Problémamegoldási stratégiák

Algoritmusok – kumulatív összegzés

Kumulatív összegzés

Adott egy N elemű számsorozat, adjuk meg a sorozat azon [a,b] intervallumát, ahol az elemek összege maximális!

Bemenet: $N \in \mathcal{N}$, $X \in \mathcal{N}^*$

Kimenet: $a,b \in \mathcal{H}^*$

Ef: $n>0$

Uf: $1 \le a \le b \le N$ és $\forall p,q(1 \le p \le q \le N)$: $\sum_{i=a}^{b} X_i \ge \sum_{i=p}^{q} X_i$

Algoritmusok – kumulatív összegzés

```
Alapmegoldás:
  MaxÉrt:=-∞
  Ciklus i=1-től N-ig
    Ciklus j=i-től N-ig
      s:=összeg(i,j)
      Ha s>Maxért akkor MaxÉrt:=s; a:=i; b:=j
    Ciklus vége
  Ciklus vége
Eljárás vége.
```

```
összeg(i,j):
  S:=0
  Ciklus k=i-től j-ig
    S:=S+X(k)
  Ciklus vége
  összeg:=S
Függvény vége.
```

Algoritmusok – kumulatív összegzés

```
Kumulatív összegzés:
   s(0):=0; MaxÉrt:=-∞
   Ciklus i=1-től N-ig
     s(i):=s(i-1)+X(i)
   Ciklus vége
   ...
```

$$\forall i \left(0 \le i \le N\right): \quad s(i) = \sum_{j=1}^{i} x_j$$

$$\forall i \left(1 \le i \le N\right) \quad s(i) = s(i-1) + x(i), \quad s(0) = 0$$

Algoritmusok – kumulatív összegzés

$$\sum_{i=a}^{b} x_i = s(b) - s(a-1)$$

```
 ...
  Ciklus i=1-től N-ig
    Ciklus j=i-től N-ig
      Ha s(j)-s(i-1)>Maxért
           akkor MaxÉrt:=s(j)-s(i-1); a:=i; b:=j
    Ciklus vége
  Ciklus vége
Eljárás vége.
```

Algoritmusok – kumulatív összegzés

A megoldások összehasonlítása

- Alapmegoldás: 3 egymásba ágyazott ciklus – $O(N^3)$
- Kumulatív összegzés: 2 egymásba ágyazott ciklus – $O(N^2)$

Meggondolandók

- ez az elv milyen struktúrákra alkalmazható?
- ez az elv milyen feladattípusokra alkalmazható?

Feladatmegoldási stratégiák

Oszd meg és uralkodj!

Több részfeladatra bontás, amelyek hasonlóan oldhatók meg, lépései:

- a triviális eset (amikor nincs rekurzív hívás)
- felosztás (megadjuk a részfeladatokat, amikre a feladat lebontható)
- uralkodás (rekurzívan megoldjuk az egyes részfeladatokat)
- összevonás (az egyes részfeladatok megoldásából előállítjuk az eredeti feladat megoldását)

Feladatmegoldási stratégiák

Ezek alapján a következőképpen fogunk gondolkodni:

- Mi a leállás (triviális eset) feltétele? Hogyan oldható meg ilyenkor a feladat?
- Mi az általános feladat alakja? Mik a paraméterei? Ebből kapjuk meg a rekurzív eljárásunk specifikációját.
- Milyen paraméterértékekre kapjuk a konkrét feladatot? Ezekre fogjuk meghívni kezdetben az eljárást!

Feladatmegoldási stratégiák

Ezek alapján a következőképpen fogunk gondolkodni:

- Hogyan vezethető vissza a feladat hasonló, de egyszerűbb részfeladatokra? Hány részfeladatra vezethető vissza?
- Melyek ilyenkor az általános feladat részfeladatainak a paraméterei? Ezekkel kell majd meghívni a rekurzív eljárást!
- Hogyan építhető fel a részfeladatok megoldásaiból az általános feladat megoldása?

