A Lineáris Keresés Buktatói

Szalayné Tahy Zsuzsanna[1], Czirkos Zoltán[2]

[1] sztzs@caesar.elte.hu
ELTE IK
[2] czirkos@eet.bme.hu
BME VIK

Absztrakt. A lineáris keresés a mindennapok egyik legismertebb algoritmusa, mégis problémát jelent a programozás oktatása illetve tanulása során. Sok diák elakad az eldöntés vagy keresés feladatok kódolásakor. Állandó vitaforrás tanár és diák között a break használata. Miért használják előszeretettel a diákok a nem tanított break utasítást? Miért tiltják a tanárok a használatát? Valóban káros a break? Hogyan lehet konfliktusmentesen és hatékonyan tanítani a lineáris keresést? Mi okoz a tanulás során akadályokat, zsákutcákat? Ebben a cikkben bemutatjuk az elméleti és ösztönös megoldásokat; egyszerű, összetett és esetenként téves gondolkodási sémákat; az egyes sémákból meghatározható készségfejlesztési szempontokat.

Kulcsszavak: lineáris keresés, break használata, logikai kifejezések, programozás oktatása.

1 Bevezető

A programozás oktatásának a tananyag felépítése, hangsúlya alapján osztályozva, kilenc különböző módszerét írja le Szlávi Péter és Zsakó László [1]: algoritmusorientált, adatorientált, specifikációorientált, problématípus-orientált, nyelvorientált, utasításorientált, matematikaorientált, hardverorientált és modellorientált. A tanítási gyakorlatban a tanterv, tanmenet és sokszor a tanulók előzetes ismeretei alapján ezen módszerek kombinációját alkalmazzuk a közoktatásban és az egyetemi képzésben is [2]. Sok éves tanítási gyakorlat és számos kombináció kipróbálása során kristályosodott ki, hogy az egyik legkritikusabb pont, amely meghatározza a tanulók programozás tudásának fejlődését, a lineáris keresés kódolása.

Egy átlagos osztályban a programozást tanulók tudásszintje nagyon eltérő. Amikor a lineáris keresés feladatok önálló megoldására kerül sor, rendszerint megtalálhatók közöttük az alábbi típusú tanulók:

A kezdő, aki már tanulta a lineáris keresést, de még nem tudja alkalmazni. Ők a feladatok megoldását gyakran úgy kezdik, hogy „talán kell valami if()” vagy felteszik a kérdést, nem lehetne-e mondjuk táblázatkezelőben FKERES() függvénnyel megoldani a feladatot.

A „programozó”, aki már írt programot iPhonera, Androidra, webes platformra.

A „programozó”, aki már „profi” PHP utasítások és szkriptek írásában.

Néhány tehetséges tanuló, akik maguk találnak megoldást a feladatokra.

A haladó ismeretekkel rendelkező tanuló, aki a többiek előtt jár tanulmányaiban és tudja, hogyan kell megoldani az adott típusú feladatot.

Kiadva egy lineáris keresést is tartalmazó feladatot, a fenti csoportokba sorolható tanulók közül csak nagyon kevés tanuló tudja az elvárt kódot megírni. Több esetben az eredmény egy, az adott esetben helyes eredményt adó kód, de a megoldás, a gondolkodásmód általánosítása számos probléma forrása. A harmadik, rendszerint a legnagyobb csoport hibás eredményt adó kódot ír. Ennek a csoportnak kis segítséget adva képesek megírni a megoldást, de a következő feladatba ismét beletörik a bicskájuk. Amennyiben sorozatosan sikertelen a feladatok megoldása, a tanulók jelentős része feladja a próbálkozást, elkönyveli, hogy soha nem fog megtanulni programozni. A módszertanilag hibás, de működő megoldást adó tanulók sokszor azért adják fel a programozás tanórai tanulását, mert úgy

gondolják, saját fejük után menve sikeresebbek lesznek. Ne felejtsük el, elsősorban 25 évnél fiatalabb tanulóknak tanítunk programozást, a kamaszok lázadásával is meg kell birkóznia a tanárnak.

A keresés programozásának oktatása idején rendszeresen felmerül a tanárokban a kérdés: Hogyan motiváljam a tanulót a programozás tanulására, hogyan motiváljam a „tiszta kód" elveinek megfelelő, illetve strukturált program írására?

