak száma

Örömteli, hogy elsöprő többségben vannak a semmi negatívum típusú válaszok. Ha párba állítjuk a közösen előforduló elemeket a pozitívumokkal (pl. Feladatok, téma, megbeszélések), akkor kivétel nélkül minden elemmel kapcsolatban több a pozitív visszajelzés. Ennek ellenére

természetesen sokkal hasznosabbak ezek a negatív visszajelzések, hiszen ezek alapján tudjuk még tovább fejleszteni a ProgTáborokat.

5.5. A ProgTáboros diákok nemzetközi diákolimpiai részvételei

Minden évben 4-4 kiválasztott magyar diák részt vehet a következő nemzetközi informatikai diákolimpiákon, amelyek mind nagy presztízsű egyéni versenyek:

- Nemzetközi Informatikai Diákolimpia (IOI)
- Közép-Európai Informatikai Diákolimpia (CEOI)
- Európai Lány Informatikai Diákolimpia (EGOI)
- Európai Junior Informatikai Diákolimpia (EJOI)

A résztvevőket az olimpiai válogatóversenyeken választják ki, ahova az országos versenyeken legjobban szereplő diákokat hívják meg. Részletes információ Erdősné doktori értekezésében (Erdősné, 2019) illetve az ELTE IK – NJSzT Tehetséggondozási Szakosztályának honlapján található (NJSzT - ELTE IK, 2025).

Az alábbi táblázatok szemléltetik a tábori csoportjaimba járó diákok eredményeit a fenti nemzetközi versenyeken. A statisztika önmagában impozáns, azonban nem szeretnék semmilyen ok-okozati következtetést levonni belőle, hiszen nagyon sok hatás éri a diákokat a tanulmányaik során, a táborok csupán egy tényező a sok közül. Kijelenthető viszont, hogy a táborokba valóban az ország legtehetségesebb diákjai járnak. Továbbá örömteli tény, hogy ilyen sok diákunk kap lehetőséget nemzetközi versenyen való szereplésre, ahol remekül helyt is tudnak állni. A táborok 2020-ban indultak, de 2022 előtt még a táborosaimnál idősebb diákok vehettek részt (az EJOI-t kivéve).

	IOI	CEOI	EGOI	EJOI	Unió
Pilot csoport	2	2	2	-	**4**
Alpaka csoport	6	4	3	6	**10**
Graphisoft csoport	3^+	4^+	1	5	**8^+**
Realogic csoport	0^+	0^+	1^+	7*	**7^+**

$^+$ Ezekben a számokban még növekedés várható az elkövetkező években.
* Az EJOI 2025 résztvevői már ismertek, az eredmények még nem, ez az oka az eltérésnek az 5.14. táblázattól.

5.13. táblázat: A nemzetközi versenyeken részt vevők száma az egyes csoportokból

	Pilot csoport	Alpaka csoport	Graphisoft csoport	Realogic csoport	Nem táboros diákok
IOI 2022	🎗️🎗️	🥈🥉			
IOI 2023	🥉🥉	🥈🥉🥉🥉🎗️	🎗️		
IOI 2024		🥇🥈🎗️	📜		
IOI 2025		🥇🥉	🥇🥉		
CEOI 2022	🥈🥈	🥈🥉			
CEOI 2023		🥈🥈📜📜			
CEOI 2024		🥇🥈	🥇🥉		
CEOI 2025			🥈🥉🥉📜		
EGOI 2022	🥉📜	🥉			🥉
EGOI 2023	📜	🥈📜			🥈
EGOI 2024		🥈🥈🎗️	🥉		
EGOI 2025		🥇	🥇	🥈	🥈
EJOI 2020		🥉📜			📜📜
EJOI 2021		🥇🥈🥉🥉			
EJOI 2022		🥈🥉📜	🥉		
EJOI 2023			🥇🥉📜📜		
EJOI 2024			🥈	🥉🥉📜	

Jelmagyarázat: 🥇 - aranyérem, 🥈 - ezüstérem, 🥉 - bronzérem, 🎗️ - dicséret, 📜 - részvétel

5.14. táblázat: A táboros diákok nemzetközi versenyeken elért eredményei

6. Online szakkörök és táborok

Az elmúlt évek lezárásai alatt az online térbe kényszerült oktatás, bár számos hátránnyal járt, új lehetőségeket is megnyitott. A virtuális világban ugyanis könnyedén és hatékonyan lehet olyan foglalkozásokat szervezni, amelyekhez az ország különböző pontjairól csatlakozhatnak a diákok. Mivel a tehetséges gyerekek jellemzően szétszórtan bukkannak fel, egy-egy hagyományos iskolai tehetséggondozó szakkörön általában vagy nagyon kevesen vesznek részt, vagy nagy különbségek mutatkoznak a tudásszintjük között. Ez különösen igaz a programozásra, ahol a tanulmányi versenyek anyaga jelentősen meghaladja az iskolai alaptantervet. Az online szakkörök lehetővé teszik, hogy azonos érdeklődésű és hasonló tudású diákok különböző helyekről alkossanak csoportokat, így közösségben, a számukra legmegfelelőbb szinten tanulhatnak.

6.1. Új lehetőségek, előnyök és hátrányok

A COVID-19 járvány miatt világszerte elrendelt iskolabezárások és az online oktatás negatívan hatottak a gyerekek mindennapi életére és fejlődésére (Masonbrink & Hurley, 2020). A szociális interakciók hiánya, az otthoni elszigeteltség és a kevésbé ingergazdag virtuális tanulási környezet veszélyeztetheti a fiatalok mentális egészségét is (Ma, et al., 2021; Yang, Zhang, Kong, Wang, & Hong, 2021). Az új oktatási forma a diákok teljesítményére is kihatással volt. Clark és kutatócsoportja megállapította (Clark, Nong, Zhu, & Zhu, 2021), hogy az online oktatás alatt a diákok többségének pontszámai és jegyei javultak, kivéve a legjobb tanulók esetében. Különösen azok a korábban gyengén teljesítő diákok fejlődtek sokat, akik virtuálisan részt vehettek az iskolájukon kívüli, magasabb színvonalú oktatók óráin. A sok nehézség és kihívás (Kollár, 2021) mellett az online oktatásban szerzett tapasztalatok a diákok és a tanárok részéről is új lehetőségeknek adtak utat (Adedoyin & Soykan, 2020).

2020 tavaszán, valamint a 2020-2021-es tanév során a programozási tehetséggondozó szakköröket és versenyfelkészítő órákat kizárólag az interneten keresztül tudtuk megtartani. Ezáltal a diákok az ország bármely részéről, sőt, a határon túlról is egyszerűen csatlakozhattak. A közoktatás során megszerzett rutin, a kidolgozott technológiai megoldások, valamint a mindenkinek rendelkezésére álló otthoni számítógépek és internetkapcsolatok révén ma már ezeknek az online foglalkozásoknak a megszervezése és lebonyolítása szinte teljesen akadálymentessé vált.

A tehetséggondozás szempontjából óriási előnyt jelent, ha megszűnnek a földrajzi korlátok, és így a távol élő, kivételes képességű gyerekekből csoportokat hozhatunk létre. A tehetséges gyerekek ugyanis gyakran elszórtan jelennek meg, és sok helyen nincs megfelelő pedagógus, vagy olyan foglalkozás, amelyen fejlődhetnének. Még ha egy-egy iskolában van is programozó szakkör, a gyerekek képességei valószínűleg nagyon eltérőek, ami megnehezíti, hogy mindenki a saját ütemében fejlődjön. Ráadásul kevesen foglalkoznak programozással, így előfordulhat, hogy egy iskolában nincs elegendő diák egy egészséges létszámú csoport kialakításához.

Az országosan szervezett online tehetséggondozó szakkörök esetében elegendő néhány kiváló tanár, akik központilag vezetik a foglalkozásokat. Ezekben a szakkörökben olyan csoportokat alakíthatunk ki, amelyekben a diákok hasonló tudásszinttel rendelkeznek. Az országos szinten már elegendő diák van ahhoz, hogy 10-20 fős csoportokban mindenki a saját szintjén fejlődhessen. Az online foglalkozások másik előnye, hogy rugalmasabb időpontokat lehet találni, hiszen hétvégén és hétköznap késő délután is lehet órákat tartani, nem csak az iskolai tanórák után. Természetesen a virtuális térnek megvannak a maga hátrányai is: technikai problémák adódhatnak, hiányoznak a személyes kapcsolatok, és a tanárok és diákok kevesebb visszajelzést kapnak.

6.2. Szoftvereszközök és platformok

Számos szoftvert és weboldalt szoktam használni online oktatás közben, az utóbbi években letisztult ezeknek a halmaza, és itt bemutatom azokat, amelyek nekem a leghasznosabbak algoritmusok és programozás online tanításához. A bemutatott weboldalak elérési útvonalát a dokumentum végén, a 10. pontban adtam meg.

6.2.1. Discord

A Discord egy sokoldalú kommunikációs platform, amely közösségi térként szolgál, számos szöveges és hangcsatornával, valamint különféle szerepekkel és jogosultságokkal. A Discord-ot eredetileg játékosok számára fejlesztették ki, hogy könnyen kommunikálhassanak egymással játék közben. A platform kiválóan alkalmas volt a valós idejű szöveges és hangalapú csevegésre, ami elengedhetetlen a csapatjátékok során. Mára azonban a Discord funkcionalitása jelentősen kibővült, és számos más területen is alkalmazzák. Az alkalmazást most már különféle közösségek, tanulócsoportok, munkacsoportok és hobbi klubok is használják. Oktatási célokra is széles körben elterjedt, ahol tanárok és diákok online órákat és szakköröket szerveznek. Ezen kívül művészek, fejlesztők és egyéb szakemberek is

előszeretettel használják projektek megbeszélésére és együttműködésre. A Discord emellett számos további funkciót kínál, mint például a botok integrációja, amelyek automatizálhatják a feladatokat és interaktív funkciókat nyújthatnak. A platform támogatja az emojik és reakciók használatát a gyors kommunikáció érdekében, valamint lehetőséget biztosít a fájlmegosztásra. A videó- és hanghívások mellett képernyőmegosztás is elérhető, amely lehetővé teszi a valós idejű együttműködést. A Discord mobil és asztali alkalmazásai révén bárhonnan elérhető, így biztosítva a folyamatos kapcsolatot és együttműködést a csapattagok között.

A szakkörök és táborok tekintetében elsősorban azért esett a választásunk a Discord használatára, mert a platform lehetőséget biztosít a csoportmunkák hatékony szervezésére privát csatornák révén, ahol csak a csoport tagjai és a mentorok férhetnek hozzá a tartalomhoz. A közös információkat külön szöveges csatornákba tudjuk rendezni, például feladatlisták, fontos hirdetmények és gyors információmegosztás számára. A közös megbeszéléseket hangcsatornákban tartjuk, ahol a képernyőt tudjuk streamelni, de sok esetben a közös látványt alternatív módszerekkel oldjuk meg, amelyek hatékonyabb együttműködést és jobb skálázhatóságot biztosítanak, mint a videó stream. Ezek közé tartozik az USACO IDE, amiben közösen tudunk programozni, és a Miro illetve az Excalidraw, amik közös online rajzolófelületet biztosítanak.

6.2.2. USACO IDE

Az USACO IDE egy valós-idejű, kollaboratív online fejlesztői környezet, amelyet a Competitive Programming Initiative hozott létre annak érdekében, hogy a programozási versenyfeladatok megoldásához legyen egy könnyen használható platform, amelyet bárhonnan, telepítés nélkül el lehet érni. Az IDE beépített szerkesztőt kínál, amely lehetővé teszi a versenyzők számára C++, Python és Java nyelven kód írását és formázását, miközben a link megosztásával valós időben mindenki láthatja ugyanazt a kódot, sőt be lehet állítani azt is, hogy mások is tudják közösen szerkeszteni. Egy gombnyomás segítségével lefuttathatjuk a programot az előre megadott bemenetekre (a kód szerver oldalon fut).

A táborokban és szakkörökön előszeretettel használjuk ezt a fejlesztőkörnyezetet, mivel könnyű megosztani a megoldásokat vele, és hatékonyan tudjuk segíteni a diákokat, például tudunk együtt hibát keresni egy-egy programban. Ideális a páros programozáshoz és csoportmunkához is, és gyakran használjuk még akkor is, amikor nem online, hanem személyesen tartjuk a táborokat.

6.2.3. Online whiteboard szolgáltatások: Miro, Excalidraw

A feladatok elméleti megoldásainak megbeszéléséhez kiválóan tudunk használni online whiteboard szolgáltatásokat, ezek közül például a Miro és az Excalidraw azok, amiket rendszeresen használunk jelenleg. Ezek a platformok lehetővé teszik a felhasználók számára, hogy virtuális táblákra rajzoljanak, írjanak, jegyzeteljenek és együtt dolgozzanak, mindezt az interneten keresztül. Tehát rugalmasan tudunk együttműködni, függetlenül attól, hogy hol vagyunk a világon, és ilyenkor a képernyőmegosztásnál jóval kisebb sávszélességet veszünk igénybe.

6.2.4. VisuAlgo

A VisuAlgo egy interaktív weboldal, amely segítségével könnyen megérthető és vizualizálható módon taníthatunk algoritmusokat és adatszerkezeteket. Az előnyei közé tartozik, hogy lehetővé teszi az algoritmusok működésének élő vizsgálatát, interaktív megjelenítést nyújt, és segít abban, hogy a felhasználók könnyen megértsék az algoritmusok logikáját és működését a gyakorlatban. A megjelenítés közben beavatkozhatunk az algoritmusok futásába, például lépésről lépésre követhetjük az algoritmus működését vagy módosíthatjuk a bemeneti adatokat. Kiváló eszköz arra, hogy olyan szemléltetést alkalmazzunk a tanítás közben, amelyet később a diákok önállóan is ki tudnak próbálni. Az algoritmusok és adatszerkezetek széles skáláját tartalmazza, amelyek kategóriák szerint vannak csoportosítva, például keresési algoritmusok, rendezési algoritmusok, gráfalgoritmusok stb.

6.2.5. Online feladatgyűjtemények és hasznos funkcióik

A 4. fejezetben már részletesen írtam arról, hogy online feladatbankokban elérhető feladatokat használunk, amelyben automatikus kiértékelés útján a diákok megoldásait ellenőrzi is a rendszer. Alább azokat a funkcióikat emelem ki, amelyek előnyt jelentenek az online tanításhoz és a szakkörök és táborok szervezéséhez.

6.2.5.1. Codeforces

A Codeforces-t már több ízben említettem, mint kiváló versenyprogramozási platform. Van néhány tulajdonsága, ami miatt különösen alkalmas a céljainkhoz:

- A beadások publikusak és valós időben láthatók: így az adott feladatnál lehet figyelni a diákok megoldásait, illetve egy-egy megoldást link segítségével könnyen meg lehet

osztani, valamint azt is mutatja, hogy milyen tesztesetekre milyen eredményt adott a beadás.

- Mashup funkció: könnyen létrehozható egy privát verseny a korábbi versenyek feladataiból válogatva. A versenyt lehet egyéneknek, de csapatoknak is hirdetni. Egy online tábor esetében így szinte teljes értékű csapatvetélkedőt tudunk szervezni.
- Groups: csoportok hozhatók létre a felhasználókból, a csoportba pedig fel lehet tenni a Mashup feladatsorokat. A csoportban be lehet állítani jogosultságokat, így a segítőim egy-egy privát verseny közben is tudják nézni a diákok beadásait.
- Lists: létrehozhatók listák a felhasználókból, amivel lehet szűrni a lista tagjaira egy-egy verseny eredményeinél, és a feladatbankban is kijelzi, hogy a tagok közül hányan oldottak meg egy-egy feladatot. Így fel lehet használni olyan feladatok keresésére, amiket a listából még nem oldott meg senki.
- Tags: a feladatoknál címkék segítségével kijelzi, hogy milyen témakörhöz kapcsolódnak, és a feladatlistából erre lehet szűrni, ha bizonyos témakörökhöz keresünk feladatot. A címkéket egy beállítás segítségével el is lehet tüntetni, ezt javasoljuk a diákoknak, hogy ne befolyásolja a gondolkodásukat.
- Rating: a felhasználók Élő-pontrendszerhez hasonlító értékelése mellett immár a feladatok is kapnak pontszámot, ami a nehézségüket jelzi, ez szintén hasznos akkor, amikor egy-egy feladatsort állítunk össze.
- Tutorialok: a versenyek után közzétesznek megoldási útmutatókat is a feladatokhoz, ezeket fel tudjuk használni a foglalkozásokra készüléskor.

6.2.5.2. AtCoder

Az AtCoder hasonlóan átfogó programozási versenyplatform, mint a Codeforces, az alábbiak miatt különösen hasznos nekünk:

- Könnyű, bevezető feladatok: az AtCoder-en hetente vannak kezdőknek szánt versenyek (AtCoder Beginner Contest), amelyekben az első néhány feladat a programozási alapismereteket igényli csak, és kezdők tanításához nagyon jól használhatóak.
- Rövid, lényegretörő feladatleírások: a feladatokat úgy fogalmazzák meg, hogy legyen mese benne, de minél kevesebb felesleges szöveget tartalmazzon, ami például a Codeforces feladatok esetében láthatóan nem cél. Így a diákoknak könnyebb érteni őket, illetve angolról magyarra fordítani is egyszerűbb. A bemenetleírás egységes és jól áttekinthető formátumban van megadva.

- A beadások publikusak és valós időben láthatók: így lehet figyelni a diákok megoldásait, mutatja a beadott programjukat, valamint az összes tesztesetre kapott visszajelzést, de sajnos a teszteseteket magukat nem, ez megnehezíti a hibakeresést.
- Editorialok: a legtöbb feladathoz elérhető megoldási útmutató angolul.

6.2.5.3. CSES

A Code Submission Evaluation System feladatbankját kifejezetten oktatási célra fejlesztették, így magában foglal számos szép és klasszikus feladatot, amelyek sok esetben egy-egy tankönyvi algoritmus megvalósítását várják el.

- Egyszerű feladatszövegek: könnyen érthető, rövid, minimális mesét tartalmazó feladatszövegek vannak.
- Letölthető tesztesetek: a beadás után megjeleníti az összes tesztesetet a weboldal, amelyek teljes terjedelmükben letölthetők, valamint azt is, hogy milyen eredményt produkált a beadott program és mi lenne a helyes kimenet. Ez nagyban segíti a hibakeresést.
- Kódmegosztási lehetőség: sajnos a beadások nem publikusak, így nem lehet látni a diákok beadott programjait, viszont van egy „Share code to others” lehetőség, amivel a feltöltött kódot egy link segítségével meg lehet osztani.
- Minimalista megvalósítás: a weboldal nagyon egyszerű, így gyorsan működik minden platformon, nincsenek felesleges funkciók, mégis kényelmes a használata.

6.2.5.4. Snakify

A Snakify weboldalt a Python nyelv feladatokon keresztül tanítására fejlesztették. A weboldal felépítése és kialakításában alkalmazott pedagógiai elvek nagyon közel állnak ahhoz a felfedeztető módszertanhoz, amelyet meg szeretnék valósítani. A Python nyelv elemei nagyon tömören, példákkal illusztrálva vannak leírva az elméleti összefoglaló lapokon, és minden témakörhöz számos, egyre nehezedő feladat tartozik, amivel el lehet sajátítani az egyes programozási alapelemek használatát. Kezdőknek online szakkör tartásában sokat segít, mivel:

- A diákok a böngészőben tudják írni a programjaikat, és automatikusan le tudják tesztelni a megoldásaikat úgy, hogy látják a tesztbemeneteket, helyes kimeneteket és a programjuk kimenetét is.

- A tanári felületen táblázatos formában lehet látni, hogy mely feladatokat oldották meg a diákok, de kijelzi a hibás próbálkozásokat is, és meg lehet nézni a legutóbb beadott programjukat minden feladatnál. A diákok csoportokba rendezhetők.
- Létre lehet hozni saját feladatot is a rendszerben.

A rendszer hátránya, hogy a feladatok angolul vannak, ami a fiatal diákoknak nehézséget okozhat, ezért a lefordított szöveget posztoljuk nekik a Discord-on. További hátrány, hogy miközben a diákok írják a programot, csak képernyőmegosztással tudjuk nézni, és ezáltal kissé nehézkes segíteni nekik. Szintén negatívum, hogy reklámok vannak a weboldalon.

6.2.5.5. Virtual Judge

A Virtual Judge weboldal nem egy online kiértékelő vagy egy feladatbank, hanem itt számos online judge feladataiból válogatva lehet összeállítani virtuális versenyeket. Többek között az összes olyan nemzetközi online értékelőrendszerhez tud kapcsolódni, amelyet a 4.3.1. pontban bemutattam. Így a diákoknak nem kell minden weboldalon saját fiók, egy rendszeren belül be tudnak adni megoldásokat. A beadott megoldásaik valós időben láthatók, és táblázatos formában is megjeleníti őket a rendszer. Tulajdonképpen a Codeforces Mashup funkcióját tudja helyettesíteni úgy, hogy közben számtalan forrásból lehet feladatokat kitűzni benne.

6.3. Online táborok

A legelső, kísérleti ProgTábort a COVID-19 járvány miatt először el kellett halasztanunk 2020 tavaszáról későbbre, majd online térben tartottuk meg 2020. október 24-25-én (szombat-vasárnap). Így több szempontból is különleges volt, de a koncepcióra nézve is releváns tapasztalatokat és visszajelzéseket gyűjtöttünk. Az online formátum előnyeit kihasználva azóta is minden csoportnak az első táborát online tartjuk, hiszen így teljesen ingyenes, nincsenek utazással, szállással és étkezéssel járó költségek. Kiderülhet, hogy mely diákok számára értékes a tábor szakmai programja annyira, hogy részt vegyen a jelenléti tábor élményei nélkül is, valamint az is, hogy a megfelelő szintű képességekkel rendelkeznek-e.

Az online táborokat a Discord segítségével tartjuk. A közös megbeszélések során egy hangcsatornában vagyunk mindannyian és az oktató megosztott képernyővel beszél. A csoportmunkák során minden csoport külön hangcsatornában van, ahol a kamera bekapcsolását is kérjük a gyerekektől, hogy barátságosabb legyen. A csoportmunka alatt a gyerekek a 6.2.2. pontban leírt online IDE-ben tudnak közösen programozni, és a 6.2.3. pontban említett online

whiteboard alkalmazások segíthetnek a közös gondolkodásban. Így az online csoportmunka megközelíti a jelenléti táborok csoportmunkás foglalkozásait, viszont hátrány a személyes együttlét hiánya, valamint az is, hogy végig számítógép előtt kell lenni.

