Backtrack-es feladat-variációk

Menyhárt László [1], Zsakó László [2]

{[1]menyhart, [2]zsako }@caesar.elte.hu
ELTE IK

Absztrakt. Ezt a cikket olyan informatika tanároknak írtuk, akik az óráikon, házifeladatnak, esetleg versenyeken backtrack-es feladatokat szeretnének kiadni. Bemutatunk pár módszert, amivel egy egyszerűbb feladatból bonyolultabb probléma generálható, melynek így komplexebb, ötletet igénylő megoldása lesz. Egy példából kiindulva egyre nehezedő feladat-variációkon vezetjük végig az olvasót.

Kulcsszavak: oktatás, feladat, variáció, backtrack

Bevezetés

Jelen cikkünk olyan informatika tanároknak szól, akik különböző nehézségű, visszalépéses kereséssel megoldható feladatokat szeretnének kiadni megoldandó feladatként óráikon, házifeladatnak, vagy versenyeken. Szükség lehet ilyen átalakításokra, ha nem szeretnénk ugyanazt a feladatot megoldatni különböző képességű csoportokkal, egy adott évben különböző osztályokkal vagy ha nem szeretnénk félévente-évente ismét számon kérni ugyanazt. Versenyeken a korosztályok különböző nehézségi szintű feladatai így is eltérhetnek.

Egy konkrét példát és annak variációit elemezzük végig, majd ezekből általánosan használható módszereket fogalmazunk meg.

A visszalépéses keresés (backtrack) a problémamegoldás igen széles területén alkalmazható algoritmus, amelynek lényege a feladat megoldásának megközelítése rendszeres próbálgatással. Néha ez a legjobb megoldás! Ennek ellenére az algoritmusokról szóló könyvek jelentős része nem foglalkozik a visszalépéses kereséssel. Néhányban található egy-egy példa backtrack-re a nyolc vezér, fa bejárás, játékok, logikai formulák kiértékelése témakörökben [1][2]. Egyetemi tananyagokban már több helyen szerepel a 8 vezér, térképszínezés, solitaire, sudoku témakörökkel [3][4][5][6]. Az ELTe Informatikai karán a Mesterséges intelligencia tárgyban találkoznak vele a hallgatók [7]. Gyakoroltatáshoz kiadható rokon feladatokat azonban, mint amilyen feladat-sorozatot ebben a cikkben bemutatunk, egyik sem tartalmaz.

A visszalépéses keresés egy olyan általános módszer, mely az összes lehetséges eset kipróbálása (brute force) helyett egy ötlet felhasználásával nagyban csökkenti a lépésszámot. Megfelelő adatábrázolásra és ügyes feltételek megfogalmazására van szükség.

Adott N sorozat, amelyek rendre M[1], M[2], ... M[N] elemszámúak, de előfordulhat, hogy azonos elemszámúak (M). Ki kell választani mindegyikből egy-egy elemet úgy, hogy az egyes sorozatokból való választások másokat befolyásolnak. Ez egy bonyolult keresési feladat, amelyben egy adott tulajdonsággal rendelkező szám N-est kell megadni úgy, hogy ne kelljen az összes lehetőséget végignézni.

E feladatok közös jellemzője, hogy eredményük egy sorozat. E sorozat minden egyes tagját valamilyen sorozatból kell kikeresni, de az egyes keresések összefüggenek egymással (például a vezért nem lehet oda tenni, ahol egy korábban letett vezér ütné; egy munkát nem lehet két munkásnak adni; ha egy pékségnek elfogyott a kenyere, akkor attól már nem lehet rendelni).

- A visszalépéses keresés olyan esetekben használható, amikor a keresési tér fastruktúraként képzelhető el, amiben a gyökérből kiindulva egy csúcsot keresünk.

- Az algoritmus lényege, hogy a kezdőpontból kiindulva megtesz egy utat a feladatot részproblémákra bontva, és ha valahol az derül ki, hogy már nem juthat el a célig, akkor visszalép egy korábbi döntési ponthoz, és ott más utat – más részproblémát választ.

Először megpróbálunk az első sorozatból kiválasztani egy elemet, ezután a következőből, s ezt addig csináljuk, amíg választás lehetséges. Y[i] jelölje az i. sorozatból kiválasztott elem sorszámát! Ha még nem választottuk, akkor értéke 0 lesz. Ha nincs jó választás, akkor visszalépünk az előző sorozathoz, s megpróbálunk abból egy másik elemet választani. Visszalépésnél természetesen törölni kell a választást abból a sorozatból, amelyikből visszalépünk. Az eljárás akkor ér véget, ha minden sorozatból sikerült választani, vagy pedig a visszalépések sokasága után már az első sorozatból sem lehet újabb elemet választani (ekkor a feladatnak nincs megoldása).

```
Keresés(N,Van,Y):
  i:=1; Y[]:=[0,…,0]
  Ciklus amíg i³1 és i≤N  {lehet még és nincs még kész}
    Jóesetkeresés(i,Van,j)
    Ha Van akkor Y[i]:=j; i:=i+1  {előrelépés}
        különben Y[i]:=0; i:=i-1  {visszalépés}
  Ciklus vége
  Van:=(i>N)
Eljárás vége.
```