A legtöbb magyarországi, programozást oktató tanár az ELTE IK-n tanult programozást és programozás módszertant. Ők a strukturált programozást algoritmusorientált módszerrel tanulták, közöttük nagyon sokan felnőttként. Ebből adódóan jellemző körükben a jó matematikai absztrakciós és logikus gondolkodás készsége. A tanárok nagyon gyakran azt a módszert preferálják a tanítás során, amilyen módszerrel ők is tanultak. A keresést igénylő feladatok programozása azért nehéz a tanulóknak, mert nem rendelkeznek azokkal a képességekkel, amellyel a tanár rendelkezett, amikor ezt a témát tanulta.

Az „első generációs programozó" képzésének problémáját kiegészíti egy másik, vitatott kérdés: Mi a programozás oktatásának a célja? Azt várjuk el a tanulóktól, hogy egy működő megoldást adjanak a problémára, vagy meg kell tanulniuk a strukturált programozást? A közoktatásban nem cél a programozói szakma megtanítása, de mégsem lehet attól teljesen független a tananyag, hiszen előkészíti a szakma tanulását is. A témában Dijkstra és Robert C. Martin jegyezte [3] [4] két legnagyobb hatású cikk és számos ismert programozó és tudós véleménye ismert, de meg kell állapítanunk, hogy a lineáris keresés tanulási útja (a tanulás egyes fázisai) módszertani szempontból nem felelnek meg a szakma gyakorlati elvárásainak. A lineáris keresés tanításának módszertani problémái fel-felbukkanó témák a tanári levelezési listákon, mint például a CAS Community of Teachers in UK: *A break és continue használata ciklusban* [5], vagy az informatikatanárok Sulinetes levelezési listáján: *Kiugrás feltételes ciklusból* [6] illetve *Programozás eldöntés* [7].

Kutatásunkban megvizsgáljuk a lineáris keresés (és eldöntés) feladatok megoldásának gyakorlatát, a megoldásokkal kapcsolatos tanulás- és programozás-módszertani kérdéseket és javaslatokat fogalmazunk meg a programozás tanításának módszerére.

2 Tipikus tanulói megoldások lineáris keresés kódolására

Különböző módszerek vannak a programozás tanítására és tanulására, de mindegyik módszer és minden tanulási út tartalmazza a keresés problémakört. Gimnazista diákokat vizsgáltunk, akik a vizsgálatot megelőzően már tanulták a vezérlési szerkezeteket. Többségük hallott már programozási tételekről; frontális vagy csoportmunka keretében megírtak már néhány programot. Miután a diákok közösen, illetve kis tanári segítséggel sikeresen oldották meg a feladatokat, egy adatbekérést, keresést (vagy eldöntést) és adatkiírást tartalmazó feladatot kaptak. A megoldást önállóan kellett elkészíteniük, felmerülő kérdéseikre semmitmondó, elbizonytalanító válaszokat kaptak. A kísérletben körülbelül harminc diákot vizsgáltuk. Egyszerre legfeljebb 4 diák dolgozott, olyanok, akik épp elérték a megfelelő szintet, csoportban már tudtak programot írni. Mialatt a tesztelt diákok önállóan dolgoztak, körülöttük más feladattal foglalkoztak a csoporttársak és általában tanári felügyelet is biztosította, hogy csak saját magukra számítsanak. [8]

A kísérlet egyik legfeltűnőbb eredménye, hogy bár többen kimondottan tanulták a lineáris keresés algoritmusát, senki nem vette észre, hogy olyan feladatról van szó, aminek az általános megoldását már tanulták (volt róla szó tanórán).

Azok, akik a kísérletet megelőző néhány hétben kezdték meg programozási tanulmányaikat, mindjárt az elején segítséget kértek, hogyan kezdjék el a kódolást. Korábbi tapasztalatuk alapján három vezérlési szerkezet egyike tűnt számukra jó megoldásnak: for () while () vagy if (). Kérdésükre „talán kockadobás segít" volt a válasz.

Az összes vizsgált eset közül két kezdő az if () mellett döntött, mindenki más for () ciklust gépelt be először. Néhányakban felmerült a while () használatának lehetősége, de a for () kézenfekvőnek látszott. Akik az if () mellett döntöttek, hamar, még a feltétel tényleges megfogalmazása előtt kiegészítették a kódot for () ciklussal.

Az első megoldás helyett öt esetben kiválogatást végeztek a tanulók. Tipikus kódrészlet:

1a példa:

```
for (int i = 0; i < list.Count; i++ )
  if («logikai kif.»)list2.Add(elem);
```

Az eredményt látva módosították a kódot, hárman a keresés helyett számlálás tételt használtak:

1b példa:

```
for (int i = 0; i < list.Count; i++)
  if («logikai kif.») jo++;
```

Majd a megoldás módosításával, logikai változó bevezetésével vagy a feltétel bővítésével, számítás átírásával, az alábbihoz hasonló végső megoldást adtak:

1c példa:

```
for (int i = 0; i < list.Count; i++)
  if («logical_expr») jo = i;
```

Bár ez a verzió már helyesen válaszol a kérdésre, a diákok érezték, hogy megoldásuk nem optimális. Nem az első jó megoldást adja, hanem az utolsót és egyébként is, feleslegesen vizsgálja meg az összes elemet. Ennek ellenére, a feladatot megoldottnak tekintették.