Az online értékelőrendszerek és a ProgTábor Bot (5.2.2.) segítségével a megoldások beadása és ellenőrzése teljesen ugyanúgy valósul meg, mint egy hagyományos táborban. A Codeforces Mashup funkcióval (6.2.5.1) és a ProgTábor Bot segítségével a csapatverseny szervezése nem különbözik a normál táboroktól, és a gyerekek számára teljes értékű vetélkedő lehet. Természetesen a személyes együttlétben a közös gondolkodás és egymásnak segítség kicsit hatékonyabban történhet, de a különbség nem jelentős.

Egy online táborban esténként közös online játékokat is szoktunk tartani. A tábor programja általában az alábbiak szerint alakul.

Péntek

- 17:00 - 19:00 - Bevezető foglalkozás, csoportok alakítása, új feladatokon gondolkodás csoportmunkában
- 19:00 - 20:00 - Szünet, vacsora
- 20:00 - 21:30 - Esti játék

Szombat

- 9:00 - 13:00 - Csoportmunkás foglalkozások és közös megbeszélések váltakoznak, új témaköröket tanulunk, közben 30 perc szünet
- 13:00 - 15:00 - Szünet, ebéd
- 15:00 - 19:00 - Megbeszélés és csoportmunka váltakozása, közben 30 perc szünet
- 19:00 - 20:00 - Szünet, vacsora
- 20:00 - 21:30 – Esti játék

Vasárnap

- 9:00 - 13:00 - Csoportmunka, megbeszélések, közben 30 perc szünet
- 13:00 - 15:00 - Szünet, ebéd
- 15:00 - 16:00 - Csoportmunka
- 16:30 - 18:30 - Csapatvetélkedő
- 18:30 - 19:00 - Táborzáró megbeszélés

Az 5.4. pontban bemutatott felmérések eredményeiben az online táborok is benne vannak. Megvizsgáltam bizonyos statisztikákat az online és jelenléti táborokra különválasztva. Az összehasonlítás olyan szempontból nem teljesen mérvadó, hogy sok egyéb tényező különbözött az online és jelenléti táborokban, például a tananyag, vagy az újdonság ereje, hiszen az online tábor a legelső, bevezető alkalom.

A rövid visszajelzéseken az élvezet és fejlődés értékek átlaga 4,36 és 3,95 az online táborok esetében, míg 4,33 és 4,16 a jelenléti táborokban. Nagyon meglepő és örömteli tény, hogy az online táborokat is ugyanannyira élvezték a diákok, viszont nem annyira bíztató a fejlődési értékekben mutatkozó különbség. Sajnos kétségkívül igaz az, hogy személyesen sokkal több visszajelzést lehet kapni oktatóként a diákoktól és hatékonyabb tudjuk segíteni őket a tanulásban, valószínűleg ebből adódik a különbség.

A további szakmai mutatók átlagainak összehasonlításában azt láthatjuk, hogy a feladatmennyiség és a tempó az online táborokban magasabb, az ideális 3-as értéktől távolabb eső értékelést kapott átlagosan, ráadásul különbség a feladatok mennyiségénél jelentős. Ez magyarázható azzal, hogy az első táborban több diáknak túl intenzív a munka és kevesebb előismerettel érkeznek, viszont mindenképpen érdemes változtatni rajta a jövőben. A feladatok érdekességében is a jelenléti táborok némileg magasabb átlagos pontszámot kapnak, de ennek lehet, hogy semmi köze a lebonyolításhoz. Viszont roppant fontos és rendkívül megnyugtató statisztika, hogy a magyarázatok érthetősége közel azonos az online táborokban is.

	Élvezet	Fejlődés	Feladatok érdekessége	Feladat-mennyiség	Tempó	Magyarázatok érthetősége
Online	4.36	3.95	4.26	3.47	3.18	4.52
Jelenléti	4.33	4.16	4.40	3.06	3.00	4.57

6.1. táblázat: Az online táborok szakmai mutatóinak átlagai a jelenléti táborokkal összehasonlításban

A fentiek alapján azt gondolom, hogy az online táborok alkalomadtán, amikor szükség úgy hozza, jól helyettesíthetik a jelenléti táborokat, de természetesen semmiképpen nem válthatják le őket teljesen.

6.4. Felmérés az online szakkörökről

6.4.1. Kérdésfelvetés

Úgy vélem, hogy a karanténidőszak alatt indult, majd azóta állandósult és kibővült online tehetséggondozó programozás szakkörök előnyei (6.1. pont) messze felülmúlják a hátrányokat, és számos diáknak teljesen új vagy a korábbiaknál sokkal jobb lehetőségeket kínálnak, így pótolhatatlan jelentőségűek. Ez volt az első hipotézisem, amelynek igazolására végeztem az itt bemutatott felmérést. Úgy gondolom, hogy az ezt eredményező legfontosabb hatás az, hogy megfelelő szintű csoportos oktatást tudunk nyújtani a kiemelkedően tehetséges diákoknak. Ez a második hipotézisem. A szakkörökön részt vevő diákok által kitöltött kérdőívek segítségével gyűjtöttem adatokat a hipotézisek vizsgálatára.

6.4.2. Szakkörök

A felmérés készítésekor egy online magániskolában tanítottam, ahol teljesen kezdő szinttől a nemzetközi versenyfelkészítésig tanulnak diákok programozni gondolkodtató módon, a fiatalabbak Logo, az idősebbek, illetve haladóbbak C++ nyelven. A diákok a tudásuk szerint vannak csoportba osztva, a belépéskor megoldott szintfelmérő feladatok alapján, illetve később az órákon mutatott teljesítményük alapján. A csoportok létszáma 10 és 20 fő közötti.

A szakkörökön a tanítás alapvetően problémamegoldáson keresztül valósul meg (Nikházy, 2020). Amikor egy-egy új témát mutatok be, mindig arra törekszem, hogy ezt megfelelő bevezető feladatok útján tegyem meg. Különösen fontosnak tartom, hogy időt adjak mindenkinek arra, hogy egyénileg vagy csoportosan gondolkodjanak a feladatokon. A megoldások megbeszélése közben hangsúlyozom, milyen módszereket tanultunk az adott feladat kapcsán. Ha egy bonyolultabb algoritmust vagy adatszerkezetet mutatok be, gondoskodom róla, hogy a diákok ezt különböző feladatokban, több új helyzetben is alkalmazzák. Korábbi tanítási tapasztalataim alapján fontosnak tartom, hogy a megbeszélt feladatokra programot is írjunk, amelyet online kiértékelővel tesztelhetünk, még akkor is, ha ez sok időt vesz igénybe a foglalkozásokból.

Ebben a felmérésben kizárólag a haladó csoportokat vizsgáltam, azokat, ahol már nem az alapokat tanítottuk, hanem problémamegoldási stratégiákat és algoritmusokat. Ha a hazai versenyek szintje alapján szeretném meghatározni, akkor nagyjából a Nemes Tihamér Verseny 1. korcsoport döntős szintjétől az Olimpiai Válogatóverseny szintjéig terjedtek a különböző

csoportok erősségei, összesen 5 csoportról van szó. Az egyes csoportokban lévő diákok létszámai: 15, 14, 19, 18 és 19 fő volt. Egy-egy csoport különböző korú, de hasonló tudású diákokból állt. A csoportok tanárai közé tartozott Csertán András, Szabó Kornél, Tóth Gellért, valamint én magam is.

Az online oktatás színtere ezúttal is a Discord platform volt. Minden csoportnak kéthetente volt 90 perces online órája. A foglalkozások között aktív egyéni munkát vártunk el, és segítettünk is nekik, ha kellett. Az óra előtt legalább egy héttel kiadtunk nekik 3-5 feladatot. Ezek egy része ismétlő feladat, amely a korábbiak elmélyítésére irányul, egy része pedig az új anyag felvezetése, tehát felfedeztetés a cél. A feladatok előre kiadása által a tanulóknak sok idejük volt rajtuk gondolkodni. Adtunk hozzájuk segítségeket is, amelyeket önkiszolgáló módon nézhettek meg. Az órán újabb 3-5 feladatot adtunk fel, amiken 2-3 fős csoportokban dolgoznak a diákok. A csoportmunkában ösztönöztük őket, hogy dolgozzanak össze, segítsenek egymásnak. A feladatokat elméletben mindenképpen közösen csinálják, a programozásra általában felkínáltuk azt a két lehetőséget, hogy 1) közösen csinálják: egyikük írja, a másik figyeli és segít („pairwise programming”), vagy 2) külön-külön megcsinálják, majd megnézik egymását és kijavítják a hibákat. A csoportok privát virtuális „szobában” dolgoztak (hang- illetve videókapcsolattal), és a tanár körbe járt a szobákban, figyelte a haladásukat és segítségeket adott.

6.4.3. Kérdőív

A felmérést az 5 haladó szakköri csoport diákjai között végeztem, és a kérdőív kitöltése önkéntes és anonim volt. A csoportok összlétszáma 85, közülük pedig 37 diák töltötte ki a kérdőívet. Olyan kérdéseket tettem fel, amelyek kifejezetten arról szóltak, hogy hogyan értékelik az online programozás szakköröket más lehetőségekhez viszonyítva:

- Hiánypótlónak érzed az online programozás szakköröket?
- Az online szakkörökön mennyire lehet hatékonyan tanulni, jelenléti foglalkozásokhoz hasonlítva?
- Az online szakkörökön mennyire lehet kényelmesen tanulni, jelenléti foglalkozásokhoz hasonlítva?
- Szerinted milyen előnye van az online szakköröknek?
- Szerinted milyen hátránya van az online szakköröknek?

Mivel a helyi programozás szakkörökkel és az iskolai informatika órákon történő programozás oktatással hasonlítottam össze az online szakköröket, ezért megkérdeztem a diákokat, hogy részt vettek-e már ilyen helyi eseményeken. Emellett érdeklődtem, hogy hányan járnak a szakkörökre, valamint általánosságban mennyi diák van az iskolában, aki hasonló szinten programoz:

- Mit gondolsz, az iskoládban kb. hányan vannak még, akik hozzád hasonló szinten foglalkoznak programozással?
- Van-e, vagy volt-e az utóbbi 1-2 évben az iskoládban, vagy a városban programozás szakkör, amire jártál?
- Nagyjából hányan vannak az iskolai szakkörön?
- Van-e, vagy volt-e az utóbbi 1-2 évben az iskolai informatika óráidon programozás oktatás?

Mindhárom típusú oktatásról feltettem ugyanazokat a kérdéseket, amelyek a hipotézisem alátámasztása szempontjából érdekesek voltak. Tehát a foglalkozások és a résztvevők szintje, a szakmai színvonal és a személyes fejlődés volt ezeknek a kérdéseknek a témája:

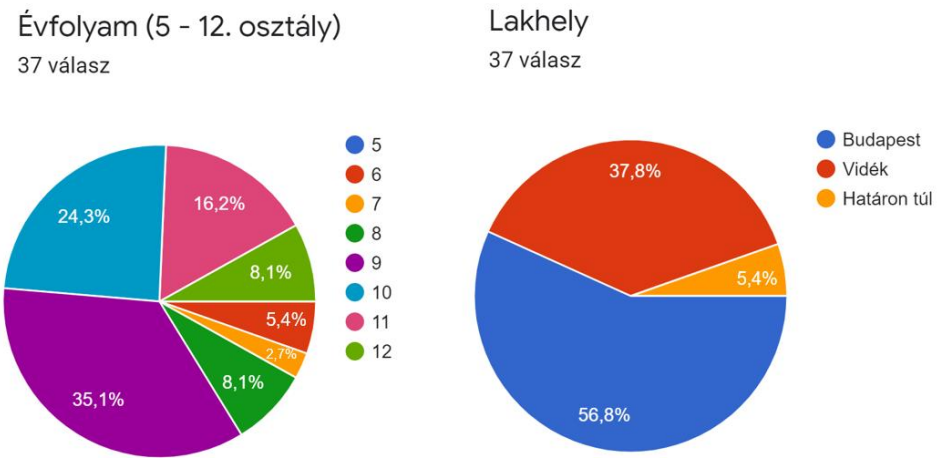
- Összességében milyennek értékeled az [online programozás szakköröket / iskolai programozás szakköröket / informatika órákon a programozás oktatást]?
- Milyen az [online szakkörök / iskolai szakkörök / informatika órákon tanult programozás] szakmai színvonala? (Értsd: nívós programozási tudást lehet megszerezni?)
- A tudásszintednek megfelelő oktatásban részesülsz? Jó neked [az online szakkör / iskolai szakkör / informatika órai programozás] szintje?
- Megítélésed szerint a társaid milyen szinten vannak hozzád képest?
- Az [online szakkörökön / iskolai szakkörökön / informatika órákon] érdemben tudsz fejlődni?
- Milyennek tartod az [online szakkörökön / iskolai szakkörökön / informatika órákon] a közösséget?

6.4.4. Eredmények

6.4.4.1. Kitöltők és iskolai, helyi oktatás

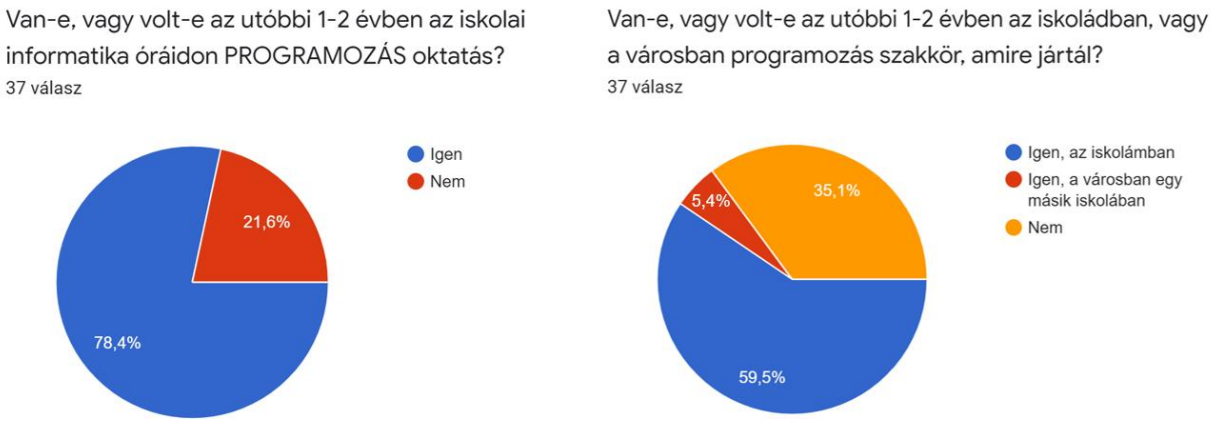
A 6.1. ábrán látható a kérdőívet kitöltő 37 diák évfolyam és lakhely szerinti eloszlása. A válaszadók mintegy háromnegyede 9-11. osztályos tanuló, azonban 6-12. osztályig minden

évfolyamból voltak diákok. Kiemelten fontosnak tartom megjegyezni, hogy a válaszadók közel fele vidéki vagy határon túli diák, és ezeket a válaszokat külön is elemezni fogom. A kérdőív anonimitása miatt nem kértem részletesebb adatokat. Bár a diákok mintája nem tekinthető teljesen reprezentatívnak, és létszámuk viszonylag kicsi, ahhoz azonban elegendő, hogy releváns következtetéseket vonjak le belőle.



6.1. ábra: Évfolyam és lakhely szerinti eloszlás.

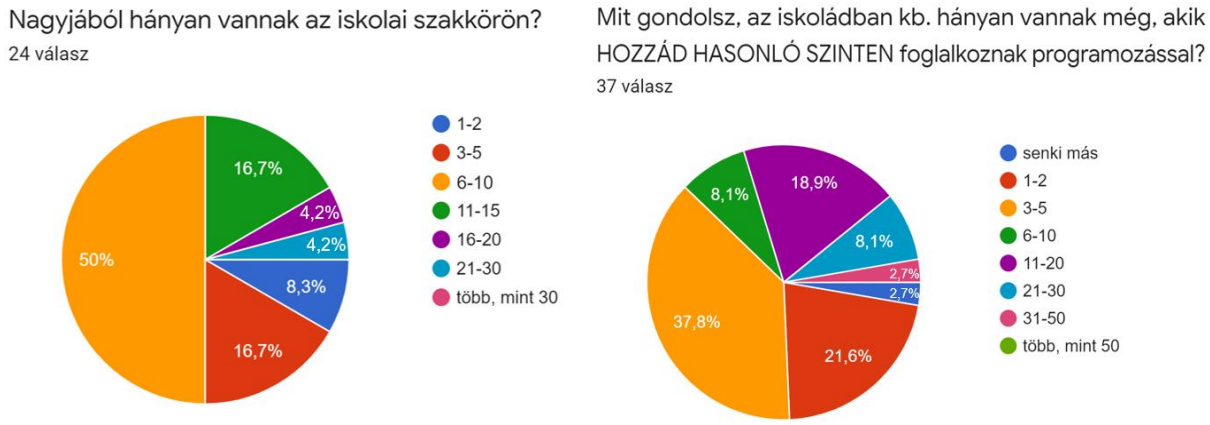
Örvendetes tény, hogy a kitöltő diákok mintegy 80%-a részt vett már programozásoktatáson az iskolában is, ezt a 6.2. ábra mutatja. Egy későbbi grafikonon látható lesz, hogy az iskolai informatika órák programozásának színvonala nem mindig felel meg az elvárásaiknak, ami érthető, mivel kiemelkedően tehetséges diákokról van szó. Ugyanakkor szomorú tény, hogy a 6.2. ábrán jobb oldalon látható kördiagram szerint az esetek 35%-ában a diákok nem jártak szakkörre az iskolájukban vagy a városukban. Ezért továbbiakban, amikor az online és az iskolai szakkörök megfelelőségéről beszélünk, fontos figyelembe venni, hogy több mint egyharmaduknak még az sem adatott meg, hogy az utóbbin részt vehessen.



6.2. ábra: Informatika órán programozás oktatás, és iskolai szakkörök.

A diákokat arról is kérdeztem, hogy részt vesznek-e az iskolai szakkörökön, és hogyan ítélik meg, hogy hányan vannak az iskolájukban, akik hasonló szinten foglalkoznak programozással. A válaszadók negyede olyan szakkörre jár, ahol legfeljebb 5 diák van, és a többi esetben sem haladja meg a résztvevők száma a 10-et. Ez alapján elmondható, hogy a szakkörök nagy részénél igen alacsony a részvétel. Összevetésképpen, az online szakkörök létszáma 14 és 19 között mozog. Szomorú tény, hogy a diákok mintegy 60%-a szerint az iskolájukban legfeljebb 5 diák van, aki hasonló szinten áll programozásban. Ebből adódóan gyakorlatilag lehetetlen megfelelő szintű szakköri csoportokat szervezni. Érdekesség, hogy az online szakkör előnyeiről szóló szabad szöveges válaszokban többen megemlítették, hogy ott magasabb a létszám:

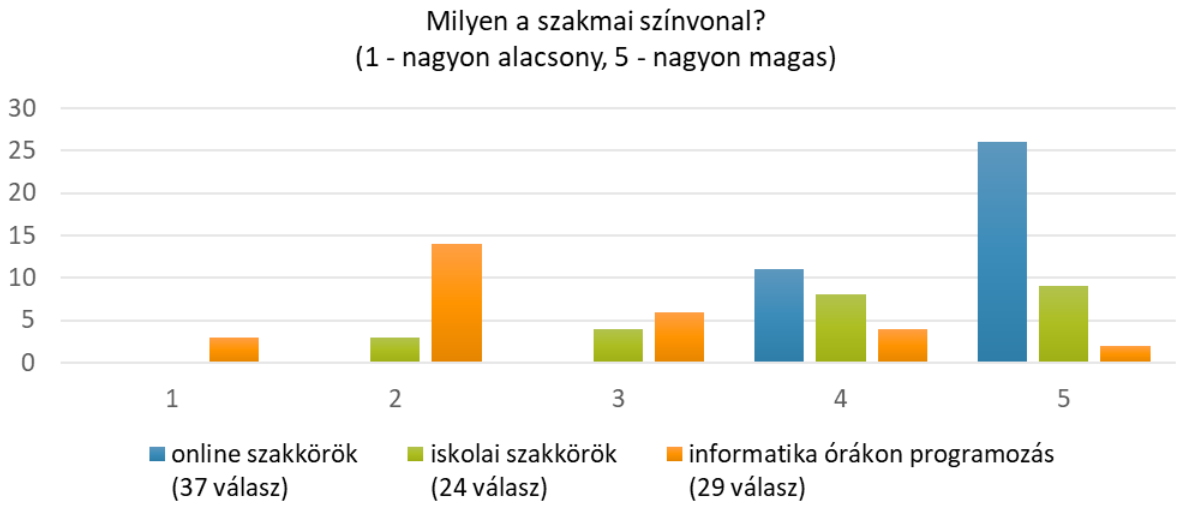
- *„Több ember részt tud venni rajta, nincs az a probléma, hogy sokan nem ugyan ott élnek.”*
- *„Így nem csak egy város diákjainak lehet a szakkört megtartani.”*
- *„Magasabb részvételi arány”*



6.3. ábra: A szakkörök létszáma és a hasonló szinten lévő iskolatársak száma.

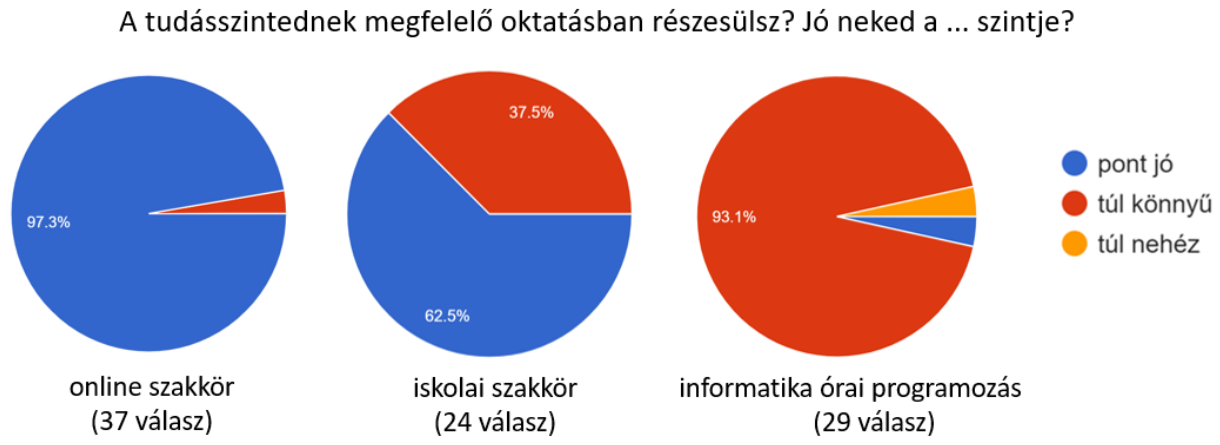
6.4.4.2. A háromféle programozás oktatás értékelése

Az online szakkörök szakmai színvonalát (nívós-e a programozás oktatás) jelentősen magasabbra értékelték a diákok, mint az iskolai szakkörök, illetve az informatika órák esetében ugyanezt (átlagok sorrendben: 4,70; 3,96; 2,59). Az eloszlásokat az alábbi oszlopdiagramon láthatjuk. Ezt a különbséget természetesen befolyásolhatja a tanár, a tananyag és a tanítási módszerek eltérése, de meglátásom szerint nagy szerepe van a résztvevő diákoknak is, hiszen az oktató alkalmazkodik hozzájuk. A továbbiakban elemzett válaszok ezt a megállapítást alátámasztják.



