

AZ MI FOGALMA

mesterséges intelligencia – MI (artificial intelligence - AI)

sokan sokfélét értenek alatta

Nem egy rész-területe az informatikának, hanem egy szemléletmód, amely az informatika fejlődését szolgálja: olyan problémákra keres számítógépes megoldásokat, amelyek megoldásában az ember jobbnak tűnik.

Erős MI

Cél: az emberi gondolkodás számítógéppel történő reprodukálása.

MI szkeptikusok

A számítógép soha nem lesz okosabb az embernél.

Gyenge MI

Cél: Azon elméletek és módszerek kutatása, fejlesztése, rendszerezése, amelyekkel az emberi intelligencia számára is érdekes és nehéz problémákra adhatunk számítógépes megoldásokat.

módszerek és célok
specializálódása

MI története

Romantikus kor

mindenféle problémát
általános eszközökkel

1956

1960

1970

1980

1990

2000

2010

Gregorics Tibor

Mesterséges intelligencia

Projektek:

kétszemélyes játékok (sakk)
beszélgető program (ELIZA, 1966)

Módszerek:

GPS, rezolúció (1966)
Lisp (1958)
mesterséges neuronhálók
evolúciós algoritmusok

Minta-válasz párok:

<a> ön engem <c>.

Úgy érzem, hogy ön mostanában engem un.

1. Miért gondolja, hogy ön *<a> én <c>*?
2. Tegyük fel, hogy én ** önt *<c>*. Mit változtat ez a dolgokon?

Ismétlés felismerése:

„Miért ismételgeti ugyanazt újra és újra?”

Folytatás:

Igen, értem. Kérem folytassa. Ez nagyon érdekes.
Még miről szeretne beszélgetni?

módszerek és célok specializálódása

MI története

Klasszikus kor

speciális problémákat
speciális módszerekkel

Romantikus kor

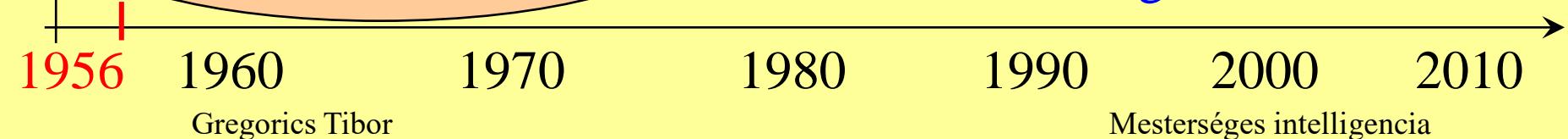
mindenféle problémát
általános eszközökkel

Projektek:

SHRDLU (1972),
BACON, AM
DENDRAL (1969-78),
MYCIN(1976)

Módszerek:

heurisztikus keresés,
tudás reprezentáció
Prolog



módszerek és célok specializálódása

MI története

Ipari kor

szakértő rendszerek
tudásalapú rendszerek

Klasszikus kor

speciális problémákat
speciális módszerekkel

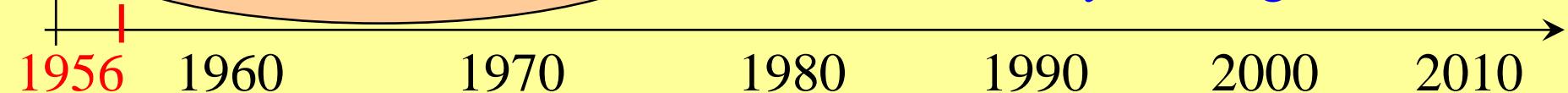
Romantikus kor

mindenféle problémát
általános eszközökkel

Projektek: XCON(1982),
PROSPECTOR (1979)

Módszerek:

shell-ek (IDE)
tudás-alapú technológia
nem-klasszikus következtetés
bizonytalanság kezelés



módszerek és célok specializálódása

MI története

MI tél, majd reneszánsz

Ipari kor

szakértő rendszerek
tudásalapú rendszerek

Klasszikus kor

speciális problémákat
speciális módszerekkel

Romantikus kor

mindenféle problémát
általános eszközökkel

gépi tanulás,
hibrid technológiák

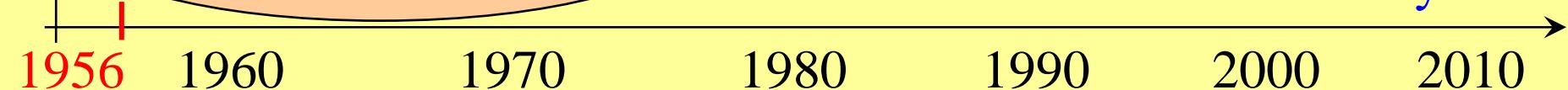
Projektek:

Deep Blue (1997)
IBM Watson (2011)

robotika

Aktuális területek:

mély hálók
felügyelet nélküli
tanulás
adattudomány



Miről ismerhető fel egy szoftverben az MI?

❑ Megoldandó feladat: nehéz

- A feladat **problémátere** 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
- általános mesterséges intelligencia

❑ Felhasznált technológiák: sajátosak

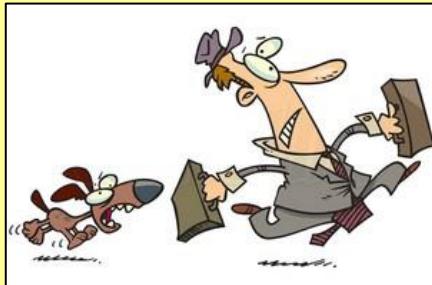
- speciális reprezentáció a feladat **modellezéséhez**
- heurisztikával megerősített hatékony **algoritmusok**
- **gépi tanulás** módszerei

Intelligens szoftver jellemzői

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

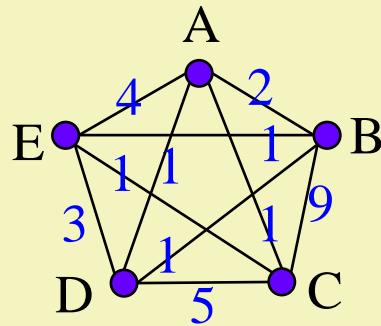
modellezés
és keresés

gépi tanulás



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

ABDECA

ADBECA

...

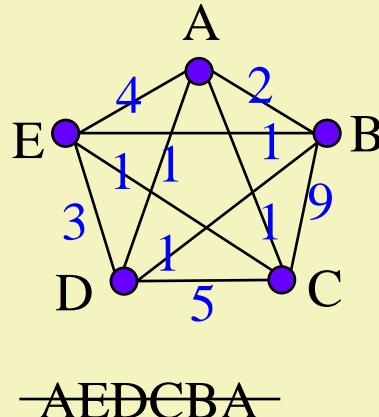
ABCEDA

problématér



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
elemek
elhagyása

szomszédos
elempár cseréje

start: ABCDEA $2+9+5+3+4=23$

ACBDEA
 $1+9+1+3+4=18$

ABDCEA
 $2+1+5+1+4=13$

ABCEDA
 $2+9+1+3+1=16$

legjobb elem
választása

ABDECA $2+1+3+1+1=8$

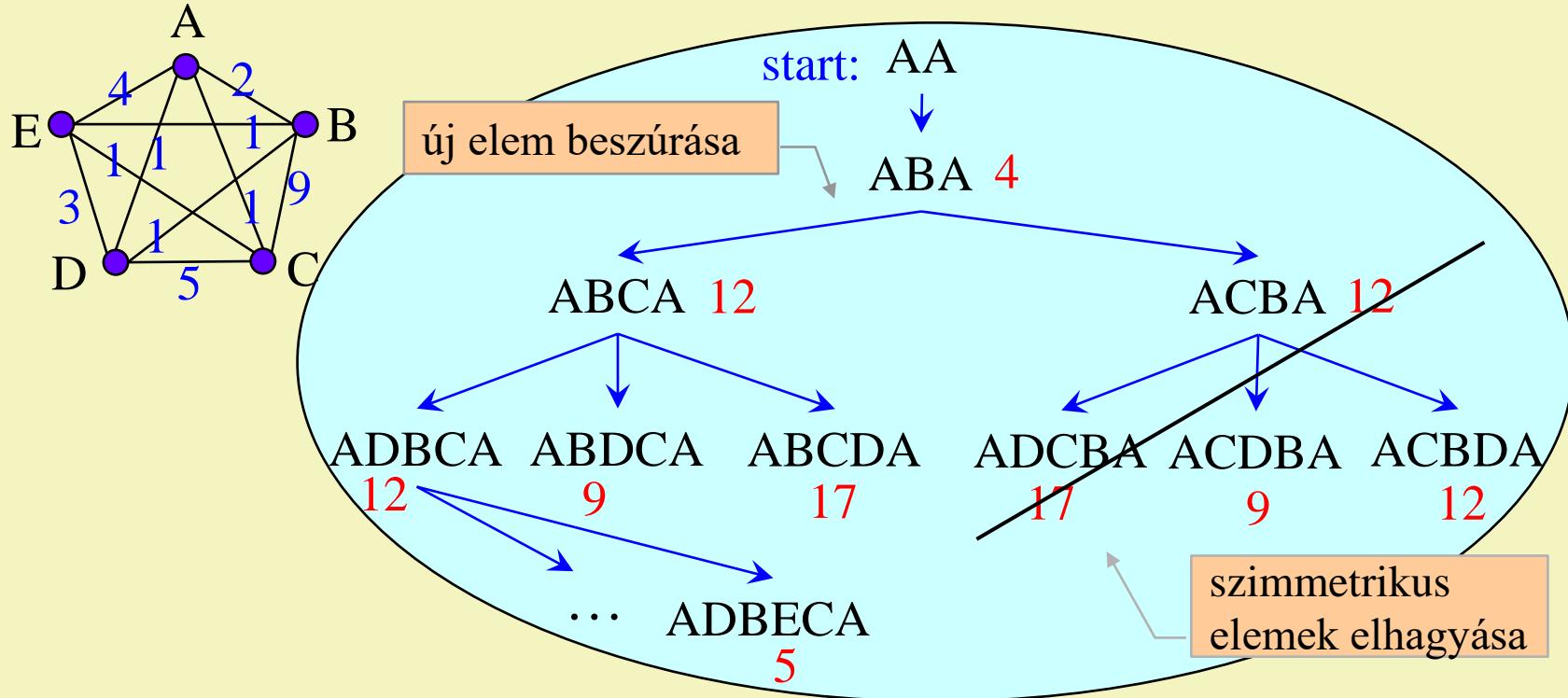
ADBECA
 $1+1+1+1+1=5$

...

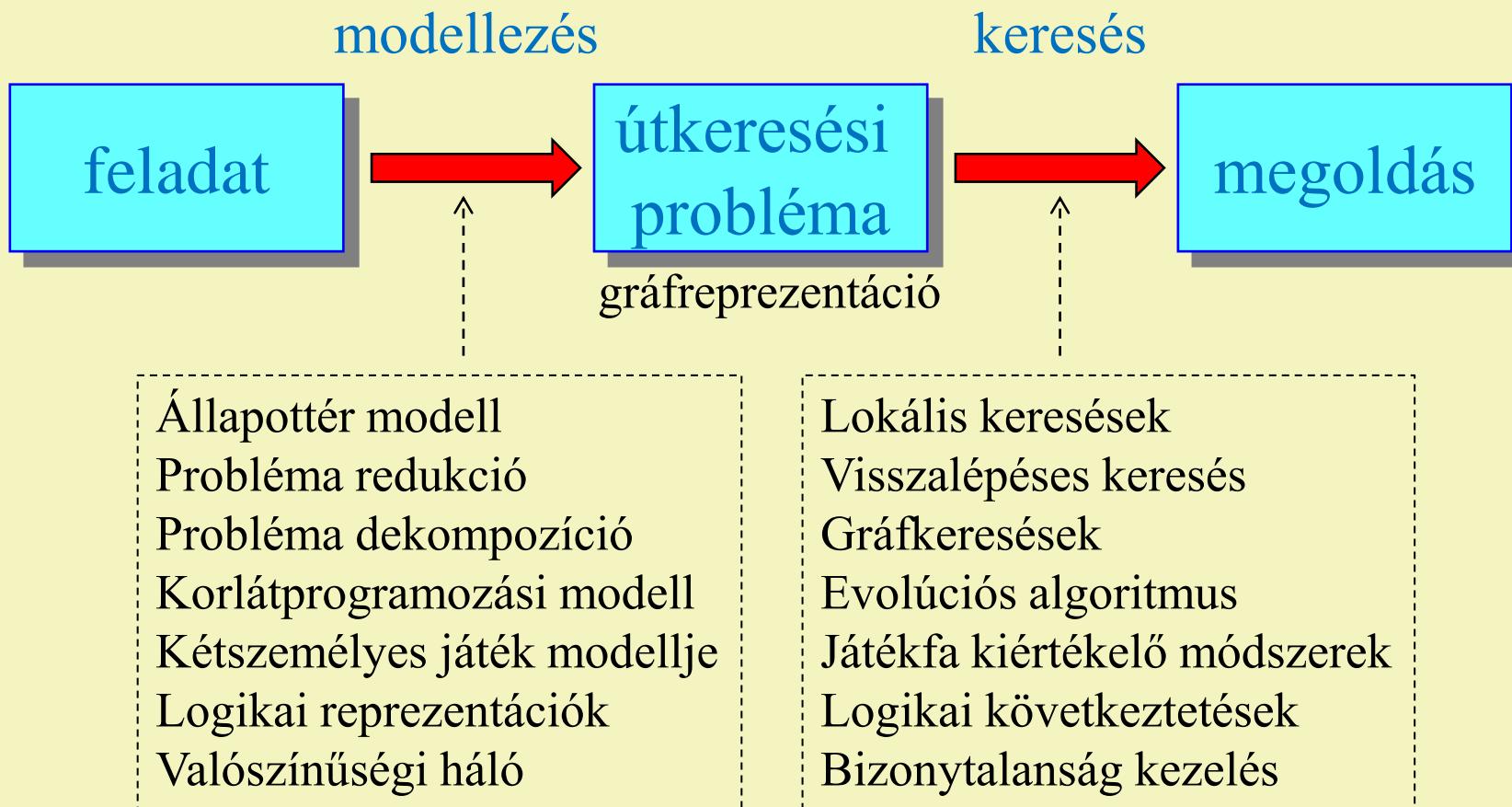


Utazó ügynök problémája

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?



MODELLEZÉS & KERESÉS



Mire kell a modellezésnek fókuszálni

- **Problémater elemei:** probléma lehetséges válaszai
- **Cél:** egy helyes válasz (megoldás) megtalálása
- **Keresést segítő ötletek** (heurisztikák):
 - Problémater 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**.

Útkeresési probléma

- Útkeresési probléma az, amelynek megoldása megfeleltethető egy élsúlyozott irányított gráfbeli
 - csúcsnak (célcsúcs), vagy még inkább
 - útnak (startcsúcsból célcsúcsba, esetleg a legolcsóbb)
- Ez a gráf (δ -gráf) lehet végtelen nagy, de
 - csúcsainak kifoka véges, és
 - élei súlyának (költségének) van egy konstans globális pozitív alsó korlátja (δ).



Számos olyan modellező módszert ismerünk, amely a kitűzött feladatot útkeresési problémává fogalmazza át.

Gráf fogalmak 1.

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

$N, A \subseteq N \times N$ (végtelen számosság)
 $(n, m) \in A \quad (n, m \in N)$
 $\Gamma(n) = \{m \in N \mid (n, m) \in A\}$
 $\pi(n) \in \Pi(n) = \{m \in N \mid (m, n) \in A\}$
 $R = (N, A)$
 $|\Gamma(n)| < \infty \quad (\forall n \in N)$
 $c: A \rightarrow \mathbb{R}$
 $c(n, m) \geq \delta > 0 \quad (\forall (n, m) \in A)$
 δ -tulajdonságú, véges sok kivezető
élű, élsúlyozott irányított gráf

Gráffogalmak 2.

- **irányított út**

δ -gráfokban ez végtelen sok út esetén is értelmes.

Értéke ∞ , 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: $|\alpha|$

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

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

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

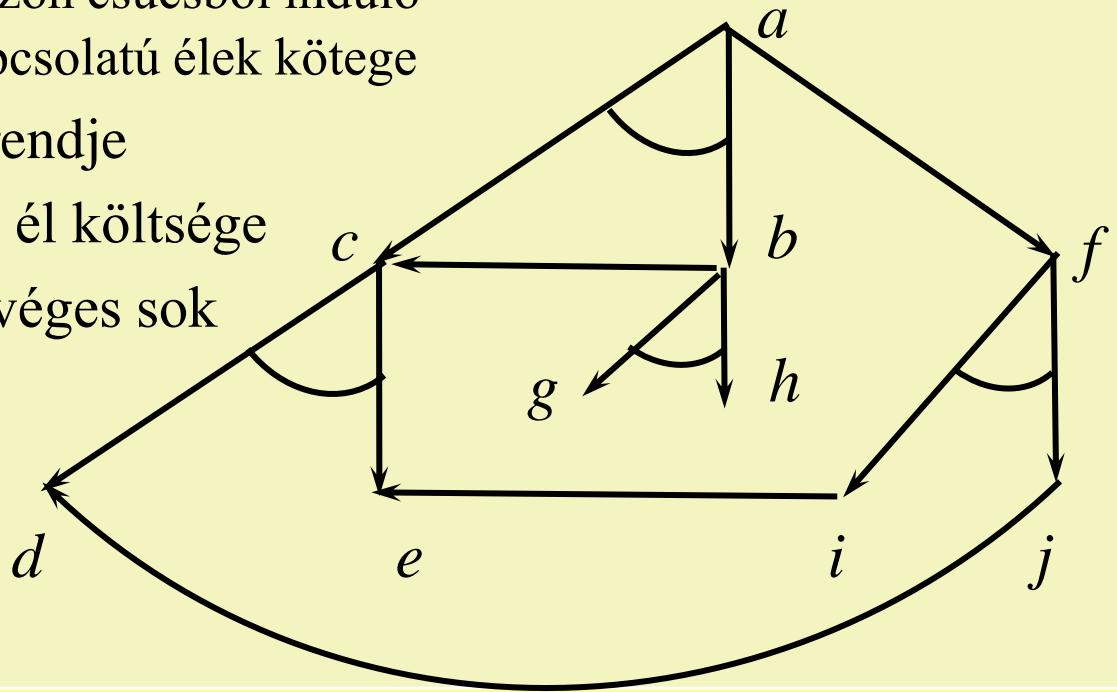
$$c^*(n, M) := \min_{\alpha \in \{n \rightarrow M\}} c^\alpha(n, m)$$

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

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

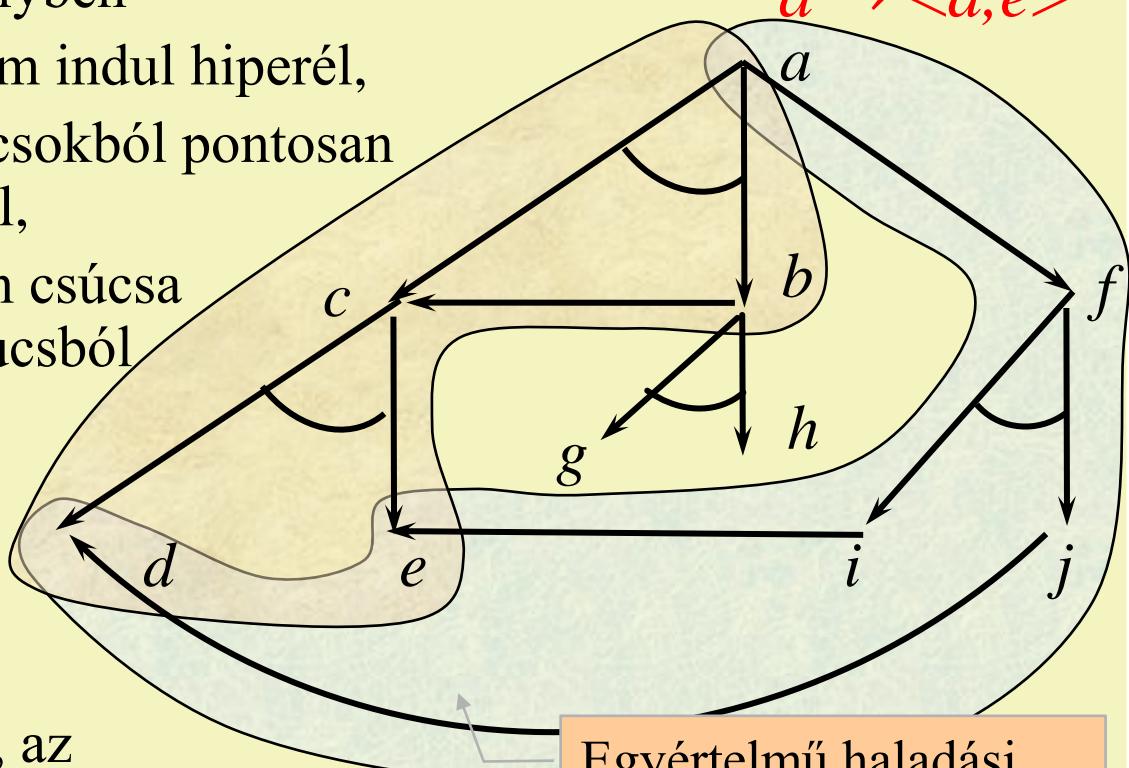
ÉS/VAGY gráfok

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



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

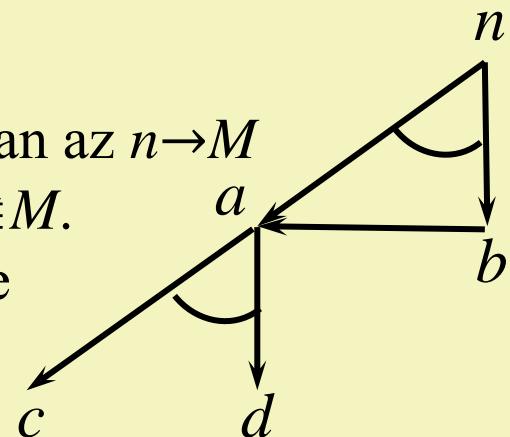
- Egy ÉS/VAGY gráf $n^{\alpha} \rightarrow M$ hiperútja ($n \in N$, $M \in N^+$) egy olyan véges részgráf, amelyben
 - M csúcsaiból nem indul hiperél,
 - M -en kívüli csúcsokból pontosan egy hiperél indul,
 - a hiperút minden csúcsa elérhető az n csúcsból egy közönséges irányított úton.
- A megoldás-gráf egy $s \rightarrow M$ hiperút, ahol s a startcsúcs, az M pedig célcímsúcsok sorozata.



A hiperút bejárása

- Az $n \rightarrow M$ hiperút bejárását a hiperéleinek adott sorrendű felsorolásával kapjuk, amelyet a **hiperút csúcsaiból képzett csúcs-sorozatok** felsorolásával is megadhatunk :

- első sorozat: $\langle n \rangle$
- C sorozatot a $C^{k \leftarrow K}$ sorozat követi, ha van az $n \rightarrow M$ hiperútból (k, K) hiperél, és $k \in C$, de $k \notin M$.
 $C^{k \leftarrow K}$ úgy kapjuk, hogy C -ben a k helyére mindenhol K -t írunk.



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

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

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

Gráfprezentáció fogalma

- minden útkeresési probléma rendelkezik egy (a probléma modellezéséből származó) gráfprezentációval, ami egy (R, s, T) hármas, amelyben
 - $R = (N, A, c)$ a **reprezentációs gráf** (δ - vagy ÉS/VAGY gráf)
 - az $s \in N$ **startcsúcs**,
 - a $T \subseteq N$ halmazbeli **célcsúcsok**.
- és a probléma megoldása:
 - t cél vagy $\langle t_1, \dots, t_m \rangle$ célcsúcs-sorozat megtalálása ($t, t_1, \dots, t_m \in T$), vagy
 - $s \rightarrow t$ vagy $s \rightarrow \langle t_1, \dots, t_m \rangle$ esetleg egy optimális $s \rightarrow^* T$ út megtalálása

Útkeresés δ-gráfban

- Egy útkeresési probléma megoldásához a reprezentációs gráfjának nagy mérete miatt speciális (nem-determinisztikus, heurisztikus) útkereső algoritmusra van szükség, amely
 - a startcsúcsból **indul** (kezdeti aktuális csúcs);
 - minden lépésben **nem-determinisztikus** módon új aktuális csúcso(ka)t **választ** a korábbi aktuális csúcs(ok) segítségével (gyakran azok gyerekei közül);
 - **tárolja** a már feltárt reprezentációs gráf egy részét;
 - **megáll**, ha célcímsúcsot talál vagy nyilvánvalóvá válik, hogy erre semmi esélye.

Útkeresés ÉS/VAGY gráfban

- ÉS/VAGY gráfbeli megoldás-gráf keresése visszavezethető egy δ -gráfban történő útkeresésre.
- A startcsúcsból induló hiperutakat (köztük a megoldás-gráfokat is) a bejárásukkal (közönséges irányított utakkal) ábrázolhatjuk, amelyek egy δ -gráfot(!) határoznak meg. A δ -gráf
 - csúcsai az eredeti ÉS/VAGY gráf csúcsainak sorozatai
 - startcsúcsa az ÉS/VAGY gráf startcsúcsából álló sorozat
 - célcsúcsai az ÉS/VAGY gráf célcsúcsaiból álló sorozatok
- Az így nyert δ -gráf megoldási újai az eredeti ÉS/VAGY gráfbeli megoldás-gráfokat reprezentálják. Ezért egy ÉS/VAGY gráfban a megoldás-gráf megkeresése a neki megfeleltetett δ -gráfban történő megoldási út megkeresésével helyettesíthető.