Feladatmegoldási stratégiák
Oszd meg és uralkodj

Gyorsrendezés (quicksort):

- felbontás: $\boxed{X_1, \ldots, X_{k-1}}\ X_k\ \boxed{X_{k+1}, \ldots, X_n}$ szétválogatás

  ahol $\forall i,j\ (1 \le i < k;\ k < j \le n)$: $X_i \le X_j$
- uralkodás: mindkét részt ugyanazzal a módszerrel felbontjuk két részre, rekurzívan
- összevonás: automatikusan történik a helyben szétválogatás miatt
- triviális eset: $n \le 1$

Feladatmegoldási stratégiák
Oszd meg és uralkodj

Gyorsrendezés (quicksort):

```
Quick(E,U):
  Szétválogatás(E,U,K)
  Ha E<K-1 akkor Quick(E,K-1)
  Ha k+1<U akkor Quick(K+1,U)
Eljárás vége.
```

Feladatmegoldási stratégiák
Oszd meg és uralkodj

Gyorsrendezés (quicksort):

```
Szétválogatás(E,U,K):
  Y:=X(1)
  Ciklus amíg E<U
    Ciklus amíg E<U és X(U)>Y
      U:=U-1
    Ciklus vége
    Ha E<U akkor X(E):=X(U); E:=E+1
                 Ciklus amíg E<U és X(E)<Y
                   E:=E+1
                 Ciklus vége
                 Ha E<U akkor X(U):=X(E); U:=U-1
  Ciklus vége
  X(E):=Y
Eljárás vége.
```

Feladatmegoldási stratégiák
Oszd meg és uralkodj

Összefésüléses rendezés (mergesort):

- felbontás: a sorozat két részsorozatra bontása (középen)

$X_1, \dots, X_k$ | $X_{k+1}, \dots, X_n$

- uralkodás: a két részsorozat rendezése (rekurzívan)
- összevonás: a két rendezett részsorozat összefésülése
- triviális eset: $n \leq 1$

Feladatmegoldási stratégiák
Oszd meg és uralkodj

Összefésüléses rendezés (mergesort):

```
Rendez(E,U):
   Ha E<U akkor K:=(E+U)/2
                Rendez(E,K); Rendez(K+1,U)
                Összefésül(E,K,U)
Eljárás vége.
```

Feladatmegoldási stratégiák
Oszd meg és uralkodj

Összefésüléses rendezés (mergesort):

```
Összefésül(E,K,U):
  i:=E; j:=K+1; a:=0; Y:=X
  Ciklus amíg i≤K és j≤U
    a:=a+1
    Ha Y(i)≤Y(j) akkor X(a):=Y(i); i:=i+1
    különben X(a):=Y(j); j:=j+1
  Ciklus vége
  Ciklus b=i-től K-ig
    a:=a+1; X(a):=Y(b)
  Ciklus vége
 Ciklus b=j-től U-ig
    a:=a+1; X(a):=Y(b)
  Ciklus vége
Eljárás vége.
```

Feladatmegoldási stratégiák
Oszd meg és uralkodj

i-edik legkisebb kiválasztása:

- felbontás: $\boxed{X_1, \ldots, X_{k-1}}\ X_k\ \boxed{X_{k+1}, \ldots, X_n}$ szétválogatás (ahol $\forall i,j\ (1 \le i \le k;\ k \le j \le n)$: $X_i \le X_j$)
- uralkodás: i<K esetén az első, i>K esetén a második részben keresünk tovább, rekurzívan
- összevonás: automatikusan történik a helyben szétválogatás miatt
- triviális eset: i=k

Feladatmegoldási stratégiák
Oszd meg és uralkodj

i-edik legkisebb kiválasztása:

```
Kiválasztás(E,U,i,Y):
   Szétválogatás(E,U,K)
   Ha i=K akkor Y:=X(K)
   különben ha i<K akkor Kiválasztás(E,K-1,i,Y)
   különben Kiválasztás(K+1,U,i-K,Y)
Eljárás vége.
```

