

# Mesterséges intelligencia

Gregorics Tibor  
[people.inf.elte.hu/gt/mi](http://people.inf.elte.hu/gt/mi)

# *Szakirodalom*

## ❑ Könyvek

- Fekete István - Gregorics Tibor - Nagy Sára: Bevezetés a mesterséges intelligenciába, LSI Kiadó, Budapest, 1990, 1999. ELTE-Eötvös Kiadó, Budapest, 2006.
- Russel, J. S., Norvig, P.: MI - modern megközelítésben, Panem Kft, 2005.
- Futó Iván (szerk): Mesterséges intelligencia, Aula Kiadó, Budapest, 1999.

## ❑ Internet

- [people.inf.elte.hu/gt/m](http://people.inf.elte.hu/gt/m)



# Bevezetés

# 1. AZ MI FOGALMA

*mesterséges intelligencia – MI (artificial intelligence - AI)*

## Erős MI

Az emberi gondolkodás reprodukálható számítógéppel.

## MI szkeptikusok

A számítógép soha nem lesz okosabb az embernél.

## Gyenge MI

*Az MI kutatja, fejleszti, rendszerezi azokat az elméleteket és módszereket, amelyek hozzájárulhatnak az intelligens gondolkodás számítógéppel való reprodukálásához.*

MI nem egy speciális részterülete az informatikának, hanem egy törekvés, hogy a számítógéppel olyan érdekes és nehéz problémákat oldjunk meg, amelyek megoldásában ma még az ember jobb.

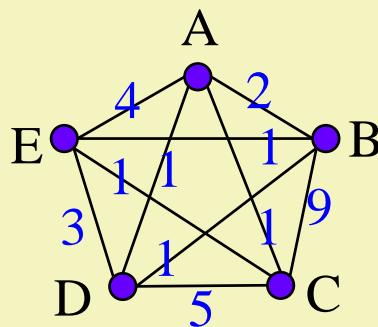
# *Miről ismerhető fel egy szoftverben az MI?*

- Megoldandó feladat: nehéz
  - A feladat **problémátere** hatalmas,
- Szoftver viselkedése
- Felhasznált eszközök



# Utazó ügynök problémája

Adott  $n$  város a közöttük vezető utak költségeivel. Melyik a legolcsóbb olyan útvonal, amely az  $A$  városból indulva mindegyik várost egyszer érintve visszatér az  $A$  városba?



lehetséges utak:

$n$	$(n-1)!$
5	24
50	$6 \cdot 10^{62}$

ABCDEA

ACBDEA

ABDCEA

ABCDEA

ABDECA

problématér

ADBECA

...

# *Miről ismerhető fel egy szoftverben az MI?*

## Megoldandó feladat: nehéz

- A feladat **problémateré** hatalmas,
- szisztematikus keresés helyett intuícióra, kreativitásra (azaz **heurisztikára**) van szükségünk ahhoz, hogy elkerüljük a kombinatorikus robbanást.

## Szoftver viselkedése

## Felhasznált eszközök



7-11

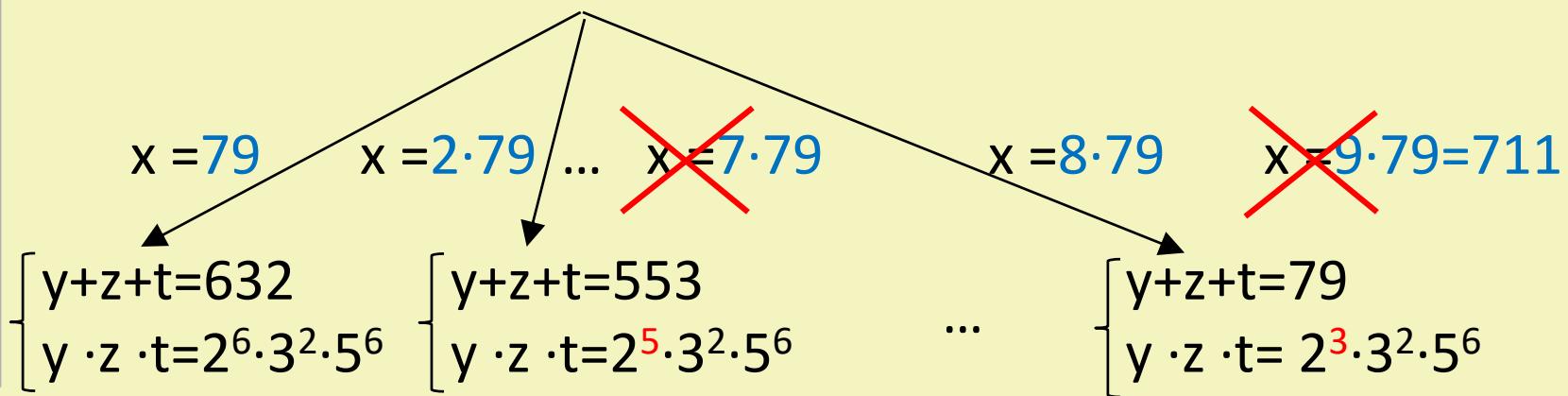
$$x=? , y=? , z=? , t=?$$

$$\begin{cases} x + y + z + t = 7.11 \\ x \cdot y \cdot z \cdot t = 7.11 \end{cases}$$

↓

$$x, y, z, t \in \{1, \dots, 708\}$$

$$\begin{cases} x + y + z + t = 711 \\ x \cdot y \cdot z \cdot t = 711 \cdot 10^6 = 2^6 \cdot 3^2 \cdot 5^6 \cdot 79 \end{cases}$$



# *Miről ismerhető fel egy szoftverben az MI?*

## Megoldandó feladat : nehéz

- A feladat **problémateré** hatalmas,
- szisztematikus keresés helyett intuícióra, kreativitásra (azaz **heurisztikára**) van szükségünk ahhoz, hogy elkerüljük a kombinatorikus **robbanást**.

## Szoftver viselkedése : intelligens

- Turing teszt

## Felhasznált eszközök

I feel you are bored with me lately.

**Pattern:** I <a> you <b> me <c>.

1. Why do you think that you <a> I <b> you <c>?
2. Let us suppose that I <b> you <c>. Would that make a difference?

**Recall:**

„I am getting tired of replying the same sentence over and over.”

**Carry on:**

What else do you want to talk about?

I see. Please continue. This is very interesting.

# Miről ismerhető fel egy szoftverben az MI?

## ❑ Megoldandó feladat: nehéz

- A feladat **problémateré** hatalmas,
- szisztematikus keresés helyett intuícióra, kreativitásra (azaz **heurisztikára**) van szükségünk ahhoz, hogy elkerüljük a **kombinatorikus robbanást**.

## ❑ Szoftver viselkedése: intelligens

- Turing teszt vs. kínai szoba elmélet
- „mesterjelölt szintű” és „egy problémára fókuszál”

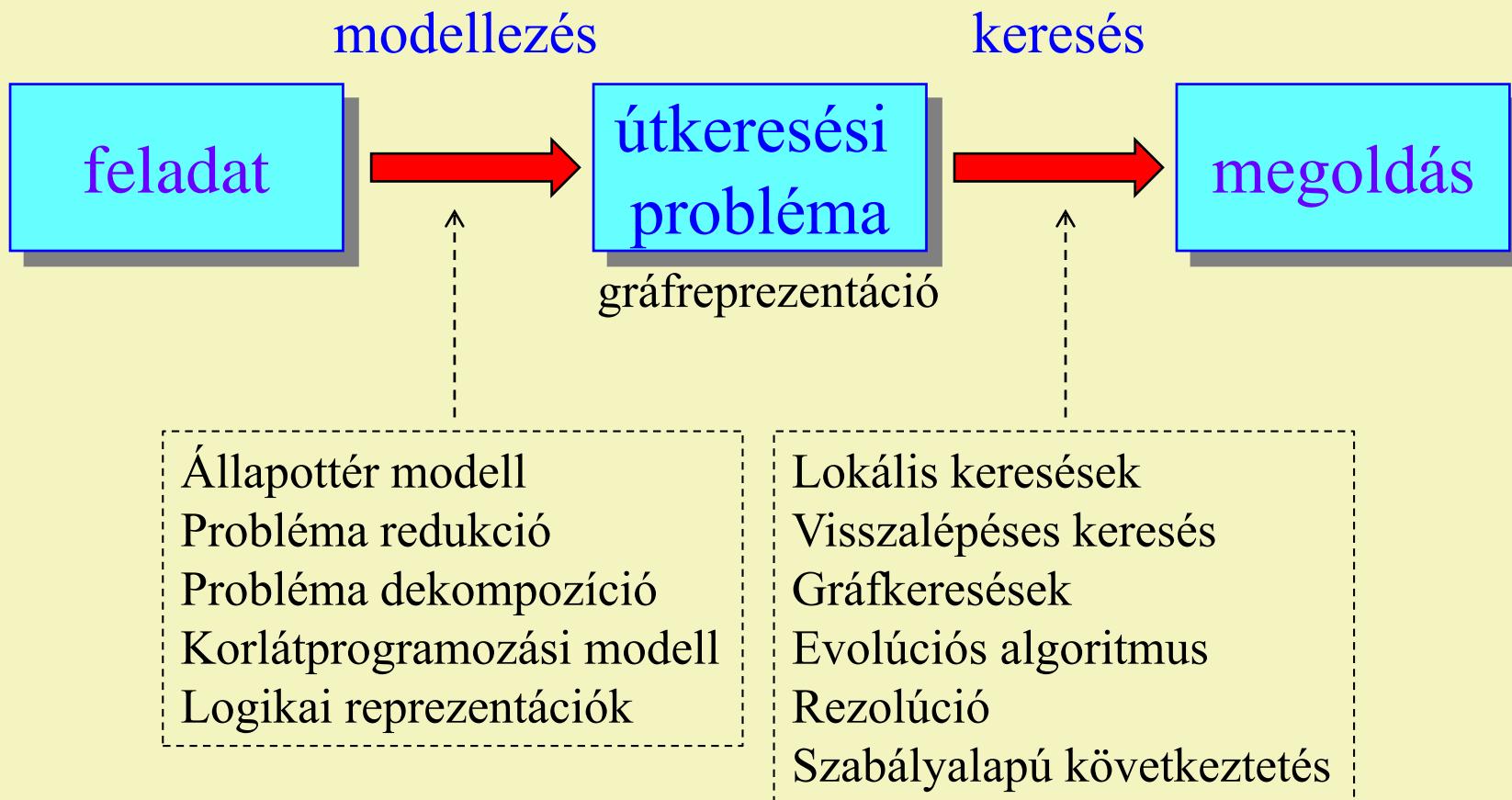
## ❑ Felhasznált eszközök: sajátosak

- átgondolt reprezentáció a feladat **modellezéséhez**
- heurisztikával megerősített hatékony **algoritmusok**
- **gépi tanulás módszerei**

## Intelligens szoftver jellemzői

- megszerzett ismeret tárolása
- automatikus következtetés
- tanulás
- term. nyelvű kommunikáció
- + gépi látás, gépi cselekvés

## 2. MODELLEZÉS & KERESÉS



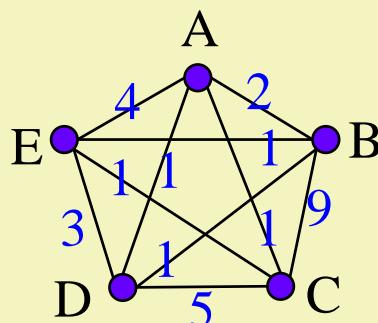
# *Mire kell a modellezésnek fókuszálni*

- **Problématér elemei:** probléma lehetséges válaszai.
- **Cél:** egy helyes válasz (**megoldás**) megtalálása
- **Keresést segítő ötletek** (heurisztikák):
  - Problématér **hasznos elemeinek** elválasztása a haszontalanoktól.
  - **Kiinduló elem** kijelölése.
  - Az elemek **szomszédsági kapcsolatainak** kijelölése, hogy a probléma tér elemeinek szisztematikus bejárását segítsük.
  - Adott pillanatban elérhető **elemek rangsorolása**.



# Utazó ügynök problémája

Adott  $n$  város a közöttük vezető utak költségeivel. Melyik a legolcsóbb olyan útvonal, amely az  $A$  városból indulva mindegyik várost egyszer érintve visszatér az  $A$  városba?

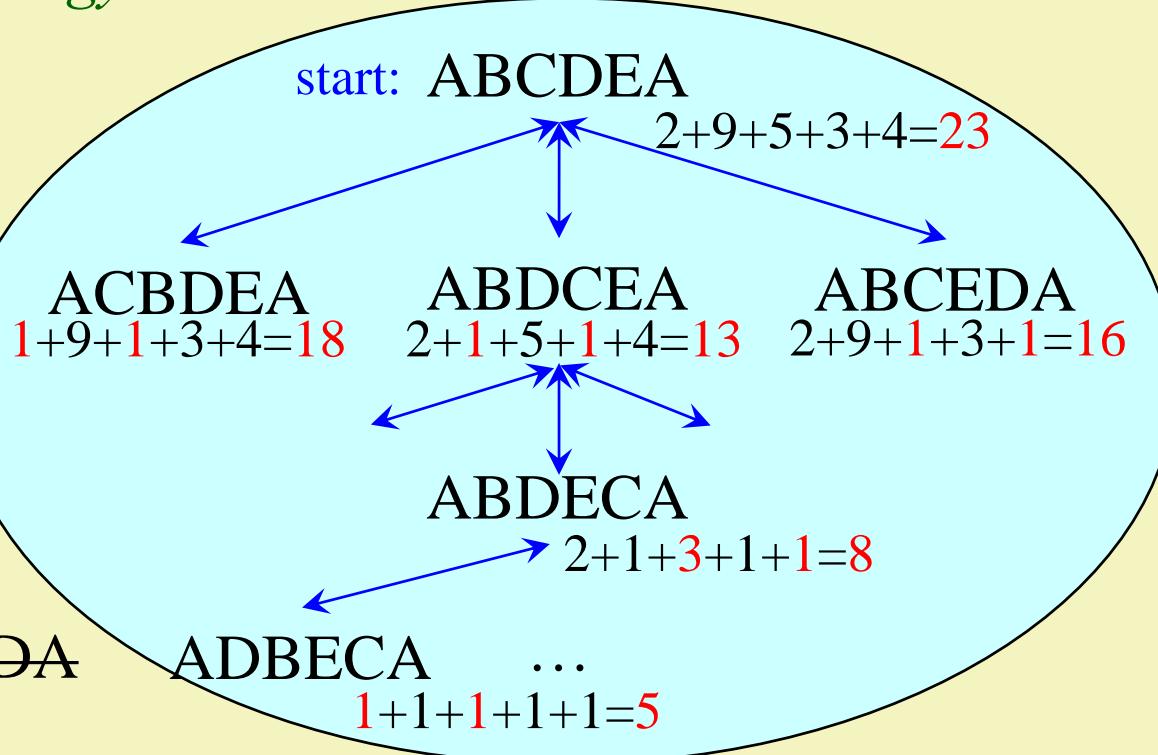


~~AEDCBA~~

~~AEDBCA~~

~~ACEDBA~~

felesleges → ~~ACEBDA~~



# Útkeresési probléma

- Számos olyan modellező módszert ismerünk, amely a kitűzött feladatot útkeresési problémává fogalmazza át.
- Az útkeresési probléma megoldását egy alkalmas élsúlyozott irányított gráfnak vagy egy adott (cél-) csúcsa, vagy egy adott (startcsúcsból célcsúcsba vezető) útja (esetleg a legolcsóbb ilyen út) szimbolizálja.
- Ez a gráf lehet végtelen nagy, de csúcsainak kifoka véges, és van egy konstans globális pozitív alsó korlátja ( $\delta$ ) az éleinek súlyának (költségének) ( $\delta$ -gráf).

# Gráf fogalmak 1.

- csúcsok, irányított élek
- él  $n$ -ből  $m$ -be
- $n$  utódai
- $n$  szülei
- irányított gráf
- véges sok kivezető él
- élköltség
- $\delta$ -tulajdonság ( $\delta \in \mathbb{R}^+$ )
- $\delta$ -gráf

$$N, A \subseteq N \times N \text{ (végtelen számosság)}$$
$$(n, m) \in A \quad (n, m \in N)$$
$$\Gamma(n) = \{m \in N \mid (n, m) \in A\}$$
$$\pi(n) \in \Pi(n) = \{m \in N \mid (m, n) \in A\}$$
$$R = (N, A)$$
$$|\Gamma(n)| < \infty \quad (\forall n \in N)$$
$$c: A \rightarrow \mathbb{R}$$
$$c(n, m) \geq \delta > 0 \quad (\forall (n, m) \in A)$$

$\delta$ -tulajdonságú, véges sok kivezető élű, élsúlyozott irányított gráf

# Gráffogalmak 2.

- irányított út

$\delta$ -gráfokban ez végtelen sok út esetén is értelmes.

Értéke  $\infty$ , ha nincs egy út se.

- út hossza
- út költsége
- opt. költség
- opt. költségű út

$$\alpha = (n, n_1), (n_1, n_2), \dots, (n_{k-1}, m)$$
$$= \langle n, n_1, n_2, \dots, n_{k-1}, m \rangle$$

$$n \rightarrow^\alpha m, n \rightarrow m, n \rightarrow M \quad (M \subseteq N)$$

$$\{n \rightarrow m\}, \{n \rightarrow M\} \quad (M \subseteq N)$$

az út éleinek száma:  $|\alpha|$

$$c(\alpha) = c^\alpha(n, m) := \sum_{i=1..k} c(n_{i-1}, n_i)$$

$$\text{ha } \alpha = \langle n=n_0, n_1, n_2, \dots, n_{k-1}, m=n_k \rangle$$

$$c^*(n, m) := \min_{\alpha \in \{n \rightarrow m\}} c^\alpha(n, m)$$

$$c^*(n, M) := \min_{\alpha \in \{n \rightarrow M\}} c^\alpha(n, m)$$

$$n \rightarrow^* m := \min_c \{ \alpha \mid \alpha \in \{n \rightarrow m\} \}$$

$$n \rightarrow^* M := \min_c \{ \alpha \mid \alpha \in \{n \rightarrow M\} \}$$

# Gráfprezentáció fogalma

- minden útkeresési probléma rendelkezik egy (a probléma modellezéséből származó) gráfprezentációval, ami egy  $(R, s, T)$  hármás, amelyben
  - $R = (N, A, c)$   $\delta$ -gráf az ún. reprezentációs gráf,
  - az  $s \in N$  startcsúcs,
  - a  $T \subseteq N$  halmazbeli célcímsúcsok.
- és a probléma megoldása:
  - egy  $t \in T$  cél megtalálása, vagy
  - egy  $s \rightarrow T$ , esetleg  $s \rightarrow^* T$  optimális út megtalálása

$s$ -ból  $T$  egyik csúcsába vezető irányított út

$s$ -ból  $T$  egyik csúcsába vezető legolcsóbb irányított út

# Keresés

- Az útkeresési problémák megoldásához azok reprezentációs gráfjainak nagy mérete miatt speciális (nem determinisztikus, heurisztikus) útkereső algoritmusokra van szükség, amelyek
  - a startcsúcsból **indulnak** (kezdeti aktuális csúcs);
  - minden lépésben **nem-determinisztikus** módon új aktuális csúco(ka)t **választanak** a korábbi aktuális csúcs(ok) alapján (gyakran azok gyerekei közül);
  - **tárolják** a már feltárt reprezentációs gráf egy részét;
  - **megállnak**, ha célcímsúcsot találnak vagy nyilvánvalóvá válik, hogy erre semmi esélyük.

# Kereső rendszer (KR)

## Procedure KR

1. **ADAT** := kezdeti érték
  2. **while**  $\neg$ terminálási feltétel(**ADAT**) **loop**
  3.     **SELECT SZ FROM alkalmazható szabályok**
  4.     **ADAT** := **SZ(ADAT)**
  5. **endloop**
- end**

**globális munkaterület**

tárolja a keresés során megszerzett és megőrzött ismeretet (egy részgráfot)  
(kezdeti érték ~ start csúcs,  
terminálási feltétel ~ célcímsúcs)

**keresési szabályok**

megváltoztatják a globális  
munkaterület tartalmát  
(előfeltétel, hatás)

**vezérlési stratégia**

alkalmazható szabályok közül  
kiválaszt egy „megfelelőt”  
(általános elv + heurisztika)

# *Kereső rendszerek vizsgálata*

- helyes-e (azaz korrekt választ ad-e)
- teljes-e ( minden esetben választ ad-e)
- optimális-e (optimális megoldást ad-e)
- idő bonyolultság
- tár bonyolultság

# 3. GÉPI TANULÁS

- ❑ Egy algoritmus tanul, ha egy feladat megoldása során olyan változások következnek be a működésében, hogy később ugyanazt a feladatot vagy hasonló feladatokat jobban (eredmény, hatékonyság) képes megoldani, mint korábban.
- ❑ Gépi tanulással a feladat modelljét (reprezentációját és/vagy heurisztikáit), illetve a megoldó algoritmust (többnyire annak bizonyos paramétereit) lehet automatikusan előállítani.
- ❑ A tanuláshoz a megoldandó probléma néhány konkrét esetére, tanító példákra van szükség. A tanulás attól függően lesz **felügyelt**, **felügyelet nélküli**, vagy **megerősítéses**, hogy a tanító példák input-output párok, csak inputok, vagy input-hasznosság párok.



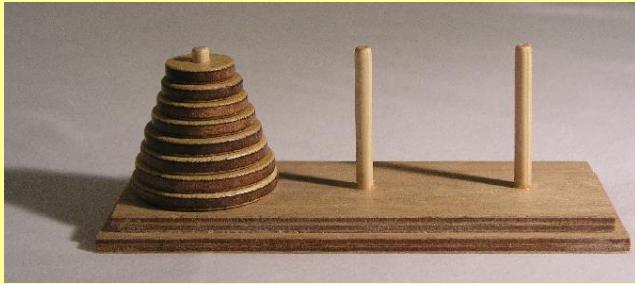
# Modellezés

# 1. Állapottér modell

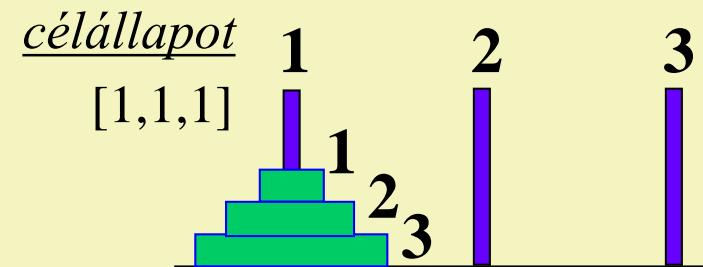
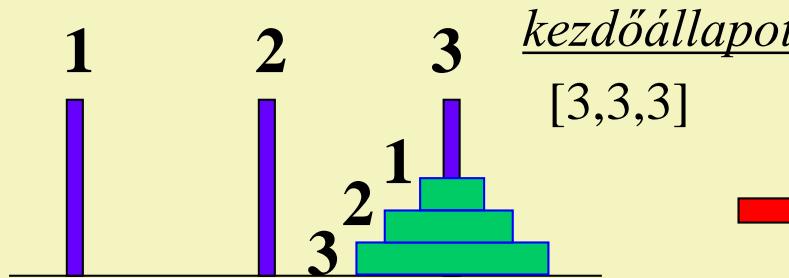
- **Állapottér**: a probléma leírásához szükséges adatok által felvett érték-együttesek (azaz **állapotok**) halmaza
  - az állapot többnyire egy **összetett szerkezetű** érték
  - gyakran egy bővebb alaphalmazzal és egy azon értelmezett **invariáns állítással** definiáljuk
- **Műveletek**: állapotból állapotba vezetnek
  - megadásukhoz: **előfeltétel** és **hatás** leírása
  - invariáns tulajdonságot tartó leképezés
- **Kezdőállapot(ok)** vagy azokat leíró kezdeti feltétel
- **célállapot(ok)** vagy célfeltétel

# *Állapottér modell állapot-gráfja*

- |   |   |  |
|---|---|--|
| <input type="checkbox"/> Állapottér modell  |   | Állapot-gráf                                       |
| ○ állapot   | ~ | csúcs  |
| ○ művelet hatása egy állapotra  | ~ | irányított él                                      |
| ○ művelet költsége  | ~ | él költsége  |
| ○ kezdő állapot   | ~ | startcsúcs   |
| ○ célállapot  | ~ | célcsúcs   |
| <input type="checkbox"/> Gráf-reprezentáció: állapot-gráf, startcsúcs, célcsúcsok |   |  |
| ○ egy műveletsorozat hatása   | ~ | irányított út                                      |
| ○ megoldás  | ~ | irányított út a<br>startcsúcsból egy<br>célcsúcsba |



# Hanoi tornyai probléma



Állapottér:  $AT = \{1, 2, 3\}^n$

1..n intervallummal indexelt egydimenziós tömb,  
amely elemei az {1,2,3} halmazból származnak.

megjegyzés : a tömb  $i$ -dik eleme mutatja az  $i$ -dik korong rúdjának számát; a korongok a rudakon méretük szerint fentről lefelé növekvő sorban vannak.

Művelet: **Rak(honnan, hova)**: $AT \rightarrow AT$

HA a honnan és hova létezik és nem azonos, és van korong a honnan rúdon, és a hova rúd legyen vagy üres vagy felső korongja nagyobb, mint mozgatandó korong (honnan rúd felső korongja)

AKKOR **this[honnan legfelső korongja] := hova**

this:AT az aktuális állapot

# Implementáció

```
template <int n = 3>
class Hanoi {
    int _a[n];           // its elements are between 1 and 3
public:
    bool move (int from, int to) {
        if ((from<1 || from>3 || to<1 || to>3) || (from==to)) return false;
        bool l1; int i;   // l1 ~ 'from' is not empty, i ~ upper disc on 'from'
        for(l1=false, i=0; !l1 && i<n; ++i) l1 = (_a[i]==from);
        if (! l1) return false;
        bool l2; int j;   // l2 ~ 'to' is not empty,   j ~ upper disc on 'to'
        for(l2=false, j=0; !l2 && j<n; ++j) l2 = (_a[j]==to) ;
        if ( ¬l2 || i<j ){ _a[i] = to; return true; } else return false;
    }
    bool final() const { bool l=true; for(int i=0; l && i<n; ++i) l = (_a[i]==1); return l; }
    void init() { for(int i=0;i<n;++i) _a[i] = 3; }
};
```

# Hanoi tornyai

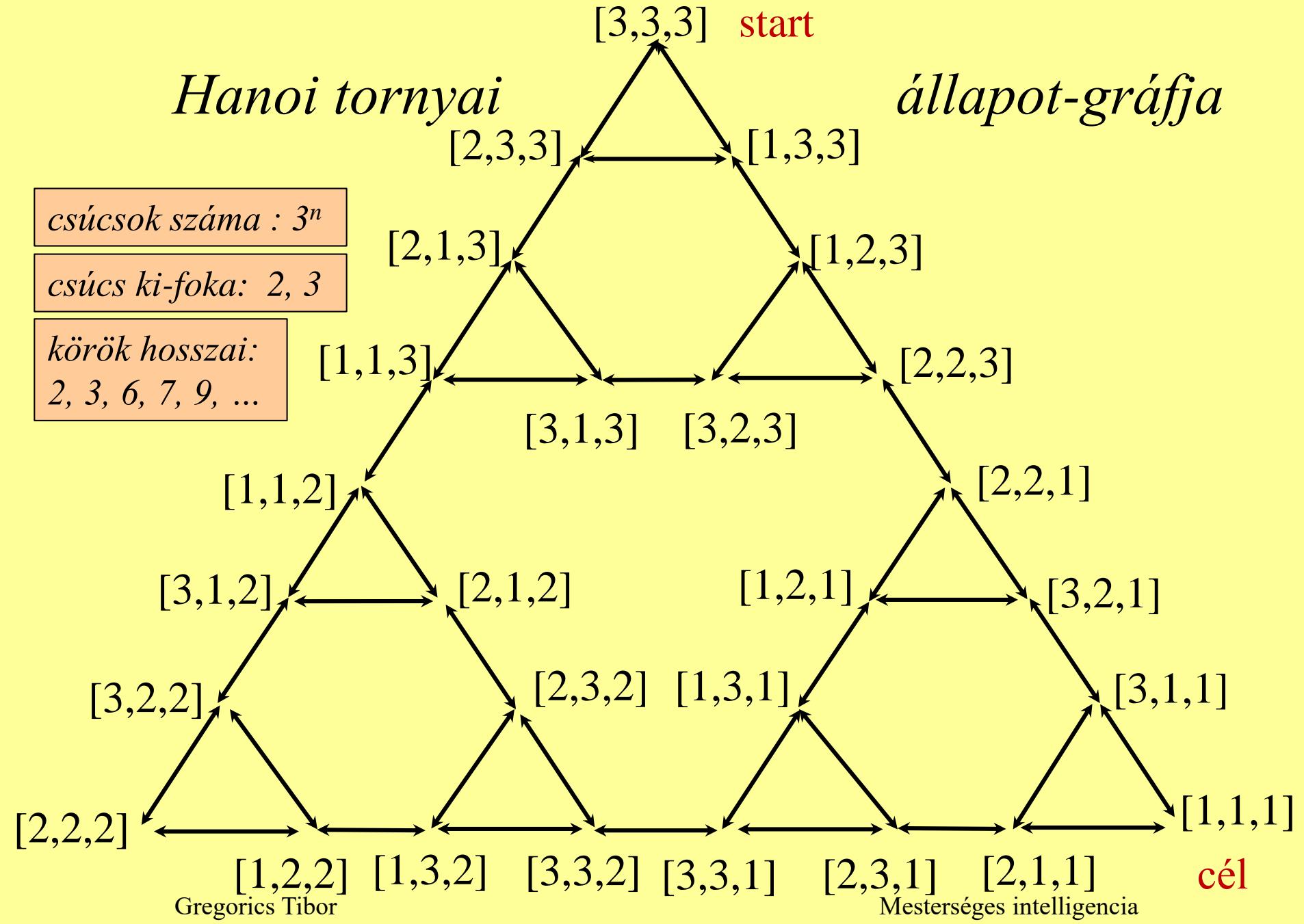
[3,3,3] start

állapot-gráfja

csúcsok száma :  $3^n$

csúcs ki-foka: 2, 3

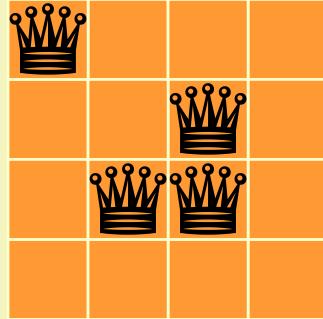
körök hosszai:  
2, 3, 6, 7, 9, ...



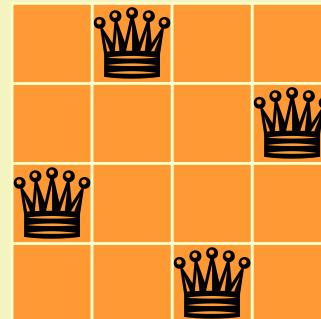


# *n-királynő probléma 1.*

általános állapot



utófeltételnek megfelelő állapot



Állapottér:  $AT = \{\text{女王}, \text{空}\}^{n \times n}$

kétdimenziós tömb ( $n \times n$ -es mátrix),  
mely elemei {女王, 空} halmazbeliek

invariáns: egy állapot (tábla) pontosan  $n$  darab királynőt tartalmaz

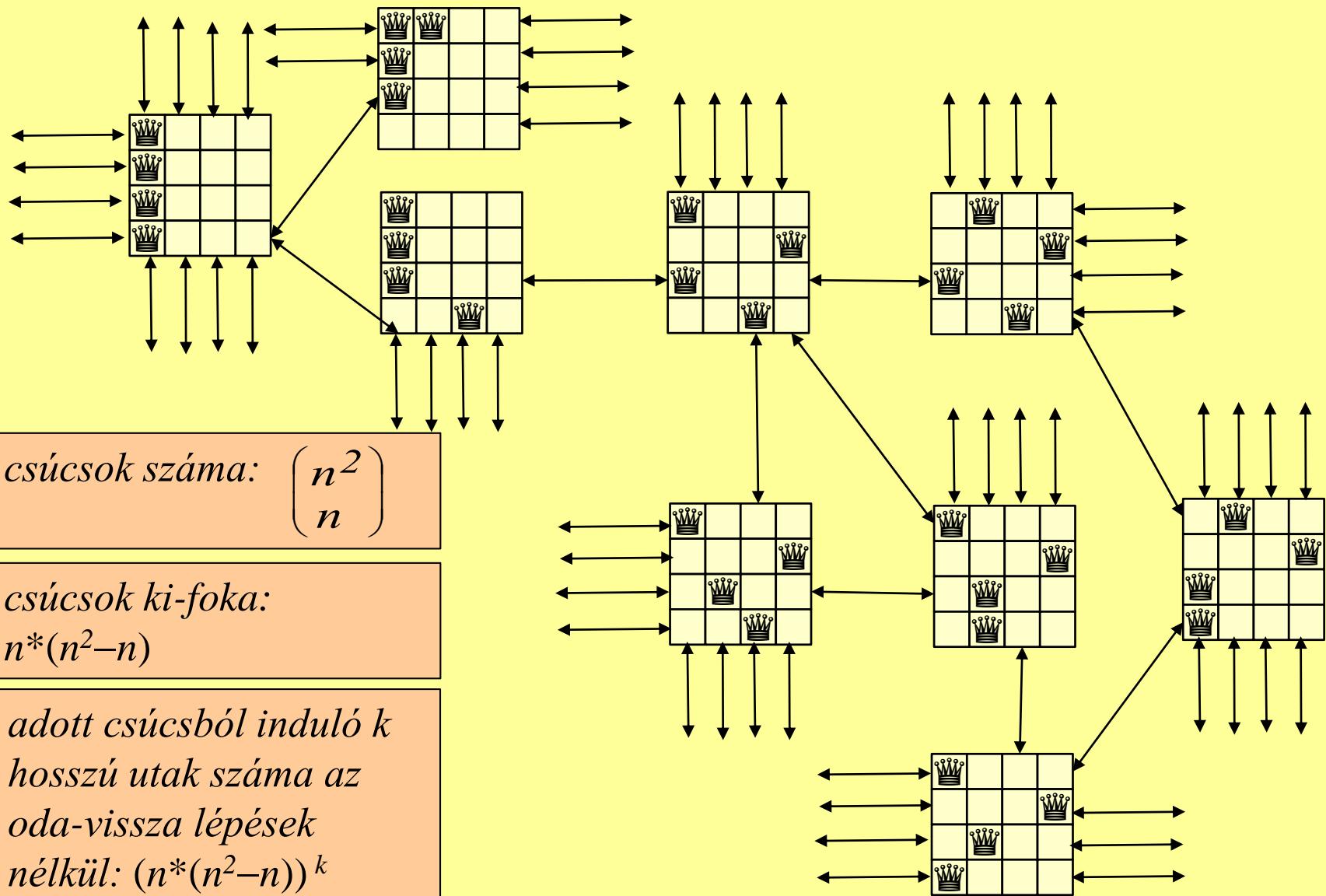
Művelet:  $\text{Athelyez}(x,y,u,v):AT \rightarrow AT$  (this:AT)

HA  $1 \leq x,y,u,v \leq n$  és  $this[x,y] = \text{女王}$  és  $this[u,v] = \text{空}$

AKKOR  $this[x,y] \leftrightarrow this[u,v]$

csere

# Állapot-gráf részlet



# *Állapottér vs. problématér*

- A problématér elemeit (lehetséges megoldásokat) a gráfprezentációbeli startcsúcsból induló különböző hosszúságú irányított utak szimbolizálják.
- Egy feladat állapottér modellje és problémateré között szoros kapcsolat áll fenn, de az állapottér nem azonos a problématérrel.
  - A Hanoi tornyai problémánál a megoldások a startcsúcsból célcsúcsba vezető irányított utak.
  - Az n-királynő problémánál egy állapotot (célcsúcsot) keresünk, de ebben az esetben is egy alkalmas operátor-sorozat (azaz irányított út) vezet el ahhoz, azaz végső soron ilyenkor is utat keresünk.

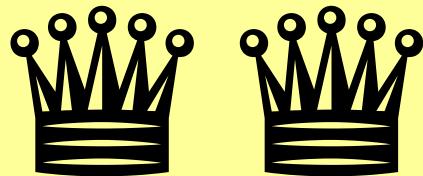
# *Állapot-gráf bonyolultsága*



- A bonyolultság a start csúcsból kivezető utak számától függ, amely nyilván függvénye a
  - csúcsok és élek számának
  - csúcsok ki-fokának
  - körök gyakoriságának, és hosszuk sokféleségének

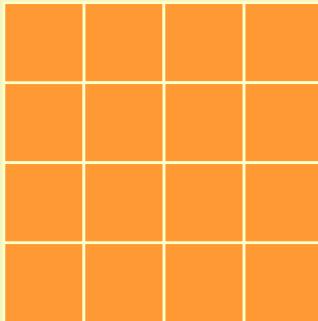
# *Csökkentsük a problématér méretét*

- Ugyanannak a feladatnak több modellje lehet : érdemes olyat keresni, amely kisebb problémateret jelöl ki.
  - Az  $n$ -királynő problématerének mérete, az előző modell szerinti a lehetséges utak száma, óriási. Adjunk jobb modellt!
  - **Bővítsük az állapotteret** az  $n$ -nél kevesebb királynőt tartalmazó állásokkal, és **használjunk új műveletet** : **királynő-felhelyezést** (kezdő állás az üres tábla).
  - **Műveletek előfeltételének szigorításával** csökken az állapotgráf átlagos ki-foka:
    - **Sorról sorra haladva csak egy-egy királynőt** helyezzünk fel a táblára!
    - **Ütést tartalmazó állásra ne tegyük királynőt!**

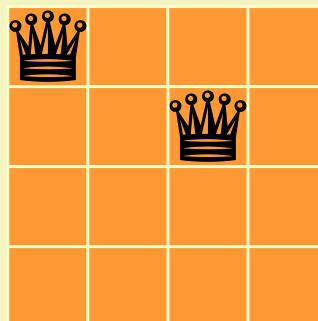


## *n-királynő probléma 2.*

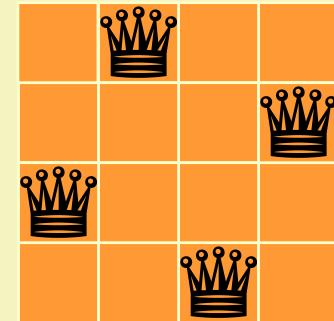
kezdőállapot



közülüső állapot



célállapot



Állapottér:  $AT = \{ \text{퀸}, \_ \}^{n \times n}$

*nincs már üres sor és nincs ütés*

*invariáns:* az első néhány sor egy-egy királynőt tartalmaz

Művelet: **Helyez(oszlop):**  $AT \rightarrow AT$       (*this:AT*)

HA       $1 \leq \text{oszlop} \leq n$  és a *this*-beli soron következő üres sor  $\leq n$   
 és nincs ütés a *this*-ben

AKKOR *this*[a *this*-beli soron következő üres sor, oszlop] :=

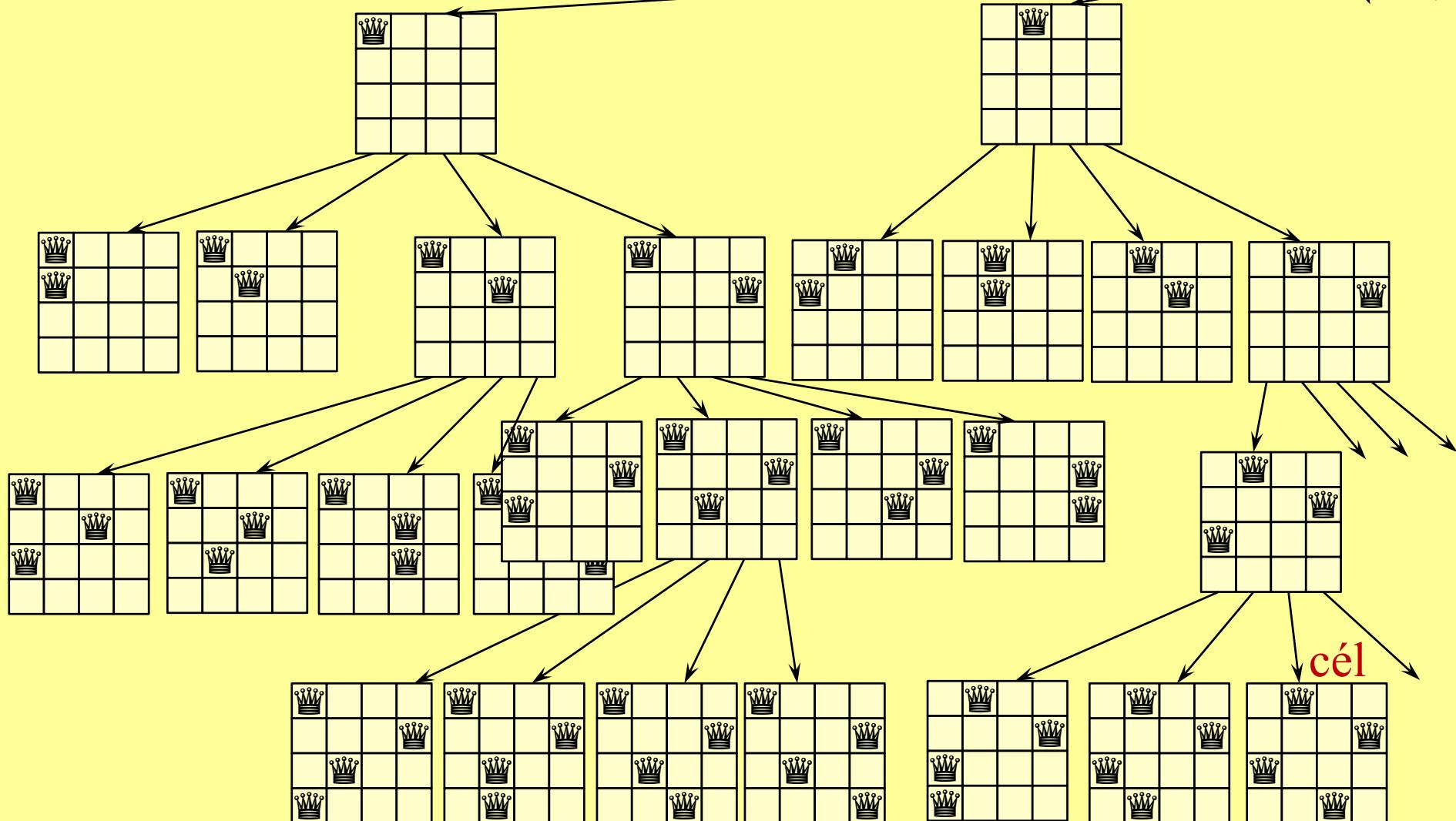
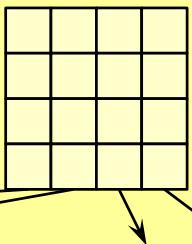
csúcsok száma <  $(n^{n+1}-1)/(n-1)$

csúcs ki-foka:  $n$

ágak száma <  $n^n$

# Állapot-gráf

start



# *Művelet végrehajtásának hatékonysága*

- ❑ A művelet kiszámítási bonyolultsága csökkenthető, ha az állapotokat extra információval egészítjük ki, vagy az invariáns szigorításával szűkítjük az állapotteret.
- ❑ Például
  - Ha egy állapotban a **tábla soron következő üres sorának sorszámát eltároljuk** a tábla mellett, akkor egy újabb királynő felhelyezésekor ezt már nem kell kiszámolni, ugyanakkor könnyen aktualizálhatjuk (eggyel növeljük).
  - **Ne engedjünk meg ütést létrehozni a táblán**, hogy ne kelljen ezt a tulajdonságot külön ellenőrizni. Ennek céljából megjelöljük az ütés alatt álló üres (tehát már nem szabad) mezőket, amelyekre nem helyezhetünk fel királynőt. Egy mező státusza három féle lesz: **szabad**, **ütés alatt álló** vagy **foglalt**, amelyeket a művelet végrehajtásakor kell karbantartani.



# *n-királynő probléma 3.*

kezdőállapot:


*köv\_sor = 1*

közbulső állapot:

○	✗	✗	✗	✗
✗	✗	○	✗	✗
✗	✗	✗	✗	✗
✗		✗	✗	

*köv\_sor = 3*

célállapot :

✗	○	✗	✗	✗
✗	✗	✗	✗	○
○	✗	✗	✗	✗
✗	✗	○	○	✗

*köv\_sor = 5*

Állapottér:  $AT = rec(t : \{ \text{○}, \text{✗} , \_ \}^{n \times n}, köv_sor : \mathbb{N})$

*invariáns:*  $köv_sor \leq n+1$ ,

az első  $köv_sor - 1$  darab sor egy-egy királynőt tartalmaz,  
királynők nem ütik egymást,

*jelölés :*  $\text{✗}$  egy királynő által ütött üres mezőt jelöli,

$\text{—}$  az ütésben nem álló (szabad) üres mezőt jelöli.



## *n-királynő probléma 3. folytatás*

Művelet: új királynő elhelyezése a soron következő sorba

**Helyez(oszlop):**  $AT \rightarrow AT$       (*this:AT*)

HA     $1 \leq \text{oszlop} \leq n$  és *this.köv\_sor*  $\leq n$   
 és *this.t[this.köv\_sor,oszlop]* = \_

AKKOR

*this.t[this.köv\_sor,oszlop]* :=  $\heartsuit$

$\forall i \in [\text{this.kövsor}+1 .. n] :$

*this.t[i, oszlop]* :=  $\times$

ha  $(i \leq n + \text{this.köv_sor} - \text{oszlop})$  akkor *this.t[i, i - this.köv\_sor + oszlop]* :=  $\times$

ha  $(i \leq \text{this.köv_sor} + \text{oszlop} - 1)$  akkor *this.t[i, this.köv\_sor + oszlop - i]* :=  $\times$

*this.köv\_sor* := *this.köv\_sor* + 1

előfeltétel számítás-igénye: konstans

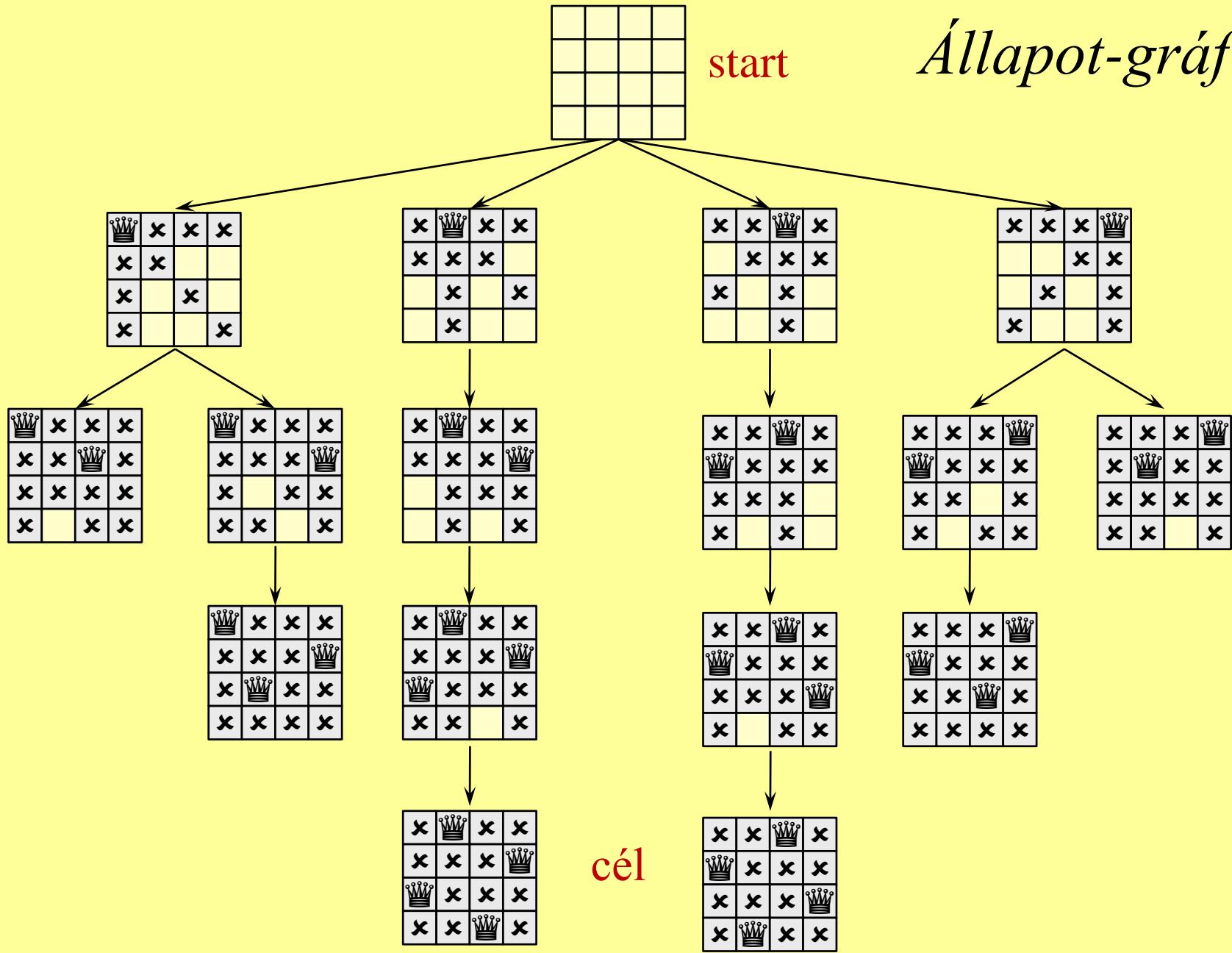
hatás számítás-igénye: lineáris

Kezdőállapot: *this.t* egy üres mátrix, *this.köv\_sor* := 1

célállapot: *this.köv\_sor* > n

célfeltétel nagyon egyszerű lett

# *Allapot-gráf*



# Tologató játék (8-as, 15-ös)

kezdőállapot:  
tetszőleges

2	8	3
1	6	4
7		5



1	2	3
8		4
7	6	5

célállapot:  
szokásos

Állapottér:  $AT = rec(\text{mátrix} : \{0..8\}^{3 \times 3}, \text{üres} : \{1..3\} \times \{1..3\})$

*invariáns:* egy állapot mátrixának sorfolytonos kiterítése a 0 .. 8 számok egy permutációja, az üres hely a 0 elem mátrixbeli sor és oszlopindexe.

Művelet:  $Tol(irány) : AT \rightarrow AT$

HA

$irány \in \{(0,-1), (-1,0), (0,1), (1,0)\}$  és

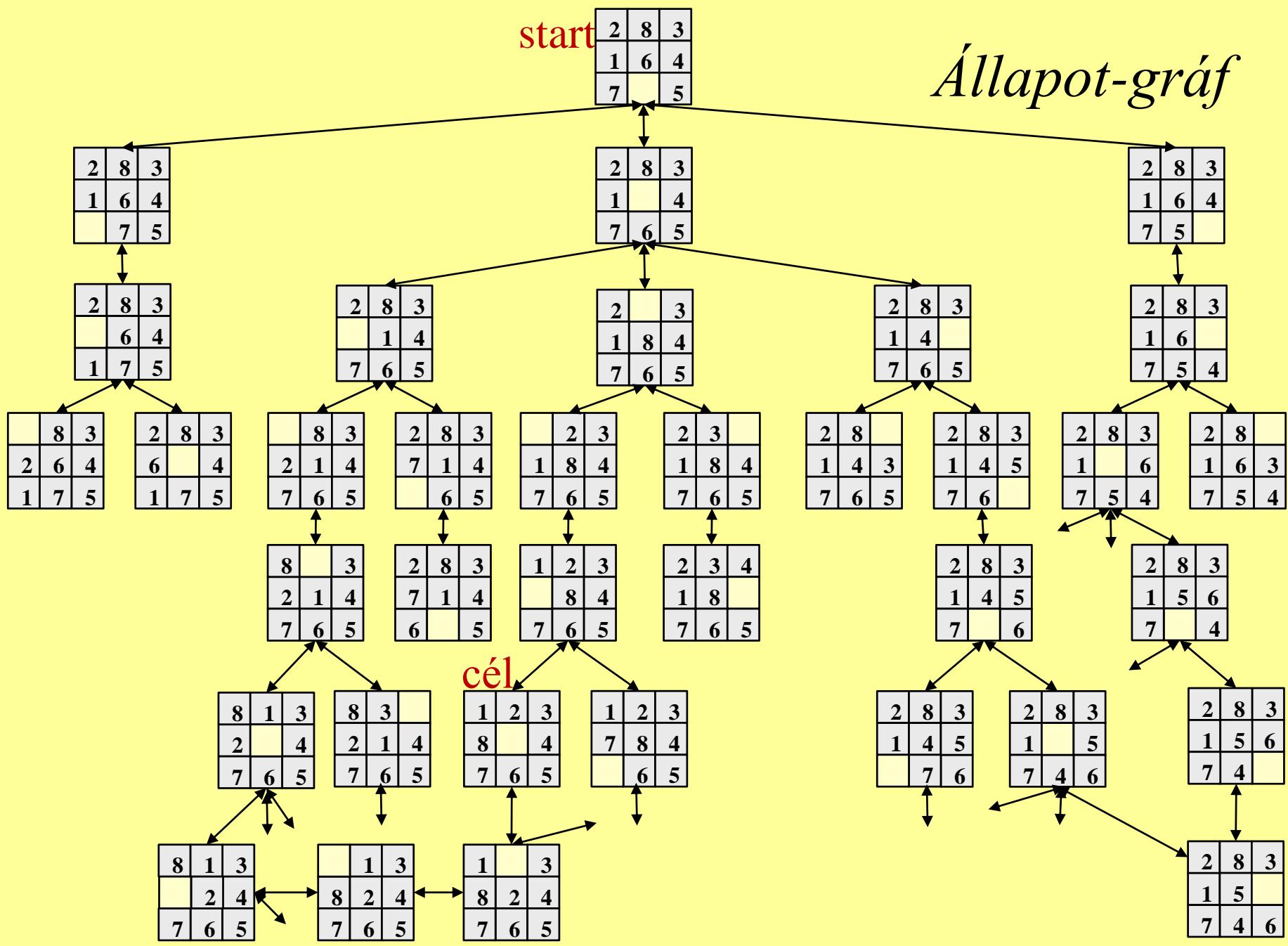
$(1,1) \leq this.\text{üres} + irány \leq (3,3)$  (this: AT)

koordinátánkénti összeadás

AKKOR

$this.\text{mátrix}[this.\text{üres}] \leftrightarrow this.\text{mátrix}[this.\text{üres} + irány]$

$this.\text{üres} := this.\text{üres} + irány$

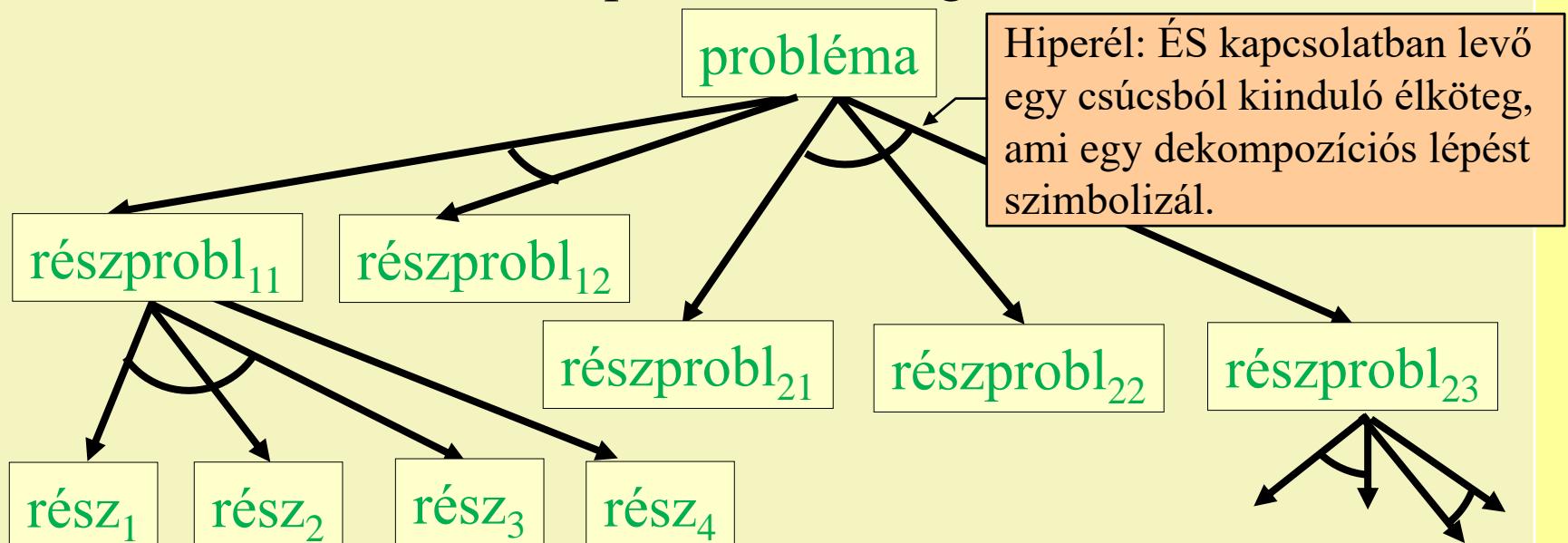


Gregorics Tibor

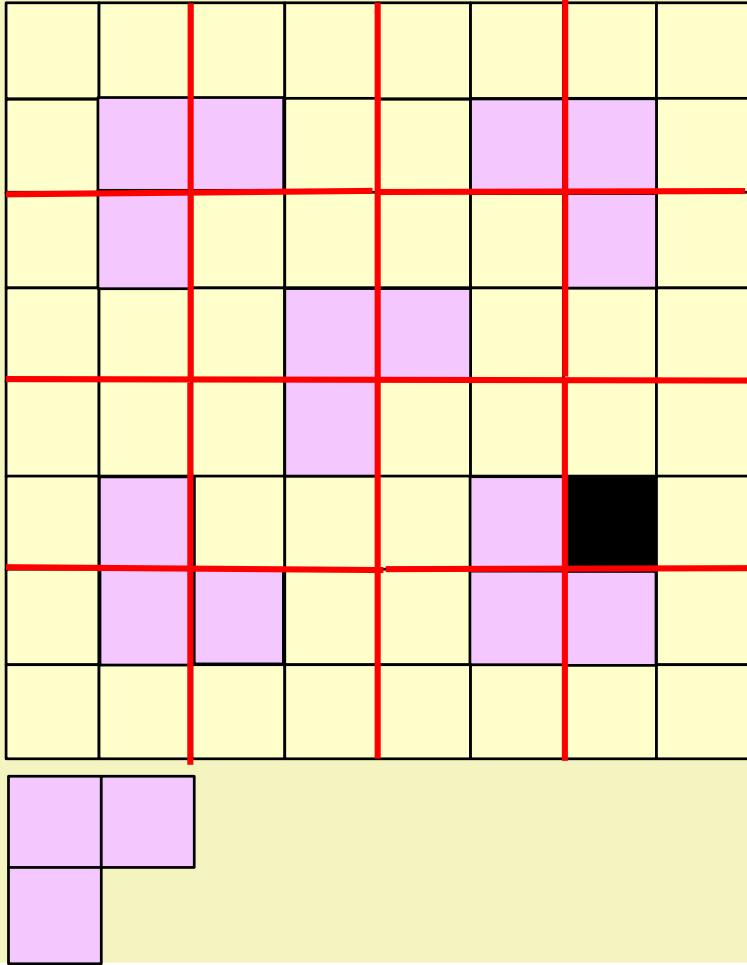
## Mesterséges intelligencia

## 2. Probléma dekompozíció

- ❑ Egy probléma dekomponálása során a problémát részproblémákra bontjuk, majd azokat tovább részletezzük, amíg nyilvánvalóan megoldható problémákat nem kapunk.
- ❑ Sokszor egy probléma megoldását akár többféleképpen is fel lehet bontani részproblémák megoldásaira.