Egy lineáris keresés kezdődik az i. sorozatban:

```
Jóesetkeresés(i,Van,j):
  j:=Y[i]+1
  Ciklus amíg j≤M[i] és (rossz(i,j) vagy tilos(j))
    j:=j+1
  Ciklus vége
  Van:=(j≤M[i])
Eljárás vége.
```

Megjegyzés: az i-edik lépésben a j-edik döntési út nem választható, ha az előzőek miatt rossz, vagy ha önmagában rossz.

Megállapíthatjuk tehát, hogy minden egyes új választás az összes korábbitól függhet (ezt formalizálja az rossz függvény), a későbbiektől azonban nem! Egyes esetekben nemcsak a korábbiaktól, hanem saját jellemzőjétől is függhet a választás (ezt írja le az tilos függvény).

```
rossz(i,j):                {1. változat}
  k:=1
  Ciklus amíg k<i és szabad(i,j,k,Y[k])
    k:=k+1
  Ciklus vége
  rossz:=(k<i)
Függvény vége.
```

Amikor az összes lehetséges megoldást végig kell nézni, akkor a visszalépést és a keresés folytatását rekurzív függvényhívással a legegyszerűbb megoldani. Itt viszont a sok függvényhívás miatt a memóriahasználat megnő, nagymennyiségű adat esetén problémás lehet.

```
Összes_megoldás(i,N,Db,Y,x):
  Ha i>N akkor Db:=Db+1; Y(Db):=x
  különben Ciklus j=1-től N-ig
             Ha nem rossz(i,j) és nem tilos(j)
                akkor x[i]:=j
                      Összes_megoldás(i+1,N,Db,Y,x)
           Ciklus vége
  Elágazás vége
Eljárás vége.
```

Feladat-variációk

Az alapfeladat

Egy vállalkozás `N` különböző állásra keres munkásokat (`M`). A jelentkezők *bizonyos* információkat osztanak meg a jelentkezéskor. A feladat annak a meghatározása, hogy ki melyik állást töltse be.

1. variáció

Az `N` állásra pontosan `N` jelentkező érkezett, ahol minden jelentkező megmondta, hogy mely munkákhoz ért, illetve amihez ért, arra mennyi fizetést kérne.

N állás, N jelentkező, összegeket adtak meg, mindenkit fel lehet venni, legolcsóbb kell.

Legyen `F[i,j]` értéke 0, ha az `i.` jelentkező a `j.` munkához nem ért, illetve `F[i,j]=A>0`, ha ért hozzá és `A` forintot szeretne kapni érte.

A vállalkozás vezetője azt szeretné, ha az összes állást betöltenék, de úgy, hogy számára ez a lehető legkisebb költséggel járna.

A következőt gondolta ki: első lépésként állítsuk elő az összes lehetséges állásbetöltést (ha van egyáltalán olyan).

Állások:	1.	2.	3.	4.	5.
1. jelentkező:	100	0	0	**100**	0
2. jelentkező:	0	**200**	0	0	0
3. jelentkező:	**200**	100	0	0	0
4. jelentkező:	0	0	**200**	0	400
5. jelentkező:	500	0	400	0	**200**

Ez azt jelenti, hogy minden jelentkezőhöz hozzárendelünk egy állás sorszámot két feltétellel:

- olyan állást választhatunk számára, amihez ért (`F(i,j)>0`);
- olyan állást választhatunk számára, amit még nem adtunk másnak (`Volt(i,j,x)` függvény értéke hamis).

```
Megoldás(N,F,Db,Y):
  Összes_állás(1,N,F,Db,Y,x)
Eljárás vége.
```

Az `Összes_állás` eljárás első paramétere az aktuálisan vizsgált jelentkező `i` sorszáma legyen, a második paramétere pedig a jelentkezők (és egyben állások) `N` száma!