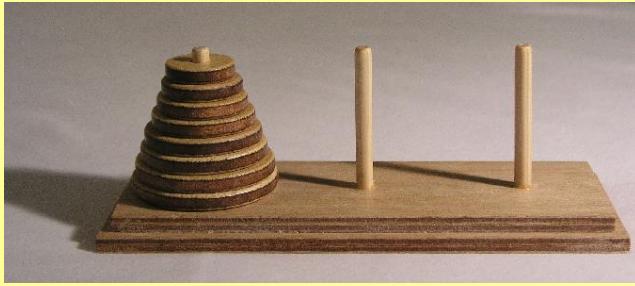
Modellezés

1. Állapottér modell

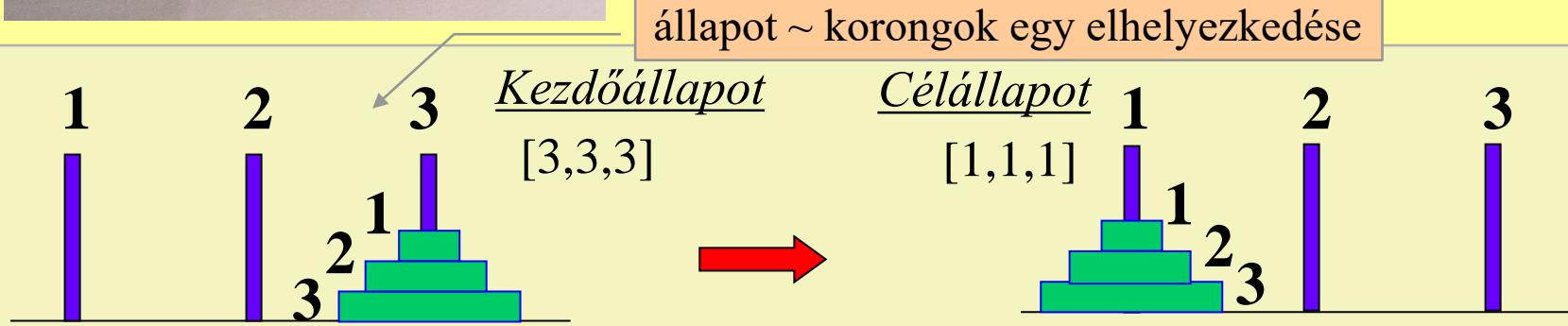
- **Állapottér**: a probléma leírásához szükséges adatok által felvett érték-együttesek (azaz **állapotok**) halmaza
 - az állapot többnyire egy **összetett szerkezetű** érték
 - gyakran egy bővebb alaphalmazzal és egy azon értelmezett **invariáns állítással** definiáljuk
- **Műveletek**: állapotból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

Állapottér modell gráfprezentációja

- Állapottér modell
 - állapot ~ Állapot-gráf
 - művelet hatása egy állapotra ~ csúcs
 • egy állapotra véges sok művelet alkalmazható (véges kifokú gráf)
 - művelet költsége ~ irányított él
 • van a műveltek költségének alsó pozitív korlátja (δ)
 - kezdő állapot ~ startcsúcs
 - célállapot ~ célcsúcs
- Gráf-reprezentáció: állapot-gráf, startcsúcs, célcsúcsok
 - egy műveletsorozat hatása ~ irányított út
 - megoldás ~ irányított út a startcsúcsból egy 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

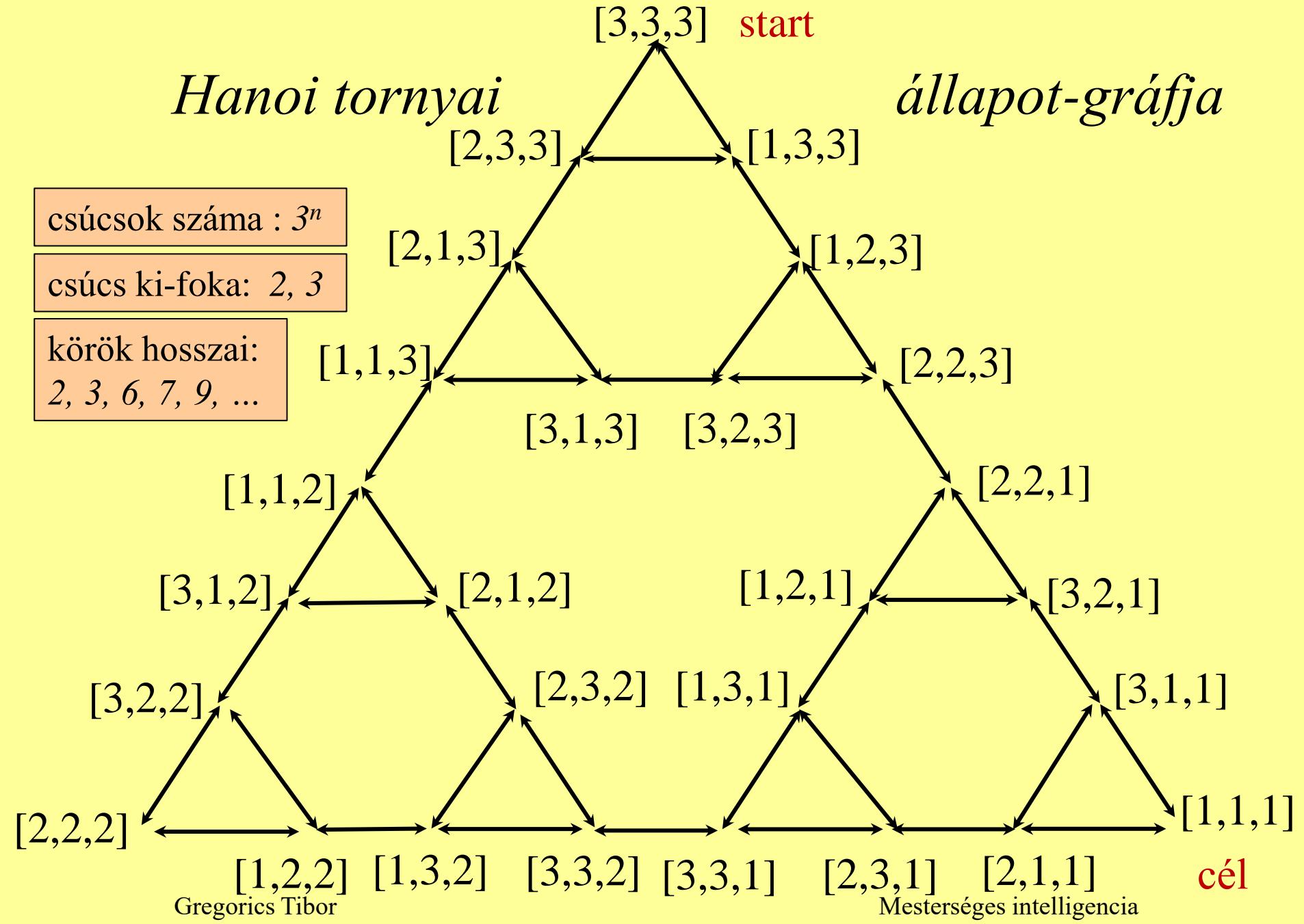
[3,3,3] start

állapot-gráfja

csúcsok száma : 3^n

csúcs ki-foka: 2, 3

körök hosszai:
2, 3, 6, 7, 9, ...

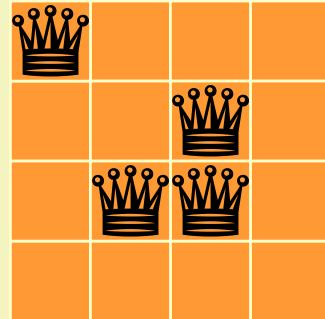




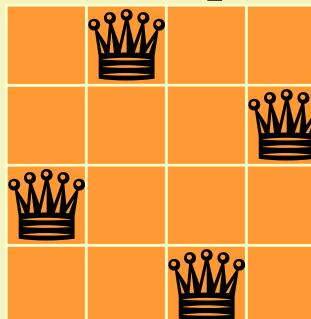
n-királynő probléma 1.

állapot ~ királynők egy elrendezése

általános állapot



Célállapot



Nem ismert,
egy feltétel
ellenőrzésével
dönthető el

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

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

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

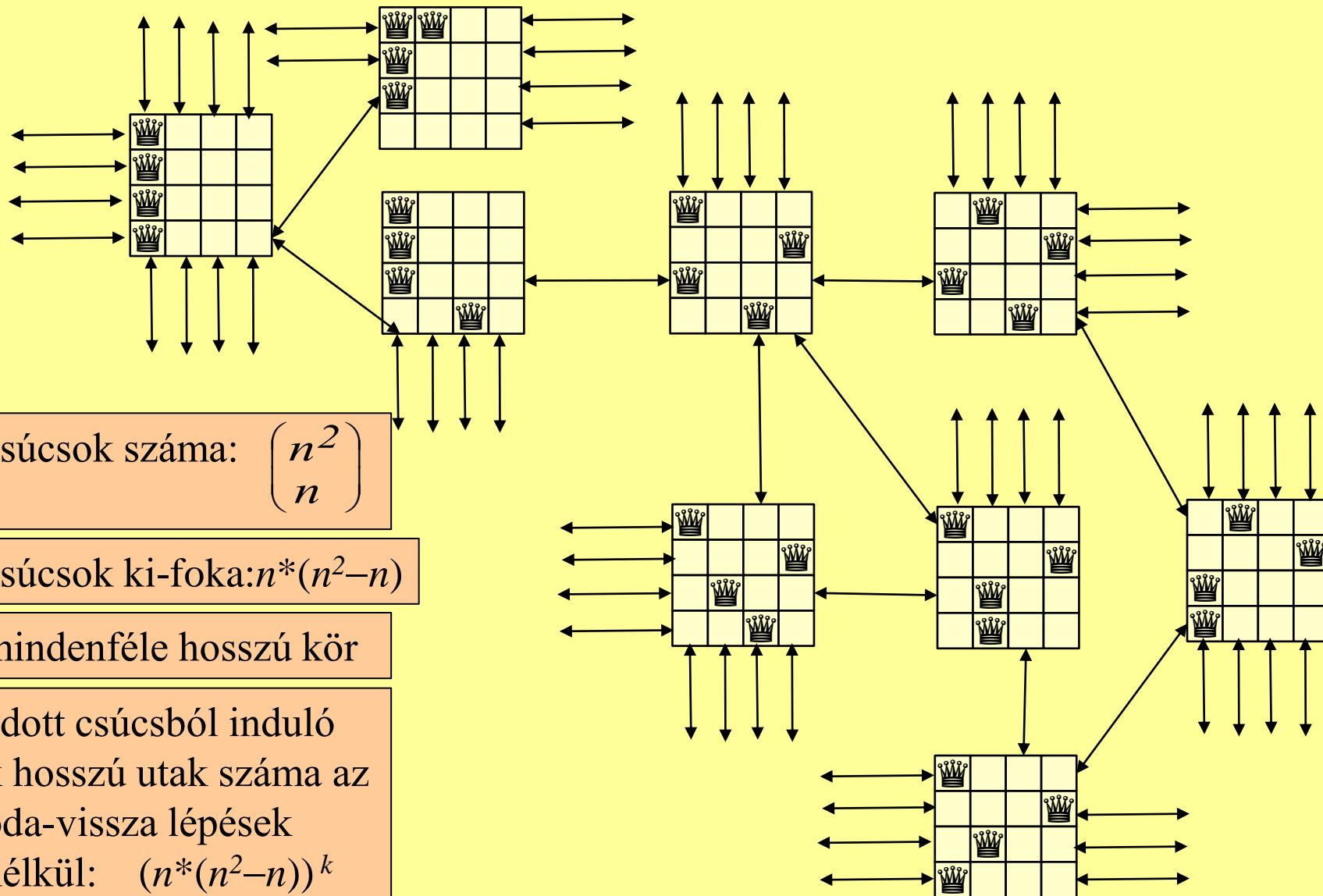
Művelet: $\text{Athelyez}(x,y,u,v):AT \rightarrow AT$ (this:AT)

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

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

csere

Állapot-gráf részlet



Állapottér vs. problématér

- Az állapottér és a problématér **kapcsolata szoros**, de ezek **nem azonosak**, hiszen a problématér elemei (a lehetséges válaszok) többnyire a kezdőállapotból kiinduló művelet-sorozatok (utak).
 - A Hanoi tornyai problémára adott lehetséges válasz nem a korongok egy állapota (elrendezése), hanem a kezdő állapotra alkalmazott művelet-sorozat. Ezek között keressük azt (a megoldást), amelyik a célállapothoz vezet.
 - Az n -királynő problémánál igaz ugyan, hogy a problémára adható válasz egy állapot (királynő-elrendezés), de a célállapotot (helyes elrendezést) ebben az esetben is egy alkalmas művelet-sorozattal érhetjük el, azaz végsősorban ilyenkor is a művelet-sorozatok között keresünk, nem az állapotok között.

Reprezentációs gráf bonyolultsága

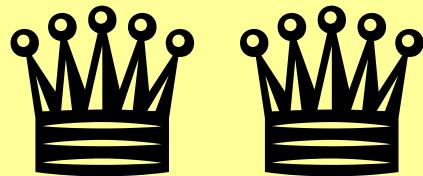
- ❑ A reprezentációs gráf bonyolultságától (a startcsúcsból induló utak számától) függ a problématér mérete, amelyen pedig a keresés hatékonysága múlik.



- ❑ 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

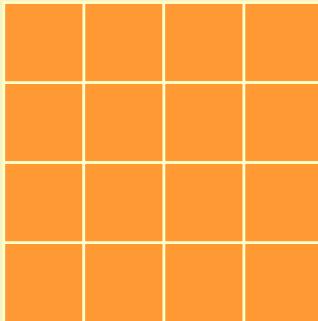
Csökkentsük a problématér méretét

- Ugyanannak a feladatnak több modellje lehet : érdemes olyat keresni, amely kisebb problémateret jelöl ki.
 - Az n -királynő problema modelljénél a problématér mérete (a lehetséges utak száma) óriási. Adjunk jobb modellt!
 - **Bővítsük az állapotteret** az n -nél kevesebb királynőt tartalmazó állásokkal, és használunk 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!**

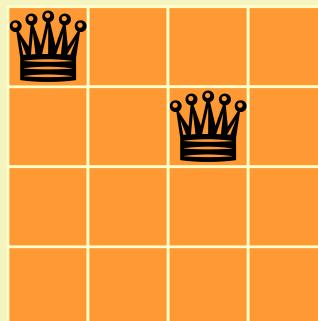


n-királynő probléma 2.

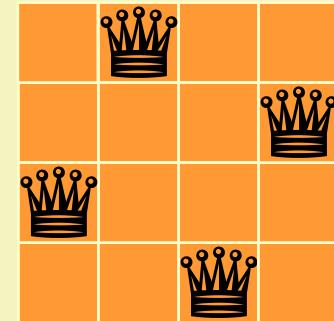
Kezdőállapot:



Közönséges állapot:



Célállapot:



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

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] :=

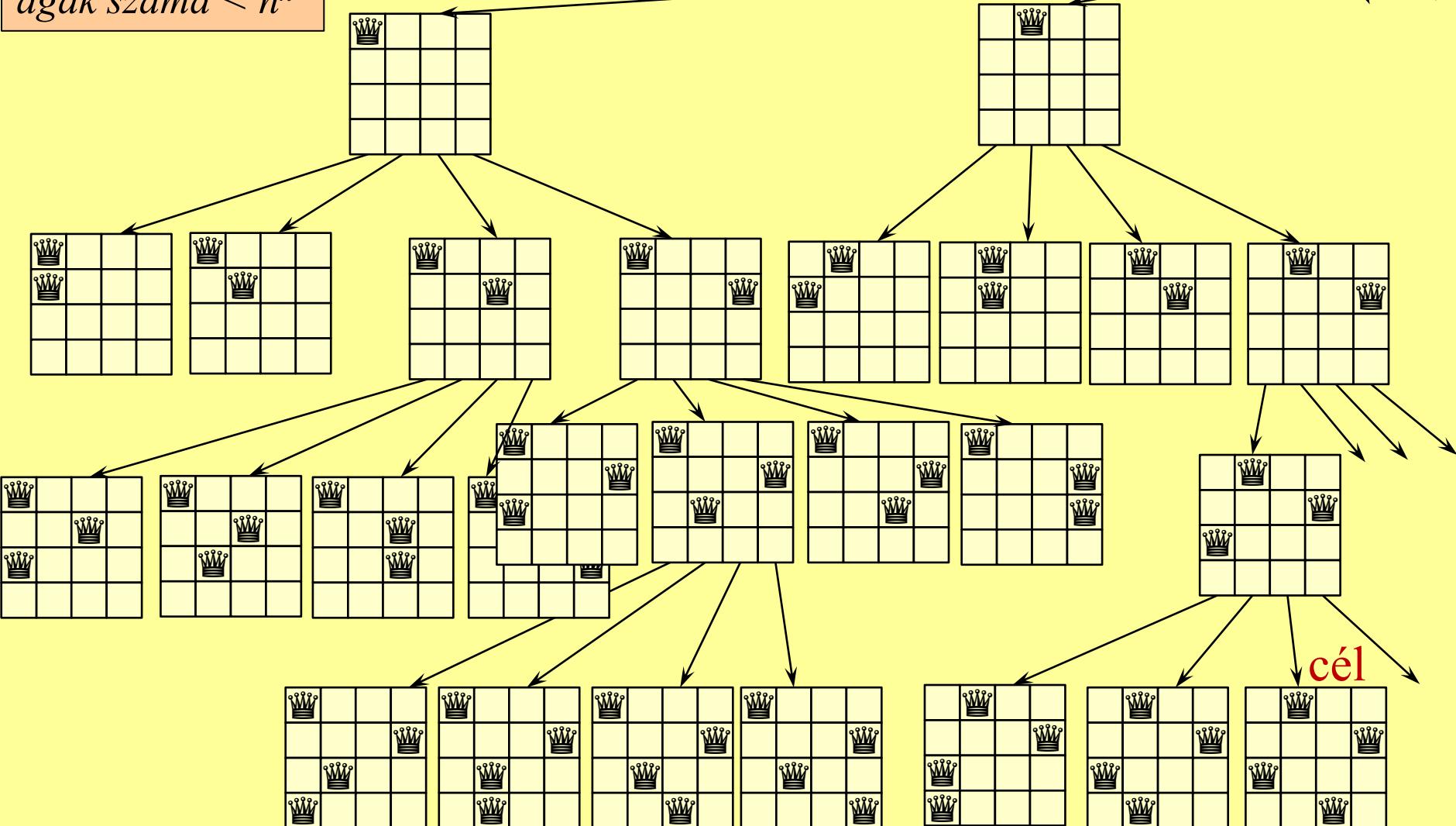
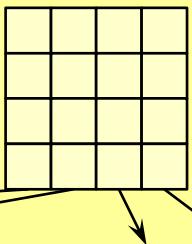
csúcsok száma < $(n^{n+1}-1)/(n-1)$

csúcs ki-foka: n

ágak száma < n^n

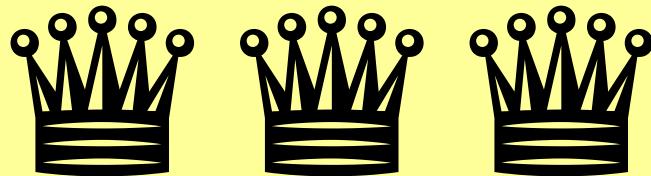
Állapot-gráf

start



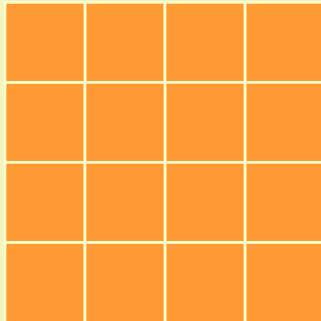
Művelet végrehajtásának hatékonysága

- A művelet kiszámítási bonyolultsága csökkenthető, ha
 - az állapotokat extra információval egészítjük ki: egy állapotban a tábla soron következő üres sorának sorszámát eltárolhatjuk a tábla mellett, így új királynő elhelyezésekor ezt nem kell kiszámolni, ugyanakkor egy művelet végrehajtásakor könnyen aktualizálhatjuk (eggyel növeljük).
 - az invariáns szigorításával szűkítjük az állapotteret: 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 így három féle lehet: szabad, ütés alatt álló vagy foglalt, amelyeket a művelet végrehajtásakor kell karbantartani.



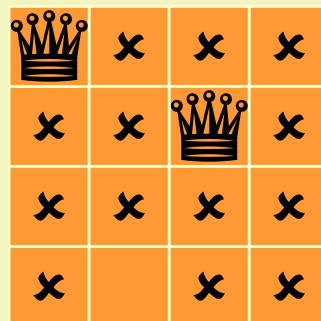
n-királynő probléma 3.

Kezdőállapot:



köv_sor = 1

Középső állapot:



köv_sor = 3

Célállapot:



köv_sor = 5

Állapottér: $AT = rec(t : \{ \text{皇后}, \text{X}, \text{空} \}^{n \times n}, köv_sor : \mathbb{N})$

invariáns: $köv_sor \leq n+1$,

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

jelölés : X egy királynő által ütött üres mezőt jelöli,

— 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] := \ddagger

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

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

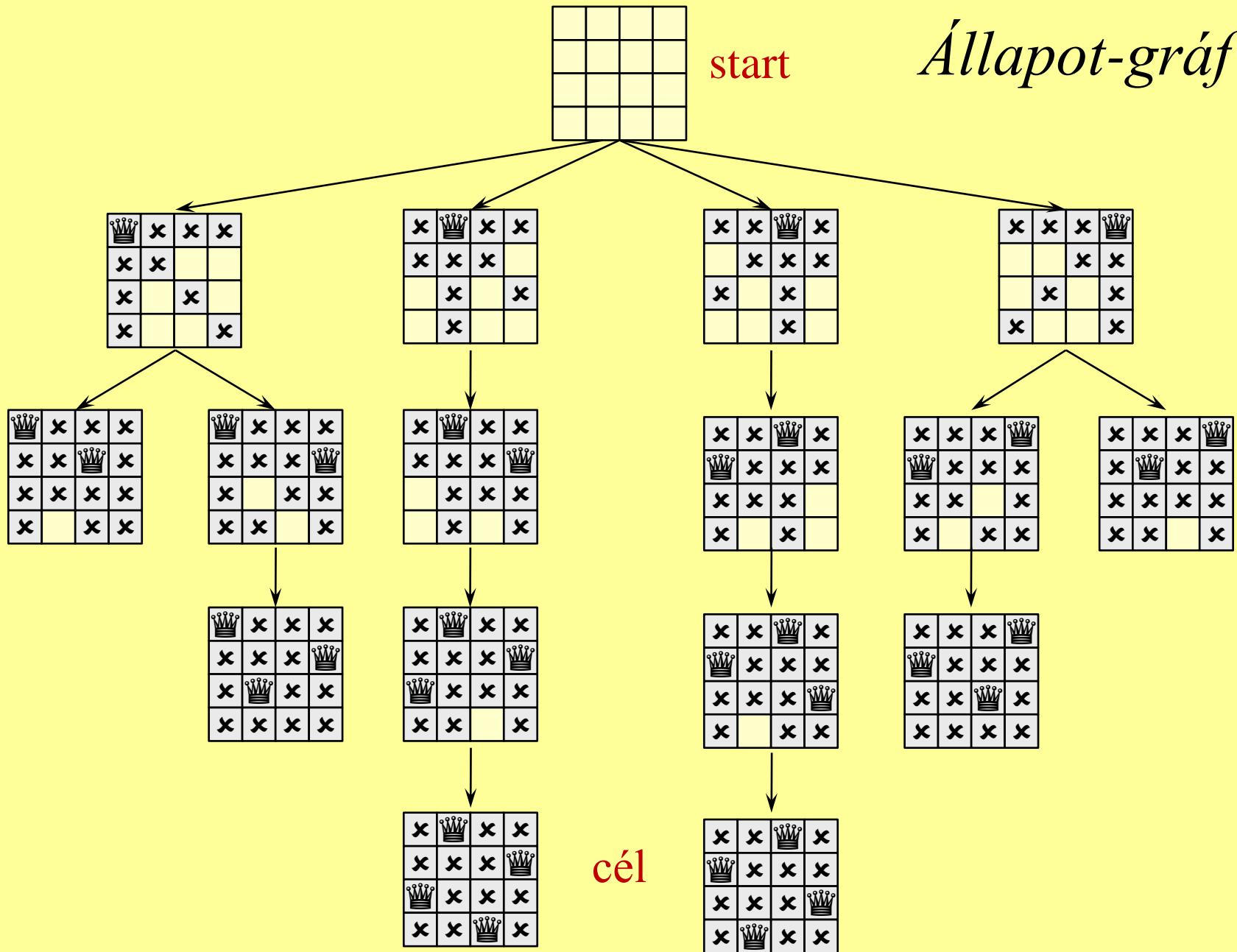
hatás számítás-igénye: lineáris

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)

állapot ~ a kirakó egy konfigurációja

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: $Tol(irány) : AT \rightarrow AT$

koordinátánkénti összeadás

HA

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

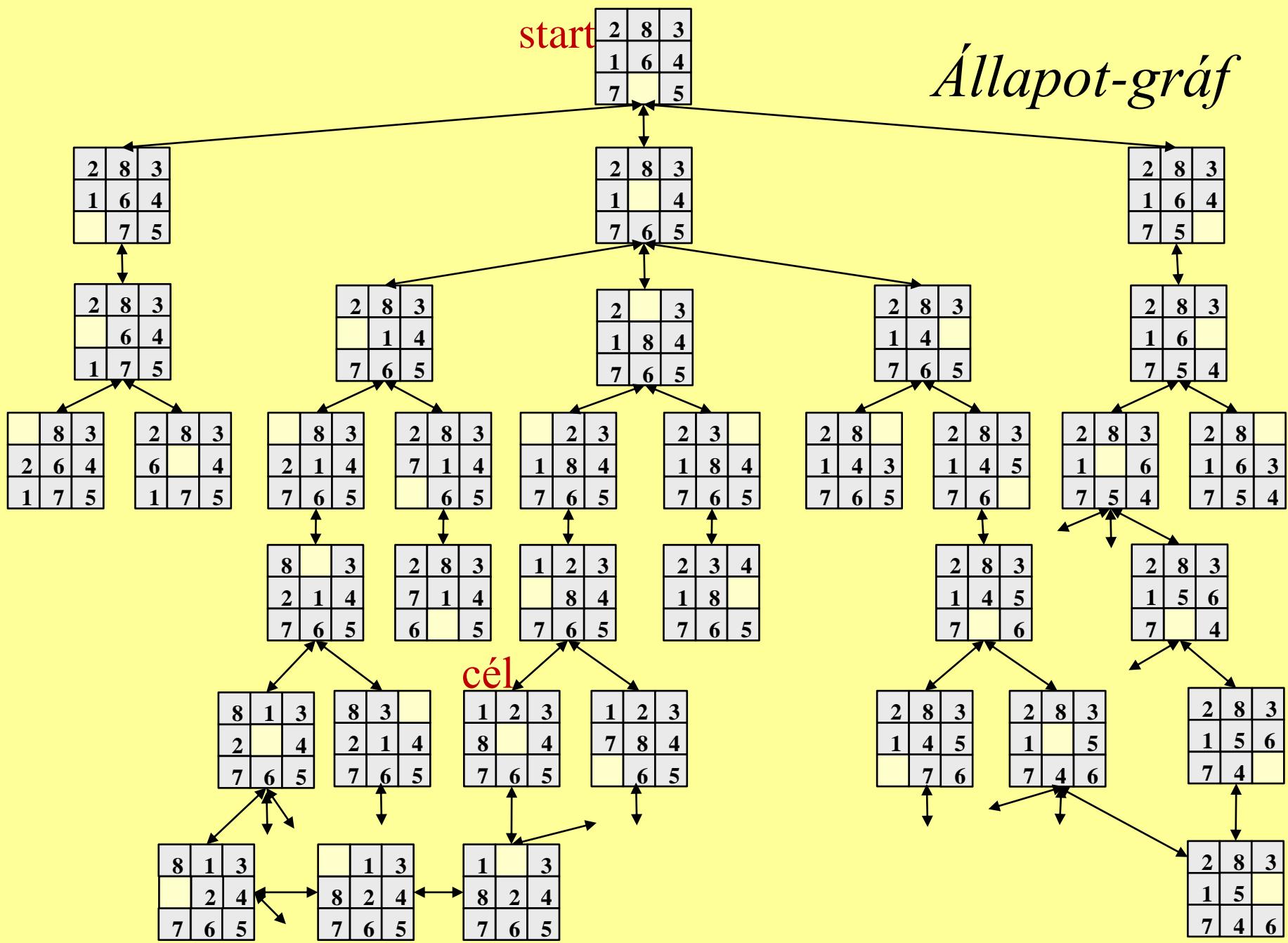
$(1,1) \leq this.\text{üres} + irány \leq (3,3)$ (this: AT)

AKKOR

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

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

Állapot-gráf

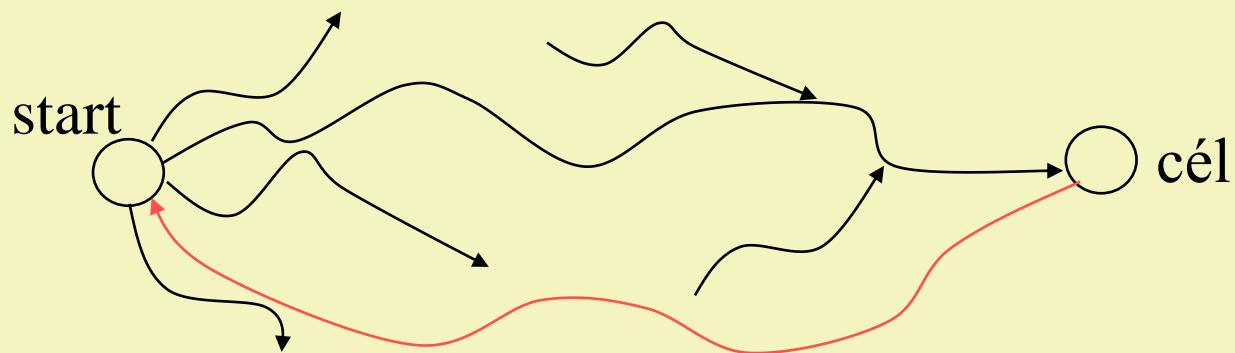


Gregorics Tibor

Mesterséges intelligencia

2. Visszafelé haladó keresés

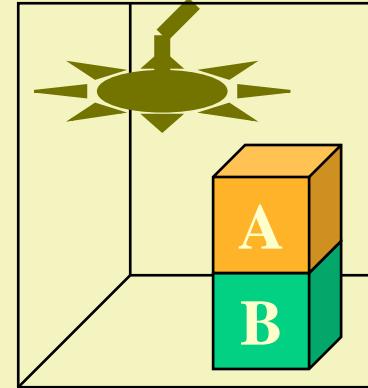
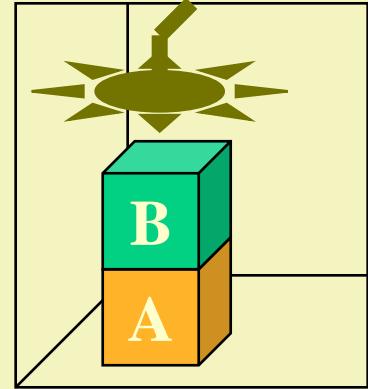
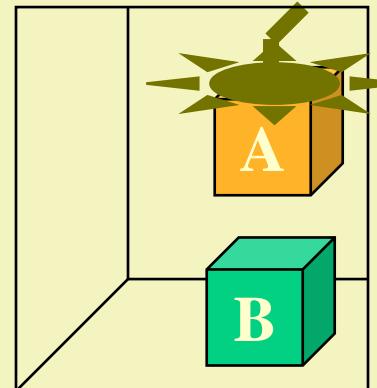
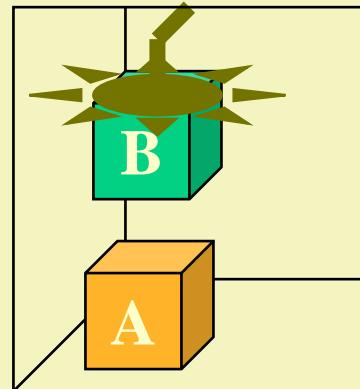
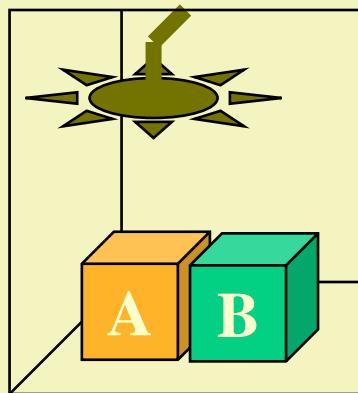
- ❑ Ha a megoldás megtalálása a cél felől nézve egyszerűbb, mint a start felől nézve (mert a problématerület így kevesebb alternatívát mutat), akkor érdemes visszafelé, a cél irányából a start felé haladva keresni a megoldást.