Hét diák a for () megírása után rögtön írta a ciklusmagba az if () utasítást, de a keresett elem jellemzőjének beírása után megakadt. Hosszú ideig gondolkodtak, de nem volt semmi ötletük, mígnem kihasználták a felügyelet gyenge pillanatát… Hárman a teremben tartózkodó haladó programozóktól kértek tanácsot, akik egy rápillantás után mondták a megoldást: break utasítás kell. Mint csodatévő kód, terjedt a break utasítás a diákok között. Az akkor még nem vizsgált diákok is megjegyezték, mint problémamegoldó eszközt. Így végül tizenkét diák megoldása így nézett ki:

2. példa:

```
for (int i = 0; i < list.Count; i++)
  if («logikai kif.»){jo = i; break;}
```

A másik csoport négy diákja is hallott a break utasításról, de nem akartak "cheat kódot" használni, ezért tanári segítséget kértek. A segítséget, a minimális kódmódosításra koncentrálva, az jelentette, hogy a break (kiugrás) helyett, a feltétel tagadása azt mondja meg, hogy folytatódjon a ciklus, azaz az if ()-ben szereplő feltétel tagadását be lehet írni a for () feltételének. A megoldás végiggondolása nehéz, jellemző, hogy az így felírt ciklusfeltételt nem tudták a diákok köznyelvi mondatban megfogalmazni. Másrészt, a kód transzformációja majdnem olyan könnyen megjegyezhető, mint a break utasítás. Az eredmény:

3. példa:

```
int i;
for (i = 0; i < list.Count && !(«logikai kif.»); i++);
```

Azok, akik korábban egyéni úton tanultak programozni, jellemzően a 2. vagy a 3. példában mutatott megoldást adták, de akik korábban a break utasítást javasolták, megtanulták az elkerülés módját. Két kezdő programozó volt nyitott a „hivatalos" megoldás megtanulására, mert zavarta őket a mag

nélküli ciklus. Mellettük három, programozás versenyeken is induló diák tanári kérésre megírta „rendesen” a kódot:

4. példa:

```
int i=0;
while (i < list.Count && !(«logikai kif.»))
  i++;
```

Három diák a mobil és web-programozás témában volt jártas. Ők nagy gyakorlattal rendelkeztek a programozási nyelv előre megírt metódusainak használatában. A feladatot nagyon gyorsan megoldották a beépített eszközök használatával:

5. példa:

```
var solution = list.Contains(«logical_expr»);
```

Ezek a diákok nehezen értették meg, hogy miért fontos tudni, hogy hogyan működik a Contains() eljárás, miért kell megírniuk egy függvényt, ami már meg van írva. Nehezebb volt őket meggyőzni, mint azokat, akik teljesen kezdőként a táblázatkezelőben történő megoldást javasolták a kódolás helyett.

Végül, de nem utolsó sorban, volt egy diák, aki úgy gondolta, hogy olyan megoldást ad, ahol nem a ciklus után, az index vizsgálatával határozza meg az eredményt, hanem beépíti ezt az elágazást is a ciklusba:

6. példa:

```
for (int i = 0; i < items.size() or
  (items.push_back(make_pair(item, 1)), false); i++)
{
  if (items[i].first == item)
  {
    items[i].second++;
    break;
  }
}
```

3 A problémák gondolkodásmódbeli háttere

3.1 Miért nem a tanult megoldást írták? Úgy, ahogy értik.

Megkérdeztük a diákokat, miért nem a while () ciklust választották a megoldáshoz, hiszen tanulták:

A feladat megfogalmazása a kiválasztáshoz volt hasonló

A while () szintaktikája túl bonyolultnak tűnt, a for () ciklus sokkal kézenfekvőbbnek látszik az elemek sorozatára és a feltételt könnyebb külön megadni.

