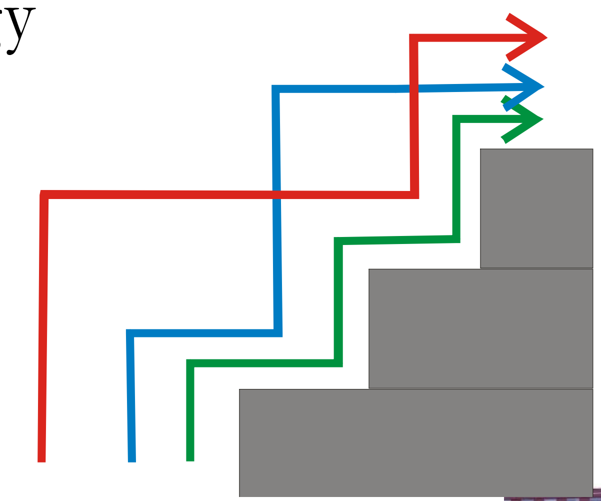
Programozási alapismeretek

13. előadás

Érdekességek - kombinatorika

- Az iskola bejáratánál N lépcsőfok van. Egyszerre maximum K fokot tudunk lépni, ugrani fölfele. Minden nap egyszer megyünk be az iskolába.

- Készíts programot, amely megadja, hogy hány napig tudunk más és más módon feljutni a lépcsőkön!



Érdekességek - kombinatorika

- Bemenet: N,K∈Egész
- Kimenet: Db∈Egész
- Előfeltétel: —
- Utófeltétel: ???

A probléma az, hogy nem látszik közvetlen összefüggés a bemenet és a kimenet között.

Érdekességek - kombinatorika

Próbáljuk megfogalmazni minden egyes lépcsőfokra, hogy hányféleképpen érhetünk el oda!

- Bemenet: $N,K \in \mathbf{N}$
- Kimenet: $Db_{0..N} \in \mathbf{N}^{N+1}$
- Előfeltétel: —
- Utófeltétel: ???

Továbbra sem látszik közvetlen összefüggés a bemenet és a kimenet között.

Érdekességek - kombinatorika

Próbáljunk meg összefüggést felírni a kimenetre önmagában!

Észrevétel: Az N-edik lépcsőfokra vagy az N-1-edikről lépünk, vagy az N-2-edikről, … vagy pedig az N-K-adikról!

- Utófeltétel: Db[0]=1 és $\forall j(1 \le j \le N): Db[j] = \sum_{\substack{i=1 \\ i \le j}}^{K} Db[j-i]$

Tehát eljutottunk a sorozatszámítás (összegzés) programozási tételhez.

Érdekességek - kombinatorika

Db[0]:=1

j=1..N

Db[j]:=0

i=1..K

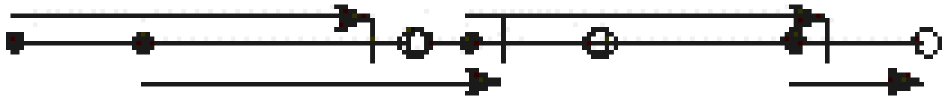
j≥i

Db[j]:=Db[j]+Db[j-i]	—

```
Db[0]=1;
for(int j=1, j<=N; j++) {
   Db[j]=0;
   for(int i=1; i<=K; i++) {
      if  (j>=i) Db[j]=Db[j]+Db[j-i]  }   }
```

Érdekességek – Mohó stratégia

- A Budapest-Párizs útvonalon N benzinkút van, az i-edik B_i távolságra Budapesttől (az első Budapesten, az utolsó Párizsban). Egy tankolás az autónak K kilométerre elég.
- Készíts programot, amely megadja a lehető legkevesebb benzinkutat, ahol tankolni kell, úgy, hogy eljuthassunk Budapestről Párizsba!