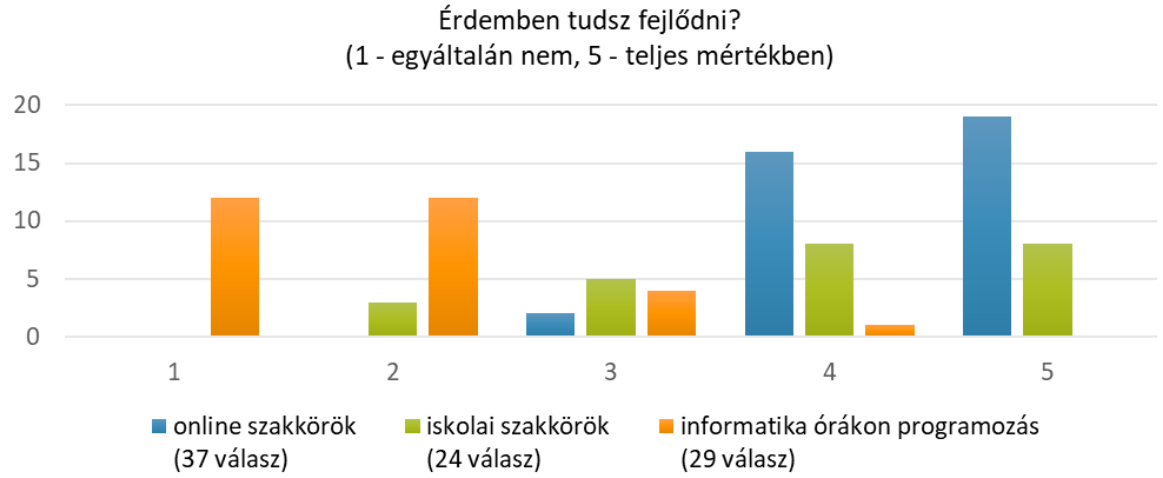
6.4. ábra: Szakmai színvonal értékelése.

A vizsgálat szempontjából kiemelten fontos kérdés, hogy a diákok a tudásszintjüknek megfelelő oktatásban részesülnek-e. Nagyon erős indikátor, hogy az online szakkörökről 97%-ban mondták azt, hogy pont jó a szakkör szintje. Az iskolai szakkörök esetében ez az arány csak 62,5%, és a diákok 37,5%-a még azt is túl könnyűnek találja. Az informatika órákon nyújtott programozás oktatás szintjére vonatkozó vélemények teljesen érthetőek, hiszen a tehetséges diákok számára gyakran túl alacsony a szint (bár differenciált oktatással ez megoldható lenne). Fontos megjegyezni, hogy nem minden iskolában van szakkör, így az említett 62,5% csak az összes kitöltő 40%-át teszi ki, vagyis még a diákok jelentős részének sincs lehetősége a megfelelő szintű szakkörre. Tovább elemezve ezeket a válaszokat kiderül, hogy a többségük budapesti diák, mindössze 4 vidéki vagy határon túli diák van, aki ezt állította, ami a vidéki / határon túli kitöltőknek a 25%-a. Ez azt jelenti, hogy a nem budapesti diákok csaknem negyede részesülhet az iskolájában vagy a városában megfelelő szintű programozás szakkörben, míg a budapesti diákoknál ez az arány több, mint a válaszadók fele.



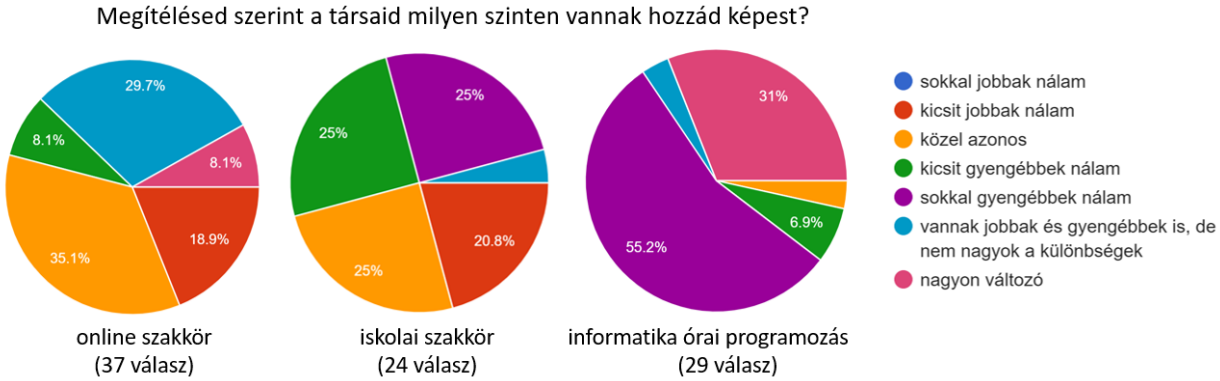
6.5. ábra: Az oktatás szintjének viszonya az egyéni tudásszinthez.

Ehhez kapcsolódik az a kérdés, hogy tudnak-e a diákok érdemben fejlődni a foglalkozáson. Az alábbi ábra mutatja a kapott válaszok eloszlását a három különböző foglalkozásra. Az eloszlásokból látható, hogy az online szakkörök esetén valósul meg ez leginkább (átlag: 4,46), de az iskolai szakkörök (átlag: 3,88) is messze az informatika órák felett vannak (átlag: 1,79).



6.6. ábra: Érdemi fejlődés értékelése.

Amikor arról kérdeztem a résztvevőket, hogy társaik milyen szinten vannak hozzájuk képest, az iskolai szakkörökről az esetek felében nyilatkoztak úgy, hogy a többiek kicsit vagy sokkal gyengébbek náluk. Viszont az online szakkörök esetében mindössze 8% mondta azt, hogy a többiek kicsit gyengébbek náluk, és senki sem nyilatkozott úgy, hogy a többiek sokkal gyengébbek nála. A résztvevők 35%-a úgy érezte, hogy közel azonos szinten van mindenki a csoportban, és további 30% nyilatkozta, hogy vannak jobbak és gyengébbek is nála, de nem jelentős a különbség. A diákok 65%-a tehát úgy érzi, hogy nagyjából a csoport átlagos szintjén van. Ez nagyon pozitív eredmény, különösen figyelembe véve, hogy az iskolai szakkör esetén ugyanez az arány csak 30%, az informatika órai programozás esetén pedig mindössze 7%.

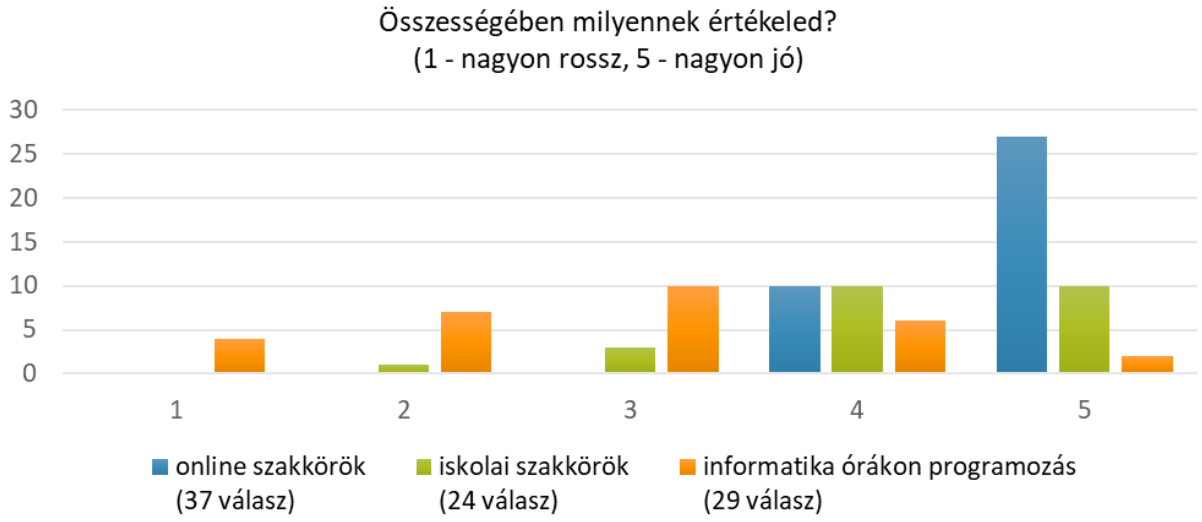


6.7. ábra: A társak szintje.

A szakmai és gyakorlati szempontok mellett kiemelt fontosságúak a diákok közti kapcsolatok és a közösségi élmények. Az erre vonatkozó kérdések alapján az iskolai szakkörök kapták a legjobb értékelést (átlag: 4,54), míg az informatika órák (átlag: 3,90) némi előnyt élveztek az online szakkörökkel szemben (átlag: 3,86). Tehát megerősítést nyert az is, hogy a diákoknak értékesebbek a közvetlen személyes kapcsolatok, mint a virtuálisak.

6.8. ábra: A közösség értékelése.

Az általános értékelés eloszlását is bemutatom a teljesség kedvéért, bár ebből a fentiekhez képest új következtetést nem vonhatunk le (átlagok: 4,73; 4,21; 2,83).



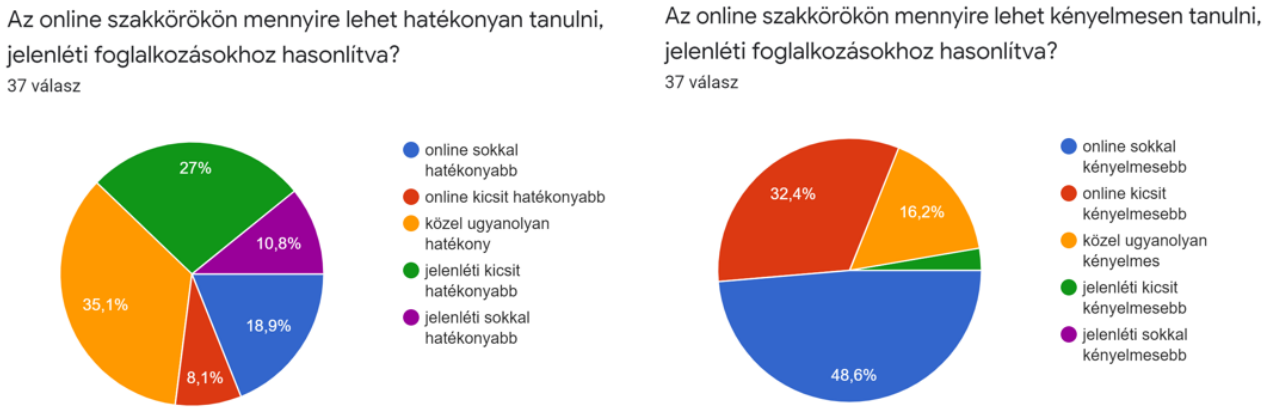
6.9. ábra: Általános értékelés.

6.4.4.3. Előnyök és hátrányok

Nagyon érdekes módon megosztott válaszokat kaptam arra a kérdésre, hogy online vagy jelenléti foglalkozásokon lehet hatékonyabban programozást tanulni. Kifejezetten fontos

számunkra, hogy csupán 11% mondja azt, hogy jelenléttel sokkal hatékonyabban lehet tanulni, ezzel szemben 19% véli úgy, hogy online sokkal hatékonyabban lehet tanulni. Mivel 35% szerint közel ugyanolyan hatékony a két forma, ezért összesen 62% állítja azt, hogy az online oktatás legalább olyan hatékony, vagy még hatékonyabb, mint a jelenléti. A kényelem szempontjából elsöprő sikere van az online szakköröknek, 81% szerint kényelmesebb az online, míg csak 3% részesíti előnyben a jelenléti órákat. Az online szakkör előnyeire adott szabad szöveges válaszok egy jelentős része is erről szólt, például:

- *„Nem megy el idő utazással"*
- *„Könnyebben megoldható az időpont, nem megy el idő utazással, saját gépet tudunk használni"*
- *„Sokkal kényelmesebb, hogy otthonról lehet tanulni"*
- *„Nem kell utaznom, praktikus, gyors, néhol személytelenebb, megmarad az elhangzott anyag, lehet hét közben is".*



6.11. ábra: Az online tanulás hatékonysága és kényelme a jelenlétihez viszonyítva.

Az online szakkörök előnyeiről szóló szabad szöveges válaszokban sokan említették azt, hogy nincsenek földrajzi korlátok, és az egyszerűen készíthető felvétellel az időbeli akadályok is részben kezelhetők:

- *„Az ország bármely pontjáról be lehet kapcsolódni, otthon kicsit kényelmesebb."*
- *„Részt tudnak venni azok is, akiknek utazással együtt nem lenne rá idejük, a szakkör közbeni helyváltoztatás kivitelezhető, a felvételek visszanézhetőek."*
- *„Az, hogy nem kell elmenni a helyszínre, és fel is lehet egyszerűen venni"*
- *„Több ember részt tud venni rajta, nincs az a probléma, hogy sokan nem ugyan ott élnek."*

Az otthoni megszokott környezet mellett még az időbeosztásról és a megspórolt utazási időről szól a válaszok jelentős része:

- *„Nem megy el idő utazással"*
- *„Könnyebben megoldható az időpont, nem megy el idő utazással, saját gépet tudunk használni."*
- *„Nem megy el időm az utazással, így könnyebben bele tudom építeni a napirendembe."*

- *„Kényelmesebb, rugalmasabban lehet tartani az órát”*
- *„Nem kell utazással eltölteni az időt, előbb haza lehet menni, rugalmasabb a többi program szempontjából”*

A hátrányok között a legtöbb válasz az esetlegesen felmerülő technikai problémákat teszi szóvá:

- *„Előfordulhat, hogy egyeseknek technikai gondjaik vannak, ez engem nem érint.”*
- *„Vannak olyanok, akiknél rossz az internet, vagy mikrofon és az eléggé zavaró tud lenni.”*
- *„Kicsit személytelenebb, esetleg lehetnek technikai problémák”*
- *„Néha nehéz kezelni a felületeket, amiken dolgozunk, általában pillanatnyi technikai okok miatt”*

Többen felvetik a személyes együttlét hiányát:

- *„Személy szerint én kicsit hatékonyabbnak érzem az oktatást, ha személyesen tudok kommunikálni a tanárokkal és a többi diákkal.”*
- *„Nagyobb távolságok vannak az emberek között. Nem biztos, hogy lesz alkalom találkozni”*
- *„Sokkal jobb, ha egy teremben ülök azokkal, akikkel együtt dolgozom”*

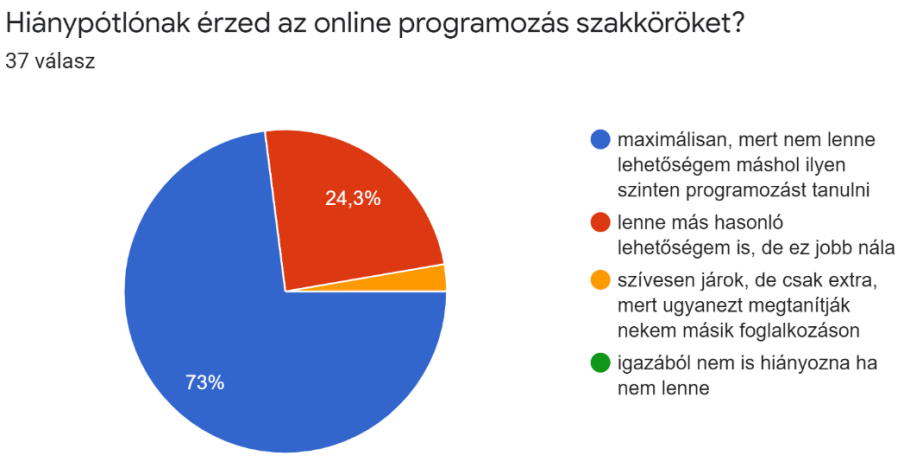
Érdekes és fontos visszajelzés az, hogy számos válaszadó szerint online nehezebb a figyelem fenntartása:

- *„Online könnyebben elkalandozik az ember figyelme, nehezebb koncentrálni”*
- *„Élőben könnyebb figyelni, megérteni az anyagot.”*

Bár van ellenvélemény is:

- *„Ha az ember fáradtabb, talán könnyebb elbambulni, és nem dolgozni. Bár még így is sokkal hatékonyabb a számomra az online óra, hiszen jelenlétinél nemcsak akkor nem tudok figyelni, amikor fáradt vagyok, hanem úgy általában.”*

Közvetlen kérdésként is feltettem a kérdőívben az elsődleges hipotézisemet, miszerint az online szakkörök hiánypótlóak. Az erre érkezett válaszok magukért beszélnek.



6.10. ábra: Hiánypótlóak-e az online szakkörök?

7. Az önálló problémamegoldás és a szemléltetett példákkal való tanulás összehasonlítása

A különböző felfedező tanulási módszerek hatékonysága a közoktatásban sokat vitatott téma. Kirschner és munkatársai (Kirschner, Sweller, & Clark, 2006) szerint a sok kidolgozott példán keresztül történő tanítás hatékonyabb, mint a minimálisan irányított oktatási módszerek. A 20. század második felében a felfedeztető tanítás különböző formái nagy figyelmet kaptak több tudományterületen, mind az egyetemi, mind az iskolai oktatásban. Számos kutatás született a különböző felfedeztető oktatási típusok előnyeiről és hátrányairól, mint például a kutatás alapú tanulás, a probléma alapú tanulás, az irányított felfedező tanulás és más típusok. Kirschner és társai mélyreható kritikai cikke egy érdekes tudományos vitát hozott elő, átfogó válaszokkal a másik oldalról (Schmidt, Loyens, Van Gog, & Paas, 2007). Én, aki a Pósa-féle matematikatáborok felfedeztető módszerével (Juhász & Katona, 2019) nagyon szerettem tanulni, és nagyon pozitív tanítási tapasztalatokat szereztem e koncepciók alkalmazásával az programozás tehetséggondozásban, hiányolok egy szempontot a fent említett cikkekből, azt, hogy a diákok mennyire élvezik a foglalkozásokat.

Kirschner és munkatársai azt javasolják, hogy egy erősen irányított, sok kidolgozott példát bemutató tanítási módszer hatékonyabb, mint a különböző felfedeztető módszerek. A hatékonyságot azonban a tanulók teljesítményében fejezik ki, miközben a tanulók elkötelezettségének ugyanolyan fontos jellemzőnek kellene lennie, hiszen nemcsak az a fontos, hogy mennyit tanulnak, hanem az is, hogy mennyire akarnak később még többet tanulni a témáról. A hosszú távú hatásokat nehéz mérni, de az elkötelezettséget rövid felmérésekkel és a nem kötelező feladatokra vonatkozó értékelésekkel lehet számszerűsíteni. Ebben a fejezetben egy ilyen, az egyetemi oktatásban végzett kutatás eredményeit mutatom be.

7.1. A tanítási kísérlet

Két párhuzamos gyakorlati csoportot tanítottam az ELTE Informatikai Karán programtervező informatikus BSc szakon tanított első féléves Programozás tárgyon a 2020/21. őszi félévben. A tananyag arra épül, hogy először egyszerű, majd összetettebb problémákat oldjunk meg alapvető algoritmikus sémák segítségével, mint például megszámolás, minimum/maximum kiválasztás, keresés, kiválogatás, sorozatszámítás és ezek kombinációi (Zsakó & Szlávi, 2008). A tárgy magában foglalja egy programozási nyelv alapelemeinek elsajátítását (ami akkor a C++ volt), miközben a hallgatóknak formális specifikációkat kell készíteniük valós

példaproblémákra, és a megoldási algoritmusokat struktogramokkal, más néven Nassi-Shneidermann-diagramokkal (Nassi & Shneiderman, 1973) kell kifejezniük. Az előadások elsősorban az elméleti hátteret mutatják be a formális specifikációk és a struktogramokon ábrázolt algoritmusok segítségével. Másrészt a gyakorlati órák a feladatok megoldására összpontosítanak C++ programokkal, hogy némi gyakorlati tapasztalattal további alapot nyújtsanak az elméletekhez. A részletes tanterv a kurzus honlapján tekinthető meg (ELTE, 2024). A COVID-19 járvány miatt minden órát online kellett megtartani. A heti 135 perces gyakorlati órák online videokonferenciáin kötelező volt a részvétel.

A két gyakorlati csoport diákjai ugyanazokat az előadásokat látogatták, ugyanazokat a teszteket írták, a feladatokat ugyanabban a rendszerben kapták meg, és a gyakorlati órák anyagai és feladatai is szinte azonosak voltak. Az egyetlen különbség a tanítás módszere volt. Az első csoportban igyekeztem minimális útmutatást adni a hallgatóknak, és mindig arra törekedtem, hogy először egyedül nézzenek szembe a problémákkal, elegendő időt biztosítva számukra a gondolkodásra, és szükség esetén segítséget nyújtva. Ezért nevezzük őket ebben a kutatásban felfedező tanulásos csoportnak (FTCS), bár a tanítási módszer nem tisztán felfedeztető, mivel a diákok számára a feladatok nem nyílt végűek és viszonylag rövidek. Viszont az teljesül, hogy a diákok a tanár által megtervezett feladatsorokon keresztül építik fel tudásukat. A második csoportban, az erősen irányított csoportban (EICS) igyekeztem minél több példafeladatot bemutatni, hogy a számonkérésekben előforduló feladattípusok széles skáláját mutassam meg.