A megoldásban használt fontos változók:

- `x` – az aktuálisan számolt megoldás: `x[i]` az `i.` jelentkezőnek adott munka sorszáma
- `Db` – a megoldások száma;
- `Y` – a megoldásokat tartalmazó vektor.

```
Összes_állás(i,N,F,Db,Y,x):
  Ha i>N akkor Db:=Db+1; Y[Db]:=x
  különben Ciklus j=1-től N-ig
             Ha nem Volt(i,j,x) és F[i,j]>0
                akkor x[i]:=j; Összes_állás(i+1,N,F,Db,Y,x)
           Ciklus vége
  Elágazás vége
Eljárás vége.

Volt(i,j,x):
  k:=1
  Ciklus amíg k<i és x[k]≠j
    k:=k+1
  Ciklus vége
  Volt:=(k<i)
Függvény vége.
```

Ezután nincs más hátra, mint az `Y` vektorban összegyűlt megoldásokból kiválasztani a vállalkozó számára leggazdaságosabbat. Nagyon hamar kiderülhet azonban, hogy túlságosan sok elem lehet az `Y` vektorban.

A megoldási ötlet: felesleges az összes lehetséges megoldást tárolni, elég csupán minden lépés után a leggazdaságosabbat.

A megoldásban használt fontos változók:

- `x` – az aktuálisan számolt megoldás: `X[i]` az `i`. jelentkezőnek adott munka sorszáma
- `Y` – a legjobb megoldás.

```
Legjobb_állás(i,N,F,Y,x):
  Ha i>N akkor Ha Költség(N,F,x)<Költség(N,F,Y) akkor Y:=x
               Elágazás vége
  különben Ciklus j=1-től N-ig
             Ha nem Volt(i,j,x) és F[i,j]>0
                akkor x[i]:=j; Legjobb_állás(i+1,N,F,Y,x)
           Ciklus vége
  Elágazás vége
Eljárás vége.

Költség(N,F,x):
  s:=0
  Ciklus i=1-től N-ig
    s:=s+F[i,x[i]]
  Ciklus vége
  Költség:=s
Függvény vége.
```

Itt egy kicsi probléma léphet fel: az első megoldást mivel hasonlítjuk? Vegyünk fel egy új változót, ami az eddigi legjobb megoldás költségét tartalmazza! Állítsuk ennek az értékét a program elején a legnagyobb egész számra! Ha egy megoldást találunk, akkor az ennél biztosan jobb lesz, azaz lecserélhetjük rá.

```
Megoldás(N,F,Maxkölt,Y):
  Maxkölt:=+∞
  Legjobb_állás(1,N,F,Maxkölt,Y,x)
Eljárás vége.
```

A megoldásban használt fontos változók:

- `x` – az aktuálisan számolt megoldás: `x[i]` az `i`. jelentkezőnek adott munka sorszáma
- `Maxkölt` – az eddigi legjobb megoldás költsége;
- `Y` – a legjobb megoldás.

```
Legjobb_állás(i,N,F,Maxkölt,Y,x):
  Ha i>N akkor Ha Költség(N,F,x)<Maxkölt
                  akkor Y:=x; Maxkölt:=Költség(N,F,x)
               Elágazás vége
  különben Ciklus j=1-től N-ig
             Ha nem Volt(i,j,x) és F[i,j]>0
             akkor x[i]:=j; Legjobb_állás(i+1,N,F,Maxkölt,Y,x)
           Ciklus vége
  Elágazás vége
Eljárás vége.
```

Előfordulhat természetesen, hogy a feladatnak egyáltalán nincs megoldása, azaz legjobb megoldás sincs:

Már csak egy apróságra gondolhatunk: ha van egy megoldásunk és a most készülő megoldásról látszik, hogy már biztosan rosszabb lesz – többe fog kerülni –, akkor azt már nem érdemes tovább vinni.

Állások:	1.	2.	3.	4.	5.
1. jelentkező:	0	0	100	**100**	0
2. jelentkező:	0	200	0	0	**100**
3. jelentkező:	200	**100**	0	0	0
4. jelentkező:	0	0	**200**	400	0
5. jelentkező:	0	0	400	200	0

Legyen az eljárás paramétere az eddigi költség, s az eljárást csak akkor folytassuk, ha még nem érjük el a korábban kiszámolt maximális költséget. Emiatt nem a megoldások elkészültekor kell számolni költséget, hanem menet közben, folyamatosan.

A megoldást is módosítanunk kell, a `Legjobb_állás` eljárásnak új paramétere lesz, az aktuális előtti választások `költ` költsége.

