Programozás

13. előadás

Beharangozás

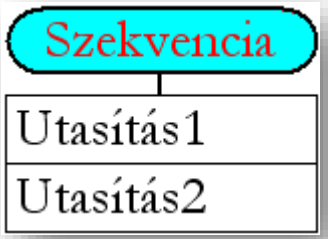
- Érdekes (nem triviálisan megoldható) feladatok következnek a
  - kombinatorika,
  - mohó módon megoldható feladatok,
  - visszalépéses keresés

  témaköréből
- Szokatlan módon nemcsak struktogrammal, hanem az ún. pszeudokóddal fogunk egy-egy algoritmust megadni.

Algoritmikus szerkezetek

struktogram ↔ pszeudokód

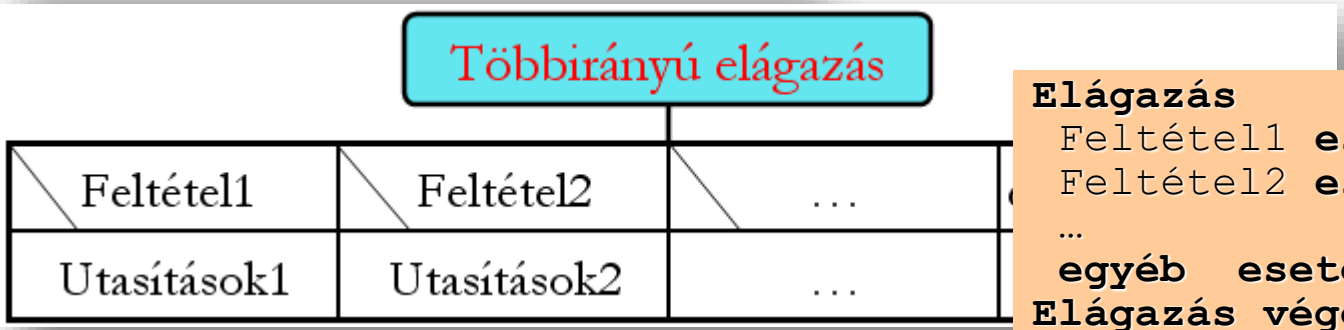
Szekvencia:




```
Utasítás1
Utasítás2
```

Elágazások:


```
Ha Feltétel akkor
   Igaz-ág utasításai
különben
   Hamis-ág utasításai
Elágazás vége
```

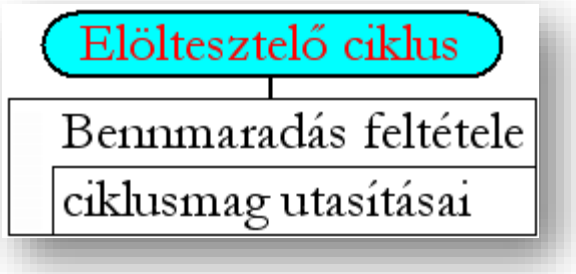



```
Elágazás
  Feltétel1 esetén Utasítások1
  Feltétel2 esetén Utasítások2
  …                …
  egyéb esetekben Utasítások
Elágazás vége
```

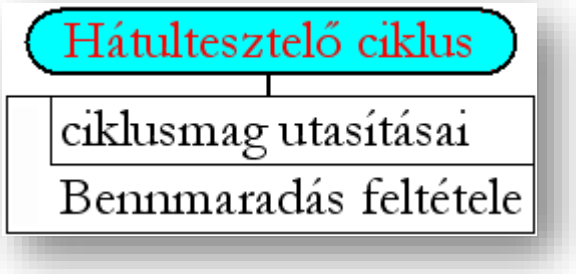
Algoritmikus szerkezetek

struktogram – pszeudokód

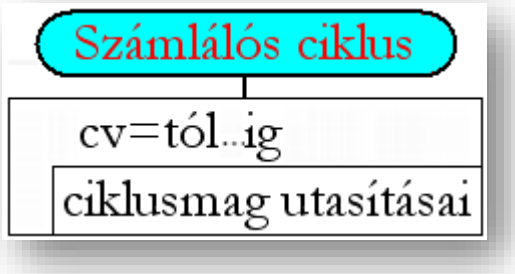
Ciklusok:




```
Ciklus amíg Feltétel
  ciklusmag utasításai
Ciklus vége
```




```
Ciklus
  ciklusmag utasításai
amíg Feltétel
Ciklus vége
```