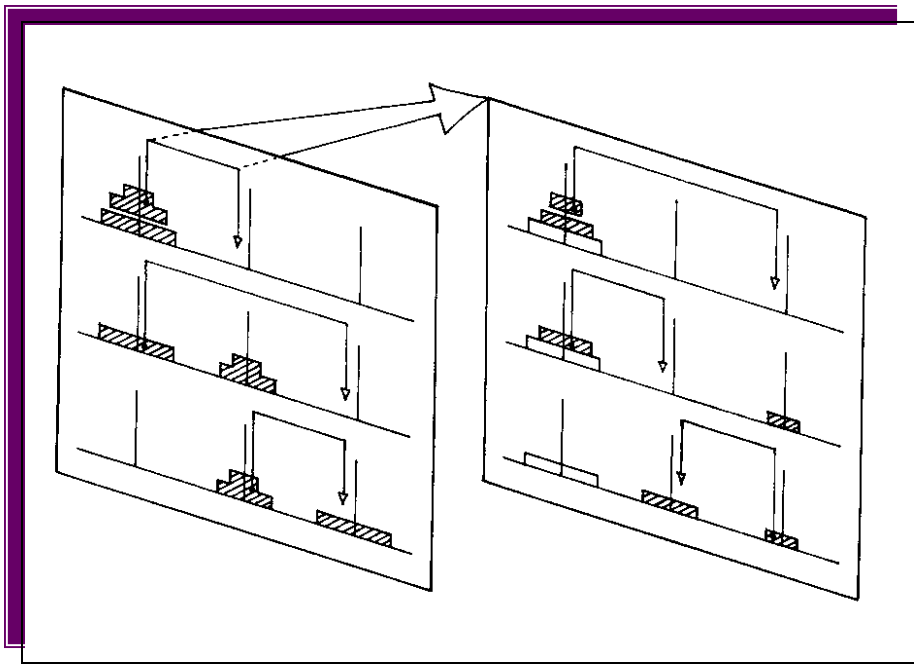
Feladatmegoldási stratégiák
Oszd meg és uralkodj

Hanoi tornyai:

Adott 3 rudacska. Az elsőn egyre csökkenő sugarú korongok vannak. Az a feladat, hogy tegyük át a harmadik rudacskára a korongokat egyenként úgy, hogy az átpakolás közben és természetesen a végén is minden egyes korongon csak nála kisebb lehet. Az átpakoláshoz lehet segítségül felhasználni a középső rudacskát.

Feladatmegoldási stratégiák
Oszd meg és uralkodj

Hanoi tornyai:



Feladatmegoldási stratégiák
Oszd meg és uralkodj

Hanoi tornyai:

```
Hanoi(n,A,B,C):
  Ha n>0 akkor Hanoi(n-1,A,C,B)
               Ki: N,A,B
               Hanoi(n-1,C,B,A)
  Elágazás vége
Eljárás vége.
```

```
Hanoi(n,A,B,C):
  Ha n>1 akkor Hanoi(n-1,A,C,B)
               Ki: n,A,B
               Hanoi(n-1,C,B,A)
       különben Ki: n,A,B
Eljárás vége.
```

Feladatmegoldási stratégiák
Oszd meg és uralkodj

Logaritmikus keresés:

- N elemű sorozatban keresés visszavezetése N/2 elemű sorozatban keresésre.