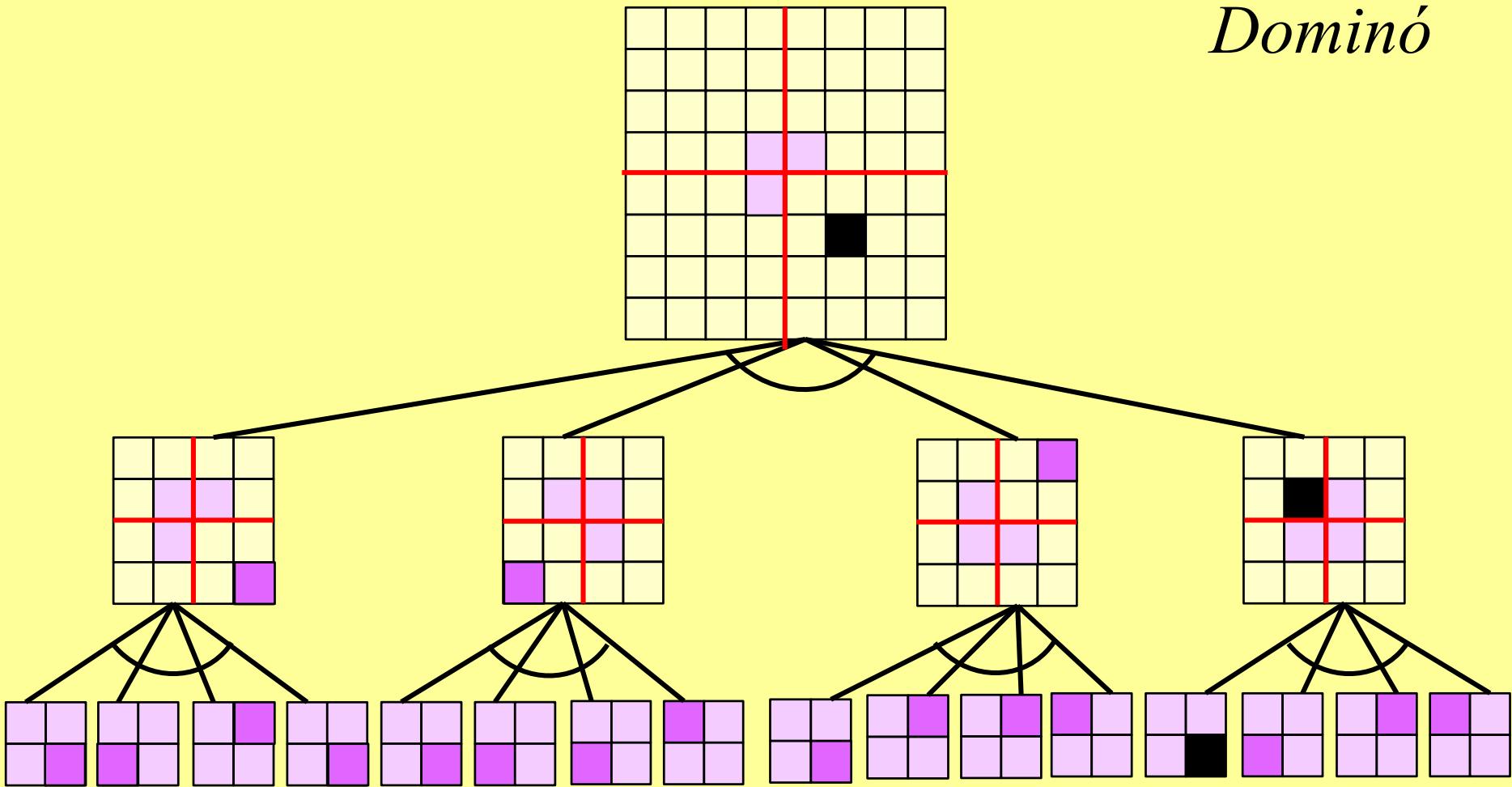


# Dominó



- **Probléma általános leírása:**  
 $2^n \times 2^n$ -es tábla egy foglalt mezővel
- **Kiinduló probléma:**  
 $8 \times 8$ -as tábla egy foglalt mezővel
- **Egyszerű probléma:**  
 $2 \times 2$ -es tábla egy foglalt mezővel
- **Dekomponáló operátor:**  
felosztja a táblát 4 egyenlő részre  
és elhelyez középre egy L alakú  
dominót úgy, hogy az ne fedjen le  
mezőt abban a részben, ahol a  
foglalt mező van

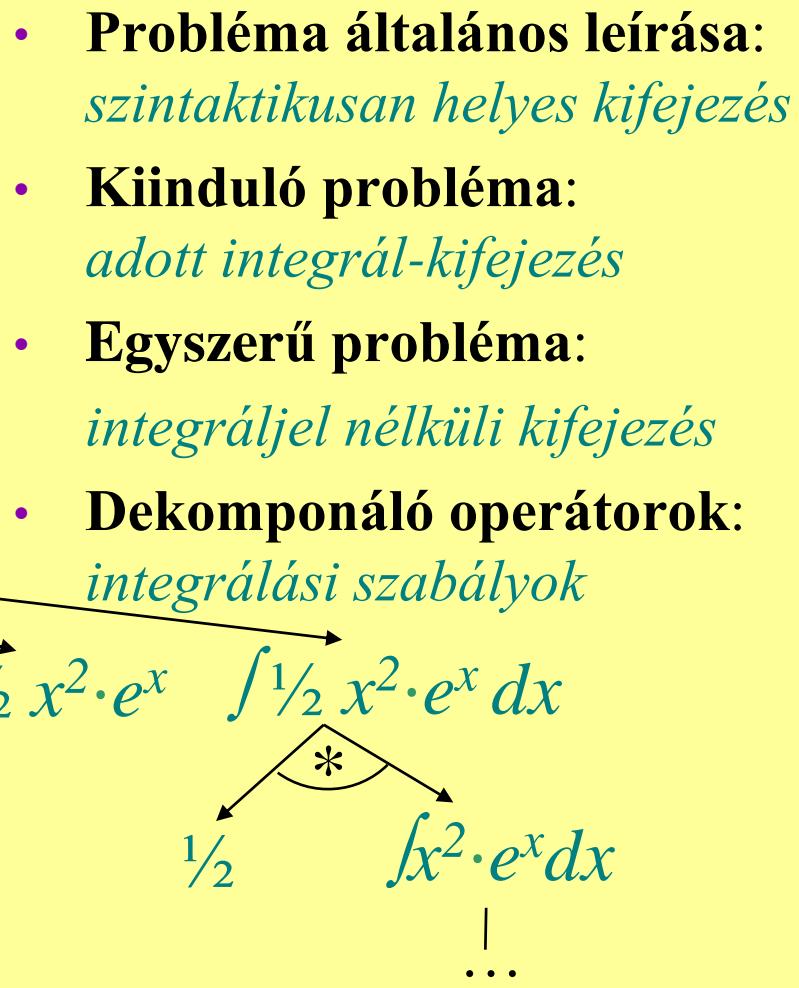
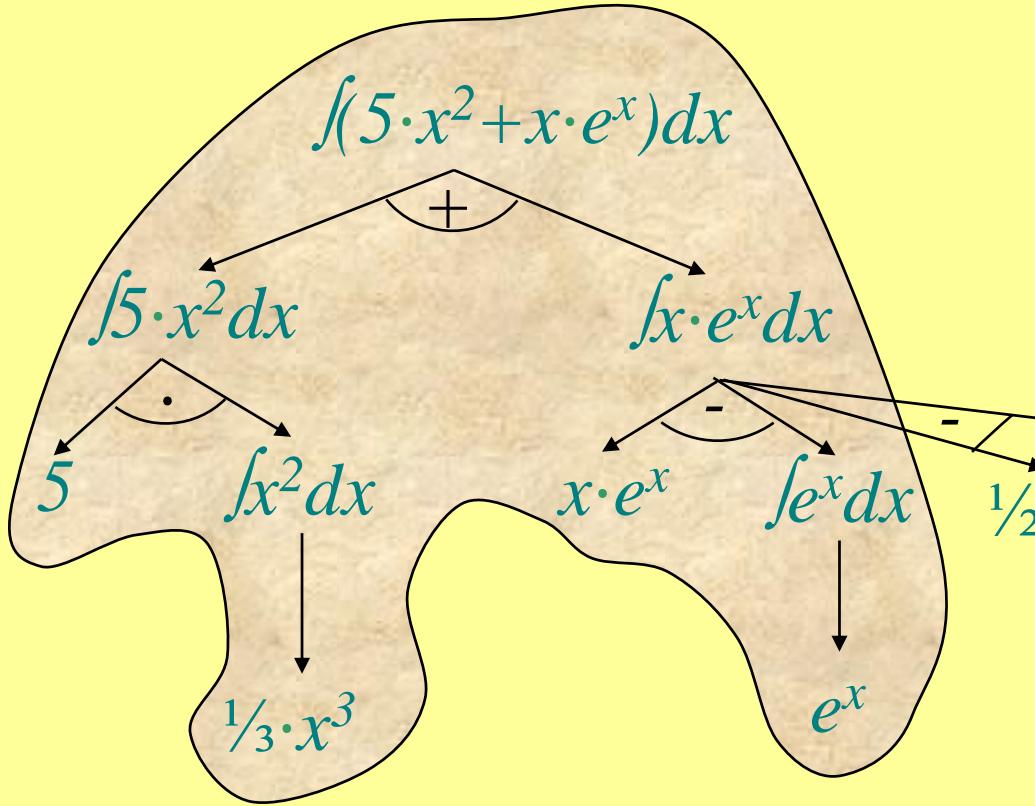
# *Dominó*



**Megoldás gráf:** kiinduló problémát egyszerű problémákra visszavezető dekomponálási folyamatot bemutató fa

**Megoldás:** a részfa elágazásai egy-egy L alakú elem elhelyezését adják.

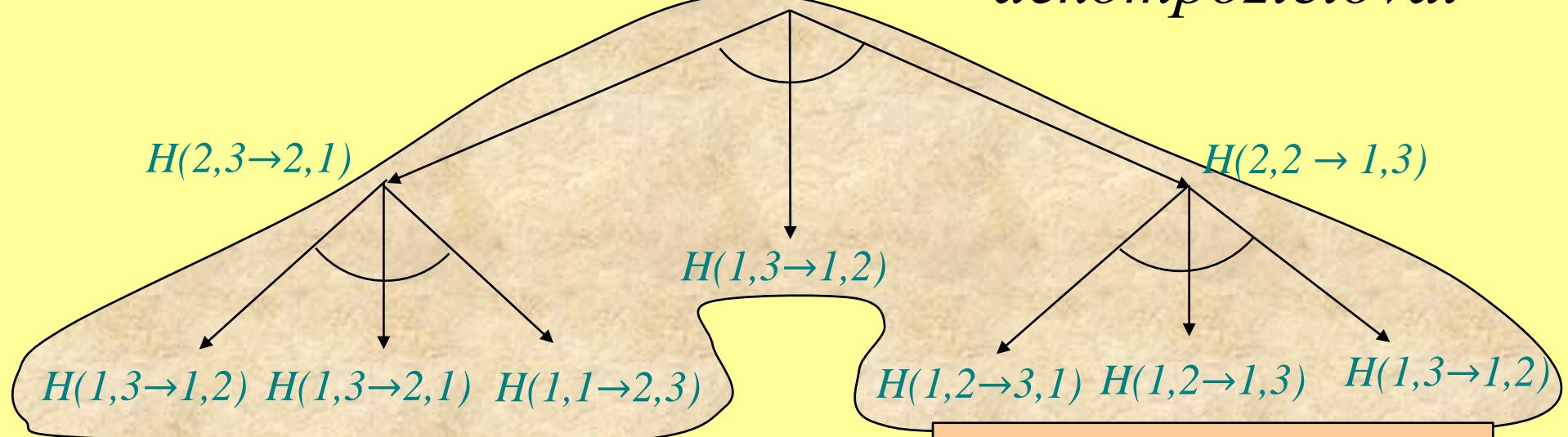
# Integrálszámítás



**Megoldás gráf:** a kiinduló problémát egyszerű problémákra visszavezető alternatívák nélküli levezetés: egy részfa, amelynek gyökere a kiinduló probléma, levelei egyszerű problémák, és belső csúcsaiból egy-egy élköteg indul.

**Megoldás:** a megoldás gráf leveleit balról jobbra haladva kötjük össze a dekomponálások algebrai műveleteivel.

# Hanoi tornyai probléma megoldása dekompozícióval



**Probléma általános leírása:**  $H(n, i \rightarrow j, k)$

$n$  korongot vigyük át az  $i$ . rúdról  
a  $j$ . rúdra a  $k$ . rúd segítségével

**Kiinduló probléma:**  $H(3, 3 \rightarrow 1, 2)$

eldönthető, hogy megoldható-e

**Egyszerű probléma:**  $H(1, i \rightarrow j, k)$

**Dekomponálás:**  $H(n, i \rightarrow j, k) \rightsquigarrow < H(n-1, i \rightarrow k, j), H(1, i \rightarrow j, k), H(n-1, k \rightarrow j, i) >$

**Megoldás gráf:** kiinduló problémát egyszerű problémákra visszavezető fa

**Megoldás:** a részfa leveleit kell balról jobbra haladva összeolvasni

# Dekompozíciós modellezés fogalma

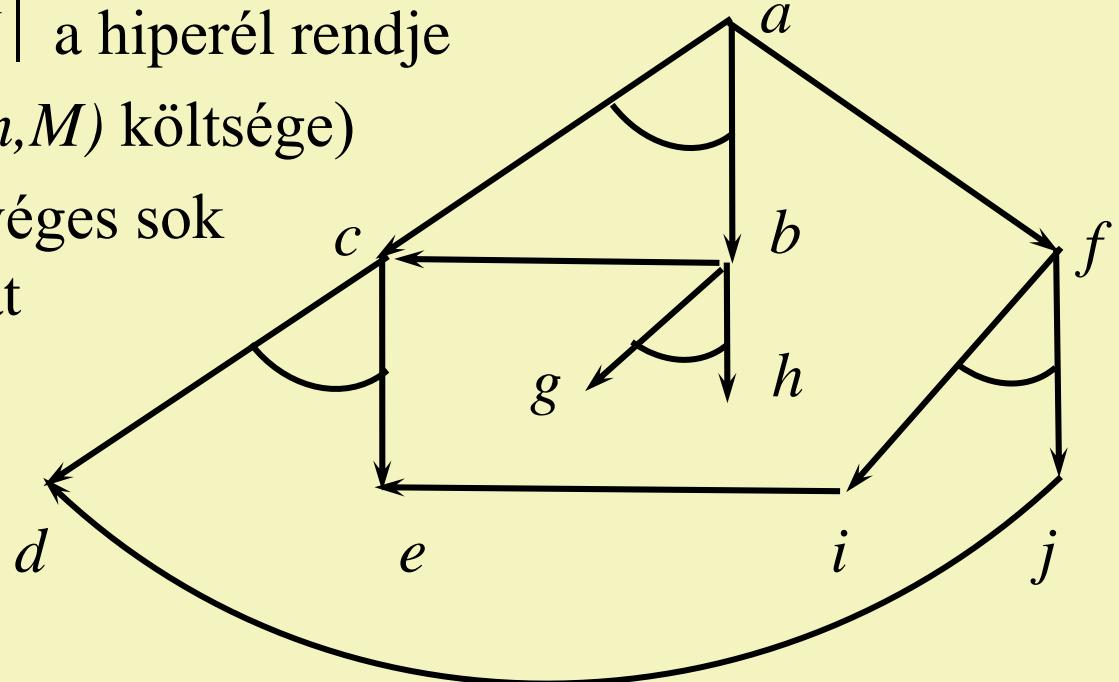
- A modellhez meg kell adnunk:
  - a feladat részproblémáinak általános leírását,
  - a kiinduló problémát,
  - az egyszerű problémákat, amelyekről könnyen eldönthető, hogy megoldhatók-e vagy sem, és
  - a dekomponáló műveleteket:
    - $D: \text{probléma} \rightarrow \text{probléma}^+$  és  
$$D(p) = \langle p_1, \dots, p_n \rangle$$

# *A dekompozíció modellezése ÉS/VAGY gráffal*

- |   |   |
|---|---|
| <input type="checkbox"/> Dekompozíciós modell                                       | <input type="checkbox"/> ÉS/VAGY gráf   |
| ○ részprobléma  | ~ csúcs   |
| ○ dekomponáló művelet<br>hatása egy problémára                                      | ~ irányított hiperél<br>ugyanazon csúcsból induló ÉS<br>kapcsolatban álló élek kötege |
| ○ művelet költsége  | ~ hiperél költsége  |
| ○ kiinduló probléma   | ~ startcsúcs  |
| ○ megoldható probléma   | ~ célcímsúcs  |
| <input type="checkbox"/> Gráf-reprezentáció: ÉS/VAGY gráf, startcsúcs, célcímsúcsok |   |
| ○ dekompozíciós folyam  | ~ hiperút   |
| ○ megoldás  | ~ megoldás-gráf: egy hiperút<br>startcsúcsból célcímsúcsokba                          |

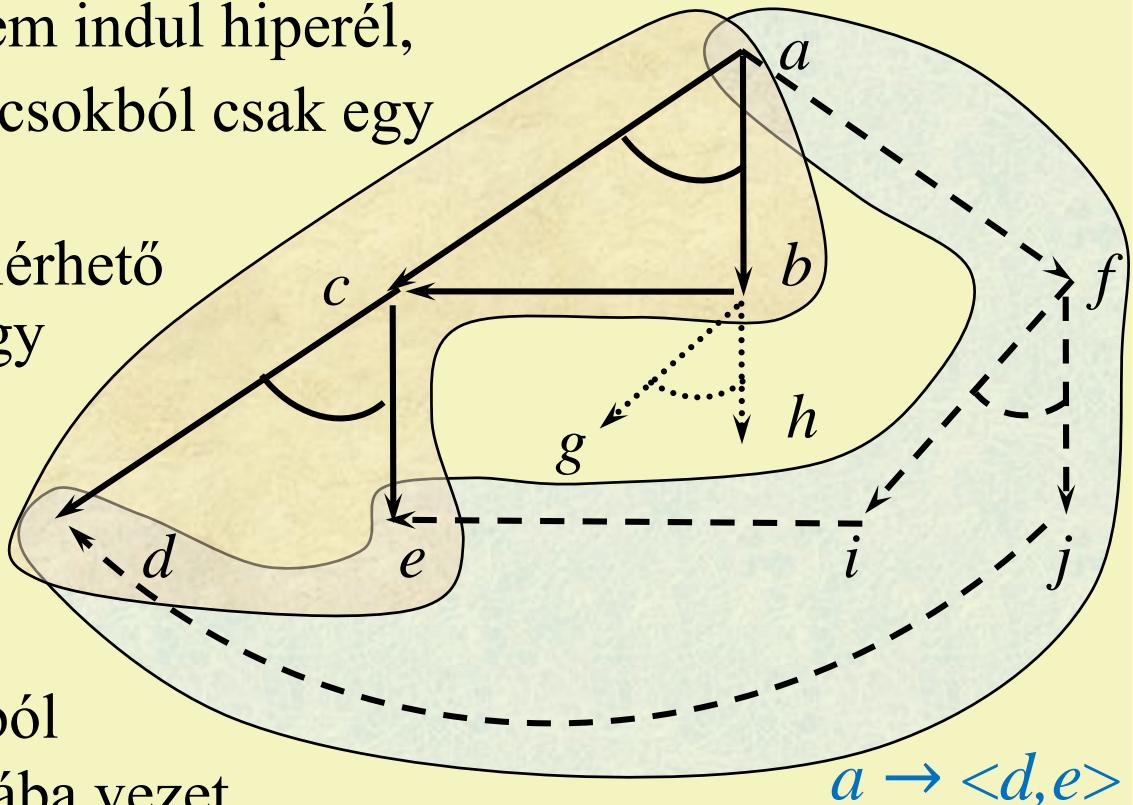
# *ÉS/VAGY gráfok*

- Az  $R=(N,A)$  élsúlyozott irányított hipergráf, ahol az
    - $N$  a csúcsok halmaza,
    - $A \subseteq \{ (n,M) \in N \times N^+ \mid 0 \neq |M| < \infty \}$  a hiperélek halmaza,  $|M|$  a hiperél rendje
    - $(c(n,M))$  az  $(n,M)$  költsége
  - Egy csúcsból véges sok hiperél indulhat
  - $(0 < \delta \leq c(n,M))$



# *Az $n$ csúcsból az $M$ csúcs-sorozatba vezető irányított hiperút fogalma*

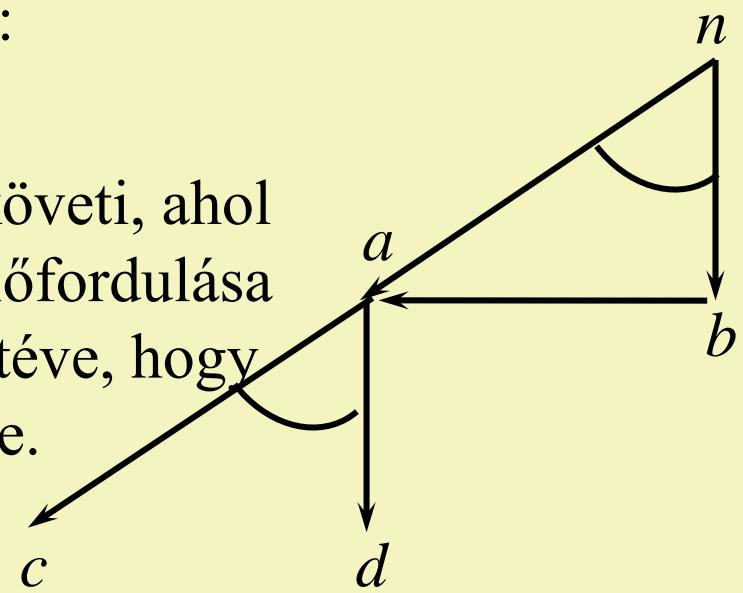
- Egy ÉS/VAGY gráf  $n^{\alpha} \rightarrow M$  hiperútja ( $n \in N$ ,  $M \in N^+$ ) egy olyan véges részgráf, amelyben
  - $M$  csúcsaiból nem indul hiperél,
  - $M$ -en kívüli csúcsokból csak egy hiperél indul,
  - minden csúcs elérhető az  $n$  csúcsból egy közönséges irányított úton.
- A megoldás-gráf egy olyan hiperút, amely a startcsúcsból célcsúcsok sorozatába vezet.



# A hiperút bejárása

- Az  $n \rightarrow M$  hiperút egy bejárásán a hiperút csúcsaiból képzett sorozatoknak a felsorolását értjük:

- az első sorozat:  $\langle n \rangle$
- a  $C$  sorozatot a  $C^{k \leftarrow K}$  sorozat követi, ahol a  $k \in C$  csúcs ( $k \notin M$ ) minden előfordulása helyére a  $K$  sorozatot írjuk feltéve, hogy van a hiperútnak  $(k, K)$  hiperéle.



- Így egy hiperutat közönséges irányított útként foghatunk fel igaz többféleképpen is, mert több bejárása is lehet:

$$\langle n \rangle \rightarrow \langle a, b \rangle \rightarrow \langle a, a \rangle \rightarrow \langle c, d, c, d \rangle$$

$$\langle n \rangle \rightarrow \langle a, b \rangle \rightarrow \langle c, d, b \rangle \rightarrow \langle c, d, a \rangle \rightarrow \langle c, d, c, d \rangle$$

# *Útkeresés ÉS/VAGY gráfban*

- ÉS/VAGY gráfbeli megoldás-gráf keresése **visszavezethető** egy közönséges irányított gráfban történő útkeresésre.
- A startcsúcsból induló hiperutakat (köztük a megoldás-gráfokat is) **a bejárásukkal ábrázolhatjuk**, amelye, mint közönséges irányított utak, egy közönséges irányított gráfot határoznak meg. Ennek
  - csúcsai az eredeti ÉS/VAGY gráf csúcsainak sorozatai,
  - startcsúcsa az ÉS/VAGY gráf startcsúcsából álló egy elemű sorozat,
  - célcímsúcsai az ÉS/VAGY gráf célcímsúcsainak egy részéből álló sorozatok.
- A megfeleltetett közönséges irányított gráf megoldási újai az eredeti ÉS/VAGY gráf megoldás-gráfja.

## 2. Visszalépéses keresés



# *Visszalépéses keresés*

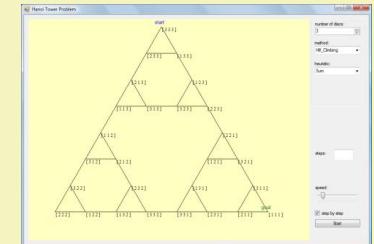
- A visszalépéses keresés egy olyan KR, amely
  - globális munkaterülete:
    - egy **út** a startcsúcsból az aktuális csúcsba (az útról leágazó még ki nem próbált élekkel együtt)
      - kezdetben a startcsúcsot tartalmazó nulla hosszúságú út
      - terminálás célcímsúcs elérésekor vagy a startcsúcsból való visszalépéskor
    - keresés szabályai:
      - a nyilvántartott út végéhez egy új (ki nem próbált) **él hozzáfűzése**, vagy a **legutolsó él törlése** (visszalépés szabálya)
      - vezérlés stratégiája a visszalépés szabályát csak a **legvégső esetben** alkalmazza

# *Visszalépés feltételei*

- ❑ **zsákutca**: az aktuális csúcsból (azaz az aktuális út végpontjából) nem vezet tovább él
- ❑ **zsákutca torkolat**: az aktuális csúcsból kivezető utak nem vezettek célba
- ❑ **kör**: az aktuális csúcs szerepel már korábban is az aktuális úton
- ❑ **méliségi korlát**: az aktuális út hossza elér egy előre megadott értéket

# Alacsonyabb rendű vezérlési stratégiák

- ❑ Az általános vezérlési stratégia kiegészíthető:
  - sorrendi szabály: amely sorrendet egy csúcsból kivezető élek vizsgálatára
  - vágó szabály: megjelöli egy csúcs azon kivezető éleit, amelyeket nem érdemes megvizsgálni
- ❑ Ezek a szabályok lehetnek
  - modellfüggő vezérlési stratégiák (a probléma modelljének sajátosságaiból származó ötlet)
  - heurisztikák (a megoldandó problémától származó információra támaszkodó ötlet)



# Első változat: VL1

- ❑ A visszalépéses algoritmus első változata az, amikor a visszalépés feltételei közül **az első** kettőt építjük be a kereső rendszerbe.
- ❑ Bebizonyítható: *Véges körmentes irányított gráfokon a VL1 minden terminál, és ha létezik megoldás, akkor talál egyet.*
- UI: véges sok adott startból induló út van.
- ❑ Rekurzív algoritmussal (VL1) szokták megadni
  - Indítás: *megoldás := VL1(startcsúcs)*

ADAT := kezdeti érték

**while**  $\neg$ terminálási feltétel(ADAT) **loop**

    SELECT SZ FROM alkalmazható szabályok

    ADAT := SZ(ADAT)

**endloop**

VLI

$A \sim$  élek

$A^* \sim$  véges élsorozat

$N \sim$  csúcsok

**Recursive procedure**  $VLI(akt : N)$  **return** ( $A^*; hiba$ )

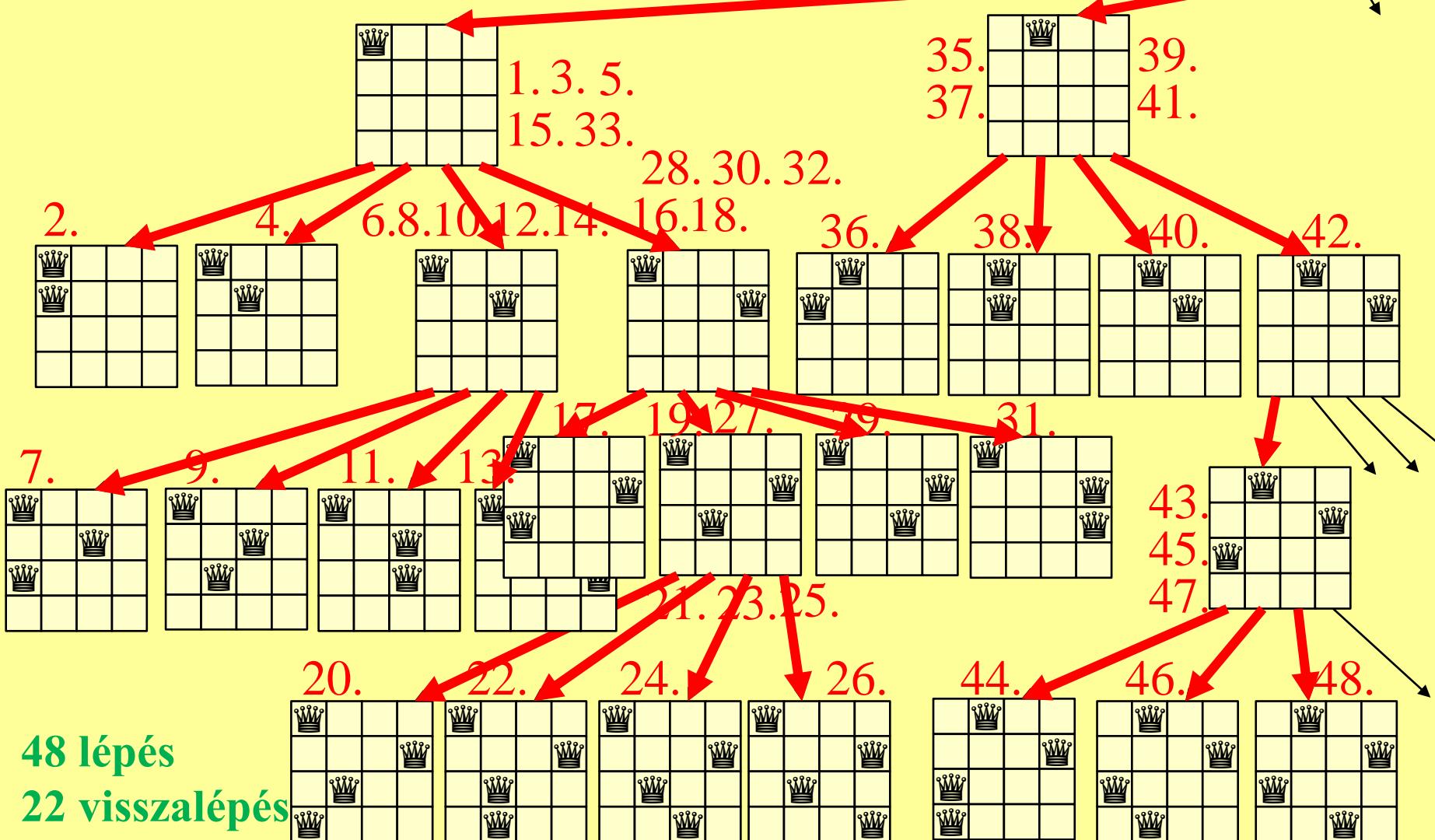
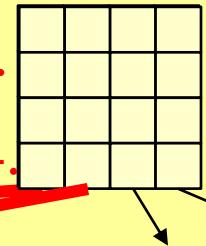
1.     **if**  $cél(akt)$  **then** **return**(nil) **endif**
  2.     **for**  $\forall új \in \Gamma(akt)$  **loop**  $\Gamma(akt) \sim akt$  gyermekei
  3.         megoldás :=  $VLI(új)$
  4.         **if**  $megoldás \neq hiba$  **then**
  5.             **return**( $fűz((akt, új), megoldás)$ ) **endif**
  6.     **endloop**
  7.     **return**( $hiba$ )
- end**

*n*-királynő probléma

2. állapotér modell

sorrendi stratégia: balról jobbra

# Statikus nyomkövetés



48 lépés

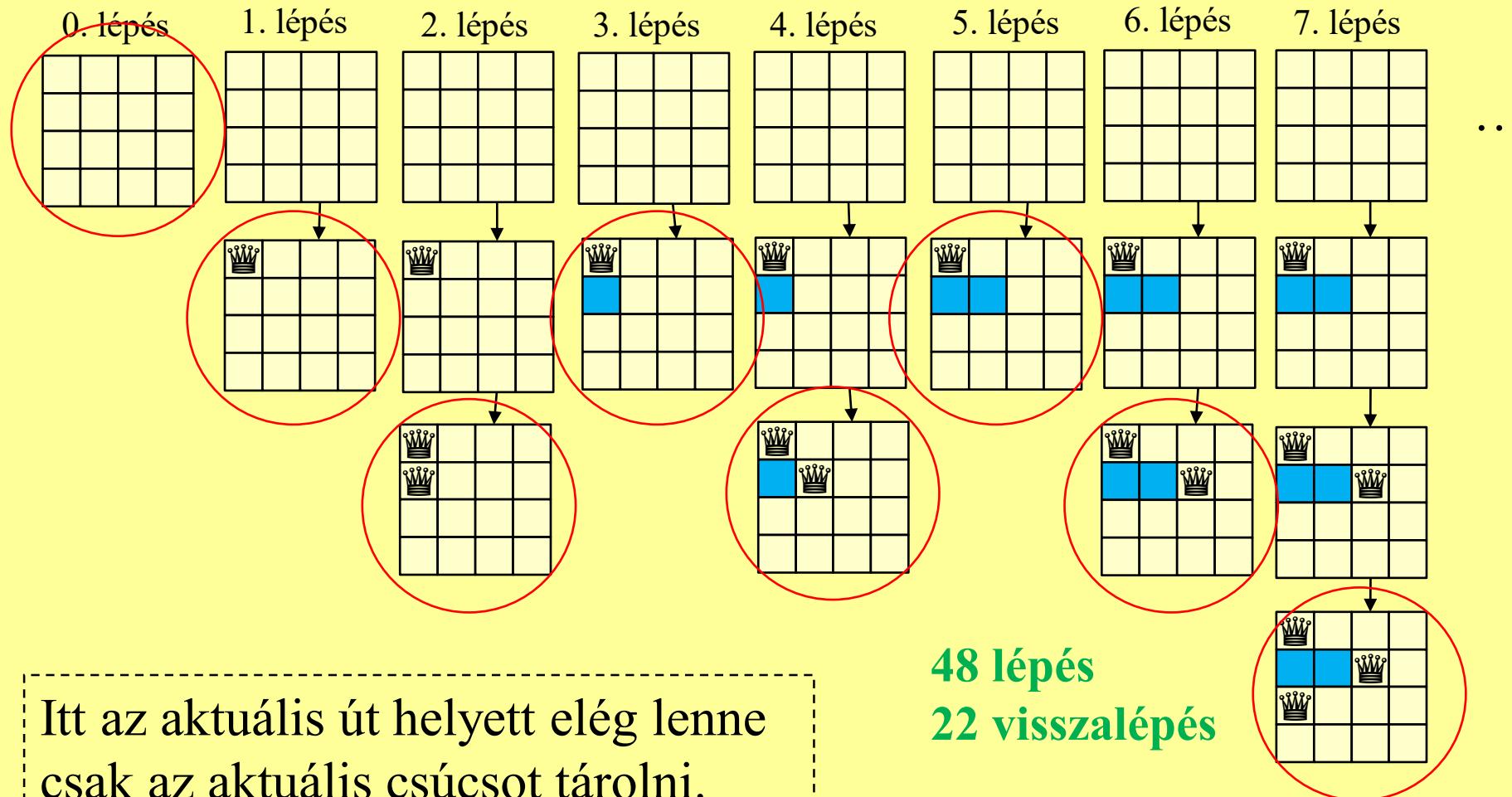
22 visszalépés

# *n*-királynő probléma

## 2. állapotér modell

sorrendi stratégia: balról jobbra

# *Dinamikus nyomkövetés*



# *Sorrendi heurisztikák az $n$ -királynő problémára*

Az  $i$ -edik sor mezőit rangsoroljuk azért, hogy ennek megfelelő sorrendben próbáljuk ki az  $i$ -edik királynő lehetséges elhelyezéseit.

- **Diagonális:** a mezőn áthaladó *hosszabb átló hossza*.
- **Páratlan-páros:** a páratlan sorokban *balról jobbra*, a páros sorokban *jobbról balra* legyen a sorrend.
- **Ütés alá kerülő szabad mezők száma:** új királynő elhelyezésével hány szabad mező kerül ütésbe

4	3	3	4
3	4	4	3
3	4	4	3
4	3	3	4

1	2	3	4
4	3	2	1
1	2	3	4
4	3	2	1

👑	✗	✗	✗
✗	✗	3	2
✗		✗	
✗			✗

# Heurisztikák az $n$ -királynő problémára

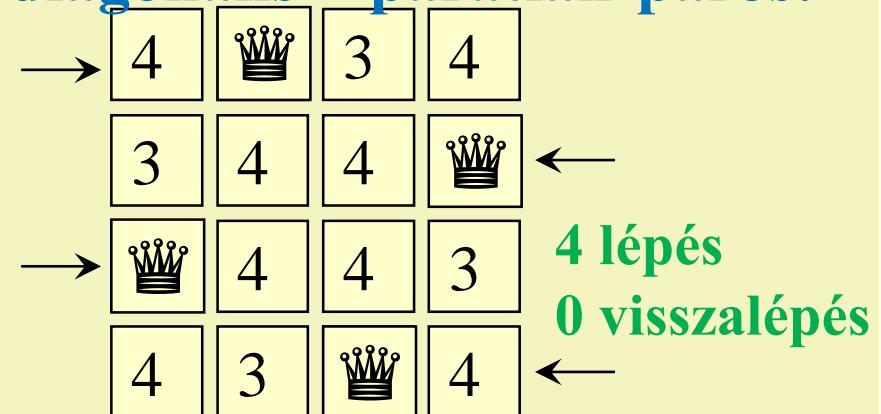
**diagonális + bal-jobb:**

4	4	3	4
	4	4	4
4	4	4	
	4	3	

**8 lépés**

**2 visszalépés**

**diagonális + páratlan-páros:**



**4 lépés**

**0 visszalépés**

2. model	nincs + bal-jobb	diag + bal-jobb
$n = 4$	22/48	2/8
$n = 5$	10/25	10/25
$n = 6$	165/336	63/132
$n = 7$	35/77	80/167
$n = 8$	868/1744	196/400

$n = 4$	nincs + bal-jobb	diag + bal-jobb	diag + ps-ptl
2. model	22/48	2/8	0/4
3. model	4/12	0/4	0/4

## *n*-királynő probléma

### 3. állapotér modell

sorrendi stratégia: balról jobbra

VLI

*heurisztika nélkül*

$$D_i = \{i\text{-dik sor szabad mezői}\}$$

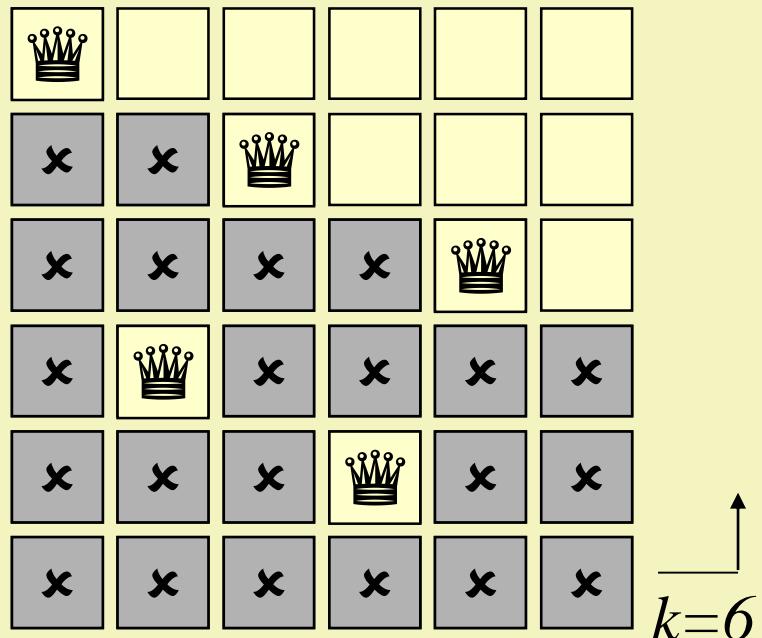
A  $k$ -adik királynő elhelyezése után a hátralevő üres sorokból töröljük az ütésbe került szabad mezőket.

*for*  $i=k+1 .. n$  *loop*

*Töröl*( $i, k$ )

*Töröl*( $i, k$ ) : törli az  $i$ -dik sor azon szabad mezőit, amelyeket a  $k$ -dik királynő üt

VLI: if  $D_k = \emptyset$  then visszalép



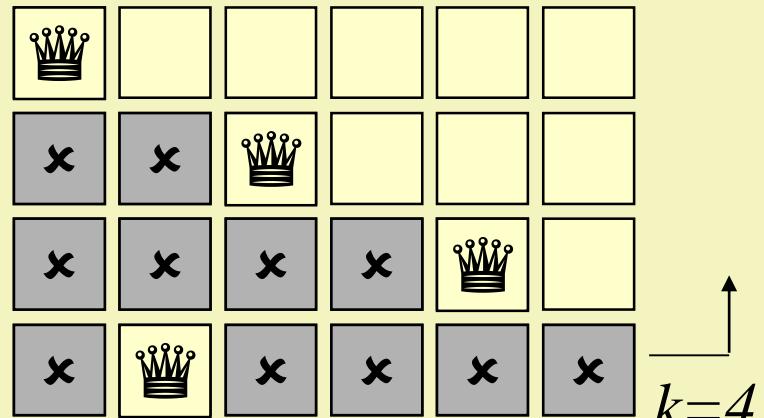
# *Forward Checking*

**FC algoritmus:**

*VLI*

+

**if**  $\exists i \in [k+1.. n]: D_i = \emptyset$   
**then** *visszalép*



$$D_6 = \emptyset$$

# Partial Look Forward

**PLF algoritmus:**

*VL1*

+

**for**  $i=k+1 \dots n$  **loop**

**for**  $j=i+1 \dots n$  **loop** ( $i < j$ )

*Szűr(i,j)*

**if**  $\exists i \in [k+1..n]: D_i = \emptyset$

**then** visszalép

*Szűr(i,j)* : törli az  $i$ -edik sor azon szabad mezőit, amelyekhez nem található a  $j$ -edik sorban vele ütésben nem álló szabad mező

👑						
✗	✗		👑			
✗	✗	✗	✗		👑	
✗	6	✗	✗	✗	✗	✗
✗		✗		✗	✗	✗
✗	✗	✗		✗	✗	✗

$k=3$

$$i = 4, j = 6 \quad D_4 = \emptyset$$

# *Look Forward*

**LF algoritmus:**

*VLI*

+

**for**  $i=k+1 \dots n$  **loop**

**for**  $j=k+1 \dots n$  **and**  $i \neq j$  **loop**

*Szűr(i,j)*

**if**  $\exists i \in [k+1..n]: D_i = \emptyset$

**then** *visszalép*

crown					
x	x	crown			
x	x	x	x		
x		x	x	x	3
x	4	x		x	x
x	4	x	4	5	x

$$i = 4, j = 3 \quad D_6 = \emptyset$$

$$i = 5, j = 4$$

$$i = 6, j = 4$$

$$i = 6, j = 5$$

# *Az $n$ -királynő probléma új reprezentációs modellje*

- Az előző vágási stratégiák alkalmazásánál az  $n$ -királynő problémának egy új modelljére volt szükség:
  - Tekintsük a  $D_1, \dots, D_n$  halmazokat, ahol  $D_i = \{1 \dots n\}$  (ezek az  $i$ -dik sor szabad mezői).
  - Keressük azt az  $(x_1, \dots, x_n) \in D_1 \times \dots \times D_n$  elhelyezést ( $x_i$  az  $i$ -dik sorban elhelyezett királynő oszloppozíciója),
  - amely nem tartalmaz ütést: minden  $i, j$  királynő párra:  
 $C_{ij}(x_i, x_j) \equiv (x_i \neq x_j \wedge |x_i - x_j| \neq |i - j|)$ .
- A visszalépéses keresés e modell változóinak értékét keresi, miközben az alkalmazott vágó stratégiák ezen változók lehetséges értékeit adó  $D_i$  halmazokat szűkítik.

# Bináris korlát-kielégítési modell

- Keressük azt az  $(x_1, \dots, x_n) \in D_1 \times \dots \times D_n$   $n$ -est ( $D_i$  véges) amely kielégít néhány  $C_{ij} \subseteq D_i \times D_j$  bináris korlátot.
- Példák:
  1. Házasságközvetítő probléma ( $n$  férfi,  $m$  nő; keressünk minden férfinak neki szimpatikus feleségjelöltet):
    - Az  $i$ -dik férfi ( $i=1..n$ ) felesége ( $x_i$ ) a  $D_i = \{1, \dots, m\}$  azon elemei, amelyekre fenn áll, hogy  $szimpatikus(i, x_i)$ .
    - Az összes  $(i,j)$ -re:  $C_{ij}(x_i, x_j) \equiv (x_i \neq x_j)$  (azaz nincs bigámia)
  2. Gráfszínezési probléma (egy véges egyszerű irányítatlan gráf  $n$  darab csúcsát kell kiszínezni  $m$  színnel úgy, hogy a szomszédos csúcsok eltérő színűek legyenek):
    - Az  $i$ -dik csúcs ( $i=1..n$ ) színe ( $x_i$ ) a  $D_i = \{1, \dots, m\}$  elemei.
    - minden  $i, j$  szomszédos csúcs párra:  $C_{ij}(x_i, x_j) \equiv (x_i \neq x_j)$ .

# *Modellfüggő vezérlési stratégia*

- A bemutatott vágó stratégiákat a modell bináris korlátaival definiálhatjuk, de ehhez a korlátok jelentését nem kell ismerni:

*Töröl(i,k):*  $D_i := D_i - \{e \in D_i \mid \neg C_{ik}(e, x_k)\}$

*Szűr(i,j) :*  $D_i := D_i - \{e \in D_i \mid \forall f \in D_j : \neg C_{ij}(e, f)\}$

- Ezekben a módszerekben tehát nem heurisztikák, hanem **modellfüggő vágó stratégiák** jelennek meg.
- **Modellfüggő sorrendi stratégiák** is konstruálhatók:
  - Mindig a legkisebb tartományú még kitöltetlen komponensnek válasszunk előbb értéket.
  - Ugyanazon korláthoz tartozó komponenseket lehetőleg közvetlenül egymás után töltük ki.

# Második változat: VL2

- A visszalépéses algoritmus második változata az, amikor a visszalépés feltételei közül mindenet beépítjük a kereső rendszerbe.
- Bebizonyítható: *A VL2 δ-gráfban minden terminál. Ha létezik a mélységi korlátnál nem hosszabb megoldás, akkor megtalál egy megoldást.*  
UI: véges sok adott korlátnál rövidebb startból induló út van.
- Rekurzív algoritmussal (VL2) adjuk meg
  - Indítás: *megoldás := VL2(<startcsúcs>)*

ADAT := kezdeti érték

**while**  $\neg$ terminálási feltétel(ADAT) **loop**

    SELECT SZ FROM alkalmazható szabályok

    ADAT := SZ(ADAT)

**endloop**

VL2

**Recursive procedure** VL2( $út : N^*$ ) **return** ( $A^*; hiba$ )

1.      $akt := \text{utolsó\_csúcs}(út)$
  2.     **if**  $cél(akt)$  **then** **return**(nil) **endif**
  3.     **if**  $hossza(út) \geq \text{korlát}$  **then** **return**(hiba) **endif**
  4.     **if**  $akt \in \text{maradék}(út)$  **then** **return**(hiba) **endif**
  5.     **for**  $\forall új \in \Gamma(akt) - \pi(akt)$  **loop**      $\Gamma(akt) \sim akt$  gyermekei  
 $\pi(akt) \sim akt$  egy szülője
  6.          $megoldás := VL2(fűz(út, új))$
  7.         **if**  $megoldás \neq hiba$  **then**
  8.             **return**( $fűz((akt, új), megoldás)$ ) **endif**
  9.     **endloop**
  10.    **return**(hiba)
- end**

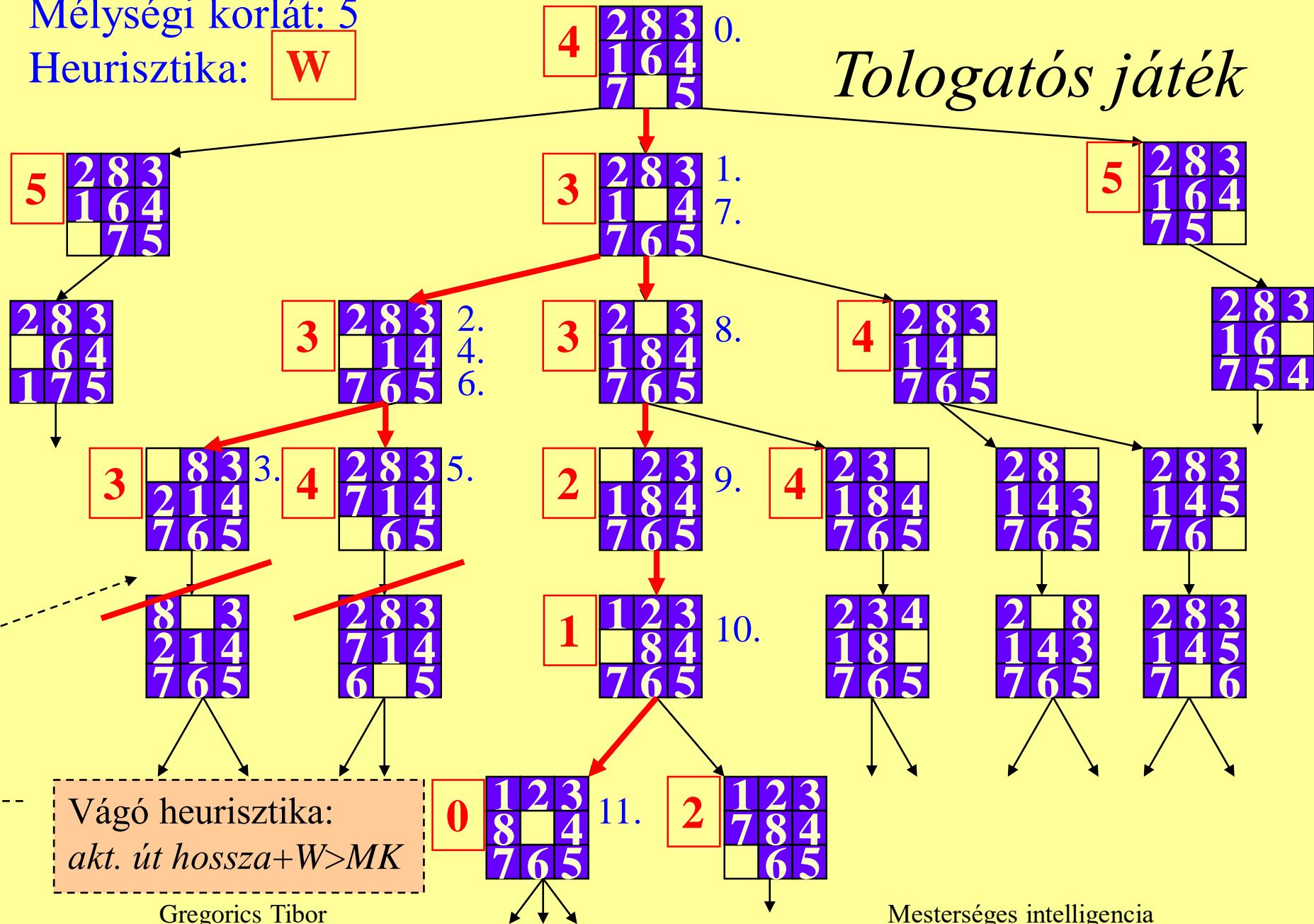
# *Mélységi korlát szerepe*

- A mélységi korlát önmagában is biztosítja a terminálást körök esetén is.
  - Ilyenkor nem kell a rekurzív hívásnál a teljes aktuális utat átadni : elég az út hosszát, az aktuális csúcsot és annak szülőjét (a kettő hosszú körök kiszűréséhez).
  - Ez az egyszerűsítés a hatékonyságon javíthat, de ha a reprezentációs gráfban vannak rövid körök is, akkor futási idő szempontjából ez nem előnyös.
- A VL2 nem talál megoldást, ha a megoldási utak a megadott mélységi korlátnál hosszabbak. (A keresés ilyenkor sikertelenül terminál.)

Mélységi korlát: 5

Heurisztika: W

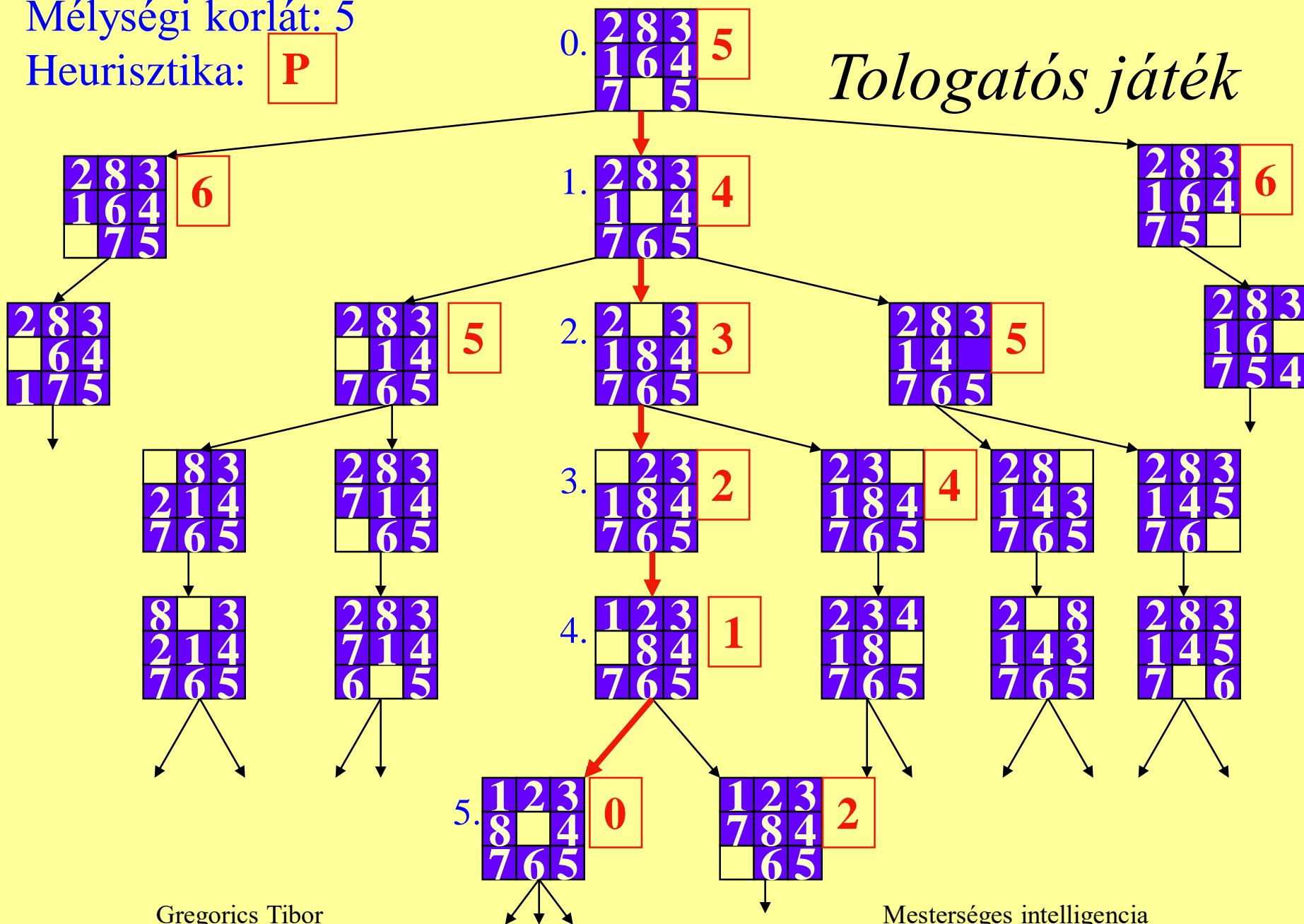
## Tologatós játék



Mélységi korlát: 5

Heurisztika: P

## Tologatós játék



# Értékelés

## □ ELŐNYÖK

- minden terminál, talál megoldást (a mélységi korláton belül)
- könnyen implementálható
- kicsi memória igény

## □ HÁTRÁNYOK

- nem ad optimális megoldást. (iterációba szervezhető)
- kezdetben hozott rossz döntést csak sok visszalépés korrigál (visszaugrásos keresés)
- egy zsákutca részt többször is bejárhat a keresés

# 3. Gráfkeresés

- ❑ A gráfkeresés olyan KR, amelynek
  - globális munkaterülete: startcsúsból kiinduló már feltárt útjai a reprezentációs gráfnak (keresőgráf), valamint a feltárt utak végei (nyílt csúcsok)
    - kiinduló értéke: a startcsúcs,
    - terminálási feltétel: vagy célcsúcsot terjeszt ki vagy nincs nyílt csúcs.
  - keresési szabálya: egy nyílt csúcs kiterjesztése
  - vezérlési stratégiája: a legkedvezőbb csúcs kiterjesztésére törekszik, és ehhez egy kiértékelő függvényt használ.

## 3.1. Általános gráfkereső algoritmus

Jelölések:

- keresőgráf ( $G$ ) : a reprezentációs gráf eddig felfedezett és egyben el is tárolt része
- nyílt csúcsok halmaza ( $OPEN$ ) : kiterjesztésre várakozó csúcsok, amelyeknek gyerekeit még nem vagy nem eléggyé jól ismerjük
- kiértékelő függvény ( $f: OPEN \rightarrow \mathbb{R}$ ) : kiválasztja a megfelelő nyílt csúcsot kiterjesztésre

# Gráfkeresés függvényei

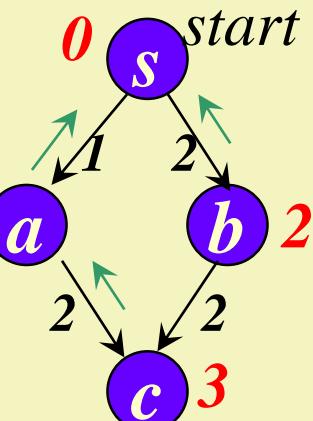
## □ $\pi: N \rightarrow N$ szülőre visszamutató pointer

- $\pi(m) = m$  csúcs már ismert szülője,  $\pi(start) = nil$ 
  - $\pi$  egy *start* gyökerű irányított feszítőfát jelöl ki  $G$ -n és segít kiolvasni a megoldási utat terminálás után.
  - Jó lenne ha egy  $m$  csúcselfedezésekor a  $\pi(m)$  a  $G$ -beli optimális  $start \rightarrow m$  utat mutatná.

## □ $g: N \rightarrow \mathbb{R}$ költségfüggvény

- $g(m) = c^\alpha(start, m)$  – egy már megtalált  $\alpha \in \{start \rightarrow n\}$  út költsége
  - Jó lenne ha egy  $m$  csúcselfedezésekor a  $g(m)$  a  $\pi$  által mutatott  $start \rightarrow m$  út költségét adná.

$m \in G$  csúcs **korrekt**, ha  $g(m)$  és  $\pi(m)$  **konzisztenz**:  $g(m) = c^\pi(start, m)$ ,  
és  $\pi(m)$  **optimális**:  $c^\pi(start, m) = \min_{\alpha \in \{start \rightarrow m\} \cap G} c^\alpha(start, m)$   
 $G$  korrekt, ha minden csúcsa korrekt.

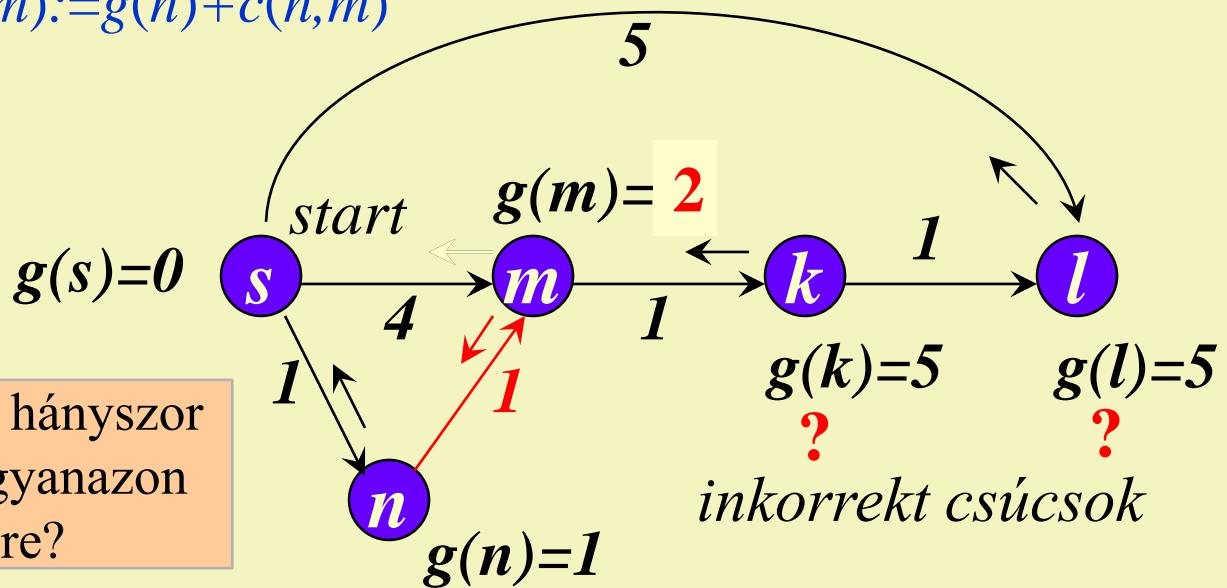


# *A korrektség fenntartása egy csúcs előállításakor*

- Kezdetben:  $\pi(start) := nil$ ,  $g(start) := 0$
- Az  $n$  csúcs kiterjesztése után minden  $m \in \Gamma(n)$  csúcsra
  - 1. Ha  $m$  új csúcs
    - azaz  $m \notin G$  akkor
$$\pi(m) := n, g(m) := g(n) + c(n, m)$$
$$OPEN := OPEN \cup \{m\}$$
    - 2. Ha  $m$  régi csúcs, amelyhez olcsóbb utat találtunk
      - azaz  $m \in G$  és  $g(n) + c(n, m) < g(m)$  akkor
$$\pi(m) := n, g(m) := g(n) + c(n, m)$$
      - 3. Ha  $m$  régi csúcs, amelyhez nem találtunk olcsóbb utat
        - azaz  $m \in G$  és  $g(n) + c(n, m) \geq g(m)$  akkor *SKIP*

# Mégsem marad korrekt a kereső gráf

Ha  $m \in G$  és  $g(n) + c(n, m) < g(m)$ , akkor  
 $\pi(m) := n$ ,  $g(m) := g(n) + c(n, m)$



- ❑ Mi legyen az olcsóbb úton újra megtalált  $m$  csúcs leszármazottaival?
  1. Járjuk be és javítsuk ki a pointereiket és költségeiket!
  2. Kerüljük el egy jó kiértékelő függvénytel, hogy ilyen történjen!
  3. Az  $m$  csúcsot helyezzük vissza OPEN halmazba!

ADAT := kezdeti érték

**while**  $\neg$  terminálási feltétel(ADAT) **loop**

    SELECT SZ FROM alkalmazható szabályok

    ADAT := SZ(ADAT)

**endloop**

## Általános gráfkereső algoritmus

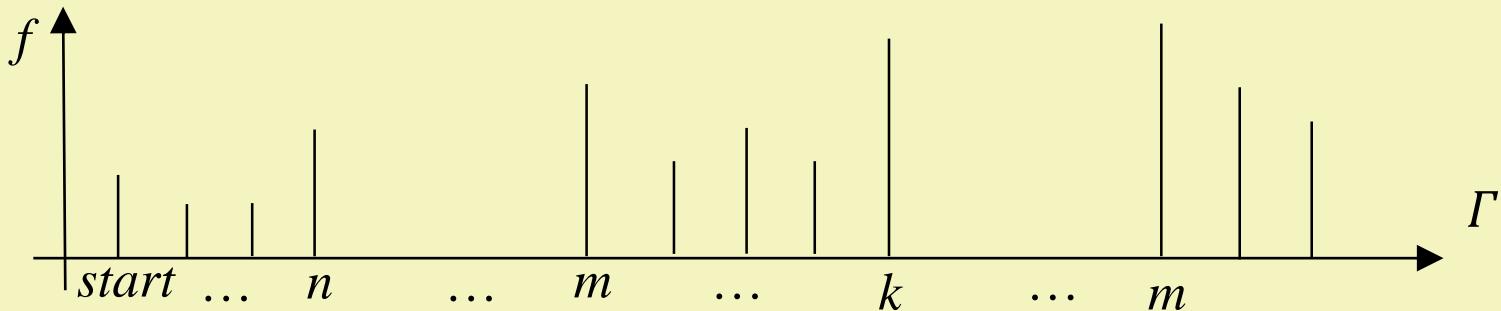
1.  $G := (\{start\}, \emptyset); OPEN := \{start\}; g(start) := 0; \pi(start) := nil$
2. **loop**
3.   **if** empty( $OPEN$ ) **then return** nincs megoldás
4.    $n := \min_f(OPEN)$
5.   **if** cél( $n$ ) **then return** megoldás
6.    $OPEN := OPEN - \{n\}$
7.   **for**  $\forall m \in \Gamma(n) - \pi(n)$  **loop**    $\Gamma(akt) \sim$  akt gyermekei  
 $\pi(akt) \sim$  akt egy szülője
8.     **if**  $m \notin G$  or  $g(n) + c(n,m) < g(m)$  **then**
9.        $\pi(m) := n; g(m) := g(n) + c(n,m); OPEN := OPEN \cup \{m\}$
10.   **endloop**
11.    $G := G \cup \{(n,m) \in A \mid m \in \Gamma(n) - \pi(n)\}$
12. **endloop**

# *Működés és eredmény*

Bebizonyítható:

- A  $GK$   $\delta$ -gráfban a működése során egy csúcsot legfeljebb véges sokszor terjeszt ki.  
(ebből következik például, hogy körökre nem érzékeny)
- A  $GK$  véges  $\delta$ -gráfban mindenkor terminál.
- Ha egy véges  $\delta$ -gráfban létezik megoldás, akkor a  $GK$  megoldás megtalálásával terminál.