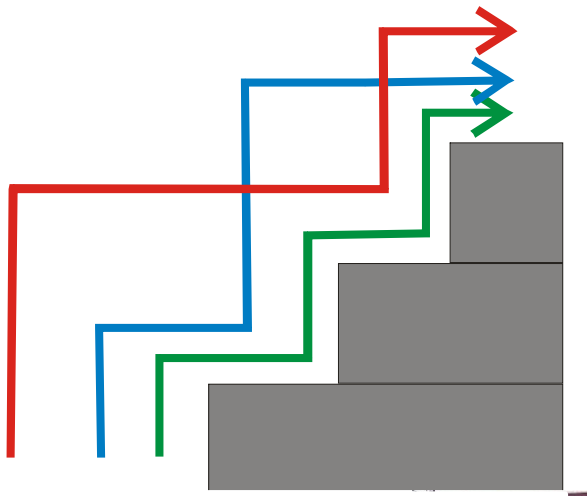



```
Ciklus cv=tól-tól ig-ig
  ciklusmag utasításai
Ciklus vége
```

Érdekességek – Kombinatorika

Feladat:

- 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!



K=2 esetén

Érdekességek – Kombinatorika

- Bemenet: $N,K \in \mathbf{N}$
- Kimenet: $Db \in \mathbf{N}$
- 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: $Db_0=1$ és ???

A „0. lépcsőfokhoz” egyféleképpen juthatunk, de továbbra sem látszik közvetlen összefüggés a bemenet és a $Db_{1..N}$ között.

Érdekességek – Kombinatorika

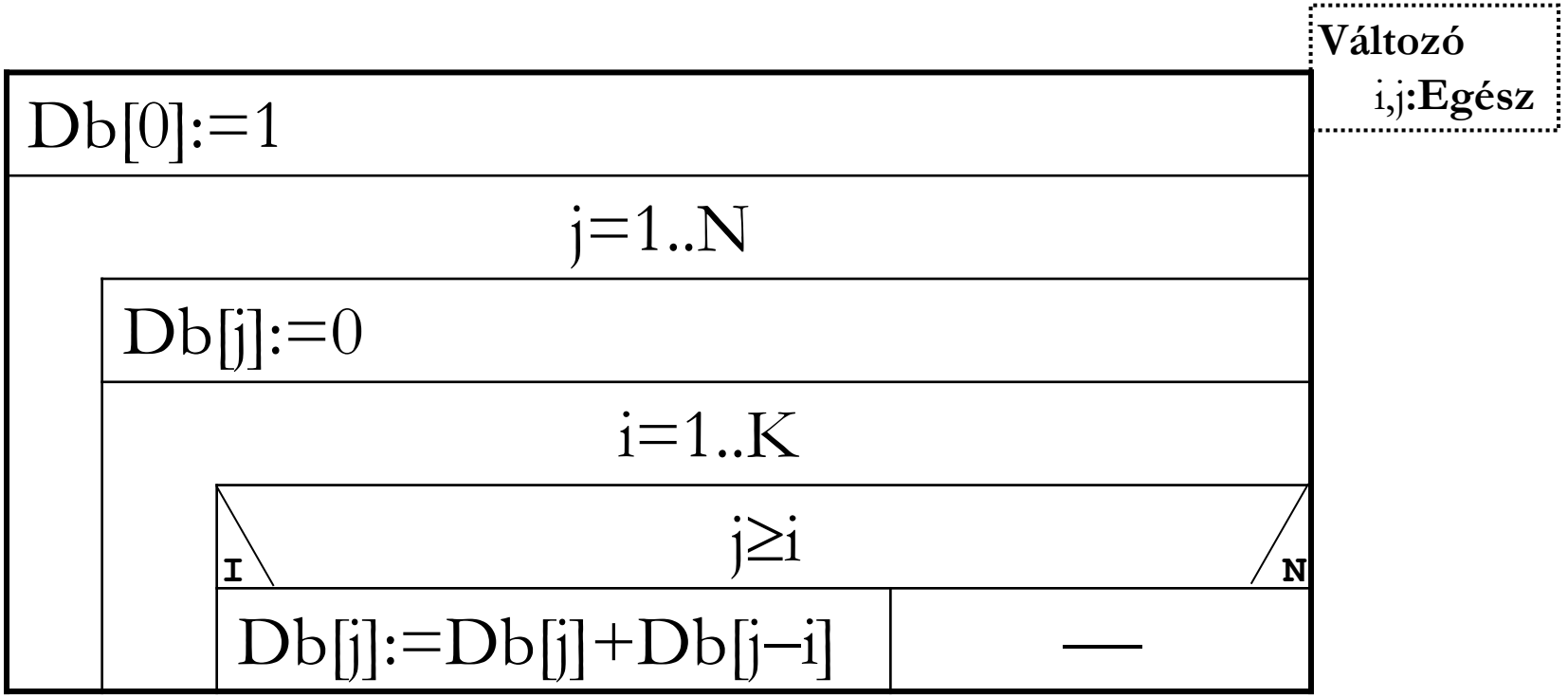
Próbáljunk meg összefüggést felírni a kimenetre önmagában!