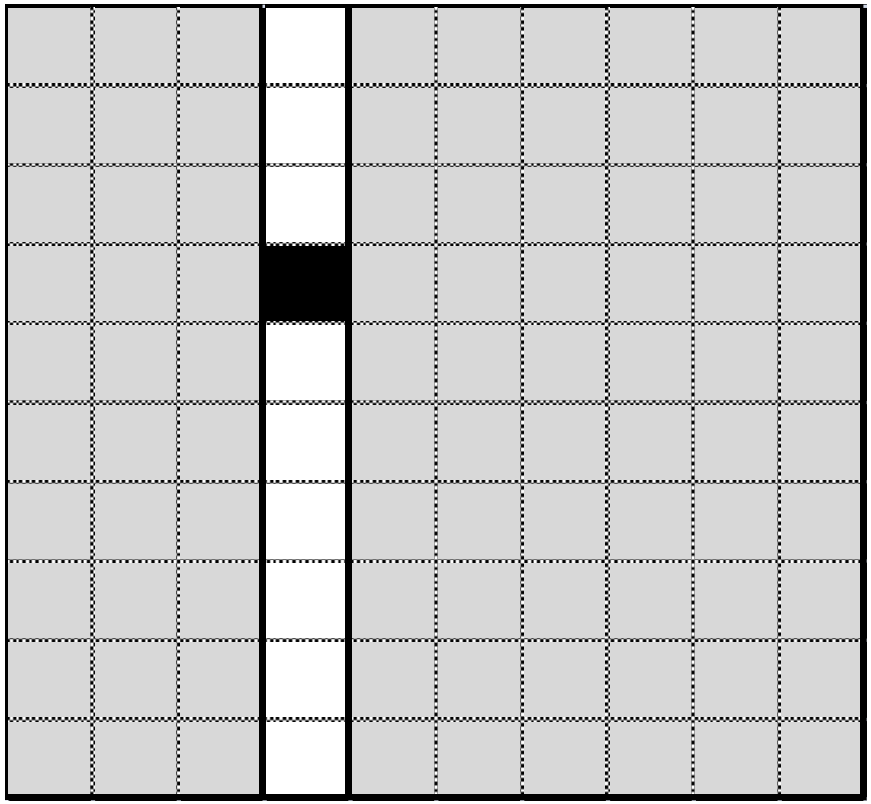
```
Logker(X,T,E,U,Van,K):
   Ha E>U akkor Van:=hamis
   különben
      K:=(E+U)/2
      Ha X=T(K) akkor Van:=igaz
      különben ha X<T(K)
         akkor Logker(X,T,E,U-1,Van,K)
      különben Logker(X,T,E+1,U,Van,K)
Eljárás vége
```

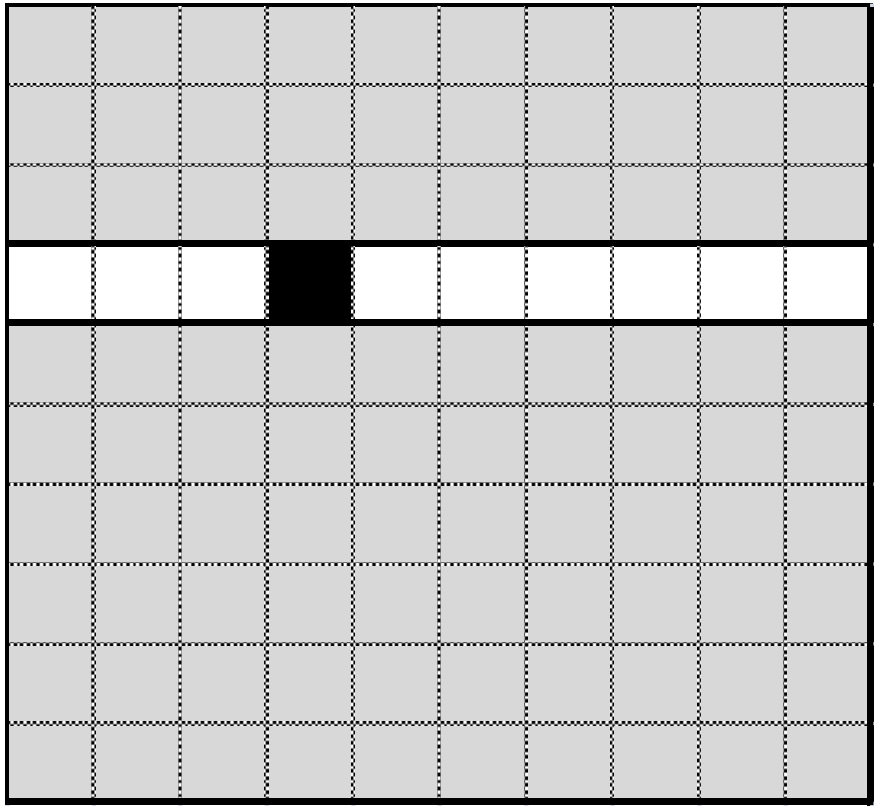
Feladatmegoldási stratégiák
Oszd meg és uralkodj

Legnagyobb üres téglalap:

- Szétbontás: Egy téglalapban levő N ponthoz a legnagyobb résztéglalap keresése, amely egyetlen pontot sem tartalmaz – egy meg-vizsgálandó téglalapot minden belső pont négy megvizsgálandó részre vág (ezek átfedők).
- Uralkodás: Ezekre rekurzívan alkalmazható az eljárás.
- Összevonás: a négy eredmény maximumát kell megadni!