A két csoport oktatásában az a döntő különbség, hogy az EICS-ben a tanár megpróbálja átadni a tudását diákoknak, a legjobb módszerekkel, mintaszerű megoldásokkal együtt, míg az FTCS-ben az alapvető tudást a diákok építik fel a fejükben, és az oktató felelőssége az, hogy ezt támogassa, rendszerezze és kiigazítsa. A két csoport diákjai intelligencia és előismeretek tekintetében hasonlóak voltak. Az egyetemre való felvételkor a legtehetségesebbeket külön gyakorlati csoportba tették, a többieket pedig a felvételi pontok alapján két félre vágták egy küszöbértéknél elválasztva, majd véletlenszerűen csoportokba osztották őket. A két csoport a hallgatók küszöbérték alatti feléből került ki, így feltehetően kevesebb programozási tapasztalattal rendelkeztek az átlagnál. A félév elején a hallgatókat megkértem, hogy töltsenek ki egy űrlapot a programozással kapcsolatos előképzettségükről és tapasztalataikról. A válaszok nem különböztek jelentősen a két csoport között.

7.2. Mérések és értékelések

A hallgatókat a félév során a következő számonkérésekkel értékeltük:

- Kódolási csoport zárthelyi (CsopZH Kód): 45 perces számonkérés a félév közepén, a gyakorlatokon szereplő példákhoz hasonló feladatot kell implementálni C++ nyelven, úgy, hogy adott a specifikáció és az algoritmus is struktogramon.
- Algoritmus csoport zárthelyi (CsopZH Alg): 45 perces dolgozat a félév vége felé, amiben egy rövid szöveggel és formális specifikációval megadott feladatra kell papíron, struktogrammal algoritmust készíteni.
- Kódolási évfolyam zárthelyi (ÉvfZH Kód): a félév végén a hallgatóknak 150 perc alatt kellett számítógépen megoldaniuk egy összetett, több részfeladatot tartalmazó feladatot, amelyet egy azonnali, teljes körű visszajelzést adó online kiértékelő automatikusan értékelt fekete doboz (input-output) teszteléssel.
- Algoritmus évfolyam zárthelyi (ÉvfZH Alg): a félév végén a hallgatók négy feladatot kaptak rövid szöveges leírással, és 90 perc alatt papíron kellett formális specifikációt írni rájuk és megoldási algoritmust adni struktogramon.
- Beadandók (B): a félév során a hallgatóknak házi feladatként négy egyszerű, 3 hetente adott feladatra kellett programokat írniuk és beadniuk egy online kiértékelő rendszerben.
- Komplex Beadandó (KB): a hallgatóknak a félév végén egy összetett feladatra készült programot dokumentációval együtt kellett benyújtaniuk.
- Szorgalmi házik (HF): minden héten egy-egy nagyobb kihívást jelentő feladatot adtam szorgalmi házi feladatként a diákoknak a Codeforces rendszerben [10]. A diákok ezekkel a feladatokkal csak pluszpontokat szerezhettek. Az ezekben elért eredményeik jól jelzik motivációjukat a programozás tanulás iránt.

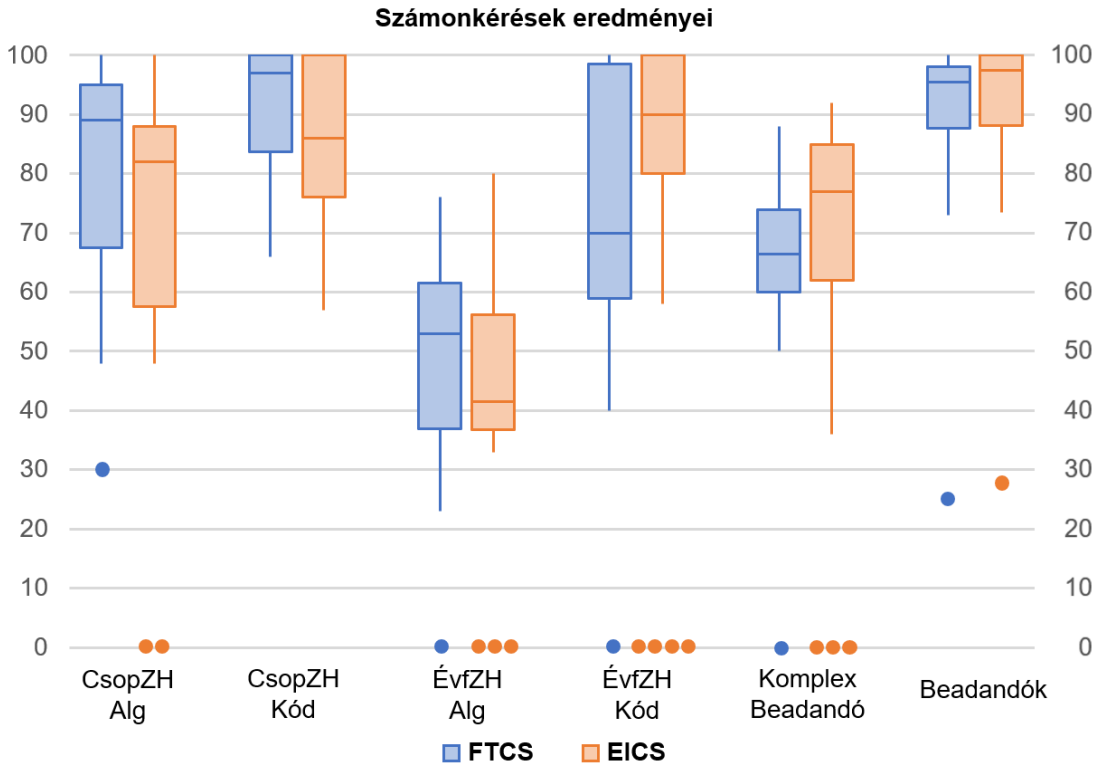
A kutatás központi kérdése az órák élvezete volt. Ezt egy egyszerű, rövid, 1-5-ig terjedő számszerű választ igénylő kérdéssel mértem minden egyes óra után. Továbbá ugyanezen a visszajelző űrlapon volt egy kérdés a fejlődés megítélésére vonatkozóan is. Egészen pontosan a következő kérdéseket tartalmazta a kérdőív:

- Mennyit tanultál/fejlődtél a mai órán egy 1-től 5-ig terjedő skálán? (1 - semmit, 5 - sokat)
- Mennyire élvezted a mai órát egy 1-től 5-ig terjedő skálán? (1 - unatkoztam, 5 - nagyon élveztem)

7.3. Eredmények, értékelések

7.3.1. Számonkérések eredménye

Az FTCS-ben 20, az EICS-ben pedig 17 diák volt. A két csoportban végzett számonkérések eredményeit az alábbi 7.1. ábra szemlélteti dobozdiagramok segítségével; az FTCS kékkel (bal oldal), az EICS narancssárgával (jobb oldal). A tanulók pontszámait százalékos arányban számoltam ki az egyes teszteknél. A dobozok középső vonala a csoport mediáneredményét jelöli, a felette és alatta lévő széles sáv az első és harmadik kvartilisig terjed, a vékony vonalak pedig a minimum és maximum értékekig húzódnak.



7.1. ábra: A számonkérések eredményei dobozdiagrammokkal ábrázolva.

A kiugró értékek pontokként jelennek meg. Ezek olyan hallgatóknak felelnek meg, akik a félév során feladták a kurzust, és nem, vagy csak részben vettek részt az adott számonkérésen. Érdemes megjegyezni egy szomorú tényt: az EICS-ben az ilyen hallgatók száma a félév során fokozatosan nőtt, egészen 4 hallgatóig, akik nem is jelentek meg a kódolási évfolyam zárthelyin. Ez erős jele a motivációvesztésnek, de nem feltétlenül emiatt a kurzus miatt. A hallgatók egy része nem sokkal a kezdés után otthagyja az egyetemet, mert rájön, hogy az informatika nem neki való. Egy másik tényező a karantén és a teljesen online oktatás volt, ami pszichés nehézségeket okozhatott.

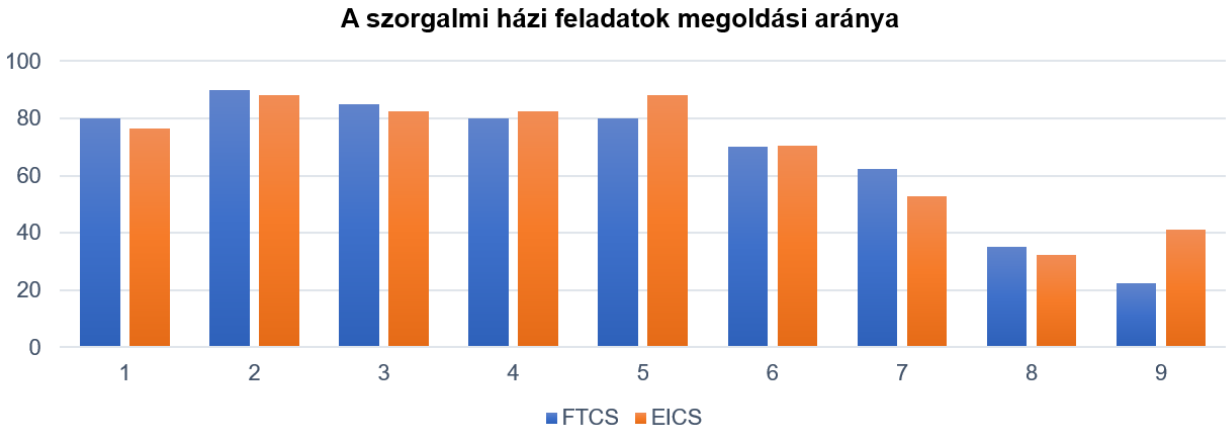
Amint a fenti ábrán látható, a két csoport hasonlóan teljesített az értékelések során. A félévközi kódolási és algoritmus csoport zárthelyik, valamint a félév végi algoritmus évfolyam zárthelyi az FTCS-ben valamivel jobban sikerült. A kódolási évfolyam zárthelyin azonban az EICS diákjai sokkal jobban teljesítettek, a beadandókban pedig valamivel jobban. Annak megállapításához, hogy ezek a különbségek szignifikánsak-e, p-értékeket számoltam ki kétmintás t-próbával, és ezek az alábbi táblázatban láthatók, a két csoport különböző számonkéréseink százalékos pontátlagaival együtt. Láthatjuk, hogy a különbség a kódolási évfolyam zárthelyinél szignifikáns ($p < 0,05$), a többi különbség nem szignifikáns, bár a többi zárthelyin (ÉvfZH Alg, CsopZH Alg és Kód) az FTCS valamivel jobb eredményeket produkált, míg a beadandókban (B és KB) az EICS valamivel jobb volt. Mi lehet az oka annak, hogy az EICS szignifikánsan jobban teljesített a kódolási évfolyam zárthelyin? Valószínűleg a bemutatott példafeladatok és mintamegoldások nagyobb változatossága segített, amikor a diákoknak szembe kellett nézniük a komplex kódolási feladattal. Ebben azonban egy további, prózai ok is szerepet játszhat. A tanulók ugyanis a számonkérések alapján egy összesített jegyet kapnak. Mivel a kódolási évfolyam zárthelyi az utolsó teszt, tudják, hogy hány pontra van szükségük a kívánt jegyhez, és lehet, hogy azonnal befejezik a zárthelyit, amint elegendő pontot kapnak az online értékelőrendszerben. Így, ha a diákok a korábbi számonkéréseken több pontot gyűjtöttek, az utolsó zárthelyin kevesebbre van szükségük.

	CsopZH Alg	**CsopZH Kód**	**ÉvfZH Alg**	**ÉvfZH Kód**	**KB**	**B**
FTCS átlag	81.316	91.900	51.053	75.009	67.605	91.421
EICS átlag	74.600	86.529	47.357	86.769	72.571	92.391
p-érték	0.135	0.089	0.239	**0.049**	0.144	0.384

7.1. táblázat: A számonkérések eredményeinek átlagai és p-értékei

A diákok motiváltságának nagyszerű mutatója a szorgalmi házik megoldásának száma. Ezek nem kötelező, nagyobb kihívást jelentő feladatok voltak, ahol a diákok egy kevés extra pontot gyűjthettek. A pontokban kifejezett jutalom tehát nem volt arányos a befektetett idővel, de ez is egy lehetőség volt arra, hogy a diákok a legjobb teljesítményüket adják, és próbára tegyék tudásuk határait. Számomra meglepő módon a két csoport eredményei szinte azonosak voltak, ahogyan az alábbi 7.2. ábrán látható. Az oszlopok magassága a helyes megoldásokat beküldő diákok százalékos arányának felel meg. A félév közepétől kezdve mindkét csoportban egyértelműen csökkenő tendencia figyelhető meg a megoldások számában. Ez azonban valószínűleg nem a motiváció csökkenése miatt van, hanem azért, mert a hallgatóknak a félév

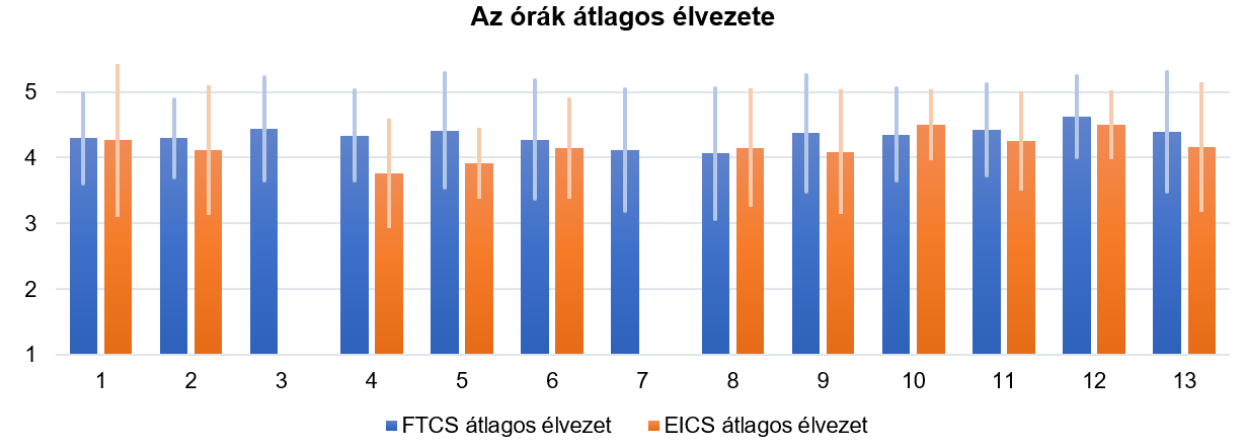
második felében a sok számonkérés miatt nagyon kevés szabadidejük van. Egy másik tényező a feladatok növekvő nehézsége. Érdekes lenne mérni, hogy melyik a döntő tényező - ennek vizsgálatához az egész félév során könnyű feladatokat adhatnánk ki. Az EICS-ben az utolsó extra feladatra sokkal több megoldás érkezett, ami a fent említett okra vezethető vissza: a félév végén több pontra volt szükségük.



7.2. ábra: A szorgalmi házi feladatok megoldási aránya az egyes hetekben

7.3.2. Visszajelzések

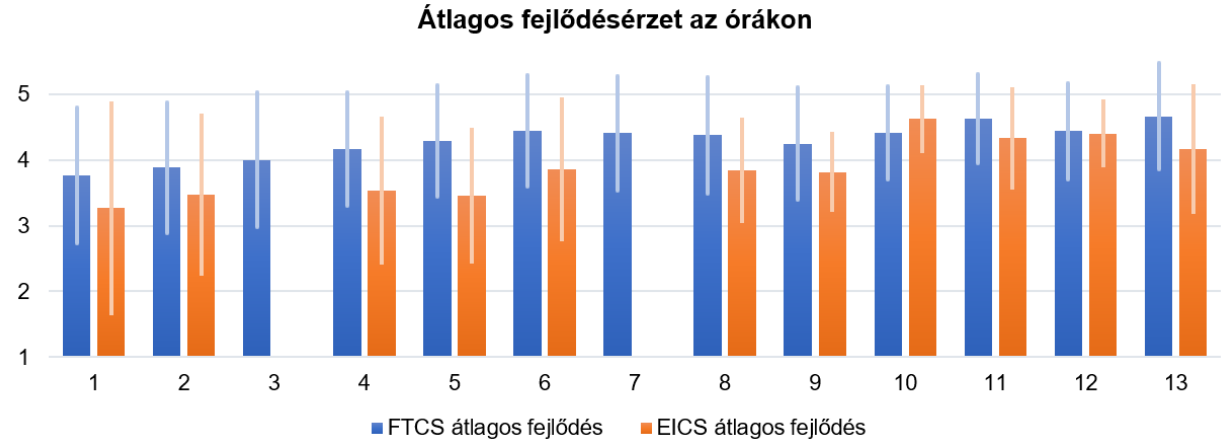
Az egyes órák végén kitöltött rövid visszajelző űrlapok központi szerepet töltöttek be ebben a kutatásban. Minden diákot arra kértem, hogy 1-5-ig értékelje, mennyire élvezte a gyakorlatot és mennyit fejlődött (érzése szerint). Az alábbi diagramokon az oszlopok magassága az egyes órák visszajelzéseinek csoportátlagait, míg a felső vonalszakaszok a számított szórásokat mutatják. Két olyan alkalom volt, amikor az EICS csoportnak nem volt órája a szünidő miatt, ezekben az esetekben az FTCS csoportban gyakorló órákat tartottunk.



7.4. ábra: A visszajelzésekben az élvezeti értékek átlaga és szórása az egyes hetekben

Az élvezeti értékelések általában elég magasak voltak, amint az a fenti 7.3. ábrán is látható. Az élvezeti értékek átlaga 9 alkalommal magasabb az FTCS-ben és 2 alkalommal alacsonyabb, mint az EICS-ben, de csak két alkalommal van szignifikáns különbség, a 4. és az 5. órán (a félév közepén). A félévnek ezen a pontján a legrosszabb visszajelzési eredmények az EICS-ben, a legjobb visszajelzési eredmények pedig az FTCS-ben fordultak elő. Ez az a pont, amikor a hallgatók már ismerik a programozási nyelv elemeit, és alapvető algoritmikus sémákat is (megszámolás, keresés, minimum/maximum kiválasztás). A mérések azt mutatják, hogy itt sokat jelent a diákoknak, ha először hagyjuk őket önállóan próbálkozni a feladatok megoldásával. Ha a példákat bemutatjuk, a téma talán egyáltalán nem tűnik vonzónak, mert könnyűnek tűnik. Másrészt ezek a feladatok azon a szinten vannak, ahol a diákok találhatnak egy kis kihívást és sikerélményt, és ebben az esetben sokkal kedvezőbb, ha nem mondjuk el nekik először a megoldásokat.

Egy másik érdekes megfigyelés, hogy az élvezeti szintek mindkét csoportban hasonlóan változnak az órák között (kivéve a korábban említett két alkalmat). Például mindkét csoportban kiválóan értékelték a 12. órát, amely a rendezési algoritmusokról, egy haladóbb témáról szólt. Az óra témája tehát mindkét tanítási módszer esetében hasonlóan befolyásolja az élvezeti értékeket.



7.4. ábra: A visszajelzésekben a fejlődési értékek átlaga és szórása az egyes hetekben

A fenti 7.4. ábra az átlagos fejlődési értékeket mutatja - a diákok mennyire érezték saját tudásuk és készségeik fejlődését. Az utolsó négy órán a két csoport eredményei közel állnak egymáshoz. Ennek oka, hogy ezeken az órákon összetettebb témákat és feladatokat tárgyaltunk. Az összes korábbi alkalommal (1-9) azonban az FTCS csoportban sokkal magasabbak a fejlődési értékek, mint az EICS csoportban. Ez egy izgalmas megállapítás. Úgy értelmezhetjük, hogy ha a téma elég egyszerű, a diákok úgy érzik, hogy jobban fejlődnek, ha

önállóan dolgoznak a megoldásokon, mintha sok mintamegoldást meghallgatnának. Erre két magyarázat is lehet. Először is lehetséges, hogy valóban többet fejlődnek. Másodszor, lehet, hogy a megoldásokat hallgató csoport nem veszi észre a feladatokban rejlő nehézségeket, mert a bemutatáskor egyszerűnek tűnik; így azt gondolják, hogy semmi újat nem tanulnak. Valószínűleg mindkét magyarázat egyszerre igaz. Mindazonáltal ez az eredmény váratlan volt ennél a mérésnél.

7.4. Konklúzió

Ebben a fejezetben egy rövid, kis léptékű vizsgálatot mutattam be, amelyben a problémamegoldáson alapuló, felfedeztető tanítás és az erősen irányított, sok kidolgozott példát bemutató módszer hatásait vizsgáltam. Egy féléven keresztül két programozási gyakorlati csoportot tanítottam a különböző technikákkal (FTCS és EICS). A két csoport teljesítménye a legtöbb számonkérésen hasonló volt. Az utolsó kódolási zárthelyin azonban jelentős különbség volt megfigyelhető; az EICS jobb eredményeket ért el, ami azzal magyarázható, hogy a hallgatók a feladatok és megoldások szélesebb választékát látták. Érdemes azonban megjegyezni, hogy az EICS-ben négy diák adta fel a kurzust, míg az FTCS-ben csak egy, ami befolyásoló tényező. A többi tesztben az FTCS diákjai valamivel jobban teljesítettek, de nem jelentősen.

A vizsgálat középpontjában az állt, hogy a diákok hogyan érzik magukat az órákon. A mért különbség az élvezetben nem volt olyan nagy, mint vártam. Csak néhány olyan óra volt, ahol a problémaalapú tanulás egyértelműen szórakoztatóbbnak tűnt, amikor néhány egyszerű algoritmikus sémát először alkalmaztak. Azonban a hallgatók fejlődése az érzésük szerint a félév első felében jelentősen jobb volt az FTCS-ben. Hasznosnak tűnik tehát az, ha a diákok úgy is megpróbálhatnak feladatokat megoldani, ha az alapjául szolgáló ismereteket korábban nem magyarázták el, feltéve, hogy azok könnyen hozzáférhetőek számukra. Meglepő eredmény, hogy a diákok úgy érezték, hogy többet fejlődtek, amikor lehetőségük volt arra, hogy maguk próbálkozzanak, mint amikor sok példa megoldását mondtam el nekik.

Van néhány fontos limitáció. A legkívánatosabb mérés a két tanítási módszer hosszú távú hatása lenne. Mekkora különbséget hoz a programozással való foglalkozás iránti motivációban? Megváltoztatja-e azt a módot, ahogyan az illető a szakmai pályafutása során egy-egy problémához közelít? Sajnos ezeket szinte lehetetlen mérni, mivel nagyon sok befolyásoló tényező van azon kívül, amire mi hatással vagyunk.

Ráadásul a jelenlegi mérések során nem tudtam feltárni más egyetemi tantárgyak hatásait, amelyek befolyásolhatják, hogy a hallgatók hogyan fejlődnek a félév során. Mégis érdekes lenne hasonló vizsgálatot végezni más csoportokkal vagy más szakterületen.