A feladat megfogalmazása valóban elrejtette a keresés (eldöntés) tétel alkalmazásának lehetőségét, mivel nem szerepelt benne a „keres” kifejezés. Helyette az „add meg, ki az|melyik|milyen az, aki egy adott tulajdonsággal rendelkezik” megfogalmazás szerepelt. Ezek a tanulók a megoldás során fokozatosan rájöttek, hogy más megoldásra van szükség és az eredeti megoldásukat addig módosították, amíg megkapták a végeredményt. A keresés kigyűjtéssel illetve megszámlálással való összetévesztésének elméleti alapja is van. Matematikailag a keresés, ahogy a kigyűjtés és a megszámlálás is, felírható összegzési tételként [9]. Az összegzés formájában történő megvalósítás csak egy elméleti levezetés érdekes mellékterméke. Mi okozza, hogy a gyakorlatlan programozó tipikusan erre gondol?

A diákok magyarázataként említett „kézenfekvő megoldás” okára rákérdezve kiderült, hogy úgy gondolkodtak, hogy végig kell nézni az elemeket, és ha megvan, ki kell írni a találatot. Ha „végig kell nézni”, akkor a for () ciklus jó – és ez mindjárt közelebb hozza az összegzés tételt is. A beszélgetést követően számos programozót, oktatót, tanárt megkérdeztünk, internetes forrásokat átolvastunk, azt vizsgálva, hogy a keresésnél „végig” kell-e nézni az elemeket. Azt tapasztaltuk, hogy tudatosan vagy sem, de minden szöveges leírás, magyarázat úgy kezdődik, hogy „végignézzük”… Természetesen a következő tagmondat már felülírja ezt, de tény, hogy a „sorra vesszük egymás után az elemeket” helyett, a szinonimájaként használt „végignézzük” szerepel, ami indukálja a for () ciklust. A programtervező képzésben a for () ciklus használata a meghatározott számosságú elem sorra vételéhez, bejárásához kapcsolódik, azaz előre ismert lépésszámú ciklusszervezésről van szó. A műszaki képzésben a nyelvi megfeleltetést erősítik azzal, hogy a for () ciklust használják minden olyan esetben, amikor a ciklusváltozó léptetésének meghatározott határa van. Azaz végig, a lezáró nulláig, a null pointer eléréséig, az adatbevitelt lezáró enterig vagy adatfolyam végét jelző -1-es értékig… végig, a végjelig. Emiatt mindazok, akik a gondolataikat követik, vagy műszaki szemlélettel tekintenek a feladatra, a for () ciklust fogják választani. Közülük azoknak, akik a programozási nyelvnek csak az elméleti alapjait ismerik, a „vége” előtt kilépés komoly problémát fog jelenteni.

A nyelvi megfogalmazás pontatlan értelmezéséből adódó problémák nem csak a középiskolai oktatásban tapasztalhatók. A BME mérnökinformatikus képzésében a sztringekkel kapcsolatos függvények tanítása során már alkalmazni kell a keresés/eldöntés algoritmust [10]. A függvények ismeretét számon kérő időközi dolgozat (kis ZH) tipikus hibája a keresés hibás megvalósítása. Az első féléves mérnökinformatikus hallgatók körében 2016-ban végzett felmérésünkben a keresés alkalmazásának elvárását explicit adtuk meg, de a feladatmegfogalmazás megoldásra gyakorolt hatásának vizsgálatára az egyszerre megírt a/b feladatsorok között a „nem talált” eset kezelését eltérően fogalmaztuk meg:

7. példa:

A csoport: Írj függvényt, amelyik az első paraméterében adott sztringben megkeresi a második paraméterében adott karakter első előfordulását! Ha talált, térjen vissza annak indexével, *egyébként* pedig –1-gyel!

B csoport: Írj függvényt, amelyik az első paraméterében adott sztringben megkeresi a második paraméterében adott karakter első előfordulását! Ha talált, térjen vissza annak indexével. *Ha sehol nincs benne olyan karakter, akkor pedig* –1-gyel!

Az elemzésben 432 hallgató eredménye szerepel, közülük 219 írt a) csoportot, 213 b) csoportot. Bár a feladat szövege szerint „írj függvényt, amelyik… megkeresi” keresésről szól a feladat, a hallgatók 12–14%-a nem írt megoldásában ciklust. Vagy nem kezdett bele a feladat megoldásába vagy rögtön az eredmény elemzésével próbálkozott – a „Ha”-ra koncentrált. A ciklust írók közül további 5–10% a cikluson belülre írta az eredménytelen keresés visszatérését, azaz az alábbi mintát írta:

„Nem keres végig”