Érdekességek – Mohó stratégia

- Bemenet: $N,K \in \mathbf{N}$
  $B_{1..N} \in \mathbf{N}^N$
- Kimenet: $Db \in \mathbf{N}$
  $T_{1..N-1} \in \mathbf{N}^{N-1}$
- Előfeltétel: $\forall i(1 \le i < N)$: $B[i+1]-B[i] \le K$
- Utófeltétel: $Db=???$ és $T[1]=1$ és
  $\forall i(1 \le i < Db)$: $B[T[i+1]]-B[T[i]] \le K$ és
  $B[N]-B[T[Db]] \le K$ és $T \subseteq (1,\dots,N-1)$

Érdekességek – Mohó stratégia

- A megfogalmazásból látható, hogy a tankolási helyek halmaza az összes benzinkút halmazának egy részhalmaza lesz.
- Állítsuk elő az összes részhalmazt, majd válogassuk ki közülük a jókat (amivel el lehet jutni Párizsba), s végül adjuk meg ezek közül a legkisebb elemszámút!
- Probléma: 2^N részhalmaz van!

Érdekességek – Mohó stratégia

A megoldás (tegyük fel, hogy van megoldás):

- Budapesten mindenképpen kell tankolni!
- Menjünk, ameddig csak lehet, s a lehető legutolsó benzinkútnál tankoljunk!
- …
- Belátható, hogy ezzel egy optimális megoldást kapunk.

→ **Kiválogatás!**

Érdekességek – Mohó stratégia

Db:=1

T[1]:=1

i=2..N-1

B[i+1]-B[T[Db]]>K	
Db:=Db+1	—
T[Db]:=i	

```
Db=1; T[1]=1;
for(int i=2; i<N; i++){
   if(B[i+1]-B[T[Db]]>K)
      { Db++; T[Db]=i }
}
```

Feladatmegoldási stratégiák
Visszalépéses keresés

Feladat

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

A vezérek 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!

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

Először megpróbáljuk az első vezért elhelyezni az első oszlopban, ezután a következőt …

Ha nem tudjuk elhelyezni, akkor visszalépünk az előző oszlophoz, s megpróbálunk abból egy másik helyet választani.

Visszalépésnél törölni kell a választást abból a sorozatból, amelyikből visszalépünk. Az eljárás akkor ér véget, ha minden vezért sikerült elhelyezni, vagy pedig a visszalépések sokasága után már az első vezért sem lehet elhelyezni (ekkor a feladatnak nincs megoldása).

Visszalépéses keresés algoritmus:

```
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 megoldás legfelső szintjén keressünk az i. oszlopban megfelelő elemet! Ha ez sikerült, akkor lépjünk tovább az i+1. oszlopra, különben lépjünk vissza az i-1.-re, s keressünk abban újabb helyet!

Visszalépéses keresés algoritmus:

```
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.
```

Megjegyzés: az i-edik lépésben a j-edik hely nem választható, ha az előző vezérek miatt rossz.

Visszalépéses keresés algoritmus:

```
rossz(i,j):                                {1. változat}
  k:=1
  Ciklus amíg k<i és nem üti(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.

```
üti(i,j,k,Y(k))=Y(k)=j vagy i-k=abs(j-Y(k))
```

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 el tudna végeztetni.

	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

Feladat – 1. változat

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

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

`M(i)` – az i. munkás ennyi munkához ért

`E(i,j)` – az i. munkás által elvégezhető j. munka

Feladatmegoldási stratégiák
Visszalépéses keresés

N munka – N jelentkező:

```
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.
```

Feladatmegoldási stratégiák
Visszalépéses keresés

N munka – N jelentkező:

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

Feladatmegoldási stratégiák
Visszalépéses keresés

N munka – N jelentkező:

```
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)
Eljárás vége.
```

E(i,j) – az i. munkás által elvégezhető j. munka

Érdekességek - keverés

- Van N elemünk (1,2,…,N), keverjük össze őket véletlenszerűen!
- Mit jelent a keverés? Az N elem összes lehetséges sorrendje egyenlő eséllyel álljon elő a keverésnél!
- (1,2,…,N) → $(X_1, X_2, \ldots, X_N)$

(A hamiskártyások egyik trükkje, hogy nem így keverik a kártyákat!)

Érdekességek - keverés

A megoldás elve:

- Válasszunk az N elem közül egyet véletlenszerűen, és cseréljük meg az elsővel!
  Így az első helyre egyenlő (1/N) eséllyel kerül bármely elem.
- A maradék N-1-ből újra válasszunk véletlenszerűen egyet, s cseréljük meg a másodikkal!