8. A tézisek alátámasztása

A doktori képzésben végzett kutatómunkámban kidolgoztam egy tehetséggondozó rendszert, amelyben felfedeztető módon lehet algoritmikus programozást tanítani. A téziseimet alátámasztják az előző fejezetekben bemutatott eredmények, az alábbiakban röviden összefoglalom miként.

I. Az algoritmikus programozás tanítása tehetséges diákoknak megvalósítható felfedeztető módszertannal, a matematika oktatásban sikeres Pósa-módszer alapvető szemléletének megtartásával, a módszertan kis módosításával. Megtarthatók a Pósa-módszernek olyan fontos elemei, mint például a nyitott kérdésfeltevés, a csoportmunka, az oktató segítő szerepe, és a feladatok egymásra épülése, összefonódása. A tanulási folyamat kiegészül egy kivitelezési lépéssel, ugyanakkor a csatlakozó kérdéseknek nehezebb teret adni. A számítógépes problémamegoldás és a matematika különbségeiből adódóan kevésbé jelenik meg a kutatás alapú tanulás, és erősebben a probléma alapú tanulás.

A 3. fejezetben elemeztem a Pósa-módszer didaktikai hátterét, és részletesen bemutattam, hogy a tézisben említett kisebb módosításokkal hogyan lehet használni a programozás tehetséggondozásban. A módszertant elkezdtem gyakorlatban alkalmazni az 5. fejezetben leírt tehetséggondozó táboraimban, az itt gyűjtött tapasztalataim is segítettek a szisztéma kialakításában. A módszer működőképességét igazolja a táborok sikeressége.

II. A programozási versenyeken szereplő feladatok alkalmasak arra, hogy problémaszálak hálózatát (web of problem threads) alkossunk belőlük, és ezzel megteremtsük a Pósa-módszer alkalmazásában központi szerepet játszó tananyagot a módszertan elméleti kereteinek megfelelő rendszerben. Ennek igazolására egy teljes tantervet hoztam létre a szükséges témák és kapcsolódási pontok azonosításával, valamint témakörönkénti feladatszálak megadásával. A tantervben 57 témakör és több, mint 600 feladat szerepel, és az irányított gráf formátumnak köszönhetően rugalmasan alakítható a tanítási sorrend.

A Pósa-módszerre jellemző tananyagtervezési keretrendszernek megfelelően létrehoztam a problémaszálak hálózatát programozási versenyfeladatok felhasználásával, amelyet a 4. fejezetben ismertettem. Részletesen prezentáltam a hozzá tartozó tantervet, amelyben a témák egy irányított gráf csúcsaiként jelennek meg. A benne szereplő 57 témakört a nemzetközi versenyekre előírt ismeretek, valamint versenyprogramozásról szóló tankönyvek és oktató weboldalak tartalma alapján gyűjtöttem össze, a saját tanítási tapasztalataimra is alapozva. A témák közötti egymásra épülési kapcsolatok ábrázolásával jól megtervezhető a témakörök

tanítási sorrendje, amely nem teljesen kötött, hanem az előfeltételek betartása mellett a csoporthoz alkalmazkodva választható. A témakörök gráfja adja a vázát a problémaszálak hálózatának, minden témakörhöz tartozik egy-egy szál. Szintén a 4. fejezetben néhány témakörre részletesen bemutattam a hozzájuk tartozó feladatszálakat egymásra épülő feladatokkal és a köztük lévő kapcsolódási pontokkal. A több, mint 600 feladatból álló teljes feladatgyűjteményt az 1. számú melléklet tartalmazza.

III. *A hétvégi programozástáborokban a felfedeztető módszertannal, csoportos problémamegoldással megvalósuló tanulás egyszerre nyújt élvezetet és fejlődést a tehetséges diákoknak. A táborokon átívelő tananyaggal és homogén életkorú csoportokkal a hosszú távú tanulás és a közösségépítés is megvalósítható.*

Az 5. fejezetben ismertettem az általam indított ProgTáborokat, amelyeket Pósa Lajos hétvégi matematikatáborainak inspirálásával hoztam létre. Az utóbbi öt évben négy csoportnak összesen 27 hétvégi tábort tartottam meg, amelyekről rendkívül pozitív visszajelzéseket kaptam, nagyon lelkesek a résztvevő diákok. Az 5.4. fejezetben leírt mérésekkel vizsgáltam azt a kérdést, hogy a felfedeztető tanítás során megvalósul-e együtt az a két cél, hogy a diákok fejlődjenek és közben élvezzék is a foglalkozásokat. A felmérések eredményei alátámasztják ezt a hipotézist.

A táborok különlegessége, hogy a diákok egy csoportja 4-5 éven keresztül együtt marad. Ez lehetővé teszi a hosszú távú építkezést, továbbá egy jó közösség kialakulását is elősegíti. Az 5.4.4. pontban láthattuk, hogy mennyire gyakran említik a táborok résztvevői pozitívumként a társaságot, tehát sikerült közösséget építeni a diákokból. A csoportok összeállításánál mindeddig az életkor volt a fő tényező, éppen a közösségépítés fontossága miatt. Azonban az utóbbi években egyre több tehetséggondozó szakkör indul algoritmikus programozásban, ami egyrészt roppant pozitív változás, másrészt a ProgTáborok szakmai anyagának tervezését megnehezíti azzal, hogy a táboros diákok csak egy része jár szakkörökre, ráadásul különböző szakkörökre, amelyek tananyagai átfedésben lehetnek a táborok anyagával. A jövőre nézve egy megoldandó kihívást jelent ez, amelyből adódóan elképzelhető, hogy változtatni kell a csoportösszeállítás módszerén, illetve a tananyagon.

IV. Online oktatásban is megvalósítható a kollaborációra és önálló munkára épülő felfedeztető tanítás a megfelelő szoftvereszközök és platformok használatával. Az online szervezés túlmutat a kényszermegoldáson, a hátrányok mellett számos előnye is van. A legfontosabb ezek közül az, hogy több diák számára válik elérhetővé ugyanaz a szakkör a földrajzi elhelyezkedéstől függetlenül, ezáltal homogénebb tudásszintű csoportok jöhetnek létre és a diákok a képességeiknek megfelelő oktatást kaphatják.

A 6. fejezetben bemutattam, hogy online oktatásban is meg lehet szervezni egy tábort, és online szakkörök keretében pedig rendszeres foglalkozásokkal, tudásszintben homogén csoportokkal a táborokhoz hasonló elveken alapuló felfedeztető módszerrel lehet tanítani algoritmikus programozást. Az online oktatás támogatására számos szoftveralkalmazást használok együttesen, ezeket a 6.2. fejezetben ismertettem. Míg egy online tábor inkább alkalmi kényszermegoldás, az online szakkörök esetében az előnyök felülmúlhatják a hátrányokat, ezt a 6.4. fejezetben részletezett felmérések eredményei igazolják.

V. Megfelelő nehézségű feladatok esetén a diákok több fejlődést tapasztalhatnak az önálló problémamegoldás útján, mint a tanár által vezetett, mintaszerű közös megoldás figyelemmel kísérése során, még akkor is, ha az utóbbi módszerrel több feladat feldolgozása történik meg ugyanannyi idő alatt.

A 7. fejezet egy egyetemi Programozás kurzus keretében végzett összehasonlító elemzésről szól, amelynek eredményeképpen adódott az, hogy önálló feladatmegoldás során fokozottabb fejlődést tapasztalhatnak a diákok, mint a tanár által sok példa szemléltetésével tartott foglalkozás során, de csak megfelelő nehézségű feladatok esetén számottevő a különbség. A ProgTáborokban összegyűjtött visszajelzések alapján (5.4. fejezet) ez az állítás további megerősítést nyert, hiszen a tehetséges diákok egyértelműen a kis csoportos feladatmegoldást részesítik előnyben a tapasztalt fejlődés szempontjából a megoldások közös megbeszélésével szemben.

9. Összefoglalás

Doktori kutatásomban egy innovatív tehetséggondozó rendszert dolgoztam ki algoritmikus programozásra, a matematika táborokban alkalmazott, nagyra értékelt Pósa-módszer adaptálásával. A gyakorlatban kipróbált és finomított rendszer hiánypótló szerepet tölt be a hazai programozás tehetséggondozásban: a tanulói elköteleződést, a mély megértést és a hosszú távú fejlődést helyezi előtérbe.

Megmutattam, hogy a Pósa-módszer alapelvei – nyitott kérdezés, csoportmunka, a tanár segítő szerepe, tudatos feladatsorozatok – csupán kis változtatások árán átültethetőek a programozás tehetséggondozó táborokra is. Lényeges módosításként beépítettem a kivitelezési fázist, és nagyobb hangsúlyt kapott a problémaközpontú tanulás, viszont jelenleg háttérbe szorul a kutatásalapú tanulás motívuma. A módszertan elméleti kereteinek megfelelő tantervet hoztam létre programozási versenyfeladatokra építve, amelynek 57 témaköre egy irányított gráfot alkot a témák között megadott előfeltételek révén. A több, mint 600 feladatból álló problémaszálak hálózata (web of problem threads) rugalmas tanítási utakat biztosít, és a felfedezéstől a szintetizálásig kíséri végig a diákokat. A 27 ProgTábor során gyűjtött visszajelzések igazolták, hogy a felfedezésen alapuló, együttműködő problémamegoldás egyszerre eredményez élményt és jelentős fejlődést. Az átgondolt tantervi felépítés és az életkor szerint homogén csoportok lehetővé tették a hosszútávú tanulási folyamatot és a közösségépítést, miközben a diákok elégedettsége következetesen magas maradt. Tapasztalataim és felméréseim alapján a felfedeztető programozástanítás online formában is hatékony. A pandémia kényszermegoldásából tartós előny lett: az online forma megszüntette a földrajzi korlátokat, lehetővé teszi homogén tudásszintű csoportok kialakítását, és szélesebb körű hozzáférést biztosít. Egy egyetemi oktatásban végzett kisléptékű összehasonlító vizsgálatom eredményei arra utaltak, hogy a megfelelő nehézségű önálló problémamegoldás nagyobb fejlődésélményt ad, mint a tanár által vezetett példamegoldás, különösen új algoritmikus fogalmak elsajátításának kezdeti szakaszában.

Összefoglalva, a kutatás egy gyakorlatban alkalmazható keretrendszert és tananyagot kínál az algoritmikus programozásban tehetséges diákok fejlesztésére. Ez az aktív felfedezésre, az együttműködésre és a problémaközpontú tantervi megközelítésre épül, és mind személyes, mind online környezetben meggyőző eredményeket mutatott.

Kulcsszavak: Programozás tehetséggondozás, Felfedeztető tanítás, Algoritmikus probléma-megoldás, ProgTábor kezdeményezés, Online programozás oktatás.

9.1. Summary

In my doctoral research, I developed an innovative talent education system in algorithmic programming, based on the adaptation of the acclaimed Pósa-method used in mathematics camps. Refined through practical application, the system addresses a significant gap in Hungary's programming education by emphasizing student engagement, deep understanding, and long-term development.

I showed that the Pósa-method's core principles – open-ended questioning, group work, the teacher's facilitative role, and carefully sequenced tasks – can be adapted to programming talent development camps with only minor modifications. Essential changes included the introduction of an implementation phase and a stronger focus on problem-based learning, while research-based learning currently plays a less significant role. Based on the theoretical framework of the methodology, I developed a curriculum built on programming contest problems, forming a directed graph of 57 topics connected through prerequisites. The network of more than 600 tasks – the "web of problem threads" – provides flexible teaching paths and guides students from discovery to synthesis of algorithmic concepts. Feedback from 27 ProgTábor camps indicated that discovery-based, collaborative problem-solving fosters both enjoyment and meaningful development. Structured curricula and age-homogeneous groups supported sustained learning and community building, with consistently high student satisfaction. My experience and surveys show that discovery-based programming education is also effective online. Beyond the pandemic necessity, online formats removed geographical barriers, enabled skill-homogeneous groupings, and broadened access. The results of a small-scale comparative study in university teaching suggested that appropriately challenging autonomous problem-solving provides a greater sense of development than teacher-led example solving, particularly in the initial stages of learning new algorithmic concepts.

In summary, this research provides a practical, adaptable framework and curriculum for developing students talented in algorithmic programming. It is grounded in active discovery, collaboration, and problem-centric curricular design, showing strong results in both in-person and online settings.

Keywords: Programming talent development, Discovery learning method, Algorithmic problem solving, ProgTábor initiative, Online programming education.

10. Irodalomjegyzék

Adedoyin, O. B., & Soykan, E. (2020). Covid-19 pandemic and online learning: the challenges and opportunities. *Interactive Learning Environments*, 1-13.

AI Arena. (2024). *AI Arena.* Forrás: GitHub: https://github.com/a-gondolkodas-orome/ai-arena-frontend

Bibergall, J. A. (1966). Learning by discovery: Its relation to science teaching. *Educational Review, Vol. 18(3)*, 222-231.

Bishop, A. P., Bertram, B. C., & Lunsford, K. J. (2004). Supporting Community Inquiry with Digital Resources. *Journal Of Digital Information, 5 (3)*.

Bóra, E. (2020). A computational thinking problem-thread for grade 7 students and above from the Pósa method. *Teaching Mathematics and Computer Science, 18(3)*, 101-110.

Bóra, E., & Juhász, P. (2023). How to Use Motion as a Problem-Solving Tool? Problems from the Pósa Camps. In *Problem Posing and Solving for Mathematically Gifted and Interested Students: Best Practices, Research and Enrichment* (old.: 125-146). Wiesbaden: Springer Fachmedien Wiesbaden.

Borthick, A. F., & Jones, D. R. (2000). The motivation for collaborative discovery learning online and its application in an information systems assurance course. *Issues in Accounting Education, Vol. 15(2)*, 181–210.

Bruner, J. S. (1961). The act of discovery. *Harvard Educational Review, 31*, 21–32.

Bruner, J. S. (1979). *On knowing: Essays for the left hand.* Harvard University Press.

Clark, A. E., Nong, H., Zhu, H., & Zhu, R. (2021). Compensating for academic loss: Online learning and student performance during the COVID-19 pandemic. *China Economic Review 68*.

Codeforces. (2024). *Blog.* Forrás: https://codeforces.com/top

Codeforces. (2024). *Problemset.* Forrás: https://codeforces.com/problemset

Competitive Programming Initiative. (2024). *USACO Guide*. Forrás: https://usaco.guide/

CP-Algorithms. (2024). *Algorithms for Competitive Programming*. Forrás: https://cp-algorithms.com/

Dewey, J. (1933). *How we think: A restatement of the relation of reflective thinking to the educative process.* Houghton Mifflin.

ELTE. (2024). *Programozási alapismeretek.* Forrás: ELTE: https://progalap.elte.hu/

Erdősné, N. Á. (2018). Grading Systems for Algorithmic Contests. *Olympiads in Informatics. Vol. 12*, 159–166.

Erdősné, N. Á. (2019). *A LOGO-tól az informatikai olimpiáig - Informatikai tehetséggondozás a középiskolában.* Budapest: Eötvös Loránd Tudományegyetem.

Erdősné, N. Á., & Zsakó, L. (2016). The Place of the Dynamic Programming Concept in the Progression of Contestants' Thinking. *Olympiads in Informatics. Vol. 10*, 61-72.

Finkle, S. L., & Torp, L. L. (1995). *Introductory Documents.* Illinois Math and Science Academy.

Forišek, M. (2015). Towards a better way to teach dynamic programming. *Olympiads in Informatics. Vol. 9*, 45-55.

GeeksforGeeks. (2024). *GeeksforGeeks*. Forrás: https://www.geeksforgeeks.org/

Győri, J. G., & Juhász, P. (2017). An extra-curricular gifted support programme in Hungary for exceptional students in mathematics. In *Teaching gifted learners in stem subjects* (old.: 89-106). Routledge.

Halim, S., Halim, F., & Effendy, S. (2018). *Competitive programming 4: The new lower bound of programming contests in the 2020s.* Lulu Independent Publish.

Juhász, P. (2019). Talent nurturing in Hungary: The Pósa weekend camps. *Notices of the American Mathematical Society, 66(6)*, 898-900.

Juhász, P., & Katona, D. (2019). Pósa method: Talent nurturing in weekend math camps. In *Including the Highly Gifted and Creative Students – Current Ideas and Fu-ture Directions: Proceedings of the 11th International Conference on Mathematical Creativity and Giftedness* (old.: 270-276). Münster: WTM.

Katona, D. (2021). Kernels of Mathematical Thinking as Task-and Curriculum Design Tool in the Pósa Method. In *Extended Abstracts Spring 2019: Advances in the Anthropological Theory of the Didactic* (old.: 141-148). Springer International Publishing.

Katona, D. (2022). *Introductory didactic analysis of the Pósa method: A study on the task design of an inquiry-based mathematics teaching practice primarily with tools of the Anthropological Theory of the Didactic.* Budapest: Eötvös Loránd University.

Katona, D., & Szűcs, G. (2017). Pósa-method & cubic geometry: A sample of a problem thread for discovery learning of mathematics. *Differences in pedagogical theory and practice*, 17-34.

Király, S. (2011). How to teach computer programming if our goal is the International Olympiad in Informatics. *Teaching Mathematics and Computer Science. Vol. 9, no. 1*, 13-25.

Kirschner, P. A., Sweller, J., & Clark, R. E. (2006). Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching. *Educational Psychologist, 41:2*, 75-86.

Kolb, D. A. (1984). *Experiential learning: Experience as the source of learning and development* . Englewood Cliffs, NJ: Prentice-Hall.

Kollár, K. N. (2021). Az online oktatás tapasztalatai és gyakorlata a pedagógusok nézőpontjából. *Iskolakultúra 31.2*, 23-53.

Laaksonen, A. (2017). *Competitive programmer's handbook.* Preprint 5.

Laaksonen, A., Salmi, R., & Talvitie, T. (2024). *Code Submission Evaluation System*. Forrás: https://cses.fi/

Ma, Z., Idris, D., Zhang, Y., Zewen, L., Wali, A. J., & Baloch, Z. (2021). The impact of COVID-19 pandemic outbreak on education and mental health of Chinese children aged 7–15 years: an online survey. *BMC pediatrics 21.1* , 1-8.

Masonbrink, A. R., & Hurley, E. (2020). Advocating for children during the COVID-19 school closures. *Pediatrics 146.3* .

Mosston, M. (1972). *Teaching: From command to discovery.* Wadsworth Publishing Company.

Nassi, I., & Shneiderman, B. (1973). Flowchart techniques for structured programming. *ACM Sigplan Notices, 8(8)*, 12-26.

Nikházy, L. (2019). A Nemes Tihamér Programozási verseny témaköreiről készült syllabus. In *INFODIDACT'2019* (old.: 199-208). Webdidaktika Alapítvány.

Nikházy, L. (2020). A Problem-based Curriculum for Algorithmic Programming. *Central-European Journal of New Technologies in Research, Education and Practice*, 76-96.

Nikházy, L. (2020). Az első ProgTábor: egy új tehetséggondozó program kezdete az algoritmikus programozásban. In *INFODIDACT'2020* (old.: 135-147). Webdidaktika Alapítvány.

NJSzT - ELTE IK. (2025). *Diákolimpiai válogatóverseny (EGOI, EJOI, CEOI, IOI).* Forrás: Tehetséggondozási Szakosztály: http://tehetseg.inf.elte.hu/valogatok/valogatok_main.html

Papert, S. (1996). An exploration in the space of mathematics educations. *International Journal of Computers for Mathematical Learning, 1(1)* , 95-123.

Piaget, J. (1971). *Psychology and Epistemology: Towards a Theory of Knowledge.* New York: Grossman.

Pósa, L. (2001). Matematika táboraim. *Természet Világa, 132. évfolyam, 3. szám*.

Reason, P., & Bradbury, H. (2001). *Handbook of Action Research: Participative Inquiry and Practice.* SAGE Publications.

Schmidt, H. G., Loyens, S. M., Van Gog, T., & Paas, F. (2007). Problem-Based Learning is Compatible with Human Cognitive Architecture: Commentary on Kirschner, Sweller, and Clark (2006). *Educational Psychologist, 42:2*, 91-97.

van Joolingen, W. (1999). Cognitive tools for discovery learning. *International Journal of Artificial Intelligence in Education, Vol. 10*, 385–397.

Verhoeff, T., Horváth, G., Diks, K., Cormack, G., Forišek , M., Peng, R., & Łącki , J. (2024). *IOI Syllabus.* Forrás: International Olympiad in Informatics: https://ioinformatics.org/files/ioi-syllabus-2024.pdf

Yang, X., Zhang, M., Kong, L., Wang, Q., & Hong, J. C. (2021). The effects of scientific self-efficacy and cognitive anxiety on science engagement with the question-observation-doing-explanation model during school disruption in COVID-19 pandemic. *Journal of Science Education and Technology 30.3* , 380-393.

Zsakó, L., & Szlávi, P. (2008). Módszeres programozás: Programozási tételek. In *Mikrológia 19*. ELTE IK.

11. Online feladatbankok, kiértékelő rendszerek és oktató weboldalak

- AtCoder, https://atcoder.jp/
- Code Submission Evaluation System (CSES), https://cses.fi/
- CodeChef, https://www.codechef.com/
- Codeforces, https://codeforces.com/
- Codeforces Polygon, https://polygon.codeforces.com/
- HackerRank, https://hackerrank.com/
- Mester, http://mester.inf.elte.hu/
- Mester feladatok GitHub repository-ja, https://github.com/asztrikx/mester-linux
- NJudge, https://njudge.hu/
- OJ.uz, https://oj.uz/
- USACO Guide, https://usaco.guide/
- USACO IDE, https://ide.usaco.guide/
- Snakify, https://snakify.org/
- Sphere Online Judge (SPOJ), https://www.spoj.com/
- Timus Online Judge, https://acm.timus.ru/
- UVa Online Judge, https://onlinejudge.org/
- VisuAlgo, https://visualgo.net/

12. A szerző disszertációhoz kapcsolódó publikációi

Nikházy, L. (2022) Online programozás szakkörök – a tehetséggondozás jövője? *In: Szlávi, Péter.; Zsakó, László (szerk.) INFODIDACT 2021.* Budapest: Webdidaktika Alapítvány. pp. 79-90.

Nikházy, L. (2021) A Comparison of Problem-Based Learning and Strongly Guided Instruction in Computer Programming Education. *In: Abonyi-Tóth, Andor; Stoffa, Veronika; Zsakó, László (szerk.) Proceedings of XXXIV. DidMatTech 2021 Conference: New Methods and Technologies in Education, Research and Practice.* Budapest: ELTE IK. pp. 5-14.

Nikházy, L., Noszály, Á., & Deák, B. (2021) Why You Should Know and Not Only Use Sorting Algorithms: Some Beautiful Problems. *OLYMPIADS IN INFORMATICS 15.* pp. 53-74.