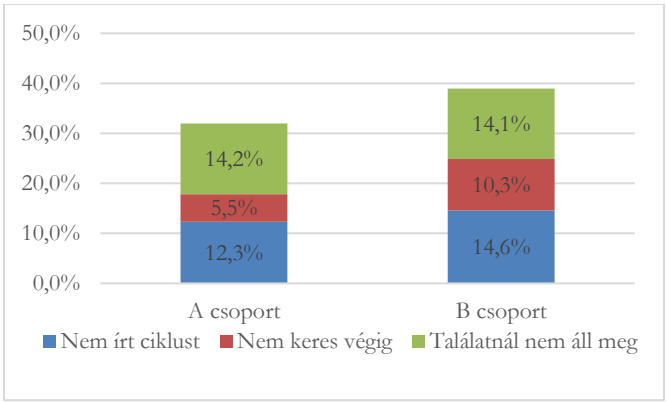
```
int keres(char* str, char mit) {
  int i;
  for (i = 0; str[i] != '\0'; ++i)
    if (str[i] == mit)
      return i;
    else
      return -1;
}
```

A ciklust felírók másik 14%-a a megtalált első érték után is folytatja a keresést, pedig a feladat kimondottan az első előfordulást kéri. (Hasonlóan az 1a – 1c példákhoz.) Az ő kódjuk jellemzően számos egyéb hiba mellett az alábbihoz hasonló:

„Találatnál nem áll meg”

```
int keres(char* str, char mit) {
  int i;
  int ez = -1;
  for (i = 0; str[i] != '\0' ; ++i)
    if (str[i] == mit)
      ez = i;
  return ez;
}
```

Az alábbi diagramon az is látszik, hogy a megfogalmazás az egyes hibák előfordulásának gyakoriságát befolyásolhatja, az ellenkező értelemben megismételt „Ha” jobban indukálja az „if() then else” feltételt, mint a semlegesebb „egyébként”. Hiába van kiemelve, hogy a második feltétel a „sehol nincs benne” esetre vonatkozik, a ciklus is több esetben hiányzik és a ciklusmagban jelentősen gyakoribb az „else ág”.



Keresés feladat hibás megoldásai

A felmérés két dologra mutat rá: Egyrészt a feladat szövege befolyásolja a helyes specifikációt, ha pontosan – de esetenként nem életszerűen – fogalmazunk, jobb lehet az eredmény. Másrészt, még a legegyszerűbb keresési feladatnál is nehézséget okoz az eredmény meghatározása.

3.2 Miért nem a tanult megoldást írták? Bűvészkedés az eredménnyel

Az 1. példa esetén és a BME-s hallgatóknál is tapasztalható, hogy a keresés/eldöntés eredményének megadása problémát okoz. Az elvárt megoldás eldöntés esetén a „nem ment a végén túlra” kifejezést értékeli ki a kívánt tulajdonság teljesülése helyett. A lehetőséget erre az adja, hogy a programozásban a logikai kifejezésre rövidzár kiértékelést használunk, nem a matematikai formulát. Az eredményt a „nem vége és nem megfelelő” feltételű ciklus megszakadásakor kell meghatározni, azaz ennek tagadása esetén: „vége vagy megfelelő”. A tagadás után újabb agytorna, hogy nem azt nézzük megfelelő-e, hanem indirekten egy implikációt alkalmazunk: „ha nem vége, akkor megfelelő, különben nem…”. A programozást tanuló diákok, hallgatók előismeretei között szerepel a matematikai logika, de a keresés eredménye meghatározásának az elméletét nem tanítják úgy, mint az indirekt bizonyítást. Ebből következik, hogy aki nem érzi, nincs megfelelő szintű logikai készsége, az kerülő utat keres. Ez több esetben egyben tévutat is jelent.

Másik diákmegoldás a nehézség elkerülésére, olyan megoldást kitalálni, ami szükségtelenné teszi az eredmény utólagos meghatározását. Ennek egyik triviális esete, hogy a programkódból függvényt

írunk, így a break utasítás nyelvtől esetleg függő hatása helyett vissza tudjuk adni a vezérlést a függvényt meghívó programnak. Az ilyen típusú megoldás a gyakorlatban megszokott és elfogadott, oktatásban viszont problémát okoz a paraméterek megfelelő át- és visszaadása.

A „szuper megoldás", ami kihívás egyes diákok számára, olyan kód, ami a nyelv specialitásait kihasználva megspórol karaktereket vagy a cikluson belül létrejön az eredmény. Erre láttuk a 6. példát. Amennyire „zseniális" a megoldás a diák szerint, annyira vitatható is. A ciklus feltétele:

```
i < items.size() or (items.push_back(make_pair(item, 1)), false)
```

A logikai „vagy" műveletre alkalmazza a rövidzár kiértékelést, azaz a második operandus csak i = items.size() esetben lesz kiértékelve.