# Gráfkeresés működési grafikonja

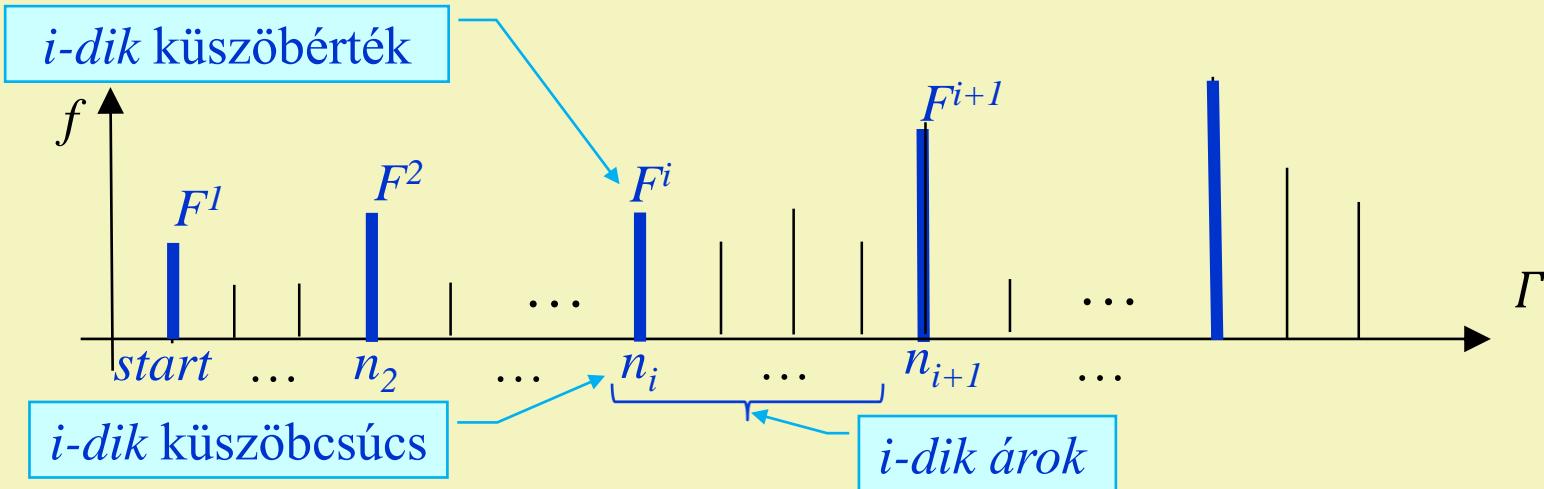


- Soroljuk fel a kiterjesztett csúcsokat kiterjesztésük sorrendjében (ugyanaz a csúcs többször is szerepelhet, hiszen többször is kiterjesztődhet) a kiterjesztésükkel mért  $f$  kiértékelő függvényértékükkel.

# *Csökkenő kiértékelő függvény*

- Egy  $GK$  kiértékelő függvénye **csökkenő**, amennyiben a egy csúcsra adott értéke az algoritmus működése során nem növekszik, viszont mindenkor mindenkor csökken, valahányszor a csúcshoz a korábbinál olcsóbb utat találunk.
  - Például a  $g$  költségfüggvény ilyen.
- Csökkenő kiértékelő függvény mellett a  $GK$ 
  - soha nem terjeszt ki inkorrekt csúcsot
  - időről időre automatikusan helyreállítja a kereső gráf korrektségét, azaz a  $\pi$  feszítő fájának optimálisságát és konzisztenciáját.

# Mikor lesz a kereső gráf korrekt csökkenő kiértékelő függvény mellett?



- Válasszuk ki az értékekből azt az  $F^i$  ( $i=1,2,\dots$ ) monoton növekedő részsorozatot, amely a legelső értékkel kezdődik, majd mindenkorábbi nem kisebb értékkel folytatódik.
- Csökkenő kiértékelő függvény használata mellett a  $GK$ 
  - kereső gráfja korrekt lesz valahányszor küszöbcsúcsot terjeszt ki
  - soha nem terjeszt ki inkorrekt csúcsot

## 3.2. Nevezetes gráfkereső algoritmusok

- Most az  $f$  kiértékelő függvény megválasztása következik.

Nem-informált

- mélységi (MGK)
- szélességi (SZGK)
- egyenletes (EGK)

- Az úgynevezett tie-breaking rule-ok (egyenlőséget feloldó szabályok) a nem-informált gráfkeresések nélkül is tartalmazhatnak heurisztikát.

Heurisztikus

- előre tekintő (mohó, best-first)
- A, A\*, A<sup>c</sup>
- B, B', A<sup>\*\*</sup>

# Nevezetes nem-informált algoritmusok

ugyanúgy mélységi stratégiát használ,  
mint a visszalépéses keresés

Algoritmus	Definíció	Eredmények
Mélységi gráfkeresés MGK	$f = -g$ , $c(n,m) = 1$	<ul style="list-style-type: none"><li>végtelen gráfokban csak mélységi korláttal garantál megoldást</li></ul>
Szélességi gráfkeresés SZGK	$f = g$ , $c(n,m) = 1$	<ul style="list-style-type: none"><li>optimális (legrövidebb) megoldást ad, ha van (még végtelen <math>\delta</math>-gráfban is)</li><li>egy csúcs kiterjesztésekor ismeri az odavezető legrövidebb utat (legfeljebb egyszer terjeszti ki)</li></ul>
Egyenletes gráfkeresés EGK	$f = g$	<ul style="list-style-type: none"><li>optimális (legolcsóbb) megoldást ad, ha van (még végtelen <math>\delta</math>-gráfban is)</li><li>egy csúcs kiterjesztésekor ismeri az odavezető legolcsóbb utat (legfeljebb egyszer terjeszt ki)</li></ul>

# Heurisztika a gráfkeresésekben

- Heurisztikus függvénynek nevezzük azt a  $h:N \rightarrow \mathbb{R}$  függvényt, amelyik egy csúcsnál megbecsüli a csúcsból a célba vezető („hátralévő”) optimális út költségét.

- $h(n) \approx h^*(n)$  ( $h^*:N \rightarrow \mathbb{R}$  többnyire nemismert, csak elméletben létező költségfüggvény)



- Példák:

- 8-kirakó :  $W, P$
- 0 (zéró függvény)?

hátralevő optimális költség  $n$ -ből a célcsúcsok ( $T$ ) valamelyikébe:

$$h^*(n) = c^*(n, T)$$

$M$ -be vezető optimális költség:

$$c^*(n, M) := \min_{m \in M} c^*(n, m)$$

$n$ -ből  $m$ -be vezető optimális költség:

$$c^*(n, m) := \min_{\alpha \in \{n \rightarrow m\}} c^\alpha(n, m)$$

# *Heurisztikus függvények tulajdonságai*

## □ Nevezetes tulajdonságok:

- **Nem-negatív:**  $h(n) \geq 0 \quad \forall n \in N$
- **Megengedhető** (admissible):  $h(n) \leq h^*(n) \quad \forall n \in N$
- **Monoton megszorítás:**  $h(n) - h(m) \leq c(n,m) \quad \forall (n,m) \in A$   
(következetes)

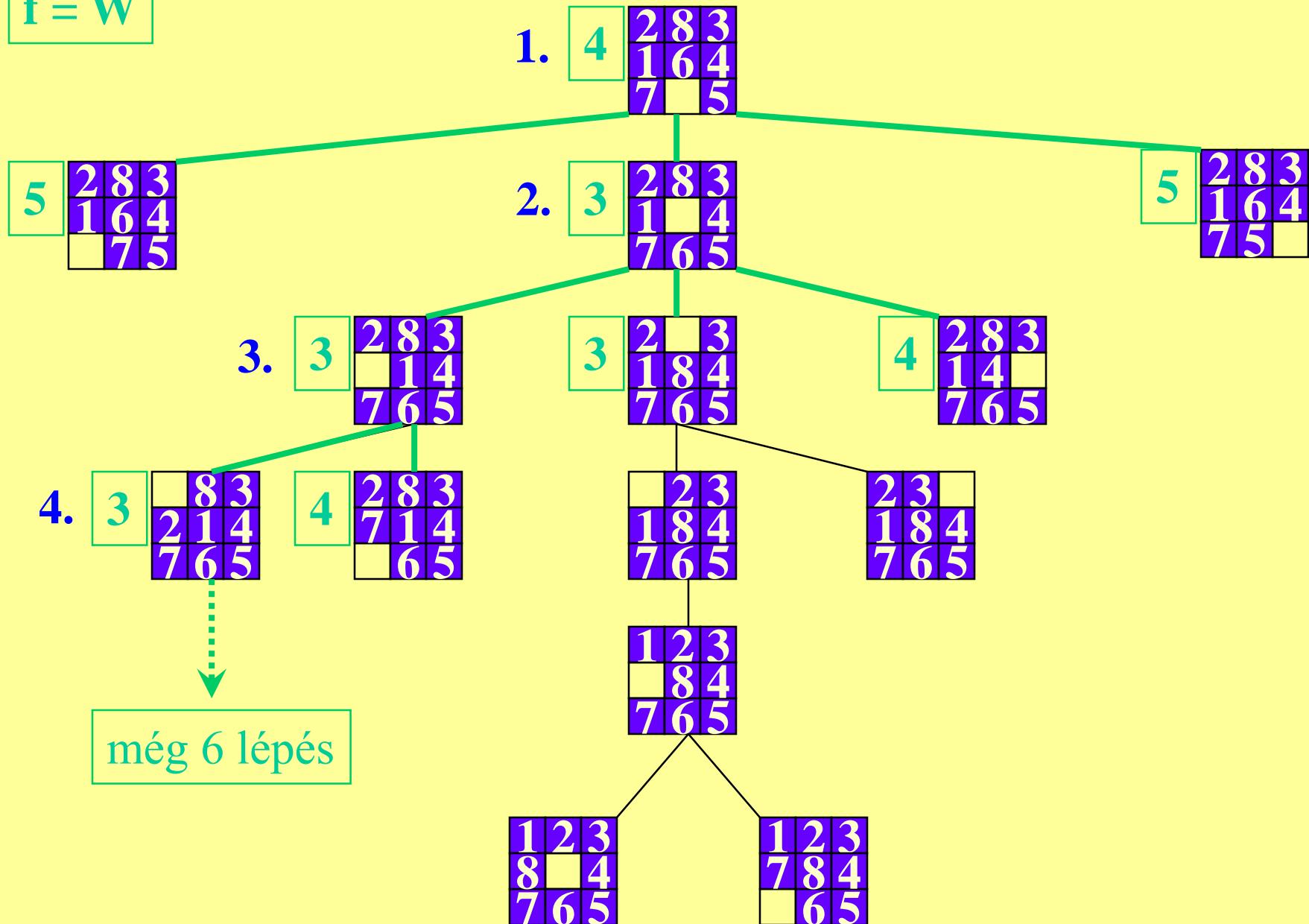
## □ Megjegyzés:

- 8-kirakó :  $W$  és  $P$  minden tulajdonsággal bír.
- $h$  monoton +  $h$  célban nulla  $\Rightarrow h$  megengedhető
- Zéró függvény minden tulajdonsággal bír.

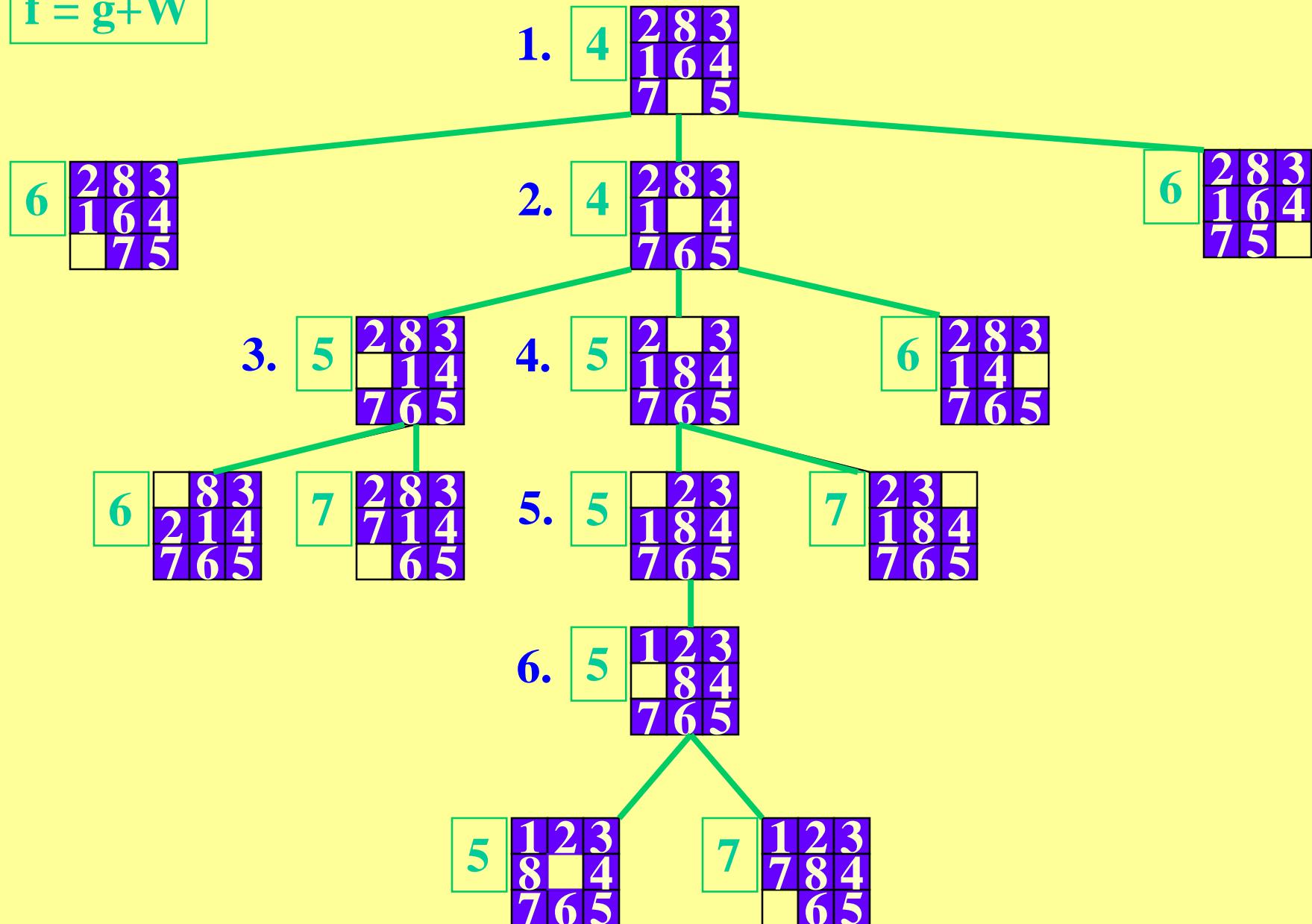
# Nevezetes heurisztikus algoritmusok

Algoritmus	Definíció	Eredmények
<i>Előre tekintő gráfkeresés</i>	$f = h$	nincs említető extra tulajdonsága
<i>A algoritmus</i>	$f = g + h$ és $h \geq 0$	<ul style="list-style-type: none"> <li>megoldást ad, ha van megoldás (még végtelen δ-gráfban is)</li> </ul>
<i>A* algoritmus</i>	$f = g + h$ és $h \geq 0$ és $h \leq h^*$	<ul style="list-style-type: none"> <li>optimális megoldást ad, ha van (még végtelen δ-gráfban is)</li> </ul>
<i>A<sup>c</sup> algoritmus</i>	$f = g + h$ és $h \geq 0$ és $h \leq h^*$ és $h(n) - h(m) \leq c(n, m)$	<ul style="list-style-type: none"> <li>optimális megoldást ad, ha van (még végtelen δ-gráfban is)</li> <li>egy csúcs kiterjesztésekor ismeri az odavezető legolcsóbb utat (legfeljebb egyszer terjeszt ki)</li> </ul>
	egyenletes gráfkeresés: $f = g + 0$	

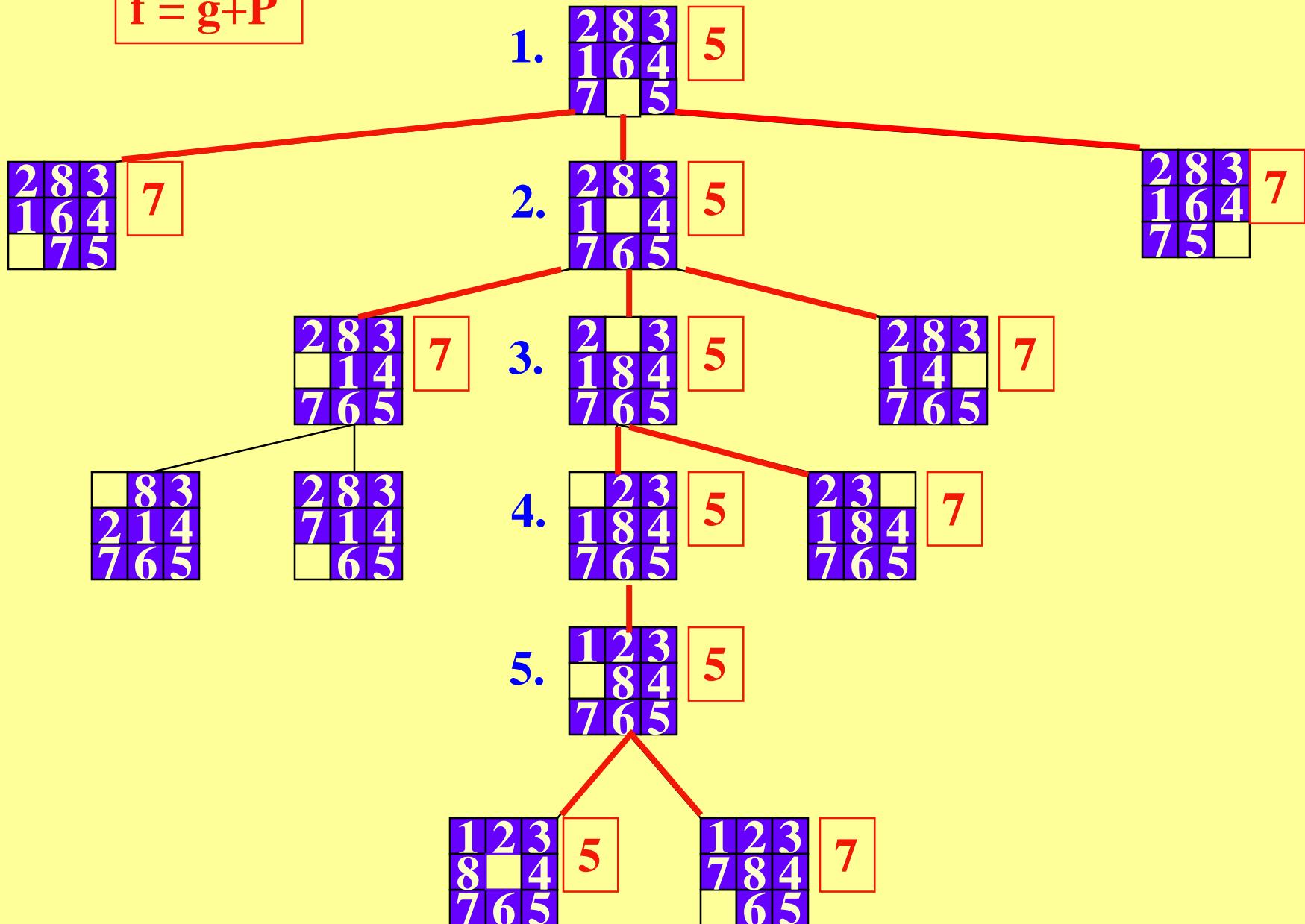
$$\mathbf{f} = \mathbf{W}$$



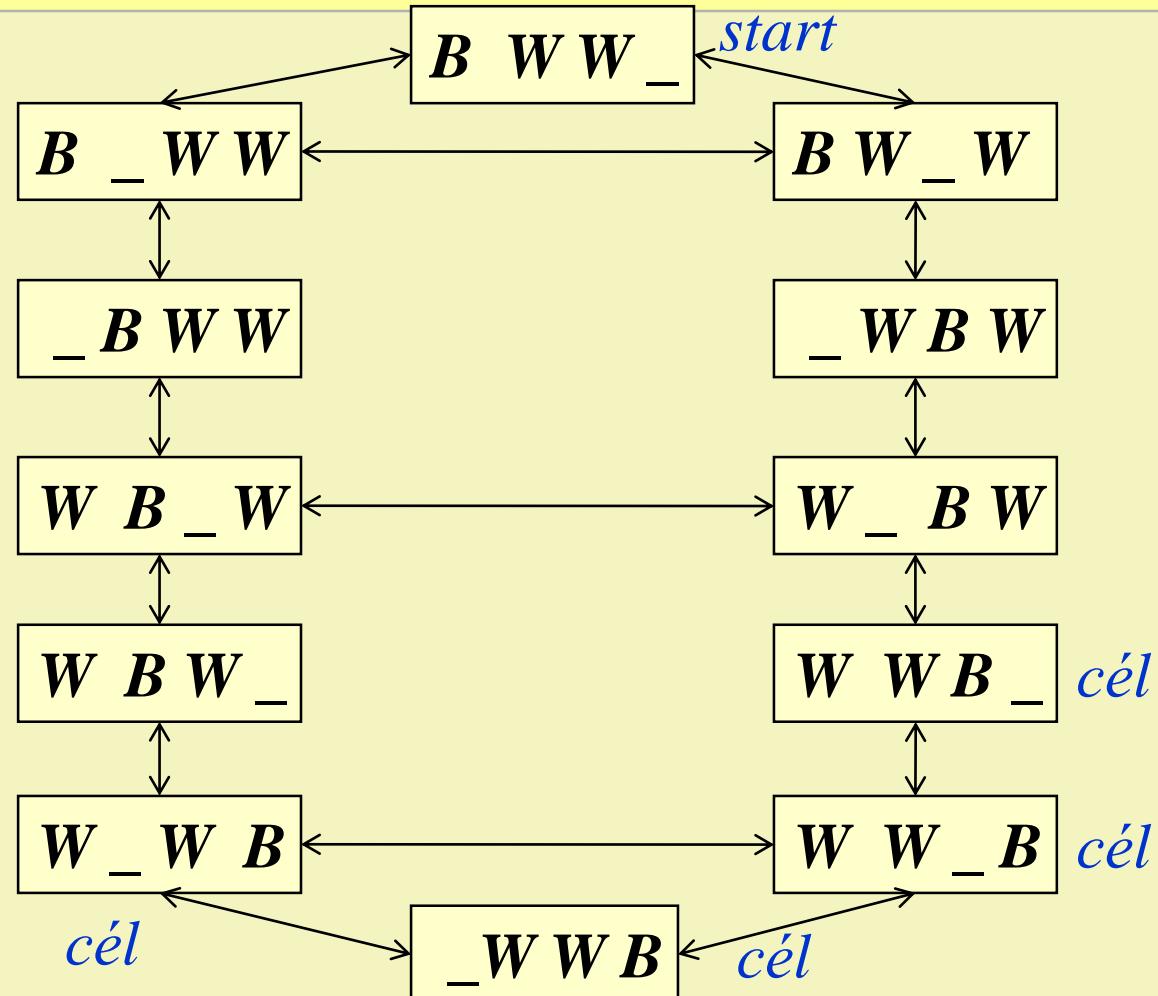
$$\mathbf{f} = \mathbf{g} + \mathbf{W}$$



$$\mathbf{f} = \mathbf{g} + \mathbf{P}$$

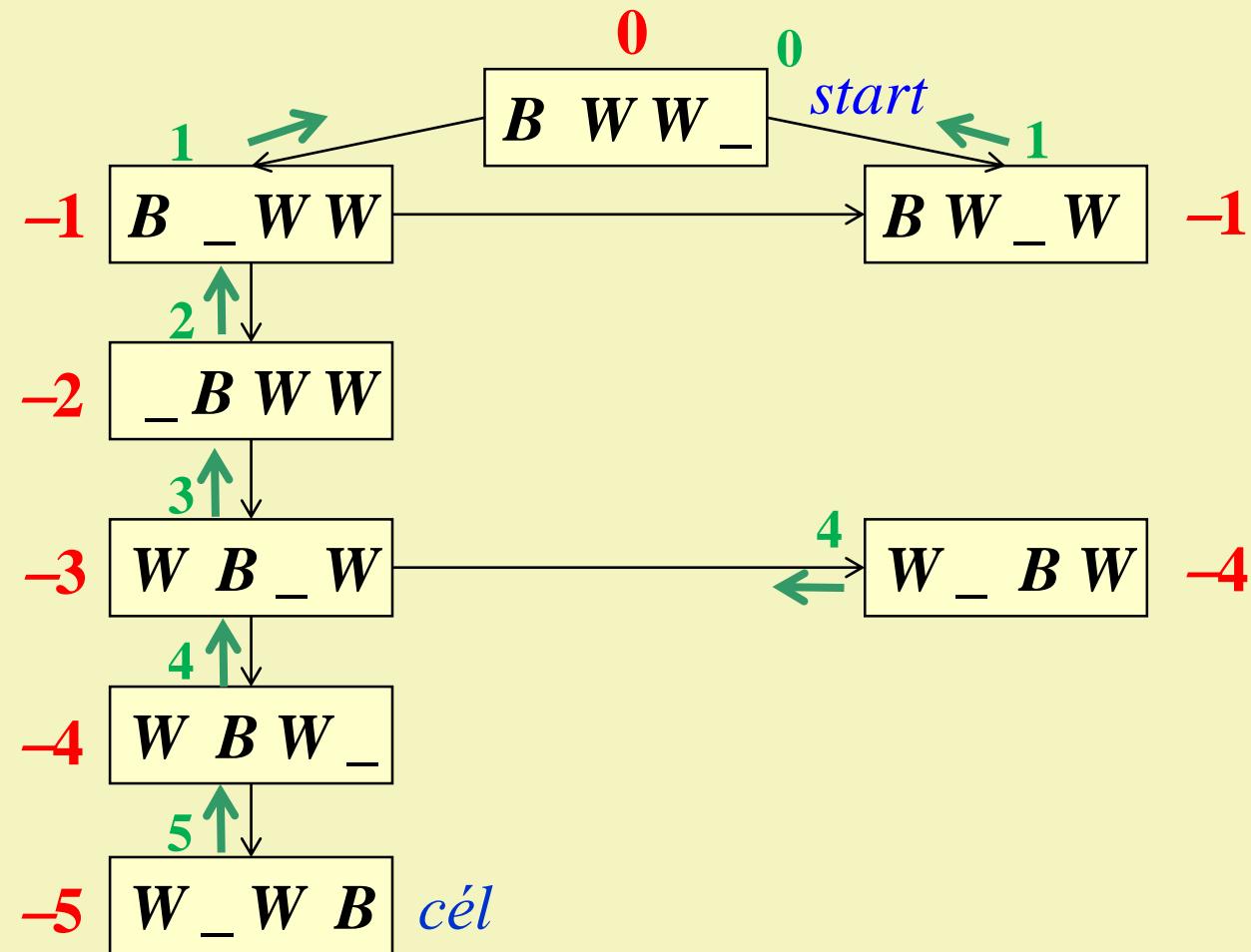


# Fekete-fehér kirakó állapot gráfja



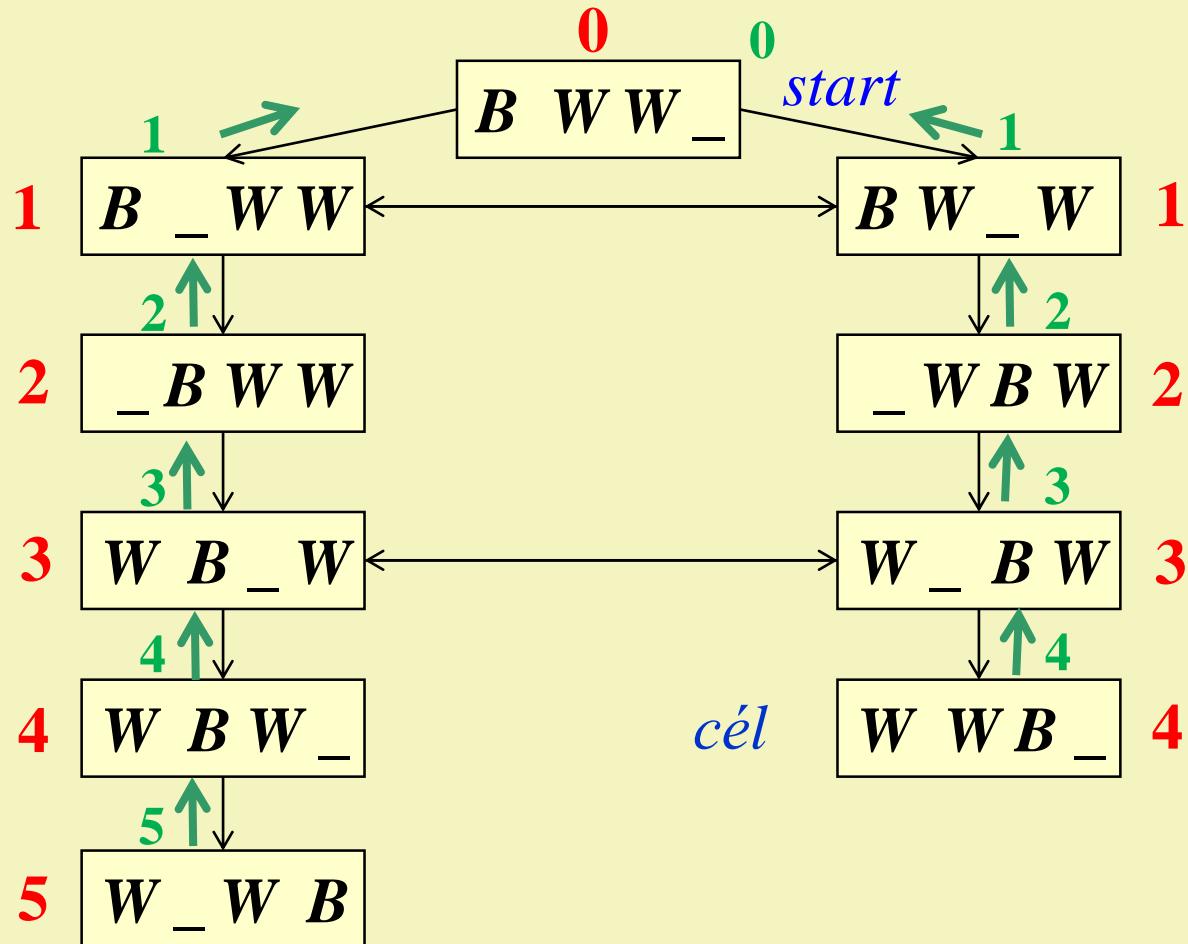
# Mélységi gráfkeresés

$$f = -g$$



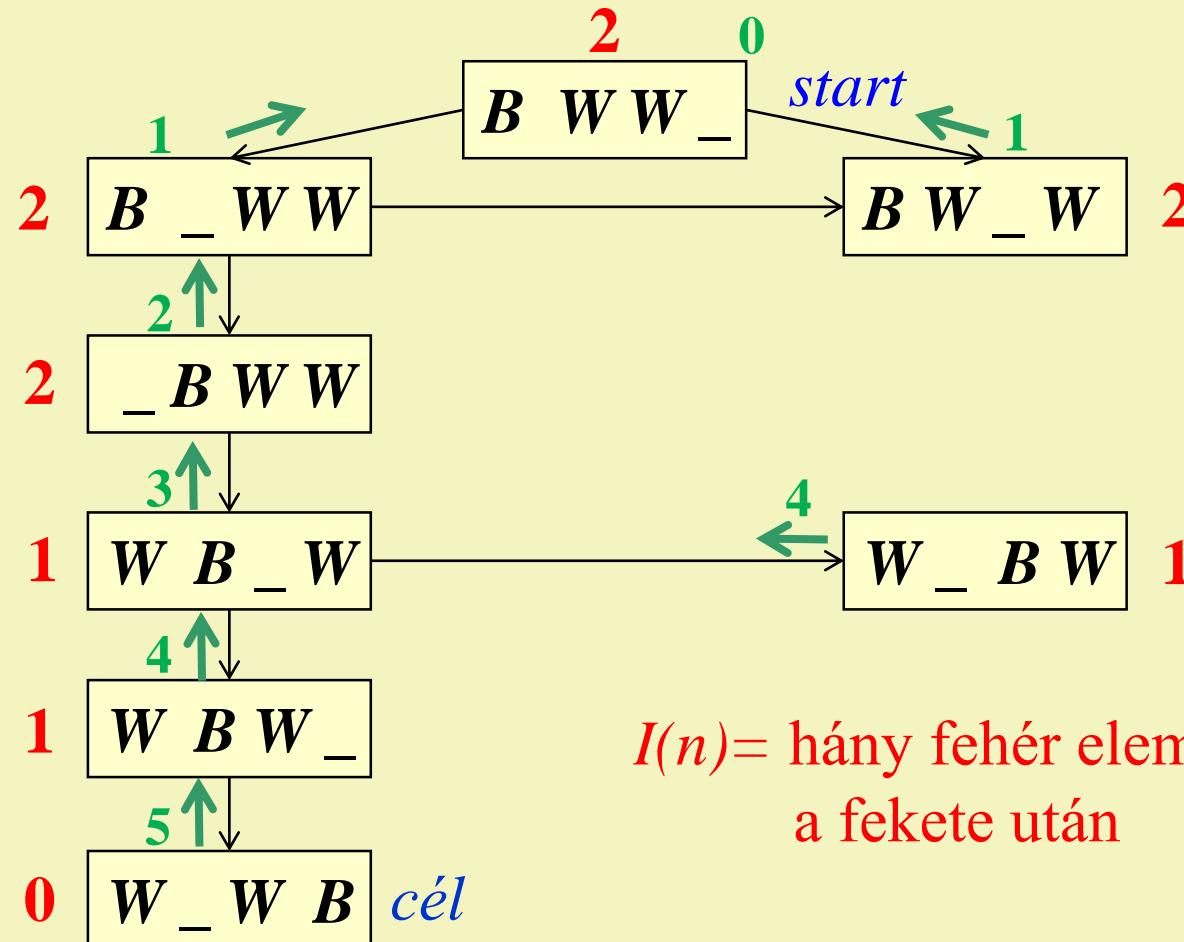
# Szélességi gráfkeresés

$$f = g$$



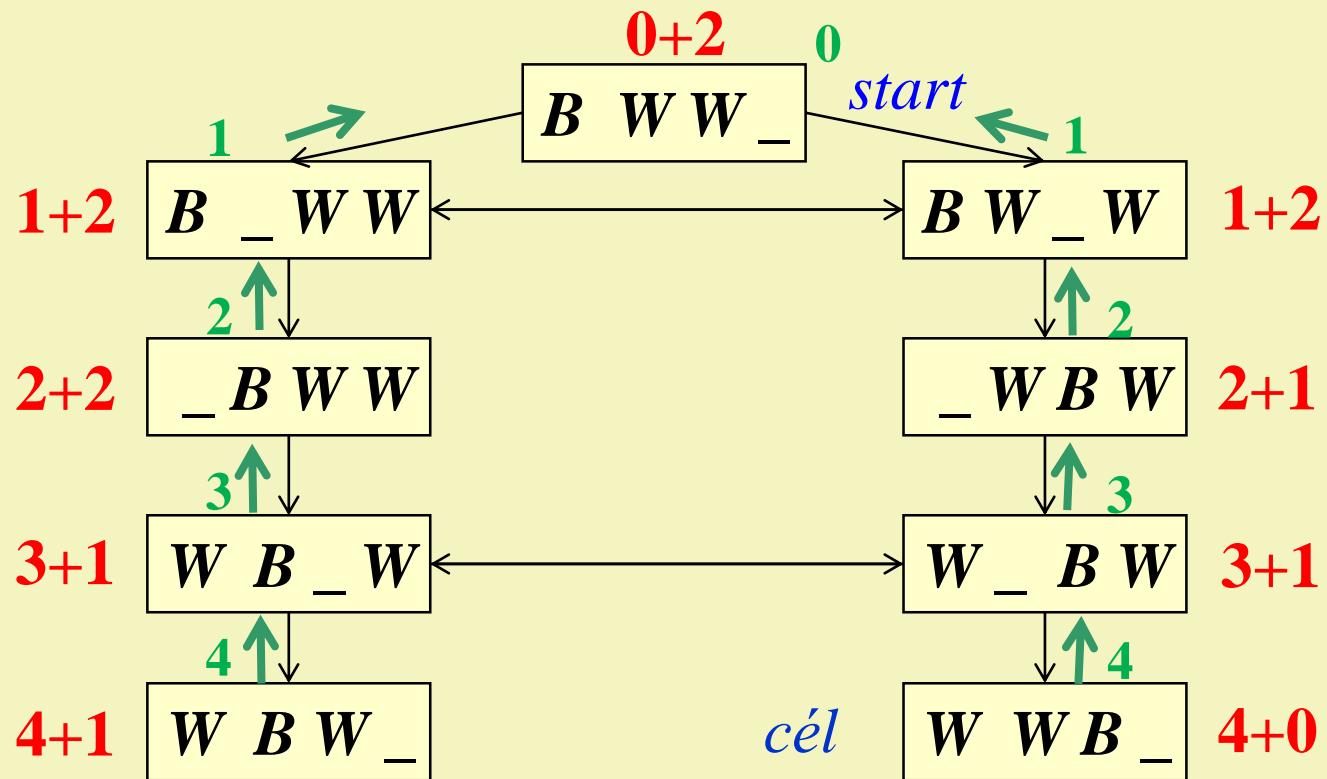
# Előre tekintő gráfkeresés

$f = I$



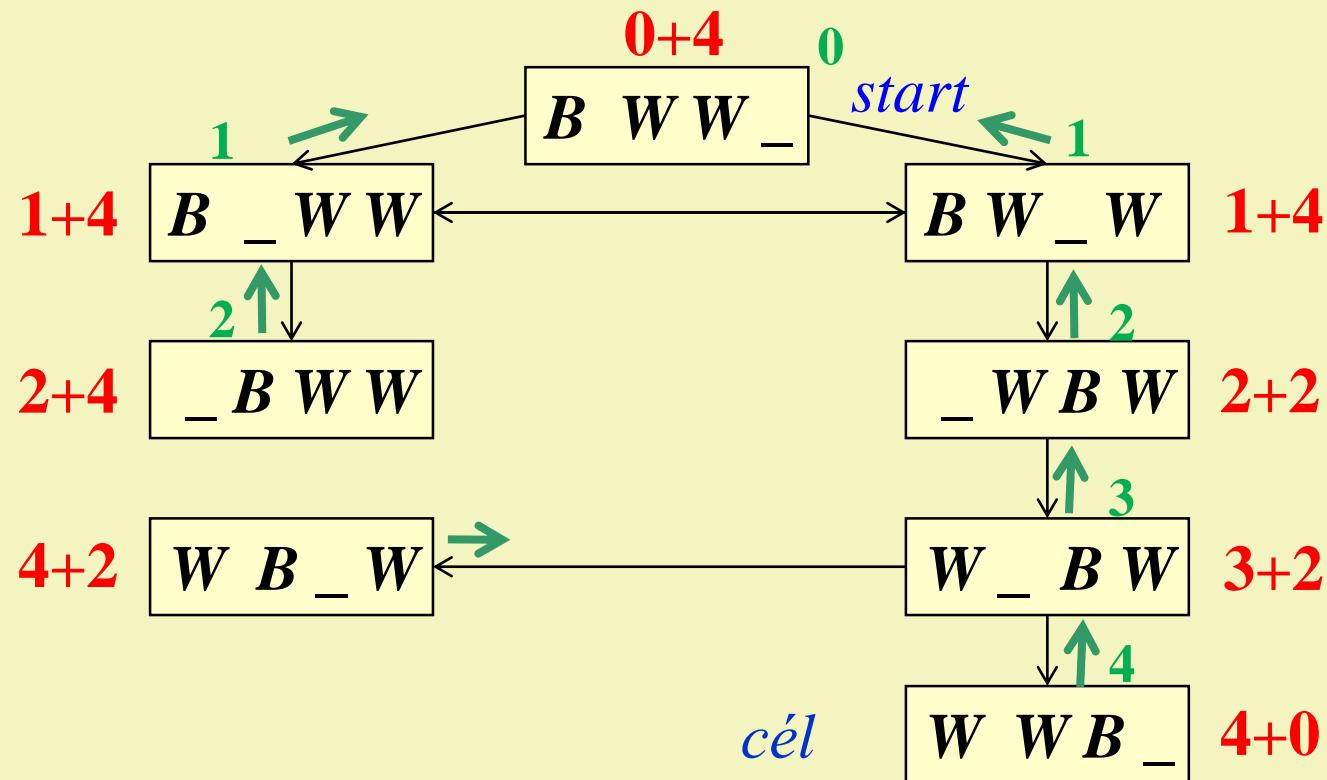
# *A algoritmus*

$$f = g + I$$



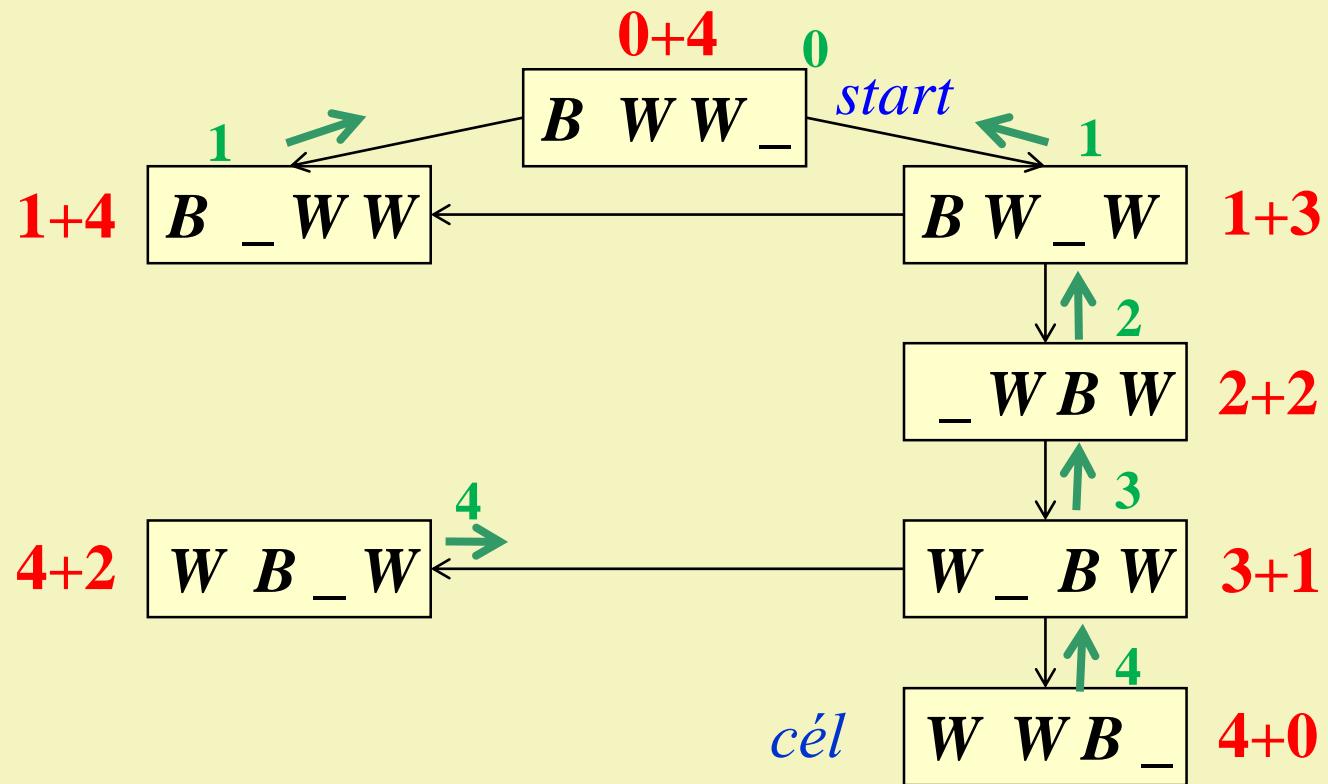
# *A algoritmus*

$$f = g + 2*I$$



# *A algoritmus*

$$f = g + 2*I - (1 \text{ ha van } BW_- \text{ vagy } _BW)$$



# *Elemzés*

$A^c alg$

f	Alg	mo	G	$\Gamma$
-g	$MGK$	5	8	5
g	$SZGK$	4	10	8
l	<i>Előre tekintő</i>	5	8	5
$g+l$	$A alg$	4	9	7
$g+2*l$	$A alg$	4	8	6
$g+2*l-1(ha...)$	$A alg$	4	7	5

$A^c alg$

$A^c alg$

### 3.3. *A<sup>\*</sup> algoritmus* hatékonysága

#### Hatókonyság

##### Memória igény

Zárt csúcsok száma termináláskor jól jellemzi a kereső gráf méretét

##### Futási idő

Kiterjesztések száma a zárt csúcsok számához viszonyítva

A hatékonyságot a **megengedhető feladatokon** vizsgáljuk, amelyeknek van megoldása és ismert egy megengedhető heurisztikája, tehát az *A<sup>\*</sup> algoritmus* optimális megoldást talál hozzájuk.

### 3.3.1. A memória igény vizsgálata

- $CLOSED_S$  ~ az  $S$  gráfkereső algoritmus által lezárt (kiterjesztett) csúcsok halmaza
- Rögzítsünk egy feladatot és két,  $X$  és  $Y$  gráfkereső algoritmust  
**Az adott feladatra nézve**
  - a.  $X$  nem rosszabb  $Y$ -nál, ha  $CLOSED_X \subseteq CLOSED_Y$
  - b.  $X$  jobb  $Y$ -nál, ha  $CLOSED_X \subsetneq CLOSED_Y$
- Ezek alapján összevethető
  1. két eltérő heurisztikájú  $A^*$  algoritmus ugyanazon a feladaton, azaz a két heurisztika.
  2. két útkereső algoritmus, például az  $A^*$  algoritmus és egy másik – szintén optimális megoldást garantáló – gráfkereső algoritmus a megengedhető problémák egy részhalmazán.

# Különböző heurisztikájú $A^*$ algoritmusok memória igényének összehasonlítása

- Az  $A_1$  ( $h_1$  heurisztikával) és  $A_2$  ( $h_2$  heurisztikával)  $A^*$  algoritmusok közül az  $A_2$  jobban informált, mint az  $A_1$ , ha minden  $n \in N \setminus T$  csúcsra teljesül, hogy  $h_1(n) < h_2(n)$ .

$$h_1(n) < h_2(n) \leq h^*(n)$$
- Bebizonyítható, hogy a jobban informált  $A_2$  nem rosszabb a kevésbé informált  $A_1$ -nél, azaz  $CLOSED_{A_2} \subseteq CLOSED_{A_1}$

# Megjegyzés

- A gyakorlatban a bizonyított állításnál enyhébb feltételek mellett látványosabb különbségekkel is találkozhatunk:
  - Sokszor akkor is jóval több csúcsot terjeszt ki az  $A_1$ , mint  $A_2$  ( $CLOSED_{A_2} \subset CLOSED_{A_1}$ ), ha csak a  $h_1 \leq h_2$  teljesül, esetleg nem is minden csúcsra.
  - Példák:
    - 8-as tologató:  $0 \leq W \leq P$  ( $\leq F$ )
    - Fekete-fehér:  $I \leq M$  ( $\leq 2 \cdot I$ )
- Minél jobban (közelebbről) becsli (ha lehet, alulról) a heurisztika a  $h^*$ -ot, várhatóan annál kisebb lesz a memória igénye.

# 15-kirakó

$f =$	$g+0$	$g+W$	$g+P$
6 lépéses megoldás	117	7	6
13 lépéses megoldás	32389	119	13
21 lépéses megoldás	n.a.	3343	145
30 lépéses megoldás	n.a.	n.a.	1137
34 lépéses megoldás	n.a.	n.a.	3971

# *Különböző gráfkereső algoritmusok memória igényének összehasonlítása*

- Célunk megmutatni azt, hogy az *A<sup>\*</sup> algoritmus* memória igénye nem rossz más, hasonló eredményű gráfkereső algoritmusok memória igényéhez képest.
- **Megengedhetőnek** nevezzük azt az gráfkereső algoritmust, amely megengedhető heurisztikájú útkeresési problémákra optimális megoldást talál, ha van megoldás.
- Példák:
  - EGK :  $f(n)=g(n)+0$
  - *A (A<sup>\*</sup>) algoritmus* :  $f(n)=g(n)+h(n)$
  - *A<sup>\*\*</sup> algoritmus*:  $f(n)=\max_{m \in start \rightarrow n} (g(m)+h(m))$  és a célcsúcs előnyben

# Bizonyítható eredmények

- *A<sup>\*</sup> algoritmus* lehet rosszabb más megengedhető algoritmusnál egy adott megengedhető feladaton. De
  - Bármelyik megengedhető algoritmus is lehet rosszabb más megengedhető algoritmusnál egy adott megengedhető feladaton.
  - *A<sup>\*</sup>* soha nem rosszabb a többi megengedhető algoritmusnál a monoton megszorításos heurisztikájú megengedhető feladatokon.
  - Az *A<sup>\*</sup>-nál nincs jobb* megengedhető algoritmus az olyan feladatokon, ahol van olyan optimális megoldási út, amelynek csúcsaira a célcsúcs kivételével  $h < h^*$  áll fenn).

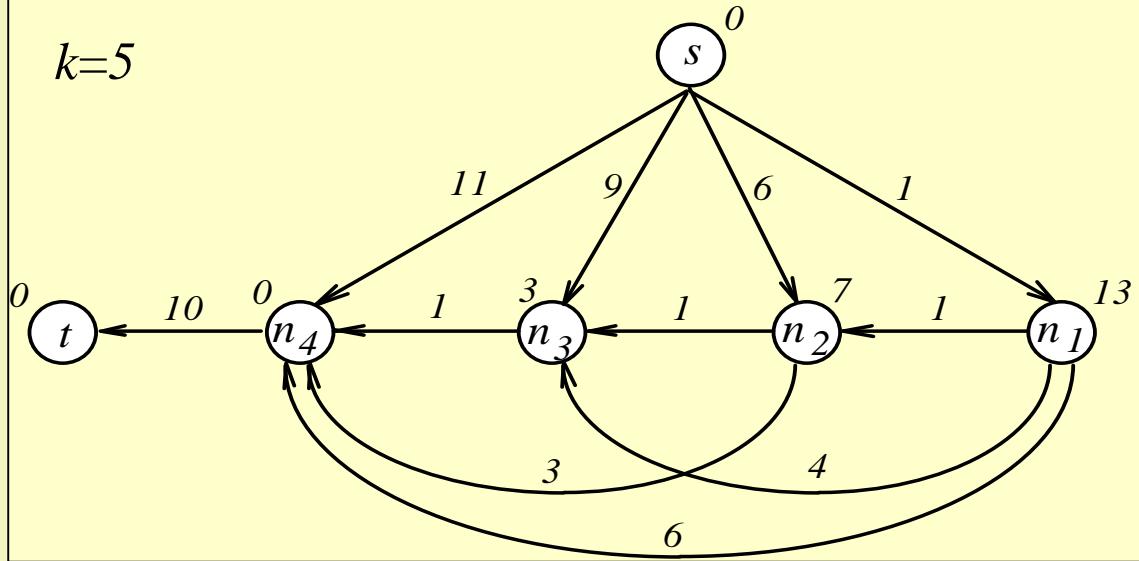
### 3.3.2. A futási idő elemzése

- Zárt csúcsok száma:  $k = |CLOSED|$
- Alsókorlát:  $k$ 
  - Egy monoton megszorításos heurisztika mellett egy csúcs legfeljebb csak egyszer terjesztődik ki,
  - habár ettől még a kiterjesztett csúcsok száma igen sok is lehet (lásd egyenletes keresés)
- Felsőkorlát:  $2^{k-1}$ 
  - lásd. Martelli példáját

# Megjegyzés

- ❑ Másik heurisztikával ugyanazon a feladaton természetesen javítható a kiterjesztések száma, bár nem biztos, hogy ez minden esetben tényleges javulás lesz, hiszen másik heurisztika esetén a  $k$  értéke is változhat.
- ❑ A kiterjesztések száma ugyanis a kiterjesztett (zárt) csúcsok számához viszonyított szám
  - $h_1$  heurisztika mellett  $k_1$  darab zárt csúcs, és  $2^{k_1-1}$  kiterjesztés
  - $h_2$  heurisztika mellett  $k_2$  darab zárt csúcs, és  $k_2$  kiterjesztés
  - Mégis lehet, hogy  $2^{k_1-1} < k_2$ , ha  $k_1 \ll k_2$ .

# Martelli példája



Az  $n_1, \dots, n_{k-1}$  csúcsokba rendre  $2^0, 2^1, \dots, 2^{k-2}$  különböző út vezet. Így elvileg  $2^{k-1}$  kiterjesztés történhet. És itt ennyi is történik.

$$N = \{n_i \mid i=0..k\} \text{ ahol } s=n_0, t=n_k$$

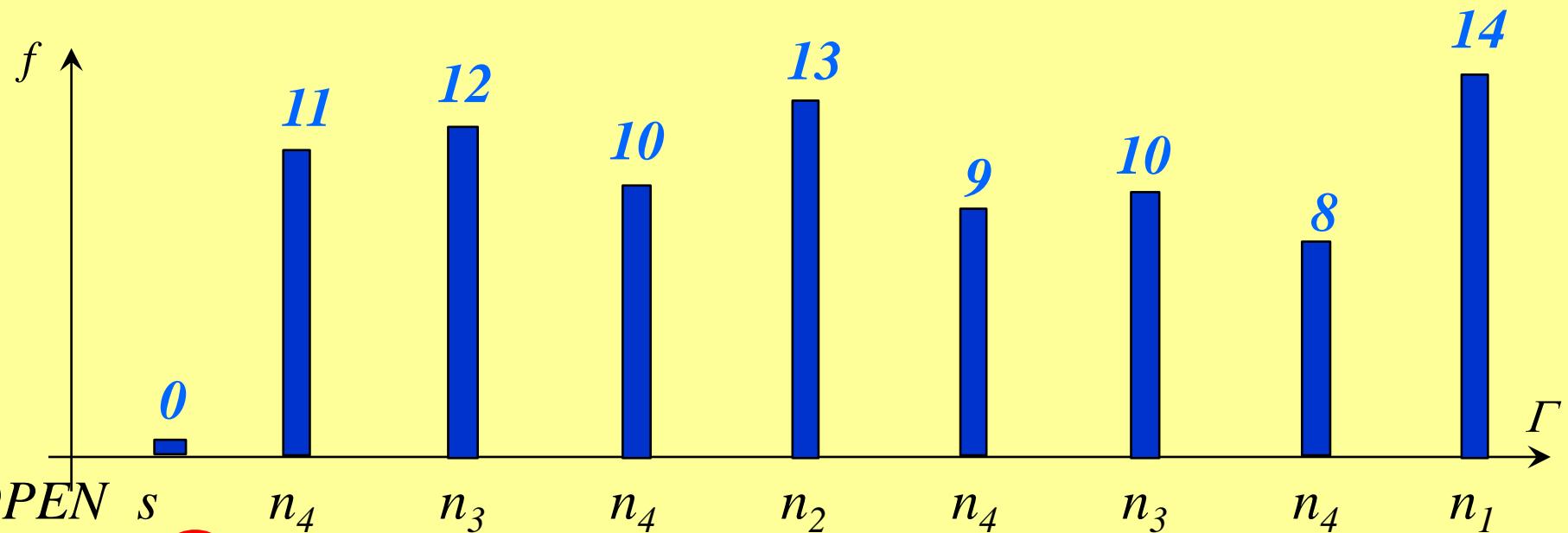
$$A = \{(n_i, n_j) \mid 0 \leq i < j < k\} \cup \{(n_{k-1}, t)\}$$

$$c(n_i, n_j) = 2^{k-2-i} - 2^{k-1-j} + j-i \quad (0 \leq i < j < k)$$

$$h(n_i) = c(s, n_{k-1}) - c(s, n_i) + k-1-i \quad (0 < i < k), \quad h(s) = h(t) = 0$$

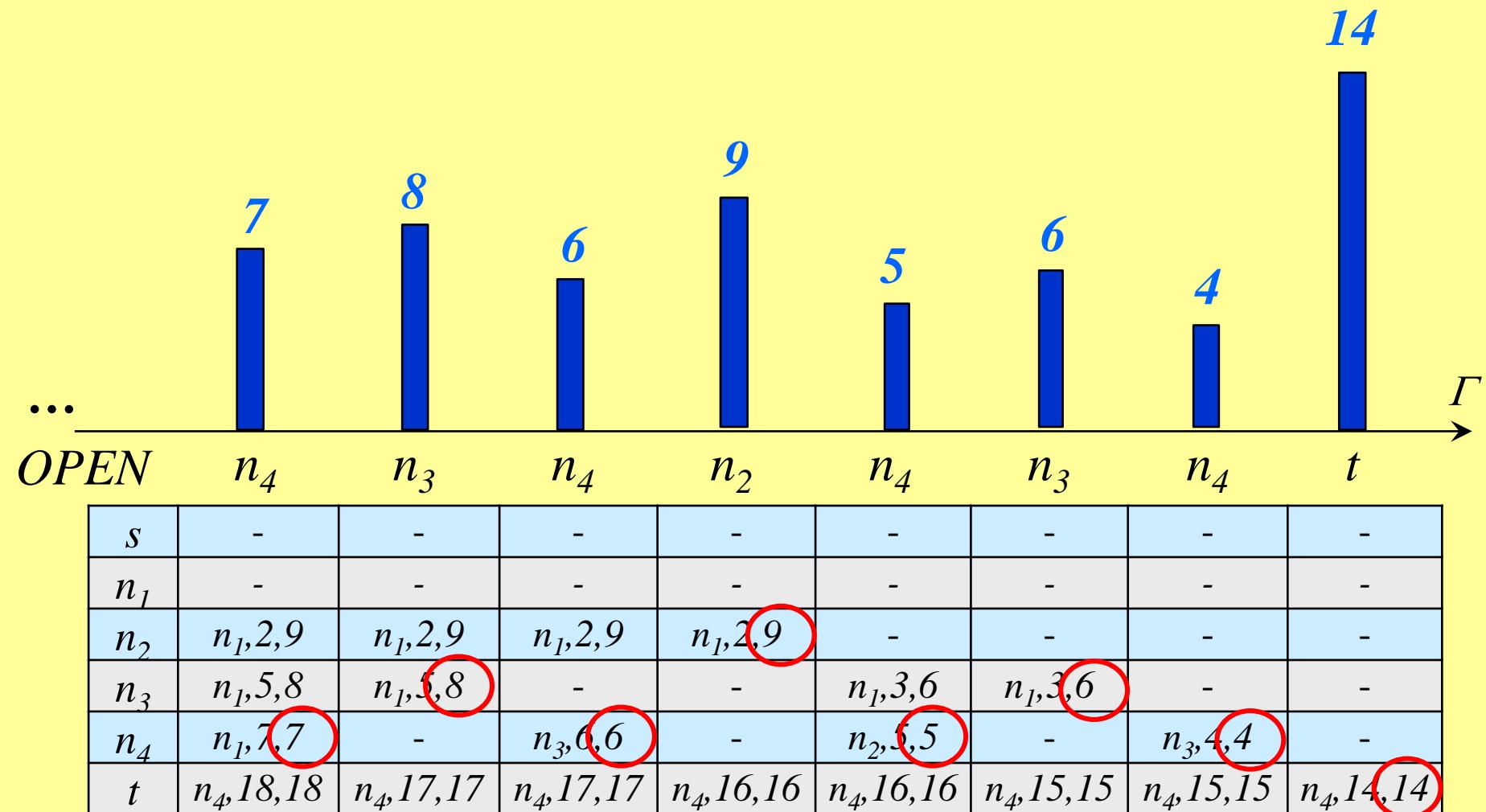
$$c(n_{k-1}, t) = h(n_1) - k + 2$$

# Működési grafikon

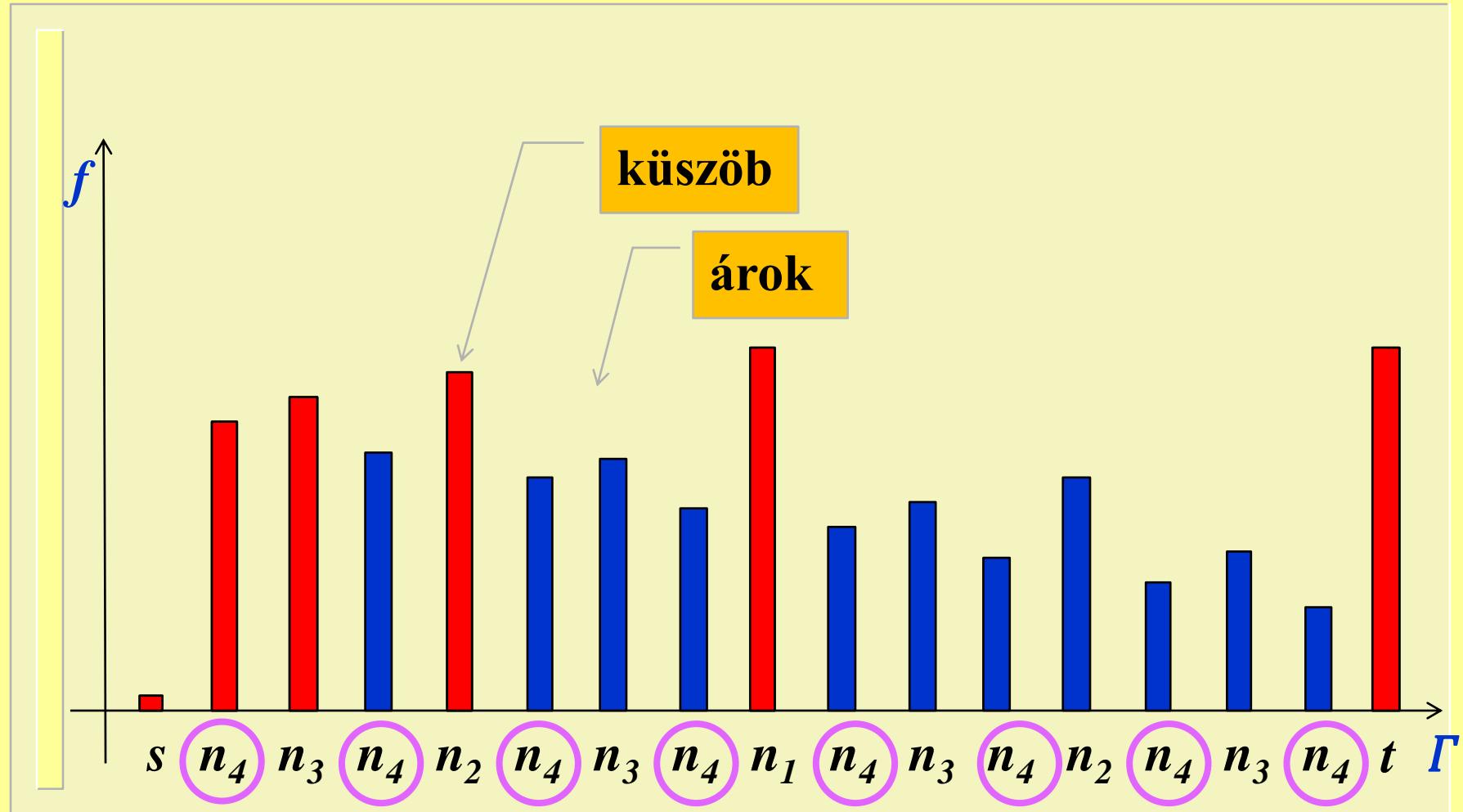


$s$	$nil, 0, 0$	-	-	-	-	-	-	-	-
$n_1$	-	$s, 1, 14$	$s, 1, 14$	$s, 1, 14$	$s, 1, 14$	$s, 1, 14$	$s, 1, 14$	$s, 1, 14$	$s, 1, 14$
$n_2$	-	$s, 6, 13$	$s, 6, 13$	$s, 6, 13$	$s, 6, 13$	-	-	-	-
$n_3$	-	$s, 9, 12$	$s, 9, 12$	-	-	$n_2, 7, 10$	$n_2, 7, 10$	-	-
$n_4$	-	$s, 11, 11$	-	$n_3, 10, 10$	-	$n_2, 9, 9$	-	$n_2, 8, 8$	-
$t$	-	-	$n_4, 21, 21$	$n_4, 21, 21$	$n_4, 20, 20$	$n_4, 20, 20$	$n_4, 19, 19$	$n_4, 19, 19$	$n_4, 18, 18$

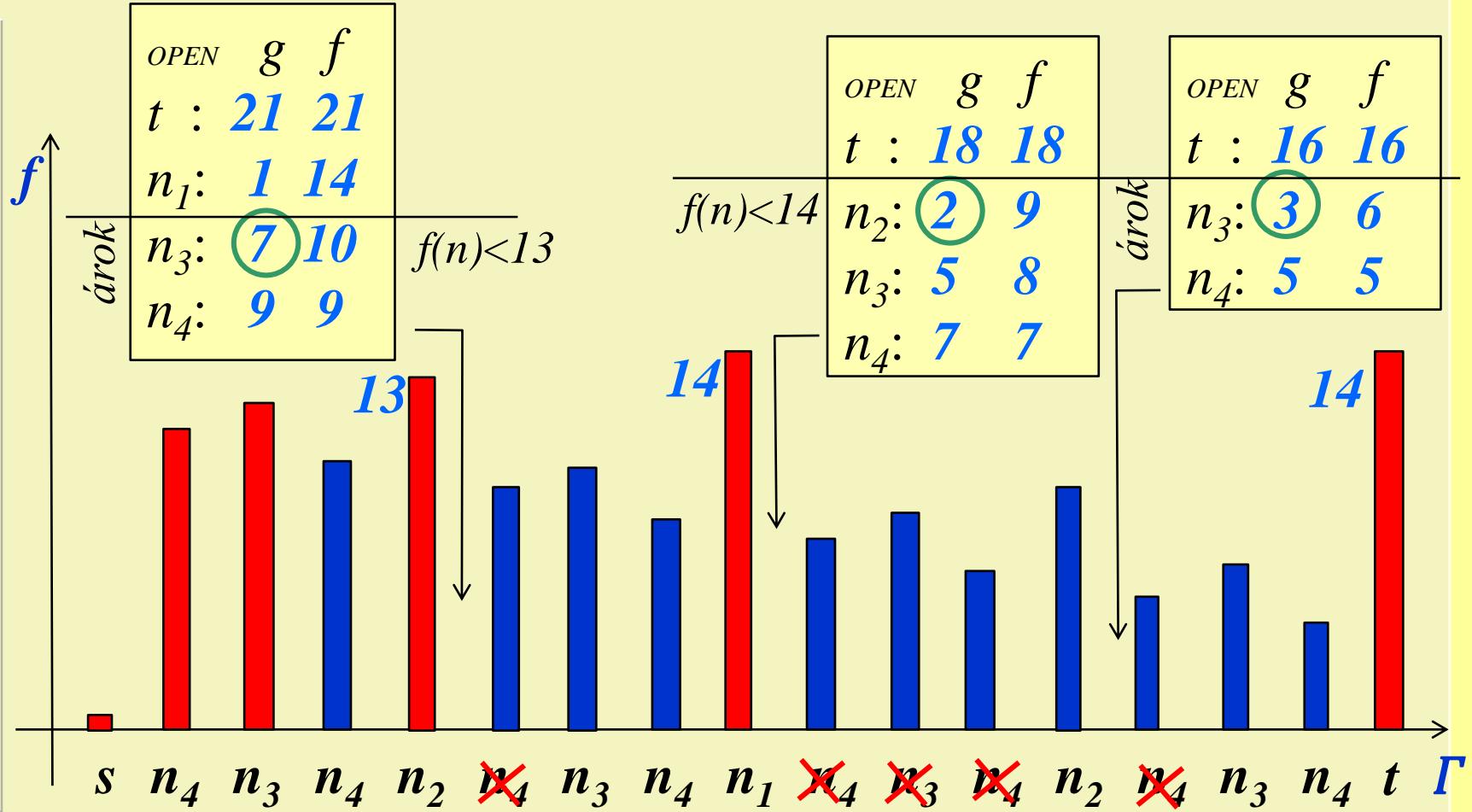
# Működési grafikon



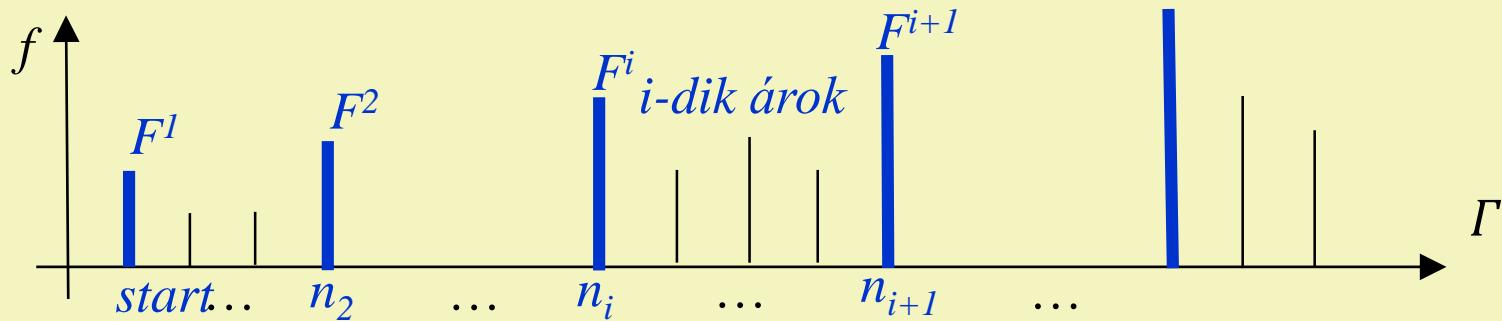
# *Árkon belüli kiterjesztések száma az A\* algoritmusnál*



# Csökkentsük a kiterjesztések számát



# *A probléma oka és csillapítása*



- Egy csúcs – még akár egy árkon belül is – többször kiterjesztődhet.
- Használunk az árkokban egy másik, egy **másodlagos (belő)** kiértékelő függvényt! Bizonyítható, hogy ettől nem változik meg az **egy árokban kiterjesztett csúcsok halmaza**, csak **a csúcsok árkon belüli kiterjesztési sorrendje** lesz más, ennél fogva pedig a küszöbcsúcsok, azok sorrendje és értékei változatlanok maradnak. Ennél a belő kiértékelő függvény csak a futási időt (kiterjesztések számát) befolyásolja.

