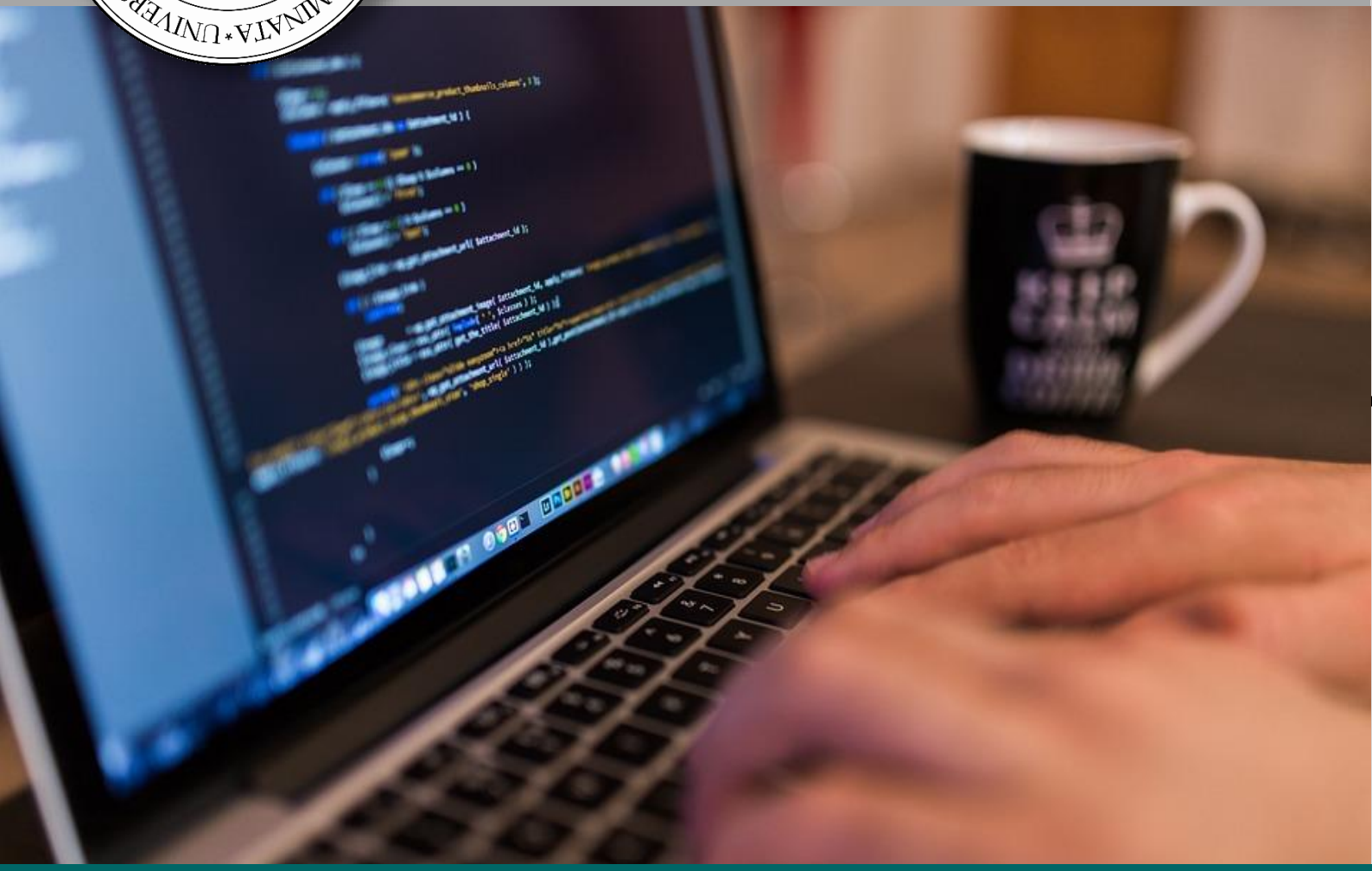
Belépő a tudás közösségébe

Informatika szakköri segédanyag



Visszalépéses keresés korlátozással

Heizlerné Bakonyi Viktória, Horváth Győző, Menyhárt László, Szlávi Péter, Törley Gábor, Zsakó László