Feladatmegoldási stratégiák
Oszd meg és uralkodj





Legnagyobb üres téglalap

Feladatmegoldási stratégiák
Oszd meg és uralkodj

Legnagyobb üres téglalap:

```
Legnagyobb(P,T,L,i,n):
  Ha i>n akkor L:=T
  különben
    t1:=T; t1.jax:=P(i).x-1; {balra}
    Legnagyobb(P,t1,L1,i+1,n)
    t2:=T; t2.bfy:=P(i).y+1 {alul}
    Legnagyobb(P,t2,L2,i+1,n)
    t3:=T; t3.bfx:=P(i).x+1; {jobbra}
    Legnagyobb(P,t3,L3,i+1,n)
    t4:=T; t4.jay:=P(i).y+1 {felül}
    Legnagyobb(P,t4,L4,i+1,n)
    …
```

Feladatmegoldási stratégiák
Oszd meg és uralkodj

Legnagyobb üres téglalap:

```
    …
    L:=L1
    Ha terület(L)<terület(L2) akkor L:=L2
    Ha terület(L)<terület(L3) akkor L:=L3
    Ha terület(L)<terület(L4) akkor L:=L4
  Elágazás vége
Eljárás vége.
```

Az eljárás javítható, hatékonyabbá tehető.

Feladatmegoldási stratégiák
Visszalépéses keresés

Visszalépéses keresés (backtrack)

A visszalépéses keresés lényege a feladat megoldásának előállítása rendszeres próbálgatással.

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. Másképp fogalmazva: egy adott tulajdonsággal rendelkező szám N-est kell megadni úgy, hogy ne kelljen az összes lehetőséget végignézni!

Feladatmegoldási stratégiák
Visszalépéses keresés

N vezér elhelyezése egy NxN-es sakktáblán

Helyezzünk el egy NxN-es sakktáblán N vezért úgy, hogy ne üssék egymást!

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

		V		
				V
	V			
			V	
V				

	V		
			V
V			
		V	

Feladatmegoldási stratégiák
Visszalépéses keresés

Munkásfelvétel: N állás – N jelentkező

Egy vállalkozás N különböző állásra keres munkásokat. Pontosan N jelentkező érkezett, ahol minden jelentkező megmondta, hogy mely munkákhoz ért. A vállalkozás vezetője azt szeretné, ha az összes jelentkezőt fel tudná venni és minden munkát elvégeztetni.

	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**

Feladatmegoldási stratégiák
Visszalépéses keresés

A visszalépéses keresés megoldási elve

E feladatok közös jellemzője, hogy eredményük egy sorozat. E sorozat minden egyes tagját valamilyen sorozatból kell kikeresni (vezért egy oszlop valamely helyére, egy munkásnak a vállalt munkák közül valamelyiket), de az egyes keresések összefüggenek egymással (vezért nem lehet oda tenni, ahol egy korábban letett vezér ütné; egy munkát nem lehet két munkásnak adni).

Feladatmegoldási stratégiák
Visszalépéses keresés

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.

Feladatmegoldási stratégiák
Visszalépéses keresés

Először megpróbálunk az első sorozatból kiválasztani egy elemet, ezután a következőből, ...

Ha nincs jó választás, akkor visszalépünk az előző sorozathoz, s megpróbálunk abból egy másik elemet választani.

Visszalépésnél 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).

Feladatmegoldási stratégiák
Visszalépéses keresés

Visszalépéses keresés algoritmus:

```
Keresés(N,Van,Y):
  i:=1
  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 megoldás legfelső szintjén keressünk az i. sorozatból megfelelő elemet! Ha ez sikerült, akkor lépjünk tovább az i+1. sorozatra, különben lépjünk vissza az i-1.-re, s keressünk abban újabb elemet!

Feladatmegoldási stratégiák
Visszalépéses keresés

Visszalépéses keresés algoritmus:

```
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.