## *B algoritmus*

- ❑ Martelli javasolta belső kiértékelő függvénynek a  $g$  költség függvényt.
- ❑ A *B algoritmust* az *A algoritmusból* kapjuk úgy, hogy bevezetjük az  $F$  aktuális küszöbértéket, majd
  - az 1. lépést kiegészítjük az  $F := f(s)$  értékadással,
  - a 4. lépést pedig helyettesítjük az  
**if**  $\min_f(\text{OPEN}) < F$   
    **then**  $n := \min_g(m \in \text{OPEN} \mid f(m) < F)$   
    **else**  $n := \min_f(\text{OPEN}); F := f(n)$   
**endif** elágazással.

# *B algoritmus futási ideje*

- A *B algoritmus* ugyanúgy működik, mint az  $A^*$ , azzal a kivétellel, hogy egy árokhoz tartozó csúcsot csak egyszer terjeszt ki.
- *Futási idő elemzése:*
  - Legrosszabb esetben
    - minden zárt csúcs először küszöbcsúcsként terjesztődik ki. (Csökkenő kiértékelő függvény mellett egy csúcs csak egyszer, a legelső kiterjesztéskor lehet küszöb.)
    - Az  $i$ -dik árok legfeljebb az összes addigi  $i-1$  darab küszöbcsúcsot tartalmazhatja (a start csúcs nélkül).
  - Így az összes kiterjeszték száma legfeljebb  $\frac{1}{2} \cdot k^2$

# Heurisztika szerepe

## □ Milyen a jó heurisztika?

- megengedhető:  $h(n) \leq h^*(n)$ 
  - Bár nincs mindenkor szükség optimális megoldásra.
- jól informált:  $h(n) \sim h^*(n)$
- monoton megszorítás:  $h(n) - h(m) \leq c(n, m)$ 
  - Ilyenkor nem érdemes *B algoritmust* használni

## □ Változó heurisztikák:

- $f = g + \phi \cdot h$  ahol  $\phi \sim d$
- $B'$  algoritmus

# *B' algoritmus*

```
if  $h(n) < \min_{m \in \Gamma(n)} (c(n,m) + h(m))$   
then  $h(n) := \min_{m \in \Gamma(n)} (c(n,m) + h(m))$   
else for  $\forall m \in \Gamma(n)$ -re loop  
    if  $h(n) - h(m) > c(n,m)$  then  $h(m) := h(n) - c(n,m)$   
endloop
```

- A  $h$  megengedhető marad
- A  $h$  nem csökken
- A mononton megszorításos élek száma nő



# Kétszemélyes játékok

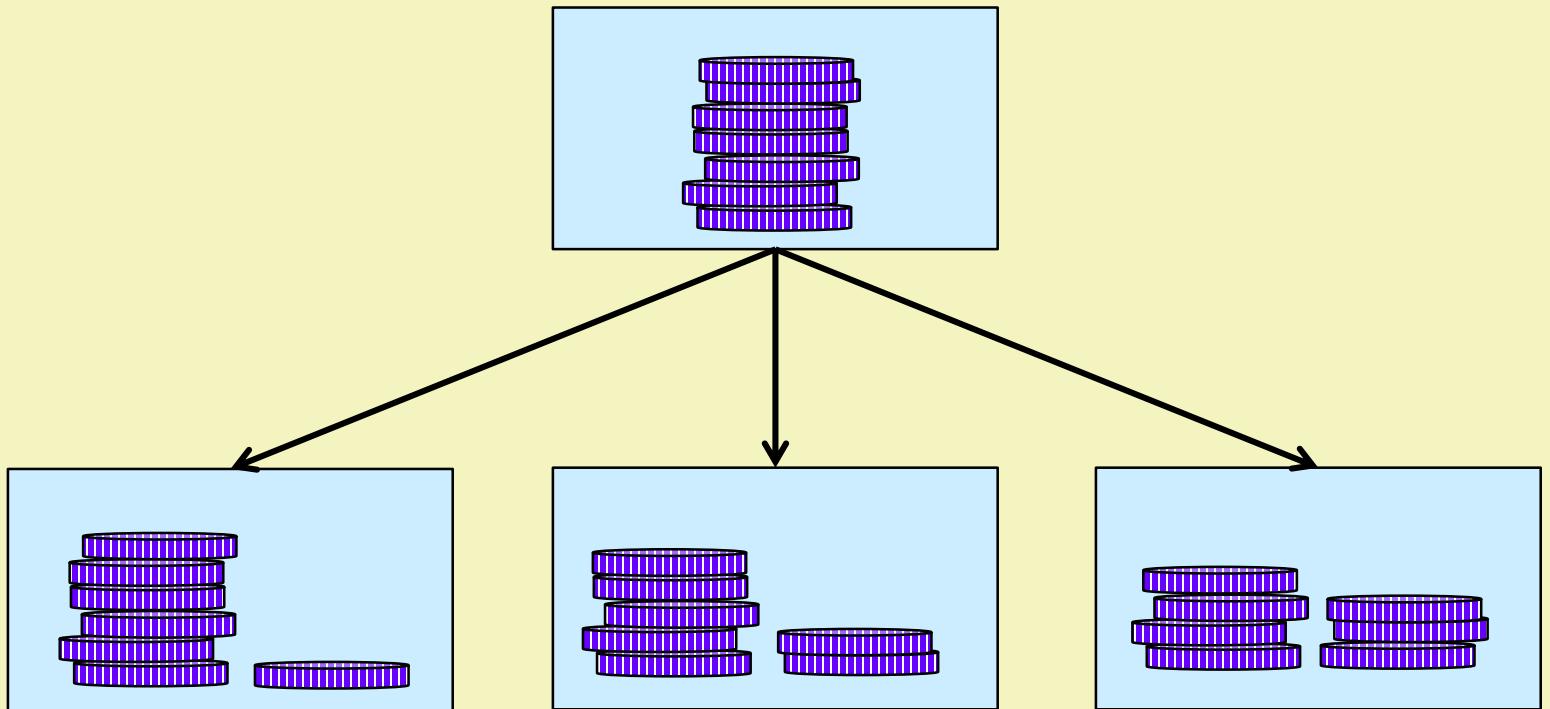
# *Kétszemélyes, teljes információjú, véges, determinisztikus, zéró összegű játékok*

- ❑ Két játékos lép felváltva adott szabályok szerint, amíg a játszma véget nem ér.
- ❑ Mindkét játékos ismeri a maga és az ellenfele összes múltbeli és jövőbeli lépéseiit és lépési lehetőségeit, és azok következményeit.
- ❑ minden lépés véges számú lehetőség közül választható, és minden játszma véges lépésben véget ér. Egy lépés determinisztikus, a véletlennek nincs szerepe.
- ❑ Amennyit a játszma végén az egyik játékos nyer, annyit veszít a másik. (Legegyszerűbb változatban két esélyes: egyik nyer, másik veszít; vagy három esélyes: döntetlen is megengedett)

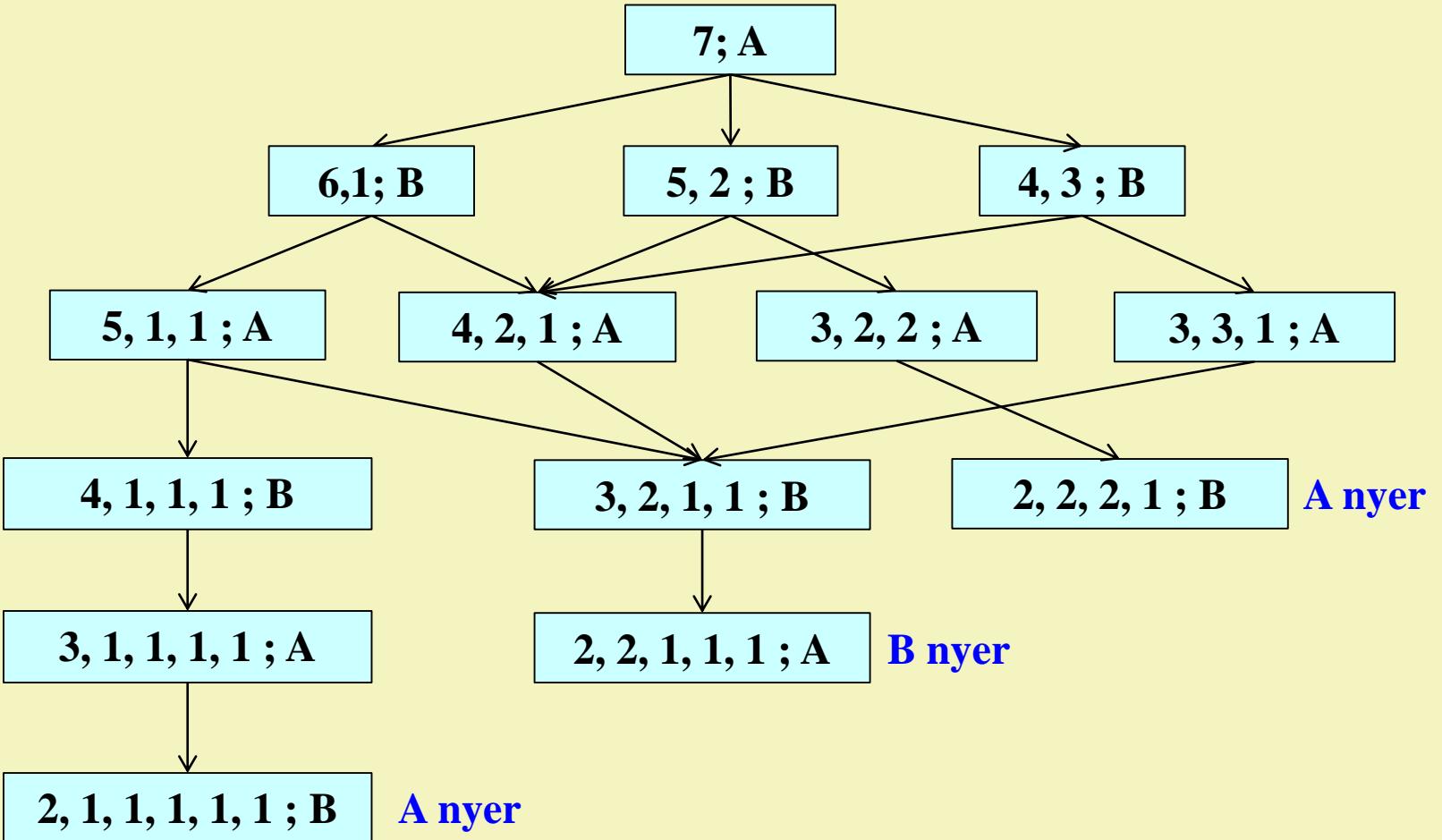
# *Állapottér modell*

- állapot – állás + soron következő játékos
- művelet – lépés
- kezdő állapot – kezdőállás + kezdő játékos
- végállapot – végállás + játékos
- + payoff függvény:  $p_A, p_B : \text{végállapot} \rightarrow \mathbb{R}$  (játékosok:  $A, B$ )
  - Zéró összegű kétszemélyes játékban:
$$p_A(t) + p_B(t) = 0 \quad \text{ minden } t \text{ végállapotra}$$
  - Speciális esetben (a továbbiakban ezt fektételezzük):
    - $p_A(t) = +1$  ha A nyer
    - $p_A(t) = -1$  ha A veszít
    - $p_A(t) = 0$  ha döntetlen

# *Grundy mama játéka*



# Grundy mama állapot-gráfja



# Grundy mama játékfája

A

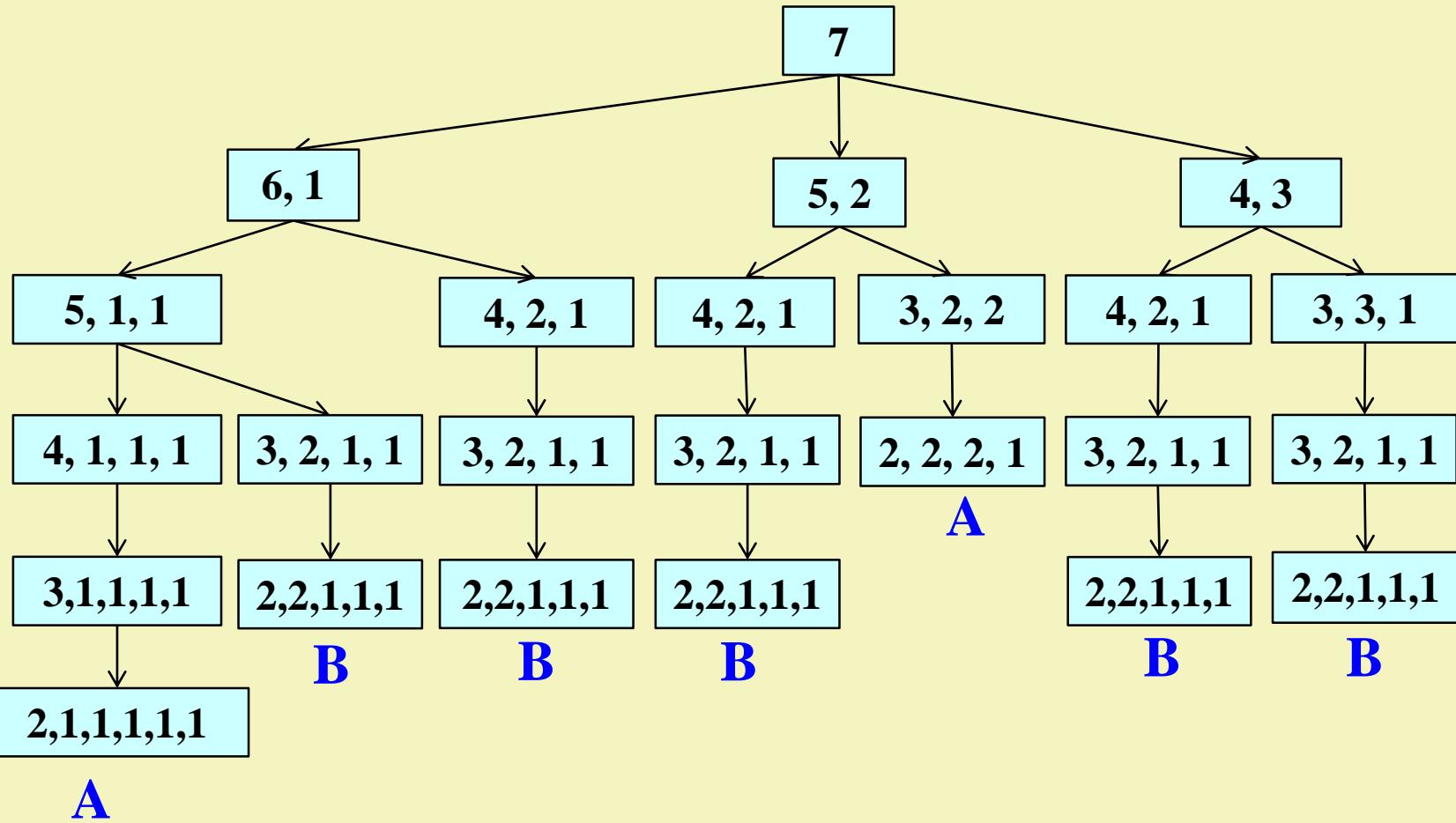
B

A

B

A

B



# *Játékfa*

- csúcs                  – állás (egy állás több csúcs is lehet)
- szint                  – játékos (felváltva az A és B szintjei)
- él                        – lépés                (szintről szintre)
- gyökér                 – kezdőállás (kezdő játékos)
- levél                    – végállások
- ág                        – játszma

# Hogyan tud a **B** játékos biztosan nyerni?

A

B

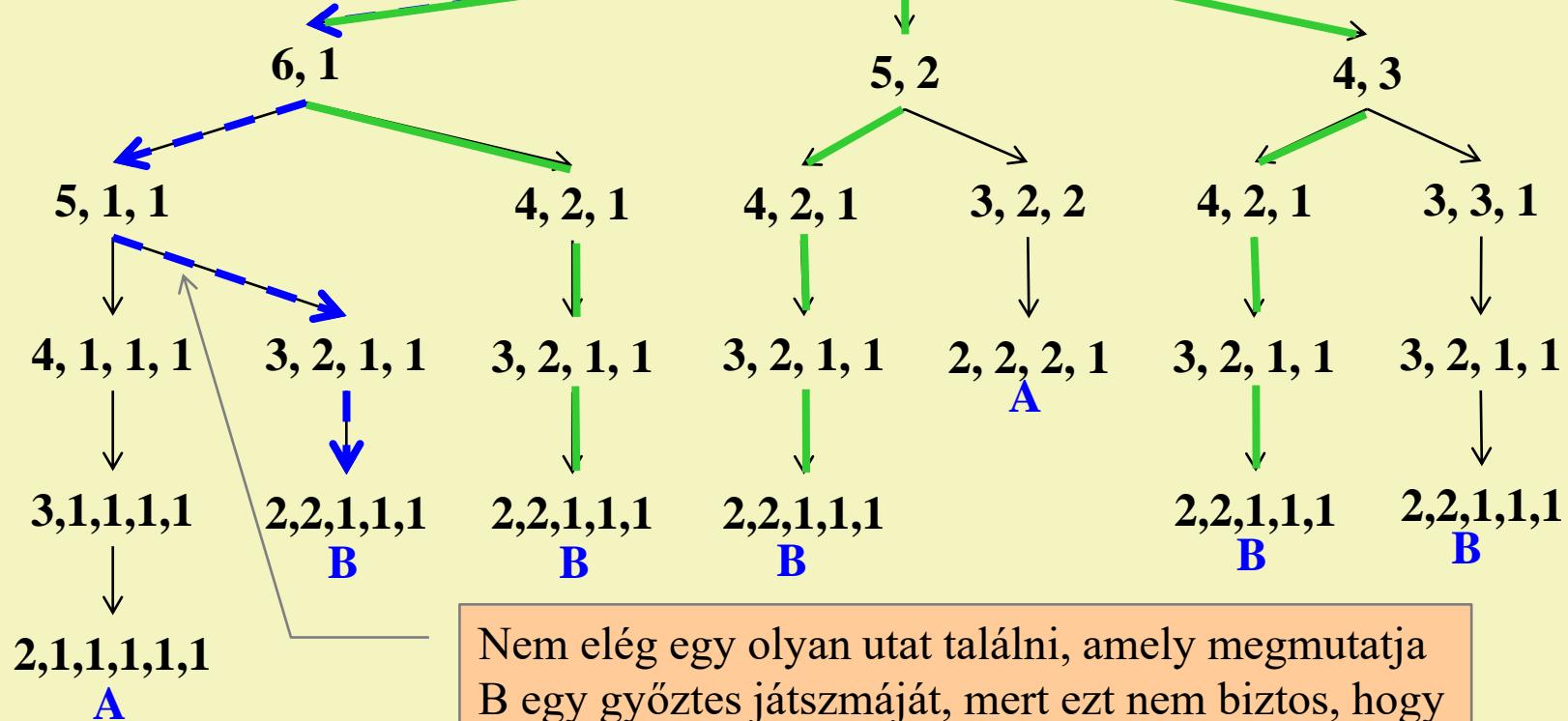
A

B

A

B

A B-nek arra van szüksége, hogy az A minden lépésére legyen olyan válaszlépése, amellyel győzni tud.



Nem elég egy olyan utat találni, amely megmutatja B egy győztes játszmáját, mert ezt nem biztos, hogy B végig tudja játszani az A válaszlépései miatt.

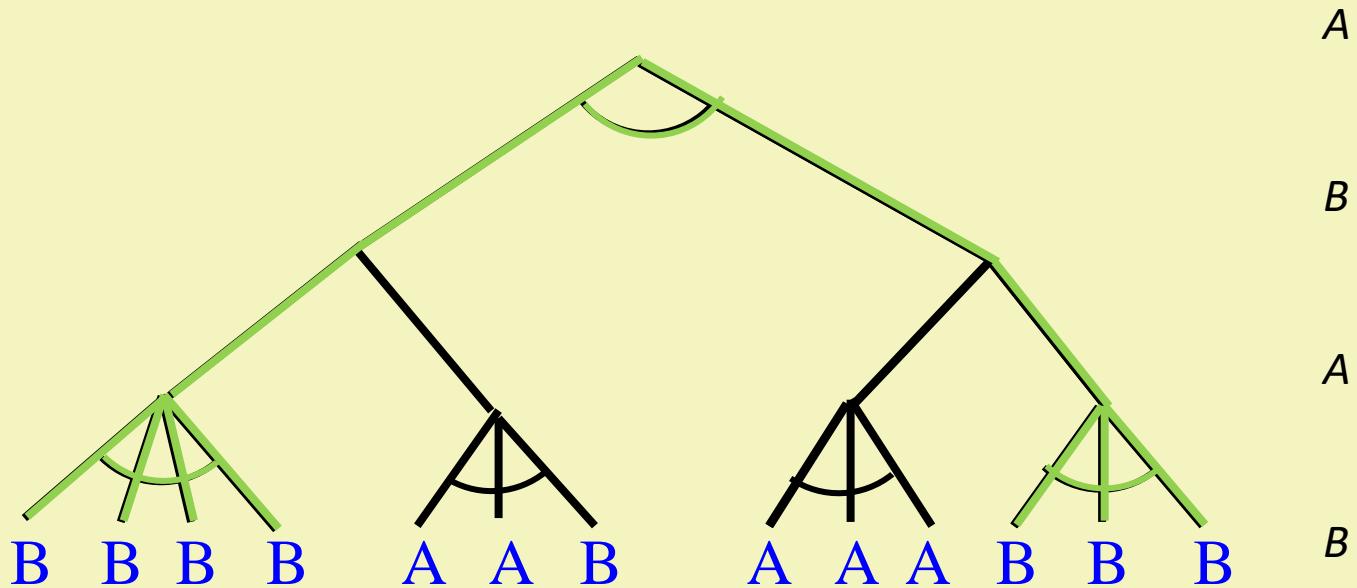
# *Nyerő stratégia*

- Egy játékos nyerő stratégiája egy olyan elv, amelyet betartva az ellenfél minden lépésére tud olyan választ adni, hogy megnyerje a játékot.
- A nyerő stratégia NEM egyetlen győztes játszma, hanem olyan győztes játszmák összessége, amelyek közül az egyiket biztos végig tudja játszani az a játékos, aki rendelkezik a nyerő stratégiával.
- Hasznos lehet a **nem-vesztő stratégia** megtalálása is, ha döntetlent is megengedő játéknál nincs győztes stratégia.
- Általános zéró összegű játékoknál beszélhetünk **adott hasznosságot biztosító stratégiáról**.

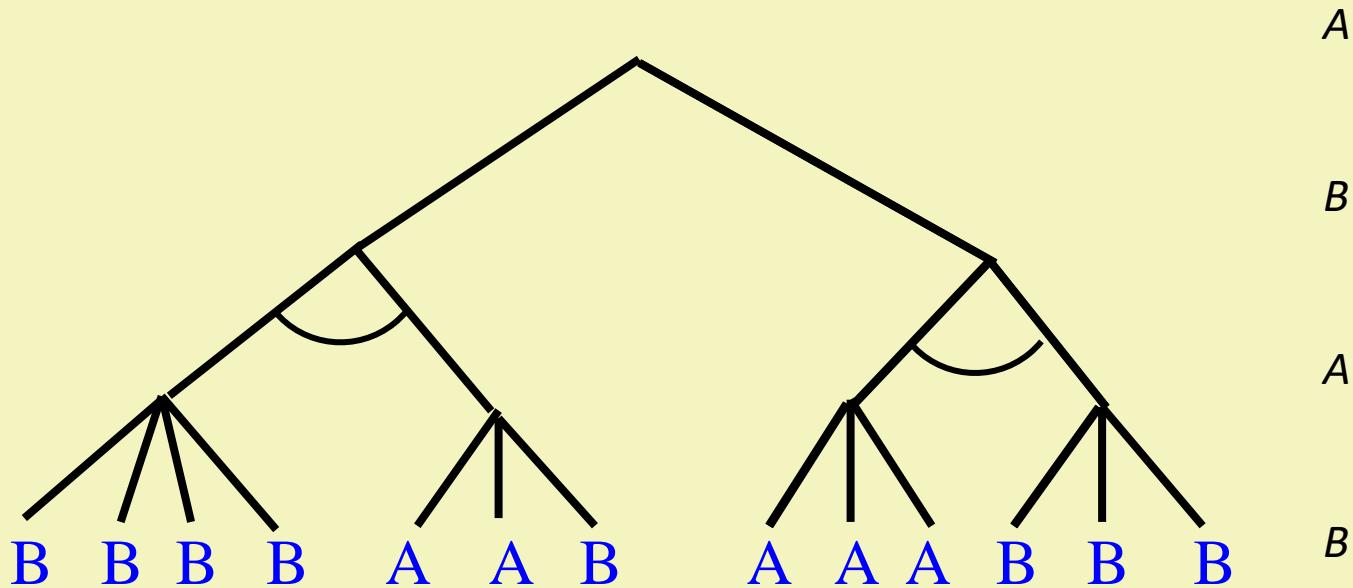
# *Megjegyzés*

- A játék az egyik játékos szempontjából egy ÉS/VAGY fával ábrázolható.
  - saját szinten egy csúcs utódai között VAGY kapcsolat van
  - ellenfél szintjén egy csúcs utódai között ÉS kapcsolat van
- A nyerő (nem-vesztő) stratégiát az ÉS/VAGY játékfa azon hiper-útja mutatja, amely a gyökér csúcsból csupa nyerő (nem-vesztő) levélcsúcsba vezet.
- A nyerő stratégia keresése tehát egy ÉS/VAGY fabeli hiper-út keresési probléma.

# *Nyerő stratégia keresése a **B** játékos ÉS/VAGY fájában*



# *Nyerő stratégia keresése az A játékos ÉS/VAGY fájában*

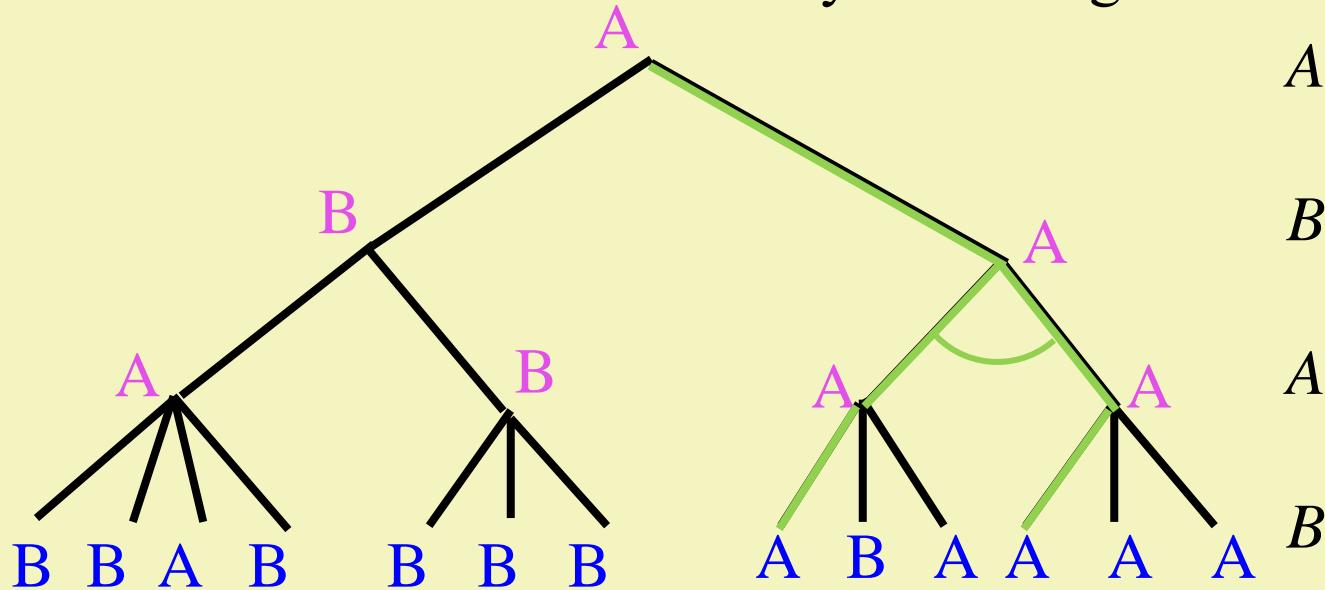


Nincs nyerő stratégia.

Csak az egyik játékosnak lehet nyerő stratégiája.

# Tétel

- A két esélyes (győzelem vagy vereség) teljes információjú véges determinisztikus kétszemélyes játékokban az egyik játékos számára biztosan létezik nyerő stratégia.



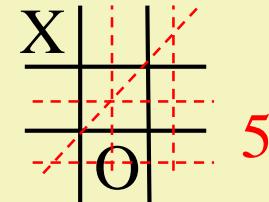
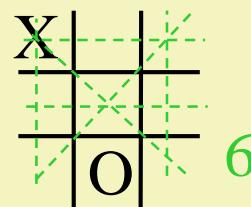
- A három esélyes játékokban (van döntetlen is) a nem vesztő stratégiát lehet biztosan garantálni.

# Részleges játékfa-kiértékelés

- ❑ A nyerő vagy nem-vesztő stratégia megkeresése egy nagyobb játékfa esetében **reménytelen**.
- ❑ Az optimális lépés helyett a **soron következő jó lépést** keressük.
  - Legyen a bennünket képviselő játékos neve mostantól MAX, az ellenfélén pedig MIN.
- ❑ Ehhez az aktuális állapotból indulva kell a játékfa
  1. **néhány szintjét felépíteni,**
  2. ezen a részfa leveleinek a **hasznosságát megbecsülni,**
  3. majd a soron **következő lépést meghatározni.**

# Kiértékelő függvény

- ❑ minden esetben szükségünk van egy olyan heurisztikára, amely a mi szempontunkból becsüli meg egy állás hasznosságát:  $f: \text{Állások} \rightarrow [-1000, 1000]$  függvény.
- ❑ Példák:
  - Sakk: (kiértékelő függvény a fehérnek)  
 $f(s) = (\text{fehér királynő száma}) - (\text{fekete királynő száma})$
  - Tic-tac-toe:  $f(s) = M(s) - O(s)$   
 $M(s) = \text{a saját lehetséges győztes vonalaink száma}$   
 $O(s) = \text{az ellenfél lehetséges győztes vonalaink száma}$

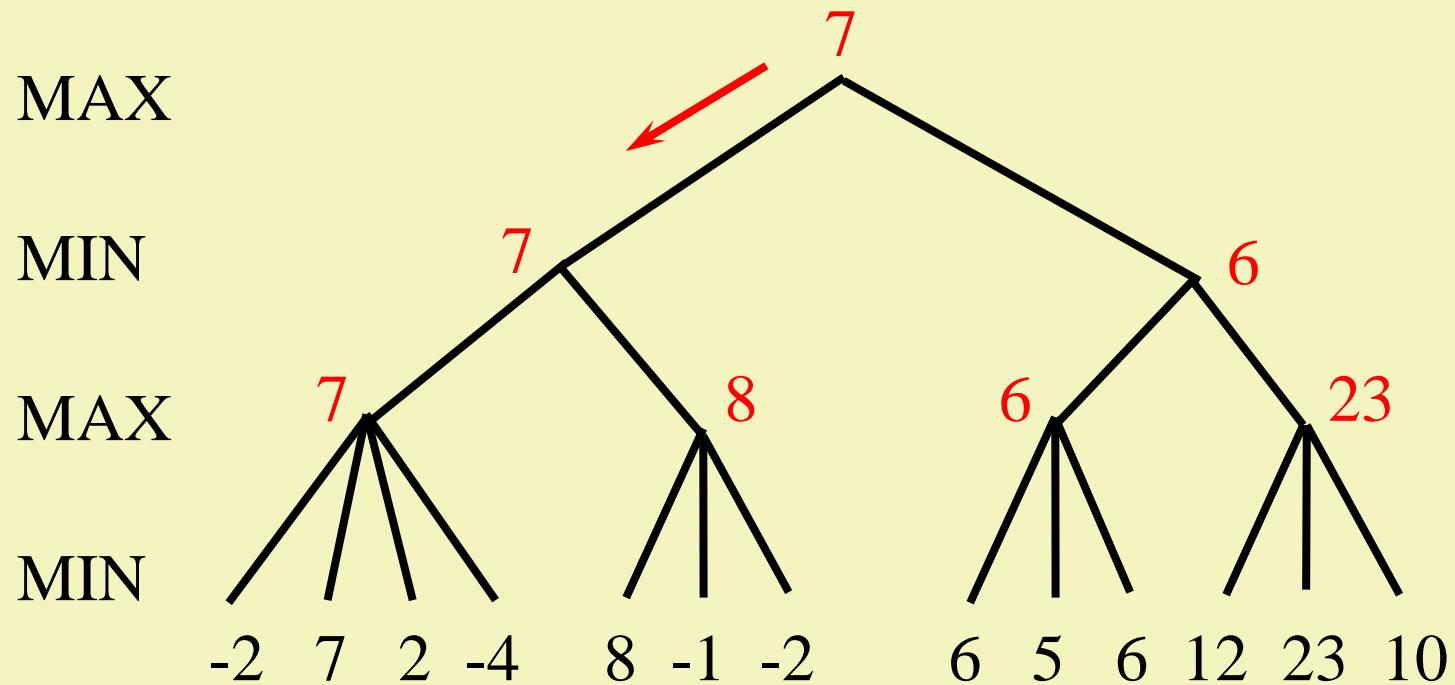


# *Minimax algoritmus*

1. A játékfának az adott állás csúcsából leágazó részfáját felépítjük néhány szintig.
2. A részfa leveleit kiértékeljük a kiértékelő függvény segítségével.
3. Az értékeket felfuttatjuk a fában:
  - A saját (MAX) szintek csúcsaihoz azok gyermekéinek maximumát:  $szülő := \max(gyerek_1, \dots, gyerek_k)$
  - Az ellenfél (MIN) csúcsaihoz azok gyermekéinek minimumát:  $szülő := \min(gyerek_1, \dots, gyerek_k)$
4. Soron következő lépések ahhoz az álláshoz vezet, ahonnán a gyökérhez felkerült a legnagyobb érték.

# Példa

Legyen a mi nevünk MAX, az ellenfélé MIN.

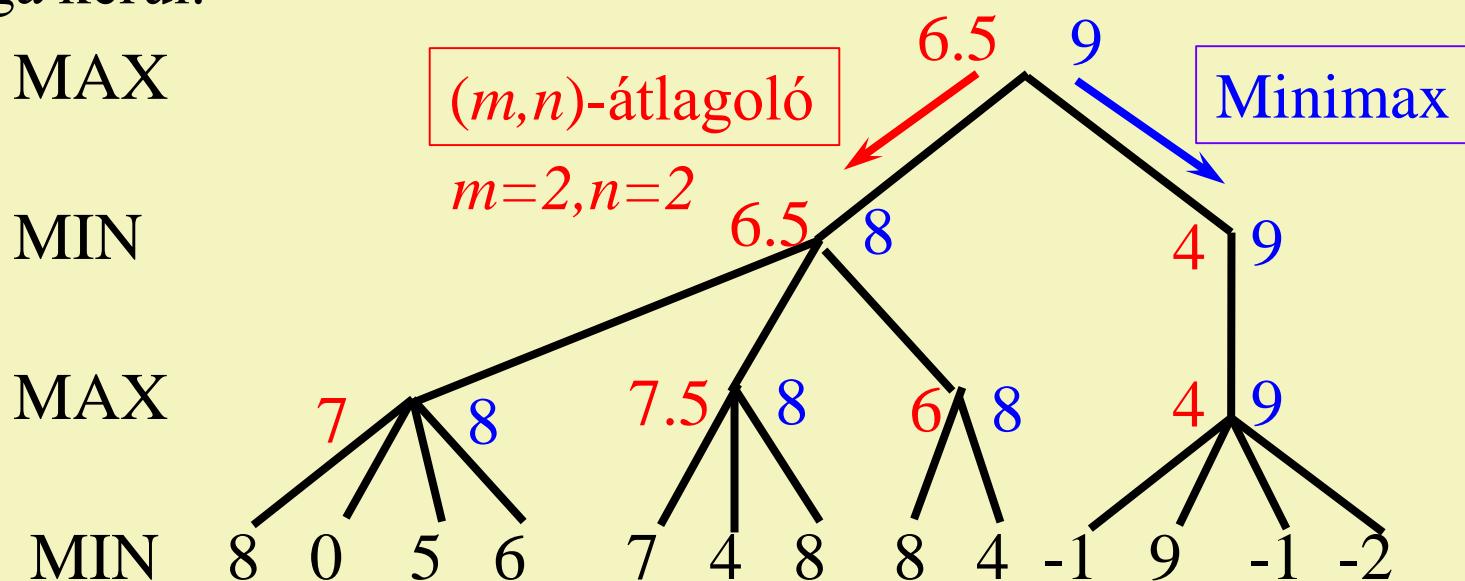


# *Megjegyzés*

- Az algoritmust minden alkalommal, valahányszor mi következünk, megismételjük, hiszen lehet, hogy az ellenfél nem az általunk várt legerősebb lépésekkel válaszol, mert:
  - eltérő mélységű részfával dolgozik,
  - más kiértékelő függvényt használ,
  - nem minimax eljárást alkalmaz,
  - hibázik.

# *Átlagoló kiértékelés*

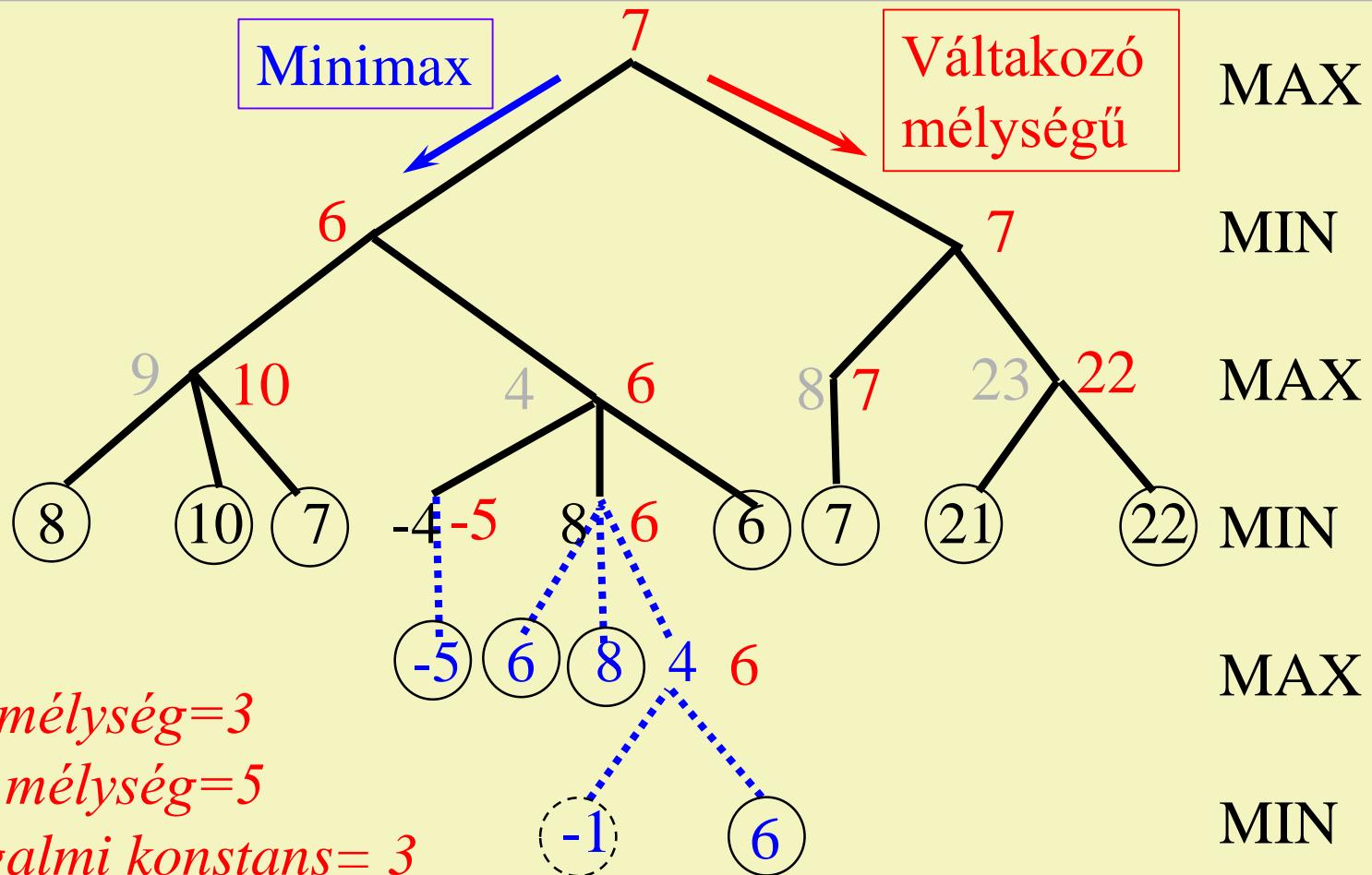
- Célja a kiértékelő függvény esetleges tévedéseinek simítása.
- MAX szintjeire az  $m$  darab legnagyobb értékű gyerek ( $\max_m$ ) átlaga, a MIN-re az  $n$  darab legkisebb értékű gyerek ( $\min_n$ ) átlaga kerül.



# Váltakozó mélységű kiértékelés

- Célja, hogy a kiértékelő függvény minden ágon reális értéket mutasson. Megtévesztő lehet egy csúcsnál ez az érték ha annak szülőjénél a kiértékelő függvény lényegesen eltérő értéket mutat: a játék ezen szakasza nincs nyugalomban.
- Egy adott szintig (**minimális mélység**) mindenkorban felépítjük a részfát,
- majd ettől a szinttől kezdve egy másik adott szintig (**maximális mélység**) csak azon csúcsok gyerekeit állítjuk elő, amelyek még nincsenek nyugalomban, amelyre nem teljesül a **nyugalmi teszt**:  $|f(\text{szülő}) - f(\text{csúcs})| < K$ ,

# Példa



# *Szelektív kiértékelés*

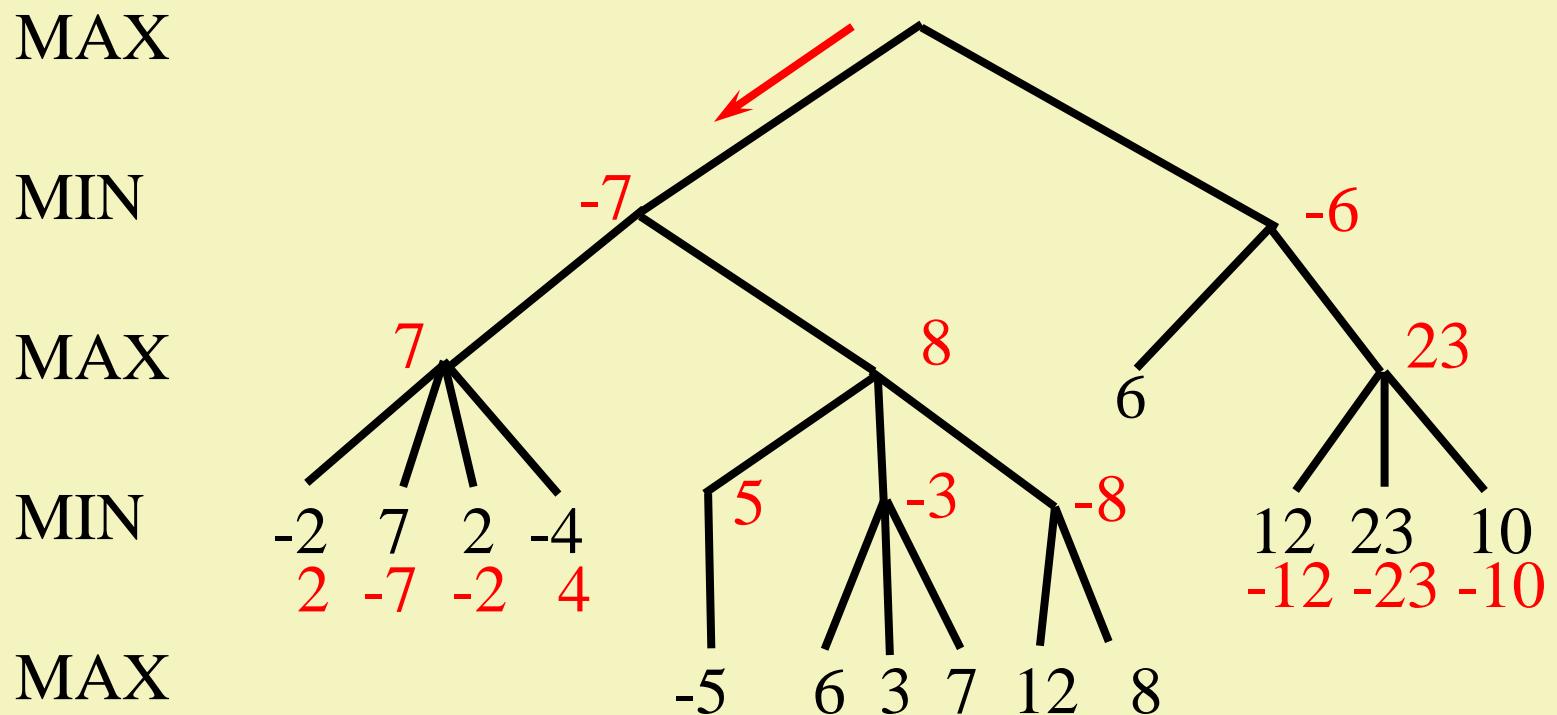
- ❑ Célja a **memória-igény** csökkentése.
- ❑ Elkülönítjük a lényeges és lényegtelen lépéseket, és csak a **lényeges lépéseknek** megfelelő részfát építjük fel.
- ❑ Ez a szétválasztás heurisztikus ismeretekre épül.

# *Negamax algoritmus*

- Negamax eljárást **könnyebb implementálni**.
  - Kezdetben  $(-1)$ -gyel szorozzuk azon levélcsúcsok értékeit, amelyek az ellenfél (MIN) szintjein vannak, majd
  - Az értékek felfuttatásánál minden szinten az alábbi módon számoljuk a belső csúcsok értékeit:

$$\text{szülő} := \max(-\text{gyerek}_1, \dots, -\text{gyerek}_k)$$

# Példa



# *Alfa-béta algoritmus*

- Visszalépéses algoritmus segítségével járjuk be a részfát (**olyan mélységi bejárás, amely mindenkor csak egy utat tárol**). Az aktuális úton fekvő csúcsok **ideiglenes értékei**:
  - a MAX szintjein  $\alpha$  érték: ennél rosszabb értékű állásba innen már nem juthatunk
  - A MIN szintjein  $\beta$  érték: ennél jobb értékű állásba onnan már nem juthatunk
- Lefelé haladva a fában  $\alpha := -\infty$ , és  $\beta := +\infty$ .
- Visszalépéskor az éppen elhagyott (gyermek) csúcs értéke (felhozott érték) módosíthatja a szülő csúcs értékét:
  - a MAX szintjein:  $\alpha := \max(\text{felhozott érték}, \alpha)$
  - a MIN szintjein:  $\beta := \min(\text{felhozott érték}, \beta)$
- Vágás: ha az úton van olyan  $\alpha$  és  $\beta$ , hogy  $\alpha \geq \beta$ .

# Példa

MAX  $\alpha =$

MIN  $\beta =$

MAX  $\alpha =$

MIN  $\beta =$

8 2

-2

7 8 4

-1  
2

2

2

2

2

2

2

4

4

# *Elemzés*

- ❑ Ugyanazt a kezdőlépést kapjuk eredményül, amit a minimax algoritmus talál. (Több egyforma kezdőirány esetén a „baloldalit” választjuk.)
- ❑ **Memória igény:** csak egy utat tárol.
- ❑ **Futási idő:** a vágások miatt sokkal jobb, mint a minimax módszeré.
  - Átlagos eset: egy csúcs alatt, két belőle kiinduló ág megvizsgálása után már vághatunk.
  - Optimális eset: egy  $d$  mélységű  $b$  elágazású fában kiértékelte levélcsúcsok száma:  $\sqrt{b^d}$
  - Jó eset: A részfa megfelelő rendezésével érhető el.

# *Kétszemélyes játékot játszó program*

- Váltakozó mélységű, szelektív,  $(m,n)$  átlagoló, negamax alfa-béta kiértékelést végez.
- Keretprogram, amely váltakozva fogadja a felhasználó lépésein, és generálja a számítógép lépésein.
- Kiegészítő funkciók (beállítások, útmutató, segítség, korábbi lépések tárolása, mentés stb.)
- Felhasználói felület, grafika
- Heurisztika megválasztása (kiértékelő függvény, szelekció, kiértékelés sorrendje)



# Evolúciós algoritmusok

# *Evolúció, mint kereső rendszer*

- A problémáról egyszerre több **egyedét** (a problémára adható lehetséges válaszokat) tároljuk az ún. **populációban**.
- Egy többnyire véletlen populációból indulunk ki, amelyet **lépésről lépésre javítjuk** azért, hogy megjelenjen benne egy célegyed vagy egy összességében jó populációhoz jussunk.
- Az egyedekeket egy ún. **rátermettségi függvény** segítségével hasonlítjuk össze. minden lépésben a kevésbé rátermett egyedek egy részét a rátermettebbekhez hasonló egyedekre cseréljük le. Ez a változtatás visszavonhatatlan. Ez tehát egy **nem-módosítható stratégiájú keresés**.

# *Evolúciós operátorok és a terminálási feltétel*

- **Szelekció:** Kiválasztunk néhány (lehetőleg rátermett) egyedet szülőnek.
- **Rekombináció (keresztezés):** Szülőkből utódok készülnek úgy, hogy a szülők tulajdonságait örököljék az utódok.
- **Mutáció:** Az utódok tulajdonságait kismértékben módosítjuk.
- **Visszahelyezés:** Új populációt alakítunk ki az utódokból és a régi populációból.
- **Terminálási feltétel:**
  - ha a célegyed megjelenik a populációban
  - ha a populáció egyesített rátermettségi függvény értéke egy ideje nem változik.