- ❑ Ügyelni kell azonban arra, hogy az így megtalált megoldási utat fordítva, a starttól a cél irányában kell tudni értelmezni.

Kocka világ probléma

start



út a célból a startba:
megoldás:

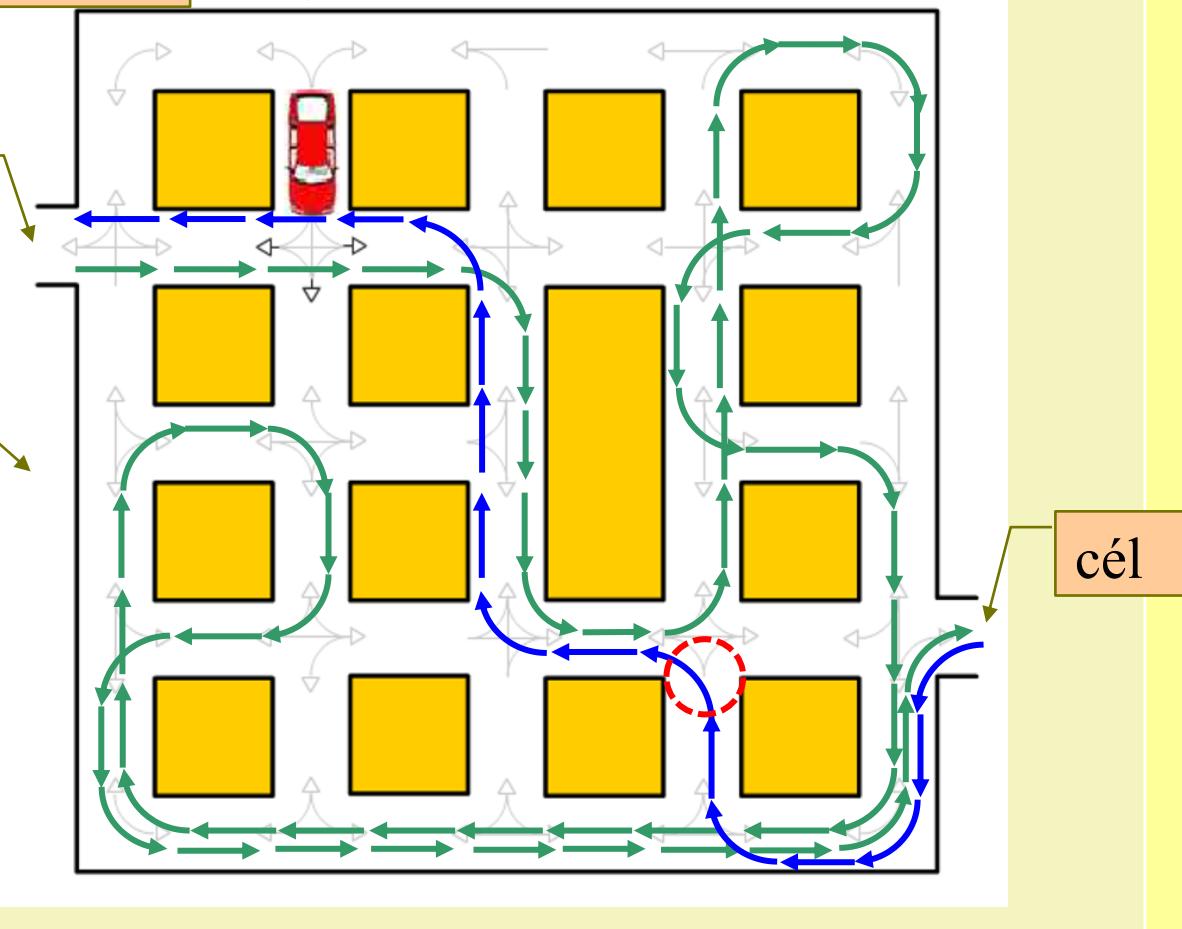
levesz(A,B), letesz(A)
felvesz(A), rátesz(A,B)

cél

Útvonal keresés városban

A cél felől indított kereséssel előállt (kék) úton nem lehet a start felől végig menni.

A start felől indulva nehéz megtalálni a (zöld) utat, mert sok alternatíva közül kell azt kiválasztani.





Kancsók problémája

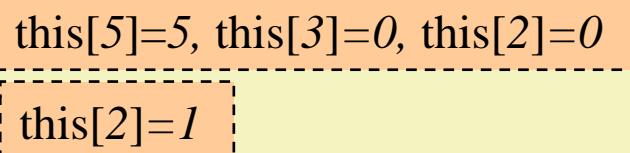
Három kancsóban együttesen 5 liter bor van. Kezdetben az öt literes van tele, a 3 és 2 literes üres. Töltögetéssel érjük el, hogy a 2 literesbe pontosan 1 liter bor kerüljön!

Allapottér: $AT = map(key:Keys, value:\mathbb{N})$ ahol $Keys = \{5, 3, 2\}$

invariáns: $\sum_{i \in Keys} this[i] = 5$ és $\forall i \in Keys : this[i] \leq i$

Kezdőállapot: $[5, 0, 0]$

Célállapot: $[x, y, 1]$

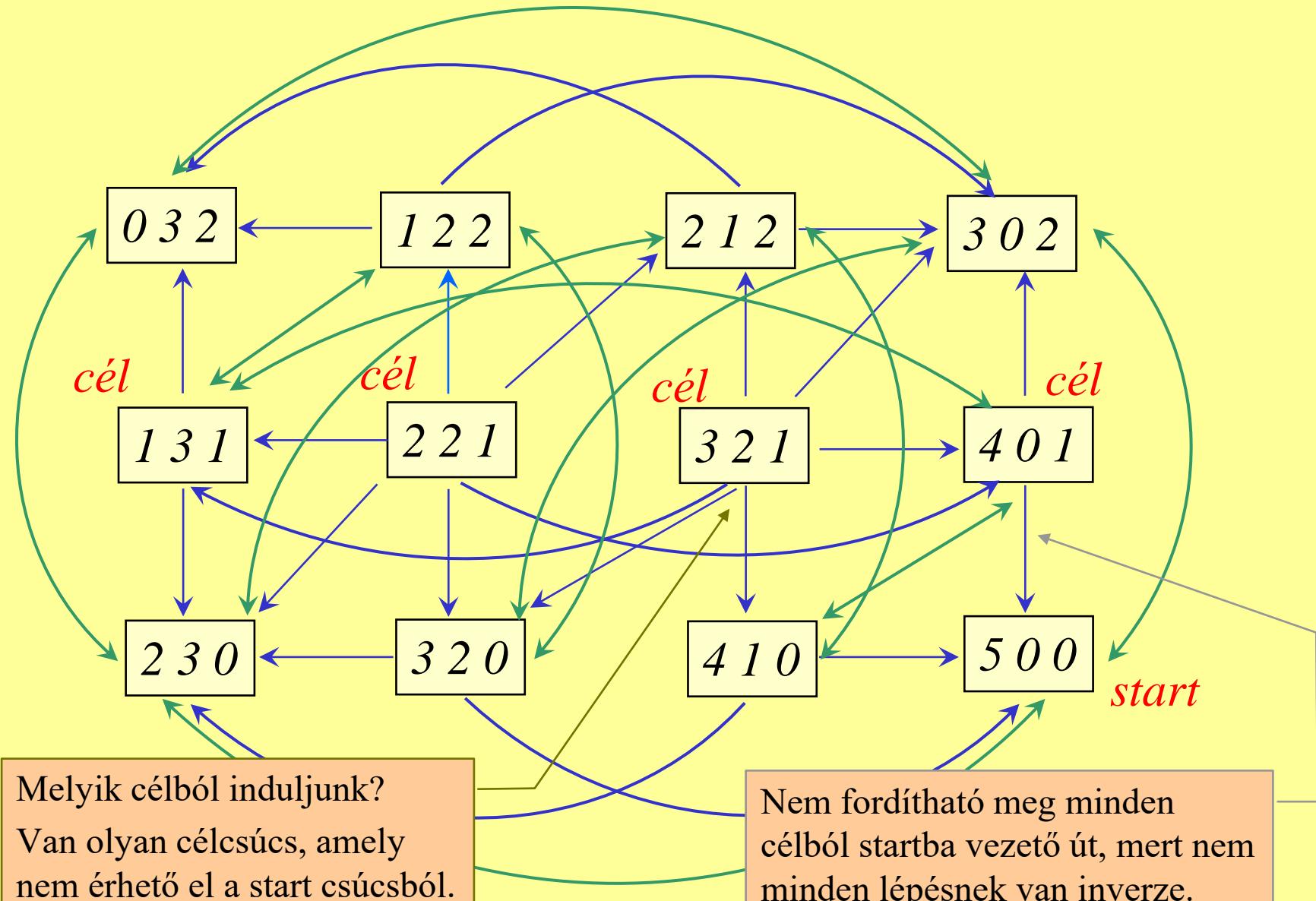


Művelet: $Tölt(i, j) : AT \rightarrow AT$ ($this : AT$)

HA $i, j \in Keys$ és $i \neq j$ és $\min(this[i], j - this[j]) > 0$

AKKOR $this[i], this[j] := this[i] - \min(this[i], j - this[j]),$
 $this[j] + \min(this[i], j - this[j])$

Kancsók-probléma állapot-gráfja



Visszafelé haladó keresés feltételei

- A reprezentációs gráf éleinek kétirányúaknak kell lenni (legalább a visszafelé megtalált megoldási út mentén).
 - Ez állapottér modellezés esetén biztosan teljesül, ha a műveletek van inverze.
- Ismerni kell egy startból elérhető célállapotot.

Ha ezek a feltételek nem állnak fenn, de mégis visszafelé haladó keresést szeretnénk végezni, akkor alkalmazzunk probléma redukciót.

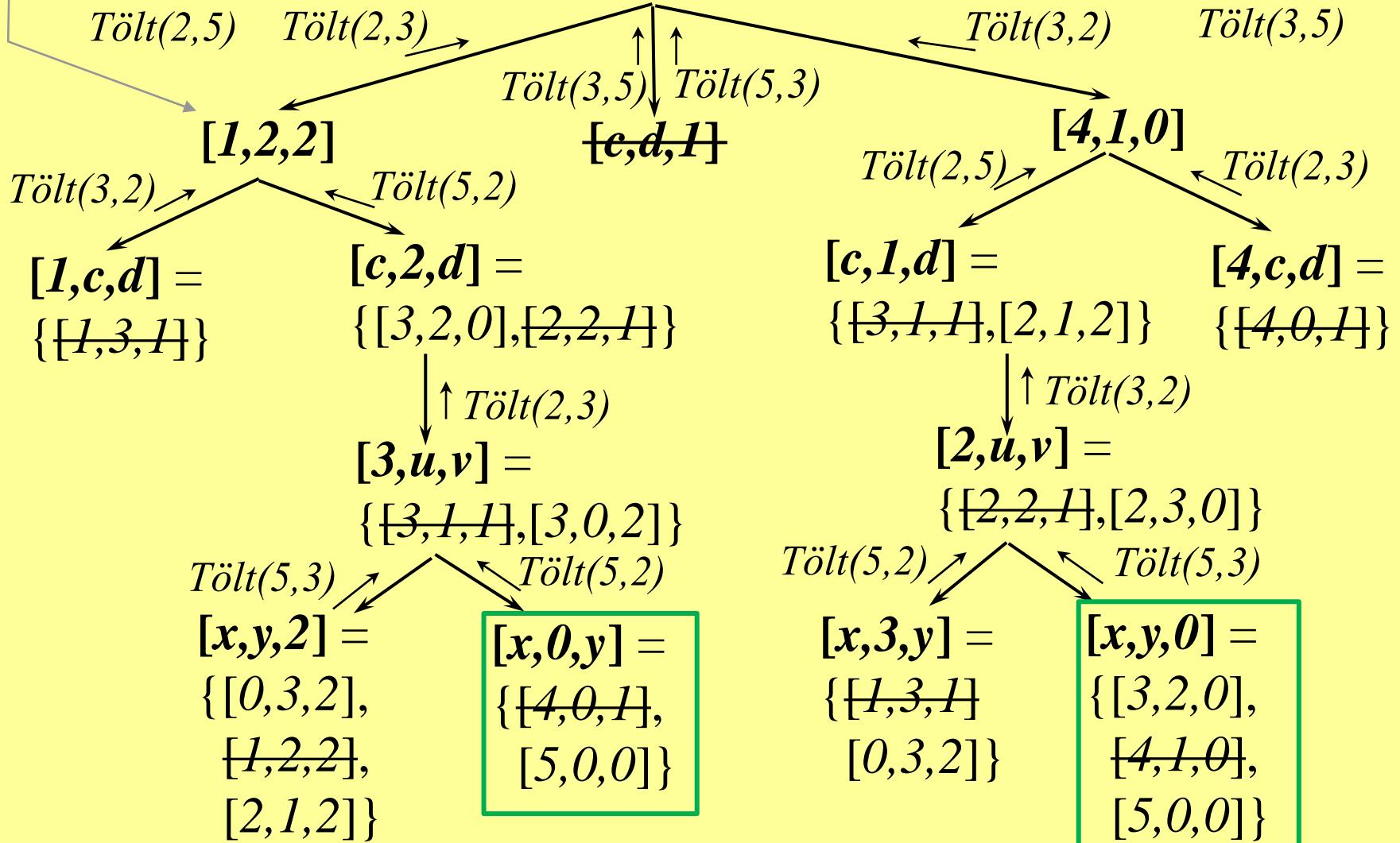
Kancsók probléma redukciós gráfja

Redukció: mely állapotokból lehet egy adott művelettel egy adott állapothalmazba eljutni?

Állapotok helyett, állapotok halmazaival dolgozzunk.

célállapotok

$$[a,b,1] = \{ [4,0,1], [3,1,1], [2,2,1], [1,3,1] \}$$



invariáns: $\sum_{i \in \text{Keys}} \text{this}[i] = 5$ és $\forall i \in \text{Keys}: \text{this}[i] \leq i$

$Tölt(i,j)$: $AT \rightarrow AT$

HA $i, j \in \text{Keys}$ és $i \neq j$ és $\min(\text{this}[i], j - \text{this}[j]) > 0$

AKKOR $\text{this}[i] := \text{this}[i] - \min(\text{this}[i], j - \text{this}[j])$

$\text{this}[j] := \text{this}[j] + \min(\text{this}[i], j - \text{this}[j])$

Kancsó probléma redukciós operátora

$$R_{Tölt(i,j)} : 2^{AT} \rightarrow 2^{AT}$$

állapothalmazból
állapothalmazba

ahol $i, j \in \text{Keys}$ és $i \neq j$

$\mathcal{D}_{Tölt(i,j)}$ -nek az i, j
paraméterekre
vonatkozó része

azon állapotok, amelyeket a $Tölt(i,j)$
a B valamelyik állapotába viszi

$\forall B \in 2^{AT} :$

$$R_{Tölt(i,j)}(B) = \{ a \in AT \mid$$

az állapottér modell
invariánsa

$\sum_{i \in \text{Keys}} a[i] = 5$ és $\forall i \in \text{Keys} : a[i] \leq i$ és

$\min(a[i], j - a[j]) > 0$ és $\mathcal{D}_{Tölt(i,j)}$ -nek a *this*-re (itt *a*-ra) vonatkozó része

$\forall b \in B : b[i] = a[i] - \min(a[i], j - a[j])$ és

a $Tölt(i,j)$ hatása
ahol input: *a* (*this*),
output: *b* (új *this*)

$b[j] = a[j] + \min(a[i], j - a[j])$ és
 $b[k] = a[k]$ (ahol $k \neq i$ és $k \neq j$) }

Probléma-redukciós modell

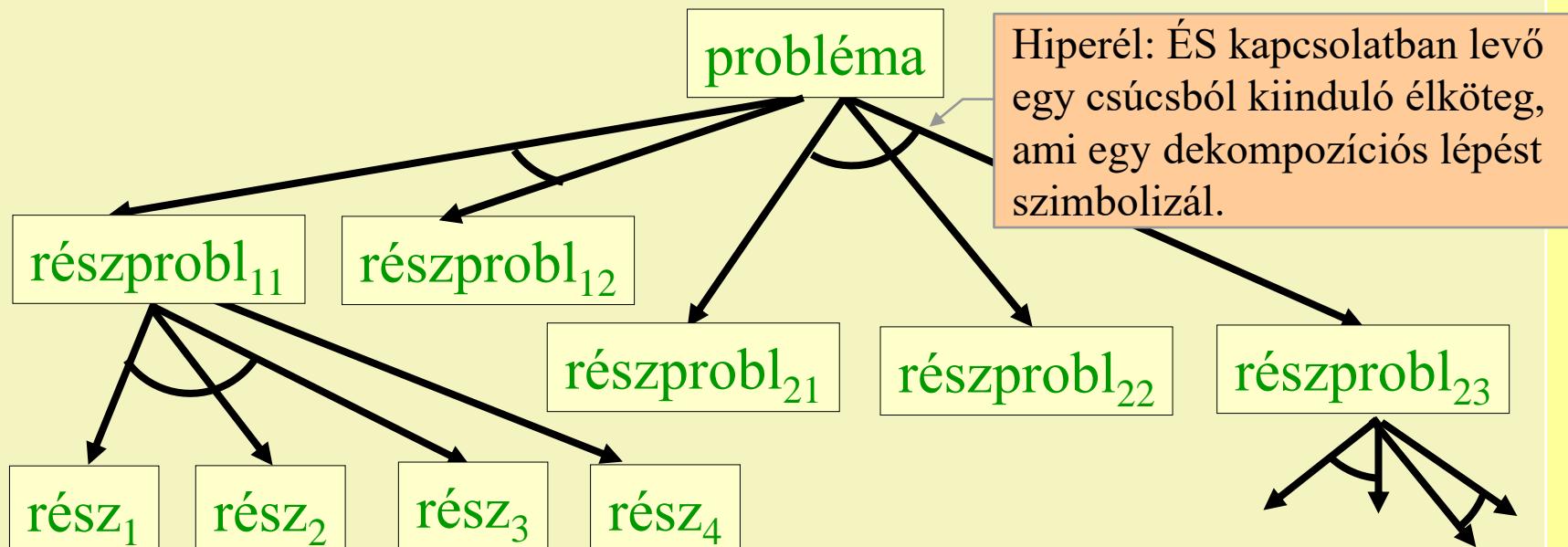
- Adott a probléma valamelyik **állapottér modellje**.
(állapottér: AT , invariáns: $Inv:AT \rightarrow \mathbb{L}$)
- Az állapottér modell minden $M:AT \rightarrow AT$ műveletéhez generálunk egy olyan $R_M:2^{AT} \rightarrow 2^{AT}$ leképezést (**redukciós operátor**), hogy az egy B állapot-halmazhoz azt az A állapot-halmazt rendeli, amely bármelyik állapotából az M művelet a B halmaz valamelyik állapotába vezet el.
 - $\mathcal{D}_{RM} = \{B \in 2^{AT} \mid B \neq \emptyset\}$
 - $\forall B \in \mathcal{D}_{RM} : R_M(B) = \{a \in AT \mid Inv(a) \text{ és } M(a) \in B\}$
- Egy **célhalmaz** csupa (de nem feltétlenül az összes) célállapotból áll.
- Egy **kezdőhalmaz** legalább egy kezdőállapotot tartalmaz.

Megjegyzés

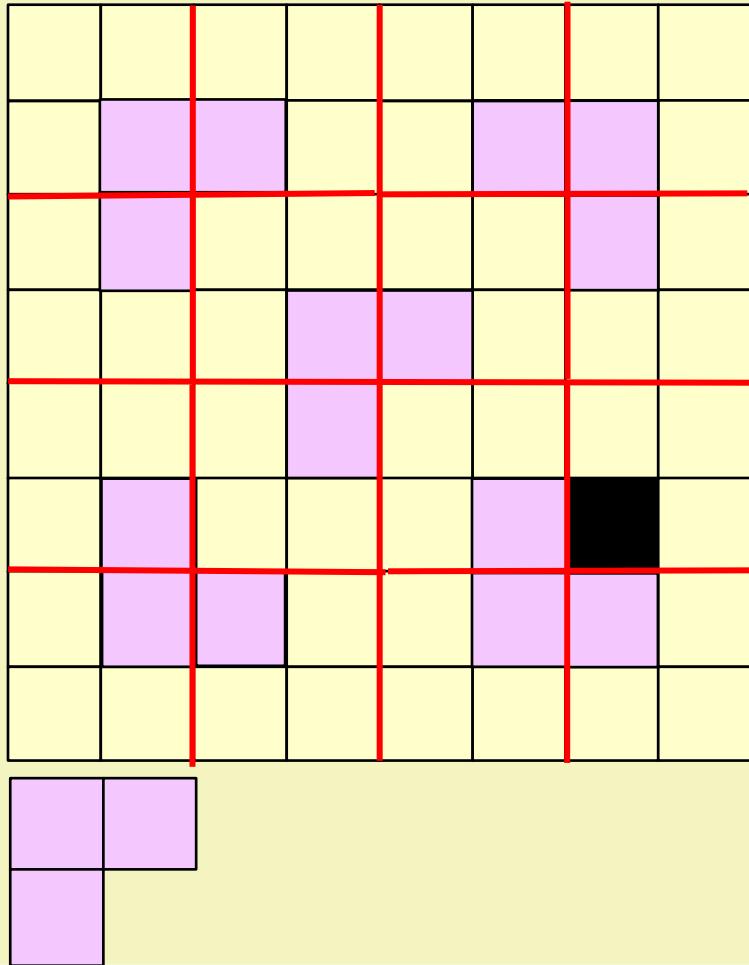
- A feladat olyan redukciós operátor-sorozat megtalálása, amely **célhalmazból kezdőhalmazba** vezet.
- A megoldás ezen sorozat redukciós operátorait generáló műveletek **fordított sorrendben kiolvasott sorozata** lesz.
- A probléma-redukció is modellezhető egy **irányított gráffal**, ahol a csúcsok állapot-halmazokat, az irányított élek a redukálásokat, a **célcsúcs** a célhalmazt, **startcsúcsok** a kezdőhalmazokat szimbolizálják, és a célcsúcsból egy startcsúcsba vezető út keresése a feladat.

3. Probléma dekompozíció

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

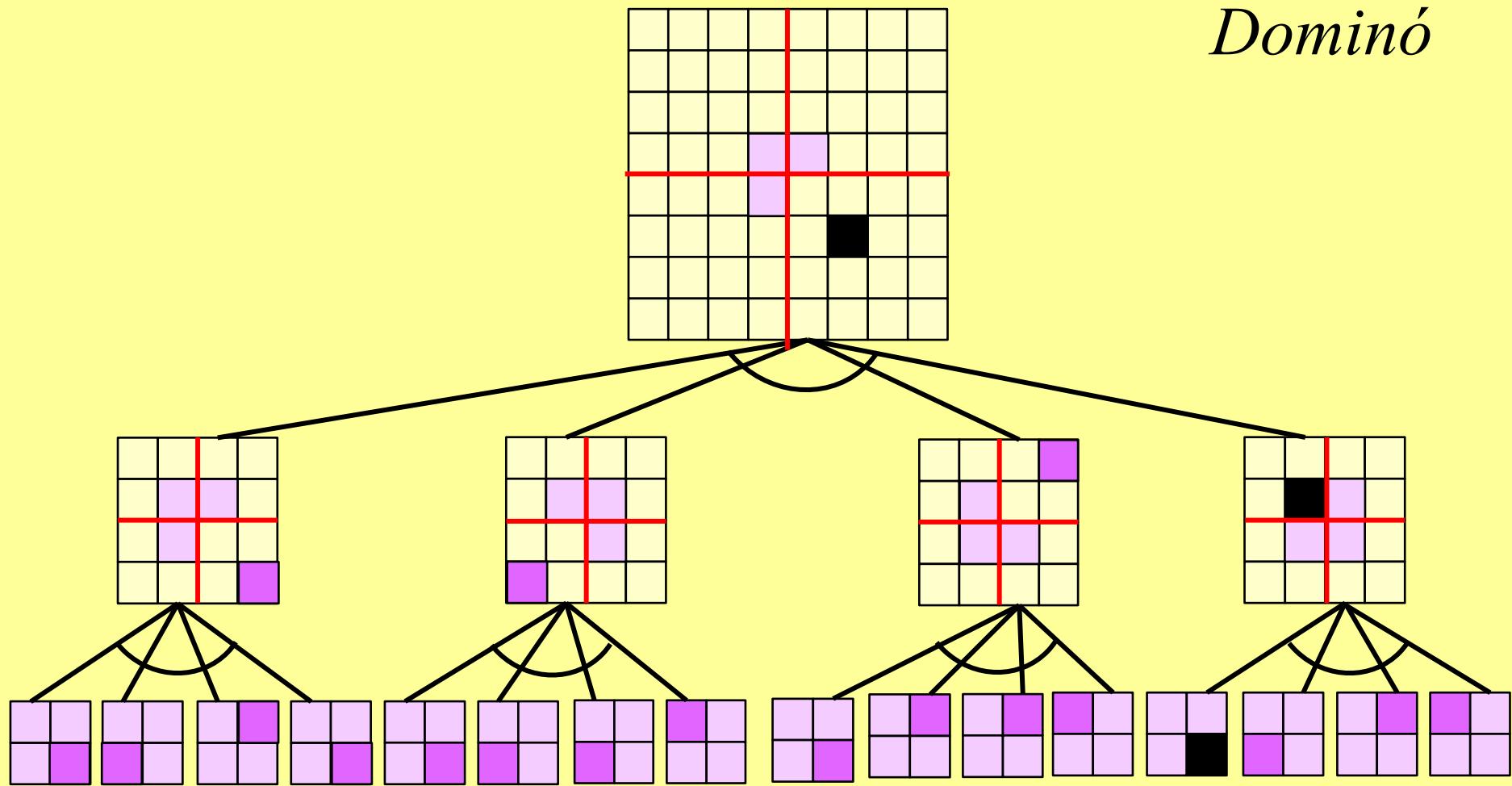


Dominó



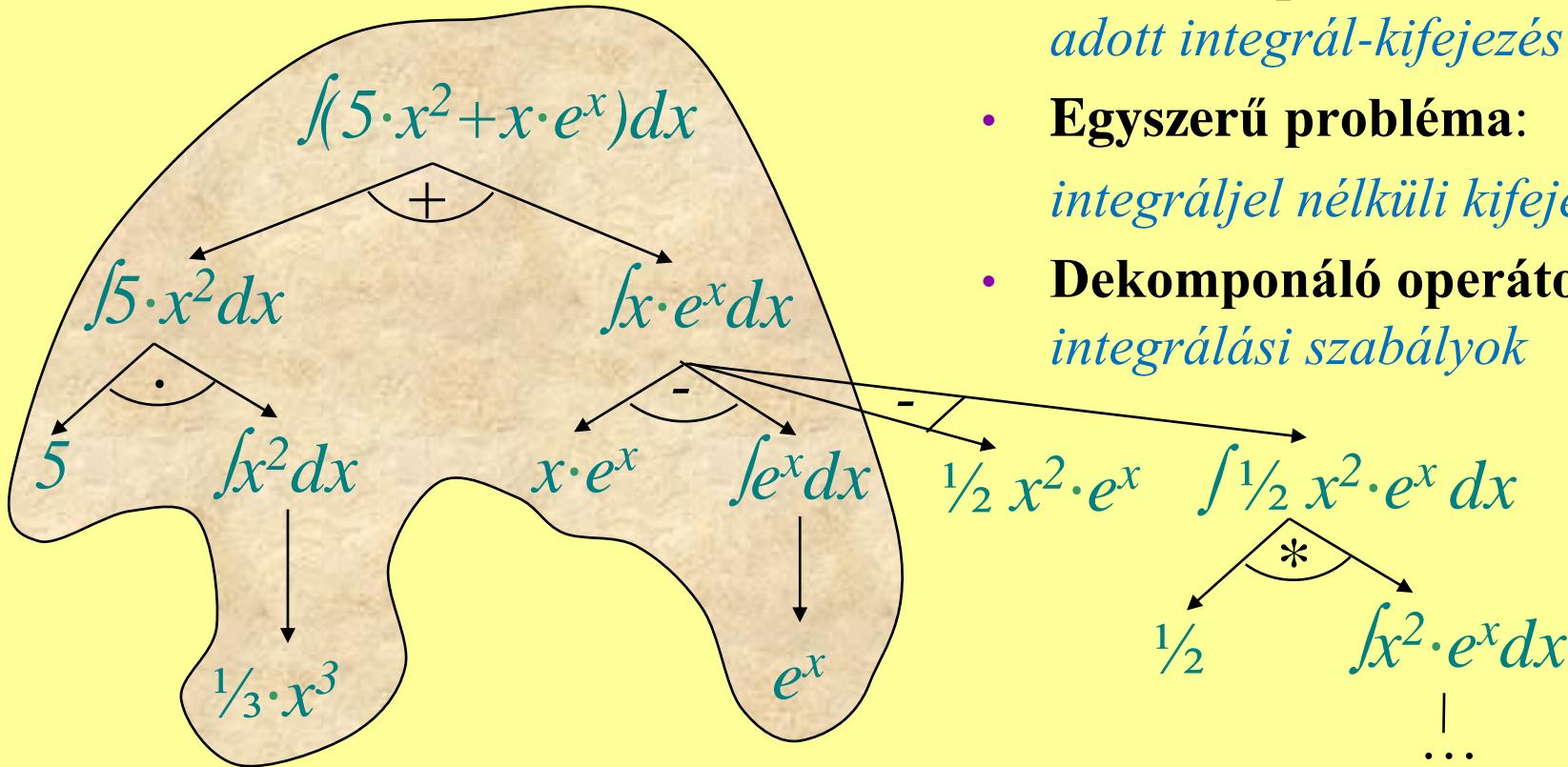
- **Probléma általános leírása:**
 $2^n \times 2^n$ -es tábla egy foglalt mezővel
- **Kiinduló probléma:**
 8×8 -as tábla egy foglalt mezővel
- **Egyszerű probléma:**
 2×2 -es tábla egy foglalt mezővel
- **Dekomponáló operátor:**
felosztja a táblát 4 egyenlő részre
és elhelyez középre egy L alakú
dominót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

Dominó



Megoldás: Egy ÉS/VAGY gráf (fa) szimbolizálja a dekomponáló lépéseket, amely megoldás gráf is egyben. Ennek hiperélei és levélcsúcsai mutatják az L alakú elemek elhelyezését.