Nikházy, L. (2020) Planets: A System for Autonomous Learning of Algorithmic Programming. *In: ISSEP 2020 : Proceedings of the International Conference on Informatics in School: Situation, Evaluation and Perspectives.* pp. 79-90.

Nikházy, L. (2020) Az első ProgTábor: egy új tehetséggondozó program kezdete az algoritmikus programozásban. *In: Zsakó, László; Szlávi, Péter (szerk.) INFODIDACT 2020.* Budapest: Webdidaktika Alapítvány. pp 135-147.

Nikházy, L., & Zsakó, L. (2020) National Programming Competitions,Team Selection and Training in Hungary. *OLYMPIADS IN INFORMATICS 14*. pp. 185-197.

Nikházy, L. (2020) A Problem-based Curriculum for Algorithmic Programming. *Central-European Journal Of New Technologies In Research Education And Practice 2:1.* pp. 76-96.

Nikházy, L. (2020) Algoritmusok tanítása problémaközpontú módszerrel. *In: Bihari, Erika; Molnár, Dániel; Szikszai-Németh, Ketrin Tavaszi Szél - Spring Wind 2019: Tanulmánykötet 2.* Budapest: Doktoranduszok Országos Szövetsége (DOSZ) pp. 554-567.

Nikházy, L. (2020) A Nemes Tihamér Programozási verseny témaköreiről készült syllabus. *In: Szlávi, Péter; Zsakó, László (szerk.) InfoDidact2019.* Zamárdi: Webdidaktika Alapítvány. 16.

Nikházy, L. (2019) The Joy of Thinking in Competitive Programming: A Curriculum Designed to Support Discovery Learning. *In: Stoffová, Veronika; Horváth, Roman (szerk.) XXXII DIDMATTECH 2019 – Proceedings : New Methods and Technologies in Education and Practice.* Trnava University. II/3.

13. Mellékletek

1. számú melléklet: Algoritmikus programozás feladatgyűjtemény, 135-149. oldal

Kód	Angol név	Magyar név	Leírás
ARRAY	Array	Tömb	Tömb használata elemek sorozatának tárolására.
SUM	Sum	Összegzés	Számsorozat összegének kiszámítása.
COUNT	Count	Megszámlálás	Egy sorozat bizonyos tulajdonsággal rendelkező elemeinek megszámlálása.
SEARCH	Search	Keresés	Egy adott tulajdonsággal rendelkező elem(ek) keresése egy sorozatban.
MIN SEL	Minimum Selection	Minimumkiválasztás	Egy sorozat legkisebb/legnagyobb értékének kiválasztása.
SORT	Sort	Rendezés	Sorozat rendezése egyszerű rendezési algoritmusokkal és a beépített rendezés (sort() függvény) használata.
NUM THEORY	Basic Number Theory	Egyszerű számelméleti algoritmusok	Egy egész szám osztóinak meghatározása, egyszerű prímteszt, Euklideszi algoritmus.
2D ARRAY	Two-dimensional array, Matrix	Két dimenziós tömb, Mátrix	Két dimenziós tömb használata táblázat, mátrix tárolására.
GREEDY	Greedy Algorithms	Mohó algoritmusok	Problémák megoldása mohó döntésekkel, annak felismerése, hogy ez az optimumhoz vezet-e.
PREFIX SUM	Prefix Sum	Prefix összegek	Egy sorozat prefix összegei, mint adatszerkezet (más néven kumulatív összegek).
2 POINT	Two Pointers Method	Két mutató technika	A két mutató technika egyes optimalizálási feladatok felgyorsítására.
REC	Recursion	Rekurzió	Problémamegoldás a rekurzió erejével, egyszerű rekurzív összefüggések.
PATH DP	DP for Best Path	Lépegetős DP	Bevezetés a dinamikus programozásba: útvonal optimalizálása mezők sorozatán vagy négyzetrácson.
GAME DP	DP for Simple Games	Játék DP	A győztes stratégia megtalálása egyszerű kétfős játékokban dinamikus programozással.
COMB	Combinatorics	Kombinatorika	Kombinatorikai alapfeladatok, mint például permutációk, kombinációk, Fibonacci-típusú sorozatok stb.
BIN SEARCH	Binary search	Bináris keresés	Bináris keresés egy rendezett sorozatban egy elem megtalálására, valamint optimalizálási feladatokban a szélsőérték megtalálására.
BITOP	Bitwise operations	Bitműveletek	Kettes számrendszerbeli alak és bitműveletek: és, vagy, xor, shiftelés, stb.
BRUTEFORCE	Brute force, exhaustive search	Kimerítő keresés	Az összes lehetőség kipróbálása, technikák az összes részhalmaz és összes permutáció vizsgálatára.
BACKTRACK	Backtrack	Visszalépéses keresés	Kimerítő kereséses megoldások gyorsítása visszalépéses kereséssel, 8 királynő probléma és rokon problémák.
KNAP DP	DP in Knapsack problem	Hátizsák DP	Néhány klasszikus DP probléma: hátizsák, pénzváltás és hasonlók.
TREES	Trees, Binary Trees	Fagráfok, bináris fák	Különböző helyzetekben megjelenő fa struktúra, pl. családfa, vállalati struktúra.
TREE REC	Recursion on trees	Fa rekurzió	Fákról szóló problémák megoldása rekurzió segítségével, hierarchikus struktúrát tartalmazó feladatok.
STACK	Stack	Verem	A verem (LIFO) adatszerkezet megértése és használata.
QUEUE	Queue, Deque	Sor, kétvégű sor	A sor (FIFO) és a kétvégű sor adatszerkezetek megértése és használata.
GRAPHS	Graphs	Gráfok	A gráfok fogalmi bevezetése a különböző problémák hátttereként.
BFS	Breadth-First Search	Szélességi bejárás	Szélességi bejárás alkalmazása a legrövidebb utak meghatározására és más problémák megoldására.
DFS	Depth First Search	Mélységi bejárás	A mélységi gráfbejárás rekurzív algoritmusa és alapvető alkalmazásai.
DAG	Directed Acyclic Graphs	Irányított körmentes gráf	DAG-ot tartalmazó problémák, mint például a topológikus rendezés, kritikus út keresés.
SHORT PATH	Shortest Path Algorithms	Legrövidebb utak	Legrövidebb utak keresése egy súlyozott gráfban. Bellman-Ford, Floyd-Warshall, Dijkstra algoritmusok.

Kód	Angol név	Magyar név	Leírás
HEAP	Heap, Priority Queue	Kupac adatszerkezet, prioritási sor	A kupac adatszerkezet és alkalmazása a prioritási sor megvalósítására, kupacrendezésre.
DSU	Disjoint Set Union / Union-find	Unió-holvan	Az unió-holvan adatszerkezet, és alkalmazásai különböző problémákban, pl. a Kruskal algoritmusban.
MST	Minimum Spanning Tree	Minimális feszítőfa	A minimális feszítőfa meghatározása súlyozott gráfokban. Kruskal és Prim algoritmusok.
RANGE DP	DP on ranges	DP intervallumokon	Dinamikus programozás intervallumokon, vagy kétváltozós részproblémákkal.
LCS DP	Longest Common Subsequence	Leghosszabb közös részsorozat	Két sorozat leghosszabb közös részsorozata és rokon problémák.
LIS DP	Longest Increasing Subsequence	Leghosszabb növekvő részsorozat	A leghosszabb növekvő részsorozat probléma úgy, hogy a dinamikus programozást bináris kereséssel felgyorsítjuk.
BIT DP	Bitmask DP, DP over subsets	Bitmaszk DP	Dinamikus programozás részhalmazokra, a halmazok kettes számrendszerbeli ábrázolásának felhasználásával.
SCC	Strongly Connected Components	Erősen összefüggő komponensek	Kosaraju és Tarjan algoritmusa és alkalmazásai.
ART POINT	Articulation Points, Bridges	Elvágó pontok, hídélek	Kétszeresen összefüggő gráfok, Tarjan algoritmusa és az L-érték az elvágó csúcsok és élek megtalálására.
EULER PATH	Eulerian Path	Euler-séta	Az Euler-út vagy -kör feltétele és megtalálása irányított és irányítatlan gráfokban.
MATCH	Maximal Matching	Maximális párosítás (páros gráfokban)	Magyar algoritmus a maximális párosításra páros gráfban.
GEOM	Geometry	Geometriai algoritmusok	Síkbeli geometriai problémák rácspontokon.
MOD INV	Modular Inverse, Fast exponentiation	Modulo inverz és gyorshatványozás	Hatékony algoritmus hatványozásra és a inverz számításra maradékosztályokon.
SIEVE	Sieve of Eratosthenes	Eratosztenészi szita	Prímek listájának meghatározása szita módszerrel.
ADV COMB	Advanced Combinatorics	Haladó kombinatorika	Különböző nehéz kombinatorikai problémák.
PERM	Permutations	Permutációk	Permutációkkal kapcsolatos feladatok, ciklusfelbontás.
INC-EXC	Inclusion-Exclusion Principle	Logikai szita	A logikai szita módszer alkalmazása kombinatorikai kérdések megválaszolására.
SEG TREE	Segment Tree	Szegmensfa	A statikusan kiegyensúlyozott bináris fák (szegmensfák) használata lekérdezések feldolgozására.
FENWICK	Binary Indexed / Fenwick Tree	Fenwick-fa	A Fenwick-fa adatszerkezet a szegmensfák alternatívájaként.
HASH	Hashing	Hashelés	Hashelés alkalmazása programozási versenyfeladatokban, hashtábla alapelve.
TRIE	Trie, Suffix Tree, Suffix Array	Trie	Adatszerkezetek szavak tárolására és keresésére: Trie, Suffix Tree, Suffix Array.
KMP, Z	Knuth-Morris-Pratt, Z-algorithm	Knuth-Morris-Pratt és Z-algoritmus	Fejlett string mintaillesztési algoritmusok: KMP, Z-algoritmus.
SQRT	SQRT Decomposition	Gyökös felbontás	A gyökös felbontás, mint problémamegoldó technika, Mo algoritmusa.
LCA	Lowest Common Ancestor	Legalacsonyabb közös ős	Egy fában két csúcs legalacsonyabb közös ősének megtalálása és alkalmazásai.
ST, RMQ	Sparse Table, Range Minimum Query	Sparse Table, Range Minimum Query	Sparse Table használata a Range Minimum Query probléma megoldására, valamint további alkalmazások.
FLOW	Network Flows	Hálózati folyamok	Feladatok modellezése hálózati folyamokkal, maximális folyam és minimális vágás meghatározása.
GAUSS	Gaussian Elimination	Gauss-elimináció	Lineáris egyenletrendszer megoldása.
2SAT	2-satisfiability problem	2-SAT probléma	A 2-SAT probléma megoldása gráfalgoritmusokkal, és alkalmazása más feladatokban.

Téma	Feladatcím	Feladatbank és azonosító / Szöveg	Szerep	Téma 2
ARRAY	10 szám oda-vissza	1. Olvassunk be 10 számot, majd a) írjuk ki őket változatlanul, b) írjuk ki őket fordított sorrendben, c) írjuk ki az összegüket!	bevezetés	
ARRAY	Fibonacci tömb	Olvass be egy N számot (N <= 20), majd tölts fel egy tömböt a Fibonacci-sorozat első N elemével! A Fibonacci sorozatban minden elem az előző kettő összege. F0=0, F1=1, F2=1, F3=2, F4=3, F5=5, F6=8, … Írd ki a tömb elemeit!	bevezetés	
ARRAY	Remove It	AtCoder ABC191_B	bevezetés	
ARRAY	Permutation Check	AtCoder ABC205_B	szintézis	
SUM	FizzBuzz Sum	AtCoder ABC162_B	bevezetés	
SUM	Can you buy them all?	AtCoder ABC209_B	bevezetés	
SUM	Missing Number	CSES 1083	elmélyítés	
SUM	K-divisible Sum	Codeforces 1476A	elmélyítés	CONSTRUCT
SUM	Log Chopping	Codeforces 1672A	szintézis	GAME
COUNT	Roller Coaster	AtCoder ABC142_B	bevezetés	
COUNT	Vanya and Fence	Codeforces 677A	bevezetés	
COUNT	Olcsó és egyben nagy lakások száma	Mester Kezdő / Programozási tételek: megszámolás / 28.	bevezetés	
COUNT	Ugró értékek	Mester Kezdő / Programozási tételek: megszámolás / 45.	elmélyítés	
COUNT	Cookies	Codeforces 129A	elmélyítés	
SEARCH	A nap mikor nem volt kapás	Mester Kezdő / Programozási tételek: keresés / 1.	bevezetés	
SEARCH	Azonos pont egymás mellett	Mester Kezdő / Programozási tételek: keresés / 3.	bevezetés	
SEARCH	Yet Another Dividing into Teams	Codeforces 1249A	elmélyítés	CONSTRUCT
SEARCH	Colourblindness	Codeforces 1722B	szintézis	STRING
MIN SEL	Leghidegebb nap	Mester Kezdő / Programozási tételek: minimum, maximum számítás / 6.	bevezetés	
MIN SEL	Legidősebb dolgozó	Mester Kezdő / Programozási tételek: minimum, maximum számítás / 8.	bevezetés	
MIN SEL	Booby Prize	AtCoder ABC213_B	elmélyítés	
MIN SEL	Choose Two Numbers	Codeforces 1206A	elmélyítés	CONSTRUCT
MIN SEL	Wet Shark and Odd and Even	Codeforces 621A	szintézis	COUNT
MIN SEL	Holiday Of Equality	Codeforces 758A	szintézis	SUM
SORT	Hallgató felező	Mester Kezdő / Programozási tételek: rendezések / 17.	bevezetés	
SORT	Petya and Staircases	Codeforces 362B	bevezetés	
SORT	Twins	Codeforces 160A	bevezetés	GREEDY
SORT	z-sort	Codeforces 652B	elmélyítés	
SORT	Ehab Fails to Be Thanos	Codeforces 1174A	elmélyítés	CONSTRUCT
SORT	Restaurant Customers	CSES 1619	szintézis	PREFSUM
NUM THEORY	Playing with Paper	Codeforces 527A	bevezetés	
NUM THEORY	Domino piling	Codeforces 50A	bevezetés	
NUM THEORY	Div. 7	Codeforces 1633A	elmélyítés	
NUM THEORY	Wrong Subtraction	Codeforces 977A	elmélyítés	
NUM THEORY	Balanced Rating Changes	Codeforces 1237A	elmélyítés	CONSTRUCT
NUM THEORY	Equation	Codeforces 1269A	elmélyítés	CONSTRUCT
NUM THEORY	Kastély	Mester NT, OKTV, IOI Válogató / Nemes Tihamér 1. 2017/18 3. forduló / 1.	elmélyítés	
NUM THEORY	Bad Ugly Numbers	Codeforces 1326A	szintézis	CONSTRUCT
NUM THEORY	Rectangular Game	Codeforces 177B2	szintézis	GAME
NUM THEORY	EhAb AnD gCd	Codeforces 1325A	szintézis	CONSTRUCT
NUM THEORY	GCD Problem	Codeforces 1617B	szintézis	CONSTRUCT
2D ARRAY	Maximális egyedszámú helységek	Mester Középhaladó / Madármegfigyelés / 18.	bevezetés	MIN SEL
2D ARRAY	Mindenhol előforduló madárfajok	Mester Középhaladó / Madármegfigyelés / 20.	bevezetés	SEARCH
2D ARRAY	Ritka madárfaj	Mester Középhaladó / Madármegfigyelés / 23.	bevezetés	SEARCH
2D ARRAY	Dinner with Emma	Codeforces 616B	szintézis	MIN SEL
2D ARRAY	Colour the Flag	Codeforces 1534A	szintézis	
GREEDY	Movie Festival	CSES 1629	bevezetés	

Téma	Feladatcím	Feladatbank és azonosító / Szöveg	Szerep	Téma 2
GREEDY	Alvállalkozó	Mester Haladó / Mohó algoritmusok / 51.	bevezetés	
GREEDY	Fénykép	Mester Haladó / Mohó algoritmusok / 9.	bevezetés	
GREEDY	Mekk Elek munkái	Mester Haladó / Mohó algoritmusok / 28.	bevezetés	
GREEDY	Olimpiai staféta	Mester Haladó / Mohó algoritmusok / 34.	bevezetés	
GREEDY	Shovels and Swords	Codeforces 1366A	bevezetés	
GREEDY	C+=	Codeforces 1368A	bevezetés	
GREEDY	Assigning to Classes	Codeforces 1300B	elmélyítés	
GREEDY	Avoid Trygub	Codeforces 1450A	elmélyítés	CONSTRUCT
GREEDY	Basketball Together	Codeforces 1725B	elmélyítés	
GREEDY	Case of the Zeros and Ones	Codeforces 556A	elmélyítés	
GREEDY	Corners	Codeforces 1720C	elmélyítés	
GREEDY	Display The Number	Codeforces 1295A	elmélyítés	
GREEDY	Domino for Young	Codeforces 1268B	elmélyítés	
GREEDY	Get an Even String	Codeforces 1660C	elmélyítés	
GREEDY	Given Length and Sum of Digits...	Codeforces 489C	elmélyítés	
GREEDY	Hit the Lottery	Codeforces 996A	elmélyítés	
GREEDY	Ilya and a Colorful Walk	Codeforces 1119A	elmélyítés	
GREEDY	New Year Parties	Codeforces 1283E	elmélyítés	
GREEDY	Omkar and Password	Codeforces 1392A	elmélyítés	
GREEDY	Phoenix and Balance	Codeforces 1348A	elmélyítés	
GREEDY	Skyscrapers (easy version)	Codeforces 1313C1	elmélyítés	DP
GREEDY	Social Distance	Codeforces 1668B	elmélyítés	
GREEDY	Traps	Codeforces 1684D	elmélyítés	
GREEDY	Twins	Codeforces 160A	elmélyítés	
GREEDY	Unstable String Sort	Codeforces 1213F	elmélyítés	
GREEDY	Virus	Codeforces 1704C	elmélyítés	
GREEDY	Yaroslav and Permutations	Codeforces 296A	elmélyítés	
GREEDY	Collecting Numbers	CSES 2216	elmélyítés	
GREEDY	Movie Festival II	CSES 1632	elmélyítés	SET / PRQUEUE
GREEDY	Tasks and Deadlines	CSES 1630	elmélyítés	
GREEDY	Removing Digits	CSES 1637	elmélyítés	DP
GREEDY	Even-Odd Game	Codeforces 1472D	szintézis	GAME
GREEDY	Knapsack	Codeforces 1446A	szintézis	CONSTRUCT
GREEDY	Pekora and Trampoline	Codeforces 1491C	szintézis	BRUTEFORCE
GREEDY	Potions (Hard Version)	Codeforces 1526C2	szintézis	DP
GREEDY	Powered Addition	Codeforces 1338A	szintézis	BITOP
GREEDY	Sequential Nim	Codeforces 1382B	szintézis	GAME DP
GREEDY	Shaass and Bookshelf	Codeforces 294B	szintézis	BRUTEFORCE
GREEDY	Skyscrapers (hard version)	Codeforces 1313C2	szintézis	STACK
GREEDY	Travelling Salesman Problem	Codeforces 1503C	szintézis	CONSTRUCT
GREEDY	Wonderful Randomized Sum	Codeforces 33C	szintézis	PREFSUM
GREEDY	Colors and Intervals	Codeforces 1552E	szintézis	CONSTRUCT
GREEDY	Decreasing Debts	Codeforces 1266D	szintézis	CONSTRUCT
GREEDY	Defender of Childhood Dreams	Codeforces 1586F	szintézis	CONSTRUCT
GREEDY	Interesting Subarray	Codeforces 1270B	szintézis	CONSTRUCT
GREEDY	Lefedés	Mester Haladó / Mohó algoritmusok / 23.	lekötő	
GREEDY	Múzeum	Mester Haladó / Mohó algoritmusok / 58.	lekötő	
GREEDY	Segély	Mester NT, OKTV, IOI Válogató / IOI / CEOI Válogató 2019 / 10.	lekötő	PREFSUM
PREFIX SUM	Static Range Sum Queries	CSES 1646	alap	
PREFIX SUM	Ilya and Queries	Codeforces 313B	bevezetés	
PREFIX SUM	Maximum Subarray Sum	CSES 1643	szintézis	GREEDY
PREFIX SUM	Fence	Codeforces 363B	bevezetés	