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 $\forall j(1\leq j\leq N): Db_j=\sum_{\substack{i=1\\ i\leq j}}^{K} Db_{j-i}$

Tehát eljutottunk a sorozatszámítás (feltételes összegzés) programozási tételhez.

Érdekességek – Kombinatorika



Érdekességek – Kombinatorika

Keverés

Feladat:

- 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!

$$(1, 2,\ldots, N) \rightarrow (X_1, X_2,\ldots, X_N)$$

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

Érdekességek – Keverés

Stratégia:

- Válasszunk az N elem közül egyet véletlenszerűen, és cseréljük meg az elsővel!
- A maradék N–1-ből újra válasszunk véletlenszerűen egyet, s cseréljük meg a másodikkal!
- …

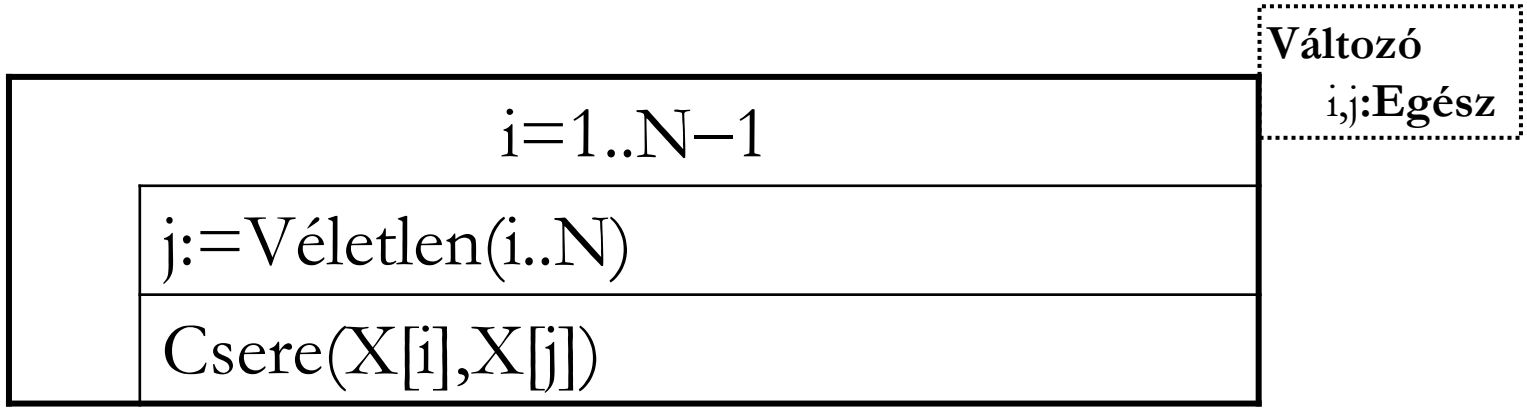
Érdekességek – Keverés

Be kellene látnunk, hogy így jó megoldást kapunk!

(nem bizonyítás, csak gondolatok)

- Nézzük meg, hogy mi annak az esélye, hogy
  - az I kerül az 1. helyre;
    mivel az (összes) N elem közül véletlenszerűen választunk egyet, amelyet megcseréljük az elsővel, ezért az első helyre egyenlő, 1/N eséllyel kerül bármely elem.
  - az I kerül a második helyre:
    ez úgy történhet, hogy az első helyre nem az I került (ennek esélye (N−1)/N), a másodikra pedig igen (ami esélye 1/(N−1));
    tehát (N−1)/N*1/(N−1)=1/N!