Feladatmegoldási stratégiák
Visszalépéses keresés

Visszalépéses keresés algoritmus:

```
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)
Eljárás vége.
```

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

Feladatmegoldási stratégiák
Visszalépéses keresés

Visszalépéses keresés algoritmus:

```
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)
Eljárás vége.
```

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

Feladatmegoldási stratégiák
Visszalépéses keresés

Visszalépéses keresés algoritmus:

```
rossz(i,j):                                  {3. változat}
  rossz:=i>1 és nem szabad(i,j,i-1,Y(i-1))
Eljárás vége.
```

Megjegyzés: Rossz egy választás, ha az előző választás miatt nem szabad – feltétel vizsgálat.

Feladatmegoldási stratégiák
Visszalépéses keresés

Visszalépéses kiválogatás rekurzív algoritmus:

```
Visszalépéses kiválogatás(N,Db,Y):
  Db:=0; X:=(0,…,0); Backtrack(1,N,X,Db,Y)
Eljárás vége.

Backtrack(i,N,X,Db,Y):
  Ha i=N+1 akkor Db:=Db+1; Y(Db):=X
  különben Ciklus j=1-től N-ig
             Ha ft(i,j) és nem Rossz(i,j)
                akkor X(i):=j
                       Backtrack(i+1,N,X,Db,Y)
           Ciklus vége
  Elágazás vége
Eljárás vége.
```

Feladatmegoldási stratégiák
Visszalépéses keresés

Visszalépéses maximumkeresés rekurzív algoritmus:

```
Visszalépéses maximumkeresés(N,Van,Y):
   X:=(0,…,0); Y:=X; Backtrack(1,N,X,Y)
   Van:=Y≠(0,…,0)
Eljárás vége.
```

Feladatmegoldási stratégiák
Visszalépéses keresés

Visszalépéses maximumkeresés rekurzív algoritmus:

```
Backtrack(i,N,X,Y):
  Ha i=N+1 akkor ha nagyobb?(X,Y) akkor Y:=X
  különben Ciklus j=1-től N-ig
             Ha ft(i,j) és nem Rossz(i,j)
                akkor X(i):=j
                       Backtrack(i+1,N,X,Y)
           Ciklus vége
  Elágazás vége
Eljárás vége.
```

Feladatmegoldási stratégiák
Visszalépéses keresés

Visszalépéses keresés: 8 vezér a sakktáblán

```
Keresés(Van,Y):
  i:=1
  Ciklus amíg i≥1 és i≤8  {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>8)
Eljárás vége.
```

Feladatmegoldási stratégiák
Visszalépéses keresés

Visszalépéses keresés algoritmus:

```
Jóesetkeresés(i,Van,j):
  j:=Y(i)+1
  Ciklus amíg j≤8 és rossz(i,j)
    j:=j+1
  Ciklus vége
  Van:=(j≤8)
Eljárás vége.
```

Feladatmegoldási stratégiák
Visszalépéses keresés

Visszalépéses keresés algoritmus:

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

Megjegyzés: Rossz egy választás, ha valamelyik korábbi vezér üti.

Feladatmegoldási stratégiák
Visszalépéses keresés

Visszalépéses keresés: N munka – N munkás

```
Keresés(N,Van,Y):
  i:=1
  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.
```

Feladatmegoldási stratégiák
Visszalépéses keresés

Visszalépéses keresés algoritmus:

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

Nem adható neki a munka, ha nem ért hozzá.

Feladatmegoldási stratégiák
Visszalépéses keresés

Visszalépéses keresés algoritmus:

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

Megjegyzés: Rossz egy választás, ha azt a munkát valamelyik korábbi munkás megkapta.

Feladatmegoldási stratégiák
Visszalépéses keresés

Visszalépéses keresés: N üzlet – M pékség

```
Keresés(N,Van,Y):
  i:=1
  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.
```

Feladatmegoldási stratégiák
Visszalépéses keresés

Visszalépéses keresés algoritmus:

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

Nem szállíthat az i. üzletbe a j. pékség, ha nincsenek kapcsolatban.

Feladatmegoldási stratégiák
Visszalépéses keresés

Visszalépéses keresés algoritmus:

```
rossz(i,j):
  s:=van(j)
  Ciklus k=1-től i-1-ig
    Ha Y(k)=j akkor s:=s-igény(k)
  Ciklus vége
  rossz:=s<igény(i)
Eljárás vége.
```

Megjegyzés: Rossz egy választás, ha abban a pékségben nem maradt az i. üzlet által igényelt menniység.