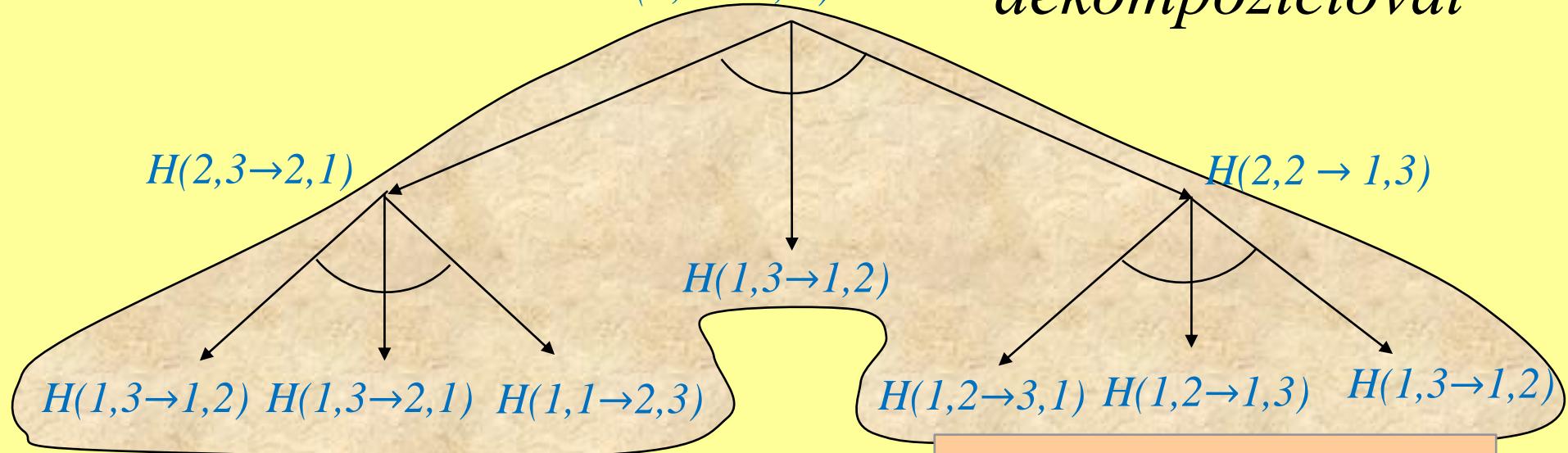
Integrálszámítás



- **Probléma általános leírása:** szintaktikusan helyes kifejezés
- **Kiinduló probléma:** adott integrál-kifejezés
- **Egyszerű probléma:** integráljel nélküli kifejezés
- **Dekomponáló operátorok:** integrálási szabályok

Megoldás: Egy ÉS/VAGY gráf (fa) egy megoldás gráfjának leveleiben tárolt szimbólumokat kell balról jobbra haladva a megfelelő hiperélekhez rendelt algebrai műveletekkel összekötni ahhoz, hogy az eredeti integrálkifejezés primitív függvényét megkapjuk.

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: $H(3, 3 \rightarrow 1, 2)$

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

eldönthető, hogy megoldható-e

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: A keresett lépéssorozat a reprezentációs fa (ami egyben
megoldásgráf is) leveleiből olvashatók ki balról jobbra haladva.

Probléma dekompozíciós modell

- 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
$$D(p) = \langle p_1, \dots, p_n \rangle$$

Probléma dekompozíció ÉS/VAGY gráffal

- | | |
|--|--|
| <ul style="list-style-type: none">❑ Dekompozíciós modell<ul style="list-style-type: none">○ részprobléma○ dekomponáló művelet hatása egy problémára<ul style="list-style-type: none">• egy problémára véges sok művelet alkalmazható (véges kifokú gráf)○ művelet költsége<ul style="list-style-type: none">• van a műveltek költségének alsó pozitív korlátja (δ)○ kiinduló probléma○ megoldható probléma❑ Gráf-reprezentáció: ÉS/VAGY gráf, startcsúcs, célcímsúcsok<ul style="list-style-type: none">○ dekompozíciós folyam<ul style="list-style-type: none">~ hiperút~ megoldás-gráfból olvasható ki | <ul style="list-style-type: none">ÉS/VAGY gráf~ csúcs~ irányított hiperél~ hiperél költsége~ startcsúcs~ célcímsúcs |
|--|--|



Keresések

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)

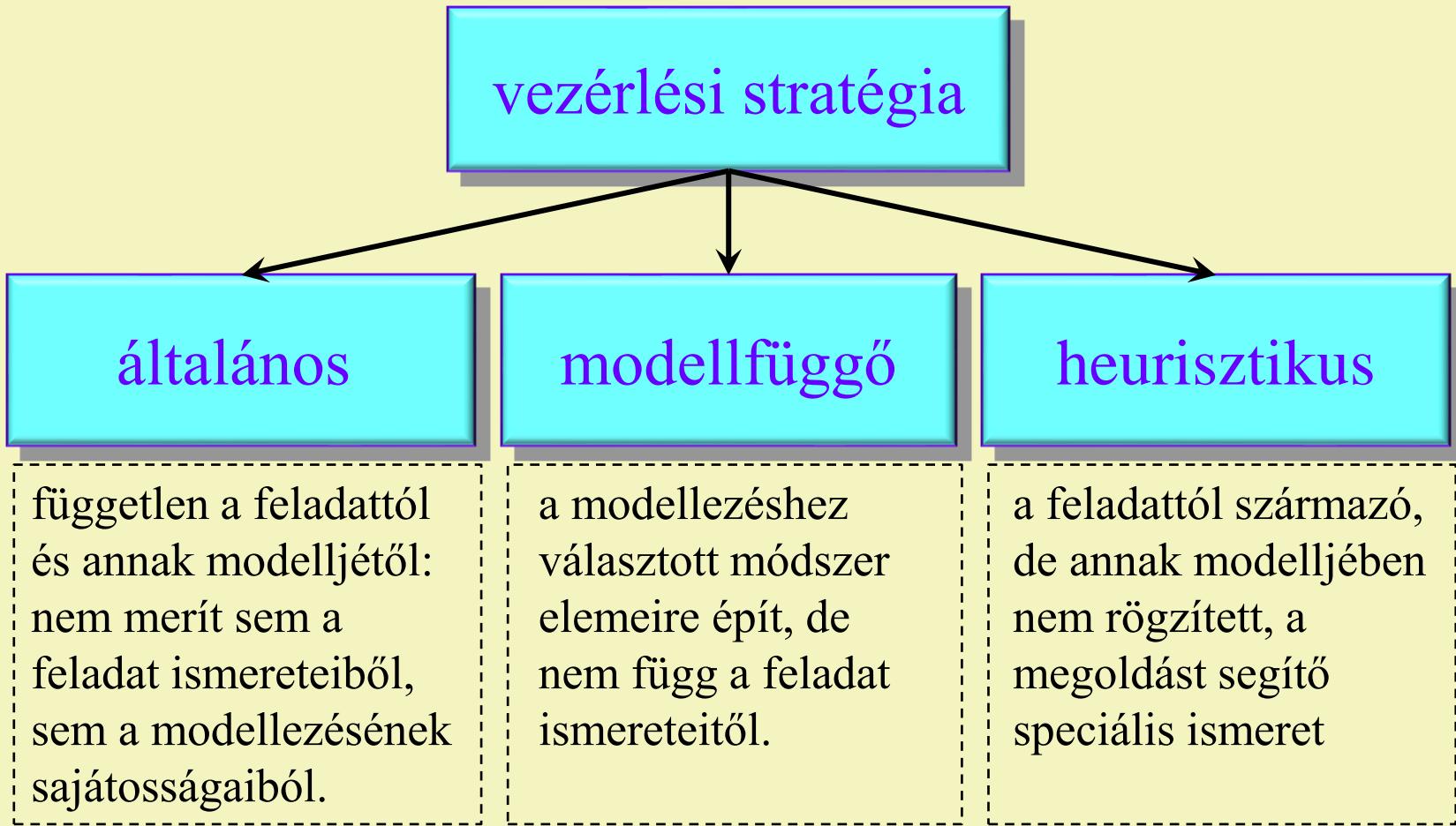
keresési szabályok

megváltoztatják a globális
munkaterület tartalmát
(előfeltétel, hatás)

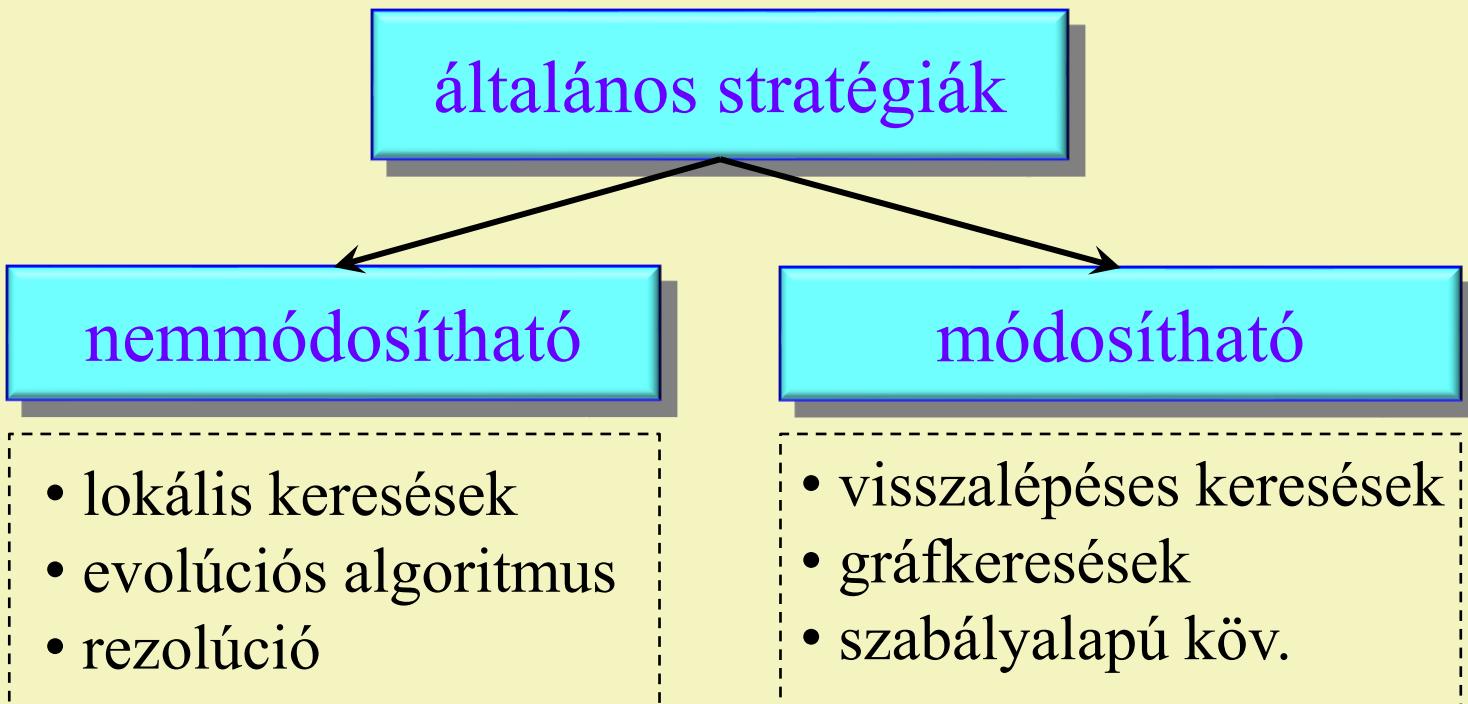
vezérlési stratégia

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

KR vezérlési szintjei



Általános vezérlési stratégiák



1. Lokális keresések

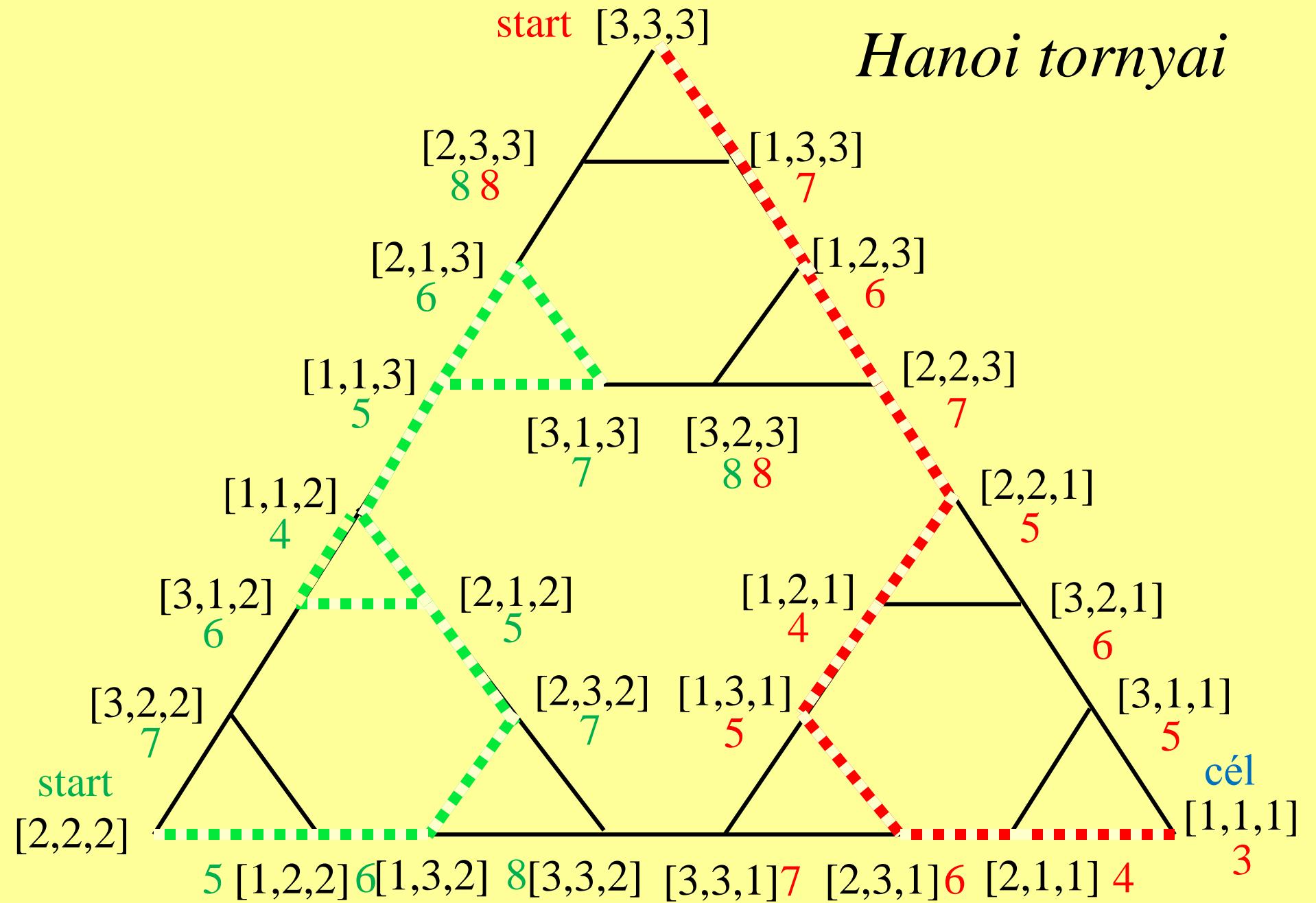
- A lokális keresés olyan KR, amely a probléma reprezentációs gráfjának egy kis részét tárolja (a globális munkaterületén).
 - Kezdetben a startcsúcsot ismeri, és
 - akkor áll le, ha a célcímsúcs megjelenik a látókörében, vagy valamilyen okból nem tud tovább keresni.
- Az eltárolt részgráf csúcsait a részgráf szűk környezetéből vett „jobb” csúcsokra cseréli le (keresési szabály).
- A „jobbság” eldöntéséhez (vezérlési stratégia) egy kiértékelő függvényt (cél-, rátermettségi-, heurisztikus függvényt) használ, amely reményeink szerint annál jobb értéket ad egy csúcsra, minél közelebb esik az a célhoz.

Hegymászó módszer

- A **globális munkaterület** egy **aktuális csúcsot** (*akt*), és annak azt a szülőjét ($\pi(akt)$) tárolja el, amely a megelőző aktuális csúcs volt.
 - Kezdetben a startcsúcs lesz az aktuális csúcs.
 - Terminál, ha az aktuális csúcs célcsúcs vagy zsákutca.
- Egy **keresési szabály** az aktuális csúcsot cseréli le annak **egy gyerekére** ($\Gamma(akt)$).
- A **vezérlési stratégia** mindenkorán a szabályt választja, amelyik az aktuális csúcs **legjobb** – de lehetőleg nem a szülőcsúcsként nyilvántartott – gyerekére lép.

Megjegyzés: Egy másik változata a hegymászó algoritmusnak nem engedi meg, hogy az aktuális csúcsot egy rosszabb értékű csúcsra cseréljük (ilyenkor a keresés inkább leáll).

Hanoi tornyai



ADAT := kezdeti érték

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

 SELECT SZ FROM alkalmazható szabályok

 ADAT := SZ(ADAT)

endloop

Hegymászó módszer algoritmusa

1. $akt := start$

2. **while** $akt \notin T$ **loop**

3. $akt := \arg \text{opt}_f(\Gamma(akt) - \pi(akt))$

$\Gamma(akt) \sim akt$ gyermekei
 $\pi(akt) \sim akt$ egy szülője

4. **endloop**

5. **return** akt

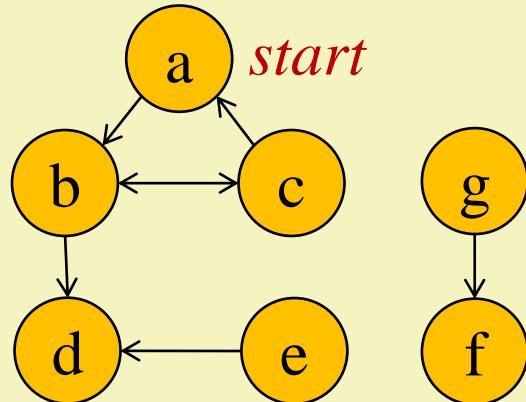
if $\Gamma(akt) = \emptyset$ **then return** nem talált megoldást
if $\Gamma(akt) = \{\pi(akt)\}$ **then** $akt := \pi(akt)$
else $akt := \arg \text{opt}_f(\Gamma(akt) - \pi(akt))$

A bejárt út megadásához az
akt egymás után felvett
értékeit is össze kell gyűjteni.

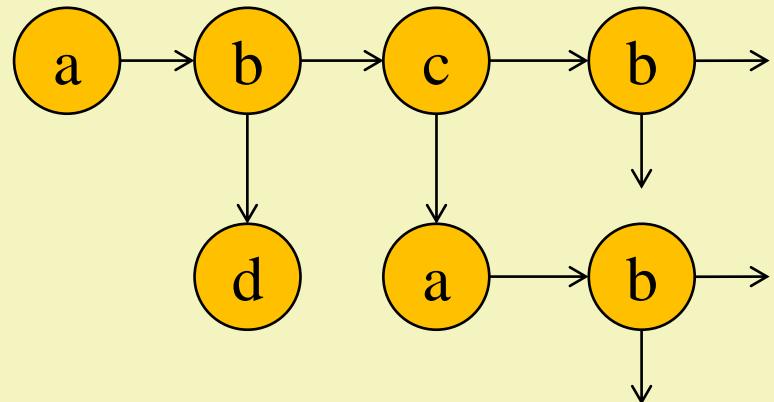
Hogyan „látja” egy keresés a reprezentációs gráfot?

- ❑ Egy keresés fokozatosan fedez fel a reprezentációs gráfot: bizonyos részeihez soha nem jut el, de a selfedezett részt sem feltétlenül tárolja el teljesen, sőt, sokszor **torzultan „látja”** azt: ha például egy csúcshoz érve nem vizsgálja meg, hogy ezt korábban már selfedezte-e, hanem új csúcsként regisztrálja, akkor az **eredeti gráf helyett egy fát** fog „látni”.

eredeti gráf:



keresés által látott gráf:



Reprezentációs gráf „fává egyenesítése”

- Ha a keresés nem vizsgálja meg, hogy egy csúcsot korábban már felfedezett-e, akkor valójában a reprezentációs gráfnak a fává **kiegyenesített** változatában keres.
Előny: eltűnnék a körök, de a megoldási utak megmaradnak
Hátrány: duplikátumok jelennek meg, sőt a körök kiegyenesítése végtelen hosszú utakat eredményez
- A kétirányú (oda-vissza) élek drasztikusan megnövelik a kiegyenesítéssel kapott fa méretét. Olcsóbb, ha mindenkorjuk egy csúcsnak azt a szülőcsúcsát, amelyik felől a csúcsot elértek. Így egy csúcsból a szülőjébe **visszavezető él** könnyen felismerhető és **figyelmen kívül hagyható**.

Hegymászó módszer értékelése

- ❑ Előny: könnyű implementálni
- ❑ Hátrányok:
 - Csak erős heurisztika esetén lesz sikeres: különben „eltéved” (nem talál megoldást), sőt zsákutcában „beragad” (leáll). Segíthet, ha:
 - véletlenül választott startcsúcsból újra- és újra elindítjuk
→ random restart local search
 - k darab aktuális csúcs legjobb k darab gyerekére lépünk
→ local beam search
 - gyengítjük és véletlenítjük a mohó stratégiáját
→ simulated annealing
 - Lokális optimum hely körül vagy ekvidisztans felületen (azonos értékű szomszédos csúcsok között) található körön, végtelen működésbe eshet. Segíthet, ha:
 - növeljük a memóriát
→ tabu search

Tabu keresés

- A **globális munkaterület** eltárolja az **aktuális csúcsot** (*akt*), a keresés során **utoljára érintett néhány csúcsot** (*Tabu*), és az **eddig legjobb csúcsot** (*opt*).
 - Kezdetben az *akt* és az *opt* a startcsúcs, a *Tabu* pedig üres.
 - Terminál, ha az *opt* célcímsúcs vagy régóta nem változik, illetve az *akt* egy zsákutca.
- Egy **keresési szabály** az **aktuális csúcsot cseréli le a legjobb gyerekére**, aktualizálja a *Tabu* halmazt (a lecserélt aktuális csúcsot elhelyezi benne: a Tabu egy „sor adatszerkezet”), és ha *akt* jobb, mint az *opt*, akkor ***opt* új értéke az *akt* lesz**.
- A **vezérlési stratégia** minden esetben a szabályt választja, amelyik az aktuális csúcsnak a **legjobb – de a *Tabu* halmazban nem tárolt – gyerekére** lép.

ADAT := kezdeti érték

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

 SELECT SZ FROM alkalmazható szabályok

 ADAT := SZ(ADAT)

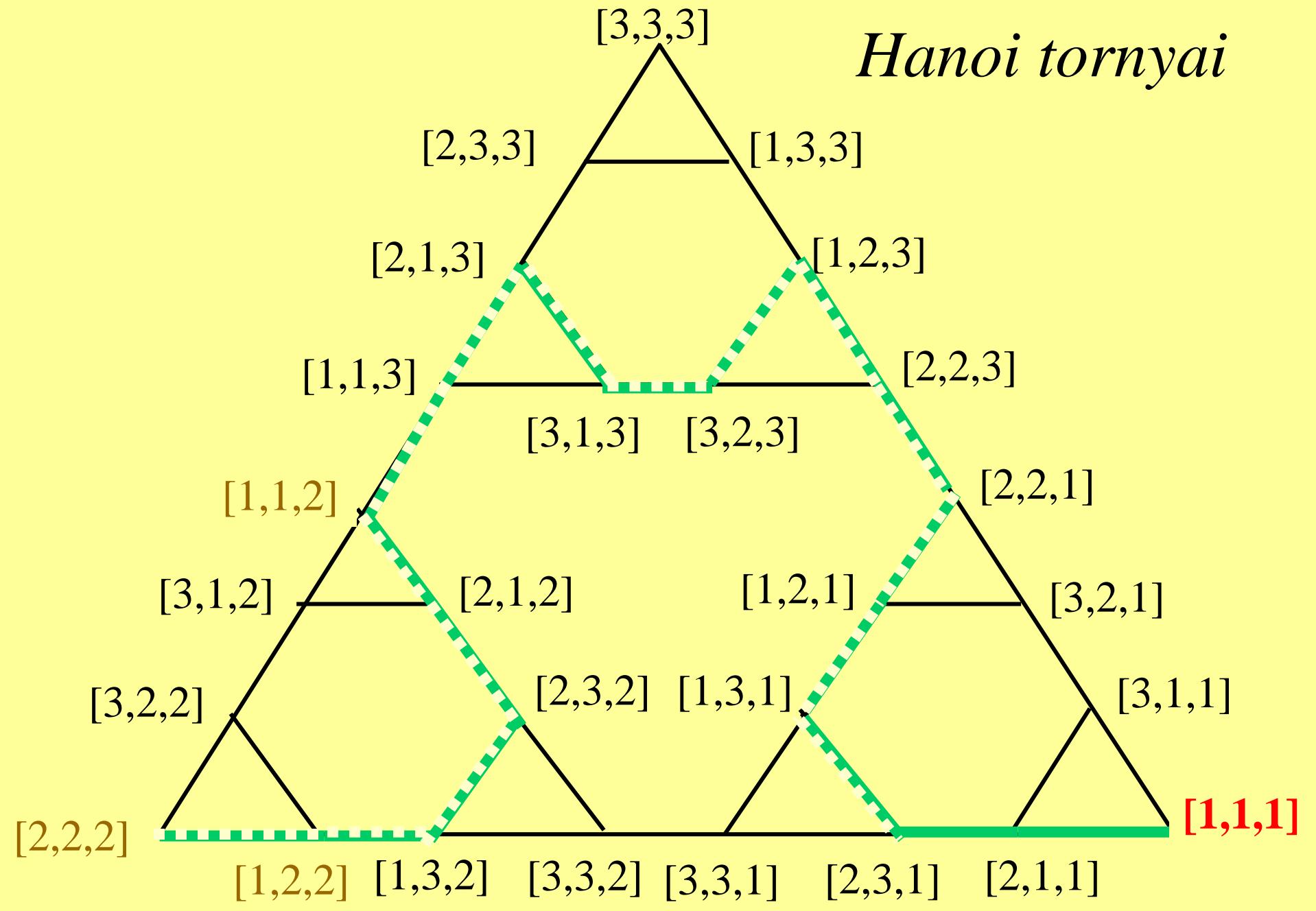
endloop

Tabu keresés algoritmusa

1. $akt, opt, Tabu := start, start, \emptyset$
2. **while** **not** ($opt \in T$ **or** opt régóta nem változik) **loop**
3. $akt := \arg \text{opt}_f(\Gamma(akt) - Tabu)$
4. $Tabu := \text{Módosít}(akt, Tabu)$
5. **if** $f(akt)$ jobb, mint $f(opt)$ **then** $opt := akt$
6. **endloop**
7. **return** akt

if $\Gamma(akt) = \emptyset$ **then return** nem talált megoldást
else if $\Gamma(akt) \subseteq Tabu$ **then** $akt := \arg \text{opt}_f(\Gamma(akt))$
else $akt := \arg \text{opt}_f(\Gamma(akt) - Tabu)$

Hanoi tornyai



Tabu keresés értékelése

Előnyök:

- tabu méreténél rövidebb köröket észleli, és ez segíthet a lokális optimum hely illetve az ekvidisztans felület körüli körök leküzdésében.