ADAT := kezdeti érték

**while**  $\neg$  terminálási feltétel(ADAT) **loop**

    SELECT SZ FROM alkalmazható szabályok

    ADAT := SZ(ADAT)

**endloop**

## Evolúció alapalgoritmusa

### *Procedure EA*

*populáció* := kezdeti populáció

**while** terminálási feltétel nem igaz **loop**

    szülők := szelekció(*populáció*)

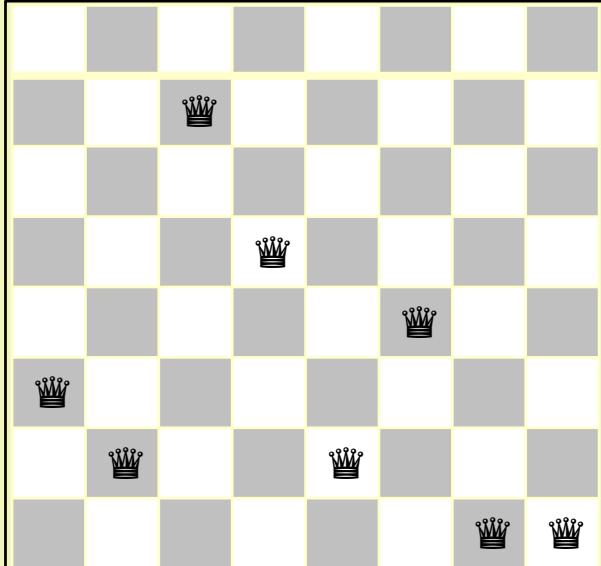
    utódok := rekombináció( szülők )

    utódok := mutáció(utódok)

*populáció* := visszahelyezés(*populáció*, utódok)

**endloop**

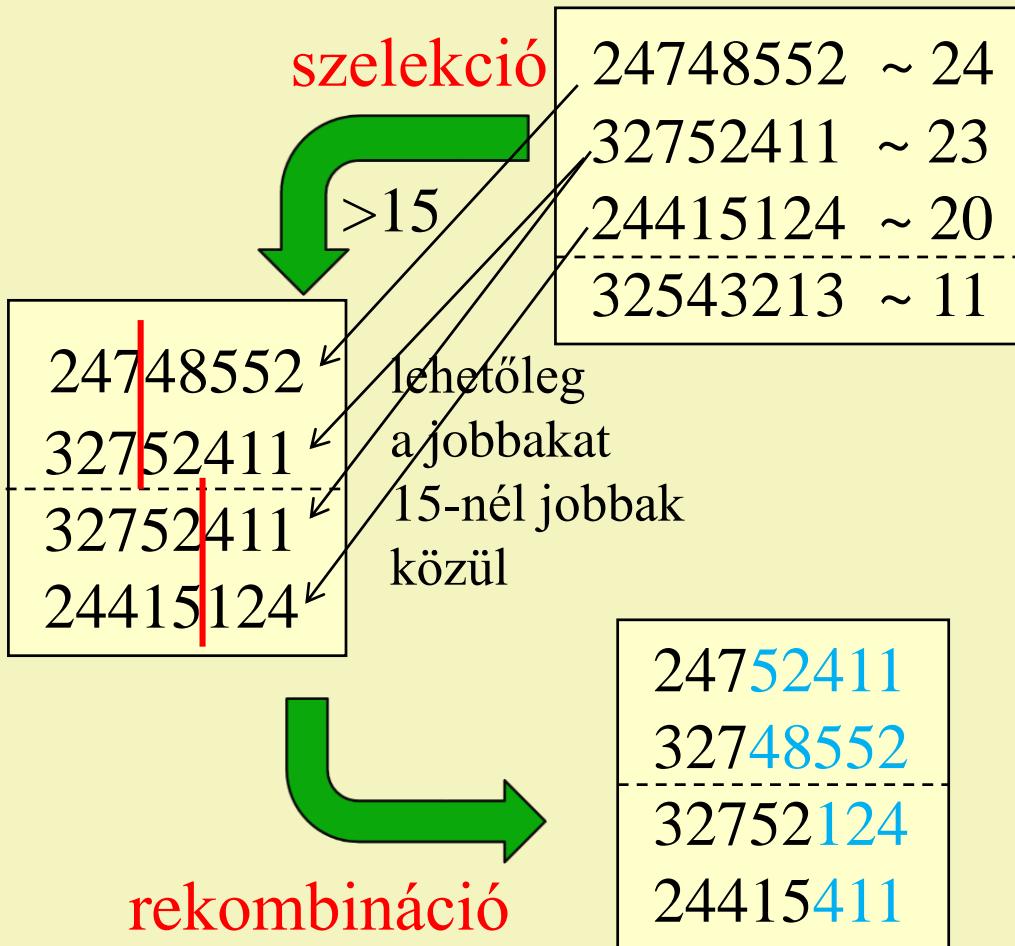
# *n-királynő* probléma 1.



rátermettségi érték: 23

- Egyed: a királynők olyan elrendezése, ahol minden oszlop pontosan egy királynőt tartalmaz
- Reprezentáció: oszloponként a királynők sorpozíciót tartalmazó sorozat
- Rátermettségi függvény: ütésekben nem levő királynő párok száma

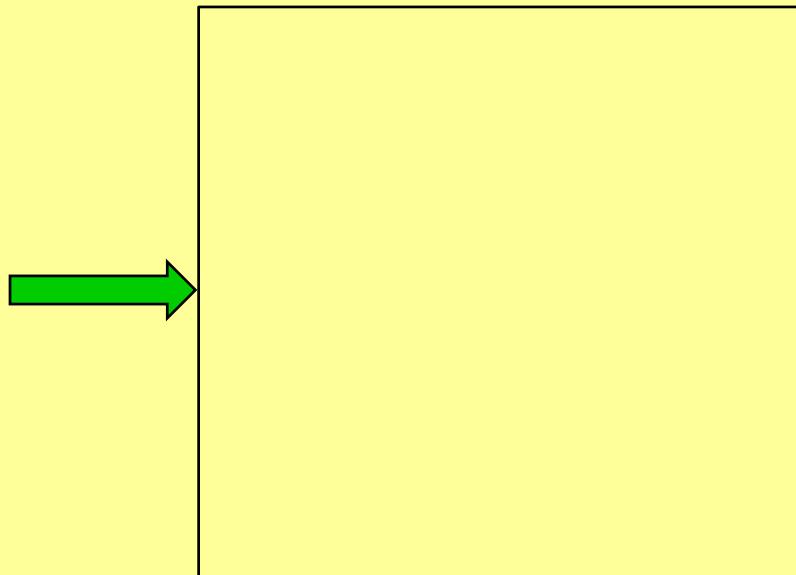
# Evolúciós ciklus



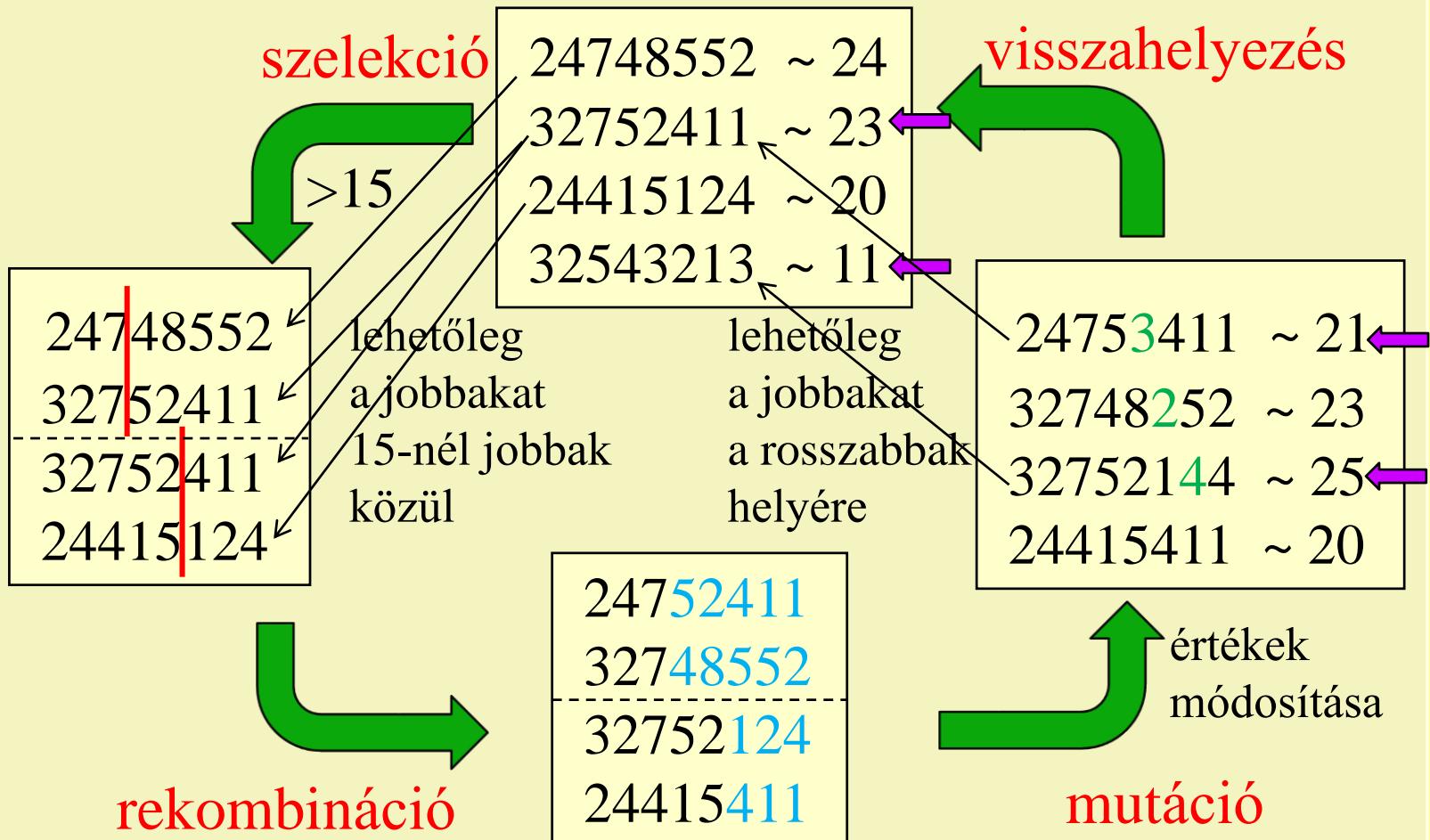
# Keresztezés

2	4	7	4	8	5	5	2
---	---	---	---	---	---	---	---

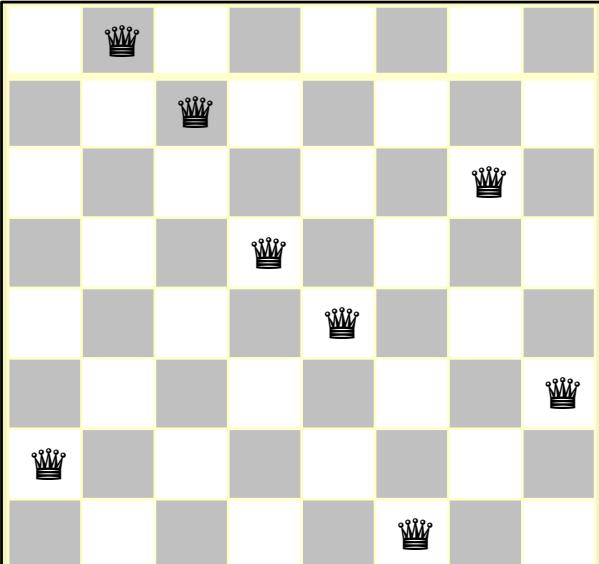
3	2	7	5	2	4	1	1
---	---	---	---	---	---	---	---



# Evolúciós ciklus



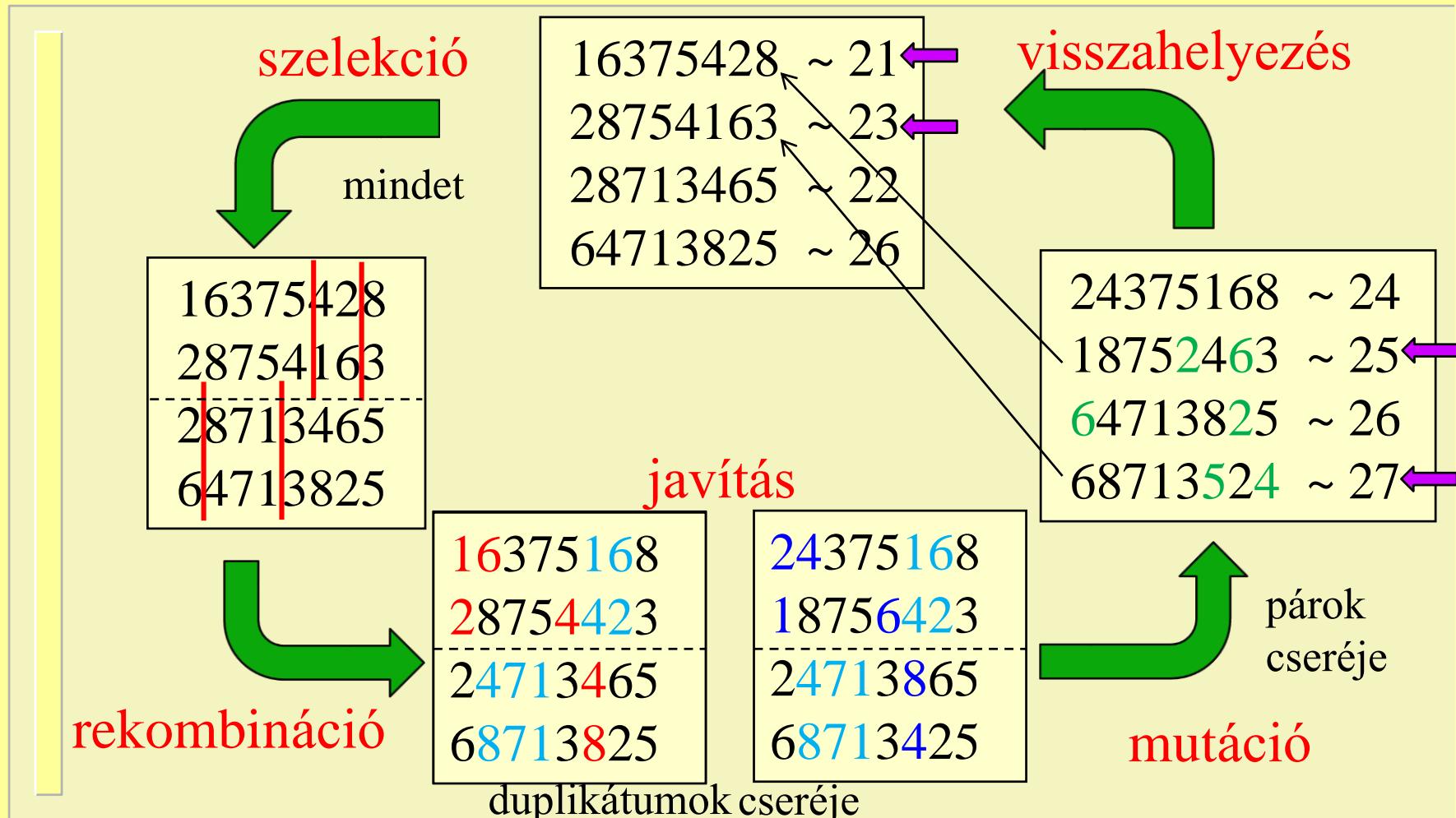
# *n-királynő* probléma 2.



rátermettségi érték: 23

- Egyed: a királynők olyan elrendezése, ahol minden sor és oszlop pontosan egy királynőt tartalmaz
- Reprezentáció: oszloponként a királynők sorpozíciót tartalmazó permutáció  
Rátermettségi függvény: ütésekben nem levő királynő párok száma

# Evolúciós ciklus



# Kielégíthetőségi probléma (SAT)

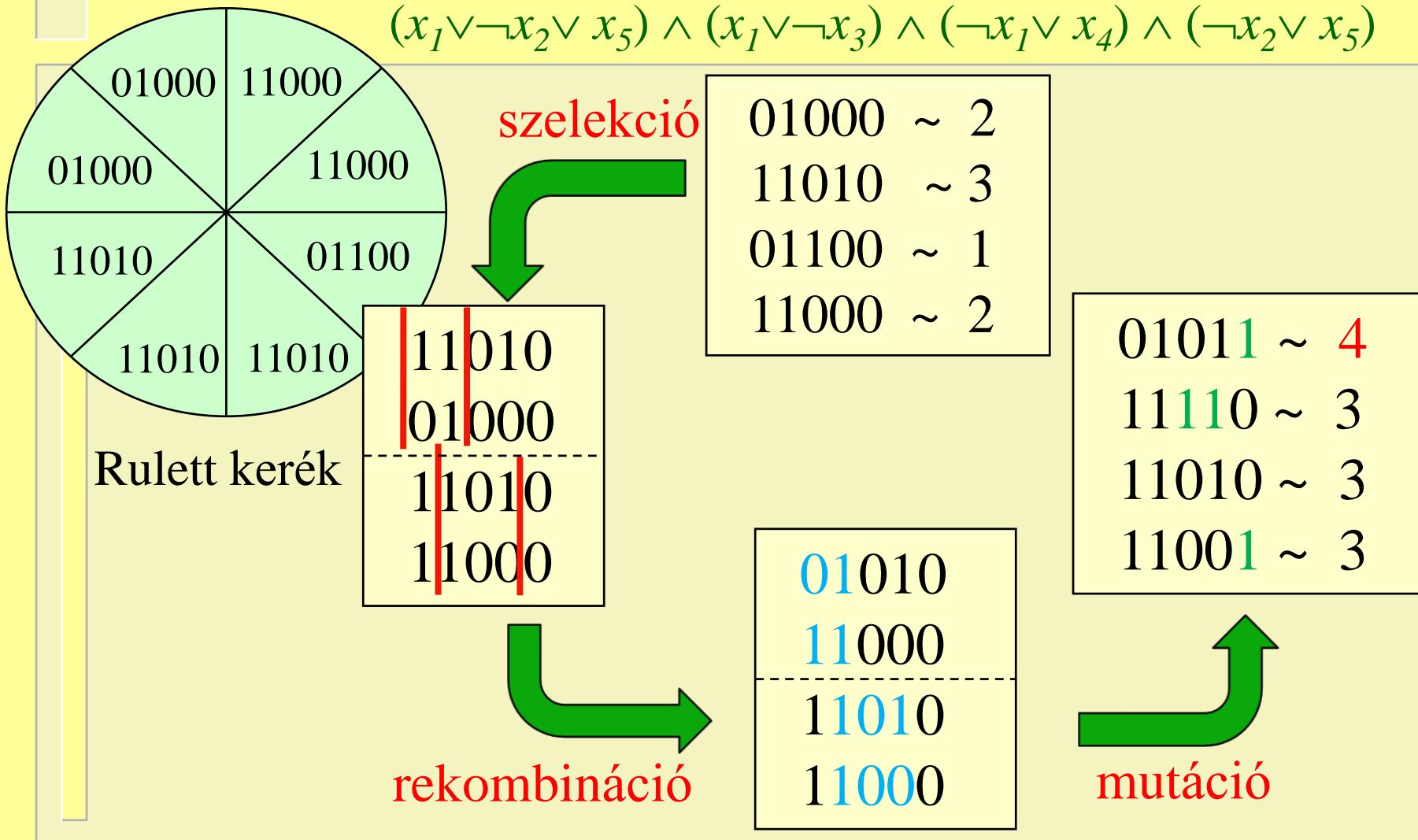
*Adott egy  $n$  változós Boolean formula KNF alakban. A változók milyen igazság kiértékelése mellett lesz formula igaz?*

E.g.:  $(x_1 \vee \neg x_2 \vee x_5) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_4) \wedge (\neg x_2 \vee x_5)$

egy megoldás:  $x_1 = \text{true}, x_2 = \text{false}, x_3 = \text{false}, x_4 = \text{true}, x_5 = \text{true}$

- Egyed: egy lehetséges igazság kiértékelés
  - Reprezentáció: logikai érték (bitek) sorozata
  - Rátermettségi függvény: Az adott formula igazra értékelt klózainak száma

# Evolúciós ciklus

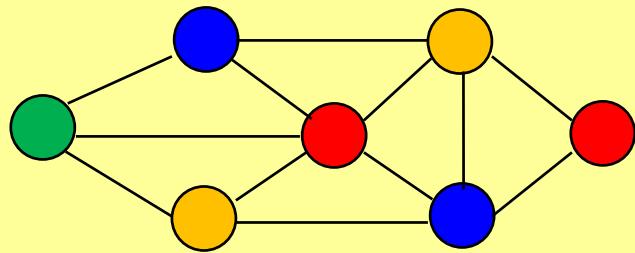


# *Evolúciós algoritmus tervezése*

- problématér egyedeinek reprezentációja: kódolás
- rátermettségi függvény (fitnesz függvény)
  - kapcsolat a kódolással és a céllal
- evolúciós operátorok
  - szelekció, rekombináció, mutáció, visszahelyezés
- kezdő populáció, megállási feltétel (cél)
- stratégiai paraméterek
  - populáció mérete, mutáció valószínűsége, utódképzési ráta, visszahelyezési ráta, stb.

# Kódolás

- Egy egyedet egy **jelsorozattal** (kromoszómával) kódolunk. A jelsorozatnak ki kell elégítenie a **kód-invariánst**.
- Az egyedekeket az őket reprezentáló kódjukon keresztül változtatjuk meg. Egy jel vagy jelcsoport, azaz a gén írja le az egyed egy tulajdonságát (attribútum-érték pájját).
  - Sokszor egy génnel a kódsorozatban elfoglalt pozíciója (lókusza) jelöli ki a gén által leírt attribútumot, amelynek értéke maga a gén (allél). A kód ekkor tulajdonságoknál **feldarabolható**: egy rövid kódszakasz megváltoztatása kis mértékben változtat az egyeden.
- Gyakori megoldások:
  - **Vektor**: valós vagy egész számok rögzített hosszú tömbje
  - **Bináris kód**: bitek rögzített hosszú tömbje
  - Véges sok elem **permutációja**



## Gráf színezési probléma

Adott egy véges egyszerű gráf, amelynek a csúcsait négy szín felhasználásával kell úgy kiszínezni, hogy a szomszédos csúcsok eltérő színűek legyenek.

*Direkt kódolás*



1. 2. 3. 4. 5. 6. 7.

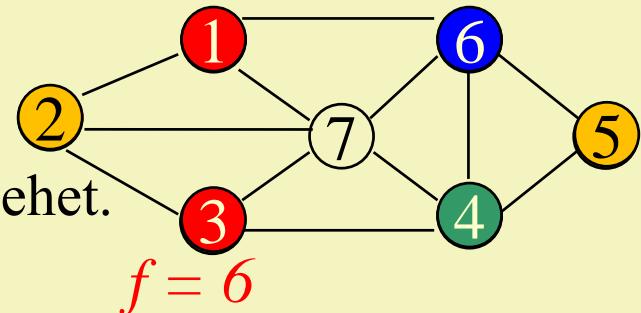
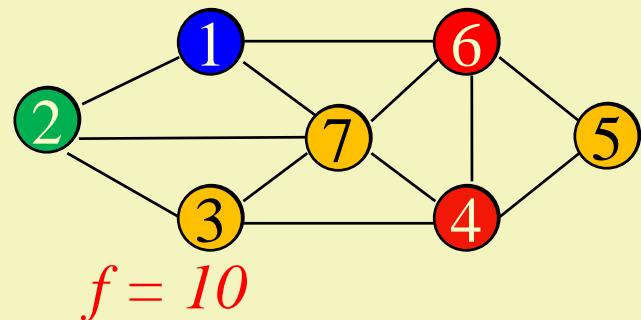
Az  $x[i]$  az  $i$ -dik csúcs színe.

$f$  a jó élek száma.

*Indirekt kódolás*



Az  $i$ -dik lépésben az  $x[i]$ -dik csúcsot színezzük ki a lehető legvilágosabb színnel a szomszédjaihoz igazodva, ha lehet.  
 $f$  a kiszínezett csúcsok száma



# *A kő-papír-olló játék stratégiája*

Alakítsunk ki jó stratégiát egy kő-papír-olló világbajnokságra!

- ❑ Olyan függvényre van szükségünk, amelyik a korábbi csaták kimenetele alapján javaslatot tesz a soron következő lépéinkre.
  - Például két korábbi csata alapján:

<i>Előzmény:</i>	<i>Én:</i>	<b>K</b>	<b>P</b>	<i>Javaslat:</i>	<b>K</b>
	<i>Ő:</i>	<b>O</b>	<b>O</b>		
  - Ez még nem a teljes stratégia, mert nem csak a fenti előzményre, hanem az összes lehetséges előzményre kell soron következő lépést javasolni.

# Kódolás

Egy stratégia (egyed) kódja:  $\{0,1,2\}^{0..80}$   
Az összes lehetséges stratégia száma:  $3^{81}$

<i>Jelek</i>	<i>Előzmény (ÉnÖÉnÖ)</i>	<i>Válasz</i>
K ~ 0	KKKK ~ 0000 ~ 0	P ~ 1
P ~ 1	KKKP ~ 0001 ~ 1	O ~ 2
O ~ 2	KKKO ~ 0002 ~ 2	K ~ 0
	KKPK ~ 0010 ~ 3	P ~ 1
	...	...
	OOOP ~ 2221 ~ 79	O ~ 2
	OOOO ~ 2222 ~ 80	K ~ 0

*A stratégia:* 1201 ... 20

# Rátermettség kiértékelése

Stratégia: 1 2 0 1 ... 2 0

Minta:

Játékos: 0 0 0 2 2 2 1 2 2 2 2 0 0 1 0 0 0

Ellenfél: 0 1 0 2 1 1 2 2 2 0 1 0 1 0 1 1

Jelek

K ~ 0

P ~ 1

O ~ 2

Eset → Javaslat

Ellenfél

Érték

0 0 0 1 → 2 0 vereség -1

0 0 0 1 → 2 1 győzelem +1

0 1 0 0 → 1 1 döntetlen 0

...

2 2 2 1 → 2 1 győzelem +1

2 2 2 2 → 0 0 döntetlen 0

# Szelekció

- **Célja:** a rátermett egyedek kiválasztása úgy, hogy a rosszabbak kiválasztása is kapjon esélyt.
  - **Rátermettség arányos** (rulett kerék algoritmus): minél jobb a rátermettségi függvényértéke egy elemnek, annál nagyobb valószínűsséggel választja ki
  - **Rangsorolásos**: rátermettség alapján sorba rendezett egyedek közül a kisebb sorszámuakat nagyobb valószínűsséggel választja ki
  - **Versengő**: véletlenül kiválasztott egyedcsoporthok (pl. párok) legjobb egyedét választja ki.
  - **Csonkolásos v. selejtezős**: a rátermettség szerint legjobb (adott küszöbérték feletti) valahány egyedből véletlenszerűen választ néhányat.

# *Rekombináció*

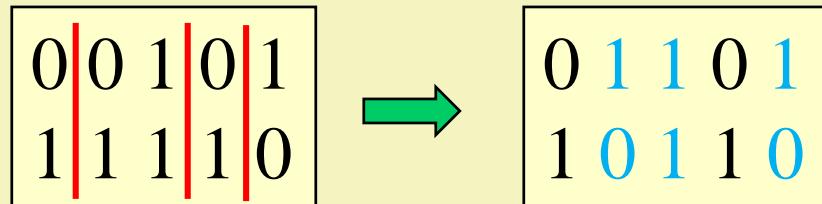
- A feladata az, hogy adott szülő-egyedekből olyan utódokat hozzon létre, amelyek a szüleik tulajdonságait "öröklik".
  - **Keresztezés**: véletlen kiválasztott pozíción jelcsoportok (gének) vagy jelek cseréje
  - **Rekombináció**: a szülő egyedek megfelelő jeleinek kombinálásával kapjuk az utód megfelelő jelét

Ügyelni kell a kód-invariáns megtartására: vizsgálni kell, hogy az új kód értelmes lesz-e (permutáció)

# Keresztezés

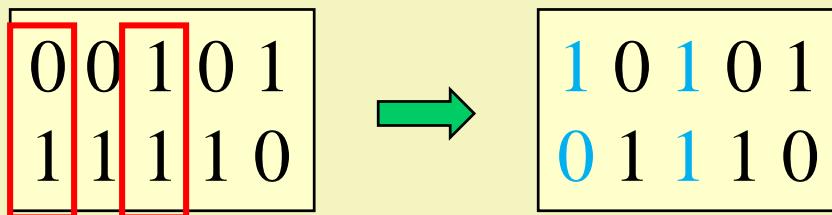
## ❑ Egy- illetve többpontos keresztezés

- Kódszakaszokat cserélünk



## ❑ Egyenletes keresztezés

- Jeleket cserélünk



# Permutációk keresztezése 1.

## □ Parciálisan illesztett keresztezés

- Egy szakasz cseréje után párba állítja és kicseréli azokat a szakaszon kívüli elemeket, amelyek megsértik a permutáció tulajdonságát.

2	3	1	5	4	6	7
1	7	4	2	5	3	6

The diagram shows the process of finding duplicates in a sequence and pairing them. A green arrow points from the first table to the second. The second table contains the same sequence as the first, but with red arrows indicating pairs of elements: (2, 7), (4, 2), (4, 6), and (7, 3). These pairs represent the 'duplicates' found in the original sequence.

2	7	4	2	4	6	7
1	3	1	5	5	3	6

duplikátumok keresése  
és párba állítása

1	7	4	2	5	6	3
2	3	1	5	4	7	6

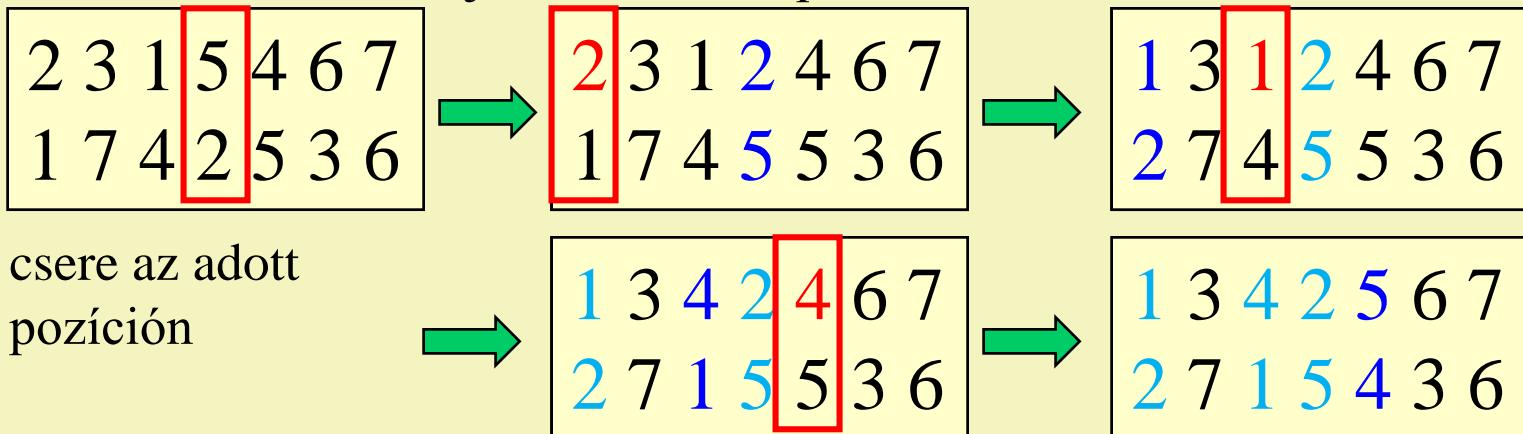
duplikátumpárok  
cseréje

# Permutációk keresztezése 2.

## □ Ciklikus keresztezés

1. Választ egy véletlen  $i \in [1..length]$ -t
2.  $a_i \leftrightarrow b_i$
3. Keres olyan  $j \in [1..length]$ -t ( $j \neq i$ ), amelyre  $a_j = a_i$ ,
4. Ha nem talál, akkor vége, különben  $i := j$
5. goto 2.

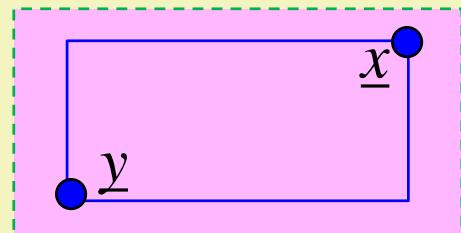
duplikátum keresése a felső utódban,  
majd csere azon a pozíción is



# Rekombináció vektorokra

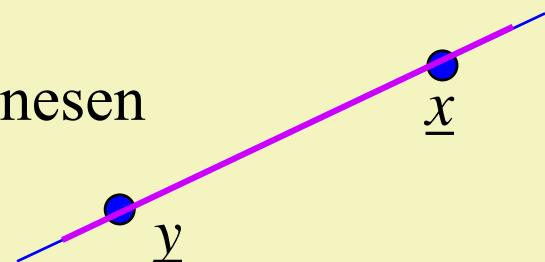
## ❑ Köztes rekombináció

- A szülők ( $\underline{x}$ ,  $\underline{y}$ ) által kifeszített hipertéglakörnyezetében lesz az utód ( $\underline{u}$ ).
- $\forall i=1 \dots n : u_i = a_i x_i + (1-a_i) y_i \quad a_i \in [-h, 1+h]$  véletlen



## ❑ Lineáris rekombináció

- A szülők ( $\underline{x}$ ,  $\underline{y}$ ) által kifeszített egyenesen a szülők környezetében vagy a szülők között lesz az utód ( $\underline{u}$ ).
- $\forall i=1 \dots n : u_i = a x_i + (1-a) y_i \quad a \in [-h, 1+h]$  véletlen



# Mutáció

- A mutáció egy egyed (utód) kis mértékű véletlen változtatását végzi.
- Valós tömbbel való kódolásnál kis  $p$  valószínűsséggel:
  - $\forall i=1 \dots n : z_i = x_i \pm domain_i \cdot p$
- Bináris tömbbel való kódolásnál kis  $p$  valószínűsséggel:
  - $\forall i=1 \dots n : z_i = 1 - x_i$  if  $random[0..1] < p$
- Permutáció esetén
  - egy jelpár cseréje
  - egy kódszakaszban a jelek ciklikus léptetése vagy megfordítása vagy átrendezése.

# Visszahelyezés

- A visszahelyezés a populációnak az utódokkal történő frissítése: Kiválasztja a populációnak a lecserélendő egyedeit, és azok helyére a kiválasztott utódokat teszi.

$$\text{utódképzési ráta (u)} = \frac{\text{utódok száma}}{\text{populáció száma}}$$

két szelekció is kell

$$\text{visszahelyezési ráta (v)} = \frac{\text{lecserélendő egyedek száma}}{\text{populáció száma}}$$

- ha  $u=v$ , akkor feltétlen cseréről van szó
  - további szelekció
- ha  $u < v$ , akkor egy utód több példánya is bekerülhet
  - további szelekció
- ha  $u > v$ , akkor az utódok közül szelektál

# Automatikus következtetés

# 1. Rezolúció

Feladat:

$A_1$ : Ha süt a nap, akkor Péter strandra megy.

$A_2$ : Ha Péter strandra megy, akkor úszik.

$A_3$ : Péternek nincs lehetősége otthon úszni.

Lássuk be, hogy ezekből következik:

$B$ : Ha süt a nap, akkor Péter nem marad otthon.

Formalizálás:

- |                        |     |         |                        |
|------------------------|-----|---------|------------------------|
| – süt a nap:           | $p$ | $A_1$ : | $p \rightarrow q$      |
| – Péter strandra megy: | $q$ | $A_2$ : | $q \rightarrow r$      |
| – Péter úszik:         | $r$ | $A_3$ : | $\neg(s \wedge r)$     |
| – Péter otthon marad:  | $s$ | $B$ :   | $p \rightarrow \neg s$ |

# Átalakítás

logikai következmény

□ Kell:  $p \rightarrow q, q \rightarrow r, \neg(s \wedge r) \Rightarrow p \rightarrow \neg s$

- minden olyan interpretáció (igazságértékelés), amely kielégíti a feltételeket, az kielégíti a következményt is.
- vagy: nincs olyan interpretáció (igazságértékelés), amely a feltételeket is, és következmény negáltját is kielégítené.
- azaz:  $(p \rightarrow q) \wedge (q \rightarrow r) \wedge \neg(s \wedge r) \wedge \neg(p \rightarrow \neg s)$  kielégíthetetlen  
vagy:  $(\neg p \vee q) \wedge (\neg q \vee r) \wedge (\neg s \vee \neg r) \wedge p \wedge s$  kielégíthetetlen

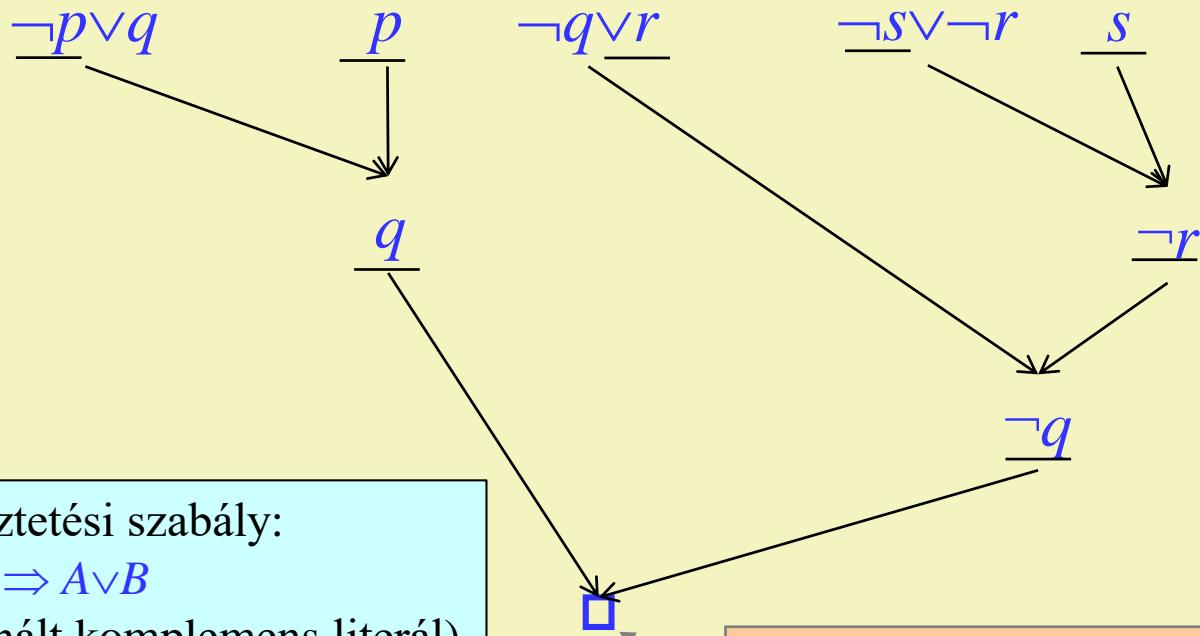
KNF: klózok között ‘és’ művelet  
klóz: literálok között ‘vagy’ művelet  
literál: ítéletváltozó vagy annak negáltja

- Tehát meg kell mutatnunk, hogy bármelyik interpretációval (igazságértékeléssel) legalább az egyik klóz *hamis* lesz.

# *Rezolúció = indirekt bizonyítás*

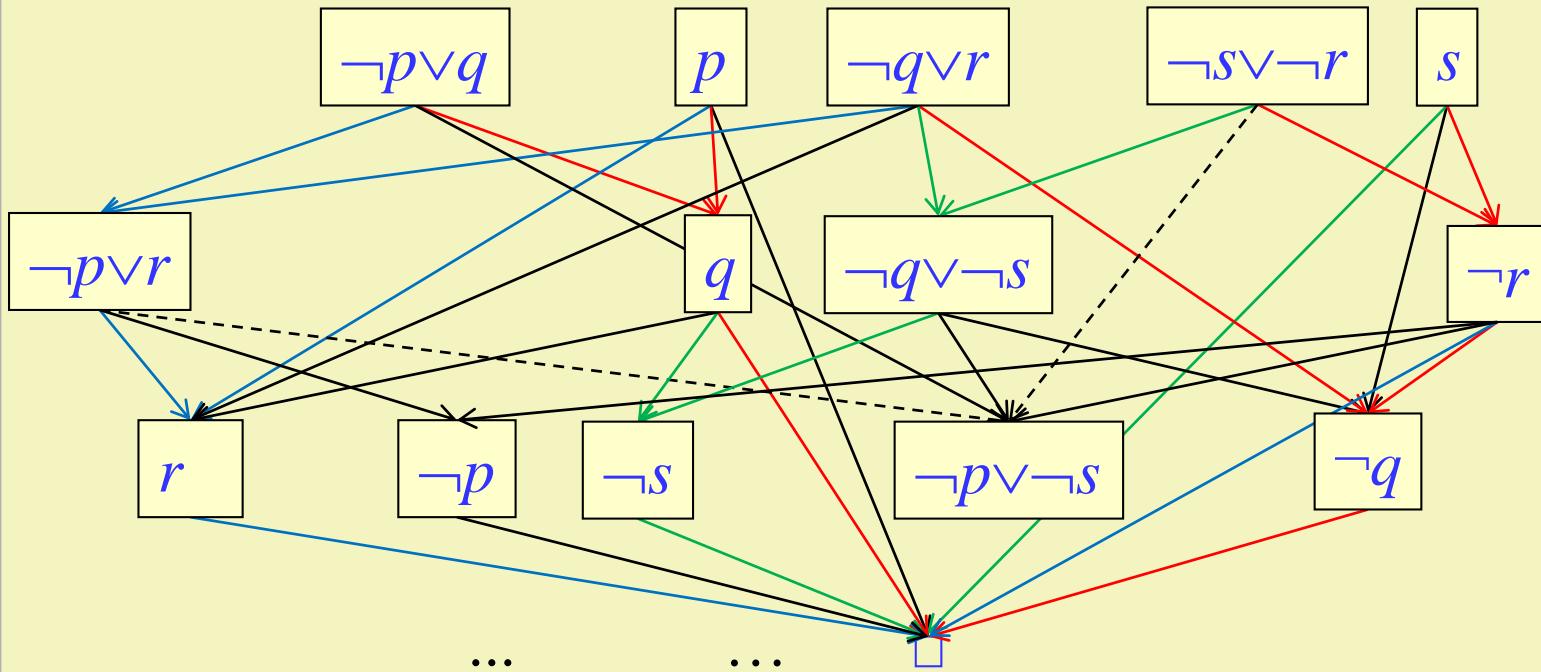
- Tekintsük a klózok halmazát és tegyük fel indirekt módon, hogy van olyan interpretáció, amikor **mindegyik klóz igaz**.
- Ekkor például  $p$  is, és  $\neg p \vee q$  is *igaz*. Ha azonban  $p$  *igaz*, akkor  $\neg p$  *hamis*, és ekkor a  $\neg p \vee q$  csak úgy lehet *igaz*, ha a  $q$  is *igaz*.
- A  $q$  – amely ugyancsak egy klóz – tehát akárcsak a többi klóz *igaz* az indirekt feltevés szerinti interpretációban. Vegyük hát hozzá az eddigi klózhalmazhoz.
- Az előbbihez hasonló módon bővítsük tovább a klózhalmazt addig, amíg ellentmondáshoz nem jutunk. (Például egyszerre megjelenik a klózhalmazban a  $q$  és  $\neg q$ .)

# Rezolúciós eljárás



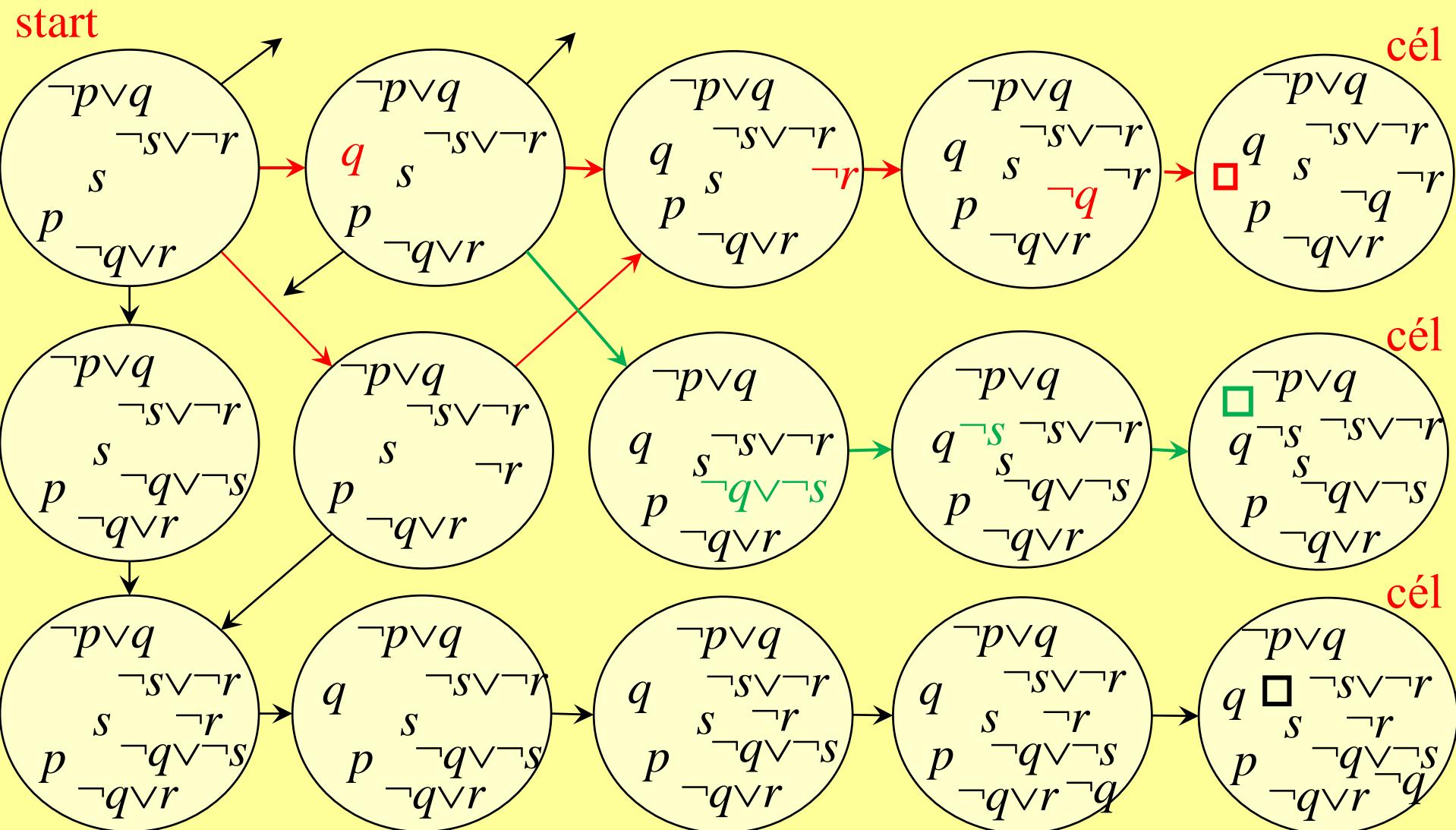
Tehát ha süt a nap, akkor Péter nem marad otthon.

# Cáfolati-, rezolúciós gráf



- Cáfolati gráf: az üres klózt előállítását bemutató gráf
- Rezolúciós gráf: az összes klóz előállítását mutató gráf

# Reprezentációs gráf



# *A reprezentációs gráf tulajdonságai*

- Egy csúcs egyetlen klózzal tartalmaz többet a szülőcsúcsánál: olyannal, amelyik a szülő csúcs két klózából vezethető le.
  - Mindegyik csúcs tartalmazza a kiinduló klózokat.
  - Nincsenek körök.
  - Az a rezolúciós lépés, amely egy csúcsban elvégezhető, az annak azon gyerek csúcsában is elvégezhető, amelyhez egy másik rezolúciós lépés árán jutottunk el.
- Ha van cáfolat, akkor minden csúcsból el lehet jutni egy üres klózt is tartalmazó célcímsúcsba.
  - Nincs rossz döntés, legfeljebb csak felesleges.
- Ítéletkalkulusban a gráfnak csak véges sok különböző csúcsa lehet, predikátumkalkulusban lehet végtelen sok is.

# Példa: Kuruzslók-e a doktorok?

$A_1$ : Van olyan páciens, aki minden doktorban megbízik.

$A_2$ : A kuruzslókban egyetlen páciens sem bízik meg.

Lássuk be, hogy

$B$  : Egyetlen doktor sem kuruzsló.

Formalizálás:

$P(x)$ :  $x$  egy páciens

$A_1 : \exists x \{ P(x) \wedge \forall y [D(y) \rightarrow M(x,y)] \}$

$D(y)$ :  $y$  egy doktor

$A_2 : \forall x \{ P(x) \rightarrow \forall y [K(y) \rightarrow \neg M(x,y)] \}$

$K(y)$ :  $y$  egy kuruzsló

$B : \forall x [D(x) \rightarrow \neg K(x)]$

$M(x,y)$ :  $x$  megbízik az  $y$ -ban

Kell:  $\exists x \{ P(x) \wedge \forall y [D(y) \rightarrow M(x,y)] \} \wedge \forall x \{ P(x) \rightarrow \forall y [K(y) \rightarrow \neg M(x,y)] \}$

$\wedge \neg \forall x [D(x) \rightarrow \neg K(x)]$  kielégíthetetlen

# *Formulák klóz-formára (SKNF) hozása*

1. Kiküszöböljük az  $\leftrightarrow$  és a  $\rightarrow$  műveleti jeleket (logikai törvények).
2. Redukáljuk a negációk hatáskörét (DeMorgan azonosságok).
3. Standardizáljuk a változókat (kvantoronkénti átnevezés).
4. Egzisztenciális kvantorok kiküszöbölése. (Skolemizálás:  
 $\forall x_1 \dots \forall x_n \exists z F(\dots, z, \dots)$  helyett  $\forall x_1 \dots \forall x_n F(\dots, f(x_1, \dots, x_n), \dots)$   
– nem ekvivalens átalakítás, de kielégíthetőség tartó)
5. Univerzális kvantorok kiemelése a formula elejére a sorrendjük megtartásával. (prenex normál forma)
6. A formula többi részét konjunktív normálformára alakítjuk (kommutatív, asszociatív, disztributív törvények).
7. Kialakítjuk a klózokat (a kvantorokat, és a konjunkciós műveleti jeleket elhagyjuk, a változókat klózonként egyedivé nevezzük át.)

# Skolemizált konjuktív normálforma (SKNF)

$$A_1: \exists x \{ P(x) \wedge \forall y [D(y) \rightarrow T(x,y)] \} = \exists x \{ P(x) \wedge \forall y [\neg D(y) \vee T(x,y)] \} \approx \\ \approx P(\textcolor{red}{a}) \wedge \forall y [\neg D(y) \vee T(\textcolor{red}{a},y)] = \forall y \{ P(a) \wedge [\neg D(y) \vee T(a,y)] \}$$

$P(a)$  ,  $\neg D(y) \vee T(a,y)$

$a \rightarrow b$  helyett  $\neg a \vee b$

Skolemizálás  
 $\textcolor{red}{a}$  a Skolem konstans

$$A_2 : \forall x \{ P(x) \rightarrow \forall y [Q(y) \rightarrow \neg T(x,y)] \} = \\ = \forall x \{ \neg P(x) \vee \forall y [\neg Q(y) \vee \neg T(x,y)] \} =$$

$a \rightarrow b$  helyett  $\neg a \vee b$

$$= \forall x \forall u \{ \neg P(x) \vee \neg Q(u) \vee \neg T(x,u) \}$$

változó átnevezés

$$\neg P(x) \vee \neg Q(u) \vee \neg T(x,u)$$

$a \rightarrow b$  helyett  $\neg a \vee b$

$$B: \neg \forall x [ D(x) \rightarrow \neg Q(x) ] = \neg \forall x [ \neg D(x) \vee \neg \neg Q(x) ] = \\ = \exists x [ D(x) \wedge Q(x) ] \approx D(\textcolor{red}{b}) \wedge Q(\textcolor{red}{b})$$

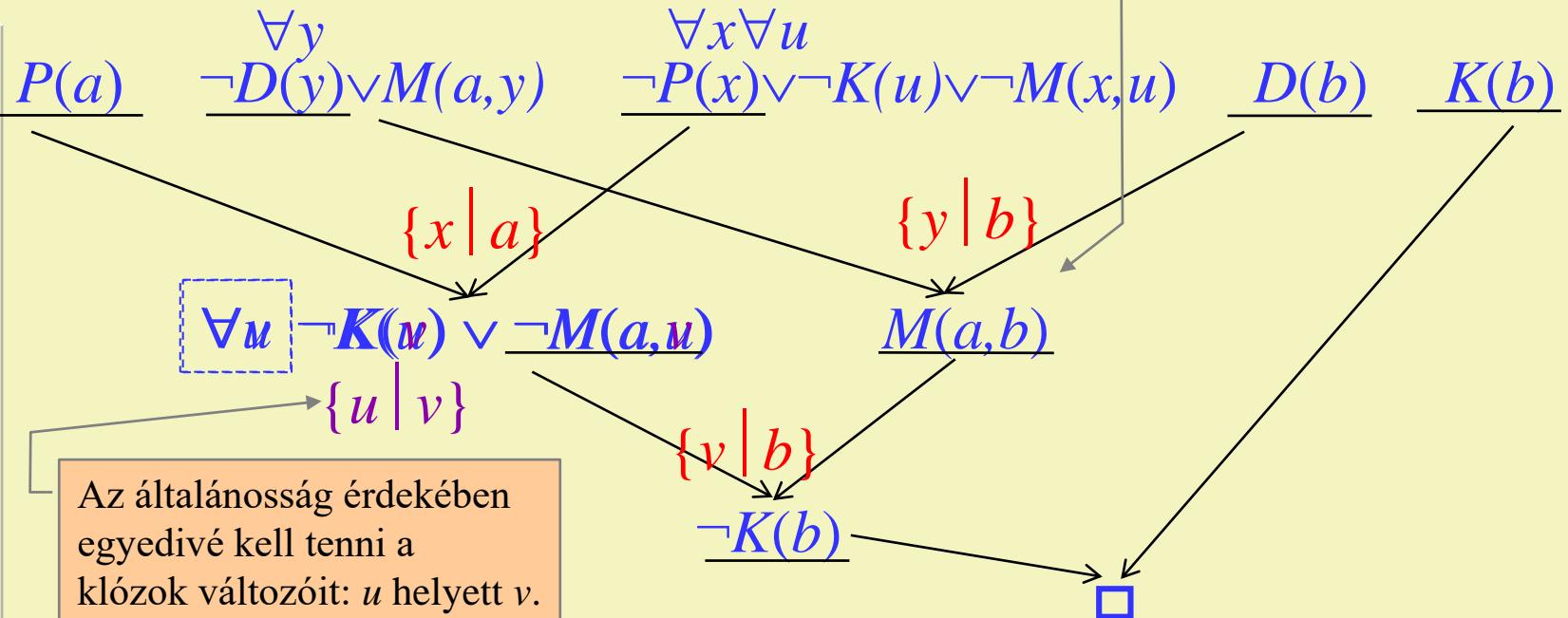
$D(b)$  ,  $Q(b)$

De Morgan's törvény

$\textcolor{red}{b}$  a Skolem konstans

Egy klózpár rezolválásához olyan változó helyettesítésére van szükség, amellyel az elhagyásra kiszemelt komplement literálok azonos alakra hozhatók. (Az [egyesítő algoritmussal](#) egy ilyen helyettesítés található meg.) Ezt követően a klózpár ezen helyettesítéssel kapott példányait rezolváljuk.

## Rezolúciós eljárás



### Általános rezolúciós szabály:

$C_1 = P(t_{11}, \dots, t_{1n}) \vee \dots \vee P(t_{rl}, \dots, t_{rn}) \vee C_1'$ ;  $C_2 = \neg P(u_{11}, \dots, u_{1n}) \vee \dots \vee \neg P(u_{sl}, \dots, u_{sn}) \vee C_2'$

$C_1'$  vagy  $C_2'$  lehet üres, de tartalmazhatják  $P(\dots)$  illetve  $\neg P(\dots)$  további előfordulásait.

ha  $P(t_{11}, \dots, t_{1n}), \dots, P(t_{rl}, \dots, t_{rn}), P(u_{11}, \dots, u_{1n}), \dots, P(u_{sl}, \dots, u_{sn})$  egyesíthetők a  $\delta$  változó-helyettesétéssel, akkor  $C_1$  és  $C_2$  rezolvense:  $R(C_1, C_2) = C_1' \delta \vee C_2' \delta$

# *Rezolúció = lokális keresés*

- globális munkaterület: aktuális klózhalmaz
- kiindulási érték: az „axiómák  $\Rightarrow$  célállítás” klázai
- terminálási feltétel:
  - sikeres
  - sikertelenüres klóz
- kereső szabály: nincs újabb rezolvens klóz
- vezérlési stratégia: rezolvens képzés
- heurisztika: nem-módosítható
- heurisztika: jó lenne a hatékonyság miatt, de sajnos nincs

ADAT := kezdeti érték

while  $\neg$ terminálási feltétel(ADAT) loop

    SELECT SZ FROM alkalmazható szabályok

    ADAT := SZ(ADAT)

endloop

## Rezolúció algoritmusa

$A_1, A_2, \dots, A_n \Rightarrow B$  helyett azt vizsgáljuk, kielégíthetetlen-e az  
 $A_1, A_2, \dots, A_n, \neg B$  formulák klöz formája

1.  $KLÓZOK := A_1, A_2, \dots, A_n$  és  $\neg B$  formulák klözai
2. **loop**
3.     **if**  $\square \in KLÓZOK$  **then return** *kielégíthetetlen*
4.     **if** nincs olyan  $C_1, C_2 \in KLÓZOK$ , amelyre  $R(C_1, C_2)$   
        még nem ismert (nincs a  $KLÓZOK$  közt)  
        **then return** *nem kielégíthetetlen*
5.     **select**  $C_1, C_2 \in KLÓZOK$ , ahol  $R(C_1, C_2)$  nem ismert
6.      $KLÓZOK := KLÓZOK \cup R(C_1, C_2)$
7. **endloop**

# *Rezolúció tulajdonságai*

- **Helyes** (eljárás): ha terminál, akkor helyes eredményt ad.  
(Üres klóz megtalálásakor a kiinduló klóz halmaz kielégíthetetlen, ha nem tud újabb klózt előállítani, akkor a kiinduló klóz halmaz kielégíthető.) Ugyanakkor elsőrendű logikában nem biztosan terminál.
- **Teljes** (eljárás): egy kielégíthetetlen klóz halmazból véges lépésekben levezethető az üres klóz.
- Elsőrendű logikában a kielégíthetetlenség csak **parciálisan dönthető el**, mert nem terminál garantáltan a módszer:  
 $\{\neg P(x), \quad P(y) \vee \neg P(f(y)), \quad P(a)\}$

# Válaszadás rezolúcióval

“Ha Fifi mindenhol követi Jánost, és  
János most az iskolában van,  
akkor hol van most Fifi?”

Formalizáció:

$H(y,x) \sim y$  dolog az  $x$  helyen van

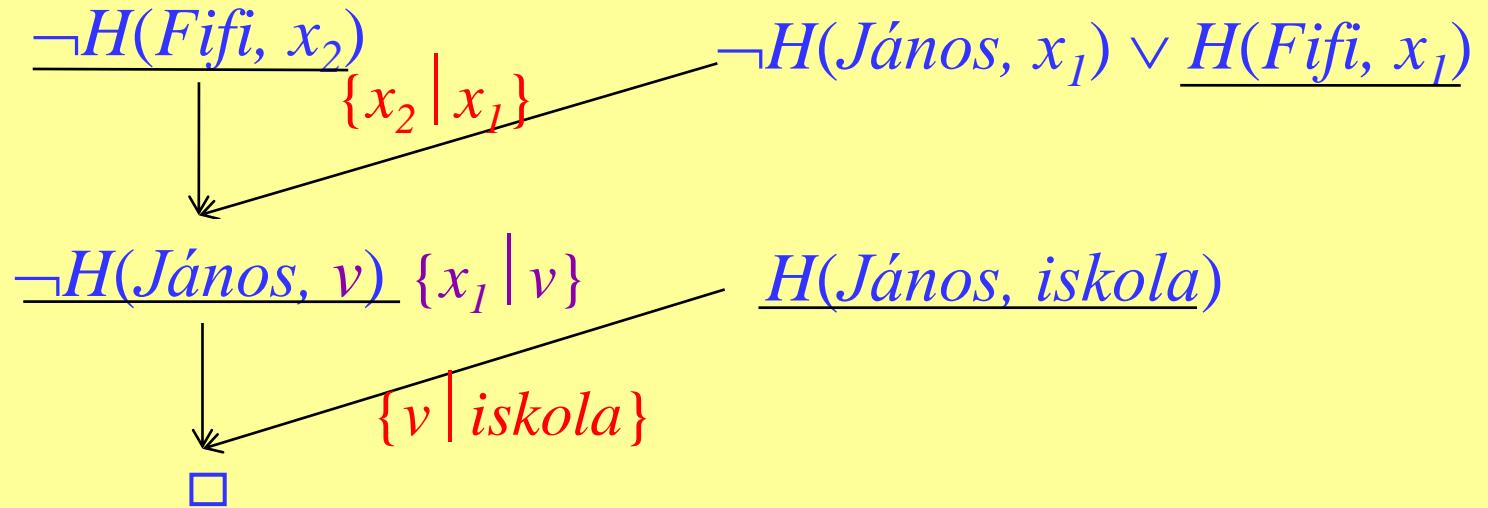
$\forall x[H(János,x) \rightarrow H(Fifi,x)]$

$H(János, \text{iskola})$

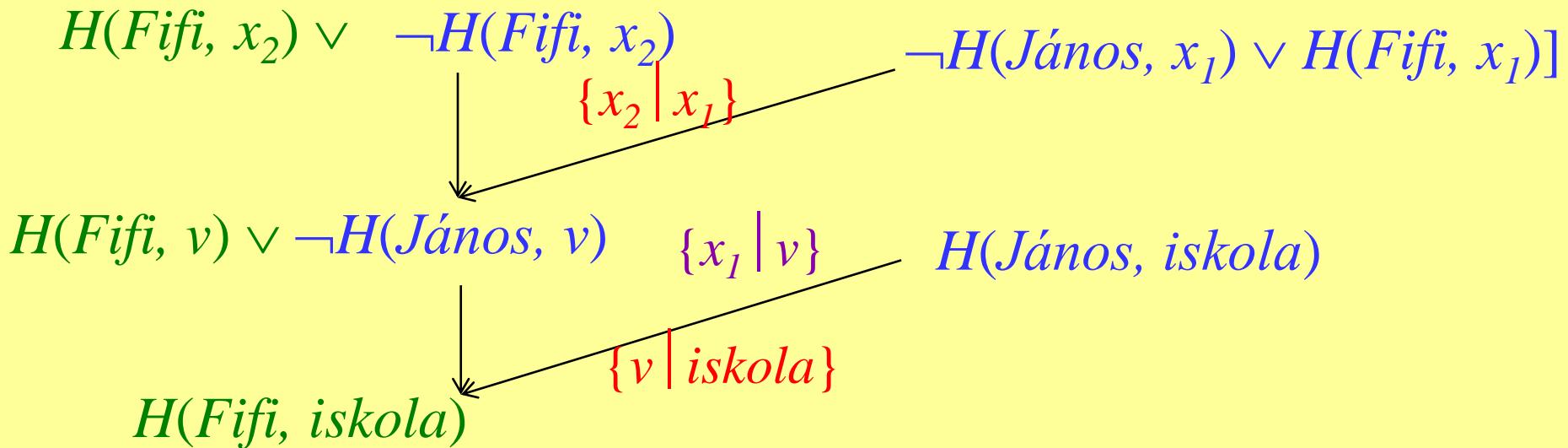
Először lássuk be, hogy létezik-e Fifi számára hely a világban?

$\exists xH(Fifi,x)$

# Cáfolati gráf



# Válaszadási gráf



# *Válaszadási eljárás*

1. A kérdést (ki, mit, hol, mikor, mennyiért) egy „van-e válasz a kérdésre” célállítással helyettesítjük.
2. Rezolúcióval belátjuk, hogy a célállítás következik az axiómákból.
3. A célállítás negáltjából származó klózokat negáltjaik hozzáfűzésével érvényes formulákká egészítjük ki.
4. A cáfolati gráf által meghatározott rezolúciót követve létrehozzuk a hasonló szerkezetű válaszadási gráfot, amelynek gyökere tartalmazza az egyik választ.

# Rezolúciós stratégiák

- A rezolúció **nem-determinisztikus**. Egy lépésben
  - egyszerre több rezolválható klóz pár lehet
  - egy klóz párból több komplement literál pár lehet
  - ugyanannak a literálnak több előfordulása lehet

$$\{P(x,f(a)) \vee P(x,f(y)) \vee Q(y), \quad \neg P(z,f(a)) \vee \neg Q(z), \quad P(u,f(a)) \vee \neg Q(a)\}$$

modellfüggő vezérlési stratégiák: csak klóz alapú reprezentáció esetén értelmezhetőek.

- Egy rezolúciós stratégia a rezolúció alapalgoritmusát kiegészítő olyan előírás, amely
  - **sorrendet ad** a rezolvens képzésekre (sorrendi stratégia)
  - **korlátozza** egy adott pillanatban előállítható rezolvensek körét (vágó strat.)

sérülhet a módszer teljessége

# *Rezolúció kritikája*

- A rezolúció nem jó MI módszer:
  - A számos modellfüggő vezérlési stratégia ellenére **sem hatékony**, sok felesleges rezolúciós lépést végez.
  - **Nem építhető heurisztika** a vezérlési stratégiába, mert az állítások a klóz-formára hozás után már nem emlékeztetnek a feladatban betöltött szerepükre, ezért nehéz „súgni”, hogy mely klózokkal érdemes próbálkozni.

# *A formulák alakja segítheti a következtetést*

Ha például be kell látnunk azt, hogy

$$A, \ C \rightarrow \neg A, \ A \rightarrow B, \ \neg B \rightarrow D \Rightarrow B$$

akkor könnyű kitalálni, hogy mely feltételekre van szükség a bizonyításnál:  $A, \ A \rightarrow B \Rightarrow B$

De a rezolúció nem képes felhasználni ezt a segítséget, hiszen eliminálja az implikációt a formulákból:

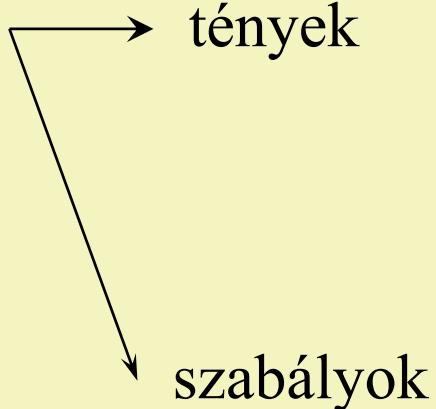
$$A, \ \neg C \vee \neg A, \ \neg A \vee B, \ B \vee D, \ \neg B$$

*Olyan következtetési eljárás kell, ahol az állítások megőrzik eredeti alakjukat, különösen az implikációt.*

## 2. Szabályalapú logikai következtetés

- Egy  $A_1, \dots, A_n \Rightarrow C$  probléma esetén az axiómákat két csoportba soroljuk: **szabályokra** és **tényekre**.

axiómák



konkrét ismeret  
implikáció nélküli  
formulákban

általános ismeret  
implikációs formulákban  
 $A \rightarrow B$

# Az előre- illetve a hátrafelé láncolás

- **Előrefelé láncolás:** egy alkalmas (illeszthető) szabály segítségével egy állításból új állítást vezet le.

tény:  $Kutya(Fifi) \wedge Postás(Jani)$

szabály:  $\forall x \forall y \text{Kutya}(x) \wedge \text{Postás}(y) \rightarrow \text{Harap}(x,y)$

$\Rightarrow \text{Harap}(Fifi, Jani)$

Nehezebb lenne, ha itt nem alaki, hanem logikai ekvivalenciát kellene igazolni. Pl. a tény:  $\neg(\neg \text{Kutya}(Fifi) \vee \neg \text{Postás}(Jani))$

Illeszthetőség vizsgálat: a tény ekvivalens a szabály előfeltételével az  $\{x | Fifi, y | Jani\}$  helyettesítés mellett (amit az egyesítő algoritmus számol ki).

- **Hátrafelé láncolás:** egy állítás bizonyítását visszavezeti egy alkalmas (illeszthető) szabály előfeltételének igazolására.

cél:  $Kutya(Fifi)$

szabály:  $\forall x \text{Ugat}(x) \rightarrow \text{Kutya}(x)$

$\rightsquigarrow$  elég belátni:  $\text{Ugat}(Fifi)$

Szerencsénk van, hogy az illesztésnél literált literállal kellett összevetni. Ekkor alaki azonosság = logikai ekvivalencia

Illeszthetőség vizsgálat: a cél ekvivalens a szabály következményével az  $\{x | Fifi\}$  helyettesítés mellett (amit az egyesítő algoritmus számol ki).