Érdekességek – Keverés



Vegyük észre, hogy ez olyan, mint a rendezés, csak nagyság szerinti hely helyett véletlenszerű helyre cserélünk.

Érdekességek – Kombinatorika

Összes, i-edik permutáció

Feladat:

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

Másik feladat:

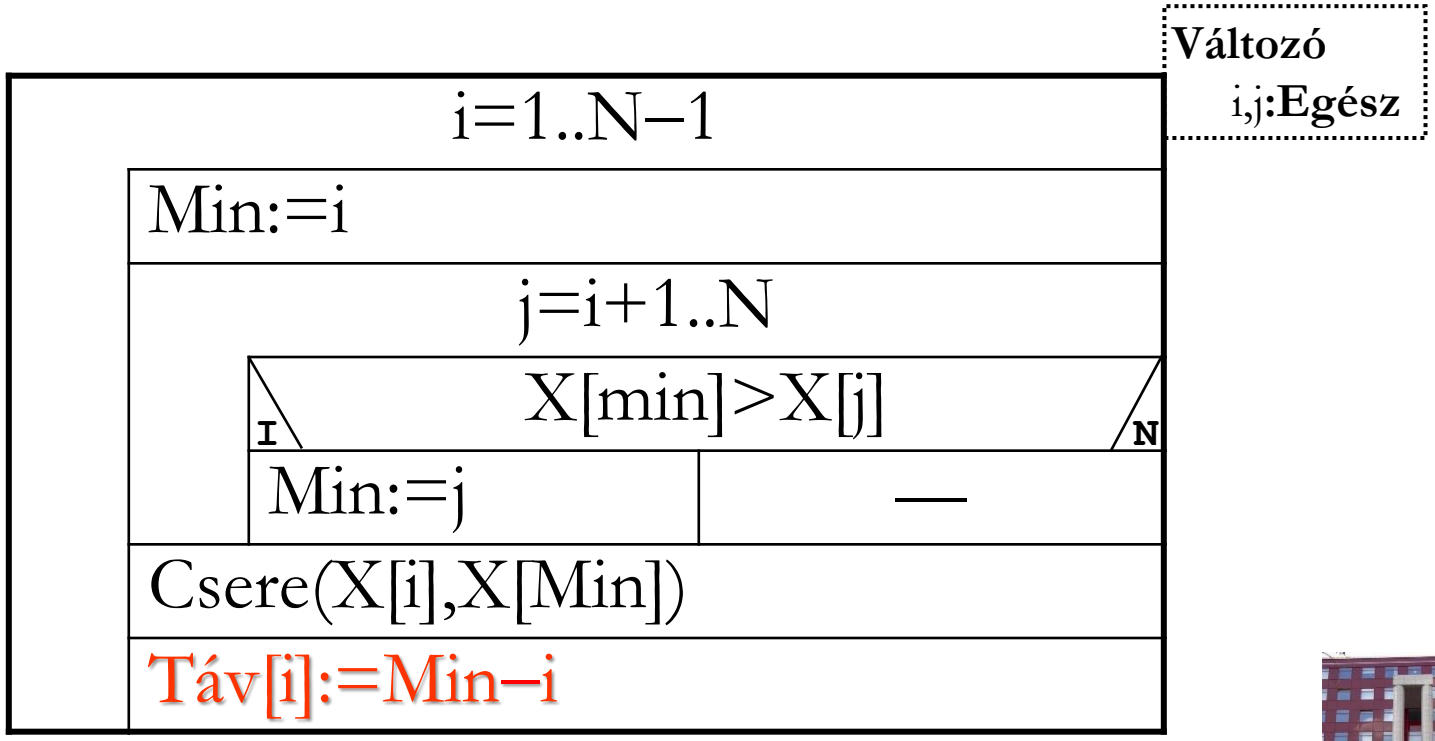
Állítsuk elő egy N elemű sorozat i-edik permutációját ($0 \leq i < n!$)!

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!

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



Érdekességek – Összes (i-edik) permutáció

A Táv vektor alapján a rendezés hatása visszaalakítható!

```
Változó
  i:Egész
i=N–1..1, -1-esével
    Csere(X[i],X[i+Táv[i]])
```

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

Kérdés: hogyan lehet egy i értékhez (értsd a permutáció sorszámához) 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 helyiértékenként 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.