Hárányok:

- a *Tabu* halmaz méretét kísérletezéssel kell belőni
- zsákutcába futva a nem-módosítható stratégia miatt beragad

Szimulált hűtés

- Csak a vezérlési stratégiája tér el a hegymászó módszertől: az aktuális csúcs (*akt*) gyermekei közül véletlenszerűen választ új csúcsot (*új*), és azt akkor fogadja el aktuális csúcsnak, ha a függvény-értéke viszonylag jó :
 - ha az *új* csúcs kiértékelő függvény-értéke nem rosszabb, mint az *akt* csúcsé ($f(\text{új}) \leq f(\text{akt})$), akkor elfogadja.
 - ha az *új* csúcs függvényértéke rosszabb ($f(\text{új}) > f(\text{akt})$), akkor az *új* csúcs elfogadásának valószínűsége fordítottan arányos az $|f(\text{akt}) - f(\text{új})|$ különbséggel:

$$e^{\frac{f(\text{akt}) - f(\text{új})}{T}} > \text{random } [0,1]$$

Hűtési ütemterv

- ❑ Egy csúcs elfogadásának valószínűségét az elfogadási képlet kitevőjének T együtthatójával szabályozhatjuk.
- ❑ Ehhez egy (T_k, L_k) $k=1,2,\dots$ ütemtervet készítünk, amely L_1 lépésen keresztül T_1 , majd L_2 lépésen keresztül T_2 , stb. lesz.

$$e^{\frac{f(akt)-f(\text{új})}{T_k}} > \text{rand}[0,1]$$

$$f(\text{új})=120, f(akt)=107$$

- ❑ Ha T_1, T_2, \dots szigorúan monoton csökken, akkor egy ugyanannyival rosszabb függvényértékű új csúcsot kezdetben nagyobb valószínűsséggel fogad el a keresés, mint később.

T	$\exp(-13/T)$
10^{10}	0.9999...
50	0.77
20	0.52
10	0.2725
5	0.0743
1	0.000002

ADAT := kezdeti érték

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

 SELECT SZ FROM alkalmazható szabályok

 ADAT := SZ(ADAT)

endloop

Szimulált hűtés algoritmusa

1. $akt := start ; k := 1 ; i := 1$
2. **while** **not**($akt \in T$ **or** $f(akt)$ régóta nem változik) **loop**
3. **if** $i > L_k$ **then** $k := k+1; i := 1$
4. $\acute{u}j := select(\Gamma(akt) - \pi(akt))$
5. **if** $f(\acute{u}j) \leq f(akt)$ **or** $e^{\frac{f(akt)-f(\acute{u}j)}{T_k}} > rand[0,1]$
 $f(\acute{u}j) > f(akt)$ **and** $e^{\frac{f(akt)-f(\acute{u}j)}{T_k}} > rand[0,1]$
6. **then** $akt := \acute{u}j$
7. $i := i+1$
8. **endloop**
9. **return** akt

if $\Gamma(akt) = \emptyset$ **then return** nem talált megoldást
if $\Gamma(akt) = \{\pi(akt)\}$ **then** $\acute{u}j := \pi(akt)$
else $\acute{u}j := select(\Gamma(akt) - \pi(akt))$



Gráfszínezés

❑ Feladat:

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

❑ Cél:

- A gráf csúcsainak olyan minimális osztályból álló osztályozását keressük, ahol egy osztályba tartozó csúcsok között nem vezet él.
- Majd az egyes osztályokba sorolt csúcsokat lehet ugyanolyan színűre színezni, a felhasznált színek száma pedig az osztályok száma lesz.

Gráfszínezés állapottér modellje

- Állapot: a csúcsoknak egy „gyengített” osztályozása, ahol
 - egy osztályhoz tartozó csúcsok között lehetnek élek
 - a gráf maximális fokszámánál több osztály van, de lehet egy osztály üres is
- Művelet: Egy osztályból egy csúcsot egy másik osztályba helyez át.
- Kezdő állapot: tetszőleges
- Célállapot: a legjobb osztályozás
(minél kevesebb első néhány osztályban legyenek csak csúcsok, amelyek között ne legyen él)
- Állapot-gráf: Exponenciális méretű az eredeti gráf csúcosszámához mérve.

Gráfszínezés kiértékelő függvénye

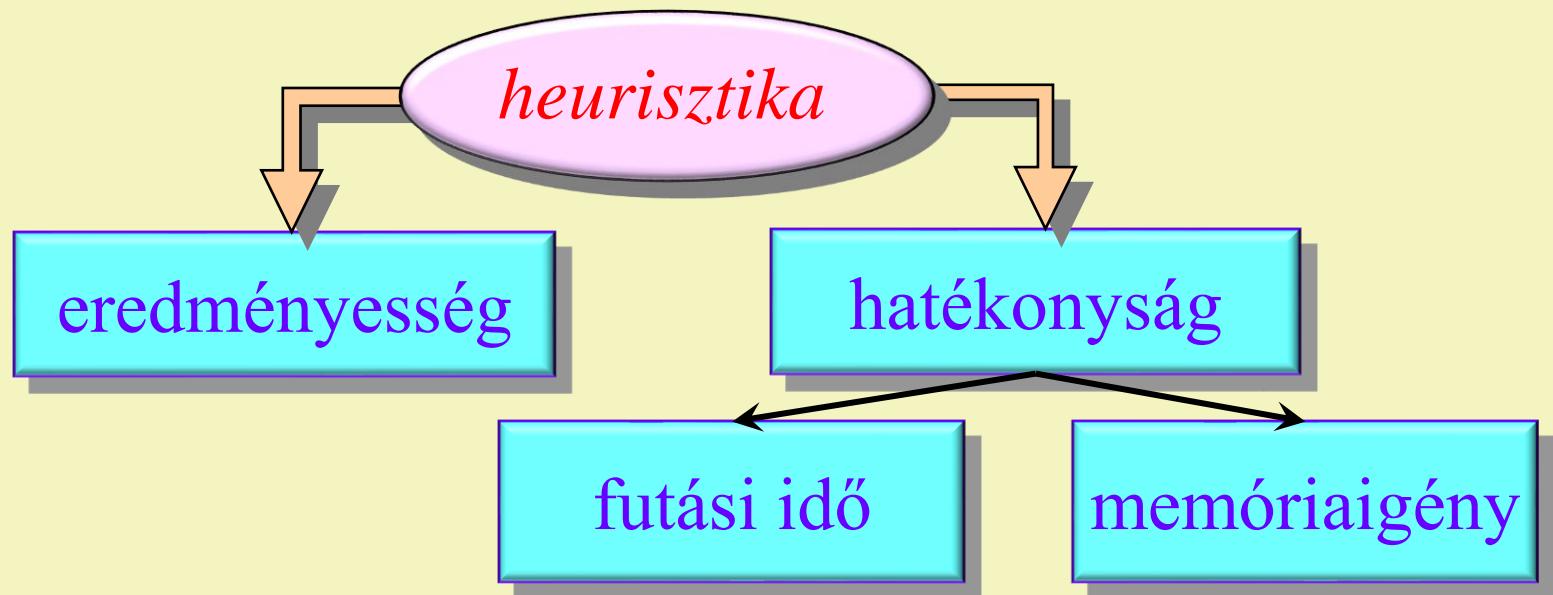
- Annál jobb egy (O_1, \dots, O_k) osztályozás,
 - minél több csúcs van az első néhány osztályában (ezáltal minél több üres osztálya van), és
 - minél kevesebb egy osztályon belül vezető élek száma.
- $f(n) = \sum_j w_j (\lambda |A(O_j)| - |O_j|)$
 - ahol $A(O_j)$ az O_j osztálybeli élek halmaza
 - a $w_j > 0$ számok szigorúan növő sorozatot alkotnak.
- Könnyű a „szomszédos” osztályozás kiértékelő függvény értékét kiszámolni.

Lokális kereséssel megoldható feladatok

- ❑ A sikerhez az kell, hogy egy lokálisan hozott rossz döntés ne zárja ki a cél megtalálását!
 - Ez például egy **erősen összefüggő** reprezentációs-gráfban automatikusan teljesül, de kifejezetten előnytelen, ha a reprezentációs-gráf egy irányított fa. (Például az n -királynő problémát csak tökéletes kiértékelő függvény esetén lehetne lokális kereséssel megoldani.)
- ❑ **Erős heurisztika** nélkül nincs sok esély a cél megtalálására.
 - **Jó heurisztikára** épített kiértékelő függvénnnyel elkerülhetőek a zsákutcák, a körök.

A heurisztika hatása a KR működésére

- A heurisztika olyan, a feladathoz kapcsolódó ötlet, amelyet közvetlenül építünk be egy algoritmusba azért, hogy annak eredményessége és hatékonysága javuljon (egyszerre képes javítani a futási időt és a memóriaigényt), habár erre általában nem ad garanciát.



ADAT := kezdeti érték

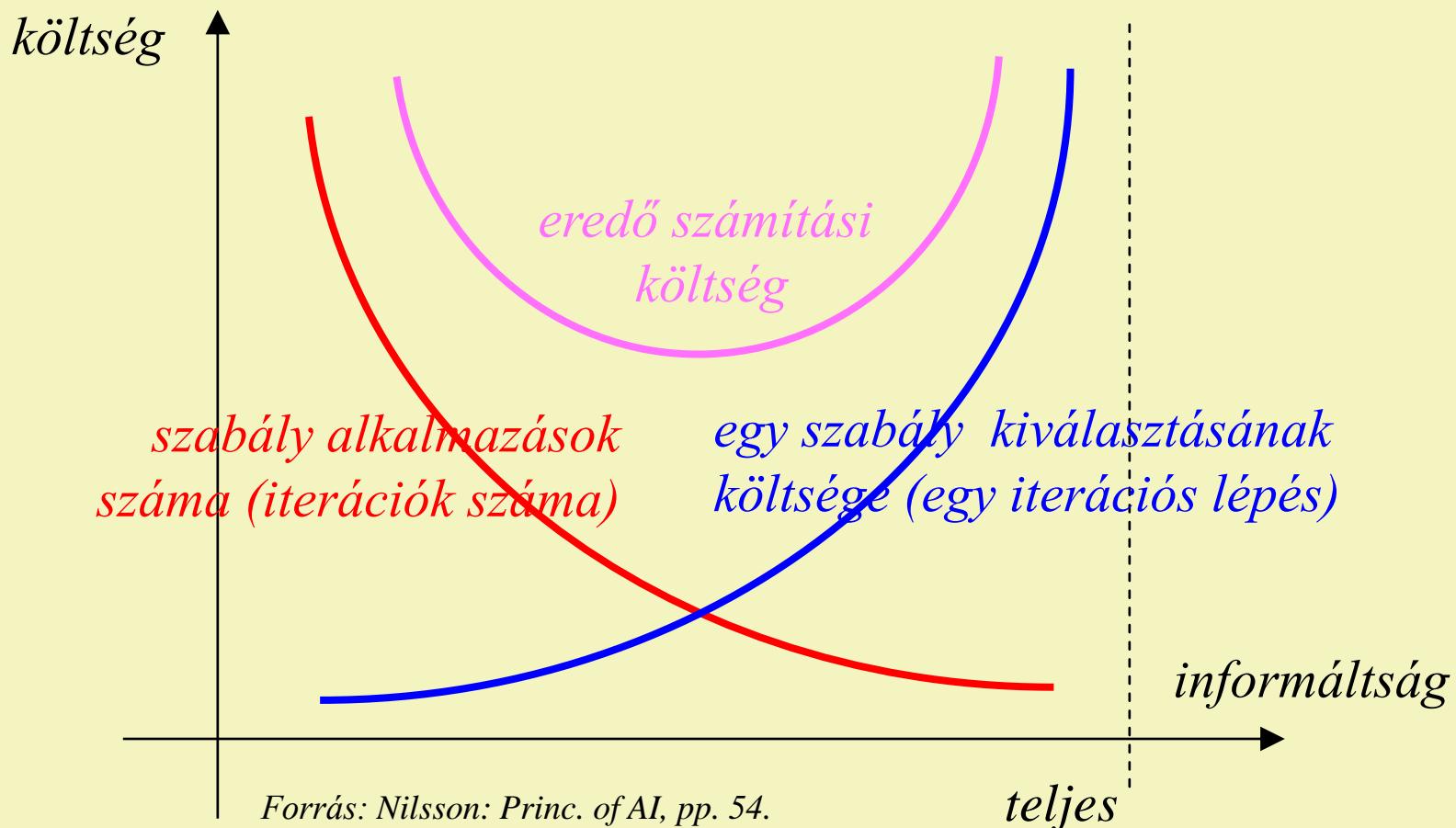
while \neg terminálási feltétel(ADAT) **loop**

 SELECT SZ FROM alkalmazható szabályok

 ADAT := SZ(ADAT)

endloop

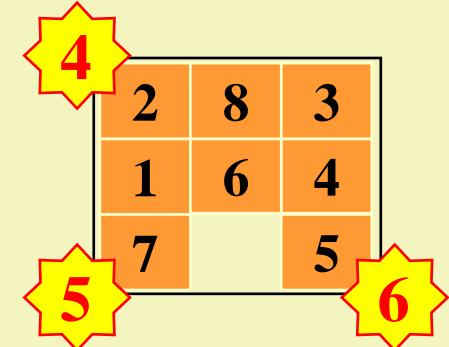
Heurisztika és KR hatékonysága



1	2	3
8		4
7	6	5

Heurisztikák a 8-as (15-ös) tologató játékra

- Rossz helyen levő lapkák száma (**W**): azon lapkák száma, amelyek nem a célbeli helyükön vannak.
- Manhattan (**P**): a lapkák célbeli helyüktől vett minimális távolságainak (függőleges és vízszintes mozgatásai számának) összege
- Keret (**F**): büntető pontokat ad
 - +1 minden olyan lapkára a szélen, amelyet nem a célbeli szomszédja követ az óra járásával megegyező irányban,
 - +2 minden olyan sarokra, ahol nem a cél szerinti lapka áll.



W

Hegymászó módszer

modell függő vezérlés:
műveletek rögzített sorrendje
left, up, right, down

2	8	3
1	6	4
7		5

2	8	3
1		4
7	6	5

2	8	3
	1	4
7	6	5

l:5, u:(3), r:5, d:-

l:(3), u:3, r:4, d:

l:-, u:(3), r:, d:4

8	1	3
2		4
7	6	5

8		3
2	1	4
7	6	5

	8	3
2	1	4
7	6	5

l:(3), u:, r:4, d:4

l:, u: -, r:4, d:(3)

l: -, u: -, r:(3), d:

8	1	3
	2	4
7	6	5

	1	3
8	2	4
7	6	5

1		3
8	2	4
7	6	5

1	2	3
8		4
7	6	5

l: -, u:(2), r:, d:4

l: -, u: -, r:(1), d:

l:, u: -, r:2, d:0

P

Hegymászó módszer

5

2	8	3
1	6	4
7		5

4

2	8	3
1		4
7	6	5

l:6, u:4, r:6, d:-

l:5, u:3, r:5, d:
W: l:3, u:3, r:4, d:

2

	2	3
1	8	4
7	6	5

3

2		3
1	8	4
7	6	5

l:-, u:-, r:, d:1

l:2, u:-, r:4, d:

1

1	2	3
	8	4
7	6	5

0

1	2	3
8		4
7	6	5

l:-, u:, r:0, d:2

Mesterséges intelligencia

F

Hegymászó módszer

6

2	8	3
1	6	4
7		5

5

2	8	3
1		4
7	6	5

l:8, u:5, r:8, d:-

l:5, u:3, r:6, d:
W: l:3, u:3, r:4, d:

3

	2	3
1	8	4
7	6	5

3

2		3
1	8	4
7	6	5

l:-, u:-, r:, d:1

l:3, u:-, r:5, d:

1

1	2	3
	8	4
7	6	5

0

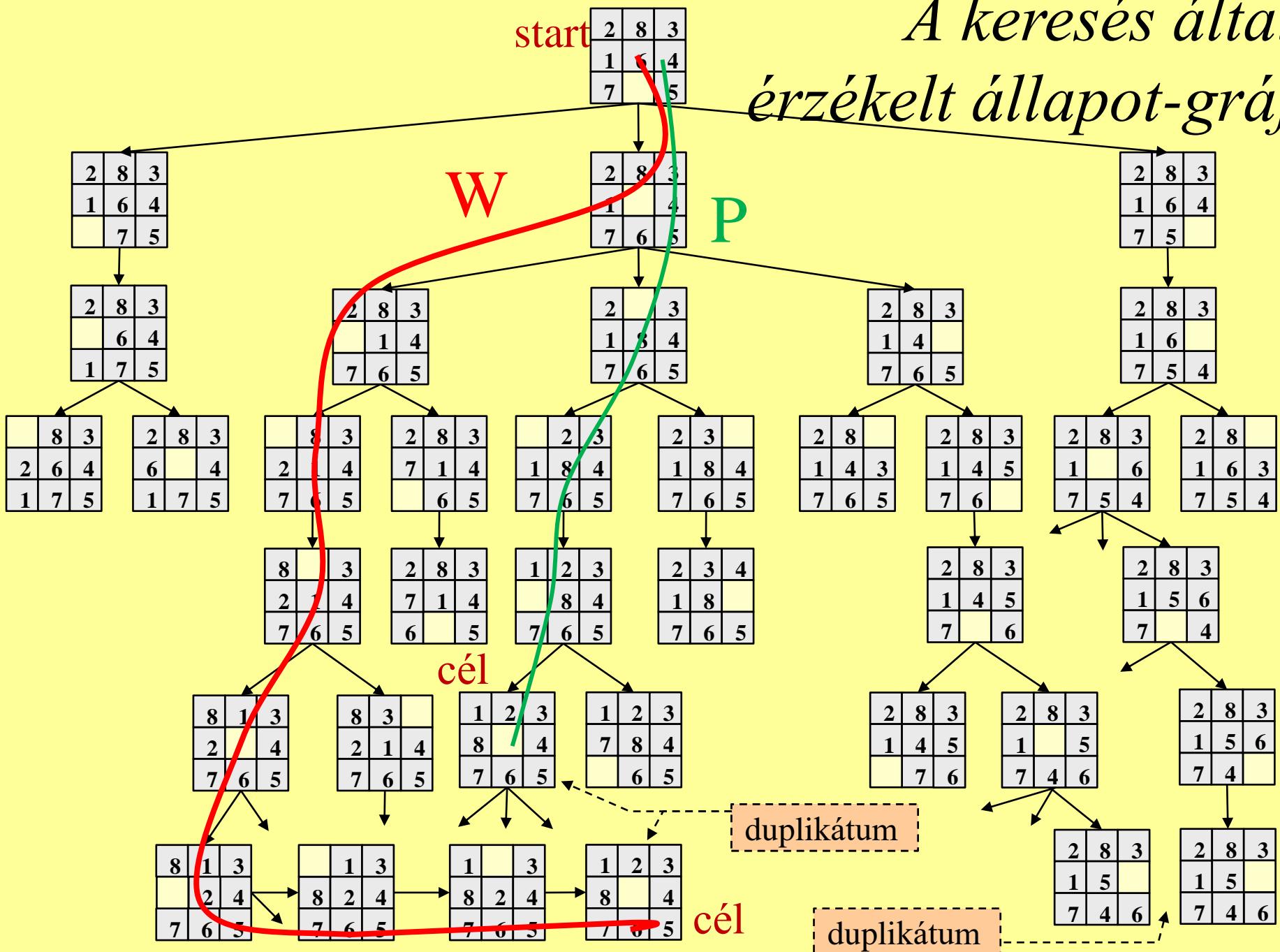
1	2	3
8		4
7	6	5

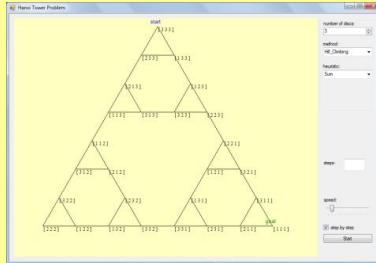
l:-, u:, r:0, d:3

Mesterséges intelligencia

A keresés által

érzékelte állapot-gráf

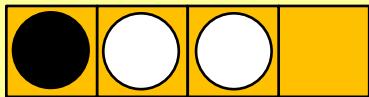




Heurisztikák a Hanoi tornyai problémára

- Darab: $C(this) = \sum_{\substack{i=1..n \\ this[i] \neq 1}} 1$
- Súlyozott darab: $WC(this) = \sum_{\substack{i=1..n \\ this[i] \neq 1}} i$
- Összeg: $S(this) = \sum_{i=1..n} this[i]$
- Súlyozott összeg: $WS(this) = \sum_{i=1..n} i \cdot this[i]$
- Módosított összeg: $EWS(this) = WS(this) - \begin{cases} 1 & if this[i-1] > this[i] \\ 2 & if this[i-1] = this[i+1] \wedge this[i] \neq this[i-1] \end{cases}$

Fekete-fehér kirakó



Egy $n+m+1$ hosszú sínen n fekete és m fehér lapka és egy üres hely van. Egy lapkát szomszédos üres helyre tolhatunk vagy a szomszéd felett üres helyre ugrathatunk. Kezdetben a feketék után jönnek a fehérek, majd az üres hely. Kerüljenek a fehérek a feketék elő!

Állapottér: $AT = rec(v : \{B, W, _\}^{n+m+1}, \text{üres} : [1.. n+m+1])$

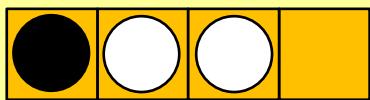
invariáns: n darab B , m darab W , 1 üres hely, üres az üres hely indexe

Műveletek: **TolBal**, **TolJobb**, **UgrikBal**, **UgrikJobb**: $AT \rightarrow AT$

TolBal : HA $this.\text{üres} \neq 1$ $(this : AT)$
AKKOR $this.v[this.\text{üres}-1] \leftrightarrow this.v[this.\text{üres}]$
 $this.\text{üres} := this.\text{üres}-1$

Kezdőállapot: $[B, \dots, B, W, \dots, W, _]$

Végállapot: $\forall i, j \in [1.. n+m+1], i < j : \neg(this.v[i]=B \wedge this.v[j]=W)$



Heurisztikák a Fekete-fehér kirakóra

- Inverziószám:

$I(this)$ = minimálisan hány csere kell ahhoz, hogy minden fehér minden feketét megelőzzön

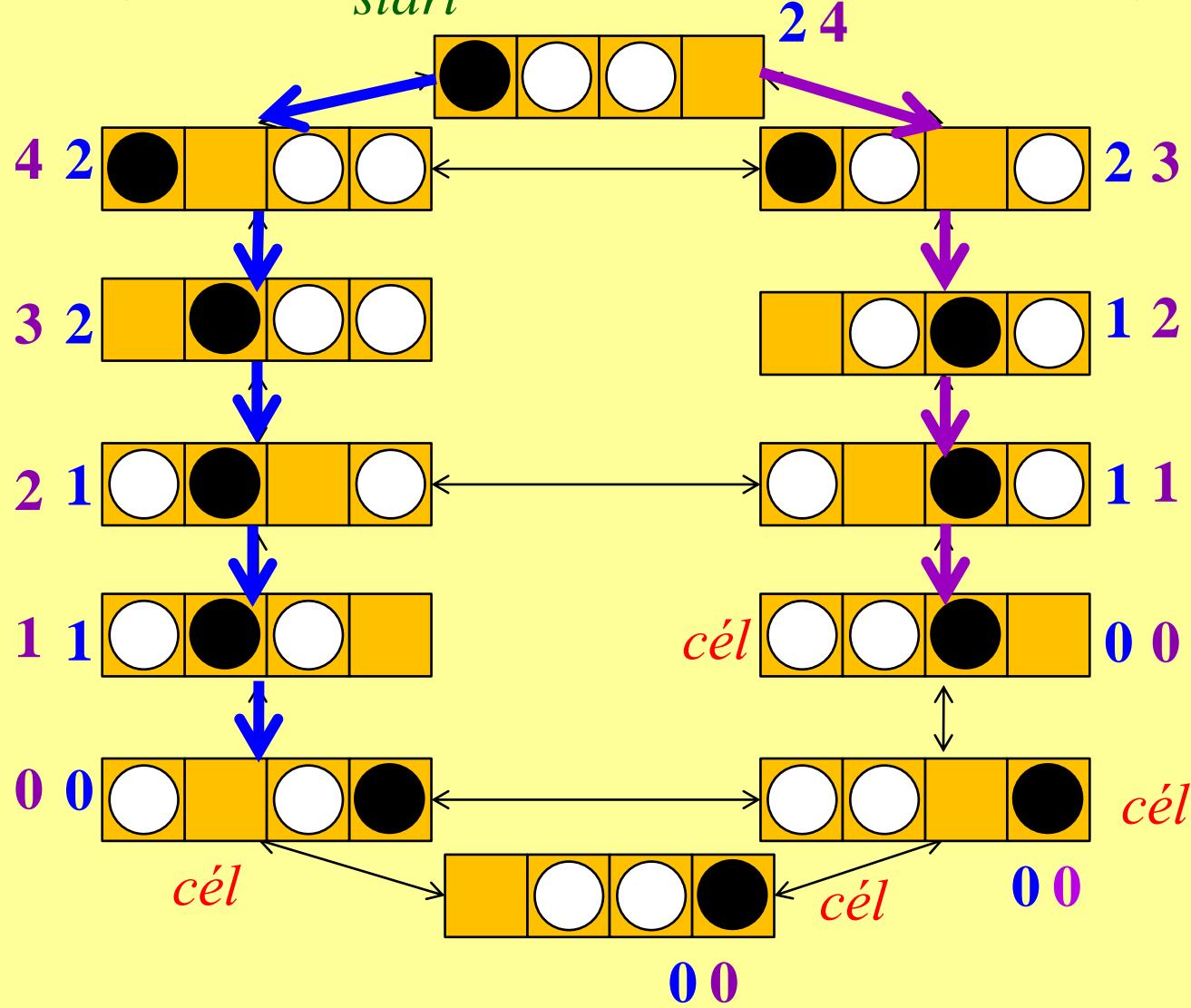
- Módosított inverziószám:

$M(this) = 2 \cdot I(this) -$
 $- (1, \text{ ha } this\text{-nek része } \square \square \square \text{ vagy } \square \square \square \square)$

Inv

start

$2*Inv - ?1$



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: célcsúcs 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