Érdekességek - keverés

Be kellene látnunk, hogy így jó megoldást kapunk! (nem bizonyítás, csak gondolatok)

- Nézzük meg, hogy mi annak az esélye, hogy az I kerül a második helyre!
- Ez úgy történhet, hogy az első helyre nem az I került (esélye (N-1)/N), a másodikra pedig igen (esélye 1/(N-1)).
- Tehát (N-1)/N*1/(N-1)=1/N!

Érdekességek - keverés

Ez olyan, mint a rendezés, csak nagyság szerinti hely helyett véletlenszerű helyre cserélünk.

i=1..N-1

j:=Véletlen(i..N)

Csere(X[i],X[j])

```
for(int i=1; i<N; i++){
    int j=i+rand() % (N-i+1);
    int y=X[i]; X[i]=X[j]; X[j]=y;
}
```

Érdekességek – közelítő számítások

- Feladat: Számítsuk ki gyök(2) értékét!
- Probléma: irracionális számot biztosan nem tudunk ábrázolni a számítógépen!
- Új feladat: Számítsuk ki azt a P,Q egész számpárt, amire P/Q elég közel van gyök(2)-höz!
- Probléma: Mi az, hogy „elég közel”?
- Ötlet: $|P^2/Q^2-2|<E$, ahol E egy kicsi pozitív valós szám.

Érdekességek – közelítő számítások

- Bemenet: $E \in \mathbf{R}$
- Kimenet: $P, Q \in \mathbf{N}$
- Előfeltétel: $E>0$
- Utófeltétel: $|P^2/Q^2-2|<E$
- Probléma: nem látszik egyszerű összefüggés P, Q és E között.
- Ötlet: Állítsunk elő (P_i,Q_i) számpárokat úgy, hogy $|P_{i+1}^2/Q_{i+1}^2-2| < |P_i^2/Q_i^2-2|$ legyen!
- Ha felülről közelítünk, az abszolút érték jel elhagyható!

Érdekességek – közelítő számítások

- Állítás: a P^2-M*Q^2=4 egyenletnek végtelen sok megoldása van, ha M nem négyzetszám.
  (Most nem bizonyítjuk.)
- Állítás: az alábbi sorozat értéke gyök(2)-höz tart, ha n tart végtelenhez (most ezt sem bizonyítjuk):

$$x_{n+1} := \frac{1}{2} * \left(x_n + \frac{2}{x_n} \right)$$

- Legyen $x_n=P_n/Q_n$!

Érdekességek – közelítő számítások

- Legyen (P_n,Q_n) a fenti egyenlet megoldása. Ekkor:

$$x_{n+1} := \frac{1}{2} * \left(x_n + \frac{2}{x_n} \right) = \frac{1}{2} * \left(\frac{P_n}{Q_n} + \frac{2 * Q_n}{P_n} \right) =$$

$$= \frac{1}{2} * \left(\frac{P_n^2 + 2 * Q_n^2}{P_n * Q_n} \right) = \frac{P_n^2 - 2}{P_n * Q_n} = \frac{P_{n+1}}{Q_{n+1}}$$

- Belátható, hogy (P_{n+1},Q_{n+1}) is megoldása az egyenletnek.
- Legyen $P_0=6$, $Q_0=4$, ami megoldása az egyenletnek!

Érdekességek – közelítő számítások

- Most nem foglalkozunk a megoldás lépésszámának vizsgálatával.

P:=6
Q:=4
$P*P-2*Q*Q \geq E*Q*Q$
Q:=P*Q
P:=P*P-2

```
P=6; Q=4;
while(P*P-2*Q*Q>=E*Q*Q){
   Q=P*Q; P=P*P-2
}
```

Érdekességek – összes (i-edik) permutáció

Feladat: Állítsuk elő egy N elemű sorozat (1,…,N) összes permutációját!

Másik feladat: Állítsuk elő egy N elemű sorozat i-edik permutációját (0≤i<n!)!

Azaz, ha az i-edik permutációt elő tudjuk állítani, akkor abból az összes permutáció egy egyszerű ciklussal kapható meg.

