ÚJ SZÉCHENYI TERV

# Ég és Föld vonzásában – a természet titkai

# Informatikai tehetséggondozás:

## Visszalépéses keresés

**TÁMOP-4.2.3.-12/1/KONV**

MAGYARORSZÁG MEGÚJUL

A projektek az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósulnak 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!

Adott N sorozat, amelyek rendre M(1), M(2), ... M(N) elemszámúak. 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 (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 rész-problémát választ.

Először megpróbálunk az első sorozatból kiválasztani egy elemet, ezután a következőből, s ezt addig csináljuk, amíg választás lehetséges. X(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).

```
Változók:
    N: Egész                    [a feldolgozandó sorozat száma]
    M: Tömb(1..N:Egész) [a feldolgozandó sorozatok elemszáma]
    VAN: Logikai                  [van-e megfelelő elemsorozat]
    Y: Tömb(1..N:Egész)           [a megfelelő elemek sorszáma]
```

A megoldás legfelső szintjén keressünk az i. sorozatból megfelelő elemet! Ha ez sikerült, akkor lépjünk tovább az i+1. sorozatra, ha pedig nem sikerült, akkor lépjünk vissza az i-1.-re, s keressünk abban újabb lehetséges elemet!

```
Keresés(N,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 Y(i):=j; i:=i+1  {előrelépés}
        különben Y(i):=0; i:=i-1  {visszalépés}
  Ciklus vége
  Van:=(i>N)
Eljárás vége.
```

A keresés befejeződik, ha mindegyik sorozatból sikerült elemet választanunk (i>N) vagy pedig a visszalépések miatt már az elsőből sem tudunk újabbat választani (i<1).

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.
```

Megjegyzés: Rossz egy választás, ha valamelyik korábbi választás miatt nem szabad – eldöntés.

```
rossz(i,j):                {2. változat}
  s:=F0
  Ciklus k=1-től i-1-ig
    s:=f(s,k,Y(k))
  Ciklus vége
  rossz:=nem szabad(s,i,j)
Függvény vége.
```

Megjegyzés: Rossz egy választás, ha a korábbiak összessége miatt nem szabad – sorozatszámítás, megszámolás, …

## N vezér elhelyezése egy NxN-es sakktáblán

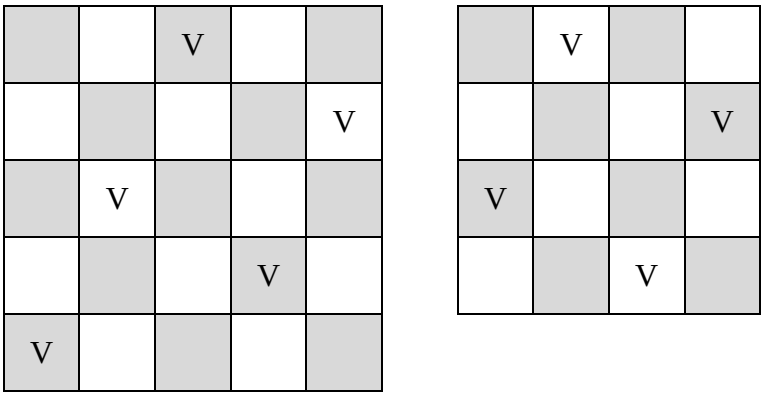
Helyezzünk el egy NxN-es sakktáblán N vezért úgy, hogy ne üssék egymást! Az összes lehetséges megoldásból nagyon sok is lehet, de egyetlen megoldás megtalálása várhatóan sokkal gyorsabb.

Megjegyzés: sok olyan feladat van – például N elem összes lehetséges sorrendjének előállítása – ahol az összes megoldás előállítására használjuk a visszalépéses keresés – pontosabban visszalépéses kiválogatás – algoritmusát, egyetlen megoldás keresésére azonban nincs rá szükség.

A vezérekről azt kell tudnunk, hogy a sorukban, az oszlopukban és az átlójukban álló bábukat üthetik. Tehát úgy kell elhelyezni a vezéreket, hogy minden sorban és minden oszlopban is pontosan 1 vezér legyen, és minden átlóban legfeljebb 1 vezér legyen!

Ebből következik, hogy a megoldásokban elég azt megmondani, hogy az 1. oszlopban levő vezér melyik sorban áll, a második oszlopban levő vezér melyik sorban áll, … és így tovább.

Egy lehetséges megoldás N=5-re és N=4-re:




Itt tehát nem elég azt megnézni egy új vezér elhelyezésekor, hogy volt-e már a j. sorban vezér, hanem azt is kell, hogy volt-e már az (i,j) mezővel egy átlóban vezér.