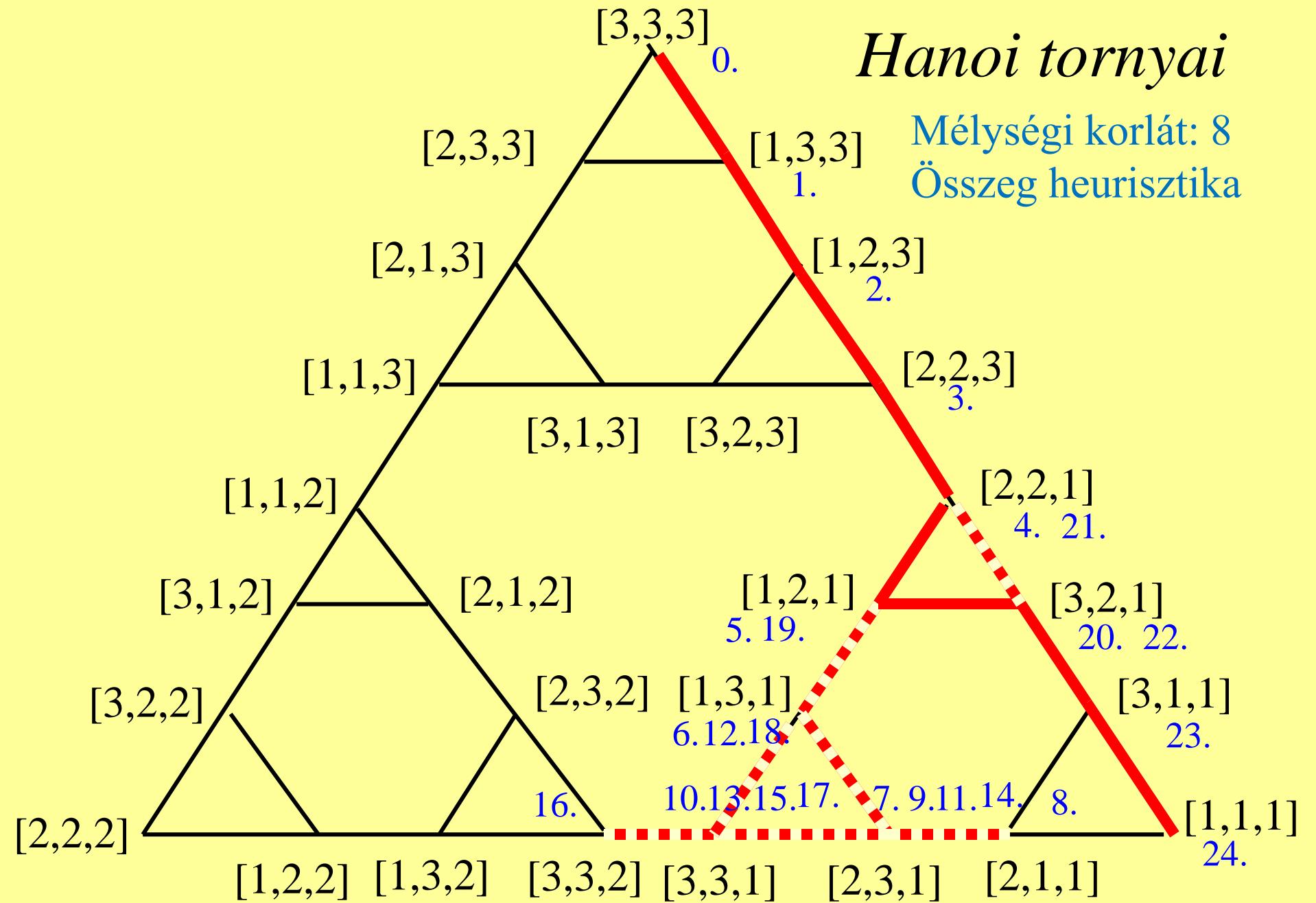
- A legvégső eset, amikor a visszalépést kell választani:
 - **zsákutca**: az aktuális csúcsból (azaz az aktuális út végpontjából) nem vezet tovább él
 - **zsákutca torkolat**: az aktuális csúcsból kivezető utak nem vezettek célba
 - **kör**: az aktuális csúcs szerepel már korábban is az aktuális úton
 - **méliségi korlát**: az aktuális út hossza elér egy előre megadott értéket

Alacsonyabb rendű vezérlési stratégiák

- Az általános vezérlési stratégia kiegészíthető:
 - **sorrendi szabállyal**: amely sorrendet ad egy csúcsból kivezető élek vizsgálatára
 - **vágó szabállyal**: kizárja egy csúcs azon kivezető éleit, amelyeket nem érdemes megvizsgálni
- Ezek a szabályok lehetnek
 - modellfüggő vezérlési stratégiák
 - heurisztikus vezérlési stratégiák

Hanoi tornyai

Mélységi korlát: 8
Összeg heurisztika



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: egy adott startcsúcsból kiinduló útból véges sok van.
- Rekurzív algoritmussal szokták implementálni
 - Indítás: $\text{megoldás} := \text{VL1}(\text{startcsúcs})$

ADAT := kezdeti érték

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

 SELECT SZ FROM alkalmazható szabályok

 ADAT := SZ(ADAT)

endloop

VLI

$A \sim$ élek halmaza
 $A^* \sim$ véges élsorozatok halmaza
 $N \sim$ csúcsok halmaza

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

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

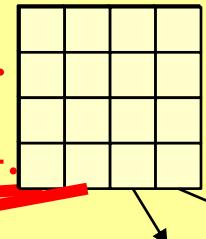
n-királynő probléma

2. állapottér modell

sorrendi szabály: balról-jobbra

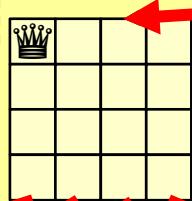
modellfüggő stratégia

Nyomkövetés kívülről nézve

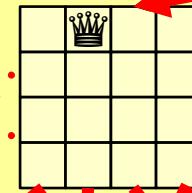


0.

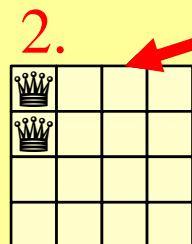
34.



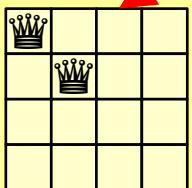
1. 3. 5.
15. 33.



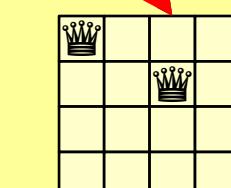
35.
37. 39.
41.



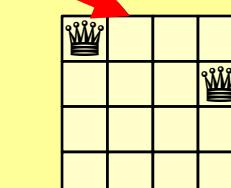
2.



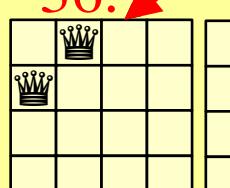
4.



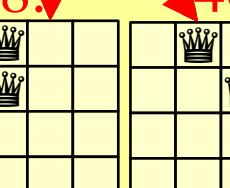
6. 8. 10. 12. 14.



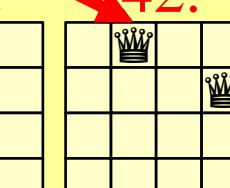
28. 30. 32.
16. 18.



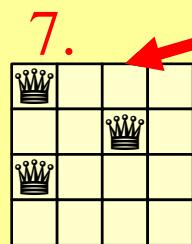
36.



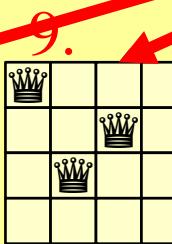
38.



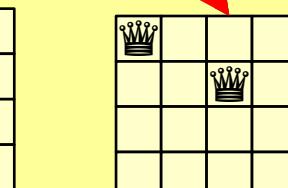
40.



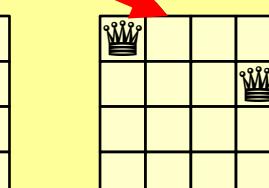
7.



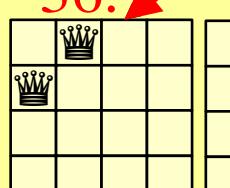
9.



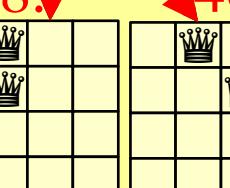
11.



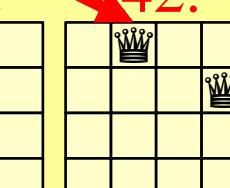
13.



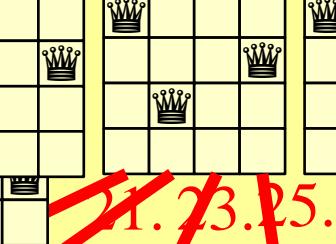
17.



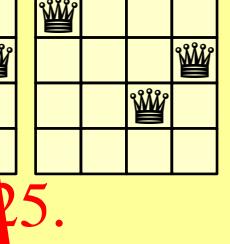
19.



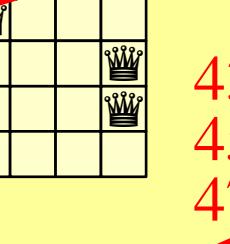
21.



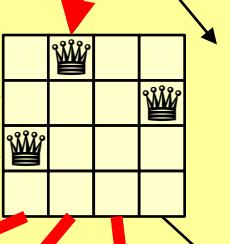
22.



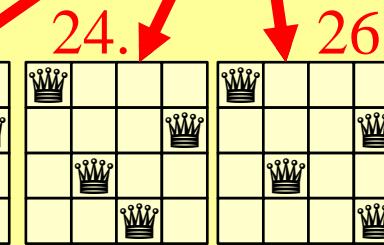
23.



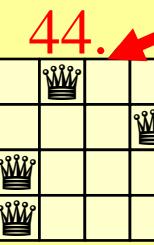
24.



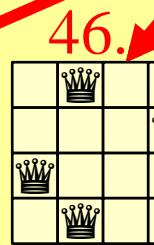
25.



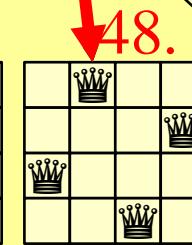
26.



44.



46.



48.

48 lépés

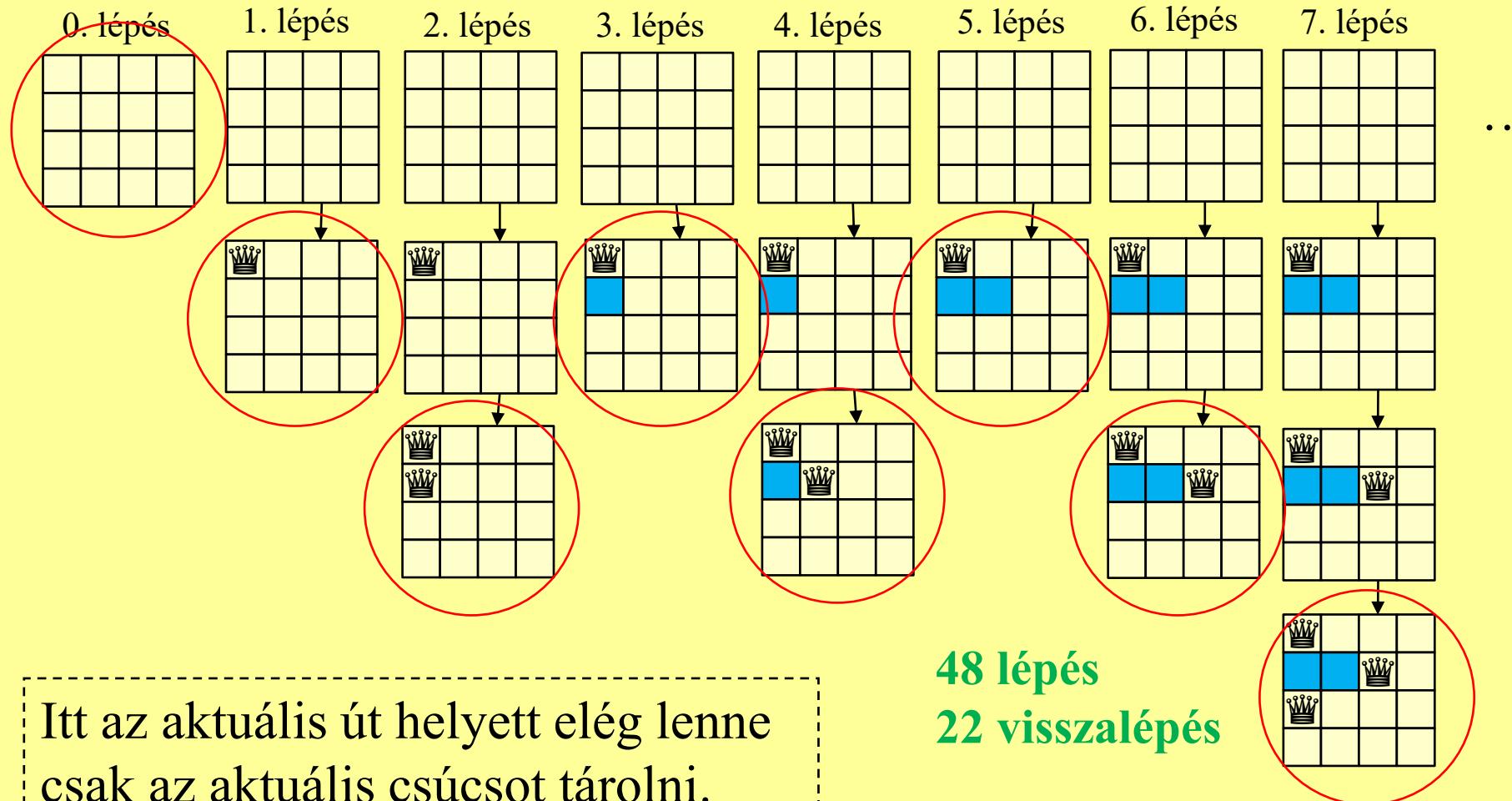
22 visszalépés

n-királynő probléma

2. állapottér modell

sorrendi szabály: balról-jobbra

Nyomkövetés belülről nézve



Sorrendi heurisztikák az n -királynő problémára

Az i -edik sor mezőit rangsoroljuk, 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:** egy adott királynő elhelyezés következtében a szabad státuszukat elvesztő mezők száma

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 + balról-jobbra:

4	👑	3	4
	4	4	👑
👑	4	4	3
4		👑	4

8 lépés

2 visszalépés

diagonális + páratlan-páros:

4	👑	3	4
3	4	4	👑
👑	4	4	3
4	3	👑	4

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. állapottér modell

sorrendi szabály: **balról-jobbra**

VLI

A k -dik lépés: a k -adik királynő elhelyezése után az üres sorok szabad mezőinek száma csökken

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

$Töröl(i,k)$: törli D_i azon mezőit, amelyeket a k -dik sor királynője üt.

k -dik lépésekben végzett törlések:

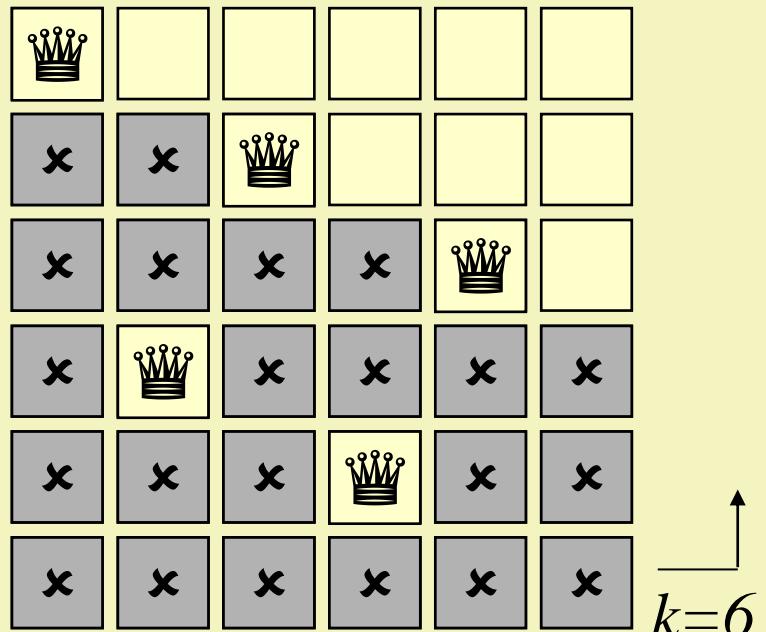
for $i=k+1 .. n$ **loop**

$Töröl(i,k)$

endloop

+

if $D_k = \emptyset$ **then** visszalép



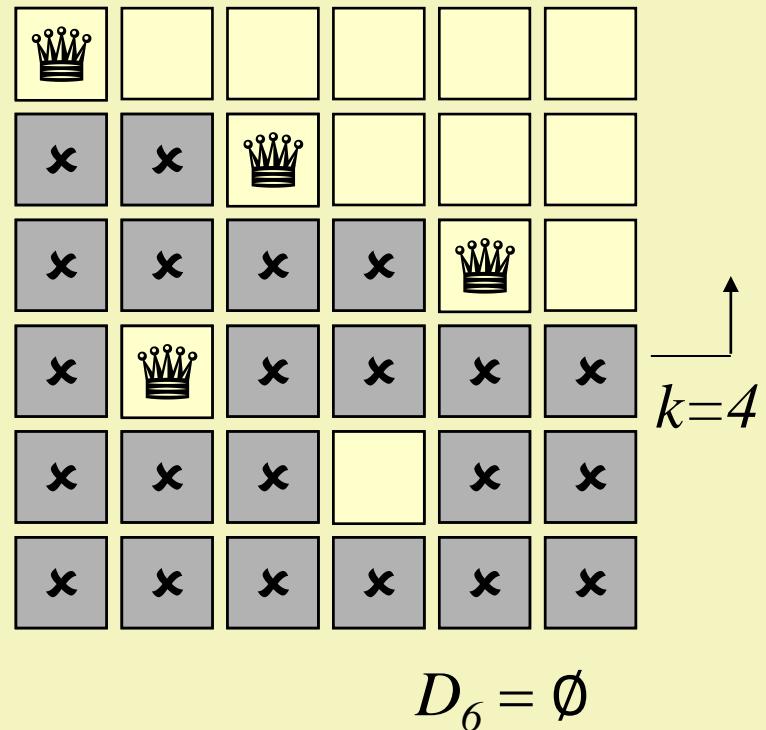
Forward Checking

FC algoritmus: VL1+vágó szabály

k -dik lépésben végzett törlések

+

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



$$D_6 = \emptyset$$

Partial Look Forward

PLF algoritmus: VL1+vágó szabály

k-dik lépésekben végzett törlések

—

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

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

$$Sz\acute{u}r(i, j) \quad (i < j)$$

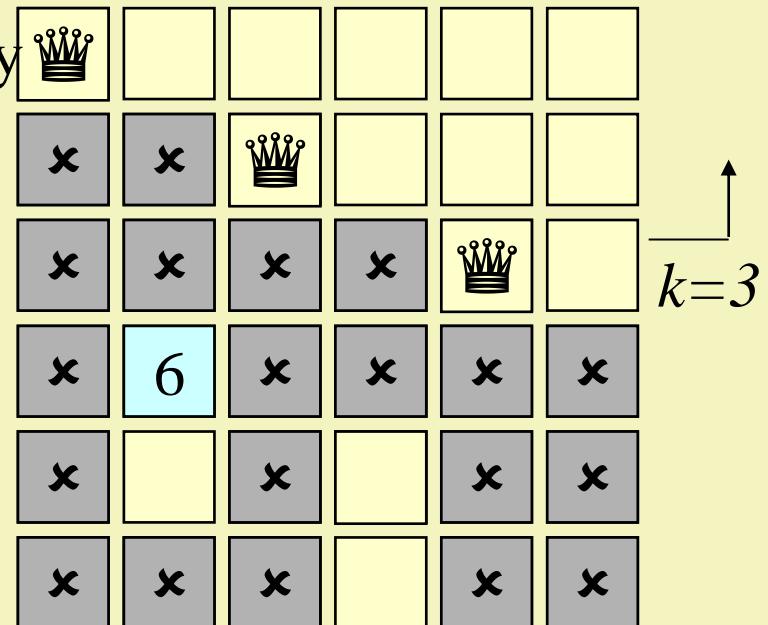
endloop

endloop

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

then *visszalép*

Szűr(i, j) : törli D_i azon haszontalan szabad mezőit, amelyek ütik a j -edik sor összes szabad mezőjét.



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

Look Forward

LF algoritmus: VLI + vágó szabály
 k -dik lépésben végzett törlések

+

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

$Szűr(i, j)$

endloop

endloop

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

then *visszalép*

crown						
x	x	crown				
x	x	x	x			
x		x	x	x		
x		x	x	x	3	
x	4	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 újabb modellje

- A vágó szabályok bemutatása közben rátaláltunk az n -királynő probléma egy új modelljére:
 - Tekintsük a D_1, \dots, D_n halmazokat, ahol D_i az i -dik sor szabad mezőinek oszlopszámai, kezdetben $D_i = \{1 \dots n\}$.
 - 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 oszlopszámai), 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 megoldást visszalépéses kereséssel adjuk meg, amely olyan vágási szabályokat használ, amelyek folyamatosan törlik a D_i -k haszontalan elemeit, és ha valamelyik D_i kiüresedik, akkor visszalépéstre kényszerítik az algoritmust.

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 $C_{ij} \subseteq D_i \times D_j$ ún. bináris korlátot.
- ❑ További 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. Gráf-színezési 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 FC , PLF , LF vágó szabályai a korlátkeilégítési modell bináris korlátjaival fogalmazhatók meg anélkül, hogy a korlátok jelentését ismernünk kellene:

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

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

- Ezek a vágó szabályok tehát nem heurisztikák, hanem modellfüggő vezérlési stratégiák, hiszen nem a feladathoz, hanem a **modellezési módszerhez** kapcsolhatók.
- Modellfüggő sorrendi szabályok is konstruálhatók:
 - Mindig a legkisebb tartományú még kitöltetlen komponensnek válasszunk előbb értéket.
 - Adott korlát által vizsgált 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 mindig 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 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**
 6. $\text{megoldás} := \text{VL2}(fűz(út, új))$
 7. **if** $\text{megoldás} \neq \text{hiba}$ **then**
 8. **return**($fűz((akt, új), \text{megoldás})$) **endif**
 9. **endloop**
 10. **return**(hiba)
- end**

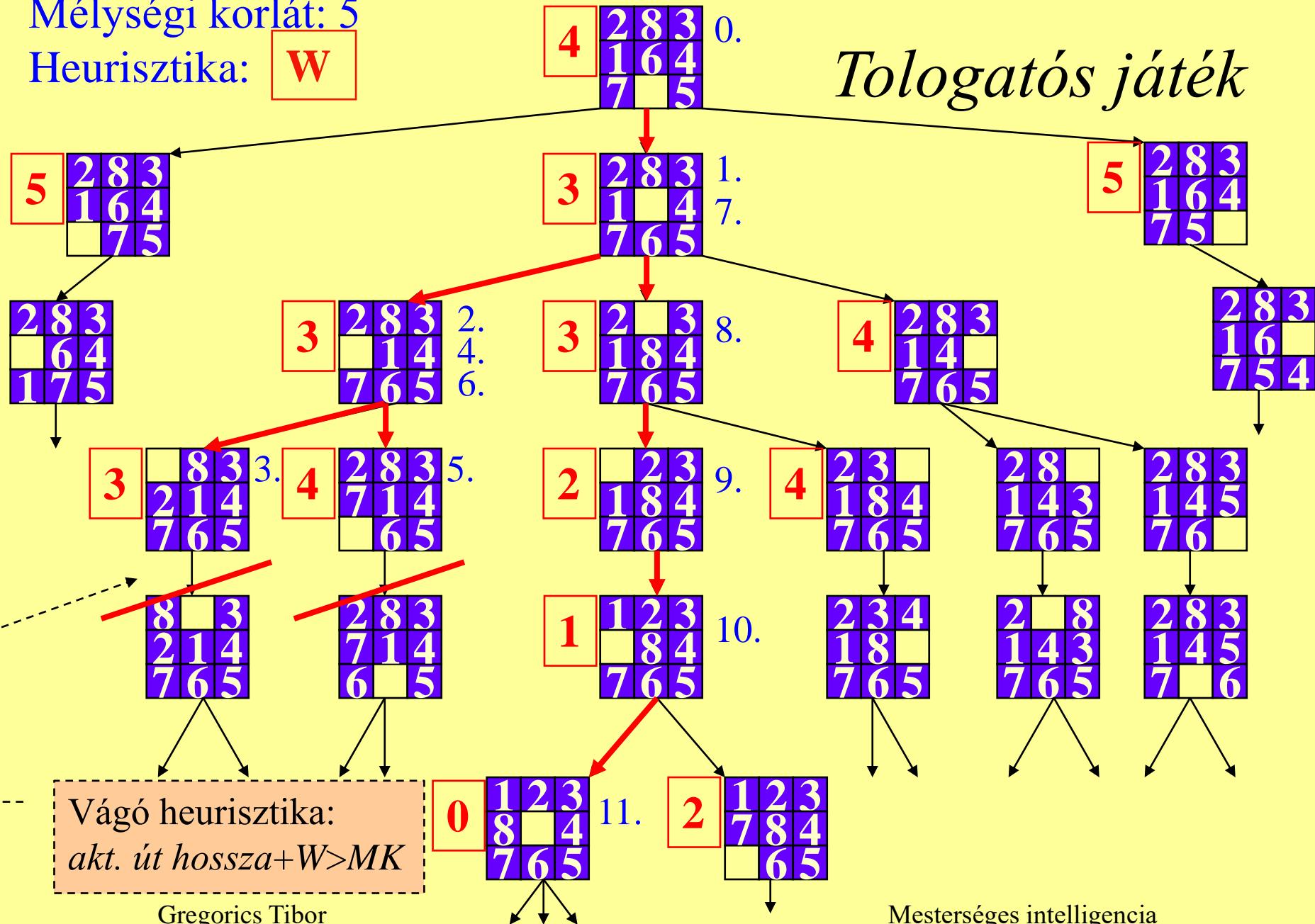
Mélységi korlát szerepe

- A mélységi korlát ellenőrzése önmagában is biztosítja a terminálást körfolyosás nélkül.
 - Ilyenkor nem kell a rekurzív hívásnál a teljes aktuális utat átadni : elég az aktuális csúcsot, annak szülőjét (ez kettő hosszú körok kiszűréséhez kell), és az aktuális út hosszát.
 - Hatékonyaság nő: nem kell utakat tárolni (csökkent a memória igény), nem végzünk körfolyosás (csökken egy lépés futási ideje), viszont ha a mélységi korlátnál rövidebb körok is vannak a reprezentációs gráfban, akkor a lépések száma nő.
- A VL2 a mélységi korlátnál hosszabb megoldási utat nem találja meg. (Ha nincs a korlátnál rövidebb megoldás, akkor a keresés sikertelenül terminál.)