Szerkesztő: Abonyi-Tóth Andor, Zsakó László

A kiadvány „A felsőoktatásba bekerülést elősegítő készségfejlesztő és kommunikációs programok megvalósítása, valamint az MTMI szakok népszerűsítése a felsőoktatásban” (EFOP-3.4.4-16-2017-006) című pályázat keretében készült 2017-ben.

Eötvös Loránd Tudományegyetem
Informatikai Kar

SZÉCHENYI 2020
MAGYARORSZÁG KORMÁNYA
Európai Unió
Európai Szociális Alap
BEFEKTETÉS A JÖVŐBE

Bevezető

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. 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).

```
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,M,Van,Y):
  i:=1; Y[]:=[0,…,0]
  Ciklus amíg i≥1 és i≤N {lehet még és nincs még kész}
    Jóesetkeresés(i,M,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.
```

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,M,Y,Van,j):
  j:=Y[i]+1
  Ciklus amíg j≤M[i] és (rossz(i,j,Y) 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,Y):                {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,Y):                {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, …

Egy másik lehetséges esetben nem tudjuk, hogy az eredmény hány elemű, csupán azt, hogy mikor van készen. Ezt a `kész(X)` logikai értékű függvény adja meg.

```
Keresés(N,M,Van,Y):
  i:=1; Y[1..N]:=[0,...,0]; DB:=0
  Ciklus amíg i≥1 és nem kész(Y)
       Jó_eset_keresés(M,Y,i,melyik,VAN)
       Ha VAN akkor  Y[i]:=melyik; i:=i+1   [előrelép]
            különben Y[i]:=0;      i:=i-1   [visszalép]
  Ciklus vége
  Van:=kész(Y)
Eljárás vége.
```

Ez a megoldás persze nem véd az ellen, hogy nagyon hosszú eredménysorozatokat állítsunk elő. Ilyenkor lesz szükség még egy lépésszám korlátra is.

```
Keresés(N,M,Korlát,Van,Y):
  i:=1; Y[1..N]:=[0,...,0]; DB:=0
  Ciklus amíg i≥1 és nem kész(Y) és i≤Korlát
       Jó_eset_keresés(M,Y,i,melyik,VAN)
       Ha VAN akkor  Y[i]:=melyik; i:=i+1   [előrelép]
            különben Y[i]:=0;      i:=i-1   [visszalép]
  Ciklus vége
  Van:=kész(Y)
Eljárás vége.
```

1. 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 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.

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.