# *Szabályalapú következtetés irányai*

- Egy tényekkel, szabályokkal és célállítással megadott probléma bizonyítható
  - ***előre haladva***: a tényekből indulva előrefelé láncolással új állításokat vezetünk le, majd azokból még újabbakat, amíg a célállítást meg nem kapjuk
  - ***visszafelé haladva***: a célt hátrafelé láncolással részcélokra cseréljük le, a részcélokat további részcélokkal váltjuk fel, amíg tények által igazolható részcélokhoz nem jutunk.

Ezek a módszerek nem teljesek:

Például a  $P \rightarrow Q$ ,  $\neg P \rightarrow Q$  szabályokból a fenti módszerek egyike sem vezeti le a  $Q$  célállítás, pedig  $P \rightarrow Q$ ,  $\neg P \rightarrow Q \Rightarrow Q$

# *Előre haladó szabályalapú reprezentáció*

célja, hogy a hátrafelé láncolásnál literált literállal kelljen összevetni

ÉS/VAGY formájú (ÉVF) kifejezések:

- literálok
- $A \wedge B$ ,  $A \vee B$  alakú formulák, ahol az  $A$  és  $B$  is ÉVF kifejezés.



□ **Tény:**

- univerzálisan kötött **tetszőleges** ÉVF kifejezés

□ **Szabályok:**

- $L \rightarrow W$  alakú univerzálisan kötött kifejezések, ahol  $L$  egy literál, a  $W$  pedig ÉVF kifejezés

□ **Cél:**

- $L_1 \vee \dots \vee L_n$  alakú egzisztenciálisan kötött kifejezés, ahol  $L_1, \dots, L_n$  literálok.

# Példa előre haladó szabályalapú következtetésre

Tény:

$$(A \vee \neg B) \wedge C$$

Szabályok:

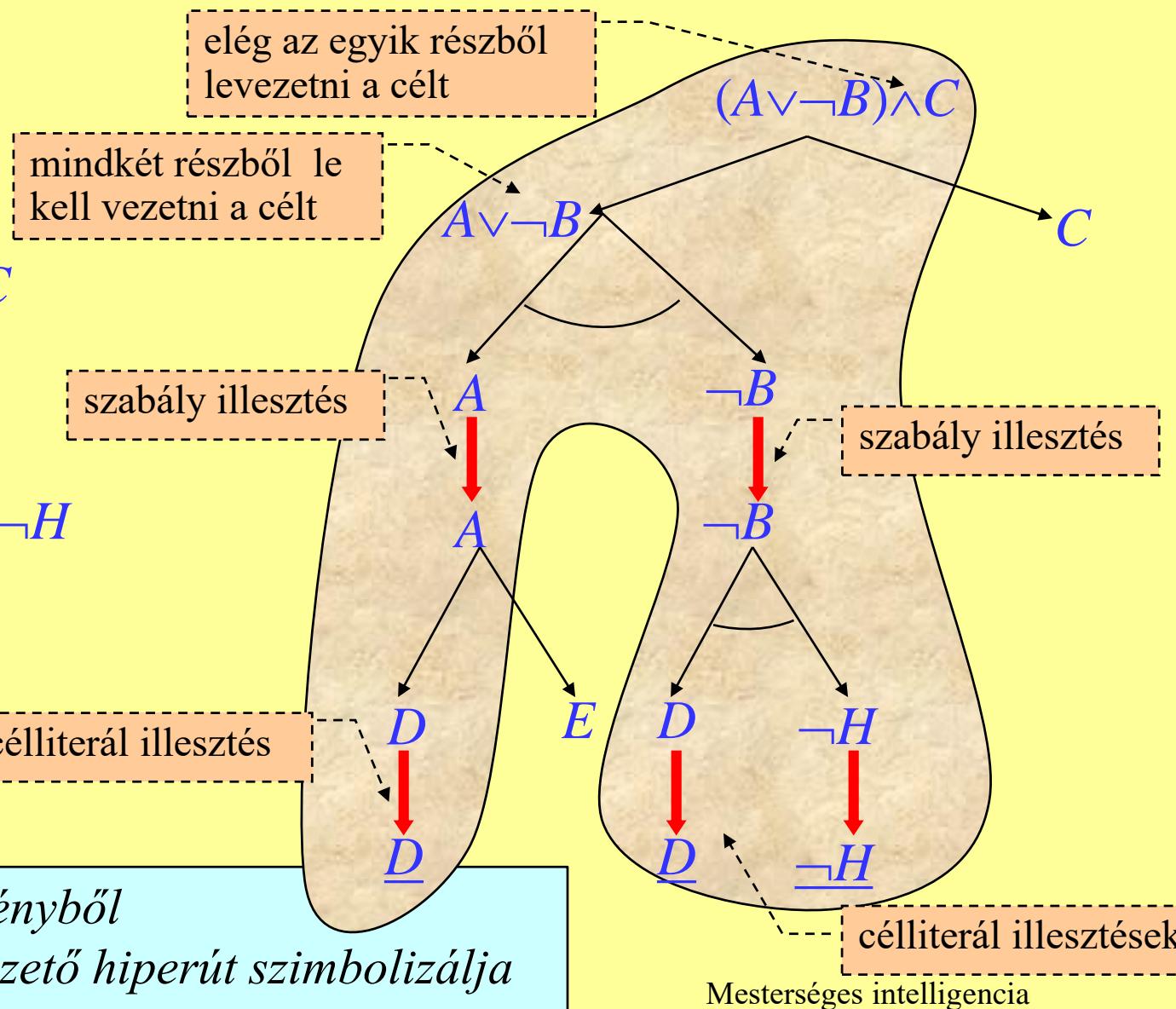
$$A \rightarrow D \wedge E$$

$$\neg B \rightarrow D \vee \neg H$$

Cél:

$$D \vee G \vee \neg H$$

A bizonyítást a tényből  
célliterálokba vezető hiperút szimbolizálja



# *Visszafelé haladó szabályalapú reprezentáció*

célja, hogy a hátrafelé láncolásnál literált literállal kelljen összevetni

## □ Tény:

- $L_1 \wedge \dots \wedge L_n$  alakú univerzálisan kötött kifejezés, ahol  $L_1, \dots, L_n$  literálok.

## □ Szabályok:

- $W \rightarrow L$  alakú univerzálisan kötött kifejezések, ahol  $L$  egy literál, a  $W$  pedig ÉVF kifejezés

## □ Cél:

- egzisztenciálisan kötött **tetszőleges** ÉVF kifejezés

ÉS/VAGY formájú (ÉVF) kifejezések:

- literálok
- $A \wedge B, A \vee B$  alakú formulák, ahol az  $A$  és  $B$  is ÉVF kifejezés.

# Példa visszafelé haladó szabályalapú következtetésre

Tény:

$$A \wedge C \wedge \neg D$$

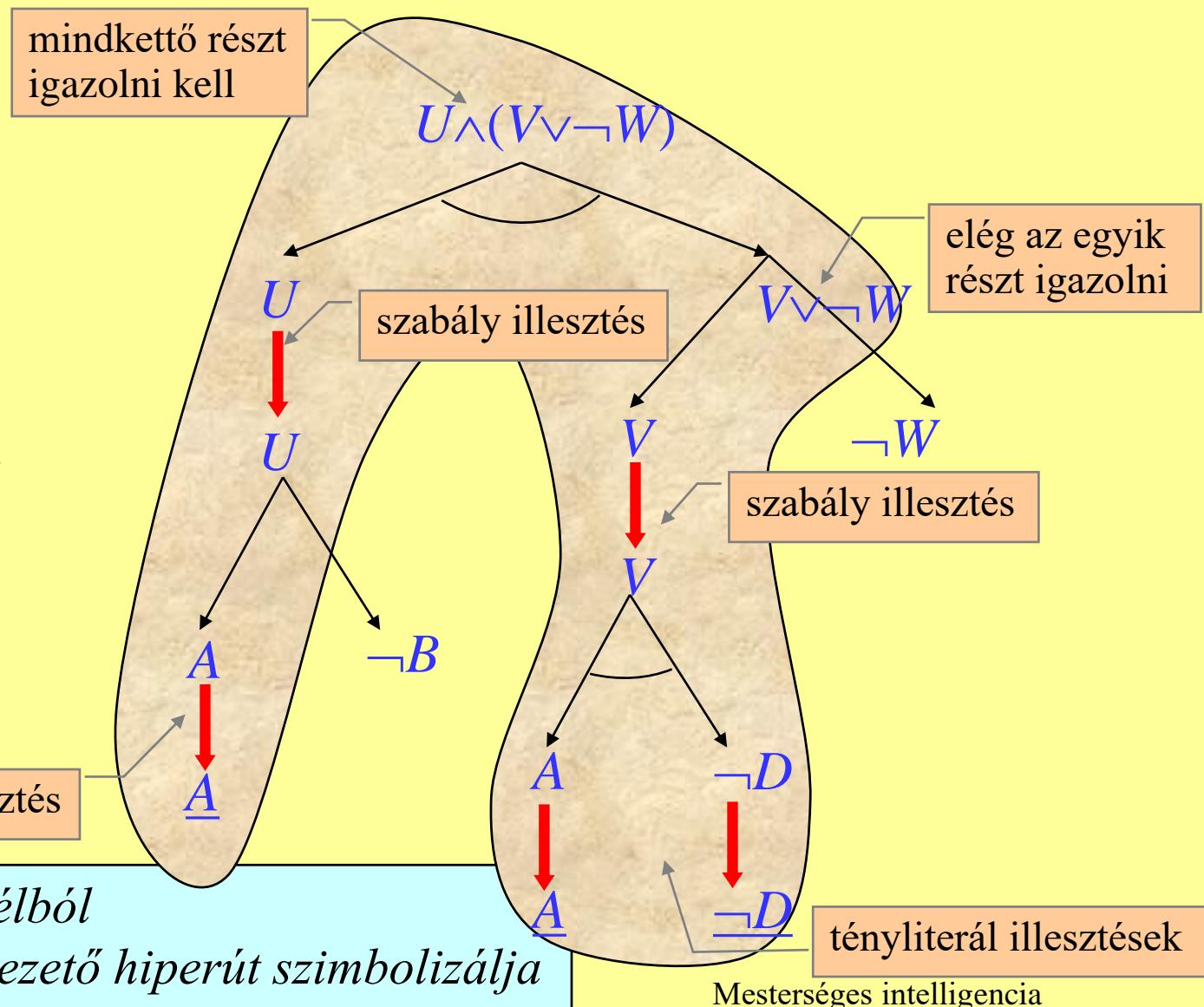
Szabályok:

$$A \vee \neg B \rightarrow U$$

$$A \wedge \neg D \rightarrow V$$

Cél:

$$U \wedge (V \vee \neg W)$$



# Példa

*Fifi és Gyilkos kutyák, Fifi csóválja a farkát, Cili nyágog. A nyágogó állatok a macskák. Az a kutya, amelyik csóválja a farkát, barátságos. A macskák nem félnek a barátságos kutyáktól. Nevezzünk meg olyan kutya-macska párt, ahol a macska nem fél a kutyától!*

Formalizálás:

$K(x) \sim x$  kutya

$M(x) \sim x$  macska

$Cs(x) \sim x$  csóvál,

$Ny(x) \sim x$  nyágog

$B(x) \sim x$  barátságos

$F(x,y) \sim x$  fél  $y$ -tól

Tény:  $K(Fifi) \wedge K(Gyilkos) \wedge Cs(Fifi) \wedge Ny(Cili)$

Szabályok:

$\forall x (Ny(x) \rightarrow M(x))$

$\forall x (K(x) \wedge Cs(x) \rightarrow B(x))$

$\forall x \forall y (K(x) \wedge B(x) \wedge M(y) \rightarrow \neg F(y,x))$

Cél:  $\exists x \exists y (M(x) \wedge \neg F(x,y) \wedge K(y))$

válaszadáshoz:  
van-e keresett kutya-macska pár

Tény:  $K(Fifi), K(Gyilkos), Cs(Fifi), Ny(Cili)$

Szabályok:  $Ny(x_1) \rightarrow M(x_1)$

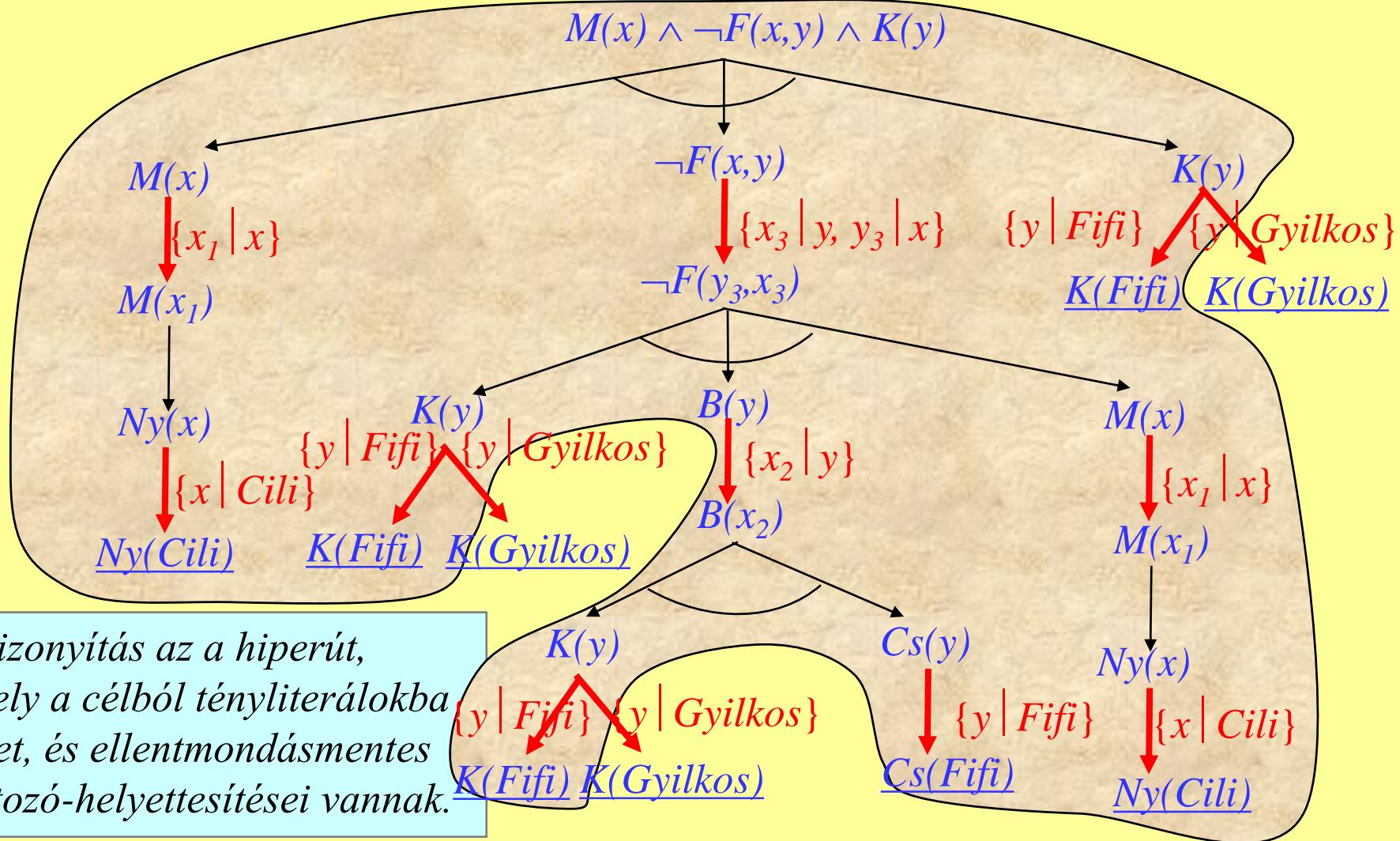
$K(x_2) \wedge Cs(x_2) \rightarrow B(x_2)$

$K(x_3) \wedge B(x_3) \wedge M(y_3) \rightarrow \neg F(y_3, x_3)$

## Bizonyítás

A hiperút változó-helyettesítéseiől olvasható ki a válasz:  $\{x \mid Cili, y \mid Fifi\}$

Cél:



A bizonyítás az a hiperút, amely a célból tényliterálokba vezet, és ellentmondásmentes változó-helyettesítései vannak.

# *Szabályalapú következtetés = visszalépéses keresés*

- Célja egy bizonyítás keresése, amit egy ÉS/VAGY gráfbeli ellentmondásmentes megoldás gráf reprezentál.
- Kereső rendszer
  - Globális munkaterület: megkezdett bizonyítás (hiperút)
  - Kereső rendszer szabályai: láncolások illetve visszalépés
  - Vezérlési stratégia (elsődleges): visszalépéses stratégia
  - Modellfüggő stratégiák
    - Formulák alakjának kihasználása
    - A tény (cél) illesztése előzze meg a szabály-illesztést.
  - Heurisztikák: az adott feladat speciális ismeretei
    - Metaszabályok, kiértékelő függvény



# Gépi tanulás

# *Tanulás fogalma*

- ❑ Egy algoritmus akkor tanul, ha egy feladat megoldása során olyan változások következnek be a működésében, hogy később ugyanazt a feladatot vagy ahhoz hasonló más feladatokat jobb eredménnyel, illetve jobb hatékonysággal képes megoldani, mint korábban.
- ❑ A tanulás során változhat a feladat
  - reprezentációja (logikai formulák, valószínűségi hálók)
  - megoldó algoritmusa (mély hálók, genetikus programozás)
  - heurisztikája ( $B'$  algoritmus)

# Tanulási modellek

- Ha a megoldandó problémát egy  $\varphi : X \rightarrow Y$  leképezés modellezzi, akkor ehhez azt az  $f : X \rightarrow Y$  leképezést kiszámító algoritmust keressük (tanuljuk meg), amelyre  $f \approx \varphi$ 
  - sokszor egy rögzített  $f : P \times X \rightarrow Y$  leképezést használunk, és annak azon  $\Theta \in P$  paraméterét keressük, amelyre  $f(\Theta, x) \approx \varphi(x)$
- *Induktív tanulási modell*
  - $f$  leképezést (illetve annak paraméterét)  $x_n \in X$  ( $n=1..N$ ) bemenetek (**minták**) alapján tanuljuk
- *Adaptív (inkrementális) tanulás*
  - Egy már megtanult  $f$  leképezést egy új minta anélkül módosít, hogy a korábbi mintákat újra meg kell vizsgálnunk.

# *Induktív modellek tanulási módjai*

- *Felügyelt tanulás*: ismeri a tanuláshoz használt minták elvárt kimenetét is, azaz az  $(x_n, \varphi(x_n))$  ( $n=1..N$ ) input-output párok alapján tanul.
- *Felügyelet nélküli tanulás*: nem ismeri a tanuláshoz használt minták elvárt kimenetét, csak  $x_n$  ( $n=1..N$ ) lehetséges inputokat; a minták illetve az azokra kiszámolt kimenetek közötti összefüggéseket próbálja felismerni, azokat osztályozni.
- *Megerősítéses tanulás*: nem ismeri ugyan a tanuláshoz használt minták elvárt kimenetét, de képes az  $x_n$  ( $n=1..N$ ) inputokra kiszámolt eredményt minősíteni, hogy az mennyire megfelelő.

# 1. Felügyelt tanulás

- A problémát modellező  $\varphi : X \rightarrow Y$  leképezés közelítéséhez választunk egy  $f : P \times X \rightarrow Y$  paraméteres leképezést, majd ennek azon  $\Theta \in P$  paraméterét keressük (*paraméteres tanulás*), amelyre az  $(x_n, y_n)$  ( $n=1..N$ ) tanító minták mellett (ahol  $y_n = \varphi(x_n)$ ) az alábbi hiba már elég kicsi (ettől reméljük, hogy  $f(\Theta, x) \approx \varphi(x)$ )

$$\frac{1}{N} \sum_{n=1}^N \ell(f(\Theta, x_n), y_n)$$

Diagram illustrating the components of the loss function:

- hiba függvény (highlighted in orange)
- elvárt kimenet (highlighted in orange)
- számított kimenet (highlighted in orange)
- $t_n$  (highlighted in blue)

The diagram shows the components of the loss function: the expected output (elvárt kimenet) and the calculated output (számított kimenet) are compared via the loss function (hiba függvény) to produce the average loss over all training samples.

- $\ell : Y \times Y \rightarrow \mathbb{R}$  **hibafüggvény**:

- $\ell(t_n, y_n)$  lehet például  $\|t_n - y_n\|_1$ ,  $\|t_n - y_n\|_2^2$ , vagy  $-\sum_i y_n i \cdot \log t_n i$ .

# Megjegyzés

- ❑ Fontos, hogy az  $f(\Theta, x)$  kiszámítása gyors legyen; nem baj, ha a megfelelő  $\Theta$  megtalálása lassú, hiszen ezt a tanító minták segítségével előre számoljuk ki.
- ❑ A  $\Theta$  megtanulása akkor működik jól, ha
  - $N$  elég nagy (Ugyanakkor számolni kell azzal, hogy a mintákat drága összegyűjteni, a  $\varphi(x_n)$ -eket költséges kiszámolni.)
  - $f$  és  $\ell$  megfelelőek (ehhez tapasztalat, sok próbálkozás kell)
  - $\Theta$  közel esik a paraméter globális optimumához
- ❑ A  $\Theta$  megtalálása egy nemkonvex optimalizálási feladat: a  $\Theta$  globális optimumát megtalálni NP-teljes probléma. Szerencsére ez nem is cél, mert ezzel túl mohó módszert kapnánk (túltanulás), amely a tanító mintákra tökéletes, de egyébként nem.

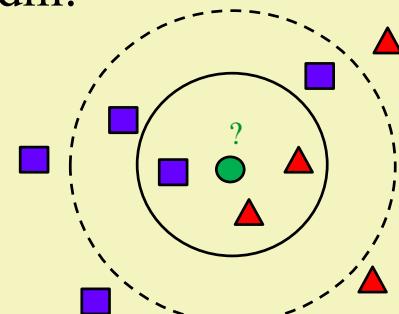
# 1.1. $K$ legközelebbi szomszéd

- Az  $f$  függvény veszi a minták közül az  $x \in X$  bemenethez legközelebb eső bemenettel rendelkező  $K$  darab mintát, és ezek kimenetei alapján (pl. átlagolással) határozza meg az  $x$  kimenetét:

$$f(\Theta, x) = \sum_{n=1}^N \frac{\mathbb{I}(x_n \text{ az egyike az } x\text{-hez legközelebb eső } K \text{ darab tanító minta inputjainak})}{K} \cdot y_n$$

igaz állításra 1-et, különben 0-t ad

- a  $\Theta$  paramétert (ami egyszerűen a mintákból, másrészről a  $K \in \mathbb{N}$  számból áll) nem kell optimalizálni, hanem előre meg kell adni.
- a legközelebbi szomszédokat az  $\|x_n - x\|_2^2$  távolságok sorba rendezésével választjuk ki
- előny:** egyszerű leprogramozni, a „tanulás” gyors
- hátrány:** ha  $N$  nagy, a tárolás, és a minták sorba rendezése erőforrásigényes, az  $f$  kiszámítása lassú



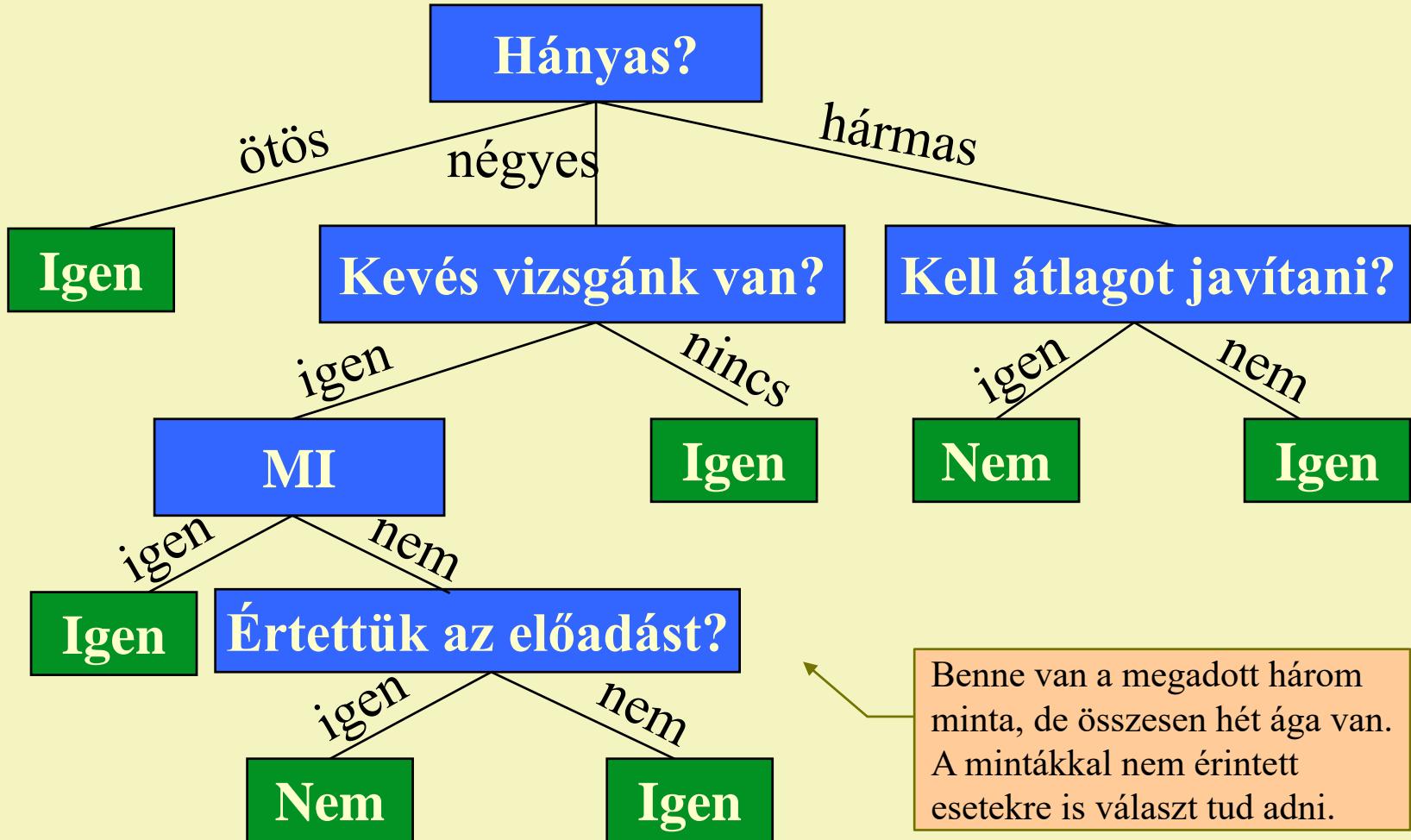
## 1.2. Döntési fa

- Tegyük fel, hogy az  $x \in X$  bemeneteknek ugyanazon tulajdonságait (adott attribútumok értékeit) ismerjük, azaz egy bemenet **attribútum-érték párok halmazával jellemzhető**.
- Képzeljük el azt az irányított fát, amelynek
  - **belső csúcsai egy-egy attribútumot** szimbolizálnak, és az abból kivezető éleket ezen attribútum lehetséges értékei címkézik
  - **ágai attribútum-érték párok halmazát** jelölik ki
  - **leveleihez azon tanító minták** rendelhetők, amelyeket a levélhez vezető út attribútum-érték pá�jaival rendelkeznek.
- Egy  $x$  bemenet az attribútum-értéke párai alapján a döntési fa egyik levelére képezhető le, és ekkor a levélhez tartozó minták kimenetei alapján számítható az  $x$ -hez tartozó kimenetet.

# Példa: Elfogadjuk-e a megajánlott vizsgajegyet?

- ❑ Minták (attribútum-érték párok és a válasz):
  - Ha az ötös, akkor feltétlenül.
  - Ha négyes és kevés vizsgánk van és értettük az előadást, akkor nem; feltéve, hogy a tárgy nem a mesterséges intelligencia.
  - Ha hármas és az átlagot kell javítanunk, akkor nem.
- ❑ Attribútumok és lehetséges érékeik:
  - hányat ajánlottak meg (1, 2, 3, 4, 5)
  - kevés vizsgánk van-e (igen, nem)
  - kell-e átlagot javítani? (igen, nem)
  - az MI tárgyról van-e szó? (igen, nem)
  - értettük-e az előadást? (igen, nem)

# *A példa egy döntési fája*



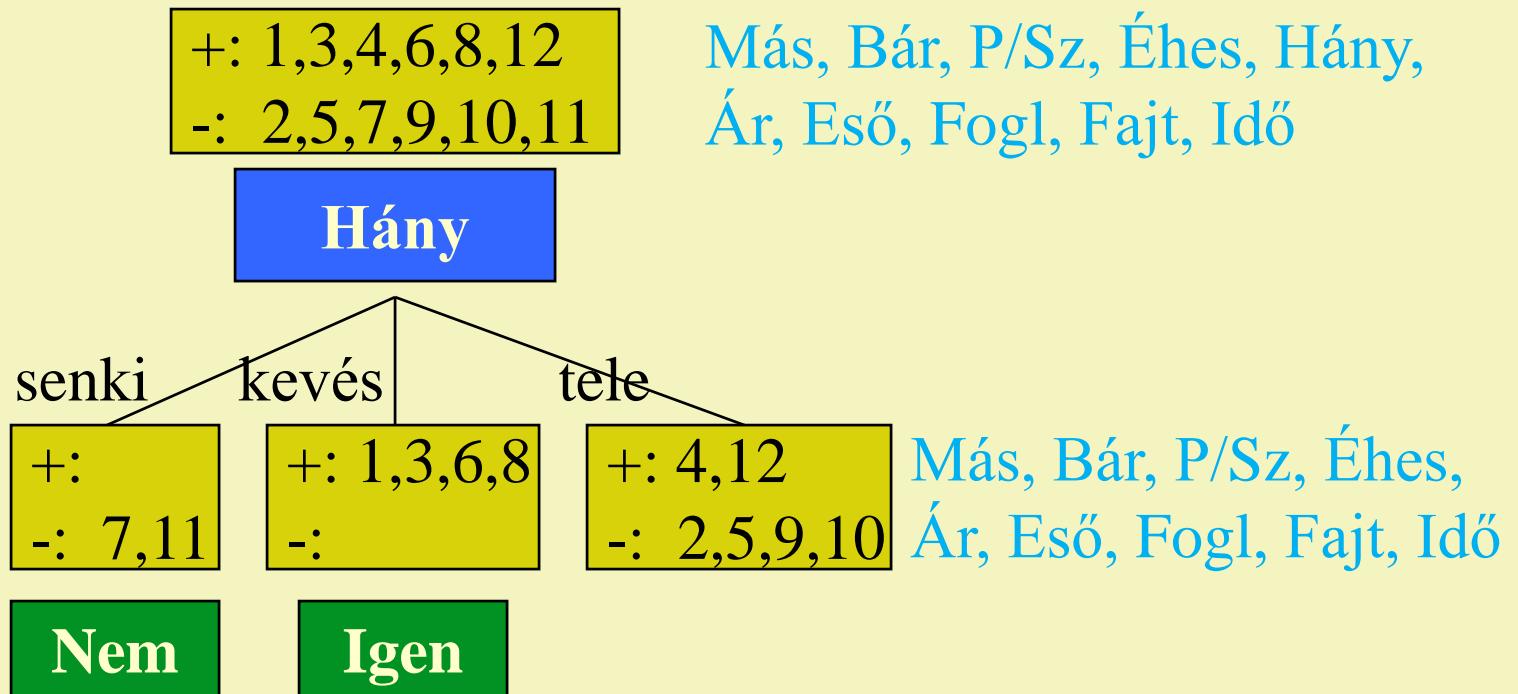
# *Döntési fa felépítése*

- A döntési fát egy  $(x_n, y_n)$  ( $n=1..N$ ) tanító mintahalmaz segítségével építjük fel (ahol  $y_n = \varphi(x_n)$ ).
  - Az építés során egy csúcshoz a tanító minták egy részhalmaza tartozik, amelyet a csúcshoz választott attribútum diszjunkt részekre vág szét, és e részeket a csúcs gyermekei kapják meg.
  - Egy levélcsúcs értékét ezen csúcshoz tartozó tanító minták kimenetei adják: ez lehet az átlaguk vagy leggyakoribb értékük. (Ha ez nem dönt, akkor a levélcsúcs szülőcsúcsának mintáit vizsgáljuk.)
- Egy tanító mintahalmazhoz több döntési fa is megadható.
- A legkisebb (legtömörebb) döntési fa megadása egy NP-teljes probléma.

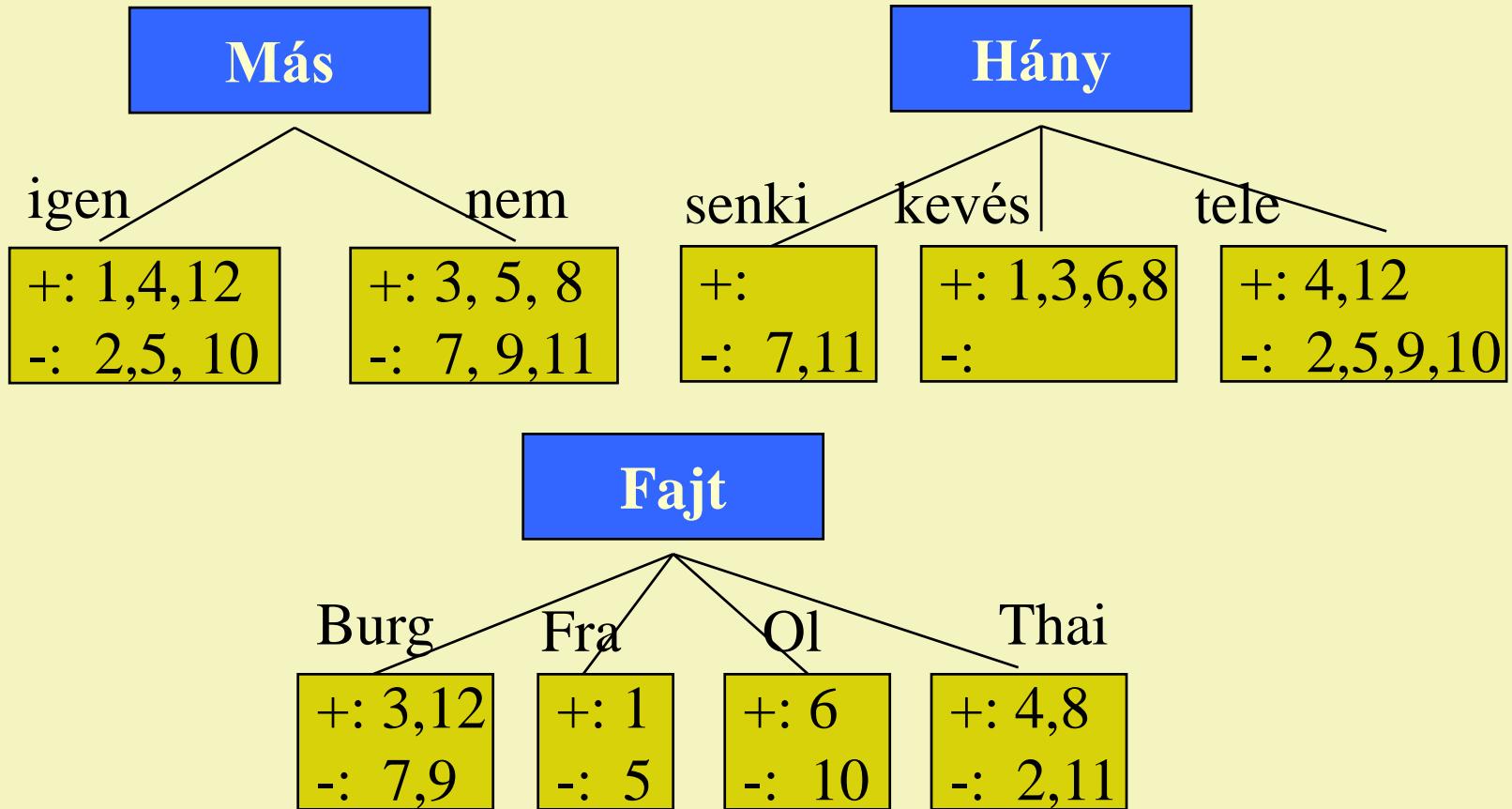
# Étterem probléma (Russel-Norvig)

Pl.	Más	Bár	P/Sz	Éhes	Hány	Ár	Eső	Fogl	Fajt	Idő	Marad
1	I	N	N	I	kevés	drá	N	I	Fra	10	I
2	I	N	N	I	tele	olcs	N	N	Tha	60	N
3	N	I	N	N	kevés	olcs	N	N	Bur	10	I
4	I	N	I	I	tele	olcs	N	N	Tha	30	I
5	I	N	I	N	tele	drá	N	I	Fra	sok	N
6	N	I	N	I	kevés	köz	I	I	Ol	10	I
7	N	I	N	N	senki	olcs	I	N	Bur	10	N
8	N	N	N	I	kevés	köz	I	I	Tha	10	I
9	N	I	I	N	tele	olcs	I	N	Bur	sok	N
10	I	I	I	I	tele	drá	N	I	Ol	30	N
11	N	N	N	N	senki	olcs	N	N	Tha	10	N
12	I	I	I	I	tele	olcs	N	N	Bur	60	I

# *Döntési fa építésének első lépése*



# *Alternatív lépések*



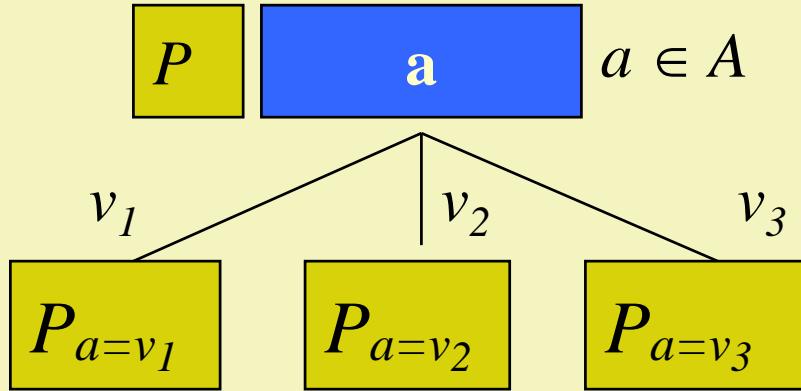
# Heurisztika

- ❑ A döntési fa minél tömörebb, egy-egy ága minél rövidebb lesz, ha
  - egy csúcshoz kiválasztott attribútum (*a*) a csúcshoz tartozó mintákat olyan részhalmazokra vágja szét, amelyeken belül a minták minél homogénebbek, minél kevésbé különböznek,
  - ezt valamilyen távolság fogalom (2-es norma, kereszt entrópia) alapján vizsgálhatjuk
    - Pl.: a **szétvágás információs előnyét** – a szétvágás előtti minta-halmaz információ tartalmának (entrópijának) és az utána kapott minta-részhalmazok információ tartalmának (számosságuk szerinti súlyozott) összege közti különbséget – maximalizáljuk.

# *Információ tartalom (Entrópia)*

- ❑ A  $P$ -beli minták információtartalma (entrópiája), ha csak kétféle kimenetű minta van:
  - $E(P) = E(p^+, p^-) = - p^+ \log_2 p^+ - p^- \log_2 p^-$
  - ahol  $p^+$  a  $P$ -beli pozitív,  $p^-$  a negatív minták aránya ( $p^+ + p^- = 1$ )
- ❑ Példa:
  - Ha  $P$ -ben 2 pozitív és 3 negatív minta van:
$$E(P) = E(2/5, 3/5) = 0.97$$
  - Ha  $P$ -ben 0 pozitív és 3 negatív minta van:
$$E(P) = E(0/3, 3/3) = 0$$

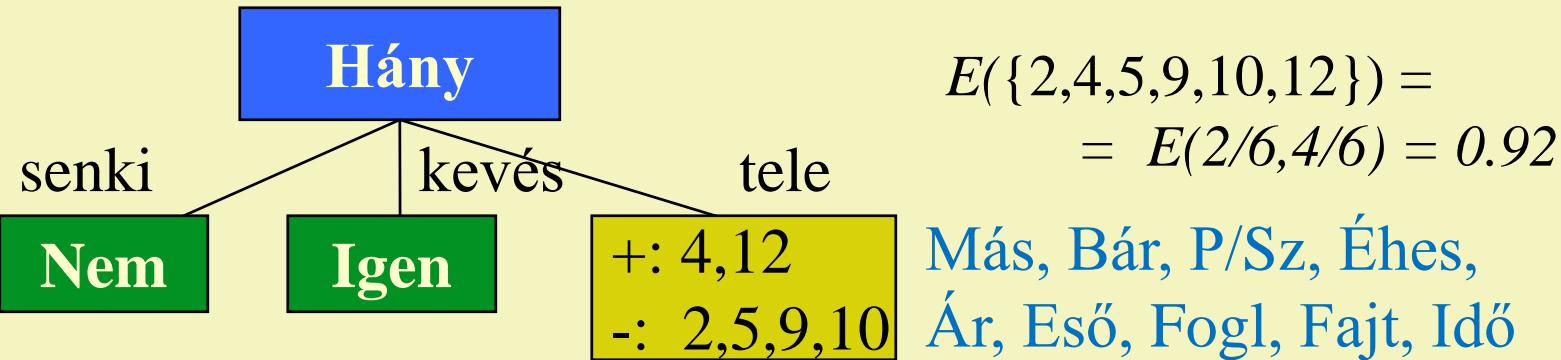
# Információs előny számítása



$$C(P,a) = E(P) - \sum_{v \in \text{Érték}(a)} \frac{|P_{a=v}|}{|P|} E(P_{a=v})$$

- ahol  $P$  a szülő csúcs mintái,  $a$  a választott attribútum,
- az  $\text{Érték}(a)$  az  $a$  attribútum által felvett értékek, és
- a  $P_{a=v} = \{ p \in P \mid p.a=v \}$

# Egy csúcs attribútumának kiválasztása 1.



- ❑ Ha a *Más* attribútumot választjuk, akkor a minták 1:5 arányban ketté válnak: {9} (*Más= hamis*), és {2, 4, 5, 10, 12} (*Más=igaz*),
  - $E(\{9\}) = E(0/1, 1/1) = 0$
  - $E(\{2,4,5,10,12\}) = E(2/5, 3/5) = 0.97$
- ❑ Az információs előny:  $C(\{2,4,5,9,10,12\}, \text{Más}) = E(\{2,4,5,9,10,12\}) - (1/6 E(\{9\}) + 5/6 E(\{2,4,5,10,12\})) = E(2/6,4/6) - (1/6 E(0/1,1/1) + 5/6 E(2/5,3/5)) = 0.92 - 0.81 = 0.11$

# *Egy csúcs attribútumának kiválasztása 2.*

$$C(\{2,4,5,9,10,12\},a) = 0.92 -$$

Más:  $1/6 E(0/1,1/1) + 5/6 E(2/5,3/5) = 0.81$

Bár:  $3/6 E(1/3,2/3) + 3/6 E(1/3,2/3) = 0.92$

P/Sz:  $1/6 E(0/1,1/1) + 5/6 E(2/5,3/5) = 0.81$

Éhes:  $4/6 E(2/4,2/4) + 2/6 E(0/2,2/2) = 0.67$

Ár:  $4/6 E(2/4,2/4) + 0/6 E(0,0) + 2/6 E(0/2,2/2) = 0.67$

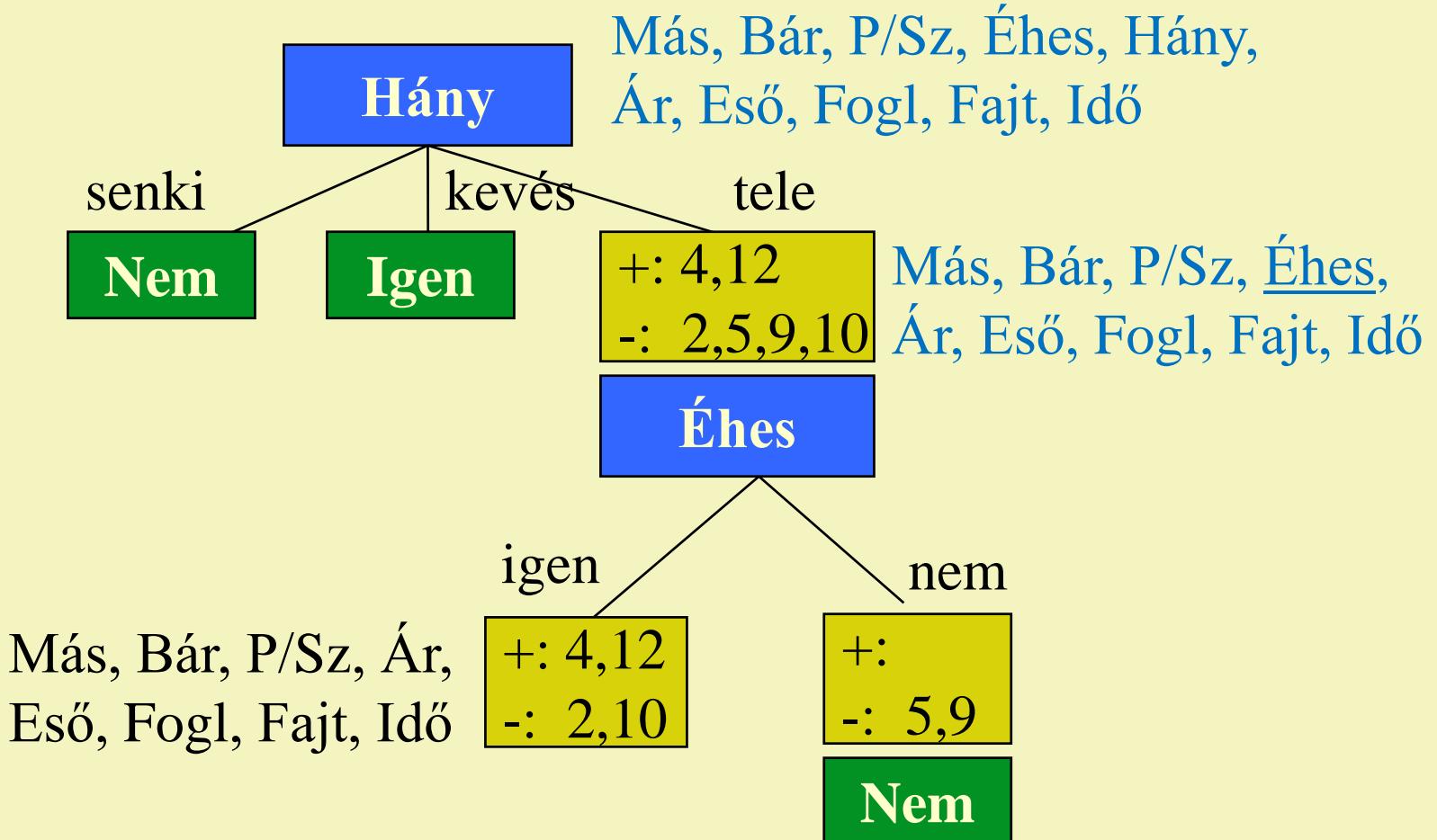
Eső:  $5/6 E(2/5,3/5) + 1/6 E(0/1,1/1) = 0.81$

Fog:  $4/6 E(2/4,2/4) + 2/6 E(0/2,2/2) = 0.67$

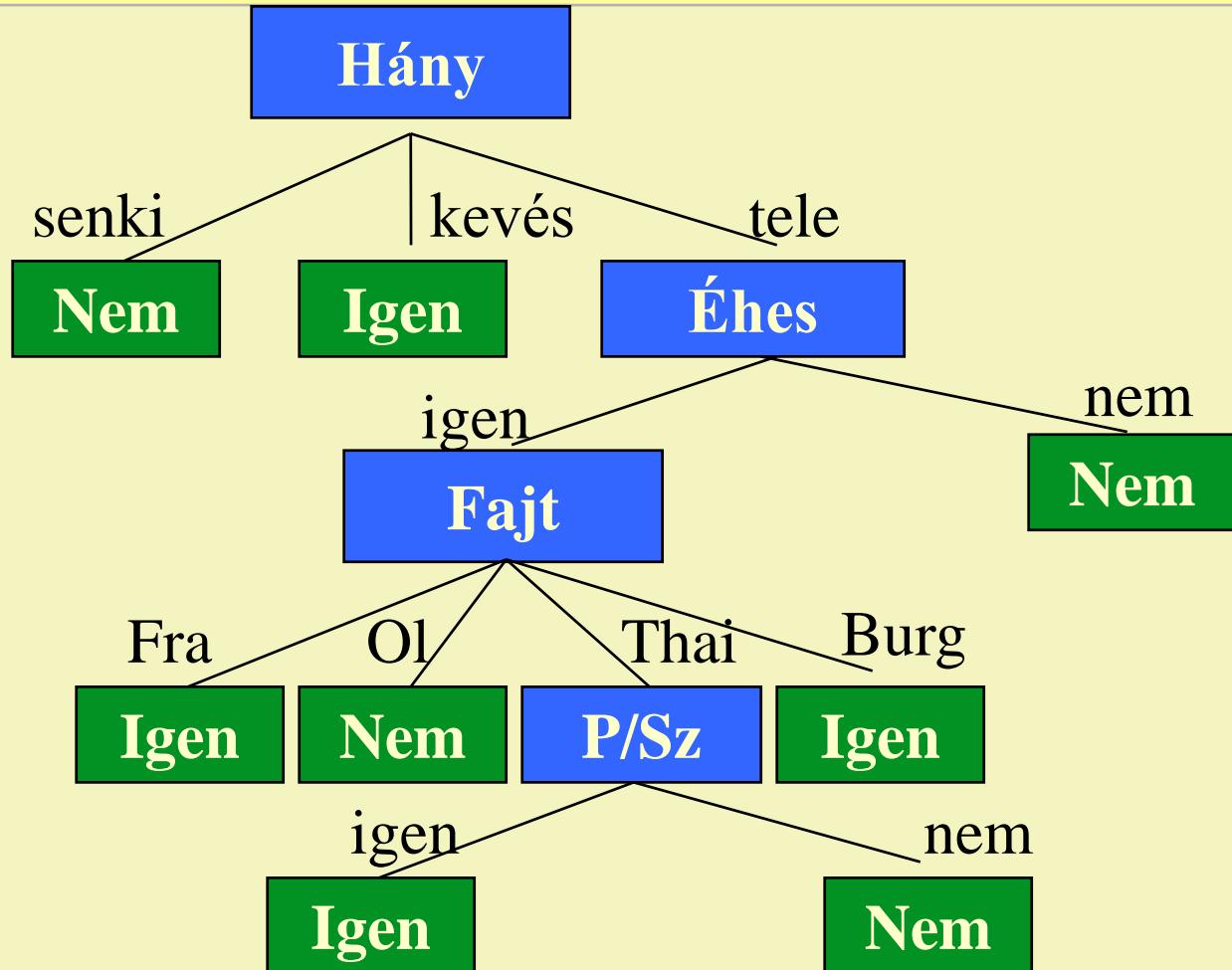
Fajt:  $2/6 E(1/2,1/2) + 1/6 E(0/1,1/1) + 1/6 E(0/1,1/1) + 2/6 E(1/2,1/2) = 0.67$

Idő:  $0/6 E(0,0) + 2/6 E(1/2,1/2) + 2/6 E(1/2,1/2) + 2/6 E(0/2,2/2) = 0.67$

# További lépések



# Étterem probléma döntési fája



# *Készítsünk algoritmust*

- ❑ Egy fokozatosan épülő döntési fában a csúcsokhoz a tanító minták egy részhalmaza, valamint a még választható (a csúcshoz vezető út csúcsainak címkéiben nem szereplő) attribútumok tartoznak. Ezek a csúcsok lehetnek
  - attribútummal **címkézett belső csúcsok**, amelyekből kivezető élek az attribútum lehetséges értékeit képviselik
  - **kiértékelt vagy értékkel nem rendelkező levélcsúcsok**
- ❑ minden lépésben egy értékkel még nem rendelkező levélcsúcsról kell eldöntenni, hogy kaphat-e értéket vagy belső csúcs legyen-e.
  - Előbbi esetben az értéke a csúcshoz tartozó minták értékei alapján (átlag vagy leggyakoribb érték) számolható.
  - Utóbbi esetben egy attribútumot választunk címkéjének, és generáljuk a gyerekeit.

# *Algoritmus*

- Kezdetben a fa egyetlen címkézettlen csúcsból áll (ez lesz majd a gyökér), amelyhez az összes mintát és attribútumot rendeljük.
- Veszünk egy értékeletlen levélcsúcsot:
  1. Ha  $A = \emptyset$ , akkor a mintái alapján kiértékeljük.
  2. Ha  $P = \emptyset$ , akkor a szülőcsúcsának mintái alapján kiértékeljük.
  3. Ha  $P$  csupa azonos kimenetű mintából áll, akkor a mintái alapján kiértékeljük.
  4. Egyébként ...

# *Algoritmus (folytatás)*

4. Egyébként a legnagyobb információs előnnyel járó  $a \in A$  attribútummal címkézzük az adott csúcsot, majd generáljuk a gyerekeit:
- Ezekhez az  $a$  lehetséges értékeivel címkézett élek vezetnek.
  - Ha az  $a$  címkéjű csúcsból egy gyerekcsúcsába a  $v$  címkéjű él vezet, akkor a gyerekcsúcshoz rendelt
    - minták:  $P_{a=v} = \{ p \in P \mid p.a = v \}$
    - választható attribútumok:  $A = A - \{a\}$
  - Végül minden gyerekre ismételjük meg rekurzív módon az 1-4 pontokat.

# *Megjegyzés*

- ❑ **Zaj:** Két vagy több eltérő besorolású minta attribútum-értékei megegyeznek.
  - Ilyenkor a minták válaszainak átlagolása félrevezethet
- ❑ **Túlzott illeszkedés:** A bemenetek olyan attribútumait is figyelembe veszünk, amelyek a kimenetre nincsenek hatással. (Például egy kocka dobás eredményére annak színe és dátuma alapján értelmetlen szabályszerűségeket találunk.)
  - A lényegtelen attribútumokat ( $C(P,a) \sim 0$ ) állítsuk félre.
- ❑ Általánosítások:
  - Hiányzó adatok (attribútum értékek) problémája
  - Folytonos értékű attribútumok

# *Tanulás döntési fával*

- ❑ Egy  $x \in X$  bemenethez azon tanító minták kimenetei alapján számol kimenetet, amely minták az előzetesen felépített döntési fában az  $x$ -re kiszámolt levélcsúcshoz tartoznak

$$f(\Theta, x) = \sum_{n=1}^N \frac{\mathbb{I}(\text{az } x\text{-re kiszámolt levélcsúcs } K' \text{ darab tanító mintájának egyike az } x_n)}{K'} \cdot y_n$$

amikor  $x$  kimenete a hozzá kiszámolt levélcsúcs mintái kimeneteinek átlaga

- Itt a  $\Theta$  a döntési fa, optimalizálása annak mohó felépítése
- **előny:** jól értelmezhető (a mintákra tökéletes eredményt, a mintákhoz hasonló inputokra többnyire jó eredményt ad);  
a tanító minták helyett csak a döntési fát kell tárolni;  
 $x$ -re adott eredmény gyorsan számolható
- **hátrány:** a faépítés NP-teljes, mohó módszerrel csak lokálisan optimális

## 1.3. Véletlen erdő

- $K$  darab döntési fát építünk a tanító minták alapján úgy, hogy egy-egy fa építéséhez a tanító mintáknak is, és a minták attribútumainak is csak egy-egy véletlen kiválasztott részhalmazát használjuk fel. Ez lesz a **véletlen erdő**.
- Egy véletlen erdő minden fájában külön-külön megállapíthatjuk, hogy egy  $x \in X$  bemenet a döntési fa melyik levelére képződik le. Ezen levelekhez tartozó tanító mintahalmazok kimeneteinek súlyozott átlagával becsüljük az  $x$  kimenetét.

# *Tanulás véletlen erdővel*

- ❑ Egy  $x$  bemenethez tartozó kimenetet a minták kimeneteinek súlyozott átlaga, ahol a súlyok attól függnek, hogy egy minta a véletlen erdő döntési fáinak  $x$ -re kiszámolt levélcsúcsaihoz tartozó mintahalmazok közül hányba esik bele, és az a halmaz hány elemű:

$$f(\Theta, x) = \sum_{n=1}^N \sum_{k=1}^K \frac{\mathbb{I}(\text{az } x_n \text{ a } k\text{-adik fa } x\text{-re kiszámolt levélcsúcsához tarto} zó } K_k(x) \text{ darab mintának az egyike})}{K \cdot K_k(x)} \cdot y_n$$

- a  $\Theta$  maga a véletlen erdő, optimalizálása az erdő felépítése
- előny: a tanító minták helyett csak az erdőt kell tárolni; a véletlen generálás miatt kevésbé mohó, elkerüli a túltanulást; az  $x$ -re adott eredmény számolása párhuzamosítható
- hátrány: az eredmény kevésbé értelmezhető; az erdő-építés NP-teljes

# 1.4. Mesterséges neuronhálók

## ❑ Mesterséges neuronhálók alkotóelemei

### ○ Mesterséges neuron

- bemenő értékekből kimenő értéket számoló egység, amelynek számítási képlete változtatható, tanítható

### ○ Hálózati topológia

- sok mesterséges neuron egymáshoz kapcsolva, ahol egyik neuron kimenete egy másik neuron bemenete lesz
- bizonyos neuronok a bemenetüket a hálózaton kívülről kapják, mások kimeneteit pedig hálózat kimenetének tekintjük

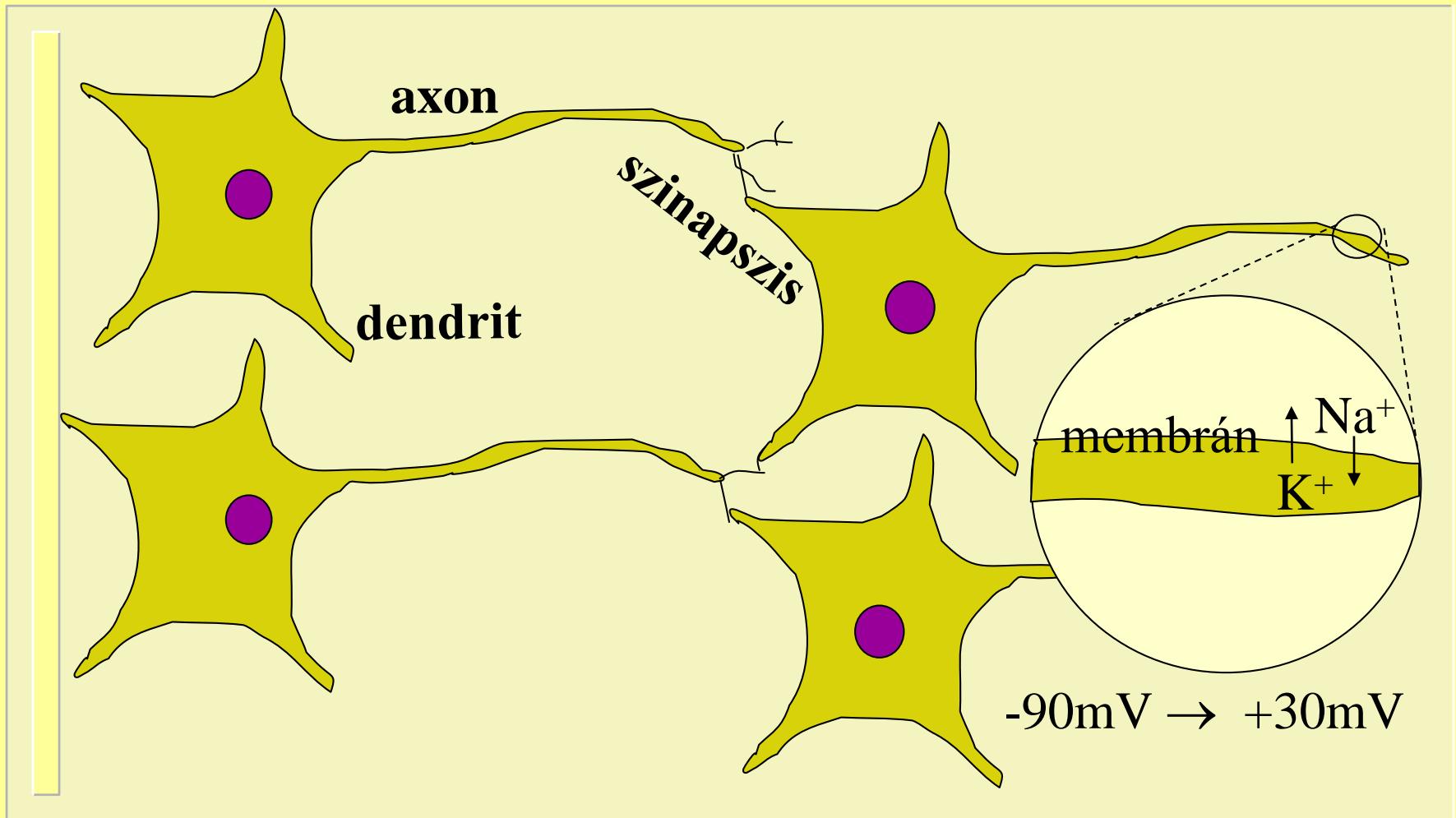
### ○ Tanulási szabály

- egy neuron számítási képletét meghatározó eljárás, amely lehet egy tanító példák alapján működő algoritmus is.