Változó
j:Egész

j=1..N–1
Táv[N–j]:=i Mod (j+1)
i:=i Div (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 szempontjából 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):Logikai
  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.
```

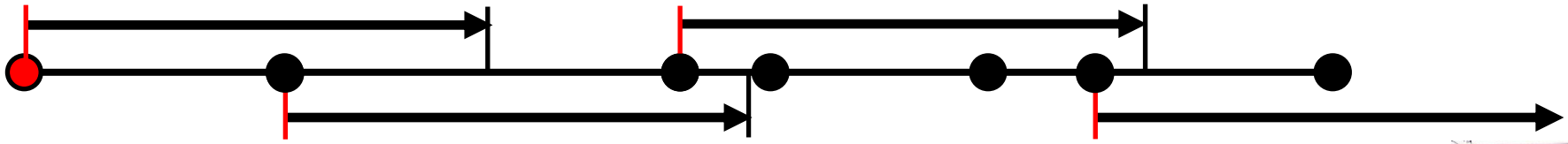
Ha a konstansokat egy Z vektorban tárolnánk, akkor a Jó függvényben X és Z skaláris szorzatát kellene kiszámolnunk.

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

Érdekességek – Mohó stratégia

Feladat:

- 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: $\forall i(1 \leq i < N): B_{i+1} - B_i \leq K$
- Utófeltétel: $Db = ???$ és $T_1 = 1$ és
  $\forall i(1 \leq i < Db): B_{T_{i+1}} - B_{T_i} \leq K$ és
  $B_N - B_{T_{Db}} \leq K$ és $T \subseteq (1,\ldots,N-1)$

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!

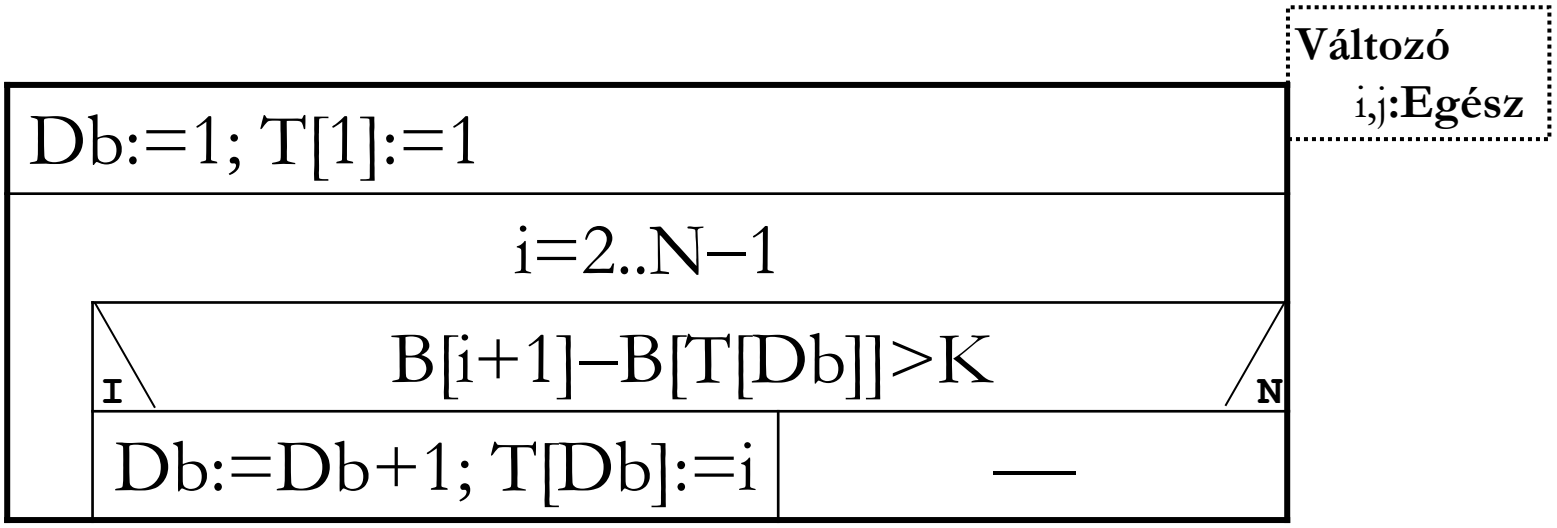
Érdekességek – Mohó stratégia

Megoldás (tegyük fel, hogy van megoldás):

- Budapesten mindenképpen kell tankolni!
- Menjünk, ameddig csak lehet, s a lehető legutolsó benzinkútnál tankoljunk!
- Mindezt addig, amíg Párizsba el nem jutunk.
- Belátható, hogy ezzel egy optimális megoldást kapunk.

→ **Kiválogatás!**

Érdekességek – Mohó stratégia



Érdekességek – Visszalépéses keresés

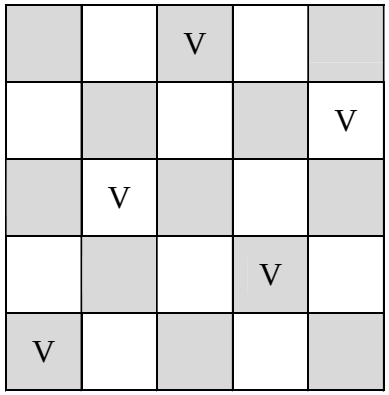
Feladat:

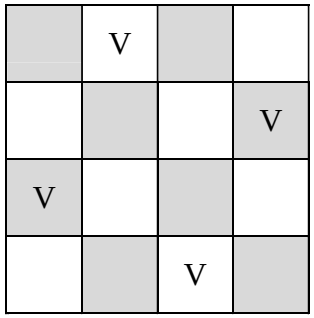
- Helyezzünk el egy N×N-es sakktáblán N vezért úgy, hogy ne üssék egymást!
- A vezérek a sorukban, az oszlopukban és az átlójukban álló bábukat üthetik. Tehát úgy kell elhelyezni a vezéreket, hogy minden sorban és minden oszlopban is pontosan 1 vezér legyen, és minden átlóban legfeljebb 1 vezér legyen!

Érdekességek – Visszalépéses keresés

N vezér elhelyezése egy N×N-es sakktáblán:

- Helyezzünk el egy N×N-es sakktáblán N vezért úgy, hogy ne üssék egymást!
- Egy lehetséges megoldás N=5-re és N=4-re:





Érdekességek – Visszalépéses keresés

Stratégia:

- 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 az oszlopból, amelyikből visszaléptünk.
- 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).

Érdekességek – Visszalépéses keresés

Visszalépéses keresés algoritmus:

```
Keresés(N,Van,S):
   i:=1; Y[1..N]:=(0,...,0)  [Y[i]: i. választás]
   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!

Érdekességek – Visszalépéses keresés

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.

Érdekességek – Visszalépéses keresés

Visszalépéses keresés algoritmus:

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

üti(i,j,k,l):Logikai
   üti:=(l=j) vagy (i-k=abs(j-l))
Függvény vége.
```

Megjegyzés: Rossz egy választás, ha valamelyik korábbi választás miatt nem szabad (eldöntés tétel).

Érdekességek – Visszalépéses keresés

Feladat: 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.

`M[i]` – az i. munkás ennyi munkához ért

`E[i,j]` – az i. munkás által elvégezhető j. munka

	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**

Érdekességek – Visszalépéses keresés

N munka – N jelentkező:

```
Keresés(N,Van,Y):
  i:=1; Y[1..N]:=(0,...,0)  [Y[i]: i. választás]
  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.
```

Érdekességek – 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.
```

Érdekességek – Visszalépéses keresés

N munka – N jelentkező:

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

E[i,j] – az i. munkás által elvégezhető j. munka

Érdekességek – Közelítő számítások

- Feladat: Számítsuk ki a 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: $E \in \mathbb{R}$
- Kimenet: $P,Q \in \mathbb{N}$
- 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) számpárok **sorozatát** úgy, hogy $|P_{i+1}^2/Q_{i+1}^2-2|<|P_i^2/Q_i^2-2|$ legyen! (konvergencia)
- 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:

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

(Most ezt sem bizonyítjuk.)

- 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. (Négyzetesen gyors.)

```
P:=6; Q:=4
    P*P–2*Q*Q≥E*Q*Q
        Q:=P*Q; P:=P*P–2
```

Programozás

13. előadás vége