A második operandus tartalmaz egy void értékű, mellékhatással rendelkező utasítást („szuper, hogy ide tetszőleges programot be lehet írni").

Ezután (vesszővel elválasztva) egy konstans false érték fogja leállítani a ciklus futását.

A for () ciklust elméletileg számlálós ismétlésre szánták. Bár ennél jóval szélesebb a felhasználási köre, minden elfogadott alkalmazása teljesíti a for (utasítás(ok); feltétel, utasítás(ok)) formátumot [11]. A diákot kifaggattuk, hogyan jutott eszébe ez a megoldás. Az interneten talált egy ehhez nagyon hasonlót.

Itt szeretnénk felhívni a figyelmet, hogy a mobil- és web-fejlesztő diákok jelentős része is kontroll nélkül, az internetről gyűjti össze ismereteit. Sokszor tanulnak meg speciális, a szakmai szabályoknak nem megfelelő kódokat. Ezekben az esetekben vagy akár a break használata esetén is, a tiltás nem használ. A probléma nem a break utasítás használata, hanem az általános kódolási elvek, módszerek ismeretének (és használatának) a hiánya.

3.3 Miért nem a tanult megoldást írták? Ahogy kellene, de nem megy

A fenti problémákra a legkézenfekvőbb megoldás az, amit a gimnáziumban tanító tanárok próbálnak megvalósítani, ha megtanítjuk a diákokat arra, hogy először tipizálják a feladatot, figyelmesen, többször elolvasva döntsék el, hogy összegzés… vagy keresés jellegű feladatról van-e szó. Ezt segíti a táblázatkezelés előzetes oktatása is: döntsd el, melyik függvénnyel oldanád meg. Ha a diák tudja, hogy keresés típusú feladatot kell megoldania, akkor már csak a megoldás sablonját kell elővenni és megírni a kódot. Azonban a megoldásnak két buktatója is van:

A diák nem tudja gyorsan és biztonságosan megállapítani a feladat típusát.

A diák nem tudja a megoldási sablont alkalmazni az adott feladatra.

A feladat típusának meghatározását gátolja a hétköznapi megfogalmazás és az a tény, hogy programozni akar, kódot akar írni, nem pedig elemezni, korábbi hasonló feladatokat keresni.

De ha rá is jön egy diák arra, hogy keresésről van szó, a while () ciklusfejébe írandó feltétel az esetek jelentős részében túl bonyolult, ezért inkább „lépésenként" oldja meg, először is „végignézi". Egy átvett csoportban, több éves előtanulmány után a diákok egyértelműen a while () ciklust választották, de a megoldást négyből csak egy diák tudta felírni.

Az ELTE IK programtervező informatikus szakán a Programozási Alapismereteket szintén az itt javasolt, algoritmus orientált módszerrel tanítják [12]. A feladatmegoldás első lépése a specifikáció, ami kezdők szintjén a feladat típusának azonosítását jelenti. Ezt követi a modellezés, a struktogram elkészítése, majd a kódolás. Ez az elmélet… A gyakorlatban a kezdők inkább kódolnak először, mert a struktogramot nem tudják kipróbálni. Az oktatási módszerből következően, a feladatokról meg tudják állapítani, hogy melyik programozási tétellel valósítható meg, de a keresés implementálása 2 hónapos tanulás után struktogrammal és kódolva is nehéz, több hallgatónak egyáltalán nem megy. A 2016 ősszel kezdő esti tagozatosokat – akik ugyan kezdők a programtervező informatikus szakon, de önismeretük alaposabb, mint egy épp érettségizettnek – kikérdezve, egy hallgató megfogalmazása: „Én azt

tudom, hogy mit keresek, nem tudom leírni, hogy mit nem keresek.". Egy másik hallgató véleménye a saját tudásáról: „Általában meg tudom írni a programokat, csak azt nem tudom, hogy a while ()-ba mit kell írni. Csak a keresés nem megy." Hozzátehetjük, hogy „továbbá az eldöntés, kiválasztás, unió, metszet tételek sem mennek." Azaz körülbelül a tételek fele okoz gondot, ami már elég kétségessé teszi a ZH-k sikerességét.