Feladatmegoldási stratégiák
Visszalépéses maximumkeresés

Példa: (1. változat)

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.

Minden munkát el kell végeztetni valakivel, mindenkinek munkát kell adni, de a legolcsóbban!

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**

Feladatmegoldási stratégiák
Visszalépéses maximumkeresés

Ha egy megoldás elkészül, akkor a költségét így számíthatjuk ki:

```
költség(X):
  S:=0
  Ciklus i=1-től N-ig
    S:=S+F(i,X(i))
  Ciklus vége
Függvény vége.
```

Kezdetben olyan – fiktív – megoldásból kell kiindulni, aminél minden valódi megoldás jobb.

Feladatmegoldási stratégiák
Visszalépéses maximumkeresés

```
Legjobb állás(N,i):
  Ha i>N akkor
       Ha költség(X)<költség(Y) akkor Y:=X
  különben
       Ciklus j=1-től N-ig
         Ha nem volt(i,j) és F(i,j)>0
            akkor X(i):=j; Legjobb állás(N,i+1)
       Ciklus vége
  Elágazás vége
Eljárás vége.
```

Ebben a megoldásban feleslegesen sokszor hívjuk a Költség függvényt.

Feladatmegoldási stratégiák
Visszalépéses maximumkeresés

```
Legjobb állás(N,i):
  Ha i>N akkor Ykölt:=költség(Y)
                Ha maxkölt<Ykölt
                   akkor Y:=X; maxkölt:=Ykölt
  különben
       Ciklus j=1-től N-ig
         Ha nem volt(i,j) és F(i,j)>0
            akkor X(i):=j; Legjobb állás(N,i+1)
       Ciklus vége
  Elágazás vége
Eljárás vége.
```

Itt feleslegesen nem hívjuk a Költség függvényt, jó maxkölt kezdőérték kell!

Feladatmegoldási stratégiák
Visszalépéses maximumkeresés

- 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.
- 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.

Feladatmegoldási stratégiák
Visszalépéses maximumkeresés

```
Legjobb állás(N,i,költ):
  Ha i>N akkor
      Ha költ<maxköltakkor Y:=X; maxköltköltés
  különben Ciklus j=1-től N-ig
              Ha nem volt(i,j) és F(i,j)>0
                 és költ+F(i,j)<maxköltés
              akkor X(i):=j
                  Legjobb állás(N,i+1,költ+F(i,j))
           Ciklus vége
  Elágazás vége
Eljárás vége.
```

Feladatmegoldási stratégiák
Visszalépéses maximumkeresés

Példa: (2. változat)

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

Mindenkinek munkát kell adni (csak egyet mindenkinek), de a legolcsóbban!

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

Feladatmegoldási stratégiák
Visszalépéses maximumkeresés

```
Legjobb állás(N,M,i,költ):
  Ha i>M akkor
      Ha költ<maxköltakkor Y:=X; maxköltköltés
  különben Ciklus j=1-től N-ig
              Ha nem volt(i,j) és F(i,j)>0
                 és költ+F(i,j)<maxköltés
              akkor X(i):=j
                Legjobb állás(N,M,i+1,költ+F(i,j))
            Ciklus vége
  Elágazás vége
Eljárás vége.
```

Feladatmegoldási stratégiák
Visszalépéses maximumkeresés

Példa: (3. változat)

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

Mindenkinek munkát el kell végezni, egy-egy embernek, de a legolcsóbban!

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

Feladatmegoldási stratégiák
Visszalépéses maximumkeresés

```
Legjobb állás(N,M,i,költ):
  Ha i>N akkor
      Ha költ<maxköltakkor Y:=X; maxköltköltés
  különben Ciklus j=1-től M-ig
              Ha nem volt(j,i) és F(j,i)>0
                 és költ+F(j,i)<maxköltés
              akkor X(i):=j
                Legjobb állás(N,M,i+1,költ+F(j,i))
           Ciklus vége
  Elágazás vége
Eljárás vége.
```

Feladatmegoldási stratégiák
Visszalépéses keresés

További visszalépéses keresés feladatok:

- Labirintusban útkeresés
- Permutációk, kombinációk előállítása
- Térképszínezés
- Pénzfelbontás adott címletekre

Feladatmegoldási stratégiák
Visszalépéses keresés

Visszalépéses keresés feladatok:

- Úthossz-korlát: Fává egyenesítünk, végtelen fát állítunk elő. Nem engedjük, hogy az aktuális út hossza meghaladja az úthossz-korlátot.
  - Ha túl rövidre választjuk az úthossz-korlátot (túl ala-csonyan vágjuk el a gráfot) akkor nem találunk megoldást.
  - Ha a start csúcsban áll elő a visszalépési feltétel, akkor:
    - nincs megoldás
    - túl rövidre választottuk az úthossz-korlátot

Feladatmegoldási stratégiák
Visszalépéses keresés

Visszalépéses keresés feladatok:

- Kör kiküszöbölése
  - lesz egy újabb visszalépési feltétel: az aktuális csúcs szerepelt-e már az aktuális úton
  - ha igen: rögtön visszalépés (így nem zárjuk be a kört).

Feladatmegoldási stratégiák
Visszalépéses keresés

A visszalépéses stratégia

- véges fákban mindig terminál (véges fákban teljes);
- végtelen gráfban úthossz-korláttal terminál (kör kizárása: az aktuális út csúcsait nem engedjük ismételni);
- egy zsákutcát többször is bejár, ha több út vezet hozzá.

Feladatmegoldási stratégiák
Elágazás és korlátozás

A backtrack alkalmas-e optimális megoldás keresésére? Van költség, és a legkisebb költségű megoldást szeretnénk elő-állítani.

- Van egy induló költségkorlát (felső becslés).
- Ennél a költségkorlátnál nem költségesebb megoldást keresünk.
- Drágább részmegoldás esetén is visszalépünk.

Feladatmegoldási stratégiák
Elágazás és korlátozás

Az aktuális pontot tetszőlegesen választhatjuk az aktív pontok közül.

A lényeg, hogy a választott aktuális pontból elérhető összes pontot generáljuk, és ha lehetséges megoldás, akkor betesszük az aktív pontok halmazába.

Tehát az algoritmus egy, az aktív pontokat tartalmazó adagolót használ az aktív pontok tárolására.

A visszalépéses stratégia esetén elég volt egy pontot, az aktuális pontot tárolni, mert a következő aktív pont mindig ennek fia, testvére, vagy apja.

Feladatmegoldási stratégiák
Elágazás és korlátozás

Adott a C(X) valós értékű célfüggvény, és olyan X megoldást keresünk, amelyre a célfüggvény C(X) értéke minimális.

A megoldáskezdeményekre meg tudunk adni olyan AK(X) alsó korlát függvényt, amelyekre teljesül az alábbi egyenlőtlenség:

- Az Y megoldás bármely X részmegoldására: $AK(X) \leq C(Y)$

Ekkor az adagoló lehet az AK szerinti minimumos prioritási sor, tehát az aktív pontok közül mindig a legkisebb alsó korlátú pontot választjuk aktuálisnak.

Feladatmegoldási stratégiák
Elágazás és korlátozás

```
Korlátozás(F):
  Minért:=+∞; Betesz(A,F)
  Ciklus amíg nem üres?(A)
    Kivesz(A,F)
    Ciklus p=F-ből kapható megoldáslépések
      Ha Megoldás(p) akkor
         Ha C(p)<Minért akkor Minért:=C(p)
                              Min:=p
      különben ha AK(p)<Minért akkor Betesz(A,p)
    Ciklus vége
  Ciklus vége
Eljárás vége.
```

Feladatmegoldási stratégiák
Elágazás és korlátozás

Ha a megoldáskezdeményekre meg tudunk adni felső korlátot is, akkor az adagoló lehet a felső korlát szerinti minimumos prioritási sor.

- Felső korlát olyan FK(X) függvény, amelyre teljesül, hogy minden Y megoldás minden X részmegoldására:

$$C(Y) \leq FK(X).$$

- Azaz egy részmegoldásnál járva tudjuk, hogy az ebből kiinduló megoldásoknak mi a felső korlátja.
- Mindig a legkisebb felső korlátú ágat válasszuk!

Problémamegoldási stratégiák