A megoldásban használt fontos változók:

- `x` – az aktuálisan számolt megoldás: `x[i]` az `i`. jelentkezőnek adott munka sorszáma
- `költ` – az aktuálisan számolt megoldás költsége;
- `Maxkölt` – az eddigi legjobb megoldás költsége;

- `Y` – a legjobb megoldás.

```
Megoldás(N,F,Maxkölt,Y):
  Maxkölt:=+∞
  Legjobb állás(1,0,N,F,Maxkölt,Y,x)
Eljárás vége.

Legjobb_állás(i,költ,N,F,Maxkölt,Y,x):
  Ha i>N akkor Ha költ<Maxkölt akkor Y:=x; Maxkölt:=költ
               Elágazás vége
  különben Ciklus j=1-től N-ig
             Ha nem Volt(i,j,x) és F[i,j]>0
                és költ+F[i,j]<Maxkölt
                akkor x[i]:=j
                      Legjobb_állás(i+1,költ+F[i,j],
                                    N,F,Maxkölt,Y,x)
    Ciklus vége
  Elágazás vége
Eljárás vége.
```

2. variáció

Egy vállalkozás N különböző állásra keres munkásokat. A hirdetésre M jelentkező érkezett (M<N), ahol minden jelentkező megmondta, hogy mely munkákhoz ért, illetve amihez ért, arra mennyi fizetést kérne.

N állás, M(<N) jelentkező, összeget mond, mindenkit fel lehet venni, legolcsóbb kell.

Legyen `F(i,j)` értéke 0, ha az `i`. jelentkező a `j`. munkához nem ért, illetve `F(i,j)=A>0`, ha ért hozzá és A forintot szeretne kapni érte.

A vállalkozás vezetője azt szeretné, ha az összes jelentkezőt fel tudná venni, de úgy, hogy számára ez a lehető legkisebb költséggel járna.

Állások:	1.	2.	3.	4.	5.
1. jelentkező:	100	0	0	**100**	0
2. jelentkező:	0	**200**	0	0	0
3. jelentkező:	**200**	100	0	0	0
4. jelentkező:	0	0	**200**	0	400

A megoldás nagyon hasonló az előzőhöz: itt nem akkor van kész egy megoldás, ha N jelentkezőnek adtunk munkát, hanem akkor, ha az M jelentkezőnek adtunk munkát:

```
Megoldás(N,M,F,Maxkölt,Y):
  Maxkölt:=+∞
  Legjobb állás(1,0,N,M,F,Maxkölt,Y,x)
Eljárás vége.
```

```
Legjobb_állás(i,költ,N,M,F,Maxköltt,Y,x):
  Ha i>M akkor Ha költ<Maxköltt akkor Y:=x; Maxköltt:=költ
               Elágazás vége
  különben Ciklus j=1-től N-ig
             Ha nem Volt(i,j,x) és F[i,j]>0
                és költ+F[i,j]<Maxköltt
                akkor x[i]:=j
                      Legjobb_állás(i+1,költ+F[i,j],
                                    N,M,F,Maxköltt,Y,x)
           Ciklus vége
  Elágazás vége
Eljárás vége.
```

3. variáció

Egy vállalkozás N különböző állásra keres munkásokat. A hirdetésre M jelentkező érkezett (M>N), ahol minden jelentkező megmondta, hogy mely munkákhoz ért, illetve amihez ért, arra mennyi fizetést kérne.

N állás, M(>N) jelentkező, összeget mond, mindenkit fel lehet venni, legolcsóbb kell.

Legyen `F(i,j)` értéke 0, ha az `i`. jelentkező a `j`. munkához nem ért, illetve `F(i,j)=A>0`, ha ért hozzá és `A` forintot szeretne kapni érte.

A megoldási ötlet: ne jelentkezőhöz keressünk állást, hanem álláshoz jelentkezőt! Így a feladat megoldása az előzőével majdnem megegyezik, csupán a két index szerepét kell felcserélni.

Állások:	1.	2.	3.	4.
1. jelentkező:	100	0	0	**100**
2. jelentkező:	0	**200**	0	0
3. jelentkező:	**200**	100	0	0
4. jelentkező:	0	0	**200**	0
5. jelentkező:	500	0	400	0

```
Legjobb_állás(i,költ,N,M,F,Maxköltt,Y,x):
  Ha i>N akkor Ha költ<Maxköltt akkor Y:=x; Maxköltt:=költ
               Elágazás vége
  különben Ciklus j=1-től M-ig
             Ha nem Volt(j,i,x) és F[j,i]>0
                és költ+F[j,i]<Maxköltt
                akkor x[j]:=i
                      Legjobb_állás(i+1,költ+F[i,j],
                                    N,M,F,Maxköltt,Y,x)
           Ciklus vége
  Elágazás vége
Eljárás vége.
```

4. variáció

Egy vállalkozás N különböző állásra keres munkásokat. Pontosan N jelentkező érkezett, ahol minden jelentkező megmondta, hogy mely munkákhoz ért, illetve amihez ért, arra mennyi fizetést kérne.

N állás, N jelentkező, összeget mond, nem lehet mindenkit felvenni, legolcsóbb kell.

Legyen `F(i,j)` értéke 0, ha az `i`. jelentkező a `j`. munkához nem ért, illetve `F(i,j)=A>0`, ha ért hozzá és `A` forintot szeretne kapni érte.