Téma	Feladatcím	Feladatbank és azonosító / Szöveg	Szerep	Téma 2
PREFIX SUM	Kuriyama Mirai's Stones	Codeforces 433B	bevezetés	
PREFIX SUM	Greg and Array	Codeforces 296C	elmélyítés	
PREFIX SUM	Kihalt állatok	Mester Kezdő / Programozási tételek: minimum, maximum számítás / 50.	elmélyítés	
PREFIX SUM	Karen and Coffee	Codeforces 816B	elmélyítés	
PREFIX SUM	Prefix Sum Addicts	Codeforces 1738B	elmélyítés	
PREFIX SUM	Promo	Codeforces 1697B	elmélyítés	
PREFIX SUM	Consecutive Subsequences	HackerRank consecutive-subsequences	elmélyítés	
PREFIX SUM	Forest Queries	CSES 1652	alap	
PREFIX SUM	Fenyőfa	Mester Haladó / Egyéb feladatok / 9.	elmélyítés	
PREFIX SUM	Legnagyobb hasznú téglalap	Mester Haladó / Egyéb feladatok / 31.	elmélyítés	
PREFIX SUM	Sivatag	Mester Kezdő / Programozási tételek: összeépítése / 134.	lekötő	
PREFIX SUM	Candies	AtCoder DP_M	lekötő	DP
PREFIX SUM	Good Subarrays	Codeforces 1398C	szintézis	MAP
PREFIX SUM	Money Transfers	Codeforces 675C	szintézis	MAP
PREFIX SUM	Telepanting	Codeforces 1552F	szintézis	BIN SEARCH
2 POINT	Balanced Team	Codeforces 1133C	bevezetés	SORT
2 POINT	Books	Codeforces 279B	bevezetés	
2 POINT	Subarray Sums I	CSES 1660	bevezetés	
2 POINT	Sum of Two Values	CSES 1640	bevezetés	MAP
2 POINT	Hard Process	Codeforces 660C	elmélyítés	
2 POINT	Kefa and Company	Codeforces 580B	elmélyítés	BIN SEARCH, PREFIX SUM
2 POINT	Points on Line	Codeforces 251A	szintézis	COMB
2 POINT	Longest k-Good Segment	Codeforces 616D	elmélyítés	
2 POINT	Slime Escape	Codeforces 1734D	elmélyítés	
2 POINT	Three Parts of the Array	Codeforces 1006C	elmélyítés	
2 POINT	Vasya and String	Codeforces 676C	elmélyítés	
2 POINT	Sum of Three Values	CSES 1641	elmélyítés	MAP, BRUTEFORCE
2 POINT	Nyaralás	Mester NT, OKTV, IOI Válogató / Nemes Tihamér 2. 2018/19 3. forduló / 6.	elmélyítés	
REC	Számjegyek összege	Írj egy minél szebb függvényt, amely egy paraméterként kapott pozitív egész szám számjegyeinek az összegét adja meg.	bevezetés	
REC	Hatványozás rekurzívan	Írj egy minél rövidebb, **rekurzív** függvényt n^k kiszámításására: `int hatvany(int n, int k)` Az n^k (n a k-adikon, más szóval n k-adik hatványa) annak a k elemű szorzatnak az értéke, amelynek minden tényezője n. Például 5^3=5*5*5=125.	bevezetés	
REC	Pascal háromszög	Írj egy rekurzív (önmagát hívó) függvényt, ami a Pascal háromszög n. sorának k. elemét adja meg: `int pascal(int n, int k)`	elmélyítés	
REC	String megfordítása rekurzívan	Codeforces Polygon reverse-string-rec	szintézis	STRING
REC	Tower of Hanoi	CSES 2165	lekötő	
PATH DP	Frog 1	AtCoder DP_A	bevezetés	
PATH DP	Frog 2	AtCoder DP_B	bevezetés	
PATH DP	Kincsek a hegyoldalon	Mester Haladó / Dinamikus programozás / 107.	bevezetés	
PATH DP	Pontgyűjtő verseny	Mester Haladó / Dinamikus programozás / 116.	bevezetés	
PATH DP	Táblás játék legkevesebb büntető mezővel	Mester Haladó / Dinamikus programozás / 89.	bevezetés	
PATH DP	Long Jumps	Codeforces 1472C	elmélyítés	
PATH DP	Working out	Codeforces 429B	elmélyítés	
PATH DP	Dobókocka	Codeforces Polygon dobokocka	szintézis	COMB
PATH DP	Grid Paths	CSES 1638	szintézis	COMB
PATH DP	Csapdák kikerülése	Mester Haladó / Dinamikus programozás / 9.	szintézis	COMB
PATH DP	The least round way	Codeforces 2B	szintézis	NUM THEO
PATH DP	Vacation	AtCoder DP_C	elmélyítés	

Téma	Feladatcím	Feladatbank és azonosító / Szöveg	Szerep	Téma 2
PATH DP	Relay Race	Codeforces 213C	elmélyítés	
PATH DP	Vacations	Codeforces 698A	elmélyítés	
PATH DP	Pig and Palindromes	Codeforces 570E	lekötő	
GAME DP	Stones	AtCoder DP_K	bevezetés	
GAME DP	Game of Stones	HackerRank game-of-stones-1	bevezetés	
GAME DP	A Chessboard Game	HackerRank a-chessboard-game-1	bevezetés	
GAME DP	1-2-K Game	Codeforces 1194D	elmélyítés	
GAME DP	Pie Rules	Codeforces 859C	elmélyítés	
GAME DP	Permutation Game	Codeforces 1033C	elmélyítés	
GAME DP	Boredom	Codeforces 455A	elmélyítés	
GAME DP	Removal Game	CSES 1097	elmélyítés	
GAME DP	Conan and Agasa play a Card Game	Codeforces 914B	szintézis	GREEDY
GAME DP	Berzerk	Codeforces 786A	szintézis	DFS, BFS
GAME DP	Funny Game	Codeforces 731E	szintézis	PREFIX SUM
GAME DP	Ádám és Éva megint játszik	Mester Haladó / Dinamikus programozás / 3.	szintézis	GRAPH
COMB	Lépcsők	Mester Haladó / Kombinatorikai algoritmusok / 11.	bevezetés	PATH DP
COMB	Merge List	HackerRank merge-list	bevezetés	DP
COMB	Coin Combinations I	CSES 1635	elmélyítés	KNAP DP
COMB	Épület színezések száma 3.	Mester Haladó / Kombinatorikai algoritmusok / 26.	bevezetés	DP
COMB	Hibás lépcsők	Mester Haladó / Kombinatorikai algoritmusok / 8.	elmélyítés	PATH DP
COMB	Shaass and Lights	Codeforces 294C	elmélyítés	
COMB	WOW Factor	Codeforces 1178B	elmélyítés	
COMB	Flowers	Codeforces 474D	szintézis	PREFIX SUM
COMB	The Number of Products	Codeforces 1215B	szintézis	PREFIX SUM
COMB	Number of Ways	CodeChef NWAYS	elmélyítés	
COMB	Karen and Test	Codeforces 816D	elmélyítés	
COMB	Kyoya and Colored Balls	Codeforces 554C	elmélyítés	DP
COMB	Independent Set	AtCoder DP_P	szintézis	DFS
COMB	Ayoub and Lost Array	Codeforces 1105C	szintézis	PATH DP
COMB	Caesar's Legions	Codeforces 118D	szintézis	DP
COMB	Colorful Bricks	Codeforces 1081C	szintézis	DP
COMB	How many trees?	Codeforces 9D	szintézis	DP
COMB	Mashmokh and ACM	Codeforces 414B	szintézis	DP
COMB	Zigzags	Codeforces 1400D	szintézis	PREFIX SUM
COMB	Lego Blocks	HackerRank lego-blocks	lekötő	DP
COMB	Coloring Brackets	Codeforces 149D	lekötő	DP, REC
BIN SEARCH	Barkochba	Írj barkochba programot, amely minél kevesebb lépéssel talál ki egy 1 és 100 közötti számot, amelyre a felhasználó gondol! Csak eldöntendő kérdéseket tehet fel a programod. Általánosítsd úgy, hogy a 100 helyére bármilyen N ≤ 10^9 számot lehessen megadni felső határnak!	bevezetés	
BIN SEARCH	Szótár	Írj programot egy szótárban való keresésre! A szótárban szereplő N darab szó ábécé sorrendben adott. Ezt követően K szó mindegyikére ki kell írnod, hogy hányadik pozícióban szerepel a szótárban! (Minden kérdéses szó benne van a szótárban. 1 ≤ K, N ≤ 10^5)	bevezetés	
BIN SEARCH	Interesting drink	Codeforces 706B	bevezetés	
BIN SEARCH	Frodo and pillows	Codeforces 760B	elmélyítés	
BIN SEARCH	Magic Powder - 2	Codeforces 670D2	elmélyítés	
BIN SEARCH	Hamburgers	Codeforces 371C	elmélyítés	
BIN SEARCH	Aggressive Cows	SPOJ AGGRCOW	bevezetés	
BIN SEARCH	Burning Midnight Oil	Codeforces 165B	elmélyítés	
BIN SEARCH	Worms	Codeforces 474B	szintézis	PREFSUM
BIN SEARCH	Sűrű részsorozat	Mester NT, OKTV, IOI Válogató / IOI/CEOI Válogató 2018 / 5.	elmélyítés	
BIN SEARCH	The Delivery Dilemma	Codeforces 1443C	elmélyítés	

Téma	Feladatcím	Feladatbank és azonosító / Szöveg	Szerep	Téma 2
BIN SEARCH	Salary Changing	Codeforces 1251D	elmélyítés	
BIN SEARCH	Max Median	Codeforces 1486D	elmélyítés	
BIN SEARCH	Csatornák	Mester NT, OKTV, IOI Válogató / OKTV 2017/2018 3. forduló / 1.	szintézis	BFS
BIN SEARCH	Quality of Living	OJ.UZ IOI10_quality	szintézis	PREFIX SUM
BIN SEARCH	Characteristics of Rectangles	Codeforces 333D	szintézis	DP
BIN SEARCH	Minimizing Difference	Codeforces 1244E	szintézis	GREEDY
BIN SEARCH	Long Way to be Non-decreasing	Codeforces 1972F	szintézis	DFS
BIN SEARCH	Minimax Problem	Codeforces 1288D	szintézis	BITMASK
BIN SEARCH	Magnets	Codeforces 1491F	szintézis	CONSTRUCT
BIN SEARCH	List Of Integers	Codeforces 920G	szintézis	INC-EXC
BITOP	Bitműveletek	Csináld meg minél kevesebb utasítással: a) Kapcsold be/ki/át az i-edik bitet b) Kérdezd le az i-edik bitet c) Csak az utolsó k bitet hagyd meg d) Az utolsó k bit legyen 0 e) A számban az utolsó 1-es számjegy helyett legyen 0 f) A számban az utolsó 0 számjegy helyett legyen 1-es g) A számvégi 1-es sorozat helyett legyenek 0-k h) A számvégi 0 sorozat helyett legyenek 1-esek i) 2 hatvány-e ? j) Csak az utolsó 1-est hagyd meg, minden más legyen 0	alap	
BITOP	Raising Bacteria	Codeforces 579A	bevezetés	
BITOP	XORwice	Codeforces 1421A	bevezetés	
BITOP	XOR Mixup	Codeforces 1698A	bevezetés	
BITOP	Little Xor	Codeforces 252A	elmélyítés	
BITOP	p-binary	Codeforces 1225C	elmélyítés	
BITOP	Mark and the Online Exam	Codeforces 1705F	elmélyítés	CONSTRUCT
BITOP	Make Good	Codeforces 1270C	szintézis	CONSTRUCT
BITOP	Bookshelves	Codeforces 981D	szintézis	DP
BITOP	AND Graph	Codeforces 986C	szintézis	DFS
BITOP	AmShZ and G.O.A.T.	Codeforces 1610E	elmélyítés	
BITOP	Xor	Mester NT, OKTV, IOI Válogató / IOI/CEOI Válogató 2020 / 5.	lekötő	GREEDY
BRUTEFORCE	Apple Division	CSES 1623	bevezetés	BIT OP
BRUTEFORCE	Creating Strings	CSES 1622	bevezetés	PERM
BRUTEFORCE	Magic Powder - 1	Codeforces 670D1	bevezetés	
BRUTEFORCE	Matrix reducing	AtCoder ABC264_C	elmélyítés	BIT OP
BRUTEFORCE	PFAST Inc.	Codeforces 114B	elmélyítés	BIT OP
BRUTEFORCE	Preparing Olympiad	Codeforces 550B	elmélyítés	BIT OP
BRUTEFORCE	Smallest number	Codeforces 55B	elmélyítés	
BRUTEFORCE	Three Logos	Codeforces 581D	szintézis	PERM, BIT OP
BRUTEFORCE	Lámpák	Mester NT, OKTV, IOI Válogató / IOI/CEOI Válogató 2019 / 4.	lekötő	BITOP
BACKTRACK	N Bástya	Mester Haladó / Visszalépéses keresés / 23.	bevezetés	
BACKTRACK	Chessboard and Queens	CSES 1624	bevezetés	
BACKTRACK	Iskolaválasztás	Mester Haladó / Visszalépéses keresés / 17.	elmélyítés	BRUTEFORCE
BACKTRACK	Szókereső	Mester NT, OKTV, IOI Válogató / Nemes Tihamér 2. 2017/18 3. forduló / 2.	elmélyítés	
KNAP DP	Nem kifizethető címletek	Mester Haladó / Dinamikus programozás / 69.	bevezetés	
KNAP DP	Minimizing Coins	CSES 1634	alap	
KNAP DP	Money Sums	CSES 1745	alap	
KNAP DP	Hátizsák probléma	Mester Haladó / Dinamikus programozás / 100.	alap	
KNAP DP	Knapsack	AtCoder DP_D	alap	
KNAP DP	Book Shop	CSES 1158	alap	
KNAP DP	Postabélyegek	Mester Haladó / Dinamikus programozás / 73.	bevezetés	
KNAP DP	Missing Coin Sum	CSES 2183	elmélyítés	
KNAP DP	The Coin Change Problem	HackerRank coin-change	elmélyítés	COMB

Téma	Feladatcím	Feladatbank és azonosító / Szöveg	Szerep	Téma 2
KNAP DP	Baby Ehab Partitions Again	Codeforces 1516C	elmélyítés	
KNAP DP	Házimunka	Codeforces Polygon munkagepek	elmélyítés	
KNAP DP	Malacpersely legkisebb értékre	Mester Haladó / Dinamikus programozás / 61.	elmélyítés	
KNAP DP	Kötélhúzás	Mester Haladó / Dinamikus programozás / 51.	elmélyítés	
KNAP DP	Vásár	Mester NT, OKTV, IOI Válogató / OKTV 2018/19 2. forduló / 3.	elmélyítés	
KNAP DP	Yet Another Minimization Problem	Codeforces 1637D	elmélyítés	
KNAP DP	Checkout Assistant	Codeforces 19B	elmélyítés	
KNAP DP	The Values You Can Make	Codeforces 687C	elmélyítés	
KNAP DP	Phoenix and Berries	Codeforces 1348E	lekötő	
KNAP DP	Turtle	Codeforces 1239E	lekötő	
KNAP DP	Igazságos osztozkodás	Mester Haladó / Dinamikus programozás / 32.	lekötő	
KNAP DP	Cloud Computing	OJ.uz CEOI18_clo	lekötő	
TREES	Ember	Mester Haladó / Gráfok, elemi feladatok / 7. Ember	bevezetés	
TREES	Titkos Társaság 1.	Mester Haladó / Rekurzív adatszerkezetek / 37.	bevezetés	
TREES	Christmas Spruce	Codeforces 913B	elmélyítés	
TREES	Jelentés	Mester Haladó / Rekurzív adatszerkezetek / 22.	elmélyítés	
TREE REC	Subordinates	CSES 1674	bevezetés	
TREE REC	Party	Codeforces 115A	bevezetés	
TREE REC	Xor-tree	Codeforces 429A	szintézis	GREEDY
TREE REC	Anton and Tree	Codeforces 734E	elmélyítés	
STACK	Nearest Smaller Values	CSES 1645	alap	
STACK	Largest Rectangle	HackerRank largest-rectangle	elmélyítés	
STACK	Minimal string	Codeforces 797C	szintézis	GREEDY
STACK	Discrete Centrifugal Jumps	Codeforces 1407D	elmélyítés	
STACK	Mike and Feet	Codeforces 547B	elmélyítés	
STACK	High Cry	Codeforces 875D	szintézis	BIT OP
STACK	Longest Regular Bracket Sequence	Codeforces 5C	szintézis	DP
STACK	Balloons	OJ.uz CEOI11_bal	lekötő	
QUEUE	Deque-STL	HackerRank deque-stl	alap	
QUEUE	Valeriy and Deque	Codeforces 1179A	bevezetés	
QUEUE	subarrays	SPOJ ARRAYSUB	bevezetés	
QUEUE	Rating Compression	Codeforces 1450D	szintézis	2 POINT
QUEUE	Mind Control	Codeforces 1290A	elmélyítés	
QUEUE	Map	Codeforces 15D	elmélyítés	
QUEUE	Chef and Rectangle Array	CodeChef CHSQARR	elmélyítés	
QUEUE	OpenStreetMap	Codeforces 1195E	elmélyítés	
GRAPHS	Friends	Codeforces 94B	bevezetés	
GRAPHS	Bakery	Codeforces 707B	bevezetés	GREEDY
GRAPHS	Christmas Spruce	Codeforces 913B	bevezetés	
GRAPHS	Network Topology	Codeforces 292B	bevezetés	
GRAPHS	System Administrator	Codeforces 22C	elmélyítés	CONSTRUCT
GRAPHS	Mahmoud and Ehab and the bipartiteness	Codeforces 862B	szintézis	BFS, DFS
GRAPHS	Hálózat kettéosztás	Mester NT, OKTV, IOI Válogató / IOI/CEOI Válogató 2018 / 9.	lekötő	REC
BFS	Távolságok	Codeforces Polygon bfs	alap	
BFS	Message Route	CSES 1667	alap	
BFS	Randi	Mester Haladó / Gráfok, szélességi bejárás / 35.	alap	
BFS	Adott ponton átmenő legrövidebb kör	Mester Haladó / Gráfok, szélességi bejárás / 5.	elmélyítés	
BFS	Futár	Mester Haladó / Gráfok, szélességi bejárás / 17.	elmélyítés	
BFS	The Two Routes	Codeforces 601A	elmélyítés	
BFS	Two Buttons	Codeforces 520B	elmélyítés	
BFS	Monsters	CSES 1194	elmélyítés	
BFS	Fight Against Traffic	Codeforces 954D	elmélyítés	

Téma	Feladatcím	Feladatbank és azonosító / Szöveg	Szerep	Téma 2
BFS	Jumping on Walls	Codeforces 198B	elmélyítés	
BFS	Nastya and Unexpected Guest	Codeforces 1340C	elmélyítés	
BFS	Nearest Opposite Parity	Codeforces 1272E	elmélyítés	
BFS	The Door Problem	Codeforces 776D	elmélyítés	
BFS	BFS-DFS	CSAcademy BFS-DFS	szintézis	DFS
BFS	A Game With Numbers	Codeforces 919F	lekötő	
BFS	Cycle In Maze	Codeforces 769C	lekötő	
BFS	The Great Mixing	Codeforces 788C	lekötő	
BFS	Két mérőkanna	Mester Haladó / Gráfok, szélességi bejárás / 24.	lekötő	
BFS	Police Stations	Codeforces 796D	szintézis	CONSTRUCT
BFS	The Shortest Statement	Codeforces 1051F	szintézis	DFS
BFS	Valid BFS?	Codeforces 1037D	szintézis	GREEDY
DFS	Counting Rooms	CSES 1192	bevezetés	
DFS	Building Roads	CSES 1666	alap	
DFS	Building Teams	CSES 1668	alap	
DFS	Rumor	Codeforces 893C	bevezetés	
DFS	Round Trip	CSES 1669	elmélyítés	
DFS	Learning Languages	Codeforces 277A	elmélyítés	BFS
DFS	Ice Cave	Codeforces 540C	elmélyítés	
DFS	Ice Skating	Codeforces 217A	elmélyítés	BFS
DFS	Chemical table	Codeforces 1012B	elmélyítés	BFS
DFS	Graph Cutting	Codeforces 405E	elmélyítés	
DFS	Cut 'em all!	Codeforces 982C	szintézis	GREEDY
DFS	Cover it!	Codeforces 1176E	szintézis	BFS
DFS	Road Improvement	Codeforces 638C	szintézis	GREEDY
DFS	Making Genome in Berland	Codeforces 638B	szintézis	DAG
DFS	Towers	Codeforces 1637F	szintézis	CONSTRUCT
DFS	Ball-Stackable	Codeforces 1876E	szintézis	STACK
DFS	Infinite Maze	Codeforces 196B	szintézis	BFS
DFS	Nested Rubber Bands	Codeforces 1338D	szintézis	CONSTRUCT
DFS	Permutation Weight (Easy Version)	Codeforces 1685D1	szintézis	CONSTRUCT
DFS	Nastya and Time Machine	Codeforces 1340D	szintézis	CONSTRUCT
DFS	Counter Attack	Codeforces 190E	lekötő	
DFS	Stations	OJ.uz IOI20_stations	lekötő	
DAG	Utak száma	Mester Haladó / Gráfok, körmentes gráfok / 15.	előzmény	
DAG	Course Schedule	CSES 1679	alap	
DAG	Munkasorrend	Mester Haladó / Gráfok, körmentes gráfok / 10.	elmélyítés	
DAG	Fox And Names	Codeforces 512A	elmélyítés	
DAG	Longest Flight Route	CSES 1680	elmélyítés	DP
DAG	Lovely Matrix	Codeforces 274D	elmélyítés	
DAG	Andrew and Taxi	Codeforces 1100E	szintézis	BIN SEARCH
DAG	Almost Acyclic Graph	Codeforces 915D	szintézis	DFS
DAG	Kingdom Connectivity	HackerRank kingdom-connectivity	szintézis	COMB
DAG	String Transformation 1	Codeforces 1383A	szintézis	GREEDY
DAG	Checker for Array Shuffling	Codeforces 1672F2	szintézis	CONSTRUCT
DAG	Ski Accidents	Codeforces 1368E	szintézis	CONSTRUCT
DAG	Minimal Labels	Codeforces 825E	szintézis	PRQUEUE
SHORT PATH	Shortest Routes I	CSES 1671	alap	
SHORT PATH	Shortest Routes II	CSES 1672	alap	
SHORT PATH	Dijkstra?	Codeforces 20C	alap	
SHORT PATH	Központ	Mester Haladó / Gráfok, elemi feladatok / 35.	alap	
SHORT PATH	Autóbusz	Mester Haladó / Gráfok, legrövidebb utak / 22.	alap	
SHORT PATH	Cycle Finding	CSES 1197	alap	
SHORT PATH	Edge Deletion	Codeforces 1076D	elmélyítés	