A képzési tematikában ráadásul van egy rejtett keresési/kiválasztási probléma is, a beolvasott adatok ellenőrzése. A legelső kódminta a jegyzetben, ami a vércsoport meghatározásához kéri be az adatot így néz ki:

8. példa:

```
do {
  cout<<"Kerem az … egyik gent (a/b/0):";
  cin>>x;
}while ((x != 'a') && (x !=' b') && (x != '0'));
//az előfeltétel szerint csak a,b vagy 0 lehet
```

A while () feltételében szereplő kifejezést olvasva, a hallgatók többsége (75%) csak elfogadja azt, valójában nem tudja az előfeltételhez kötni. Csak a korábban gyakorlattal rendelkező hallgatókban merül fel, hogy a de Morgan azonosság miatt igaz az állítás, de még ők sem biztos, hogy önállóan meg tudnák írni. Az előfeltétel logikai kifejezéssé alakítása:

```
(x == 'a') || (x == 'b') || (x != '0')
```

A kódban ennek tagadása szerepel átalakítva, amit a hallgatók nem tudnak fejben elvégezni. Többen írásban sem gyakorlottak a logikai kifejezések átalakításában, ha valaha tanulták is, az csak két kifejezésre, igazságtábla felírásával történt.

A nehézséget feloldaná, ha a beolvasáskor az adat elfogadásához először az előfeltételt kódolnák és nem egyből annak tagadását. Általánosan: a keresés/eldöntés/kiválasztás tételekkel megoldható feladatok kódolását, modellezését könnyítené, ha először az utófeltételt kódolnák a hallgatók. Korábbi informatika tanulmányokban és a mindennapi életben is adódik olyan helyzet, hogy előbb rögzítjük, mi felel meg, majd utána kezdünk megoldást keresni. Például adatbázis lekérdezések tervező rácsán vagy táblázatkezelőben adatkeresésnél ez történik. Az előzetes feltétel megadás lényegében egy előzetesen megírt „bool Megfelel(…)" függvény lenne. Ezt azonban kezdő programozók nem tudják megírni, mert a függvények írása későbbi tananyag. Mit lehet tenni, ha nem tudunk függvényt írni, de mégis először a feltételt szeretnénk kódolni?

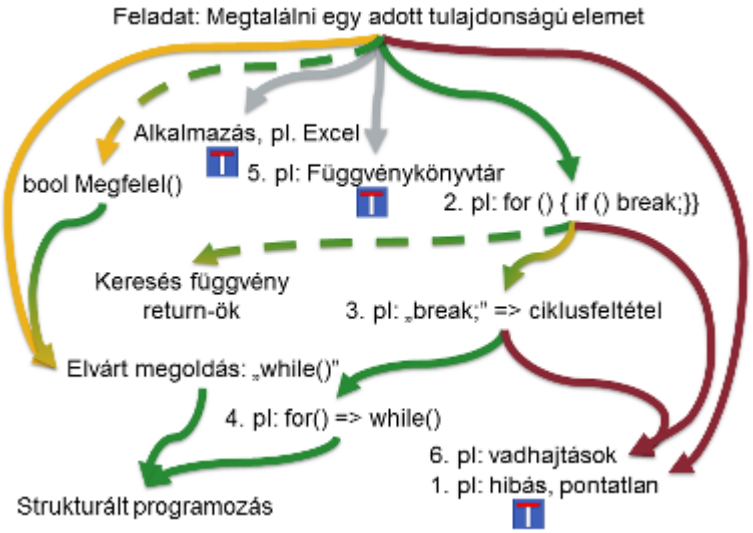
1. Megírjuk a feltételt – ilyet keresünk.
2. Köré írjuk a ciklusfejet és tagadjuk a feltételt – amíg meg nem találjuk
3. Kiegészítjük a ciklust megfelelő indexeléssel, léptetéssel
4. Értékeljük az eredményt

Nem valószínű, hogy bárhol ezt a megoldási módot tanítanák. Bár a kódot nem fogalmazásként, sorfolytonosan írjuk, a ciklusfej belsejével kezdeni a kód írását… nem szokás. Jellemző, hogy elkezdjük írni a ciklust, majd utána gondolkodunk, hogy mi legyen a feltétel…

A kör itt bezárult, visszajutunk ahhoz a megoldáshoz, amit a diákok adtak. Az „addig keresem, amíg nem megfelelő, amit vizsgálok éppen" kódolásának megtervezése túl nehéz tervezési lépés.

4 Merre, hogyan a kisebb–nagyobb buktatók és veszélyek között?

Összefoglalva lineáris keresés (eldöntés) kódolására talált megoldásokat, az alábbi gondolkodási térképet vázolhatjuk fel:



Lineáris keresés/eldöntés feladat megoldási módszerei

1. A közoktatásban (valamint ELTE IK-n) tanító tanárok által preferált megoldási mód a diákok, hallgatók számára túl nehéz, a tanulás időszakában a könnyítést jelentő Megfelel() segéd függvény nem szerepel az előzetes ismeretek között.
2. A diákok által választott és a BME-VIK-en oktatott módszer több lépésből áll, amelyek között van nehezebben megtanulható megoldás is, de számos veszélyt rejt az „ösztönös" kódolás.