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

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

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 végje
  s:=s+F[i,j]
  Költség:=s
Függvény vége.
```

A megoldásban a `Költség` függvény figyelembe vétele 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!

Minta kódok

C++	cpp/1_Munkasfelvetel/feladat.cpp
C#	cs/1_Munkasfelvetel/feladat.cs
Java	java/1_Munkasfelvetel/feladat.java
Pascal	pas/1_Munkasfelvetel/feladat.pas
Python	py/1_Munkasfelvetel/feladat.py

2. Térképszínezés

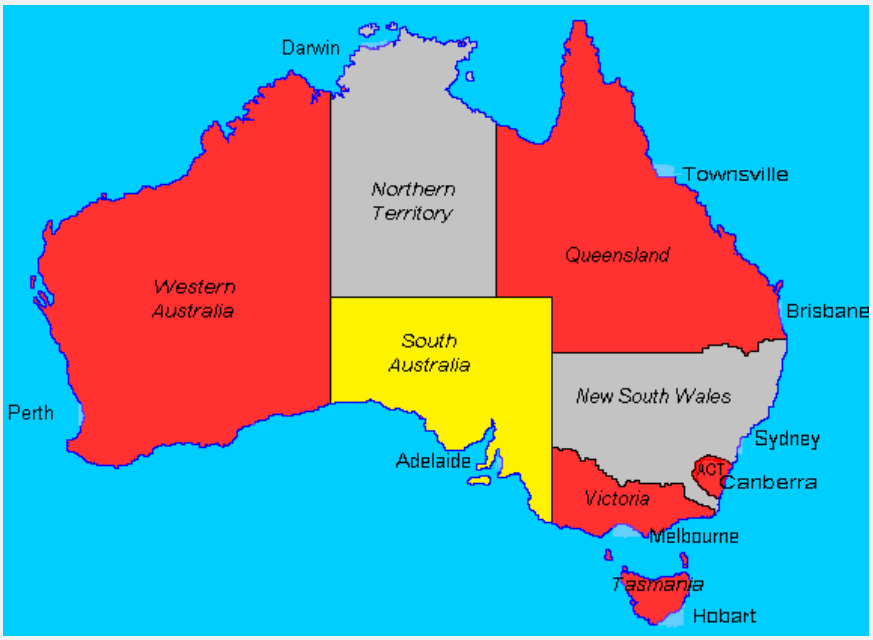
Ismerjük N ország elhelyezkedését a térképen, azaz tudjuk, hogy melyik országnak melyek a szomszédai.

Adjuk meg a térkép egy olyan kiszínezését, amelyben szomszédos országok különböző színűek lesznek![1]

A feladat nem lenne nehéz, ha akárhány színt használhatnánk. A színek számát azonban korlátozhatjuk, például az alábbi ábrán Ausztrália tagállamai kiszínezéséhez elég 3 szín. [2]

[1] Elképzelhető lenne a feladat olyan módosítása, amiben adott országoknak (területeknek) előre rögzítenénk a színét.

[2] A visszalépéses keresésről persze tudjuk, hogy nagyon lassú is lehet, ha sokszor kell visszalépni. Emiatt érdemes lehet a használható színek számát a minimálisnál nagyobbra választani. Egy másik gyorsítási lehetőség, ha a nagy szomszédszámú országokat előre vesszük.



A feladat pontosan tehát egy olyan színezést kér, amelyben legfeljebb K szín használható. A feladat más lenne, ha azt a kérést adnánk, hogy a lehető legkevesebb színnel színezzük ki a térképet.[3] Ez utóbbihoz a négyszínsejtés miatt persze felhasználhatnánk az előző megoldást, ha végig vizsgáljuk K=1, K=2, K=3 és K=4 értékekre. (K=1, ha senkinek sincs szomszédja; K=2, láncok vagy páros hosszú körök esetén.)

Ausztrália államai szomszédságát az alábbi mátrix-szal írhatjuk le, ahol a tagállamok kezdőbetűit akár az 1..8 sorszámokkal is helyettesíthetjük:

	WA	NT	SA	Q	NSW	ACT	V	T
WA	hamis	igaz	igaz	hamis	hamis	hamis	hamis	hamis
NT	igaz	hamis	igaz	igaz	hamis	hamis	hamis	hamis
SA	igaz	igaz	hamis	igaz	igaz	hamis	igaz	hamis
Q	hamis	igaz	igaz	hamis	igaz	hamis	hamis	hamis
NSW	hamis	hamis	igaz	igaz	hamis	igaz	igaz	hamis
ACT	hamis	hamis	hamis	hamis	igaz	hamis	hamis	hamis
V	hamis	hamis	igaz	hamis	igaz	hamis	hamis	hamis
T	hamis	hamis	hamis	hamis	hamis	hamis	hamis	hamis

A megoldásban használt fontos változók:

- `N` – a térkép területei száma
- `Sz` – adott két terület szomszédos-e (`Sz[i,j]` – ha az `i`. és a `j`. terület szomszédos, hamis egyébként)
- `Y` – az aktuálisan számolt megoldás: `Y[i]` az `i`. terület színe
- `K` – a használható színek száma

[3] A négyszínsejtés szerint bármely térkép kiszínezhető legfeljebb 4 színnel, azaz még a korlátnak sem kell nagynak lenni.