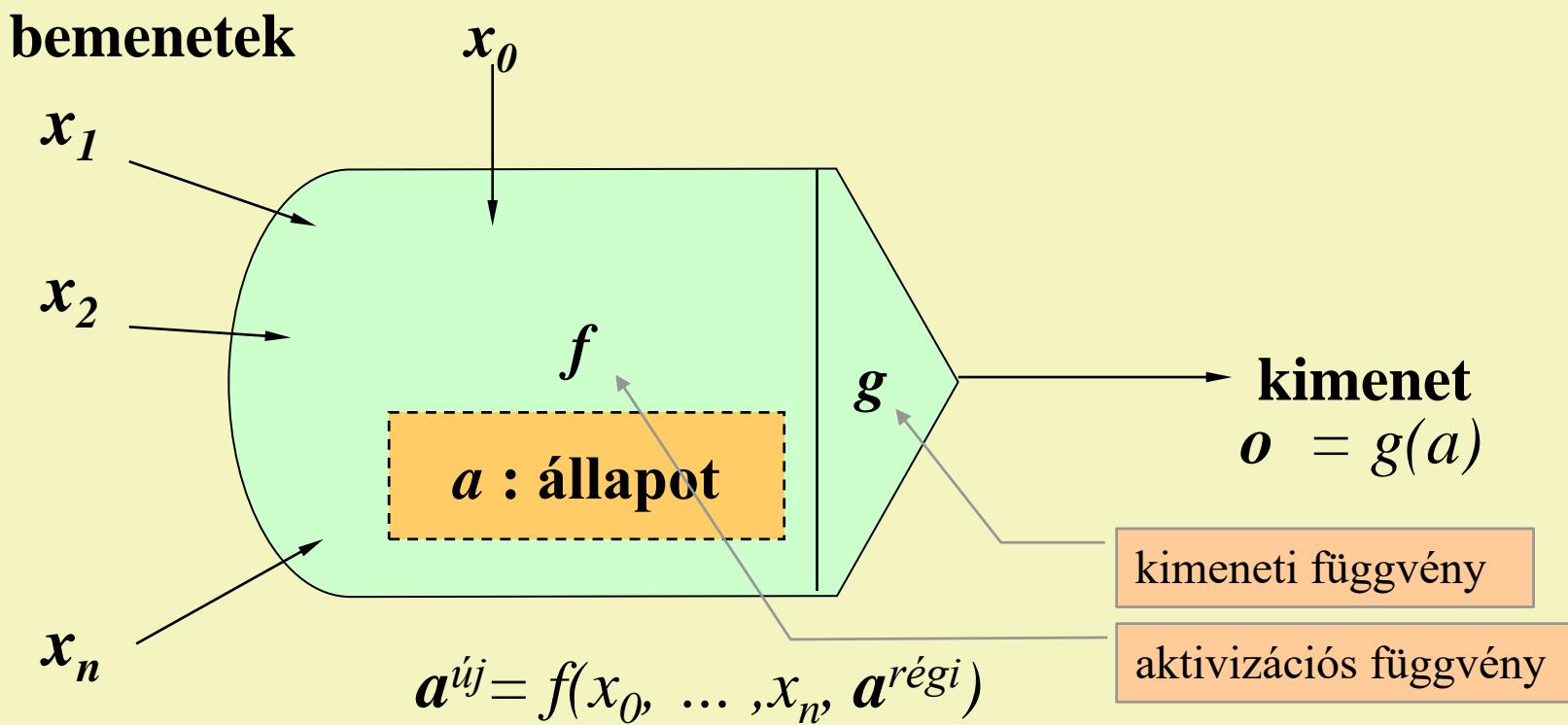
## ❑ Alkalmazás

- osztályozás, approximáció, optimalizálás, asszociatív memória,

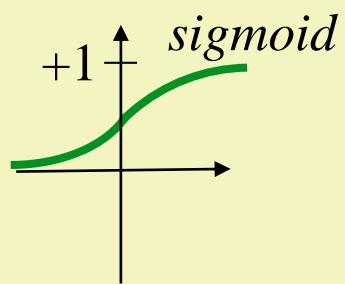
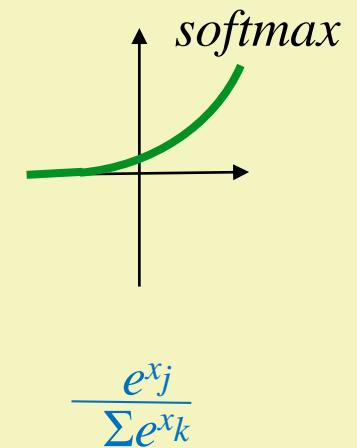
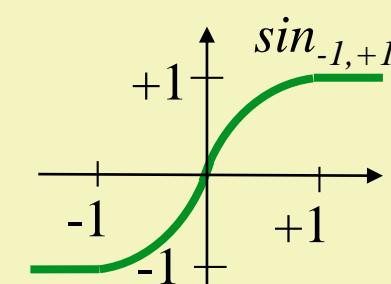
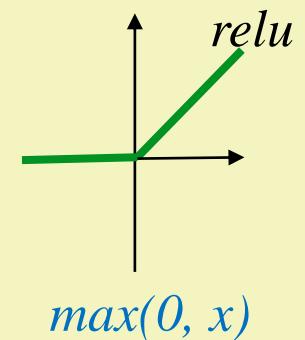
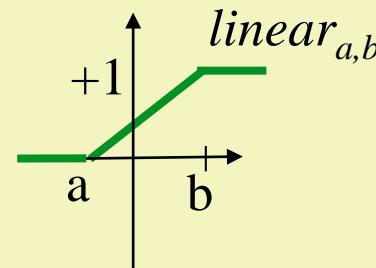
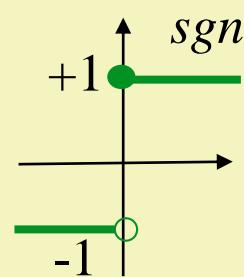
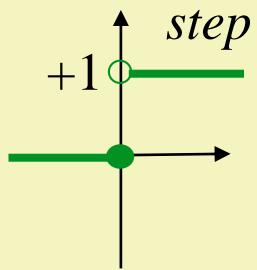
# *Természetes neuronhálók*



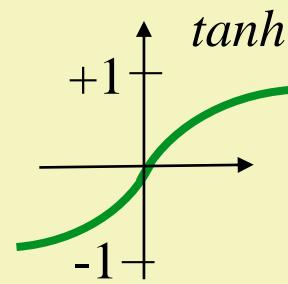
# *Általános mesterséges neuron*



# Kimeneti függvények

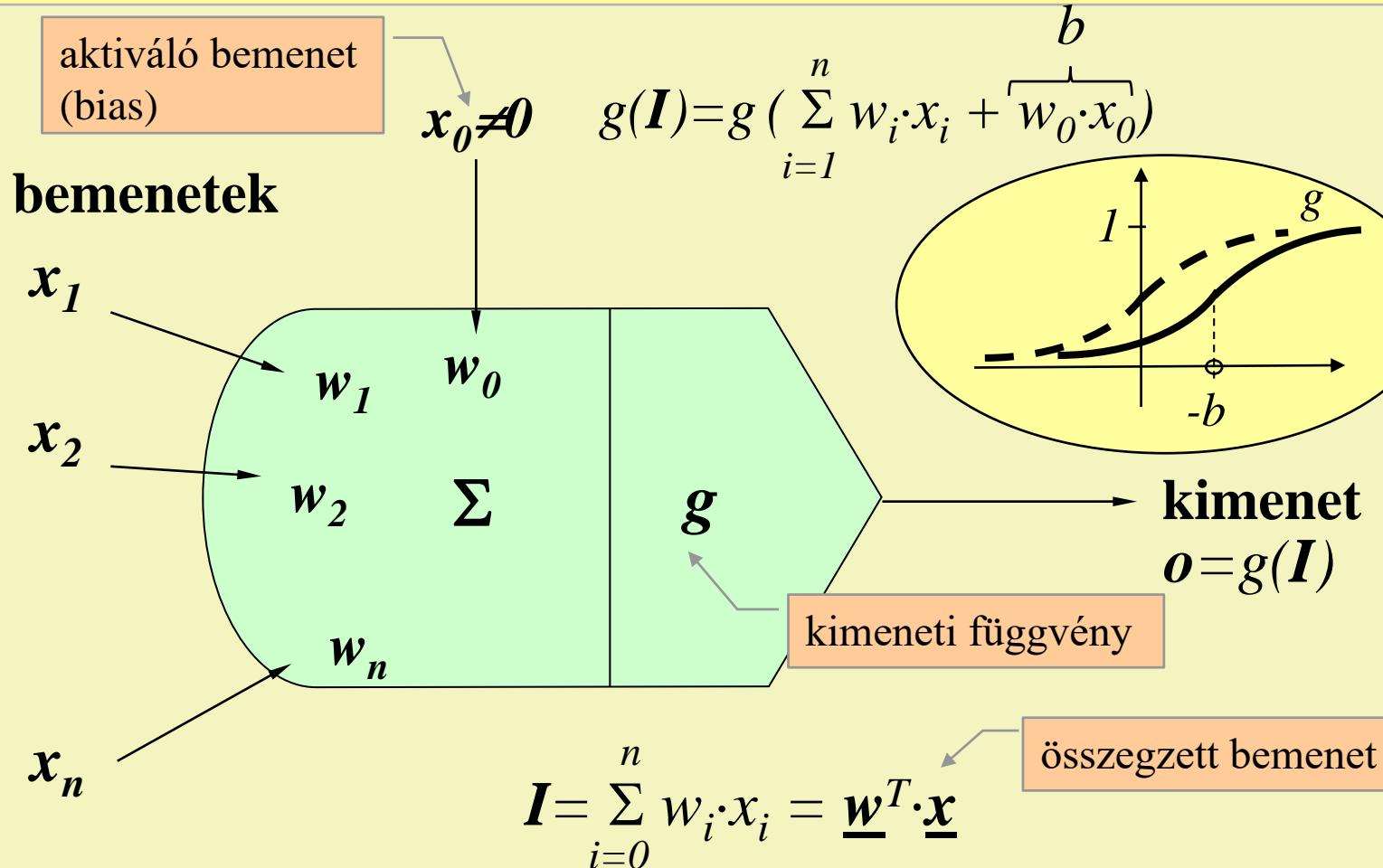


$$\frac{1}{1+e^{-x}}$$



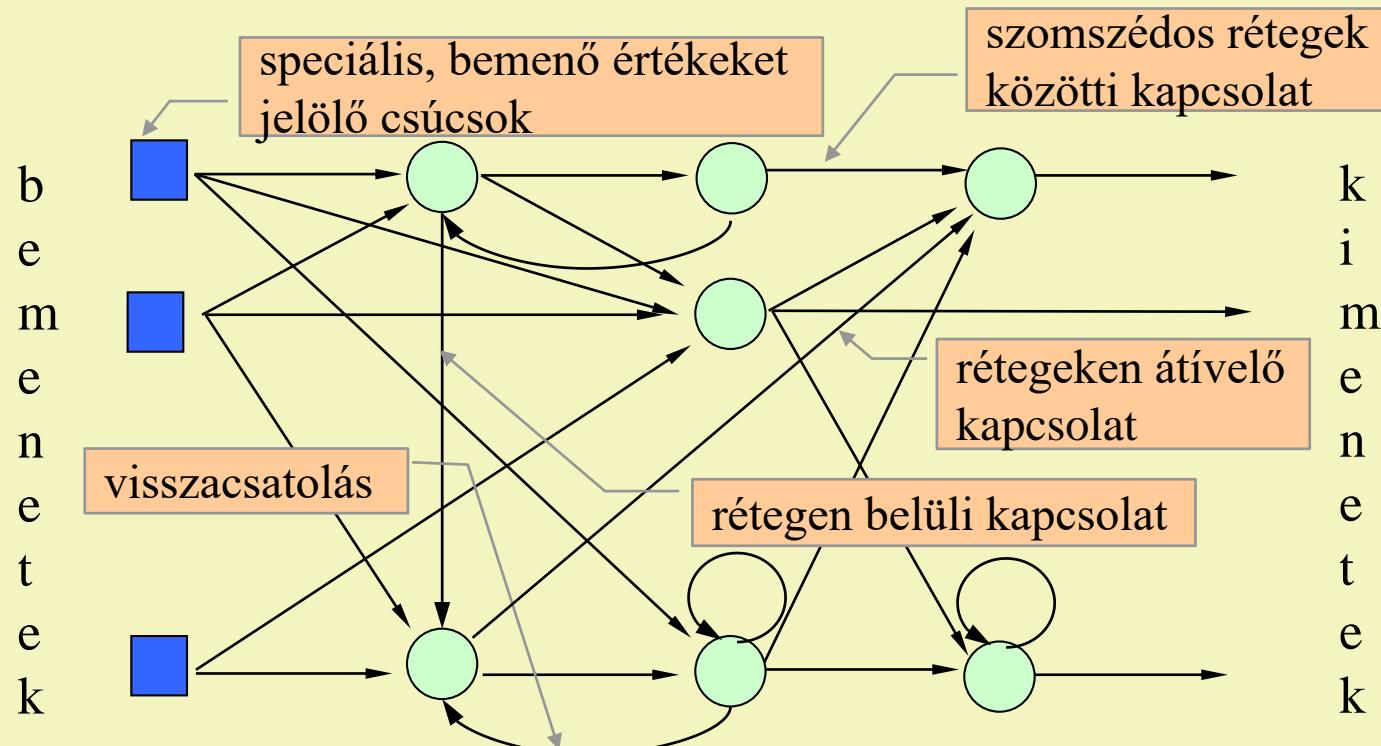
$$\frac{1-e^{-x}}{1+e^{-x}}$$

# Általánosított perceptron

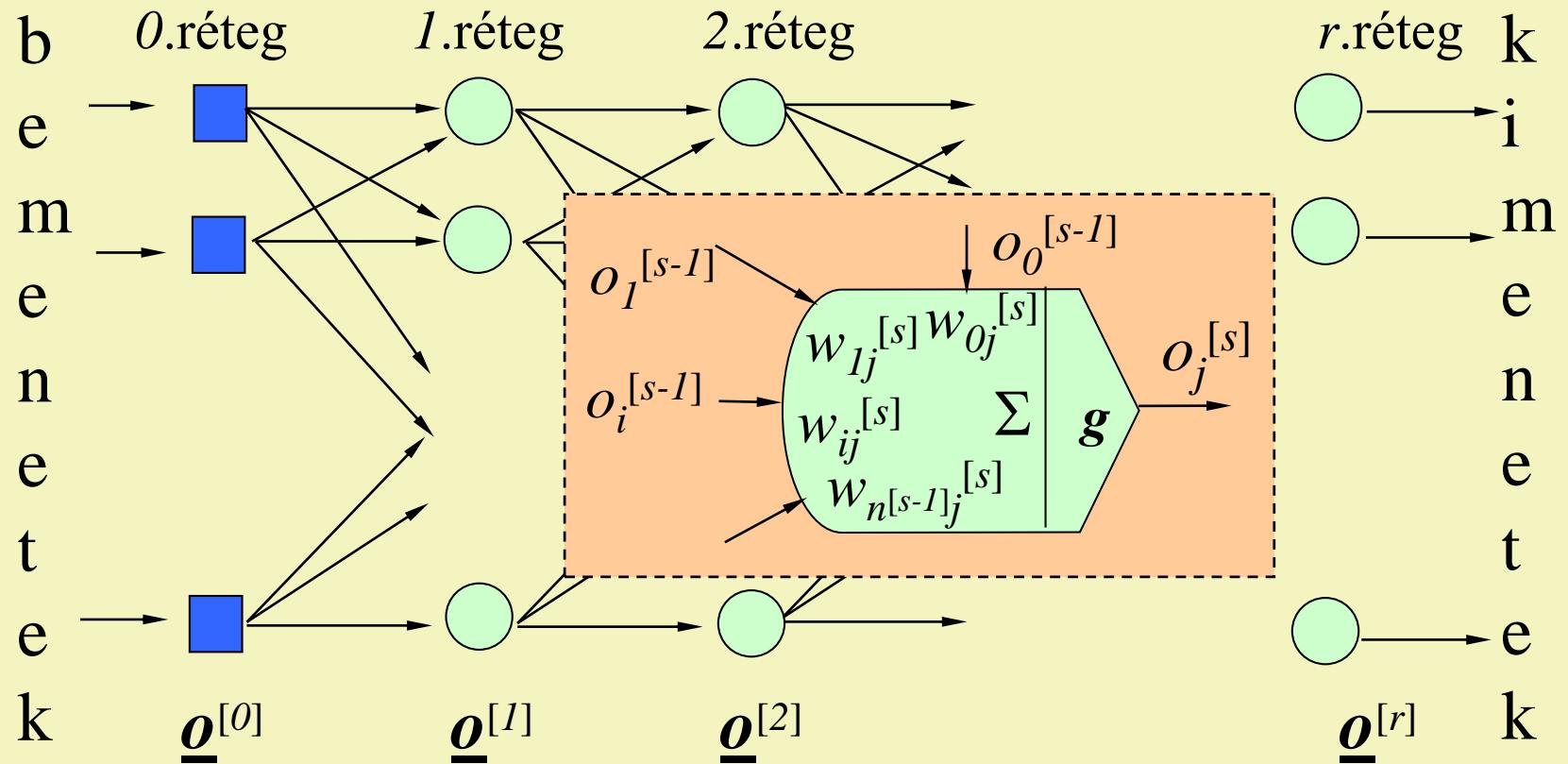


# Hálózati topológia

Irányított gráf, amelynek csúcsai mesterséges neuronok, amelyek rétegekbe csoportosíthatók. Az irányított élek az adatáramlás irányát jelölik:  
 $a \rightarrow b$  : az  $a$  neuron kimeneti értékét kapja meg a  $b$  neuron bemenetként.

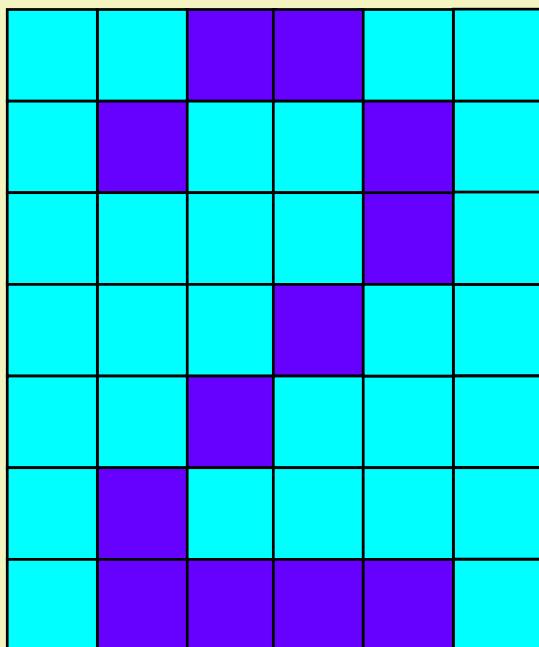


# Többrétegű előrecsatolt hálózat (feed forward MLP)



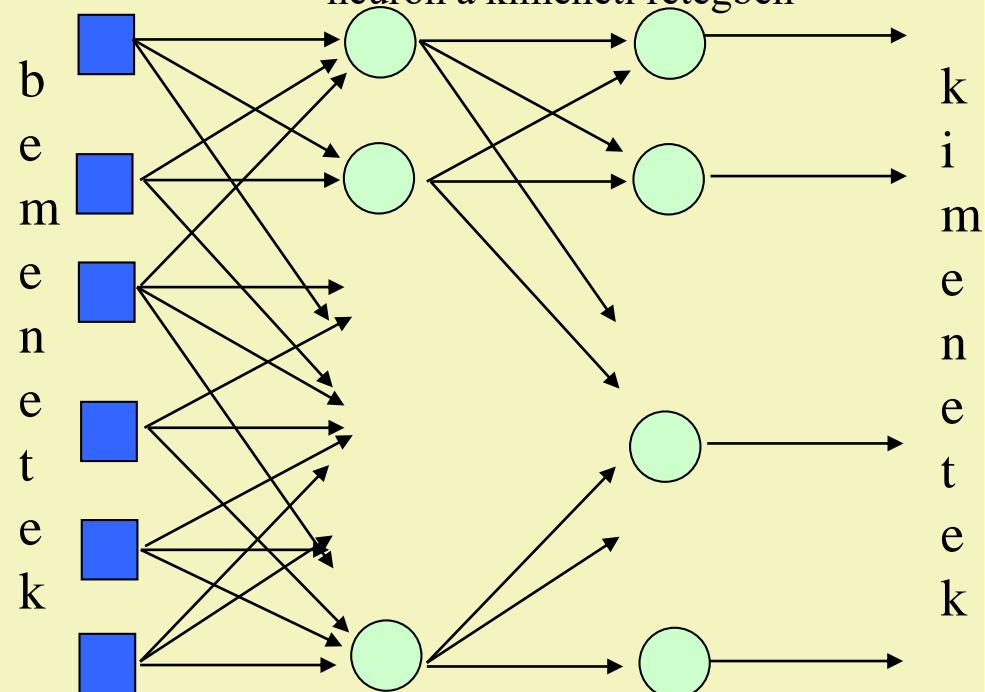
# Számjegy felismerés

Bemeneti értékek száma: 42



Kimenetek száma: 10

Minden számjegyhez tartozik egy neuron a kimeneti rétegben

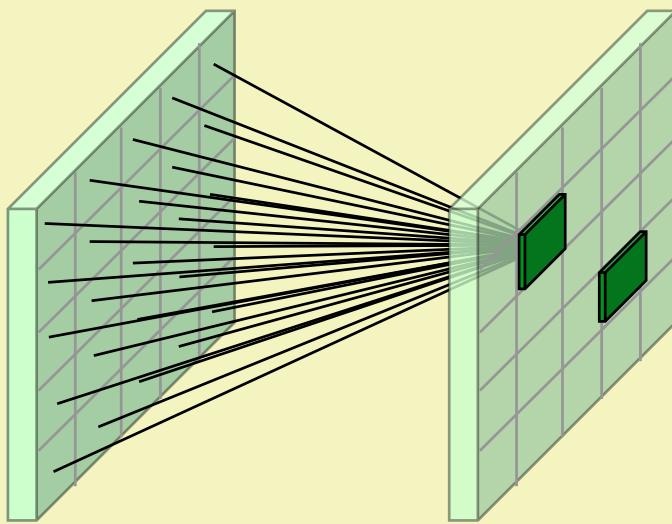


Beállítások: közbülső réteg neuronjainak száma: 11

$$x_i \in \{0,1\}, f(x) = \text{sigmoid}(x), o^s_i \in (0,1), w^s_{ij} = \text{rand}(-0.1, 0.1), o^s_0 = 1.0, \eta = 0.35$$

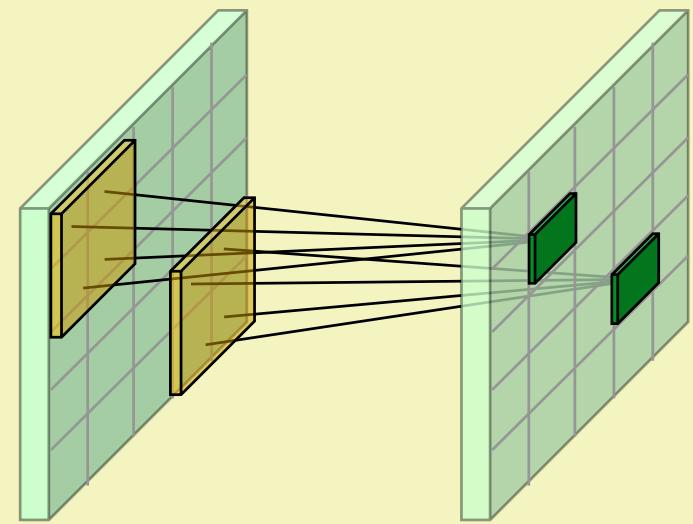
# *Konvolúciós neuron hálózat*

Teljesen összekötött (sűrű)  
fully connected neural net



Pl.: két  $1000 \times 1000$  réteg esetén  
a második réteg egy neuronjában  
 $10^6$  darab súlyt kell tárolni.

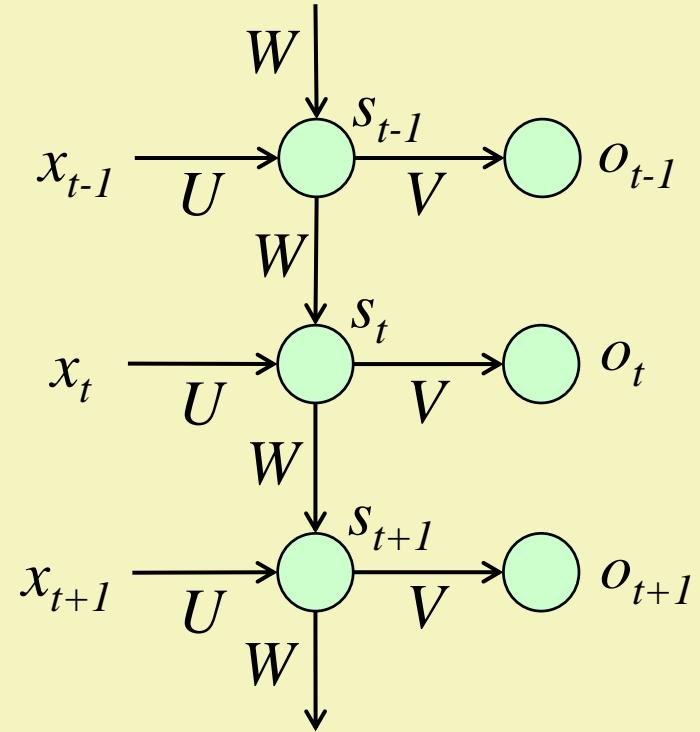
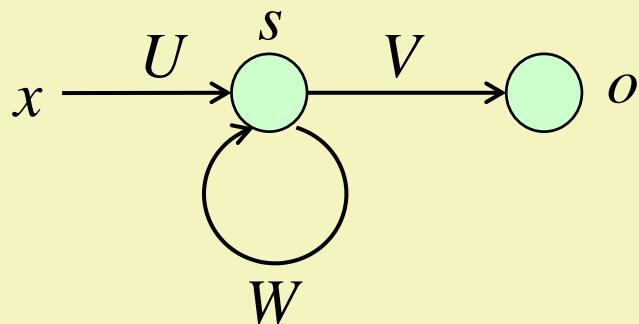
Lokálisan összekötött  
convolutional neural net



Pl.: két  $1000 \times 1000$  réteg esetén  
egy  $2 \times 2$ -es szűrőt használva  
a második réteg egy neuronjában  
csak 4 súlyt kell tárolni.

# *Rekurrens neurális hálózat*

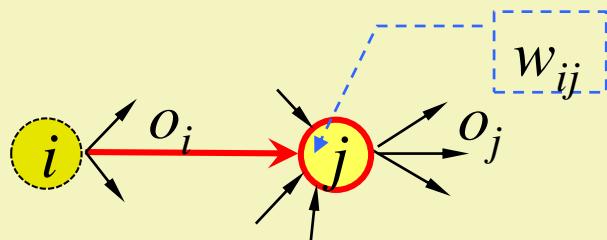
- ❑ Az input és/vagy az output változó hosszúságú sorozat.
- ❑ A számítás a belső memória segítségével emlékezik a megelőző inputokra.



# *Általánosított perceptron tanulása*

- Egy neuron számítási képletét a neuron  $w$  súlyai határozzák meg.
- A súlyok implicit módon a hálózat topológiáját is kijelölik, hiszen neuronhoz vezető nulla értékű súllyal ellátott él lényegében az él figyelmen kívül hagyását (törleszt) jelenti.
- Tanulás során a súlyokat fokozatosan módosítjuk ( $w := w + \Delta w$ ).
- A  $\Delta w$  a neuron **bemeneti értékeitől** és a neuron által **kiszámított kimeneti értéktől** függ.
  - Felügyelt tanulás esetén felhasználjuk az **elvárt kimenetet**.
  - Felügyelet nélküli tanulás esetén az elvárt kimenetre nincs szükség.

# Példák egy neuron súlyát módosító tanuló szabályokra



$o_i$  az  $i$  neuron számított kimenete és  
egyben a  $j$  neuron  $i$ -dik bemenete is  
 $w_{ij}$  a  $j$  neuron  $i$ -dik bemenetének súlya  
 $o_j$  a  $j$  neuron számított kimenete

## Felügyelt tanulás

Pl: Delta szabály:

$$\Delta w_{ij} = \eta \cdot o_i \cdot (y_j - o_j)$$

$y_j$  a  $j$ -dik neuron várt kimenete

## Felügyelet nélküli tanulás

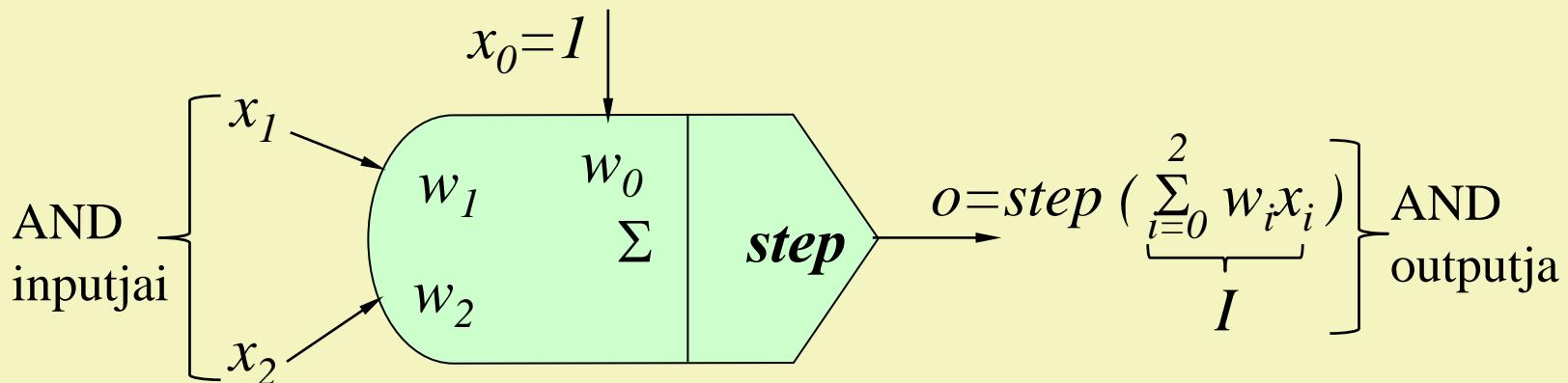
Pl: Hebb szabály:

$$\Delta w_{ij} = \eta \cdot o_i \cdot o_j$$

$\eta$  tanulási együttható a tanulás sebességét befolyásolja

# Példa felügyelt tanulásra

Tanítsuk meg egy egyszerű számoló egységnek (egyetlen mesterséges neuronnak) a logikai AND művelet működését!



$x_i$  a neuron  $i$ -dik bemenete ( $x_i \in \{0,1\}$ ,  $x_0 = 1$ )

$w_i$  a neuron  $i$ -dik bemenetének súlya ( $w_0, w_1, w_2 \in \mathbb{R}$ )

$I$  a neuron összegzett bemenete

$o$  a neuron számított kimenete ( $o \in \{0,1\}$ )

*i*-edik súly változása

felügyelt tanulási szabály:  $\Delta w_i = \eta \cdot x_i \cdot (y - o)$

hiba (*e*)

$y \sim$  várt

$o \sim$  számított

$\eta = 0.1$

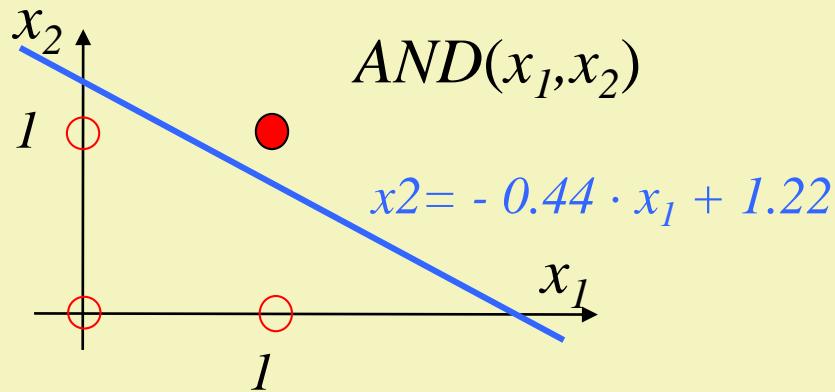
	$x_1$	$x_2$	$y$	$w_0$	$w_1$	$w_2$	$I$	$o$	$e$
1.	1	0	0	0.08	0.08	0.08			
				0.08	+ 0.08	+ 0	= 0.160	1	-1
2.	0	1	0	-0.02	-0.02	0.08			
				-0.02	+ 0	+ 0.08	= 0.06	1	-1
3.	1	1	1	-0.12	-0.02	-0.02			
				-0.12	+ -0.02	+ -0.02	= -0.16	0	1
	első epoch			-0.02	0.08	0.08			
4.	1	0	0	-0.02	0.08	0			
				-0.02	+ 0.08	+ 0	= 0.06	1	-1
5.	0	1	0	-0.12	-0.02	0.08			
				-0.12	+ 0	+ 0.08	= -0.04	0	0
6.	1	1	1	-0.12	-0.02	0.08			
				-0.12	+ -0.02	+ 0.08	= -0.06	0	1
...	második epoch								
13.				-0.22	0.08	0.18			
14.	1	0	0	-0.22	+ 0.08	+ 0	= -0.14	0	0
15.	0	1	0	-0.22	+ 0	+ 0.18	= -0.04	0	0
16.	1	1	1	-0.22	+ 0.08	+ 0.18	= 0.04	1	0
17.	0	0	0	-0.22	+ 0	+ 0	= -0.22	0	0

# Mit tanultunk meg?

- A példában a lehetséges bemenet-párok alkotta sík egy egyenesét, pontosabban az egyenes

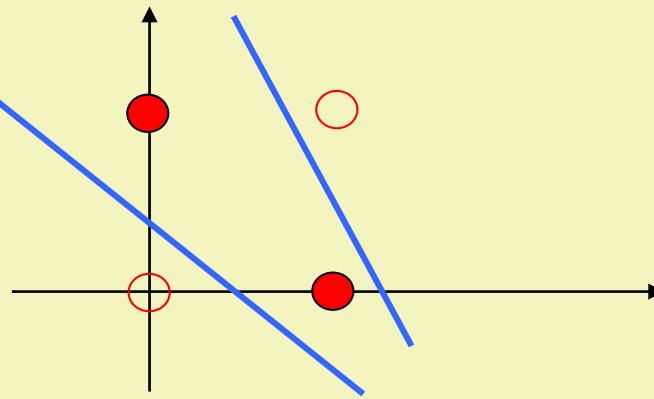
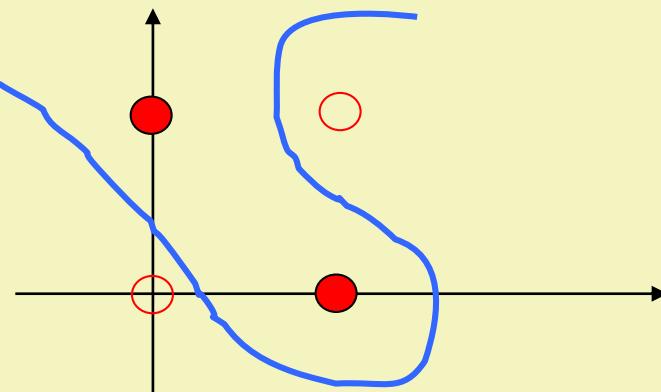
$I(x_1, x_2) = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2$  képletében szereplő  $w$  együtthatókat tanultuk meg. Ez az egyenes a lehetséges bemenet-párokat (sík bizonyos pontjait) két csoportra vágja.

- $I(x_1, x_2) \leq 0$  esetén a  $step(I(x_1, x_2)) = 0 = AND(x_1, x_2)$
- $I(x_1, x_2) > 0$  esetén a  $step(I(x_1, x_2)) = 1 = AND(x_1, x_2)$



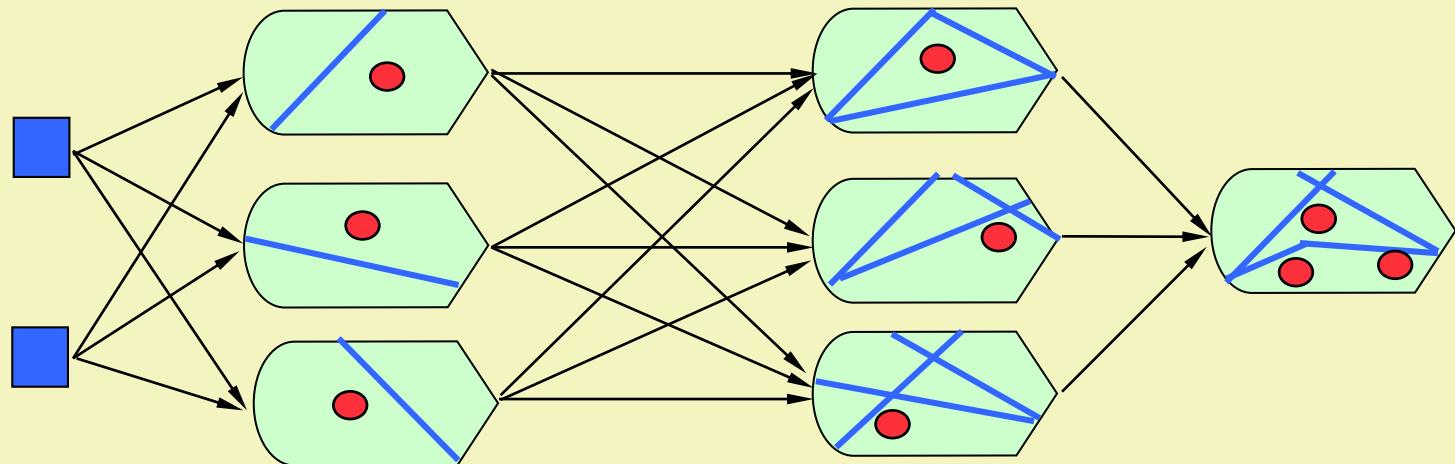
# *Lineáris szeparálhatóság*

- ❑ Egyetlen perceptronnal olyan bonyolultságú feladatot vagyunk képesek megoldani, ahol az eredményük alapján a bemeneteket úgy lehet két csoportba sorolni, hogy azokat egy hipersík választja szét, azaz **lineárisan szeparálhatók**.
- ❑ Nem lehet például a XOR műveletet egyetlen perceptronnal megvalósítani, mert ez a feladat lineárisan nem szeparálható.



# Rétegek „tudása”

- ❑ Egy egyrétegű perceptron modell neuronjai tehát csak féltereket képesek felismerni (osztályozni), de egy kétrétegű már ezek kombinációját, azaz konvex poliéderket is, egy három rétegű pedig tetszőleges poliéderket.



# Homogén MLP háló számítási modellje

## ❑ Egy réteg számítási modellje

$$o_j^{[s]} = g(\underline{w}_j^{[s]} \cdot \underline{o}^{[s-1]}) = g\left(\sum_{i=0}^{n^{[s-1]}} w_{ij}^{[s]} \cdot o_i^{[s-1]}\right)$$

*n<sup>[s-1]</sup>*

*s-1*-edik réteg neuronjainak száma =  
*s*-dik réteg neuronjainak bemenet száma

az *s*-edik réteg *j*-edik neuronjában  
az *i*-edik bemenethez tartozó súly

*s-1*-edik réteg *i*-dik neuronjának  
kimenete = az *s*-edik réteg  
neuronjainak *i*-dik bemenete is

## ❑ A teljes háló számítási modellje:

$$f(\Theta, \underline{x}) = g(\underline{w}^{[r]} \cdot \dots \cdot g(\underline{w}^{[s]} \cdot \dots \cdot g(\underline{w}^{[2]} \cdot g(\underline{w}^{[1]} \cdot \underline{x})) \dots) \dots)$$

a  $\Theta$  paraméter tehát a  $(w_{ij}^{[s]})$  súlyok összessége

# Hiba visszaterjesztés módszere (error backpropagation)

A modell hibafüggvénye:  $L(\Theta) = \frac{1}{2} \sum_{j=1}^n (y_j - o_j^{[r]})^2$

Az  $L(\Theta)$  egy olyan több változós függvény, amely a ( $w_{ij}^{[s]}$ ) súlyoktól függ. Ennek a függvénynek keressük a minimum helyét gradiens módszerrel:

$$w_{ij}^{[s]} := w_{ij}^{[s]} - \eta \cdot \frac{\partial L}{\partial w_{ij}^{[s]}}$$

$$\frac{\partial L}{\partial w_{ij}^{[s]}} = \eta \cdot \frac{\partial L}{\partial I_j^{[s]}} \cdot \frac{\partial I_j^{[s]}}{\partial w_{ij}^{[s]}} = \eta \cdot \frac{\partial L}{\partial I_j^{[s]}} \cdot o_i^{[s-1]}$$

hiszen  $I_j^{[s]} = \sum_{i=0}^n w_{ij}^{[s]} o_i^{[s-1]}$

a számítási hibának az  $s$ -edik réteg  $j$ -edik neuronjára jutó hányada

# *Backpropagation tanuló algoritmus*

1. Az  $\underline{x}$  bemeneti vektorból kiindulva rétegenként kiszámoljuk a neuronok kimeneteit:  $o_j^{[s]}$ , így eljutunk a kimeneti réteg kimeneteihez  $o_j^{[r]}$  is.
2. A kimeneti réteg minden neuronjára kiszámoljuk a lokális hibát:  $e_j^{[r]} = o_j^{[r]} \cdot (1 - o_j^{[r]}) \cdot (t_j - o_j^{[r]})$  ↪ ha  $g$  a szigmoid függvény
3. Rétegenként hátulról előre haladva számoljuk a belső neuronok hibáit:

$$e_j^{[s]} = o_j^{[s]} \cdot (1 - o_j^{[s]}) \cdot \left( \sum_{k=1}^{n^{[s+1]}} e_k^{[s+1]} \cdot w_{jk}^{[s+1]} \right)$$

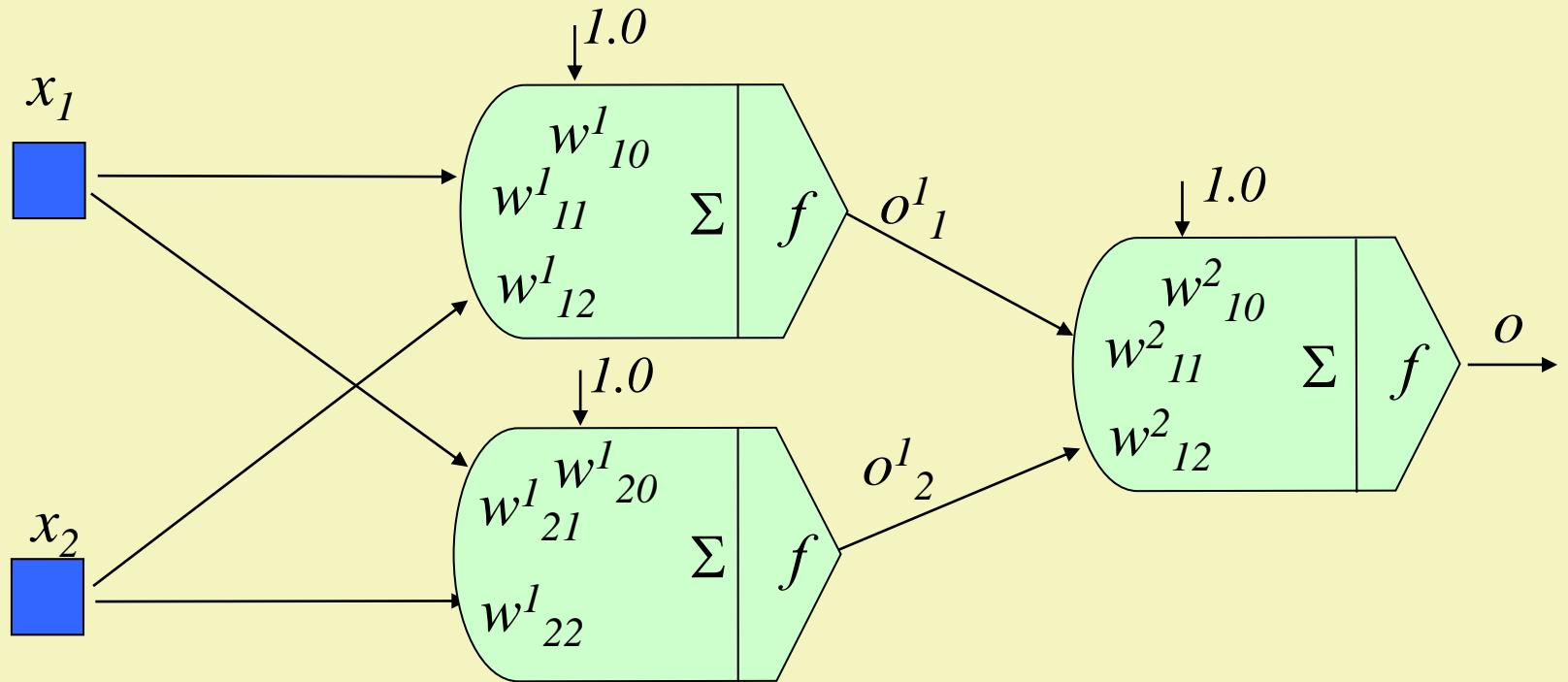
↗ ha  $g$  a szigmoid függvény

4. Végül módosítjuk a hálózat súlyait:  $w_{ij}^{[s]} := w_{ij}^{[s]} + \Delta w_{ij}^{[s]}$  ahol a súlytényező-változás:  $\Delta w_{ij}^{[s]} = \eta \cdot e_j^{[s]} \cdot o_i^{[s-1]}$

# *XOR művelet példája*

Beállítások:  $x_1, x_2 \in \{0, 1\}$   $f(x) = \text{logisztikus}(x)$   $o^s_i \in (0, 1)$

$$w_{ij}^s = \text{rand}(-0.1, 0.1) \quad o^s_0 = 1.0 \quad \eta = 1.0$$



# Rétegenként eltérő számítási képlet

- Egy neuronháló számítási modellje:

$$f : P \times X \rightarrow Y$$

$$f(\Theta, \underline{x}) = g_r(\underline{\underline{w}}^{[r]}, g_{r-1}(\dots g_2(\underline{\underline{w}}^{[2]}, g_1(\underline{\underline{w}}^{[1]}, \underline{x})) \dots))$$

- $\Theta = \{w^{[1]}, \dots, w^{[r]}\}$  azaz a paraméterek a súlyok
- $g_s : P \times X^{s-1} \rightarrow X^s$   $s$ -edik réteg kimeneti függvénye  
( $X = X^0$ ,  $Y = X^r$ )

- Hibafüggvény:

$$L(\Theta) = \frac{1}{N} \sum_{n=1}^N \ell \left( \underbrace{f(\Theta, x_n)}_{t_n}, y_n \right)$$

$\ell : Y \times Y \rightarrow \mathbb{R}$  hibafüggvény:  $\ell(t_n, y_n)$  lehet például  $\|t_n - y_n\|_1$ ,  $\|t_n - y_n\|_2^2$ , vagy  $-\sum_i y_n i \cdot \log t_n i$

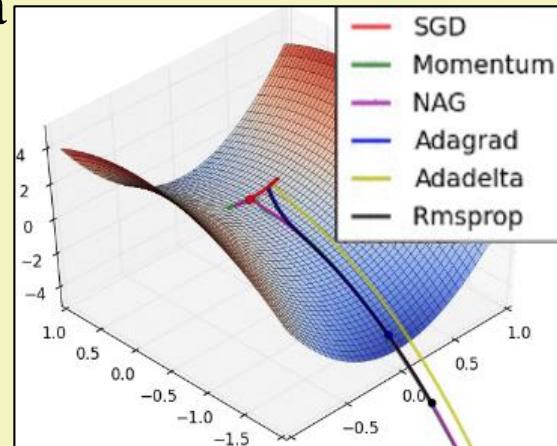
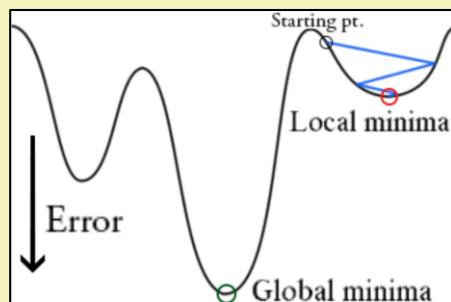
# Gradiens módszer

- ❑ A gradiens elméleti kiszámítása nehéz:

$$\Theta_{\text{új}} := \Theta_{\text{régi}} - \eta \cdot \frac{\partial L(\Theta)}{\partial \Theta} \quad \Bigg|_{\Theta = \Theta_{\text{régi}}}$$

ezért helyette numerikus módszereket használnak.

- ❑ A lokális minimum-, illetve a nyeregpontok problémája:



- ❑ A tanító minták kiválasztása nehéz (homogén minták, overfitting)
- ❑ A hiperparaméterek ( $N$ ,  $\eta$ ) megtanulásának kérdése

A mélytanulás módszerei ezeken a hátrányokon igyekeznek javítani.

# Gépi Tanulás Előadás 2

## Felügyelt Tanulás: Mesterséges Mély Neurális Hálózatok

Milacski Zoltán Ádám<sup>1</sup>

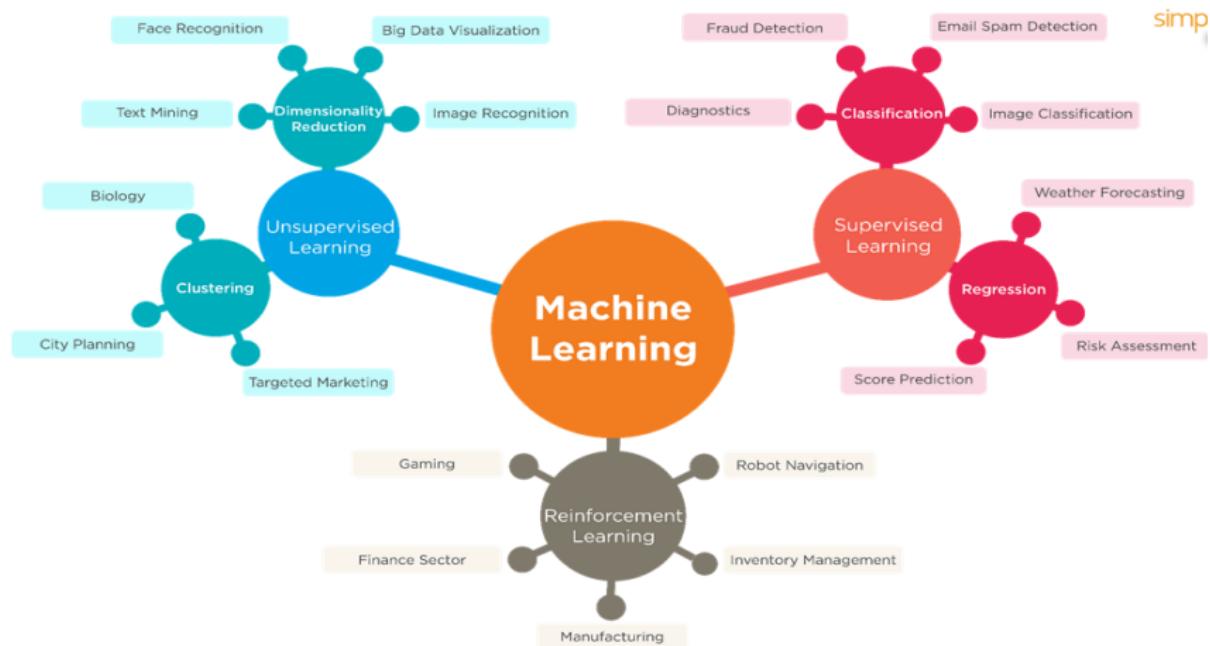
<sup>1</sup>Eötvös Loránd Tudományegyetem  
Programozáselmélet és Szoftvertechnológiai Tanszék  
[srph25@gmail.com](mailto:srph25@gmail.com)

2019. május 8.



# Gépi Tanulás

## Felügyelt, Felügyeletlen és Megerősítéses



- Múlt órán: Felügyelt, KNN, RF;
- Ma: Felügyelt, Mesterséges Mély Neurális Hálózatok (ANN, DNN);
- Jövő órán: Felügyeletlen, Megerősítéses.



# Felügyelt Tanulás

Formálisan: Optimalizációs Feladat

- Adottak az  $(x_n, y_n), n = 1, \dots, N$  tanítópárok, keressük az optimális  $\theta^*$  paraméterbeállítást az  $f(\theta, \cdot)$  paraméteres leképezéshez úgy, hogy  $f(\theta^*, x_n) \approx y_n$  (közelítsük az  $x_n \mapsto y_n$  leképezést):

$$\min_{\theta} L(\theta, x_n, y_n) = \frac{1}{N} \sum_{n=1}^N \ell\left(\underbrace{f(\theta, x_n)}_{\hat{y}_n}, y_n\right),$$

ahol  $\ell(\hat{y}_n, y_n)$  a hibafüggvény és  $\theta$  a paramétervektor.

Becslés ( $f(\theta^*, x_n)$  kiszámítása adott  $\theta^*$ -ra) nagyon gyors!

Tanítás (minimalizáló  $\theta^*$  megkeresése) nagyon lassú!

Általában: nemkonvex optimalizálási feladat, NP-nehéz a  $\theta^*$  globális optimumot megtalálni.

- Jó hírek: működik, de csak ha...

- ...  $N$  elég nagy! Az  $y_n$ -ek összegyűjtése drága humán munka... (UL?),
- ...  $\ell(\hat{y}_n, y_n)$ ,  $f(\theta, x_n)$  és az optimalizálási módszer megfelelően vannak megválasztva! Nehéz humán munka, sok kísérlet... (UL?),
- ...  $\theta$  közel van  $\theta^*$ -hoz! Egyelőre nem tudjuk bizonyítani, de megy, így feltehetően igaz... ( $\theta^*$  egyébként is túl mohó és túltanulásra vezet...).



# Paraméteres leképezés és Optimalizálási módszer

## Összehasonlítás

Paraméteres leképezés és Optimalizálási módszer szorosan összetartozi.

Összehasonlítás:

Módszer	$\theta$	$f(\theta, x_n)$	$\min_{\theta}$	Megjegyzés
KNN	$(x_n, y_n), n = 1, \dots, N$	$\sum_{n=1}^N w_n^{KNN} y_n$	nincs, $\theta^*$ ismert	nem tömörít, lassú, glob. opt. $\theta^*$ , $K$ -ra érzékeny
RF	legj. vágó vált., $t$ küszöbök	$\sum_{n=1}^N w_n^{RF} y_n$	mohó, faépítés	tömörít, gyors, nódusra opt., össz. szubopt. $\theta^*$
DNN	$w_j$ súlyok	$h(\sum_{j=1}^J w_j x_{n,j} + b)$ többsz. össz. fv.	gradiens- módszer	tömörít, gyors, össz. lok. opt. glob. opt $\theta^*$ ?

# Mesterséges Mély Neurális Hálózatok

- Mesterséges Mély Neurális Hálózat (DNN): Összetett függvény alakú paraméteres leképezés:

$$f(\theta, x_n) = g_K(\theta_K, g_{K-1}(\dots g_2(\theta_2, g_1(\theta_1, x_n)) \dots))$$

ahol  $\theta = \{\theta_1, \dots, \theta_K\}$  a parameterek (súlyok). Nemkonvex, mert  $\theta_i$  és  $\theta_j$  ( $i \neq j$ ) szorzata megjelenik  $f(\theta, x_n)$ -ben.

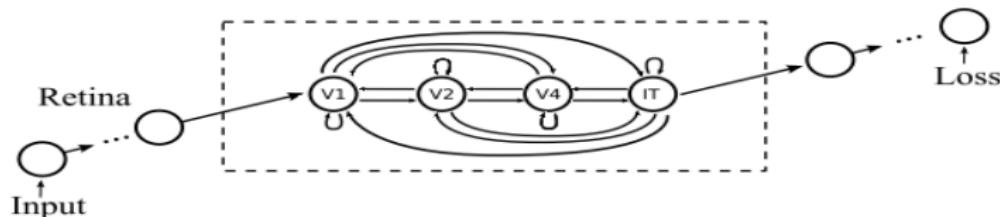
Réteg:  $x_n^{(k)} = g_k(\theta_k, x_n^{(k-1)})$

Hálózat (előreterjesztés):  $f(\theta, x_n)$ .

A rétegek alacsonyabb szintű leíró változókat kombinálnak össze magasabb szintűekké az összetett függvény alak miatt.

A főemlősök látórendszerét utánozza:

Pre-net



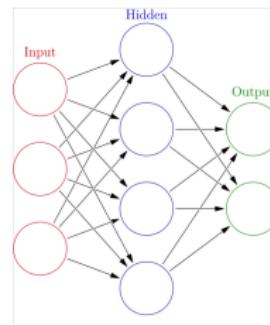
Post-net

# Mesterséges Mély Neurális Hálózatok

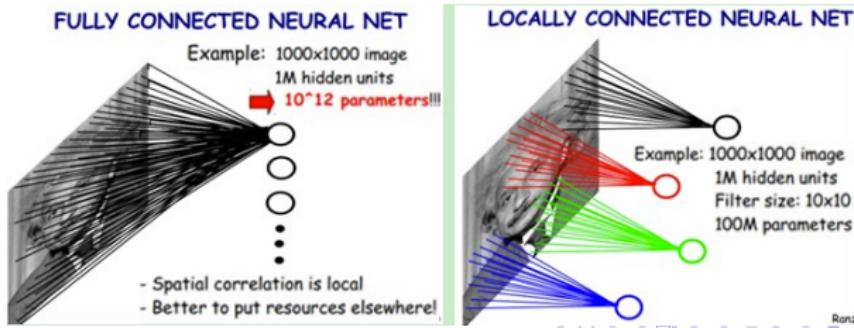
## Rétegek és Nemlineáritások

- Rétegek:

- Sűrű (Teljes Konnektivitású, 2D vektorok):  $x_n^{(k)} = h_k(W_k x_n^{(k-1)} + b_k)$



- Konvolúciós (Lokális Konnektivitású, 4D képek):  $x_n^{(k)} = h_k(W_k * x_n^{(k-1)} + b_k)$

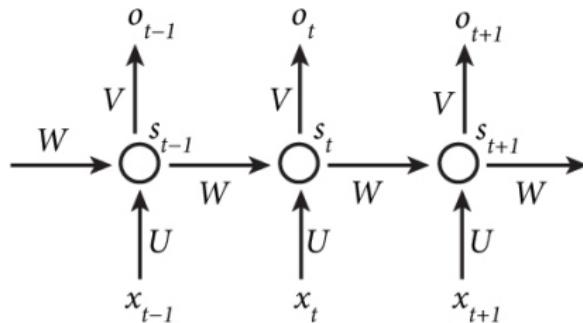


# Mesterséges Mély Neurális Hálózatok

## Rétegek és Nemlinearitások

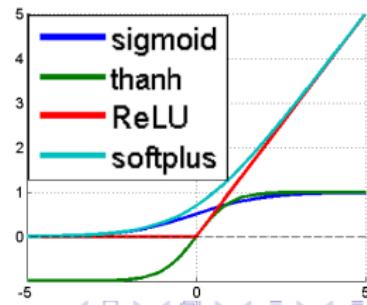
- Rétegek:

- Rekurrens (3D idősorok):  $x_{n,t}^{(k)} = h_k(W_k^{hh}x_{n,t-1}^{(k)} + W_k^{xh}x_{n,t}^{(k-1)} + b_k)$



- Nemlineáritások:  $h_k$  elemenkénti nemlineáris függvény a rétegekben.

- sigmoid:  $h_k(z) = \frac{1}{1+e^{-z}}$ ,
- tanh:  $h_k(z) = \tanh(z)$ ,
- relu:  $h_k(z) = \max(0, z)$ ,
- softmax:  $h_k(z)_j = \frac{e^{z_j}}{\sum_{i=1}^l e^{z_i}}$ .



# Mesterséges Mély Neurális Hálózatok

## Optimalizálási Módszer: Gradiens-módszer

Keressük  $\theta^*$ -ot!  $f(\theta, x_n)$  deriválható  $\theta$  szerint, így  $L(\theta, x_n, y_n)$  is.

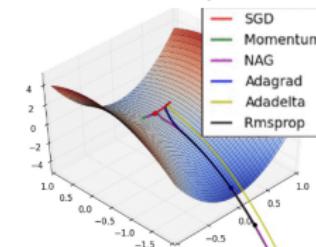
- Sztochasztikus Gradiens-módszer (SGD): Negatív gradiens a lokális optimum fele mutat, lépjünk picit ebbe az irányba! Legyen  $\theta_0 \sim \mathcal{N}(0, 0.001^2)$ , ezután:

$$\theta_{I+1} := \theta_I - \alpha \frac{\partial L(\theta, x_n, y_n)}{\partial \theta} \Big|_{\theta=\theta_I}.$$

Tanulási ráta (lépésköz):  $\alpha$ , pici szám, pl. 0.001.

Visszaterjesztés:  $\frac{\partial L(\theta, x_n, y_n)}{\partial \theta}$ , automatikus gépi deriválással (láncszabály...).

Minibatch:  $(x_n, y_n)$  párok véletlen részhalmaza minden lépésben.



**Elakadhat nyeregpontokban!**

- Kvázi-Newton módszerek: ki tudnak mozdulni nyeregpontokból adaptív tanulási rátákkal, pl. RMSProp, Adadelta, Adam.

# Mesterséges Mély Neurális Hálózatok

## Szoftvereszközök

Használunk **GPU**-t gyorsasághoz: a tenzorműveletek zavarbaejtően párhuzamosak. SGD megfelelő kevés memóriához.

**Probléma:** GPU programozás túl alacsony szintű (lassú és hibákkal teli fejlesztés)...

**Megoldás:** kódolunk magas szinten (pl. Python-ban) és fordítsuk le alacsony szintű GPU kódra!

Ehhez speciális szoftvereszközök szükségesek:

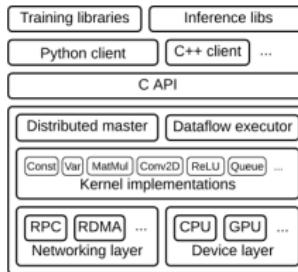
- CUDA, OpenCL: Alacsony szintű, GPU kód
- **Tensorflow**, Theano, CNTK, PyTorch: Közepes szintű (Back-end), szimbolikus előreterjesztés, automatikus szimbolikus deriválás, fordítás GPU kódra és meghívás konkrét numerikus értékekkel
- **Keras**: Magas szintű (Front-end), nemlineáritások, rétegek, hibafüggvények, gradiens-módszerek
- OpenAI Gym: Megerősítéses Tanulás framework
- Hyperopt: Hiperparaméter keresés
- Sacred: Kísérletek logolása és reprodukciója
- Elasticcluster: Elosztott számítások felhőben



# Mesterséges Mély Neurális Hálózatok

## GPU programozás Tensorflow-ban

- Fejezzük ki algoritmusunkat szimbolikus formában, **számítási gráf** felépítésével
- Építés fázis:
  - Tensor: típusos többdimenziós tömb (statikus típus, dimenzió, méret).
  - Operation (op): kap nulla vagy néhány tenzort, számol velük, majd visszaad nulla vagy néhány tenzort (szimbolikus gradiense implementálva van).
  - Variable: állapotok tárolása több hívás (végrehajtás) között (tf.assign op).
  - Placeholder: bemenetek, kijelölik a 'feed' műveleteket.
- Végrehajtási fázis:
  - Session: műveleteket eszközre (CPU vagy GPU) helyezi, metódusokat ad a végrehajtásukhoz, tenzorokat ad vissza numpy ndarray-ként.
  - Fetch: műveletek kimeneteinek kinyerése, gráf végrehajtása.
  - Feed: művelet kimeneteinek lecserélése konkrét tenzor értékre.



```
input1 = tf.placeholder(tf.float32)
input2 = tf.placeholder(tf.float32)
output = tf.mul(input1, input2)

with tf.Session() as sess:
    print(sess.run([output], feed_dict={input1:[7.], input2:[2.]}))

# output:
# [array([ 14.], dtype=float32)]
```



# Mesterséges Mély Neurális Hálózatok

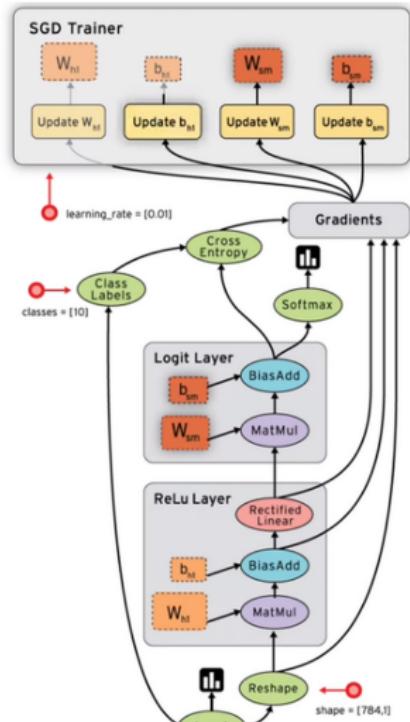
Mély Hálók implementációja Tensorflow-ban

## Hogyan csinálunk Mesterséges Mély Neurális Hálózatot Tensorflow-ban?

Name	Math	Tensorflow
Bemenet, kimenet	$x_n, y_n, n = 1, \dots, N$	
Paraméter	$\theta$	tf.placeholder
Előreterjesztés	$f(\theta, x_n)$	tf.Variable
Hibafüggvényion	$L(\theta, x_n, y_n) = \frac{1}{N} \sum_{n=1}^N l(f(\theta, x_n), y_n)$	tf.losses ops
Szimbolikus gradiens	$\frac{\partial L(\theta, x_n, y_n)}{\partial \theta}$	tf.gradients op
Inicializáció	$\theta_0$	tf.random_normal_initializer
Gradiens-módszer	$\theta_{l+1} := \theta_l - \alpha \frac{\partial L(\theta, x_n, y_n)}{\partial \theta} \Big _{\theta=\theta_l}$	tf.train.Optimizer
Tanít, tesztel, becsül	$x_n := X_n, y_n := Y_n$	tf.Session.run(), fetch, feed

**Előny:** Az  $\frac{\partial L(\theta, x_n, y_n)}{\partial \theta}$  szimbolikus gradiens automatikusan számolható a tf.gradients op által (láncszabály többszöri ismétlésével a számítási gráfon, ahol minden op-nak ismert a gradiense; ezt rendkívül nehéz lenne papíron levezetni). Ez nagyban egyszerűsíti a kísérletezgetést (csak az Előreterjesztést kell cserélgetni, ami könnyű!).