E két módszer egymás problémáit kiküszöböli. A 2. módszer nagyobb esélyt ad a sikerélményre, ezzel motivál a programozás tanulására, de nem feltétlenül a strukturáltabb megoldás felé orientálja a tanulót. Ezért a lineáris keresés oktatása során célszerűbb ezt az utat választani, de folyamatosan figyelni kell, hogy egyre igényesebb legyen a tanuló a kódjának olvashatósága az általános kódolási elvek figyelembevétele szempontjából.

3. A lineáris keresés/eldöntés nehézségét elsősorban, a – sokszor többszörösen – összetett logikai kifejezés megfogalmazása jelenti. Számos módszer létezik a kifejezés komplex megfogalmazásának elkerülésére, azonban ezzel a gondolkodás fejlesztését kerüljük ki. Ezért a tanítási gyakorlatban a magasszintű függvénykönyvtárak, alkalmazások kevésbé hatékonyak.
4. A keresés másik buktatója az eredmény komplex megfogalmazása, ami elkerülhető, ha a feladatra függvényt írunk, amely az egyes esetek teljesülésekor az eredményt azonnal visszaadja. A függvény alkalmazása elegáns helyettesítése lenne a break-nek, de a paraméterek helyes használata gondot jelent.

Két esetben is megoldást jelentene a függvények alkalmazása, amely egyben a programtervezési elveknek is megfelelne. Megfontolandó, az algoritmusok előtt, már az első kódok írásakor tanítani a függvények egyszerű alkalmazását. A paraméterek érték szerinti átadásának alkalmazását jól előkészíti, ha programozás előtt táblázatkezelés tanulása megfelelő szinten történik, már akkor ismertté válnak a keresés függvényei.

Irodalom

1 Szlávi P. és Zsakó L.: *Methods of teaching programming.* In: Teaching mathematics and Computer Science, vol. 01(2), pp. 247–258., 2003. DOI: 10.5485/TMCS.2003

2 Szalayné Tahy Zs. és Czirkos Z.: *"ProgAlap" és ami mögötte van*. 8. InfoDIDACT, Zamárdi: Webdidaktika Alapítvány, 2015. ISBN: 978 963123892 1

3 Dijkstra, E. W.: *Letters to the Editor: Go to Statement Considered Harmful*. In: Communications of the ACM, vol. 11, no. 3, pp. 147–148, 1968. DOI: 10.1145/362929.362947

4 Martin, R. C.: *Clean Code: A Handbook of Agile Software Craftsmanship*. NJ, USA, Prentice Hall PTR, 2008. ISBN: 978 013235088 4

5 CAS Community: *Using break and continue in loops*. Computing at School, 06 2014. [Online]: http://community.computingatschool.org.uk/forums/63/topics/2835 (utoljára megtekintve: 2016.06.28.)

6 Informatika tanári levelező lista: *Kiugrás feltételes ciklusból*. Sulinet, 04 2016. [Online]: http://lista.sulinet.hu/mailman/private/informatika/2016-April/021186.html. (utoljára megtekintve: 2016.06.28.)

7 Informatika tanári levelező lista: *Programozás eldöntés*. Sulinet, 05 2016. [Online]: http://lista.sulinet.hu/mailman/private/informatika/2016-May/021668.html. (utoljára megtekintve: 2016.06.28.)

8 Szalayné Tahy Zs. és Czirkos Z.: *Linear search - the breaks in teching-learning practice* In: XXIX. DidMatTech 2016, „New methods and technologies in education and practice" Budapest (2016). 63-71. (ISBN: 978-963-284-816-7)

9 Gregorits T.: *Force of Summation* In: Teaching Mathematics and Computer Science, Debrecen, 12/2 (2014), 185-199

10 Czirkos Z.: *Programozási tételek*. BME EET, Budapest, 2016. [Online]: https://infoc.eet.bme.hu/ea03/ (utoljára megtekintve: 2016.11.03.)

11 Meyer, B.: *Object-oriented Software Construction (2Nd Ed.),* NJ, USA: Prentice-Hall, Inc., 1887. ISBN: 0-13-629155-4

12 Papp G., Horváth G., Szlávi P. és Zsakó L.: *Programozási alapismeretek 2. előadás*. ELTE IK, Budapest, 2015. [Online]: http://progalap.inf.elte.hu/downloads/eloadas/progalap_ea2.zip (utoljára megtekintve: 2016.11.03.)