```
Térkép(N,K,Sz,Van,Y):
  i:=1; Y[]:=[0,…,0]
  Ciklus amíg i≥1 és i≤N {lehet még és nincs még kész}
    Jóesetkeresés(N,K,Sz,i,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.
```

A második szinten egyszerűsítenünk kellett, az `M[i]` helyébe `K` került, a `tilos(j)` függvényt pedig elhagyhattuk.

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

Akkor lehet az `i`. területnek a `j`. színt választani, ha minden korábbitól vagy különböző színű, vagy nem szomszéd.

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

Minta kódok

C++	cpp/2_Terkepszinezes/feladat.cpp
C#	cs/2_Terkepszinezes/feladat.cs
Java	java/2_Terkepszinezes/feladat.java
Pascal	pas/2_Terkepszinezes/feladat.pas
Python	py/2_Terkepszinezes/feladat.py

3. Iskolaválasztás

Egy pályaválasztási intézet elhatározza, hogy a 8. osztályos tanulók iskolaválasztásai alapján (minden jelentkezési lapon maximum két iskolát lehet megjelölni) megpróbál olyan 'beiskolázást' megvalósítani, amelyben minden tanulót az általa megjelölt valamelyik iskolába fel is vesznek. (Tudjuk az egyes iskolákba felvehetők számát.)

Igények	
1	2
1	2
1	–
2	3
2	4
1	4
1	2
3	4
3	–

Iskola kapacitások
2
3
2
2

Készíts programot, amely megad egy lehetséges jó beiskolázást!

A megoldásban használt fontos változók:

- `N` – a tanulók száma
- `M` – az iskolák száma
- `Igény` – adott tanuló igényei (`Igény[i,j]` – ha az `i`. tanuló `j`. iskolaigénye, vagy 0)
- `Y` – az aktuálisan számolt megoldás: `Y[i]` az `i`. tanuló iskolája
- `Kapacitás` – az iskolák kapacitása.

```
Iskola(N,M,Igény,Kapacitás,Van,Y):
  i:=1; Y[]:=[0,…,0]
  Ciklus amíg i≥1 és i≤N {lehet még és nincs még kész}
    Jóesetkeresés(i,Igény,Kapacitás,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]:=Igény[i,Y[i]]
               Ciklus vége
Eljárás vége.
```

Jól látható, hogy a fő eljárásban a tanuló igénysorszáma helyett a tanuló által kapott iskolát kell eredményül adnunk.

A második szinten egyszerűsítenünk kellett, az `M[i]` helyébe 1 vagy `2` került, a `tilos(j)` függvényt pedig elhagyhattuk.

```
Jóesetkeresés(i,Igény,Kapacitás,Van,j):
  j:=Y[i]+1
  Ha Igény[i,2]=0 akkor K:=1 különben K:=2
  Ciklus amíg j≤K és rossz(i,j,Igény,Kapacitás)
    j:=j+1
  Ciklus vége
  Van:=(j≤K)
Eljárás vége.
```

Akkor lehet az `i`. gyereknek a `j`. igényt választani, ha a korábbi igények szerint abban az iskolában van még hely.

```
rossz(i,j,Igény,Kapacitás):
  db:=1
  Ciklus k=1-től i-1-ig
    Ha Igény[k,Y[k]]=Igény[i,j] akkor db:=db+1
  Ciklus vége
  rossz:=(db>Kapacitás[Igény[i,j]]
Függvény vége.
```

A feladat megoldása tesztelhető az elkészült forráskód feltöltésével itt:

Weboldal	https://mester.inf.elte.hu/
Szint	
Téma	
Feladat	