Mélységi korlát: 5

Heurisztika: W

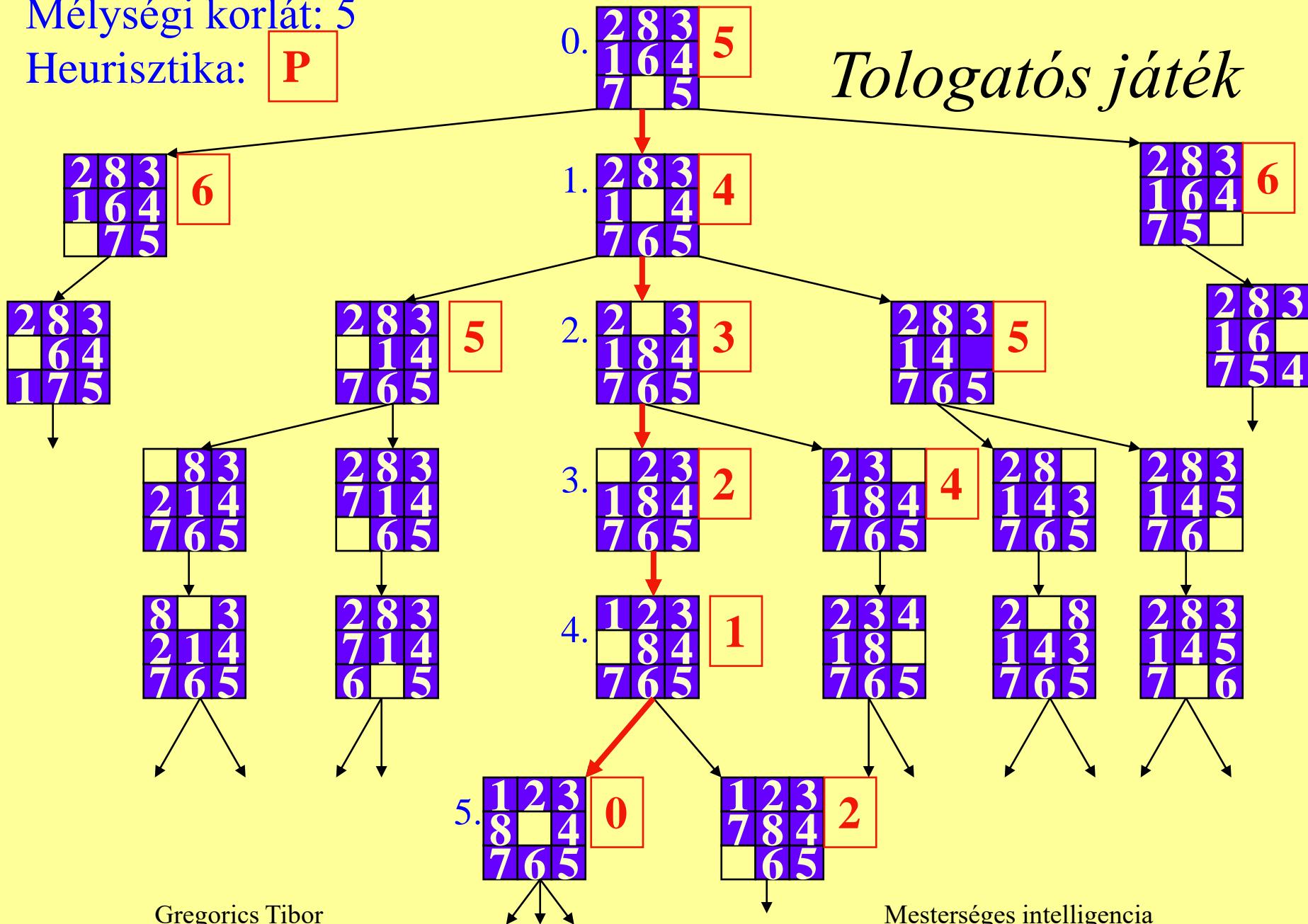
Tologatós játék



Mélységi korlát: 5

Heurisztika: P

Tologatós játék



Értékelés

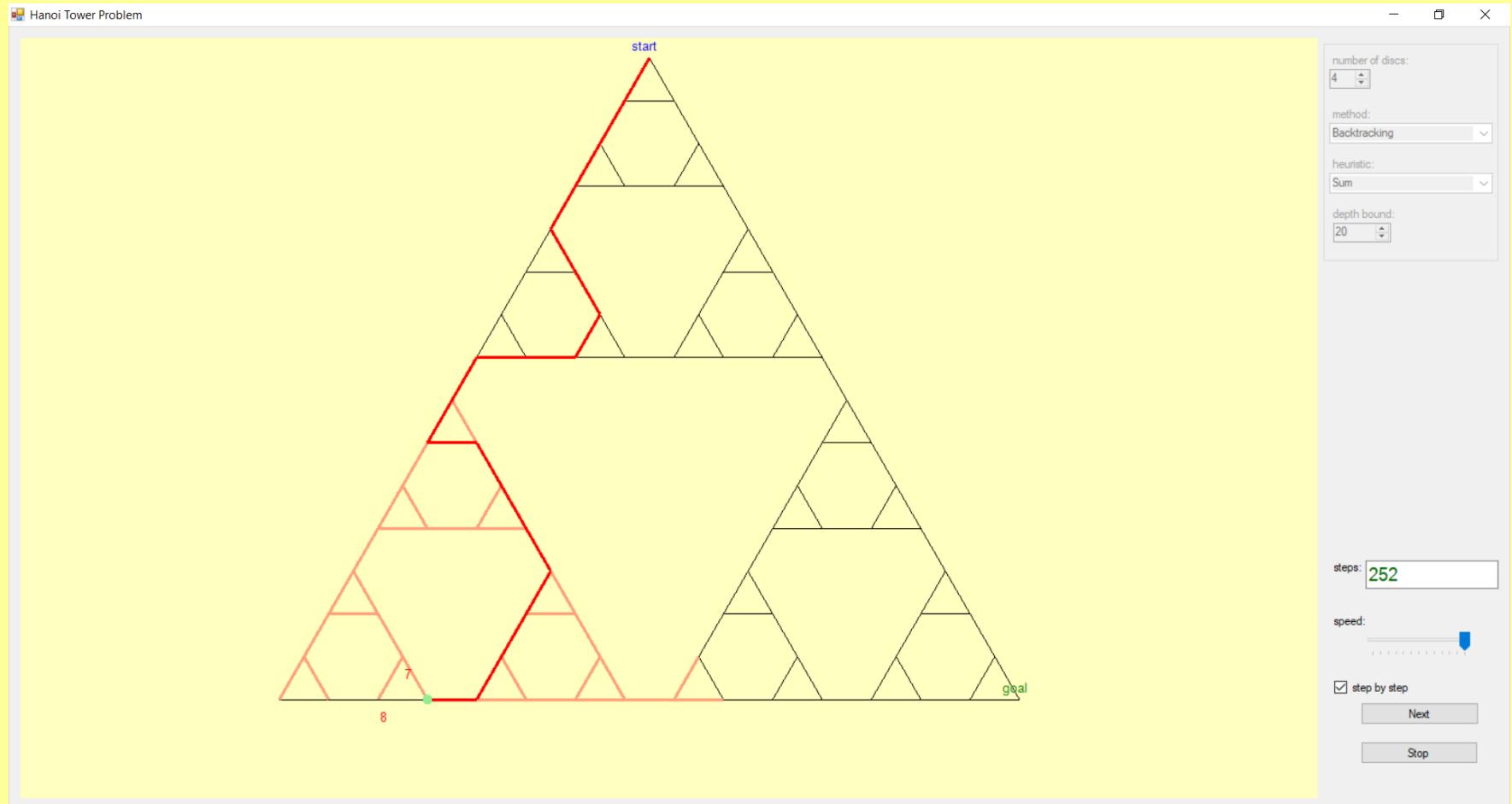
□ ELŐNYÖK

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

□ HÁTRÁNYOK

- nem ad optimális megoldást
(iterációba szervezhető)
- kezdeti rossz döntés csak sok
visszalépéssel korrigálható
(visszaugrásos keresés)
- egy zsákutca részt többször is
bejárhat a keresés

Hanoi Tower Problem



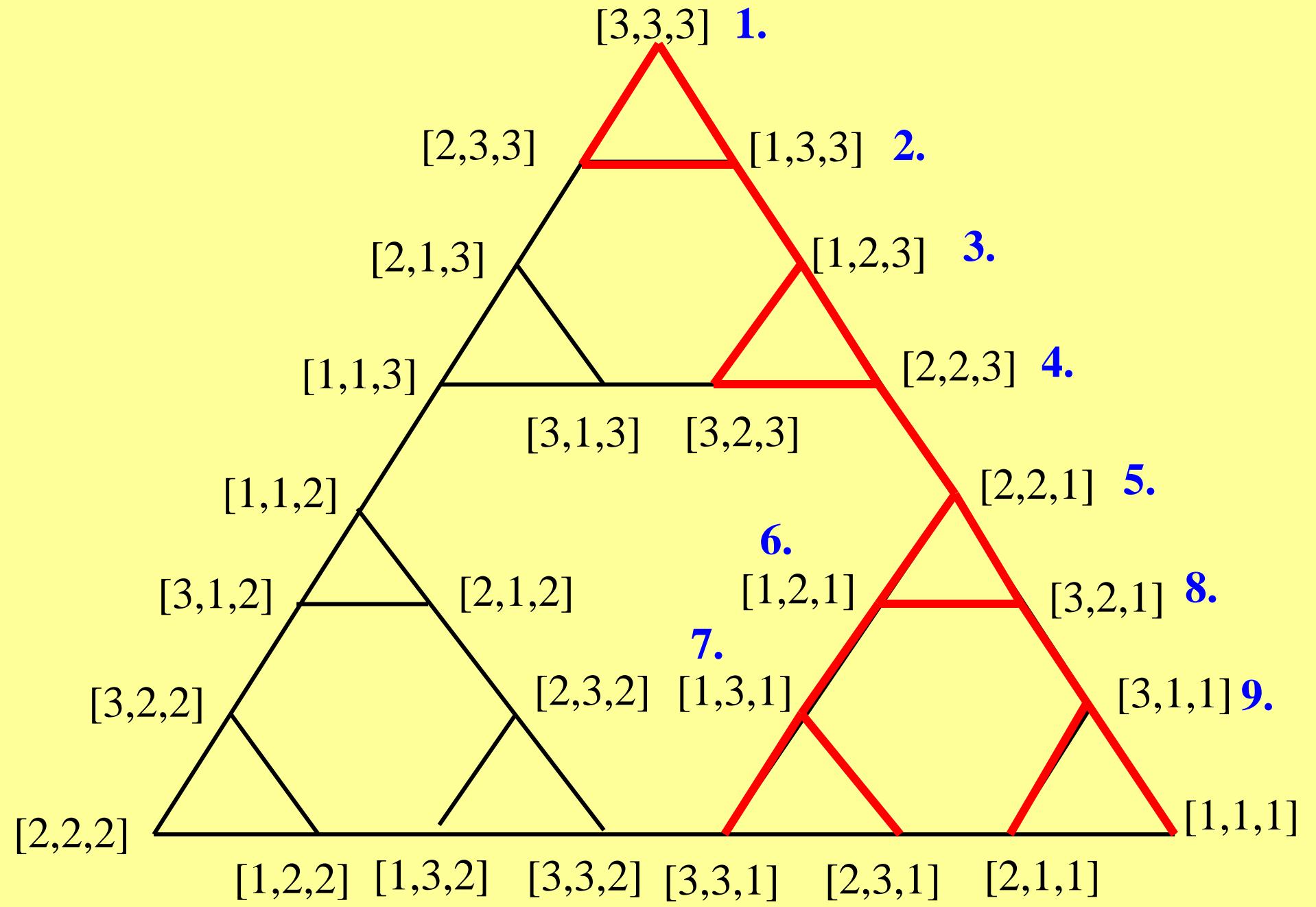
3. Gráfkeresés

- ❑ 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 célcímsúcsot terjeszt ki vagy nincs nyílt csúcs.
 - **keresési szabálya**: egy nyílt csúcs kiterjesztése (összes gyermekének hozzájuk vezető élekkel való előállítása)
 - **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



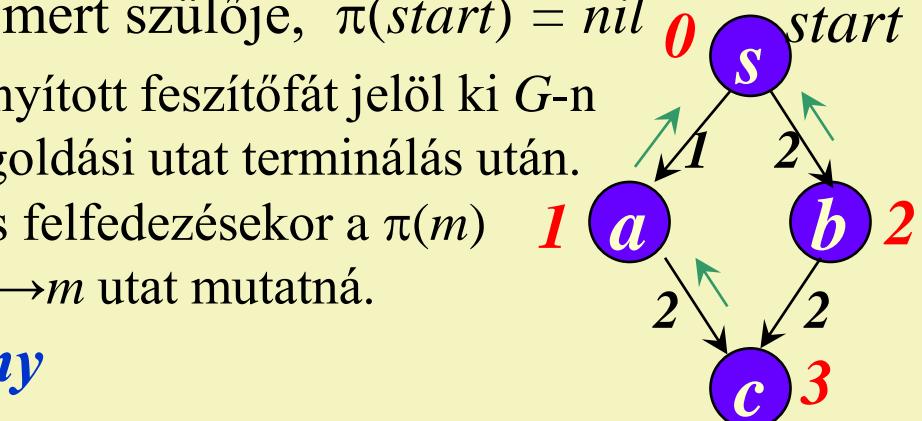
Gráfkeresés függvényei

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

- $\pi(m) =$ az m csúcs egy ismert szülője, $\pi(start) = nil$
 - π 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 π által mutatott $start \rightarrow m$ út költségét adná.



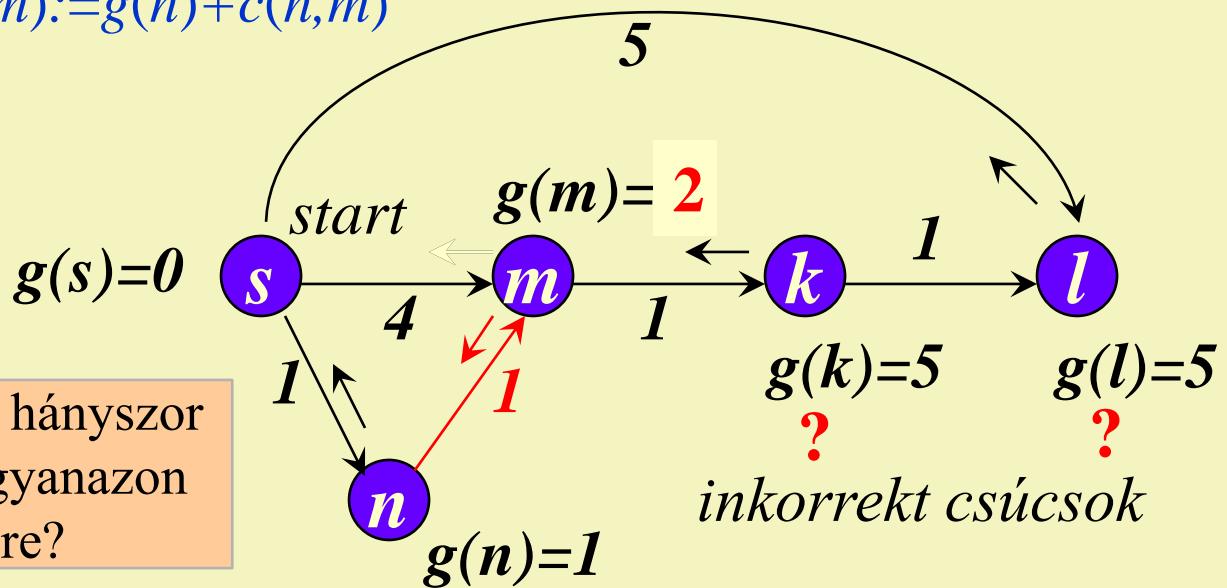
A G korrekt, ha minden csúcsa korrekt. Az $m \in G$ csúcs **korrekt**, ha $g(m)$ és $\pi(m)$ **konzisztenz**: $g(m) = c^\pi(start, m)$, és $\pi(m)$ G -ben **optimális**: $c^\pi(start, m) = \min_{\alpha \in \{start \rightarrow m\} \cap G} c^\alpha(start, m)$

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. $n := \arg \min_f(OPEN)$
5. **if** $cél(n)$ **then return** megoldás
6. $OPEN := OPEN - \{n\}$
7. **for** $\forall m \in \Gamma(n) - \pi(n)$ **loop**
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 δ -gráfban a működése során egy csúcsot legfeljebb véges sokszor terjeszt ki.
(ebből következik például, hogy körökre nem érzékeny)
- A GK véges δ -gráfban mindenkor terminál.
- Ha egy véges δ -gráfban létezik megoldás, akkor a GK megoldás megtalálásával terminál.

3.2. Nevezetes gráfkereső algoritmusok

- Válasszunk f kiértékelő függvényt.

Nem-informált

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

- A másodlagos vezérlési stratégiák, az úgynevezett tie-breaking rule-ok (egyenlőséget feloldó szabályok) a nem-informált gráfkeresésekben is tartalmazhatnak heurisztikát.

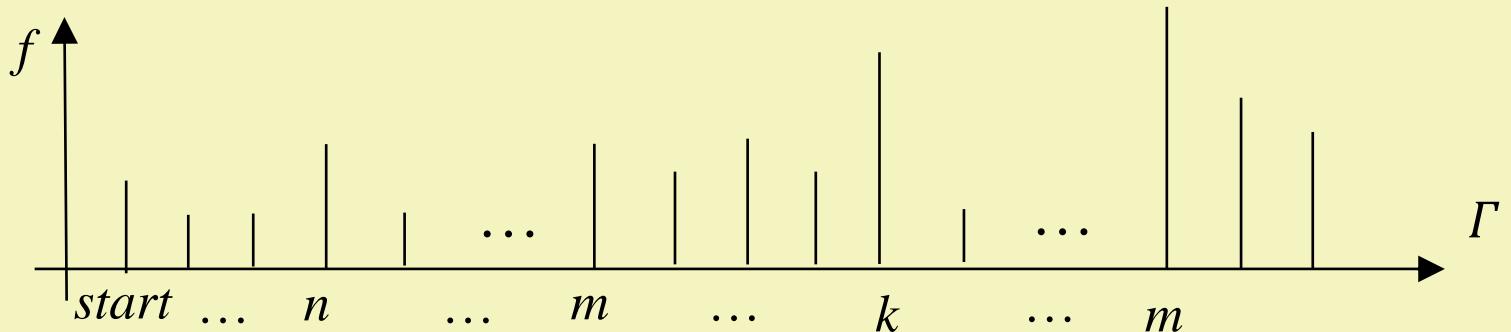
Heurisztikus

- előre tekintő (mohó, best-first)
- A , A^* , A^c
- B , B' , A^{**}

Csökkenő kiértékelő függvény

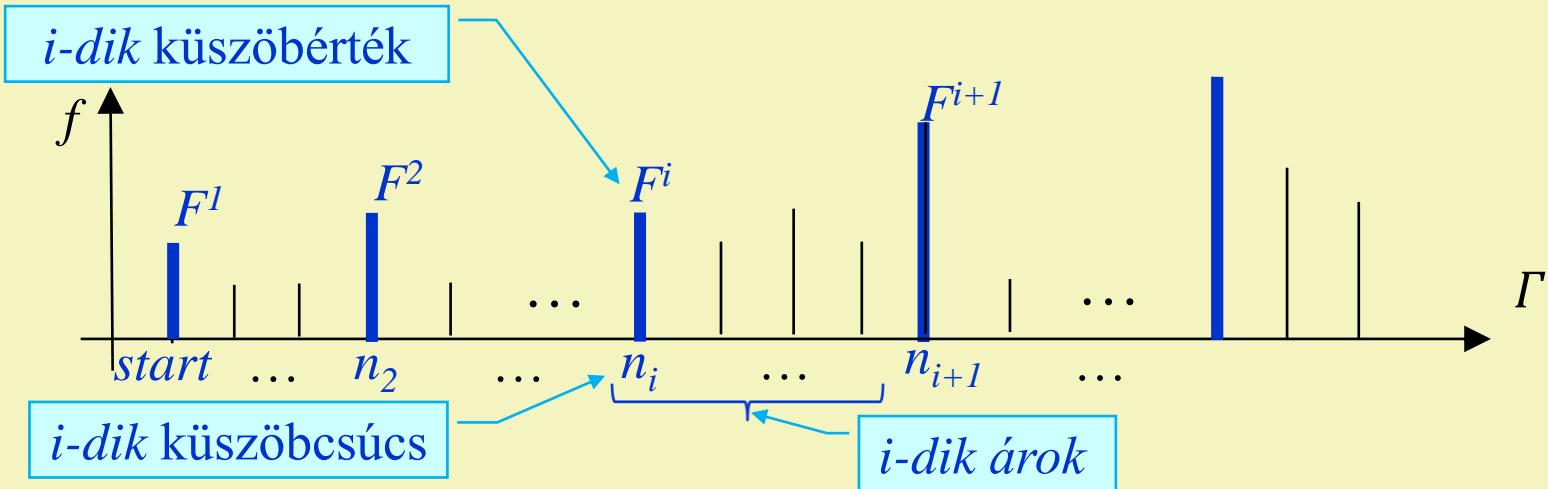
- Egy GK kiértékelő függvénye **csökkenő**, amennyiben a egy csúcsra adott értéke az algoritmus működése során nem növekszik, viszont mindenkor mindenkor csökken, valahányszor a csúcshoz a korábbinál olcsóbb utat találunk.
 - Például a g költségfüggvény önmagában ilyen.
- Csökkenő kiértékelő függvény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

Gráfkeresés működési grafikonja



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

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 valahányszor küszöbcsúcsot terjeszt ki a GK a G kereső gráf korrekt lesz.

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

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 nem ismert, 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.