A vállalkozás vezetője azt szeretné, ha az összes állást betöltenék, de úgy, hogy számára ez a lehető legkisebb költséggel járna.

Nem biztos azonban, hogy ez lehetséges. Akkor is érdekes lehet azonban egy olyan megoldás, amiben a lehető legtöbb munkát adhatjuk ki.

Állások:	1.	2.	3.	4.	5.
1. jelentkező:	100	0	0	**100**	0
2. jelentkező:	0	**200**	0	0	0
3. jelentkező:	**200**	100	0	0	0
4. jelentkező:	0	0	**200**	0	400
5. jelentkező:	500	400	0	0	0

A megoldás ötlete: Vezessünk be egy N+1. ún. fiktív állást, amihez azt gondoljuk, hogy mindenki ért! Legyen ennek az ára nagyobb, mint minden más összeg a táblázatunkban! Az N+1. állásra engedjük meg, hogy többen is válasszák!

Belátható, hogy a leggazdaságosabb megoldásban ekkor a lehető legkevesebb fiktív állás lesz, azaz a legtöbb állást tudjuk betölteni.

Állások:	1.	2.	3.	4.	5.	6.
1. jelentkező:	100	0	0	**100**	0	1000
2. jelentkező:	0	**200**	0	0	0	1000
3. jelentkező:	**200**	100	0	0	0	1000
4. jelentkező:	0	0	**200**	0	400	1000
5. jelentkező:	500	400	0	0	0	**1000**

```
Legjobb_állás(i,költ,N,F,maxért,Maxköltl,Y,x):
  Ha i>N akkor Ha költ+maxért<Maxköltl
                  akkor Y:=x; Maxköltl:=költ+maxért
               Elágazás vége
  különben Ciklus j=1-től N-ig
             Ha nem Volt(i,j,x) és F[i,j]>0
                és költ+F[i,j]<Maxköltl
                akkor x[i]:=j
                      Legjobb_állás(i+1,költ+F[i,j],
                                    N,F,maxért,Maxköltl,Y,x)
           Ciklus vége
  Elágazás vége
Eljárás vége.
```

5. variáció

Egy vállalkozás `N` különböző állásra keres munkásokat. Pontosan `N` jelentkező érkezett, ahol minden jelentkező megmondta, hogy mely munkákhoz ért.

N állás, N jelentkező, nem mond összeget (csak ért-e hozzá), mindenkit fel lehet venni

Legyen `F[i,j]` értéke `hamis`, ha az `i.` jelentkező a `j.` munkához nem ért, illetve `igaz`, ha ért hozzá.

Állások:	1.	2.	3.	4.	5.
1. jelentkező:	igaz	hamis	hamis	igaz	hamis
2. jelentkező:	hamis	igaz	hamis	hamis	hamis
3. jelentkező:	igaz	igaz	hamis	hamis	hamis
4. jelentkező:	hamis	hamis	igaz	hamis	hamis
5. jelentkező:	igaz	hamis	igaz	hamis	igaz

```
Munkák(N,F,Van,Y):
  i:=1; Y[]:=[0,…,0]
  Ciklus amíg i≥1 és i≤N  {lehet még és nincs még kész}
    Jóesetkeresés(i,F,Y,Van,j)
    Ha Van akkor Y[i]:=j; i:=i+1  {előrelépés}
        különben Y[i]:=0; i:=i-1  {visszalépés}
  Ciklus vége
  Van:=(i>N)
Eljárás vége.
```

Jól látható, hogy a fő eljárásban konkrét feladat miatt semmi teendőnk nincs, egyszerűen csak lemásoljuk az általános sémát.

```
Jóesetkeresés(i,F,Y,Van,j):
  j:=Y[i]+1
  Ciklus amíg j≤N és (rossz(i,j,Y) vagy nem F[i,j])
    j:=j+1
  Ciklus vége
  Van:=(j≤N)
Eljárás vége.
```

A második szinten egyszerűsítenünk kellett, `M[i]` helyébe `N` került, a `tilos(j)` függvényt pedig egy mátrix elemre hivatkozással helyettesítjük.

```
rossz(i,j,Y):
  k:=1
  Ciklus amíg k<i és Y[k]≠j
    k:=k+1
  Ciklus vége
  rossz:=(k<i)
Függvény vége.
```

A harmadik szinten semmi teendőnk nem volt.

6. variáció

Egy vállalkozás `N` különböző állásra keres munkásokat. Pontosan `N` jelentkező érkezett, ahol minden jelentkező megmondta, hogy mely munkákhoz ért.

N állás, N jelentkező, nem mond összeget (csak ért-e hozzá), mindenkit fel lehet venni. Módosított adatábrázolással.

Legyen `D[i]` az i. ember által vállalható munkák száma, `E[i,j]` értéke pedig az általa vállalt `j.` munka sorszáma!