```
Vezérek(N,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 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,Van,j):
  j:=Y(i)+1
  Ciklus amíg j≤N és rossz(i,j)
    j:=j+1
  Ciklus vége
  Van:=(j≤N)
Eljárás vége.
```

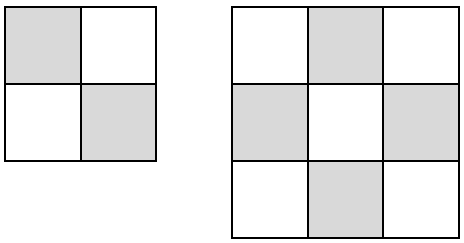
A második szinten egyszerűsítenünk kellett, `M(i)` helyébe `N` került, valamint nincs szükség a `tilos(j)` függvényre.

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

Ez az az eljárás, ahol ki kellett fejteni a `szabad(i,j,k,Y(k))` függvényt, de a többi része itt is változatlan.

Megjegyzés: A vezéres feladatnak N=2 és N=3 esetén biztosan nincs egy megoldása sem, más N-ek esetén pedig vannak megoldások, de nem túlságosan sok. A sakktábla szimmetria tulajdonságai miatt – a megoldások különböző tükörképei, 90 fokos elforgatásai is megoldások, kicsit hatékonyabb megoldást is készíthetnénk..

Példa: az alábbi sakktáblákra nem lehet elhelyezni 2, illetve 3 vezért úgy, hogy ne üssék egymást:



## 2. Munkásfelvétel: N állás – N jelentkező

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

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

| | Darab | Állások: | 1. | 2. | 3. |
|---|---|---|---|---|---|
| 1. jelentkező: | 2 | | 1 | **4** | |
| 2. jelentkező: | 1 | | **2** | | |
| 3. jelentkező: | 2 | | **1** | 2 | |
| 4. jelentkező: | 1 | | **3** | | |
| 5. jelentkező: | 3 | | 1 | 3 | **5** |

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

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,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 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):=Y(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,Van,j):
  j:=Y(i)+1
  Ciklus amíg j≤D(i) és rossz(i,j)
    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):
  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.

A vállalásokat másképp is ábrázolhatjuk. 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,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 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,Van,j):
  j:=Y(i)+1
  Ciklus amíg j≤N és (rossz(i,j) 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):
  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.

# Konferencia

Egy konferencián párhuzamosan ***K*** szekcióban tarthatnak előadást. Minden szekció ***L*** egymás utáni előadásból áll. Az előadásokat témák szerint csoportosították, összesen ***N*** darab témát neveztek meg. Teljesül, hogy ***K*L=az előadások száma.***

Készíts programot, amely úgy osztja a témákat szekciókba, hogy minden téma előadásai azonos szekcióban, egymás után legyenek! Tudjuk, hogy biztosan létezik megoldás.

A *KONF.BE* szöveges állomány első sorában a szekciók száma (***1≤K≤5***), az egymás utáni blokkok száma (***1≤L≤10***), valamint a témák száma (***1≤N≤50***) van, egy-egy szóközzel elválasztva. A következő ***N*** sorban az egyes témák neve (pontosan 5 karakteren), s tőle egy szóközzel elválasztva a témához tartozó előadások száma van.

A *KONF.KI* állományba pontosan ***K*** sort kell írni, az i-edik sorba az i-edik szekció témanevei kerüljenek, a beosztás sorrendjében, azaz pontosan ***L*** darab 5 karakteres név, egy-egy szóközzel elválasztva!

Példa:

KONF.BE

```
2 3 4
AAAAA 1
BBBBB 2
CCCCC 2
DDDDD 1
```

KONF.KI

```
BBBBB BBBBB AAAAA
CCCCC CCCCC DDDDD
```

A feladat visszalépéses kereséssel oldható meg.

A lehetséges gyorsítás miatt először rendezzük a szekciókat előadásszám szerint csökkenő sorrendbe! A nagy előadásszámú témák előrehozásával feltehetően nem lesz szükség azok helyének későbbi megváltoztatására, azaz a visszalépéses keresés nem lép vissza azokra a szintekre.

```
Konferencia(N,K,L,e,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 Y(i):=j; i:=i+1  {előrelépés}
        különben Y(i):=0; i:=i-1  {visszalépés}
  Ciklus vége
  Van:=(i>N)
Eljárás vége.
```

Ha tudjuk, hogy biztosan van megoldás, akkor az `i≥1` feltétel az első ciklusnál nem szükséges.

```
Konferencia(N,K,L,e,Y):
  i:=1; Y():=(0,…,0)
  Ciklus amíg i≤N  {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.
```

Meg kell keresni a K szekció közül azt a szekciót, ahova a téma blokkjai befférnek:

```
Jóesetkeresés(i,Van,j)
  j:=Y(i)+1
  Ciklus amíg j≤K és nem befér(i,j)
    j:=j+1
  Ciklus vége
Eljárás vége.
```

Egy téma akkor fér be egy szekcióba, ha a szekcióba tartozó blokkok száma nem haladja meg az L értékét.

```
befér(i,j):
  h:=e(i).db
  Ciklus c=1-től i-1-ig
    Ha x(c)=j akkor h:=h+e(c).db
  Ciklus vége
  befér:=(h≤L)
Függvény vége.
```

Ezután már csak az maradt hátra, hogy az Y vektor alapján előállítsuk a kért eredményt.