Téma	Feladatcím	Feladatbank és azonosító / Szöveg	Szerep	Téma 2
SHORT PATH	Greg and Graph	Codeforces 295B	elmélyítés	
SHORT PATH	Jzzhu and Cities	Codeforces 449B	elmélyítés	
SHORT PATH	Roads in Berland	Codeforces 25C	elmélyítés	
SHORT PATH	String Problem	Codeforces 33B	elmélyítés	
SHORT PATH	Féreglyukak	Codeforces Polygon negative-cycles-3	elmélyítés	
SHORT PATH	Féreglyukak - NEHÉZ	Codeforces Polygon negative-cycles-2	elmélyítés	
SHORT PATH	Flight Discount	CSES 1195	elmélyítés	
SHORT PATH	Flight Routes	CSES 1196	elmélyítés	
SHORT PATH	High Score	CSES 1673	elmélyítés	
SHORT PATH	Investigation	CSES 1202	elmélyítés	
SHORT PATH	Permutation Graph	Codeforces 1696D	szintézis	CONSTRUCT
SHORT PATH	Capitalism	Codeforces 1450E	szintézis	CONSTRUCT
SHORT PATH	Flights	Codeforces 241E	szintézis	DAG
SHORT PATH	Labyrinth	Codeforces 1063B	szintézis	BFS
SHORT PATH	Telephelyek	Mester Haladó / Gráfok, elemi feladatok / 34.	szintézis	
SHORT PATH	Bracket Walk	AtCoder ARC173_D	lekötő	
SHORT PATH	Segments	SPOJ SEGMENTS	lekötő	BIN SEARCH
HEAP	Prioritási sor működése	Hogy működhet a prioritási sor? Találj ki egy adatszerkezetet, amely hatékonyan végzi el a műveleteket!	bevezetés	
HEAP	Heap struct írása	Írjunk egy saját Heap struct-ot, azaz kupac adatszerkezetet. vector-t használjunk benne, legyen top(), push(int), pop(), size() művelet.	alap	
HEAP	Apartments	CSES 1084		
HEAP	Chef Teams	CodeChef CTEAMS		
DSU	Disjoint Sets Union	Codeforces DSU course 1A	alap	
DSU	Consecutive Letters	SPOJ CONSEC	elmélyítés	
DSU	Road Construction	CSES 1676	elmélyítés	
DSU	Asya And Kittens	Codeforces 1131F	szintézis	GREEDY
DSU	Phase Shift	Codeforces 1735C	szintézis	DFS
DSU	Royal Questions	Codeforces 875F	elmélyítés	
DSU	DSU with rollback	Codeforces DSU course 3A	lekötő	
DSU	Dynamic Connectivity	CSES 2133	lekötő	
MST	Road Reparation	CSES 1675	alap	
MST	Malmokból szállítás	Mester Haladó / Gráfok, feszítőfák / 7.	alap	
MST	Ellenséges Városok	Mester Haladó / Gráfok, feszítőfák / 2.	elmélyítés	
MST	Villamosítás	Mester Haladó / Gráfok, feszítőfák / 9.	elmélyítés	
MST	Cost	SPOJ KOICOST	elmélyítés	
MST	Maximum Distance	Codeforces 1081D	elmélyítés	
MST	0-1 MST	Codeforces 1242B	elmélyítés	
MST	Design Tutorial: Inverse the Problem	Codeforces 472D	szintézis	SHORTPATH
MST	Edges in MST	Codeforces 160D	szintézis	ART POINT
MST	Roads in HackerLand	HackerRank johnland	szintézis	BIT OP
RANGE DP	Rúd felvágása	Mester Haladó / Dinamikus programozás / 80.	bevezetés	
RANGE DP	Fehér és fekete korongok	Mester Haladó / Dinamikus programozás / 22.	szintézis	GAME DP
RANGE DP	Zuma	Codeforces 607B	elmélyítés	
RANGE DP	Clear the String	Codeforces 1132F	elmélyítés	
RANGE DP	The Sports Festival	Codeforces 1509C	elmélyítés	
RANGE DP	Convex Countour	Codeforces 838E	szintézis	GEOM
RANGE DP	Timetable	Codeforces 946D	elmélyítés	
RANGE DP	The Hard Work of Paparazzi	Codeforces 1427C	elmélyítés	
RANGE DP	Mice and Holes	Codeforces 797F	elmélyítés	QUEUE
LCS DP	LCS	AtCoder DP_F	alap	
LCS DP	Edit Distance	CSES 1639	elmélyítés	

Téma	Feladatcím	Feladatbank és azonosító / Szöveg	Szerep	Téma 2
LCS DP	Jelek	Mester NT, OKTV, IOI Válogató / Nemes Tihamér 2. 2018/19 3. forduló / 4.	elmélyítés	
LCS DP	Leghosszabb tükörszó hossza	Mester Haladó / Dinamikus programozás / 55.	elmélyítés	
LIS DP	Towers	CSES 1073	előzmény	GREEDY
LIS DP	Konténeroszlopok	Mester Haladó / Mohó algoritmusok / 19.	előzmény	GREEDY
LIS DP	Increasing Subsequence	CSES 1145	alap	
LIS DP	Mysterious Present	Codeforces 4D	elmélyítés	
LIS DP	Greenhouse Effect	Codeforces 269B	elmélyítés	
LIS DP	Optical Experiment	Codeforces 67D	elmélyítés	
LIS DP	Clique in the Divisibility Graph	Codeforces 566F	szintézis	NUM THEORY
LIS DP	Supernumbers in a permutation	SPOJ SUPPER	elmélyítés	
LIS DP	Zip-line	Codeforces 650D	elmélyítés	
LIS DP	LIS of Sequence	Codeforces 486E	elmélyítés	
LIS DP	Longest Increasing Subsequence	Codeforces 568E	szintézis	SET MAP
LIS DP	Global Warming	OJ.uz CEOI18_glo	lekötő	
BIT DP	Switch The Lights	UVa 11974	bevezetés	BFS
BIT DP	Vásárlások	Mester NT, OKTV, IOI Válogató / IOI/CEOI Válogató 2018 / 10.	bevezetés	
BIT DP	Matching	AtCoder DP_O	bevezetés	
BIT DP	Kefa and Dishes	Codeforces 580D	elmélyítés	
BIT DP	A Simple Task	Codeforces 11D	elmélyítés	
BIT DP	Another Sith Tournament	Codeforces 678E	elmélyítés	
BIT DP	Keyboard Purchase	Codeforces 1238E	elmélyítés	
BIT DP	XOR Triangle	Codeforces 1710C	elmélyítés	
BIT DP	Elevator Rides	CSES 1653	elmélyítés	
BIT DP	Traveling Graph	Codeforces 21D	szintézis	EULER
BIT DP	Hamiltonian Flights	CSES 1690	szintézis	GRAPHS
BIT DP	Counting Tilings	CSES 2181	lekötő	COMB
SCC	Nincs visszaút	Mester Haladó / Gráfok, mélységi bejárás / 37.	előzmény	
SCC	Karavánok	Mester NT, OKTV, IOI Válogató / OKTV 2018/19 3. forduló / 2.	előzmény	
SCC	Planets and Kingdoms	CSES 1683	alap	
SCC	Checkposts	Codeforces 427C	elmélyítés	
SCC	Reachability from the Capital	Codeforces 999E	elmélyítés	
SCC	Körben lévő pontok	Mester Haladó / Gráfok, mélységi bejárás / 22.	elmélyítés	
SCC	Fedor and Essay	Codeforces 467D	elmélyítés	
SCC	Scheme	Codeforces 22E	elmélyítés	
SCC	Mr. Kitayuta's Technology	Codeforces 506B	elmélyítés	
SCC	Ralph and Mushrooms	Codeforces 894E	elmélyítés	
SCC	Bertown roads	Codeforces 118E	szintézis	ART POINT
SCC	Phoenix and Odometers	Codeforces 1515G	szintézis	NUM THEORY
ART POINT	Körutak	Mester NT, OKTV, IOI Válogató / Nemes Tihamér 2. 2016/17 3. forduló / 5.	előzmény	
ART POINT	Lajhár	Egy lajhár egy fán él, amely gráfelméleti értelemben vett fa is egyben. A gyökér van a fa alján, a fa élei felfelé mutatnak. Van viszont pluszban M csúszda él, amelyek egy csomópontot köthetnek össze valamelyik ősével a fában. Minden csúcsra számítsuk ki az L-értéket, amelyet úgy definiálunk, hogy a legalacsonyabb magasság, amire el lehet jutni egy csúcsból úgy, hogy felfelé mehetünk tetszőlegesen és legfeljebb egyszer lecsúszhatunk egy csúszda élen. N<=10^5, M<=10^5.	bevezetés	
ART POINT	Necessary Roads	CSES 2076	alap	
ART POINT	Necessary Cities	CSES 2077	alap	
ART POINT	Burning Bridges	Codeforces GYM 100200 B	alap	
ART POINT	SUBMERGE - Submerging Islands	SPOJ SUBMERGE	alap	
ART POINT	We Need More Bosses	Codeforces 1000E	elmélyítés	

Téma	Feladatcím	Feladatbank és azonosító / Szöveg	Szerep	Téma 2
ART POINT	Forbidden Cities	CSES 1705	elmélyítés	
ART POINT	One-Way Streets	OJ.uz CEOI17_oneway	elmélyítés	
ART POINT	Case of Computer Network	Codeforces 555E	szintézis	SCC
ART POINT	Tourist Reform	Codeforces 732F	szintézis	SCC
ART POINT	Split the Attractions	OJ.uz IOI19_split	lekötő	
EULER PATH	Morning walk	UVa 10596	alap	
EULER PATH	Mail Delivery	CSES 1691	alap	
EULER PATH	Teleporters Path	CSES 1693	alap	
EULER PATH	Play on words	UVa 10129	alap	
EULER PATH	De Bruijn Sequence	CSES 1692	alap	
EULER PATH	Tanya and Password	Codeforces 508D	elmélyítés	
EULER PATH	Two Paths	Codeforces 36E	elmélyítés	
EULER PATH	One-Way Reform	Codeforces 723E	szintézis	SCC
EULER PATH	Weird journey	Codeforces 789D	szintézis	COMB
EULER PATH	Minimum Euler Cycle	Codeforces 1334D	szintézis	CONSTRUCT
MATCH	School Dance	CSES 1696	alap	
MATCH	Ada and Plants 2	SPOJ ADAPLNTS	elmélyítés	
MATCH	Attacking Rooks	UVa 12668	elmélyítés	
MATCH	Kamehameha	Codeforces Polygon kamehameha	elmélyítés	
MATCH	Elementary Math	Codeforces GYM 101485E	elmélyítés	
MATCH	Brevity is Soul of Wit	Codeforces 120H	elmélyítés	
MATCH	Array and Operations	Codeforces 498C	elmélyítés	
MATCH	Maximize Mex	Codeforces 1139E	elmélyítés	
MATCH	Ada and Cities	SPOJ ADACITY	szintézis	SHORT PATH
GEOM	Point Location Test	CSES 2189	alap	
GEOM	Rotation	Codeforces 100I	bevezetés	
GEOM	Line Segment Intersection	CSES 2190	alap	
GEOM	Point in Polygon	CSES 2192	alap	
GEOM	Bekerítő háromszög	Mester Haladó / Geometriai algoritmusok / 3.	bevezetés	
GEOM	Where do I Turn?	Codeforces 227A	elmélyítés	
GEOM	Arpa and an exam about geometry	Codeforces 851B	bevezetés	
GEOM	Han Solo and Lazer Gun	Codeforces 514B	elmélyítés	
GEOM	Tell Your World	Codeforces 849B	elmélyítés	
GEOM	Crazy Town	Codeforces 498A	elmélyítés	
GEOM	Zoo	Codeforces 183B	elmélyítés	
GEOM	Constellation	Codeforces 618C	elmélyítés	
GEOM	Balls of Steel	Codeforces 1450B	elmélyítés	
GEOM	Full Turn	Codeforces 1468F	elmélyítés	
GEOM	Vasya and Triangle	Codeforces 1030D	elmélyítés	
GEOM	Pair Of Lines	Codeforces 961D	elmélyítés	
GEOM	Épület kerülete	Mester Haladó / Geometriai algoritmusok / 6.	elmélyítés	
GEOM	Lefedő egyenesek	Mester Haladó / Geometriai algoritmusok / 15.	elmélyítés	
GEOM	Zárt poligon készítése	Mester Haladó / Geometriai algoritmusok / 24.	bevezetés	
GEOM	Convex Hull	CSES 2195	alap	
GEOM	Autópálya	Mester Haladó / Geometriai algoritmusok / 2.	elmélyítés	
GEOM	Polygons	Codeforces 166B	elmélyítés	
GEOM	Water Balance	Codeforces 1299C	elmélyítés	
GEOM	Polygon Area	CSES 2191	alap	
GEOM	Circle Game	Codeforces 1451D	szintézis	GAME
GEOM	Number of Parallelograms	Codeforces 660D	szintézis	SET MAP
GEOM	Divide Points	Codeforces 1270E	szintézis	CONSTRUCT
GEOM	Gregor and the Odd Cows (Easy)	Codeforces 1548D1	szintézis	NUM THEORY
GEOM	Nezzar and Nice Beatmap	Codeforces 1477C	szintézis	CONSTRUCT
GEOM	Vanya and Triangles	Codeforces 552D	szintézis	HASH

Téma	Feladatcím	Feladatbank és azonosító / Szöveg	Szerep	Téma 2
GEOM	Winding polygonal line	Codeforces 1158D	szintézis	CONSTRUCT
GEOM	Intersection Points	CSES 1740	szintézis	SEG TREE
GEOM	Freelancer's Dreams	Codeforces 605C	lekötő	
GEOM	Rectangular Polyline	Codeforces 1444D	lekötő	
GEOM	Contain	CodeChef CONTAIN	lekötő	
GEOM	Set of Points	Codeforces 277B	lekötő	
GEOM	Facsemeték bekerítése	Mester Haladó / Geometriai algoritmusok / 26.	lekötő	
MOD INV	Exponentiation II	CSES 1712	alap	
MOD INV	Lineáris kongruencia	ax = b (p) megoldása (a, b, p <= 10^9 , p prím)	alap	
SIEVE	Bear and Prime Numbers	Codeforces 385C	elmélyítés	
SIEVE	Kuzya and Homework	Codeforces 1582G	lekötő	STACK
SIEVE	Simple Subset	Codeforces 665D	elmélyítés	
SIEVE	T-primes	Codeforces 230B	elmélyítés	
SIEVE	Two Divisors	Codeforces 1366D	szintézis	CONSTRUCT
SIEVE	K-th prime	Codeforces Polygon kth-prime	alap	
SIEVE	Common Divisors	CSES 1081	elmélyítés	
ADV COMB	Binomial Coefficients	CSES 1079	alap	
ADV COMB	Game of Thrones II	HackerRank game-of-throne-ii	bevezetés	
ADV COMB	Placing Jinas	Codeforces 1696E	elmélyítés	
ADV COMB	Beautiful Numbers	Codeforces 300C	szintézis	NUM THEORY
ADV COMB	Counting Arrays	Codeforces 893E	szintézis	SIEVE
ADV COMB	Gerald and Giant Chess	Codeforces 559C	szintézis	DP
ADV COMB	Balanced Domino Placements	Codeforces 1237F	szintézis	DP
ADV COMB	Devu and his three Hands	CodeChef DEVHAND	lekötő	
PERM	Mathematical Analysis Rocks!	Codeforces 180F	bevezetés	
PERM	Playing with Permutations	Codeforces 251B	bevezetés	
PERM	Restoration of the Permutation	Codeforces 67B	elmélyítés	
PERM	Sorting by Subsequences	Codeforces 844C	szintézis	DFS
PERM	Square Root of Permutation	Codeforces 612E	szintézis	DFS
PERM	Array Shuffling	Codeforces 1672F1	szintézis	CONSTRUCT
INC-EXC	Prime Multiples	CSES 2185	bevezetés	
INC-EXC	Almost Prime Numbers Again	SPOJ KPRIMESB	bevezetés	
INC-EXC	Grid 2	AtCoder DP_Y	elmélyítés	
INC-EXC	Jzzhu and Numbers	Codeforces 449D	elmélyítés	
INC-EXC	Make It One	Codeforces 1043F	elmélyítés	
INC-EXC	Kompici	SPOJ KOMPICI	elmélyítés	
INC-EXC	Playing around the table	Codeforces 1646F	lekötő	
SEG TREE	Moves on Binary Tree	AtCoder ABC243_D	előzmény	
SEG TREE	Static Range Minimum Queries	CSES 1647	alap	
SEG TREE	Maximum Sum	SPOJ KGSS	elmélyítés	
SEG TREE	Hotel Queries	CSES 1143	elmélyítés	
SEG TREE	Range Update Queries	CSES 1651	elmélyítés	
SEG TREE	Prefix Sum Queries	CSES 2166	szintézis	PREFIX SUM
SEG TREE	A Simple Task	Codeforces 558E	szintézis	BIT OP
FENWICK	Dynamic Range Sum Queries	CSES 1648	alap	SEG TREE
FENWICK	Inversion Count	SPOJ INVCNT	elmélyítés	
FENWICK	Enemy is weak	Codeforces 61E	elmélyítés	
FENWICK	Inreasing Subsequence II	CSES 1748	szintézis	LIS DP
HASH	Ada and Spring Cleaning	SPOJ ADACLEAN	bevezetés	
HASH	Double Profiles	Codeforces 154C	bevezetés	
HASH	Finding Periods	CSES 1733	elmélyítés	
HASH	Palindromic characteristics	Codeforces 835D	elmélyítés	
HASH	Check Transcription	Codeforces 1056E	elmélyítés	
HASH	Periodikus szavak	NJudge KK24_periodicwords	elmélyítés	

Téma	Feladatcím	Feladatbank és azonosító / Szöveg	Szerep	Téma 2
HASH	Túlcsordulás	NJudge KK24_carry	elmélyítés	
HASH	Tree Isomorphism I	CSES 1700	szintézis	TREE REC
HASH	Tree Isomorphism II	CSES 1701	szintézis	TREE REC
TRIE	No Prefix Set	HackerRank no-prefix-set	bevezetés	
TRIE	Good Substrings	Codeforces 271D	elmélyítés	
TRIE	Short Code	Codeforces 965E	elmélyítés	
TRIE	Vasiliy's Multiset	Codeforces 706D	elmélyítés	
TRIE	Distinct Substrings	SPOJ DISUBSTR	elmélyítés	KMPZ
TRIE	A Lot of Games	Codeforces 455B	szintézis	GAME DP
TRIE	Beautiful Subarrays	Codeforces 665E	szintézis	PREFIX SUM, BIT OP
TRIE	Maximum Xor Subarray	CSES 1655	szintézis	PREFIX SUM, BIT OP
TRIE	Sausage Maximization	Codeforces 282E	szintézis	BIT OP
TRIE	Xor-MST	Codeforces 888G	szintézis	MST
TRIE	Diccionário Portuñol	UVa 12359	szintézis	COMB
KMP, Z	Pistike	Pistike kapott egy N elemű számsorozatot a szülinapjára: S[0..N-1]. Meg szeretné határozni minden pozícióra, hogy mekkora az a leghosszabb ott végződő szelet, ami megegyezik egy kezdőszelettel (és nem a legelején kezdődik): P[i] = max {k : S[0..k-1] = S[i-k+1..i] , k <= i}	alap	
KMP, Z	Zolika	Zolika is kapott egy N elemű számsorozatot a szülinapjára: S[0..N-1]. Meg szeretné határozni minden pozícióra, hogy mekkora az a leghosszabb ott kezdődő szelet, ami megegyezik egy kezdőszelettel: Z[i] = max {k: S[0..k-1] = S[i..i+k-1] , k <= N - i }	alap	
KMP, Z	String Functions	CSES 2107	alap	
KMP, Z	String Matching	CSES 1753	alap	
KMP, Z	Shift The String	CodeChef TASHIFT	elmélyítés	
KMP, Z	Password	Codeforces 126B	elmélyítés	
KMP, Z	MUH and Cube Walls	Codeforces 471D	elmélyítés	
KMP, Z	Prefix-Suffix Palindrome (Hard version)	Codeforces 1326D2	elmélyítés	
KMP, Z	Prefixes and Suffixes	Codeforces 432D	elmélyítés	
KMP, Z	Compress Words	Codeforces 1200E	elmélyítés	
KMP, Z	Martian Strings	Codeforces 149E	elmélyítés	
KMP, Z	String Compression	Codeforces 825F	elmélyítés	
KMP, Z	Extend to palindrome	UVa 11475	elmélyítés	
KMP, Z	Anthem of Berland	Codeforces 808G	szintézis	DP
KMP, Z	Holiday Wall Ornaments	Codeforces 1575H	szintézis	DP
KMP, Z	Concatenation with intersection	Codeforces 1313E	szintézis	FENWICK
KMP, Z	Remove The String	SPOJ PSTRING	lekötő	
SQRT	Points on Plane	Codeforces 576C	bevezetés	
SQRT	Holes	Codeforces 13E	bevezetés	
SQRT	Train Maintenance	Codeforces 1580C	elmélyítés	
SQRT	Give Away	SPOJ GIVEAWAY	elmélyítés	
SQRT	D-Query	SPOJ DQUERY	elmélyítés	
SQRT	Powerful array	Codeforces 86D	elmélyítés	
SQRT	Xenia and Tree	Codeforces 342E	szintézis	LCA, BFS
SQRT	Sum of Divisors	CSES 1082	szintézis	NUM THEORY
LCA	Kth Ancestor	HackerRank	előzmény	
LCA	LCA	SPOJ LCA	alap	
LCA	Distance in the Tree	Timus 1471	alap	
LCA	Lowest Common Ancestor	CodeChef TALCA	elmélyítés	
LCA	Tree and Extra Edge	CodeChef TREEDGE	elmélyítés	
ST, RMQ	Static Range Minimum Queries	CSES 1647	alap	
ST, RMQ	Matchsticks	CodeChef MSTICK	bevezetés	

Téma	Feladatcím	Feladatbank és azonosító / Szöveg	Szerep	Téma 2
ST, RMQ	Turn Off The TV	Codeforces 863E	elmélyítés	
FLOW	Download Speed	CSES 1694	alap	
FLOW	Police Chase	CSES 1695	elmélyítés	
FLOW	Distinct Routes	CSES 1711	elmélyítés	
FLOW	Soldier and Traveling	Codeforces 546E	elmélyítés	
FLOW	Fox And Dinner	Codeforces 510E	elmélyítés	
FLOW	Rectangle Painting 2	Codeforces 1198E	elmélyítés	
FLOW	Starry Night Camping	Codeforces 1517G	elmélyítés	
FLOW	Oda-vissza független út megadása	Mester Haladó / Gráfok, mélységi bejárás / 38.	elmélyítés	
FLOW	Delivery Bears	Codeforces 653D	szintézis	BINSEARCH
FLOW	Not Too Many Balls	AtCoder ABC332_G	lekötő	
FLOW	Pairing Wizards	AtCoder ARC142_E	lekötő	
FLOW	Zebraness	AtCoder ABC193_F	lekötő	
GAUSS	Central Heating	Timus 1042	alap	
GAUSS	Snakes and Ladders	UVa 12910	elmélyítés	
GAUSS	XOR Maximization	SPOJ XMAX	elmélyítés	
GAUSS	Lost Array	Codeforces 1713F	szintézis	BITOP
GAUSS	No Game No Life	Codeforces 1464E	szintézis	GAME
2SAT	Bűnösök és ártatlanok	N ember közül néhányan egy bűntényt követtek el. A kihallgatás alatt mindenki csak olyan állításokat mondott, hogy "X bűnös", vagy "Y ártatlan". Azt tudjuk, hogy az ártatlanok igazat mondanak, a bűnösök viszont hazudhatnak is (és mondhatnak igazat is). Add meg a lehető legtöbb olyan embert, akiről az állítások alapján biztosan lehet tudni, hogy bűnös! Bemenet formátuma: N M : emberek száma, állítások száma (1<=N,M<=10^3) Ezt követően M sorban egy-egy állítás, az alábbi két formátumban lehet: U -X : U azt állítja, hogy X bűnös (1<=U,X<=N) U X : U azt állítja, hogy X ártatlan (1<=U,X<=N) Kimenet formátuma: A biztosan bűnösek felsorolva.	bevezetés	
2SAT	Prove Them All	UVa 13057	bevezetés	
2SAT	Giant Pizza	CSES 1684	alap	
2SAT	Catowice City	Codeforces 1239D	elmélyítés	
2SAT	Radio Stations	Codeforces 1215F	elmélyítés	
2SAT	Too Many Constraints	Codeforces 1697F	elmélyítés	