A vállalkozás vezetője azt szeretné, ha az összes jelentkezőt fel tudná venni és minden munkát elvégeztetni.

	Darab
1. jelentkező:	2
2. jelentkező:	1
3. jelentkező:	2
4. jelentkező:	1
5. jelentkező:	3

Állások:	1.	2.	3.
	1	4	
	2		
	1	2	
	3		
	1	3	5

A megoldásban használt fontos változók:

- `N` – a munkák és a munkások száma
- `D` – a munkások hány munkához értenek (`D[i]` – az `i`. munkás ennyi munkához ért)
- `E` – az adott munkások mely munkákhoz értenek (`E[i,j]` – az `i`. munkás által elvégezhető `j`. munka)
- `Y` – az aktuálisan számolt megoldás: `Y[i]` az `i`. jelentkezőnek adott munka sorszáma

```
Munkák(N,D,E,Van,Y):
  i:=1; Y[]:=[0,…,0]
  Ciklus amíg i≥1 és i≤N  {lehet még és nincs még kész}
    Jóesetkeresés(i,D,Y,E,Van,j)
    Ha Van akkor Y[i]:=j; i:=i+1  {előrelépés}
        különben Y[i]:=0; i:=i-1  {visszalépés}
  Ciklus vége
  Van:=(i>N)
  Ha Van akkor Ciklus i=1-től N-ig
                 Y[i]:=E(i,Y[i])
               Ciklus vége
Eljárás vége.
```

Jól látható, hogy a fő eljárásban konkrét feladat miatt majdnem semmi teendőnk nincs, egyszerűen csak lemásoljuk az általános sémát.

Az egyetlen módosítandó: mivel a megoldásban a `j` a választás sorszáma, emiatt a végén ebből elő kell állítanunk a munka sorszámát.

```
Jóesetkeresés(i,D,Y,E,Van,j):
  j:=Y[i]+1
  Ciklus amíg j≤D[i] és rossz(i,j,Y,E)
    j:=j+1
  Ciklus vége
  Van:=(j≤D[i])
Eljárás vége.
```

A második szinten egyszerűsítenünk kellett, nincs szükség a `tilos(j)` függvényre.

```
rossz(i,j,Y,E):
  k:=1
  Ciklus amíg k<i és E[k,Y[k]]≠E[i,j]
    k:=k+1
  Ciklus vége
  rossz:=(k<i)
Függvény vége.
```

Mivel j és Y[k] sem munka sorszám, hanem adott munkás munka felsorolásbeli sorszáma, ezért a hasonlítás kissé bonyolultabb lett.

7. Variáció

Egy vállalkozás N különböző állásra keres munkásokat. Pontosan N jelentkező érkezett, ahol minden jelentkező megmondta, hogy mely munkákhoz ért, illetve amihez ért, arra mennyi fizetést kérne. A vállalkozás vezetője azt szeretné, ha az összes jelentkezőt fel tudná venni és minden munkát elvégeztetni, de úgy, hogy számára ez legfeljebb X összegbe kerüljön.

N állás, N jelentkező, összeget mond, mindenkit fel lehet venni, egy megoldás kell, de meghatározunk egy maximális fedezeti összeget

Legyen `F[i,j]` értéke `=0`, ha az `i`. jelentkező a j. munkához nem ért, illetve `>0`, ha ért hozzá, és akkor ez az érték jelentse azt, hogy a munkát ennyiért végezné el.

Ha X=1300, akkor a példában vastagon szedett megoldás helyett a dőlten szedett megoldás is helyes.

Állások:	1.	2.	3.	4.	5.
1. jelentkező:	100	0	0	***100***	0
2. jelentkező:	0	***200***	0	0	0
3. jelentkező:	***200***	100	0	0	0
4. jelentkező:	0	0	**200**	0	*400*
5. jelentkező:	500	0	*400*	0	**200**

A példában az is látszik, hogy bármely munkás megbízásával a költség folyamatosan emelkedik, amit a megoldásban ki is használunk.

A megoldásban használt fontos változók:

- N – a munkák és a munkások száma
- F – az adott munkások mely munkákhoz értenek (F[i,j] – az i. munkás által elvégezhető j. munka ára)
- `Y` – az aktuálisan számolt megoldás: `Y[i]` az `i`. jelentkezőnek adott munka sorszáma
- `X` – a rendelkezésre álló összeg

```
Munkák(N,X,F,Van,Y):
  i:=1; Y[]:=[0,…,0]
  Ciklus amíg i≥1 és i≤N {lehet még és nincs még kész}
    Jóesetkeresés(i,Van,j)
    Ha Van akkor Ha Költség(N,i,j,F,Y)≤X
                    akkor Y[i]:=j; i:=i+1  {előrelépés}
                 különben Y[i]:=0; i:=i-1  {visszalépés}
         különben Y[i]:=0; i:=i-1           {visszalépés}
  Ciklus vége
  Van:=(i>N)
Eljárás vége.
```

Jól látható, hogy a fő eljárásban kell figyelembe venni a költség függvényt: ha az adott lépésig a költség nem haladja meg az X értéket, akkor a megtalált munka hozzárendelés jó, ha meghaladja, akkor pedig rossz, visszalépést okoz.