**Hátrány:** Nehéz debug-olni.



# Mesterséges Mély Neurális Hálózatok

## Mély Hálók implementációja Keras-ban

**Probléma:** Tensorflow-ban rendre ugyanazokat a számítási részgráfokat (rétegek, nemlineáritások) kell megadni, újra implementálhatni őket felesleges. Ezek magasabb absztrakciós szintet képviselnek.

**Megoldás:** Keras magasabb szinten rendszerezi őket objektum-orientáltsággal (osztályok és öröklődés)!

- Könnyű és gyors prototípus gyártás (felhasználóbarát, moduláris, bővíthető).
- Sok beépített réteg osztály, amik kombinálhatóak is.
- Használhat TensorFlow-t, CNTK-t vagy Theano-t a háttérben (keras.backend).
- Csak Python (nincs szükség egyéb konfigurációs fájlokra, így bővíthető).
- Egyszerű keras.models.Model metódusok tanításra, tesztelésre, becslésre: fit(), evaluate(), predict().

```
from keras.models import Sequential
model = Sequential()
from keras.layers import Dense, Activation
model.add(Dense(units=64, input_dim=100))
model.add(Activation('relu'))
model.add(Dense(units=10))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
# x_train and y_train are Numpy arrays --just like in the Scikit-Learn API.
model.fit(x_train, y_train, epochs=5, batch_size=32)
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)
classes = model.predict(x_test, batch_size=128)
```



# Mesterséges Mély Neurális Hálózatok

## Megoldatlan Kérdések



**Algorithm 1:** The layered thresholding algorithm.

Assuming two layers for simplicity, Algorithm 1 can be summarized in the following equation

$$\mathbf{f}_2 = \mathcal{P}_{\beta_2} \left( \mathbf{D}_2^T \mathcal{P}_{\beta_1} \left( \mathbf{D}_1^T \mathbf{X} \right) \right).$$

Comparing the above with Equation (1), given by

$$f(\mathbf{X}, \{\mathbf{W}_i\}_{i=1}^2, \{\mathbf{b}_i\}_{i=1}^2) = \text{ReLU} \left( \mathbf{W}_2^T \cdot \text{ReLU} \left( \mathbf{W}_1^T \mathbf{X} + \mathbf{b}_1 \right) + \mathbf{b}_2 \right),$$

one can notice a striking similarity between the two.

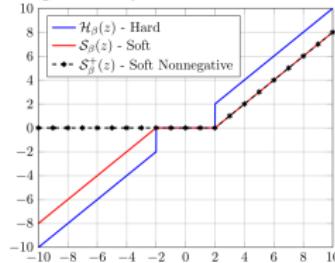


Figure 3: The thresholding operators for a constant  $\beta = 2$ .

- **Tételek:** ekvivalencia konvolúciós ReLU hálók és ritka reprezentáció között (utóbbira sok bizonyított téTEL van, pl. globálisan optimális  $\theta^*$ -ra). De lesz több téTEL is hamarosan... .
- Jobb rétegek: Kapszulák (Hinton) magasabb rendű invariáns változókkal.
- Jobb fejlesztői eszközök: imperatív programozás, debug-olás... .

# Ajánlott Irodalom

- Online Kurzusok

- Vincent Vanhoucke kurzusa: Tensorflow
- Andrew Ng kurzusa: Gép Tanulás Bevezető
- NVIDIA kurzusa: Szoftvereszközök
- Geoffrey Hinton kurzusa: Elmélet
- Andrej Karpathy kurzusa: Konvolúciós Hálók
- Stephen Boyd kurzusa: Konvex Optimalizálás
- Georgia Tech kurzusa: Megerősítéses Tanulás

- Forráskódok

- Keras útmutató
- Keras példák
- Keras GitHub (haladó)

- Tudományos Cikkek

- Google Scholar (haladó): Yann LeCun, Joshua Bengio, Geoffrey Hinton, Ilya Sutskever, Christian Szegedy, Alex Krizhevsky, Andrew Ng, Quoc Le, Vincent Vanhoucke, Diederik Kingma



# Alkalmazások

Youtube videók



# Nem felügyelt tanulás

Pintér Balázs

2019-05-15

# Tartalom

## 1 Bevezetés

## 2 Klaszterezés

- Hard clustering – k-means
- Soft clustering – témamodellek

## 3 Dimenziócsökkentés

- Kovariancia, korreláció
- Főkomponens analízis

## 4 Autoenkóderek

# Tartalom

## 1 Bevezetés

## 2 Klaszterezés

- Hard clustering – k-means
- Soft clustering – témamodellek

## 3 Dimenziócsökkentés

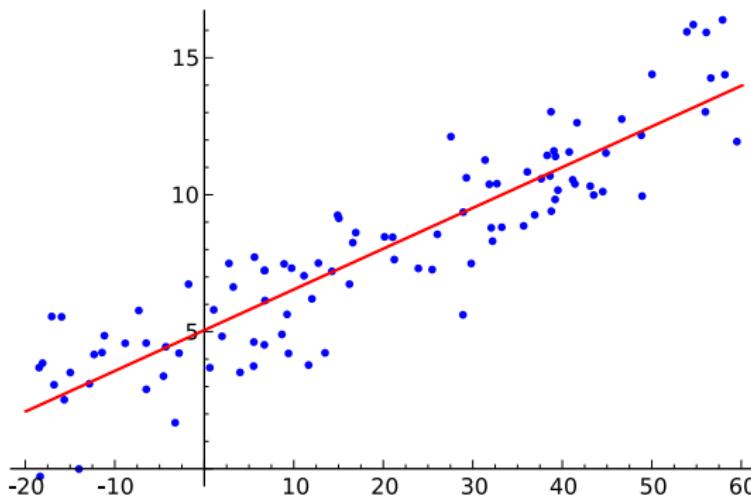
- Kovariancia, korreláció
- Főkomponens analízis

## 4 Autoenkóderek

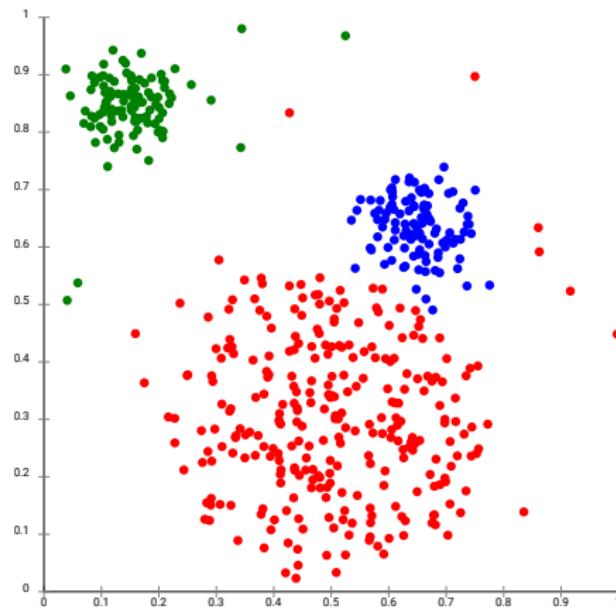
# Felügyelt tanulás – osztályozás

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3  
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4  
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5  
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6  
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7  
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8  
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

# Felügyelt tanulás – regresszió



# Nem felügyelt tanulás – klaszterezés



# Nem felügyelt tanulás

- Felügyelt tanulás: címkézett adatokból tanulunk valamilyen függvényt
- Más megközelítések
  - 1 **Nem felügyelt tanulás**
  - 2 Semi-supervised learning
  - 3 Megerősítéses tanulás
  - 4 Evolúciós algoritmusok
  - 5 Neuroevolúció

<http://www.youtube.com/watch?v=qv6UV0Q0F44>

# Tartalom

## 1 Bevezetés

## 2 Klaszterezés

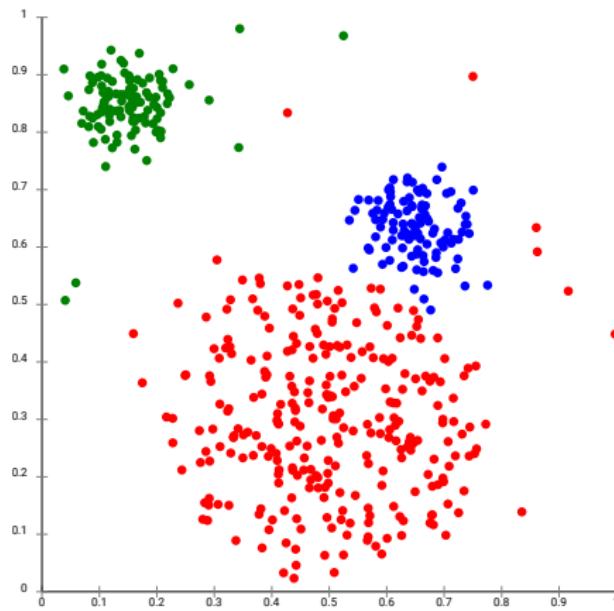
- Hard clustering – k-means
- Soft clustering – témamodellek

## 3 Dimenziócsökkentés

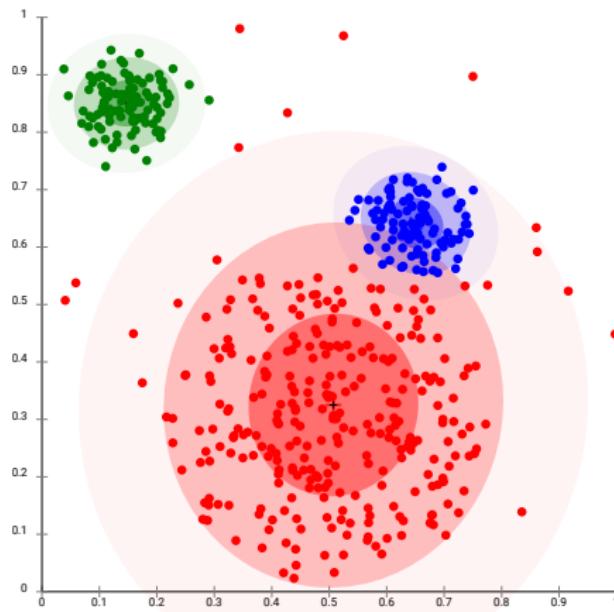
- Kovariancia, korreláció
- Főkomponens analízis

## 4 Autoenkóderek

## Példa – eloszlás alapú klaszterezés



## Példa – eloszlás alapú klaszterezés



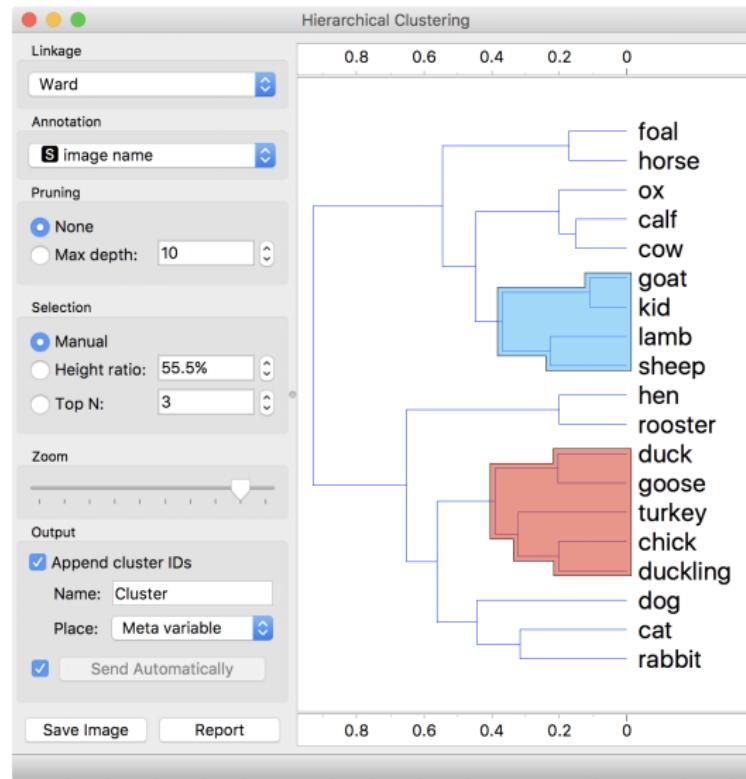
# Feladat

- Úgy csoportosítunk dolgokat, hogy a hasonlóak egy csoportba kerüljenek
  - Klaszteren belül minél hasonlóbbak
  - Klaszterek között minél kevésbé hasonlóak
- A dolgok általában  $\mathbb{R}^n$ -beli (vagy gráfbeli) pontok, pl.:
  - Ügyféladatok piacssegmentáláshoz
  - Dokumentumok szózsákkal modellezve, témák meghatározásához, keresési találatok összegzésére
  - Szavak kontextusai, jelentések indukálásához
  - Szerverek adatai (melyikek aktívak általában együtt)
- Egy csoportot egy *klaszternek* hívunk

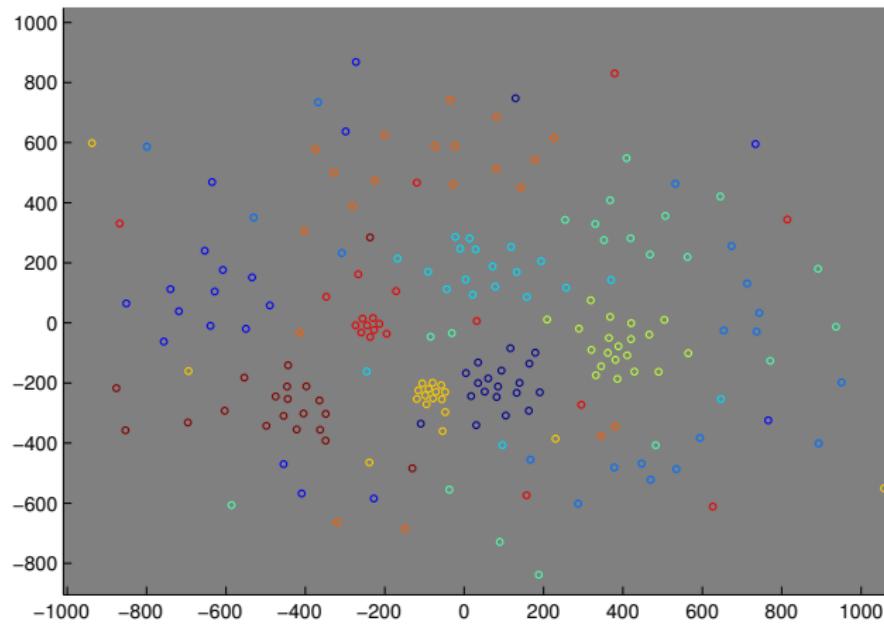
# Fajtái

- Lehet hard vagy soft clustering
  - Hard clustering: egy adatpont csak egy klaszterben szerepelhet
  - Soft clustering: minden adatpontra megvan, hogy mennyire tartozik az egyes klaszterekbe
- Átmenetek
  - Átfedő klaszterezés: egy elem több klaszterbe is tartozhat, de vagy beletartozik, vagy nem
  - Hierarchikus klaszterezés: a klasztereket hierarchiába szervezzük, a gyerek klaszterbe tartozó elemek a szülőbe is beletartoznak

# Hierarchikus klaszterezés



## Példa természetes klasztereződésre – azonos értelmű szavak jelentései (t-SNE)



# Tartalom

## 1 Bevezetés

## 2 Klaszterezés

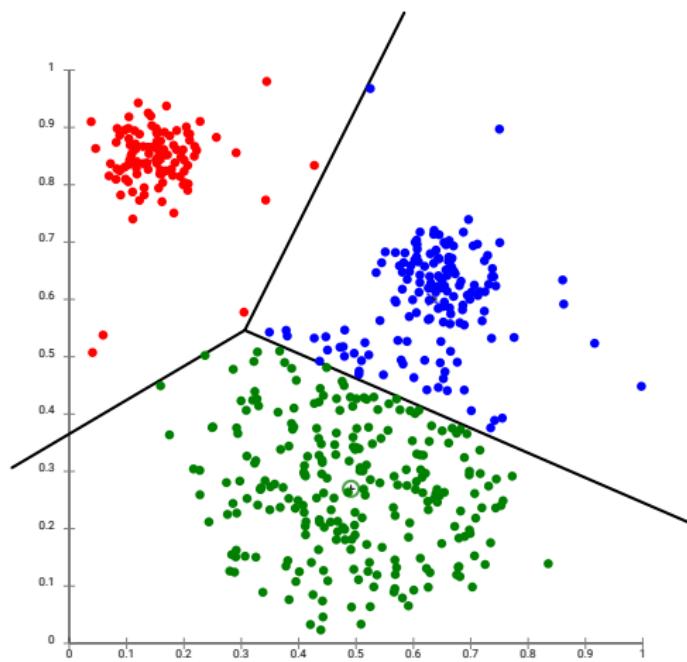
- Hard clustering – k-means
- Soft clustering – témamodellek

## 3 Dimenziócsökkentés

- Kovariancia, korreláció
- Főkomponens analízis

## 4 Autoenkóderek

# Példa



# Feladat

- Adott:  $k$ , a klaszterek száma
- minden klasztert a középpontjával reprezentálunk
  - Centroid, a klaszter pontjainak átlaga
- A feladat: keressük meg a  $k$  klaszter középpontot és az adatpontokat rendeljük ezekhez hozzá úgy, hogy a klaszteren belüli, középponttól számított távolságnégyzeteket minimalizáljuk
  - Ekvivalens a páronkénti távolságnégyzetek minimalizálásával
  - NP-nehéz, így approximáljuk
  - Csak lokális optimumot találunk
  - Többször futtathatjuk különböző véletlen inicializációkkal

## k-means feladat

$$\arg \min_{\mathbf{S}} \sum_{i=1}^k \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 = \arg \min_{\mathbf{S}} \sum_{i=1}^k \frac{1}{2|S_i|} \sum_{\mathbf{x}, \mathbf{y} \in S_i} \|\mathbf{x} - \mathbf{y}\|^2$$

# Algoritmus

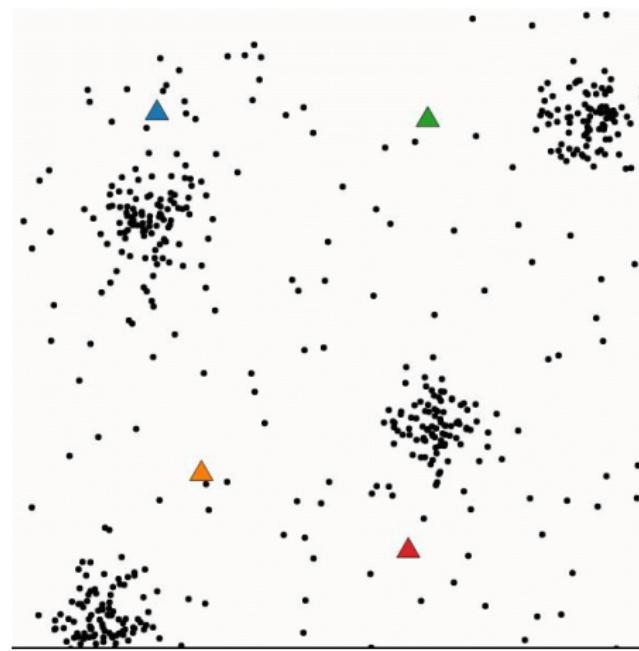
- Kezdetben adott  $k$  és az adatpontok  $\mathbb{R}^n$ -ben
- Inicializáljuk az  $m_1^{(1)}, m_2^{(1)}, \dots, m_k^{(1)}$  centroidokat
  - Véletlenszerűen kiválasztunk  $k$  adatpontot, vagy
  - minden adatpontot véletlenszerűen egy klaszterbe sorolunk és kiszámoljuk a centroidokat
- Váltogatjuk a következő két lépést, amíg nem konvergálunk
  - 1 minden adatpontot a legközelebbi centroidhoz rendelünk:

$$S_i^{(t)} = \{x_p : \|x_p - m_i^{(t)}\|^2 \leq \|x_p - m_j^{(t)}\|^2 \forall j, 1 \leq j \leq k\}$$

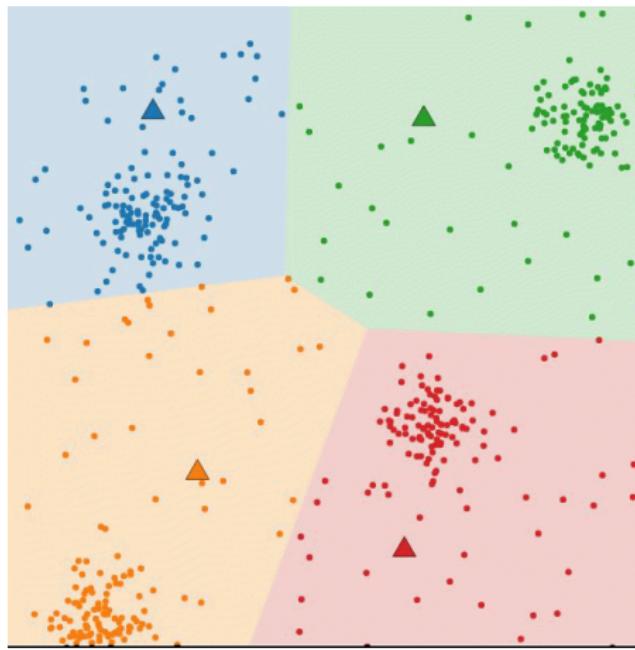
- 2 Kiszámítjuk az új centroidokat

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$$

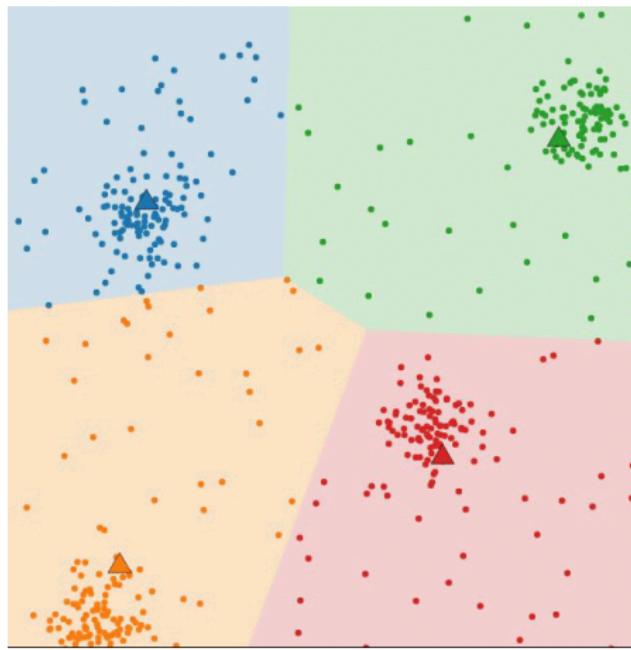
# Algoritmus



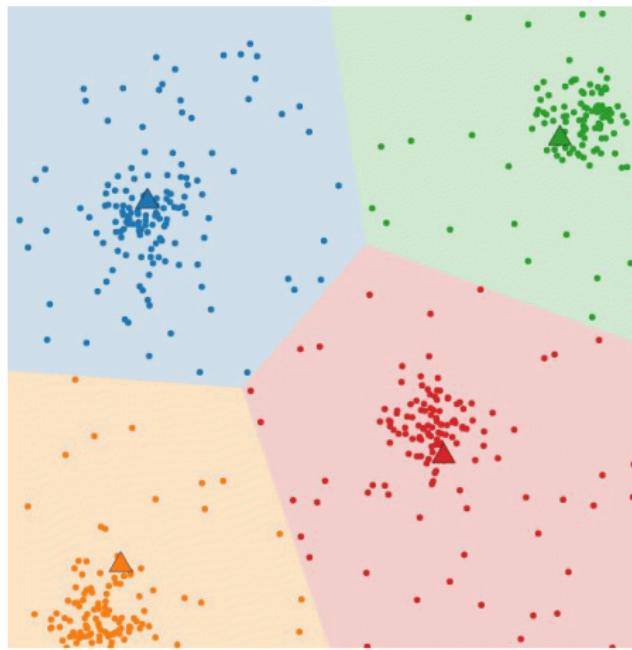
# Algoritmus



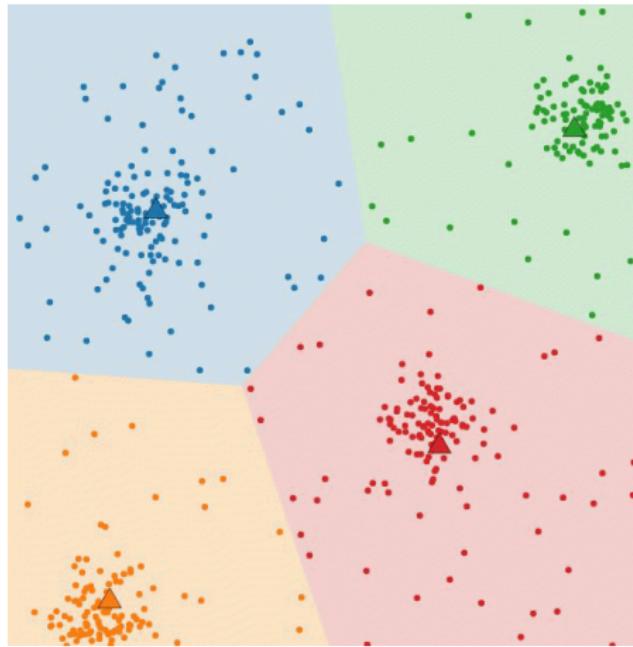
# Algoritmus



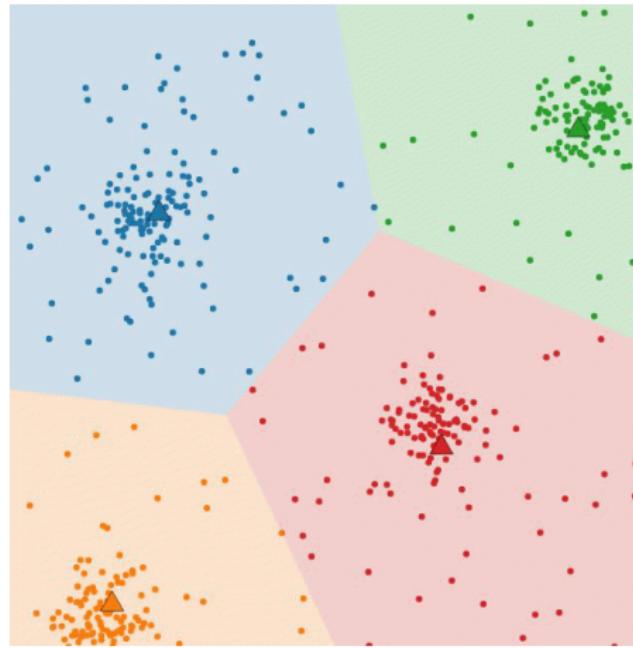
# Algoritmus



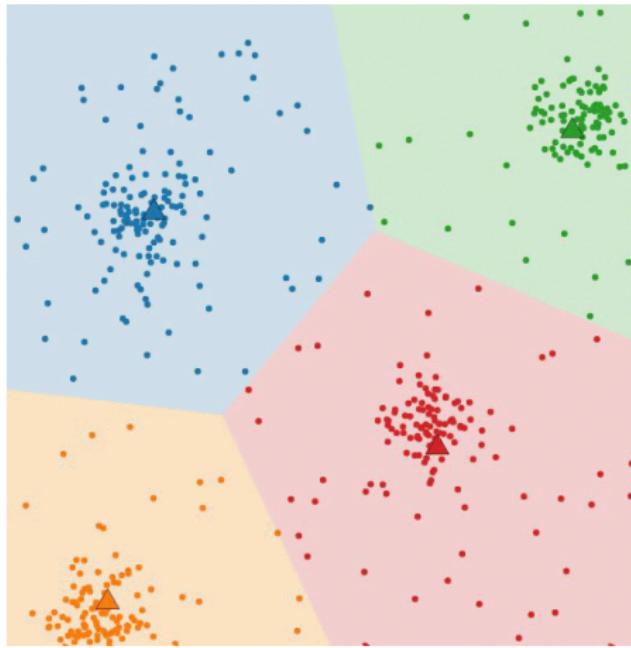
# Algoritmus



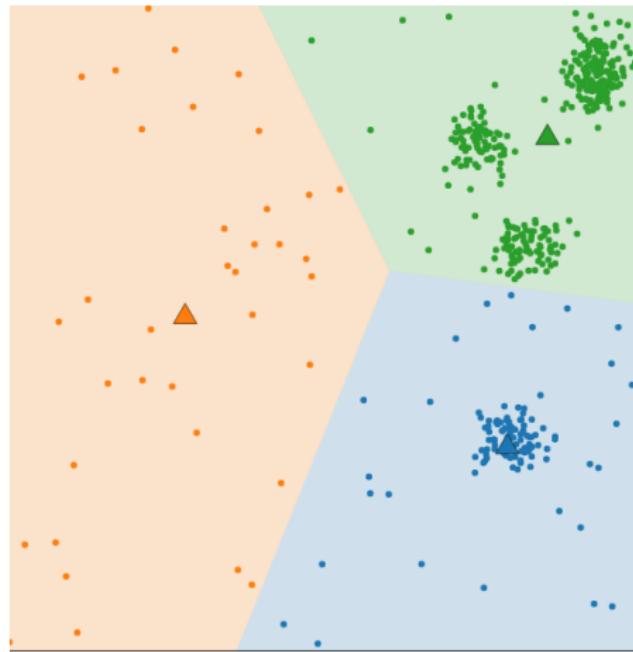
# Algorithmus



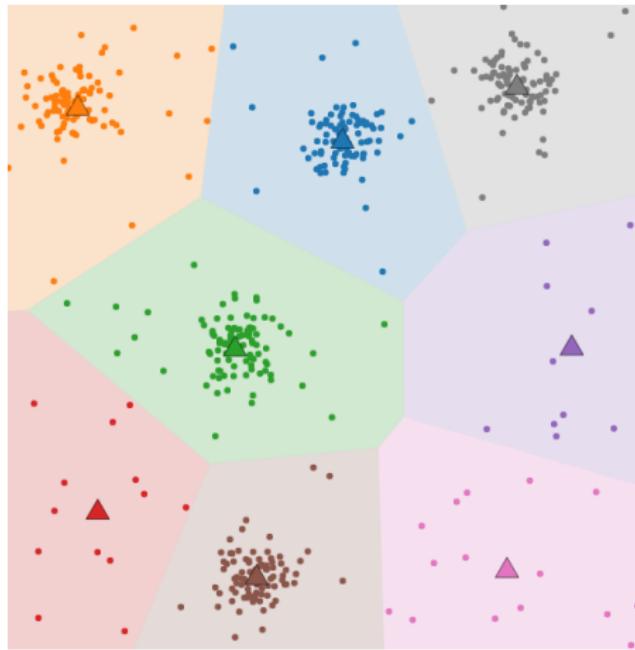
# Algoritmus



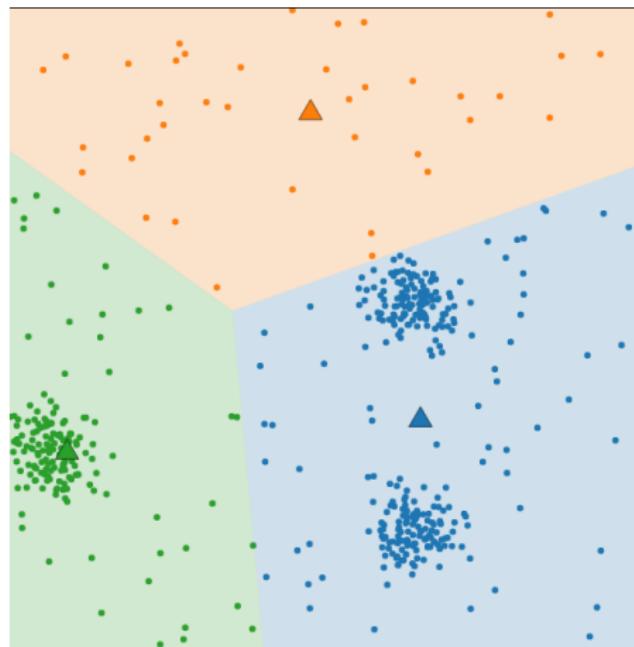
## Problémák – túl kicsi k-t adunk meg



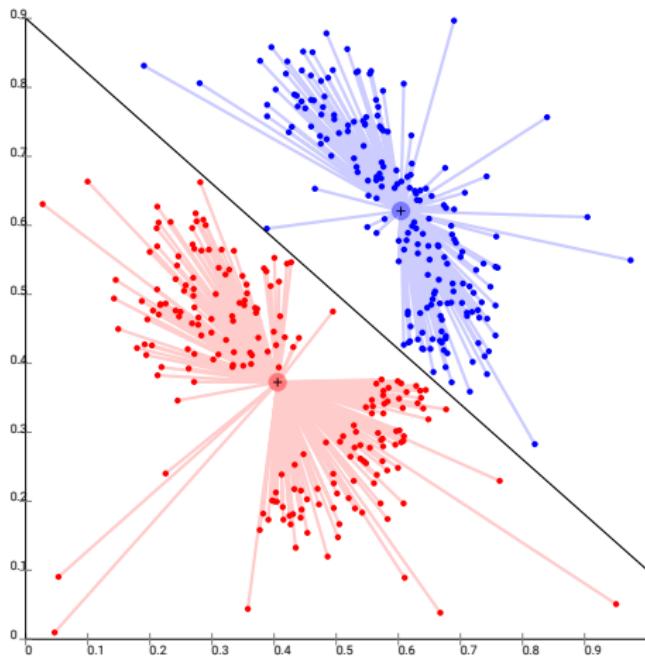
## Problémák – túl nagy k-t adunk meg



## Problémák – rossz inicializáció



# Problémák – sűrűség alapúak a klaszterek



## Python példák

- Kép kvantálás: [http://scikit-learn.org/stable/auto\\_examples/cluster/plot\\_color\\_quantization.html](http://scikit-learn.org/stable/auto_examples/cluster/plot_color_quantization.html)
- Dokumentumok klaszterezése:  
[https://scikit-learn.org/0.19/auto\\_examples/text/document\\_clustering.html](https://scikit-learn.org/0.19/auto_examples/text/document_clustering.html)

# Tartalom

## 1 Bevezetés

## 2 Klaszterezés

- Hard clustering – k-means
- Soft clustering – témamodellek

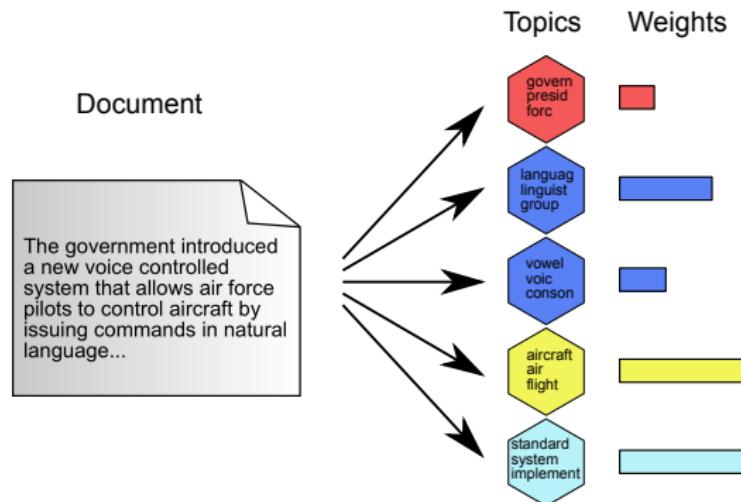
## 3 Dimenziócsökkentés

- Kovariancia, korreláció
- Főkomponens analízis

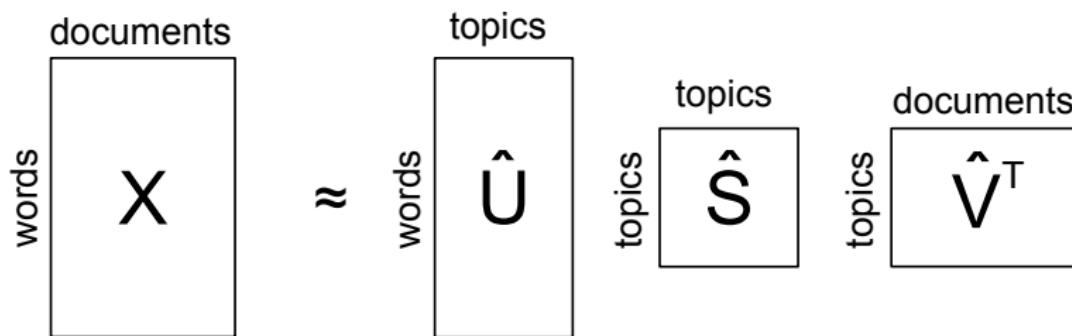
## 4 Autoenkóderek

# Témamodellek

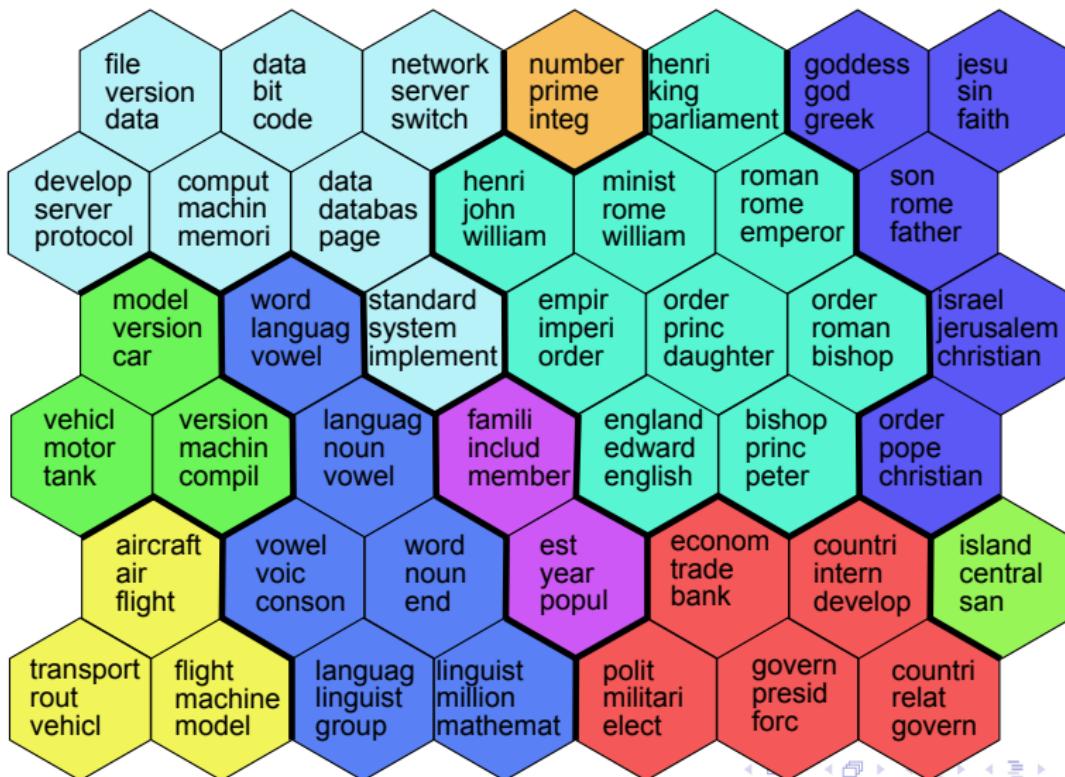
- Soft clusteringre példa: témamodellek
  - Egy dokumentum mennyire szól az egyes témákról
  - Egy témába milyen szavak tartoznak?
  - Pl. Latent Semantic Analysis (SVD)



# Latent Semantic Analysis



# Kiterjesztés csoportritka regularizációval



# Példa: kakukktojás játék

Egybetartozó szavak				Kakukktojás
cao	wei	liu	emperor	<b>king</b>
superman	clark	luthor	kryptonite	<b>batman</b>
devil	demon	hell	soul	<b>body</b>
egypt	egyptian	alexandria	pharaoh	<b>bishop</b>
singh	guru	sikh	saini	<b>delhi</b>
language	dialect	linguistic	spoken	<b>sound</b>
mass	force	motion	velocity	<b>orbit</b>
voice	speech	hearing	sound	<b>view</b>
athens	athenian	pericles	corinth	<b>ancient</b>
data	file	format	compression	<b>image</b>
function	problems	polynomial	equation	<b>physical</b>

# Tartalom

## 1 Bevezetés

## 2 Klaszterezés

- Hard clustering – k-means
- Soft clustering – témamodellek

## 3 Dimenziócsökkentés

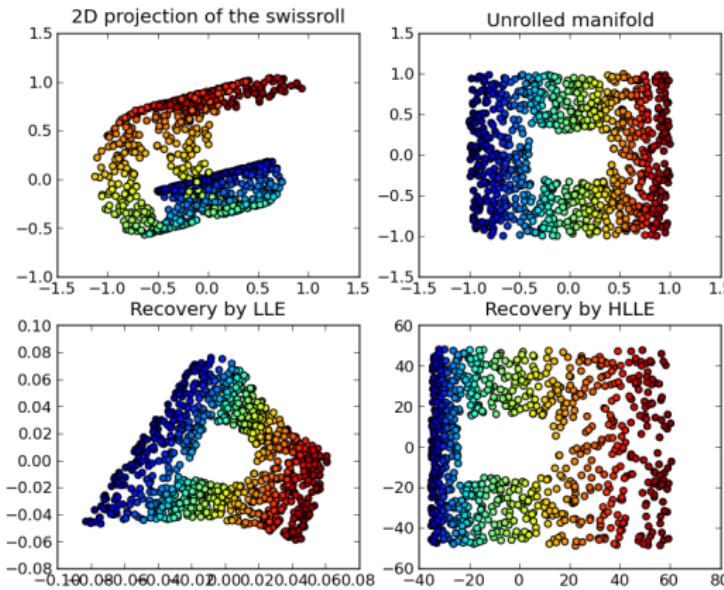
- Kovariancia, korreláció
- Főkomponens analízis

## 4 Autoenkóderek

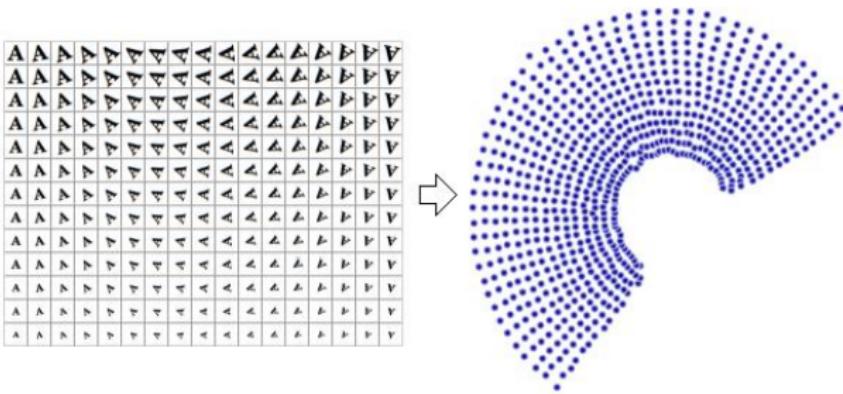
## Miért dimenziócsökkentünk?

- Az adatok valójában alacsonyabb dimenziósak, csak magasabb dimenziós térben vannak
- Láttatjuk az adatokat
- Eltüntetjük a zajt
- Csökkentjük a tanulási feladat bonyolultságát (jobb eredmények, kisebb futási idő, ...)
- A csökkentett dimenziójú adatokon új törvenyszerűségeket, sejtéseket láthatunk meg
- A probléma megoldásához kisebb dimenziós és/vagy sűrű reprezentációra van szükségünk

# Példa: Swiss roll



## Példa: A betű forgatása



# Tartalom

## 1 Bevezetés

## 2 Klaszterezés

- Hard clustering – k-means
- Soft clustering – témamodellek

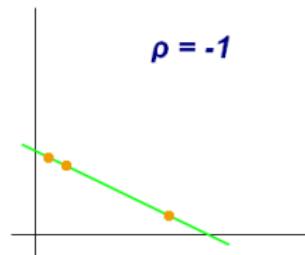
## 3 Dimenziócsökkentés

- Kovariancia, korreláció
- Főkomponens analízis

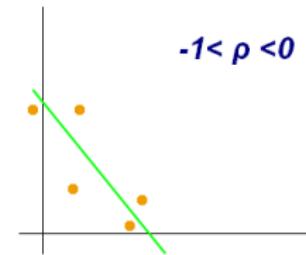
## 4 Autoenkóderek

# Kovariancia, korreláció

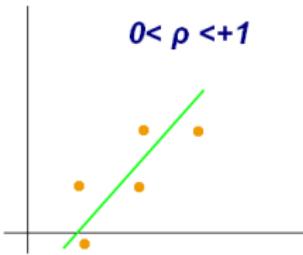
$$\rho = -1$$



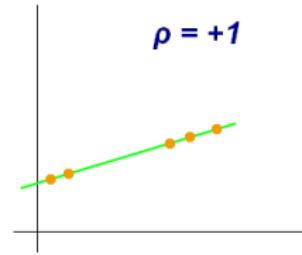
$$-1 < \rho < 0$$



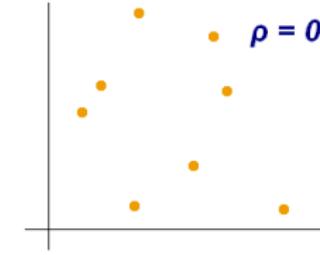
$$0 < \rho < +1$$



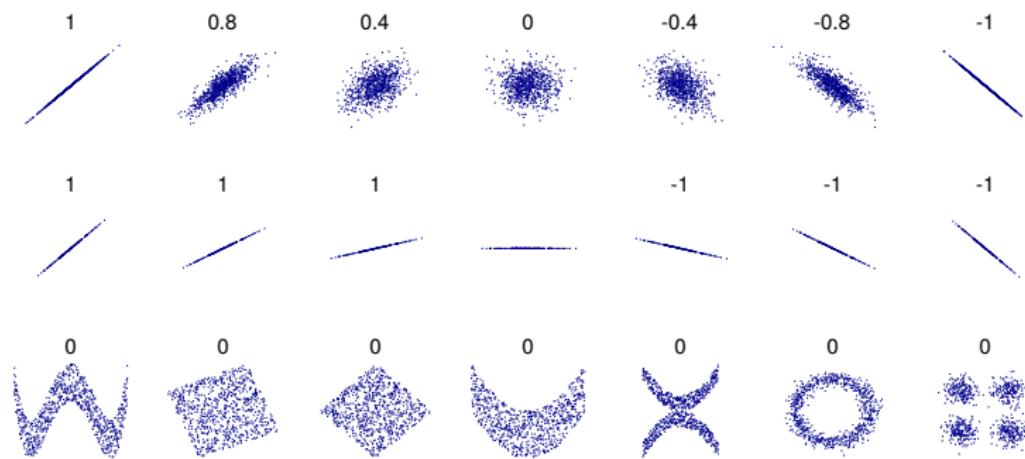
$$\rho = +1$$



$$\rho = 0$$



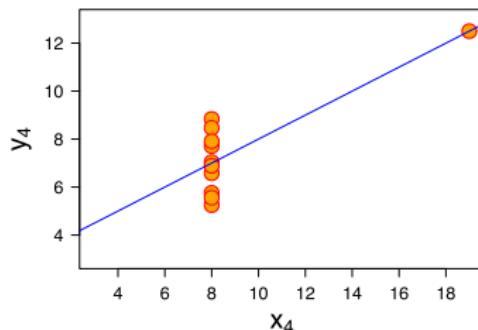
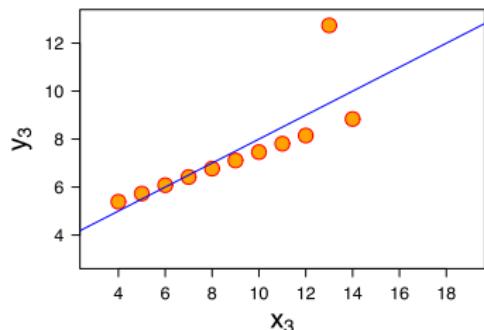
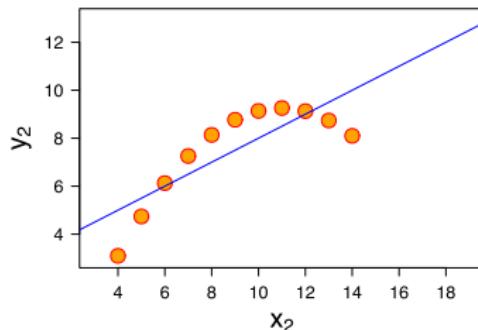
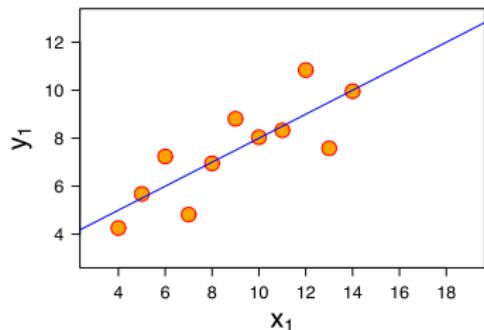
# Kovariancia, korreláció



## Kovariancia, (Pearson) korreláció

- Azt mérik, hogy  $X$ ,  $Y$  val. változók mennyire mozognak együtt
- Lineáris kapcsolatot mutatnak
- $\text{Cov}(X, Y) = E[(X - E[X])(Y - E[Y])]$
- Pl.: Pozitív: Ha  $X > E(X)$ , akkor  $Y > E(Y)$ , ha  $X < E(X)$ , akkor  $Y < E(Y)$
- $\text{Cov}(X, Y) = E[XY] - E[X]E[Y]$
- Korreláció: „Normalizált” kovariancia, -1 és 1 között
- $\rho_{X,Y} = \text{corr}(X, Y) = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}$ 
$$\boxed{r_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2}}}$$

Mind a négy adathalmaz korrelációs együtthatója 0.816



# Kovariancia mátrix

- $\mathbf{X}$  egy vektor, aminek az elemei val. változók
- A kovariancia mátrix elemei  $X_i, X_j$  közti kovarianciák
- $\Sigma_{ij} = \text{cov}(X_i, X_j) = E[(X_i - \mu_i)(X_j - \mu_j)]$
- $\mu_i = E(X_i)$

- 
- $$\begin{bmatrix} E[(X_1 - \mu_1)(X_1 - \mu_1)] & E[(X_1 - \mu_1)(X_2 - \mu_2)] & \cdots & E[(X_1 - \mu_1)(X_n - \mu_n)] \\ E[(X_2 - \mu_2)(X_1 - \mu_1)] & E[(X_2 - \mu_2)(X_2 - \mu_2)] & \cdots & E[(X_2 - \mu_2)(X_n - \mu_n)] \\ \vdots & \vdots & \ddots & \vdots \\ E[(X_n - \mu_n)(X_1 - \mu_1)] & E[(X_n - \mu_n)(X_2 - \mu_2)] & \cdots & E[(X_n - \mu_n)(X_n - \mu_n)] \end{bmatrix}$$
- A főátlóban a szórások vannak.
  - Ekvivalens:  $\Sigma = E(\mathbf{X}^\top \mathbf{X}) - \mu^\top \mu$

# Tartalom

## 1 Bevezetés

## 2 Klaszterezés

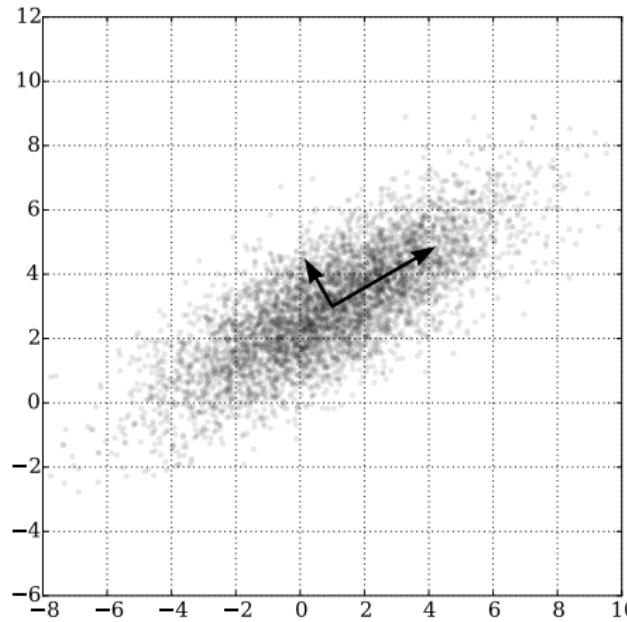
- Hard clustering – k-means
- Soft clustering – témamodellek

## 3 Dimenziócsökkentés

- Kovariancia, korreláció
- Főkomponens analízis

## 4 Autoenkóderek

## Példa - 2d normáleloszlás



# Főkomponens analízis

- Principal component analysis (PCA)
- Demo:  
<http://setosa.io/ev/principal-component-analysis/>
- Az adathalmazt egy új koordinátarendszerben ábrázoljuk, a tengelyek merőlegesek
- Az adathalmaz vetítései közül a legnagyobb szórású az első tengelyen (főkomponensen) van
- A második legnagyobb szórású a második főkomponensen, ...
- Új változók/adatok: a főkomponensekre vetítjük le az eredeti változókat. Ezek már korrelálatlanok
- Dimenziócsökkentés: eldobjuk azokat a tengelyeket (és koordinátákat), amiken kicsi a szórás

## Főkomponens analízis

- $\mathbf{X} \in \mathbb{R}^{n \times p}$ : adathalmaz, egy sor egy adatpont
- $\mathbf{t}_{(i)} = (t_1, \dots, t_l)_{(i)}$ : az adatpontok az új koordinátarendszerbe transzformálva  $\mathbf{w}_{(k)} = (w_1, \dots, w_p)_{(k)}$ -val

$$t_{k(i)} = \mathbf{x}_{(i)} \cdot \mathbf{w}_{(k)} \quad \text{for } i = 1, \dots, n \quad k = 1, \dots, l$$

- Szórás maximalizálása

$$\mathbf{w}_{(1)} = \arg \max_{\|\mathbf{w}\|=1} \left\{ \sum_i (t_1)_{(i)}^2 \right\} = \arg \max_{\|\mathbf{w}\|=1} \left\{ \sum_i (\mathbf{x}_{(i)} \cdot \mathbf{w})^2 \right\}$$

# Főkomponens analízis

- Ugyanez mátrixosan:

$$\mathbf{w}_{(1)} = \arg \max_{\|\mathbf{w}\|=1} \{\|\mathbf{X}\mathbf{w}\|^2\} = \arg \max_{\|\mathbf{w}\|=1} \left\{ \mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} \right\}$$

- Mivel  $\mathbf{w}$  egységevektor:

$$\mathbf{w}_{(1)} = \arg \max \left\{ \frac{\mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w}}{\mathbf{w}^T \mathbf{w}} \right\}$$

- Ez a Rayleigh-hányados, a legnagyobb lehetséges érték az  $\mathbf{X}^T \mathbf{X}$  legnagyobb sajátértéke lesz, ahol  $\mathbf{w}$  a hozzá tartozó sajátvektor
- A többi komponensre is így van → a főkomponensek az  $\mathbf{X}^T \mathbf{X}$  sajátvektorai

## Főkomponens analízis – algoritmus

- Az  $\mathbf{X}$  mátrixban vannak az adataink
- Nulla átlagúra hozzuk az adatokat (kivonjuk az átlagot)
- Kiszámoljuk a  $\mathbf{Q} = \mathbf{X}^T \mathbf{X}$  kovariancia mátrixot
- Meghatározzuk ennek a mátrixnak a sajátértékeit, és a sajátvektorait
- A sajátvektorok a főkomponensek, a belőlük álló bázis az új koordinátarendszer
- A legnagyobb sajátértékhez tartozó főkomponens a legnagyobb szórású, és így tovább
- Dimenziócsökkentés: csak a  $k$  legnagyobb sajátértékű főkomponensem tartjuk meg

# PCA és SVD

## SVD

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{W}^T$$

## PCA SVD-vel

$$\begin{aligned}\mathbf{X}^T\mathbf{X} &= \mathbf{W}\Sigma^T\mathbf{U}^T\mathbf{U}\Sigma\mathbf{W}^T \\ &= \mathbf{W}\Sigma^T\Sigma\mathbf{W}^T \\ &= \mathbf{W}\hat{\Sigma}^2\mathbf{W}^T\end{aligned}$$

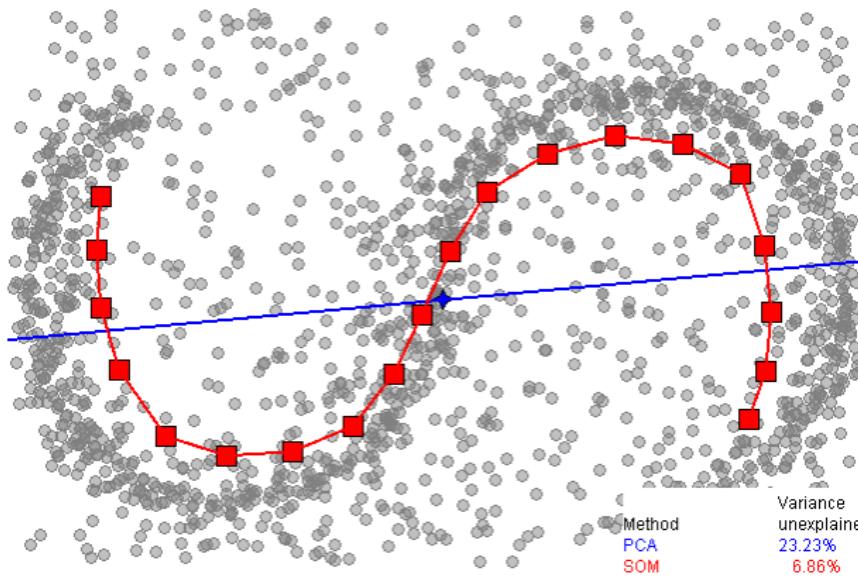
- $\mathbf{W}$ -ben már  $\mathbf{X}^T\mathbf{X}$  sajátvektorai vannak. A szinguláris értékek a sajátertékek négyzetgyökei.

Nem felügyelt tanulás

└ Dimenziócsökkentés

└ Főkomponens analízis

## A PCA is lineáris



## Python példák

- A feature scaling fontossága:

[http://scikit-learn.org/stable/auto\\_examples/preprocessing/plot\\_scaling\\_importance.html](http://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html)

# Tartalom

## 1 Bevezetés

## 2 Klaszterezés

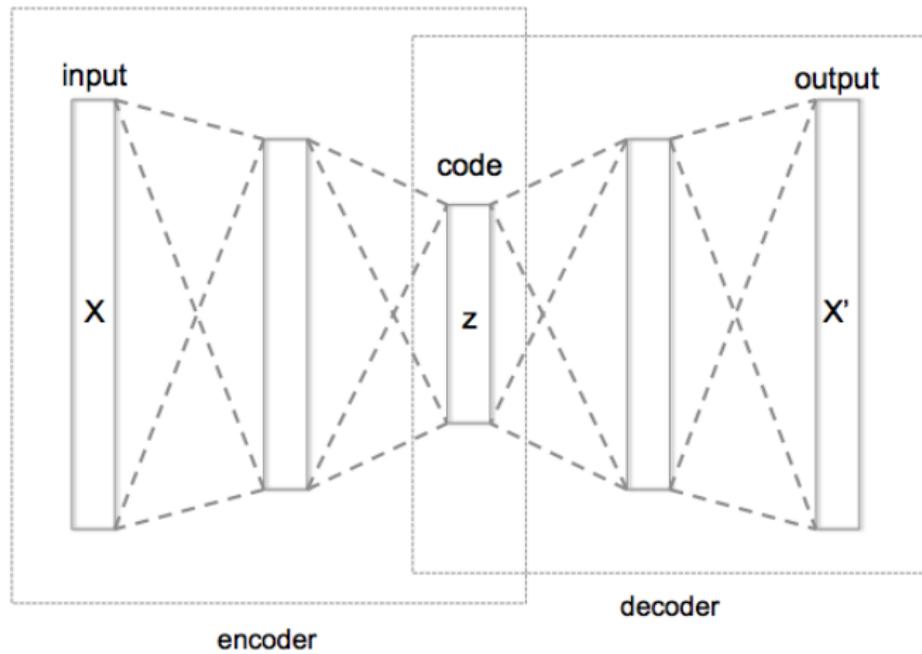
- Hard clustering – k-means
- Soft clustering – témamodellek

## 3 Dimenziócsökkentés

- Kovariancia, korreláció
- Főkomponens analízis

## 4 Autoenkóderek

# Autoenkóderek



# Autóenkóderek

## Egyszerű autóenkóder

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2 = \|\mathbf{x} - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')\|^2$$

- Ez az egyszerű autoenkóder a PCA alterébe projektál
- Flexibilis, sokféle variáció létezik
  - Denoising autoencoder: zajos inputból kell zajtalan outputot előállítani
  - Sparse autoencoder: csak néhány egység lehet aktív a rejtett reprezentációban
  - VAE: Egy valószínűségi modellt feltételez, a poszterior eloszlást approximálja
- Sokszor fontosak egy felügyelt mély háló előtanításában
- <https://transcranial.github.io/keras-js/#/mnist-vae>

# Köszönöm a figyelmet!

Köszönöm a figyelmet!