Érdekességek – összes (i-edik) permutáció

Vegyünk egy rendező módszert!

```
i=1..N-1
    Min:=i
    j=i+1..N
        X[min]>X[j]
            Min:=j | —
    Csere(X[i],X[Min])
    Táv[i]:=Min-i
```

Tároljuk azt, hogy az egyes lépésekben milyen messzire kellett cserélni!

Érdekességek – összes (i-edik) permutáció

A Táv vektor alapján a rendezés hatása visszaalakítható!

i=N-1..1, -1-esével	
	Csere(X[i],X[i+Táv[i]])

```
for(int i=N-1; i>=1; i--){
   y=X[i]; X[i]=X[i+Tav[i]]; X[i+Tav[i]]=y;
}
```

Belátható, hogy minden permutációhoz más és más Táv vektor tartozik.

Kérdés: hogyan lehet egy i értékhez Táv vektort rendelni?

Érdekességek – összes (i-edik) permutáció

- Táv[N-1] értéke 0 vagy 1.
- Táv[N-2] értéke 0, vagy 1, vagy 2.
- …
- Táv[1] értéke 0, vagy 1, …, vagy N-1.
- Azaz Táv egy N-1 jegyű egész szám egy olyan számrendszerben, aminek helyi-értékenként más és más az alapszáma!
- Megoldás: Az i egész szám átírása ebbe a számrendszerbe.

Érdekességek – összes (i-edik) permutáció

- A fenti programrészt összeépítve a rendezés vissza-alakításánál készítettel megadtuk az i-edik permutáció előállításának algoritmusát.

```
j=1..N-1
    Táv[N-j]:=i mod (j+1)
    i:=i div (j+1)
```

```
for(int j=1; j<=N-1; j++){
    Tav[N-j]=i % (j+1);
    i=i/(j+1);
}
```

Az összes permutáció alkalmazása

Feladat:

Jól ismert fejtörő, amelyben egy aritmetikai művelet kapcsol egybe szavakat. A feladat az, hogy a szavak egyes betűinek feleltessünk meg egy számjegyet úgy, hogy a művelet helyes eredményt szolgáltasson a szavakon.

Pl. SEND + MORE = MONEY.

Megoldás:

A szavakban előforduló jelekhez (SENDMORY) keressük a 0..9 számjegyek egyértelmű hozzárendelését.

Az összes permutáció alkalmazása

Megoldási ötlet:

- Az összes permutáció algoritmusára építünk.
- A `Jó` eljárás ellenőrzi a permutáció – a feladat szem-pontjából való – helyességét, és gondoskodik az esetleges megoldás gyűjtéséről vagy kiírásáról.

Az összes permutáció alkalmazása

A megfelelőség a (*) SEND + MORE – MONEY = 0 egyenletre. Ha

- 'S' X(1) értékű, akkor a (*)-ban X(1) *1000-rel van jelen;
- 'E' X(2) értékű, akkor X(2)*(100+1-10)=X(2)*91-gyel;
- 'N' X(3) értékű, akkor X(3)*(10-100)= X(3)*(-90)-nel;
- 'D' X(4) értékű, akkor X(4)*(1)-gyel;
- 'M' X(5) értékű, akkor X(5)*(1000-10000)=X(5)*(-9000)-rel;
- 'O' X(6) értékű, akkor X(6)*(100-1000)= X(6)*(-900)-zal;
- 'R' X(7) értékű, akkor X(7)*10-zel;
- 'Y' X(8) értékű, akkor X(8)*(-1)-gyel van jelen.
- továbbá az S és az M betűhöz nem rendelhetünk nullát, azaz X(1)≠0 és X(5) ≠0!

Az összes permutáció alkalmazása

```
jó(X):
   jó:=(X(1)*1000+X(2)*91+X(3)*(-90)+X(4)*1+X(5)*(-9000)+
        X(6)*(-900)+X(7)*10+X(8)*(-1)=0 és X(1)≠0 és X(5)≠0)
Függvény vége.
```

Ha a konstansokat egy Z vektorban tárolnánk, akkor a Jó függvényben X és Z skaláris szorzatát kellene kiszámolnunk.

```
jó(X):
   jó:=(skalárszorzat(X,Z)=0 és X(1)≠0 és X(5)≠0)
Függvény vége.
```

Programozási alapismeretek

13. előadás vége