Megjegyzés: hatékonysági szempontból a két különben ág összevonható:

```
Munkák(N,Van,Y):
  i:=1; Y[]:=[0,…,0]
  Ciklus amíg i≥1 és i≤N {lehet még és nincs még kész}
    Jóesetkeresés(N,i,F,Y,Van,j)
    Ha Van és Költség(N,i,j,F,Y)≤X
                    akkor Y[i]:=j; i:=i+1  {előrelépés}
                 különben Y[i]:=0; i:=i-1  {visszalépés}
  Ciklus vége
  Van:=(i>N)
Eljárás vége.
```

A második szinten egyszerűsítenünk kellett, `M[i]` helyébe `N` került, a `tilos(j)` függvényt pedig egy mátrix elemre hivatkozással helyettesítjük.

```
Jóesetkeresés(N,i,F,Y,Van,j):
  j:=Y[i]+1
  Ciklus amíg j≤N és (Rossz(i,j,Y) vagy F[i,j]=0)
    j:=j+1
  Ciklus vége
  Van:=(j≤N)
Eljárás vége.
```

A Rossz függvénnyel pedig semmi tennivalónk nem lesz:

```
Rossz(i,j,Y):
  k:=1
  Ciklus amíg k<i és Y[k]≠j
    k:=k+1
  Ciklus vége
  Rossz:=(k<i)
Függvény vége.
```

A költségszámítás nagyon egyszerű, összegzés tétellel elvégezhető.

```
Költség(N,i,j,F,Y):
  s:=0
  Ciklus k=1-től i-1-ig
    s:=s+F[i,Y[k]]
  Ciklus vége
  s:=s+F[i,j]
  Költség:=s
Függvény vége.
```

A megoldásban a `Költség` függvény figyelembevétele volt a **korlátozás**, emiatt hamarabb lehetett szükség visszalépésre, ebből derült ki ugyanis, hogy már nem találhatunk jó megoldást.

A korlátozás lényege tehát: a teljes megoldás elkészülte előtt visszalépést okozni!

8. További variációk

Álljon itt pár példa megoldás nélkül, ami a feladat komplexitását változtatja.

- Minden állásra kell ember
- Van, aki több mindenhez ért, ilyen legyen a prioritás
- Több helyszínen vannak az állások, közelben lakók prioritás
- Ha nincs hozzáértő, jöhet olyan is, aki vállalja a betanulást
- Tanulás költségét is bevezethetjük.
- *Betanuló* fizetését plusz a *tanítás* költségét hasonlítsuk össze a *Szakértő* fizetésével

9. Az alapfeladat módosítása

További variálási lehetőség az alapfeladat módosítása. Erre pár lehetőség:

- Konferencián N idősávra M előadó jelentkezik.
- Iskolában N napon tanulmányi versenyek, M tanuló melyiken induljon
- N cég, M ellenőrt küldenek ki, de legfeljebb K helyre, lehetőleg közeli helyekre
- N témavezetőhöz M szakdolgozatos jelentkezik. Osszuk be őket attól függően, hogy ki melyik témához ért, de maximum K-an lehetnek egy témavezetőnél.
- N üzletben M termék, hol vásároljunk, hogy legjobban járjunk. Bevezethetünk az ár mellé szállítási és/vagy utazási költségeket is.

Módszerek

A bemutatott feladatban az állások számát és a jelentkezők számát változtattuk, módosítottunk a megkapott információkon aszerint, hogy tudjuk-e a fizetési igényeket. Úgy változtattuk az adatokat, hogy a legjobb megoldást kellett megtalálnunk vagy egy bármilyet, illetve korlátoztuk az adatokat.

Általánosságban tehát elmondhatjuk, hogy visszalépéses keresés alapjául szolgáló többdimenziós adatainak a nagysága (`N` és `M`), azok egymáshoz való viszonya szolgálhat a variáció elkészítésére.

A megadott adatok típusa, jelentése és egyéb kiegészítő, pontosító vagy esetleg új információ, mint például a korlát bevezetése újabb lehetőségeket ad egy másik változat elkészítéséhez.

Az alapfeladat cseréjével is egyszerűen módosítható a probléma, így további nagyszámú feladat-variáció-csomaghoz jutunk. Feladattól függetlenül megfogalmazhatjuk a következő kombinatorikai kérdéseket:

Összes ismétlés nélküli permutáció

Backtrack: $\forall i\ (1 \le i \le n)$: $\forall j\ (1 \le j < i)$: $X_j \neq X_i$ (alesete: olyan permutációk, ahol $X_i <> i$)

Összes ismétléses permutáció

Backtrack: $\forall i\ (1 \le i \le \Sigma L_i)$: $\forall j\ (1 \le j < i)$: $X_j = X_i$ legfeljebb L_i-szer

Például az egyik előző feladatban: ugyanarra a munkára több jelentkező is felvehető.

Összes ismétlés nélküli variáció

Backtrack: $\forall i\ (1 \le i \le k)$: $\forall j\ (1 \le j < i)$: $X_j \neq X_i$

Összes ismétlés nélküli kombináció

Backtrack: $\forall i\ (1 \le i \le k)$: $\forall j\ (1 \le j < i)$: $X_j < X_i$

Összes ismétléses kombináció

Backtrack: $\forall i\ (1 \le i \le k)$: $\forall j\ (1 \le j < i)$: $X_j \le X_i$

Összes partíció

Olyan K-jegyű számok, ahol a számjegyek összege pontosan N: $\forall i\ (1 \le i \le k)$: $X_j \ge 0$ és $\Sigma X_i = N$

Összes diszjunkt felbontás

N felbontása pozitív (>0) számok összegére: $\forall i\ (1 \le i \le m)$: $Xj > 0$ és $\Sigma Xi = N$

Ezen feladatok hatékony megoldása azonban egyes esetekben nem backtrack-es.

Összefoglalás

A módszer matematikai megalapozásával Fóthi Ákos és kollégái foglalkoztak [8], ehhez azonban a közoktattásban másképp kell hozzáállni.

Cikkünket azzal a céllal írtuk, hogy különböző komplexitású backtrack-es feladatok elkészítésének lehetőségeit felmérjük és bemutassuk. Egy példán keresztül bemutattuk, hogy egy alapfeladat módosításai milyen variációkra ad lehetőséget.

A felsorolt példák tanulságaiból általánosan megfogalmaztuk a módszereket, melyek segítségével még ennek a feladatnak a további variálására is lehetőség lenne. Sőt az alapfeladat módosításával, akár csak a téma lecserélésével is már egy másik problémát kap a feladatmegoldó. Fontos azonban megemlíteni, hogy a feladat módosítása után mindig ellenőrizzük vissza a megoldást, mert bizonyos esetekben előfordulhat, hogy a feladat megoldása oly mértékben eltér, hogy azt már nem is az eredeti módszer szerint kell megoldani. Például előfordulhat, hogy a backtrack helyett dinamikus programozásra lesz szükség, amit esetleg még nem ismernek a diákjaink.

Bízunk benne, hogy az itt bemutatott minták és módszerek segítik olvasóinkat későbbi munkájuk könnyebb elvégzésében, amikor különböző komplexitású backtrack-es feladatok kitűzését végzik.

Hivatkozások

1. Alexander Shen. Algorithms and Programming: Problems and Solutions, Springer, 2010
2. S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani: Algorithms. 2006
3. David G. Sullivan, Recursion and Recursive Backtracking, Harvard Extension School (2012) https://sites.fas.harvard.edu/~cscie119/lectures/recursion.pdf (utoljára megtekintve: 2018. október 31.)
4. Douglas Wilhelm Harder, Backtracink, University of Waterloo, Ontario, Canada, https://ece.uwaterloo.ca/~dwharder/aads/Algorithms/Backtracking/ (utoljára megtekintve: 2018. október 31.)
5. Steven Skiena, Analysis of Algorithms, Backtracing, Stony Brook University New York, (2017) https://www3.cs.stonybrook.edu/~skiena/373/newlectures/lecture15.pdf (utoljára megtekintve: 2018. október 31.)

6. J Zelenski: Exhaustive recursion and backtracking (2008) https://see.stanford.edu/materials/icspacs106b/h19-recbacktrackexamples.pdf (utoljára megtekintve: 2018. október 31.)
7. Lőrentey Károly; Fekete István; Fóthi Ákos; Gregorics Tibor: On the wide variety of backtracking algorithms. pp. 165-174. In: Kovács, Emőd; Winkler, Zoltán (szerk.) Proceedings of the 5th international conference on applied informatics : Education and other fields of applied informatics, computer graphics, computer statistics and modeling. Eger, Magyarország : Molnár és Társa 2001 Kft., (2001) p. 250.
8. Harangozó Éva; Nyékyné Gaizler Judit; Fóthi Ákos; Konczné Nagy Márta: Demonstration of a Problem-Solving Method ACTA CYBERNETICA 12 : 1 pp. 71-82. , 12 p. (1995)