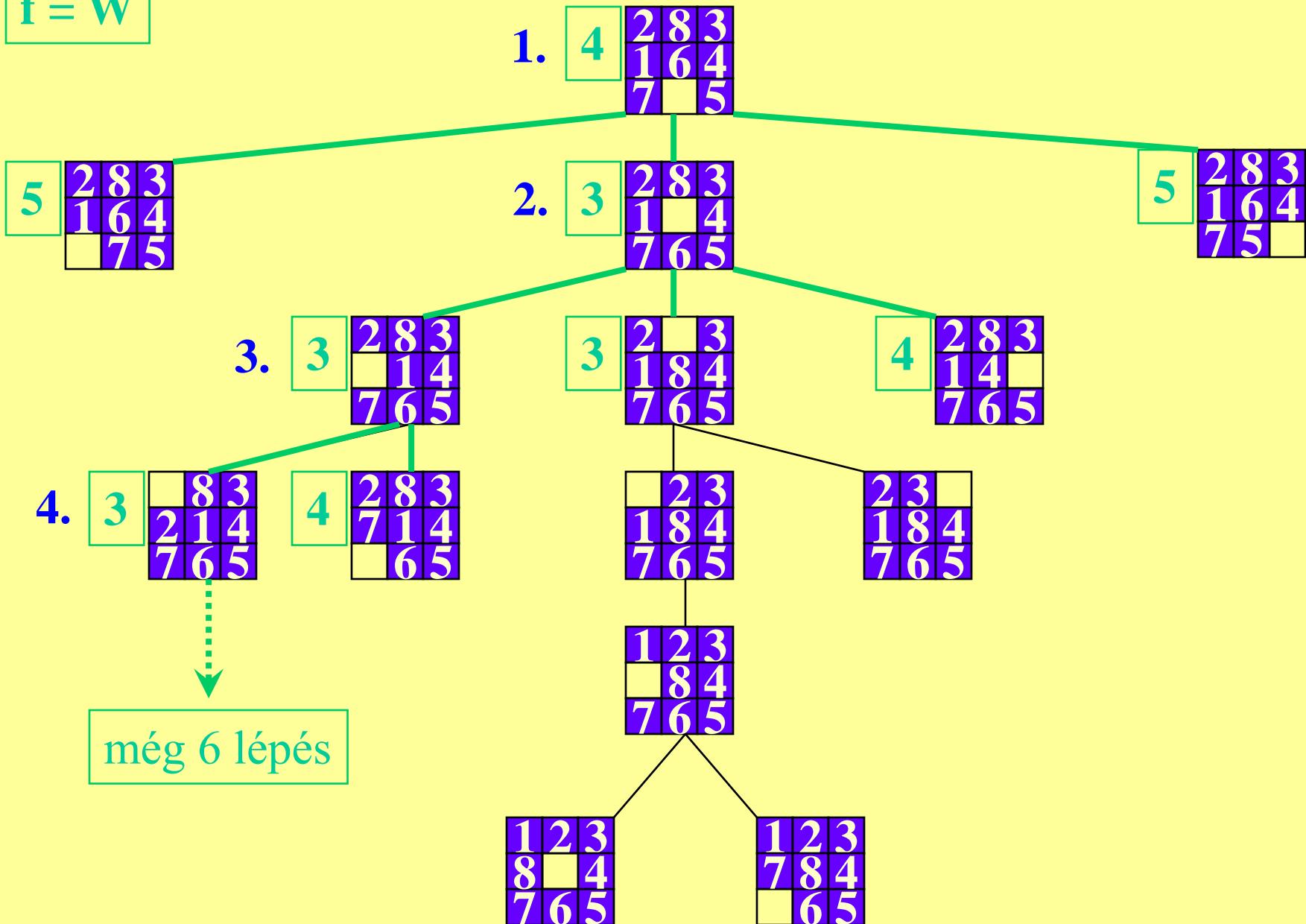
Nevezetes heurisztikus algoritmusok

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

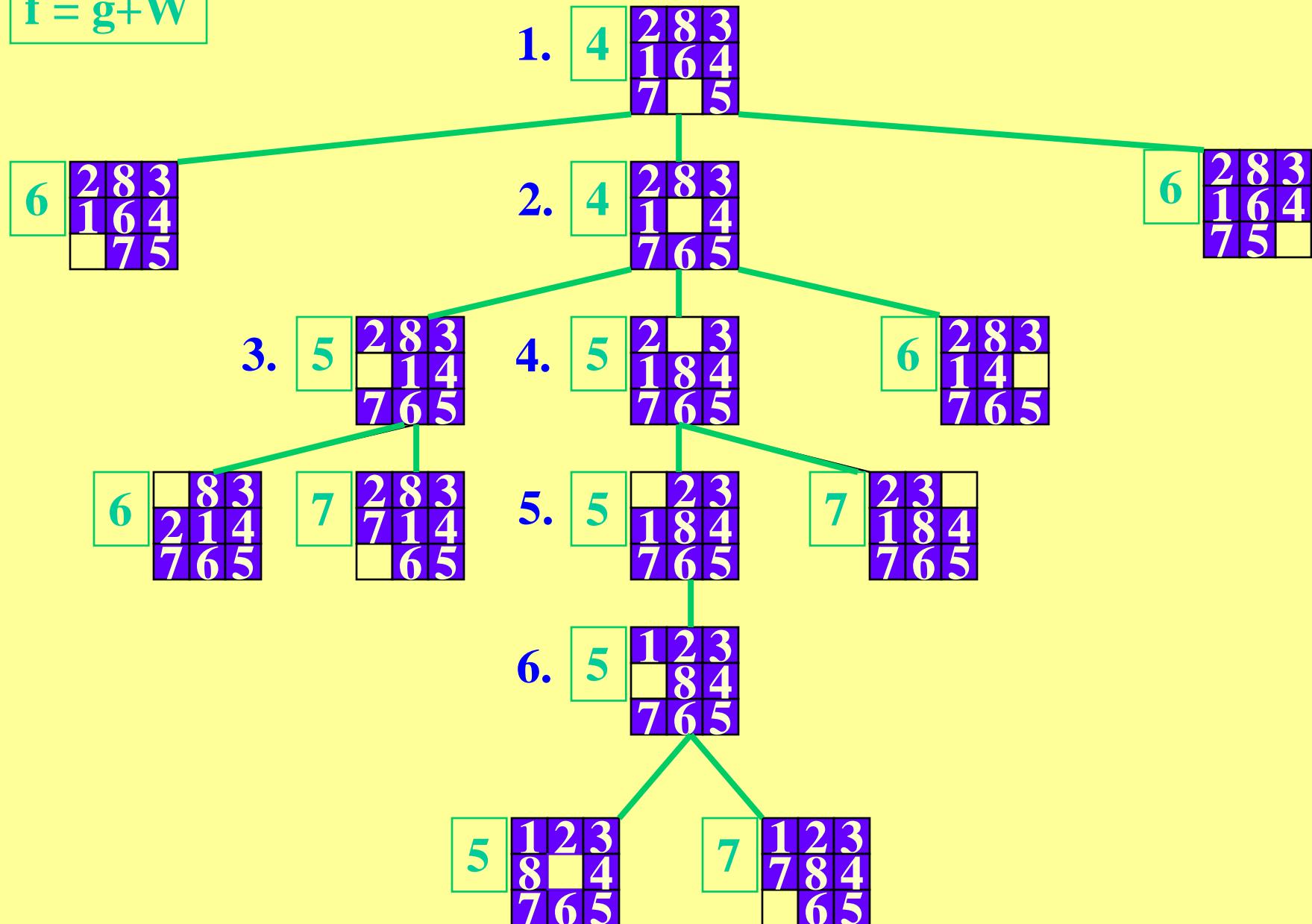


egyenletes gráfkeresés:
 $f = g + 0$

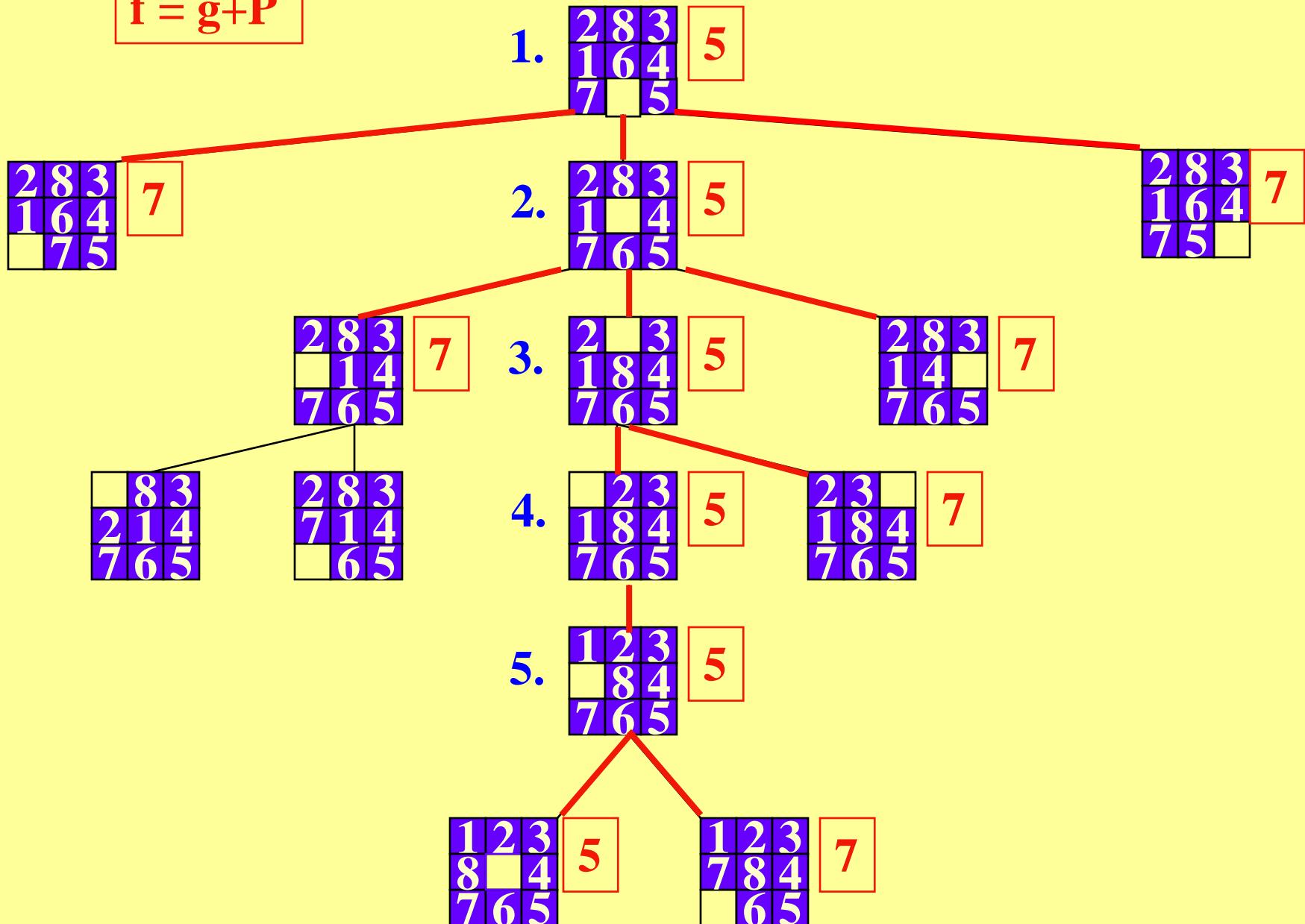
$f = W$



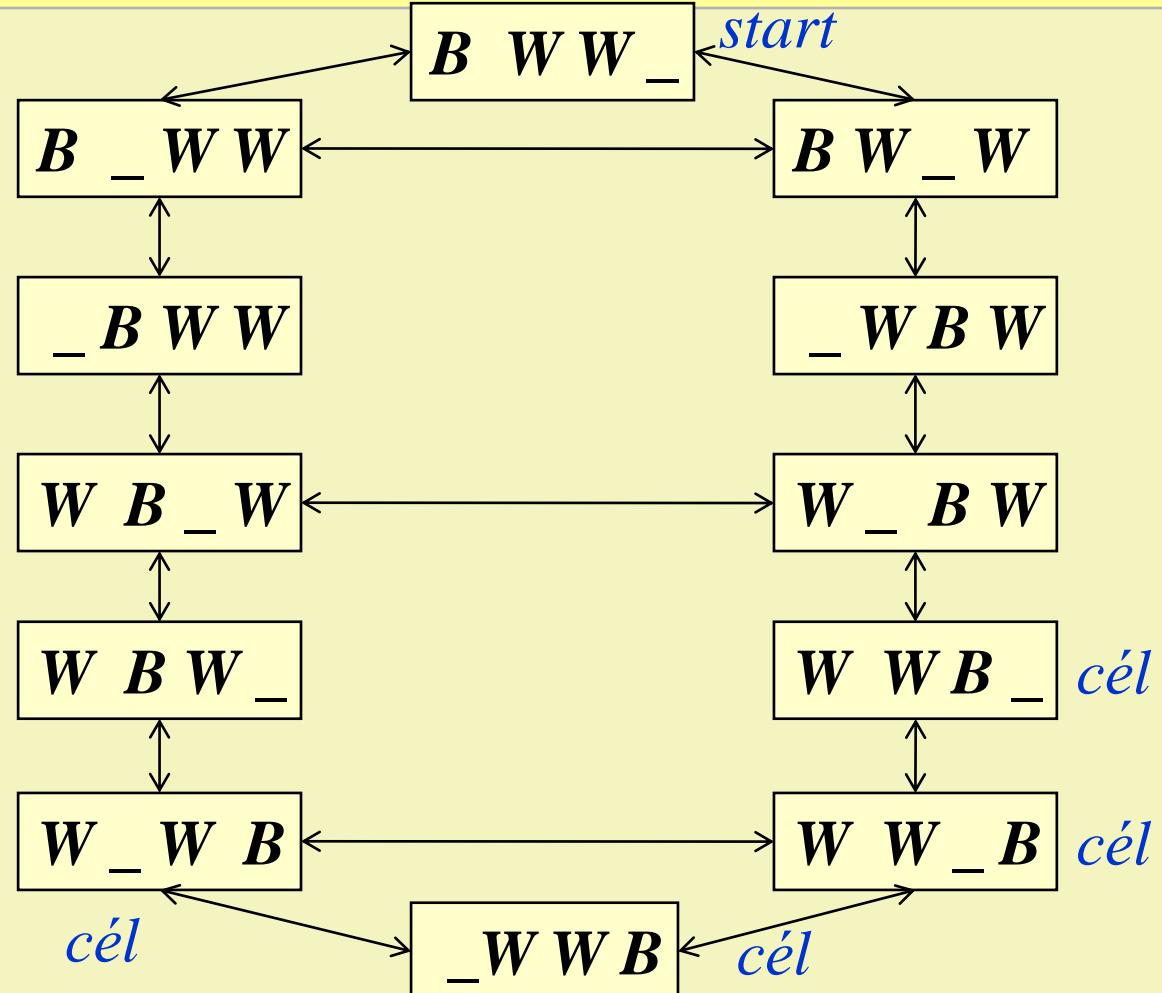
$$\mathbf{f} = \mathbf{g} + \mathbf{W}$$



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

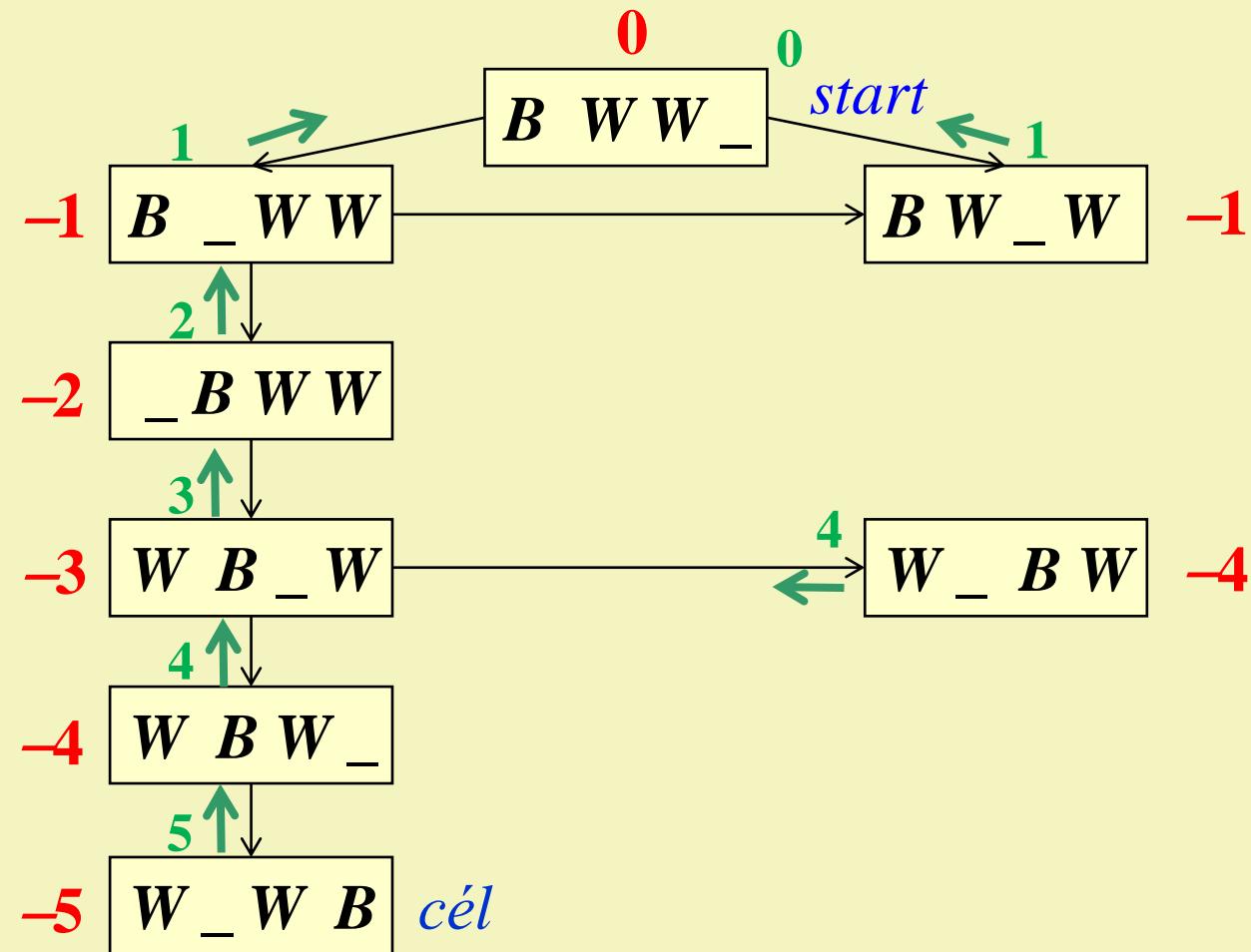


Fekete-fehér kirakó állapot gráfja



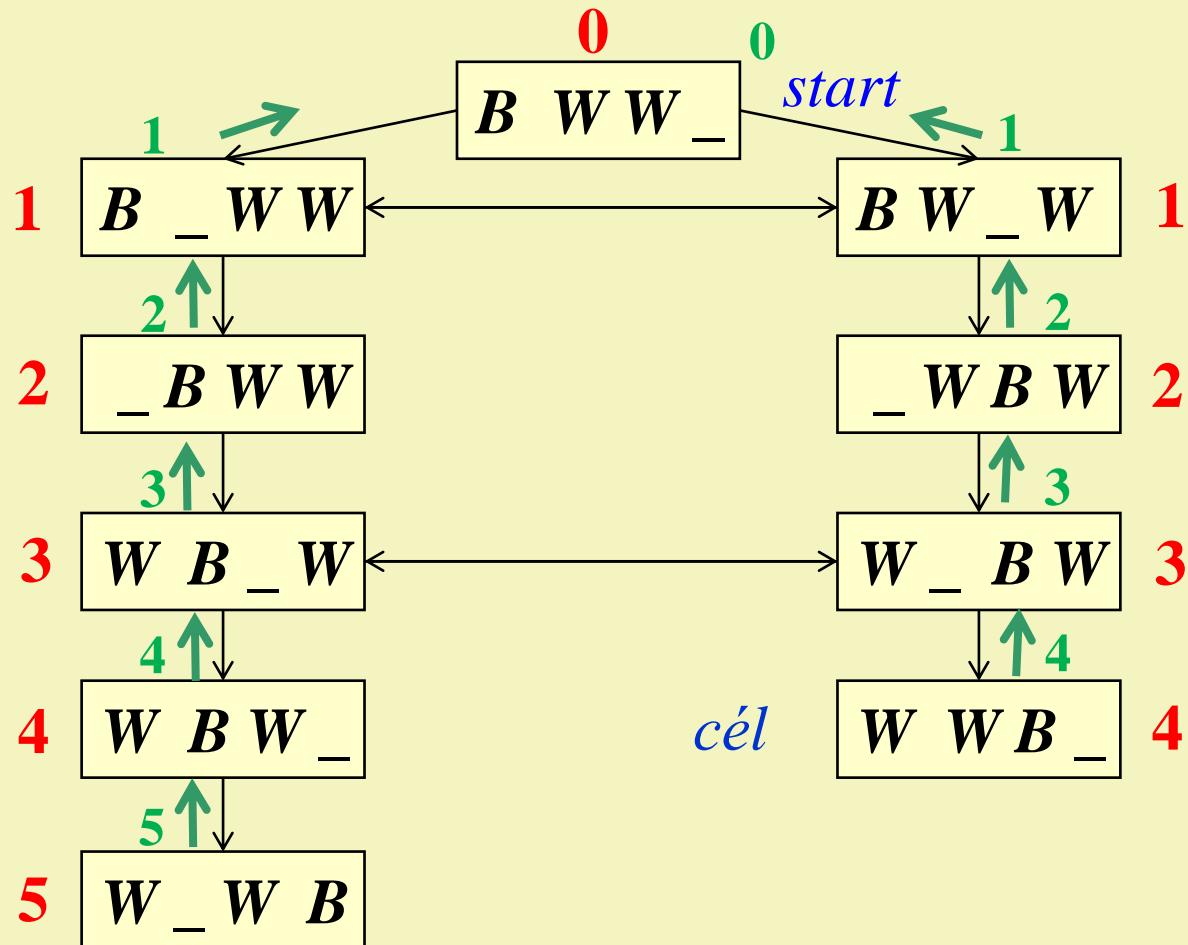
Mélységi gráfkeresés

$$f = -g$$



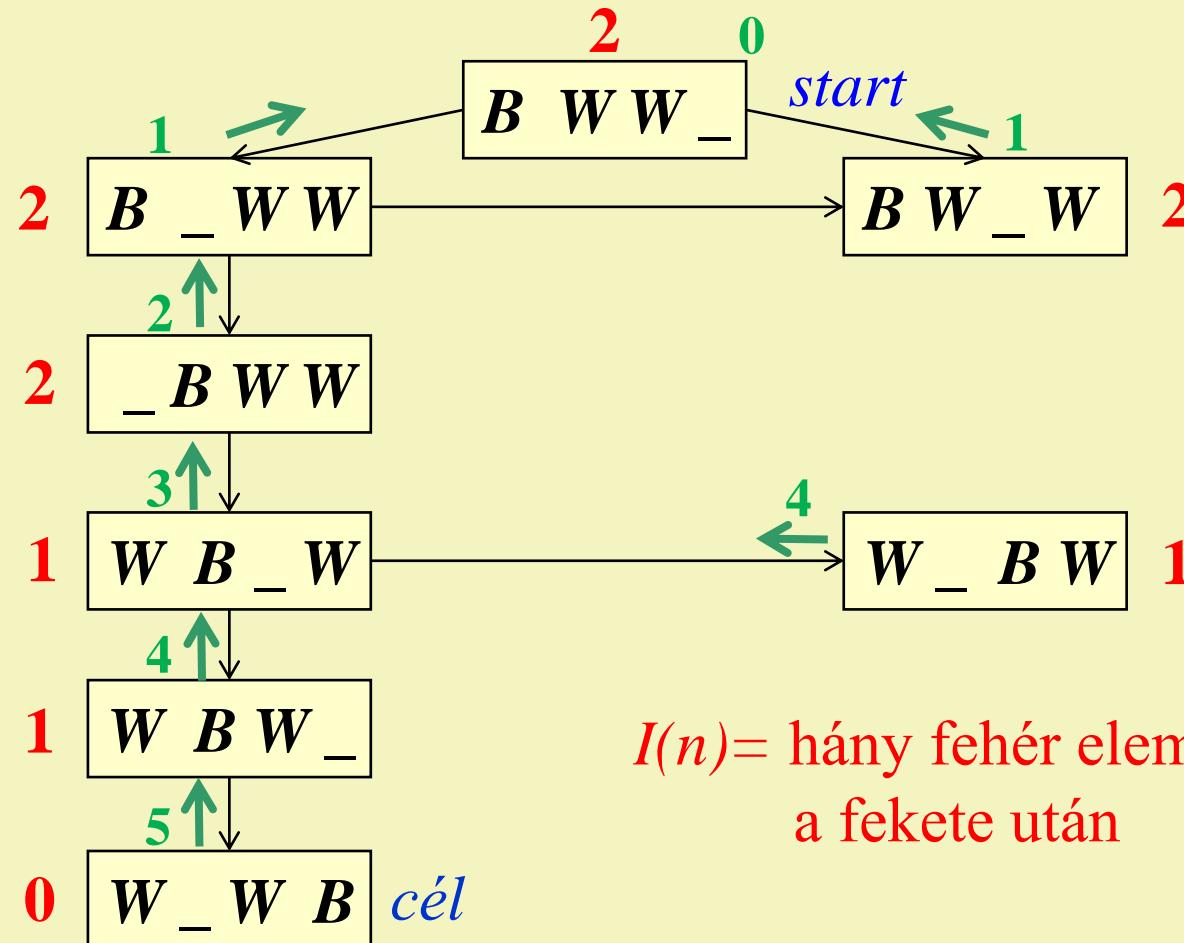
Szélességi gráfkeresés

$$f = g$$



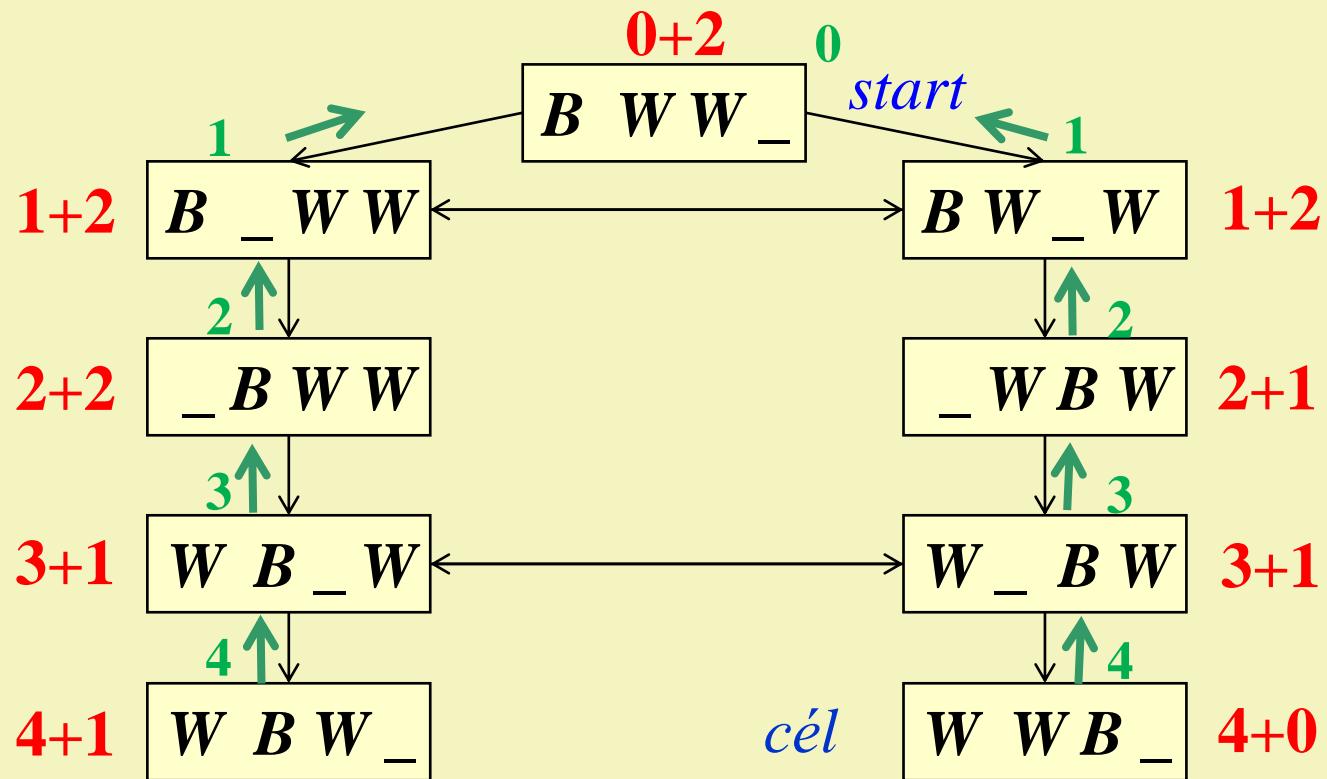
Előre tekintő gráfkeresés

$f = I$



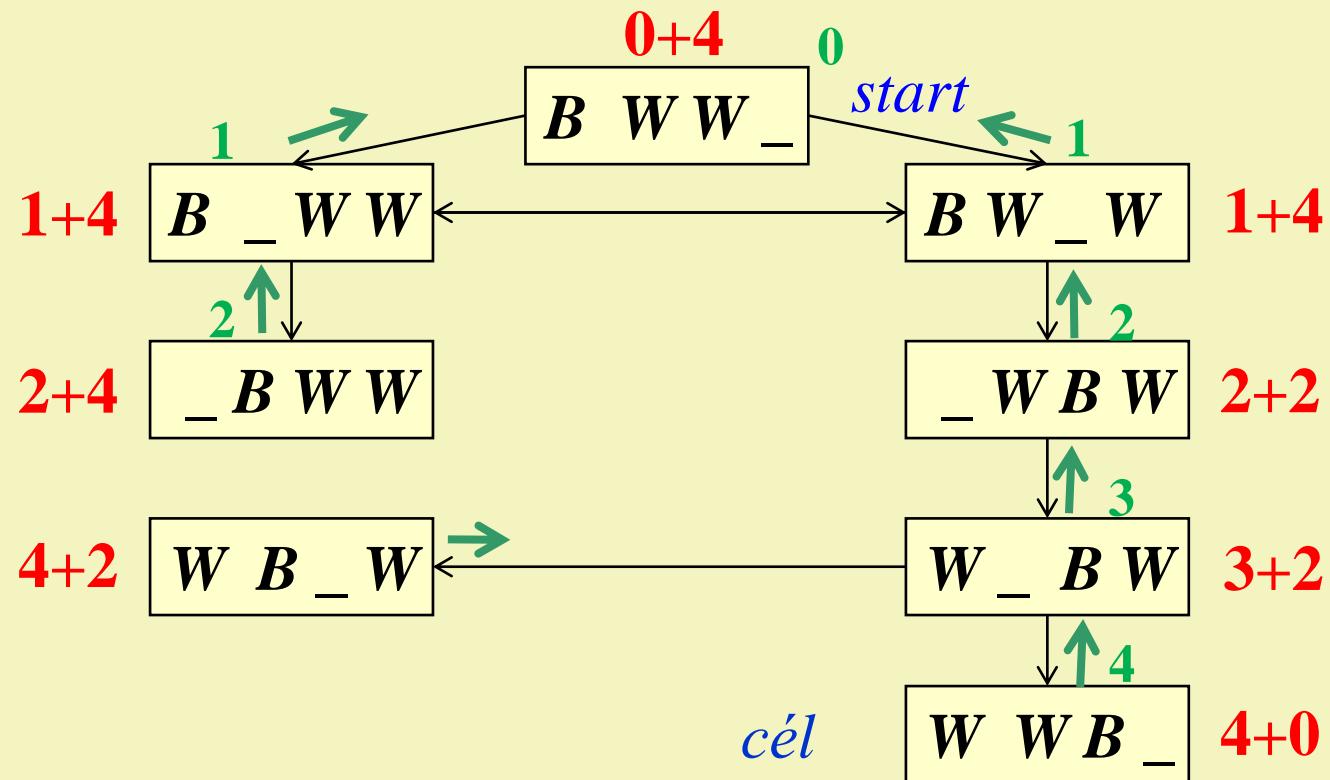
A algoritmus

$$f = g + I$$



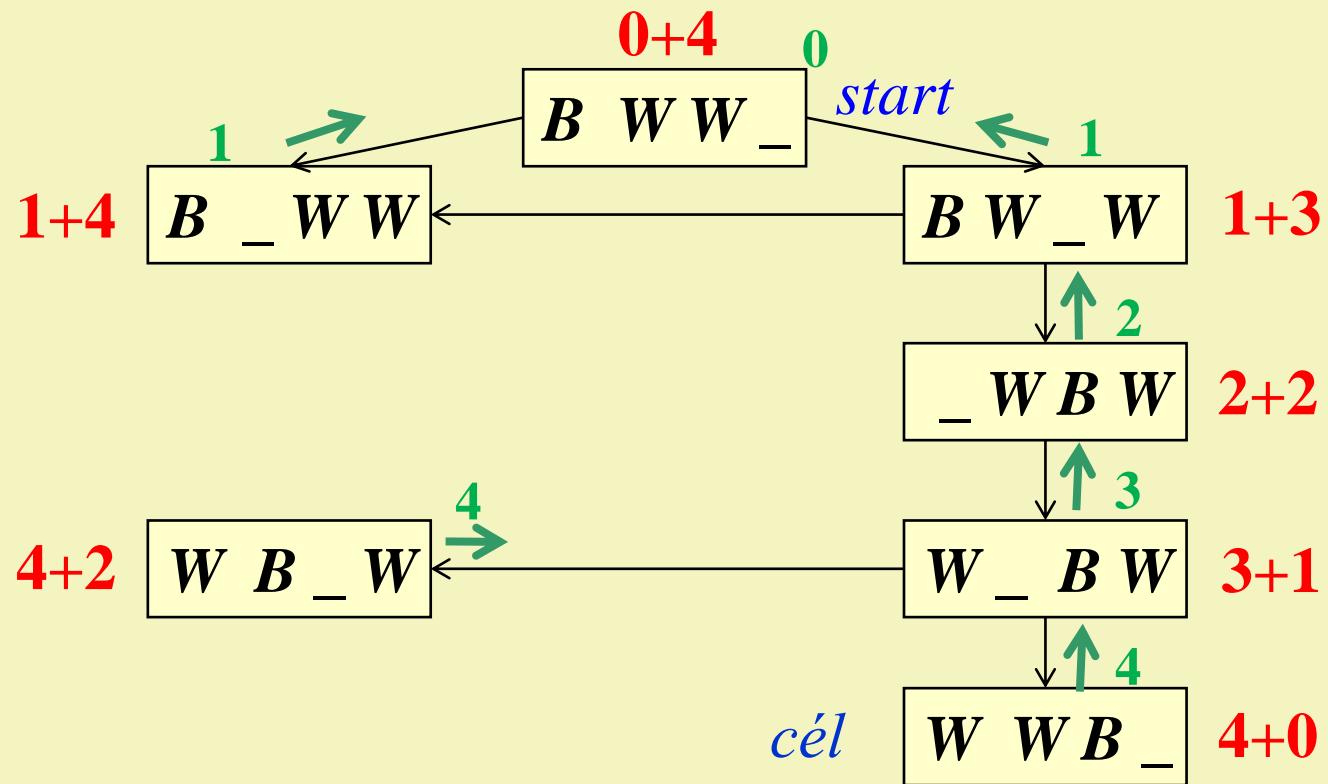
A algoritmus

$$f = g + 2*I$$



A algoritmus

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



Elemzés

$A^c alg$

f	Alg	mo	G	Γ
-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^{*} 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^{*} 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 gráfkereső algoritmus, például az A^* algoritmus és egy másik – szintén optimális megoldást találó – 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: $0 < W \leq P$ ($\leq F$)
 - 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

- Meg lehet mutatni azt, hogy az *A^{*} algoritmus* memória igénye nem rosszabb más, hasonló eredményt produkáló (megengedhető heurisztika esetén optimális megoldást garantáló, ún. megengedhető) gráfkereső algoritmusok memória igényéhez képest.
- Példák:
 - *EGK* : $f=g+0$
 - *A(^{*}) algoritmus* : $f=g+h$
 - *A^{**} algoritmus*: $f(n)=\max_{m \in start \rightarrow n} (g(m)+h(m))$ és a célcímsúcs előnyben
 - *B algoritmus*: váltogatva $f = g + h$ vagy $f = g$
 - *B' algoritmus*: $f=g+h$, ahol változnak a h értékei

Bizonyítható eredmények

- Azt nem lehet belátni, hogy az *A^{*} algoritmus* jobb, sőt azt sem, hogy nem rosszabb a többi megengedhető algoritmusnál az összes megengedhető heurisztikájú feladatra nézve.
 - De olyan másik megengedhető algoritmus sincs, amelyik jobb, vagy legalább nem rosszabb lenne a többinél az összes megengedhető heurisztikájú feladatra nézve.
 - A **monoton megszorításos heurisztikájú** feladatokon viszont az *A^{*}* nem rosszabb a többi megengedhető algoritmusnál.
 - A **nem patologikus feladatokon** (amelyek rendelkeznek olyan optimális megoldási úttal, amely csúcsaira a célcsúcs kivételével fenn áll, hogy $h < h^*$) az *A^{*}* nem rosszabb a többi megengedhető algoritmusnál.

A^{}-nál nincs jobb*

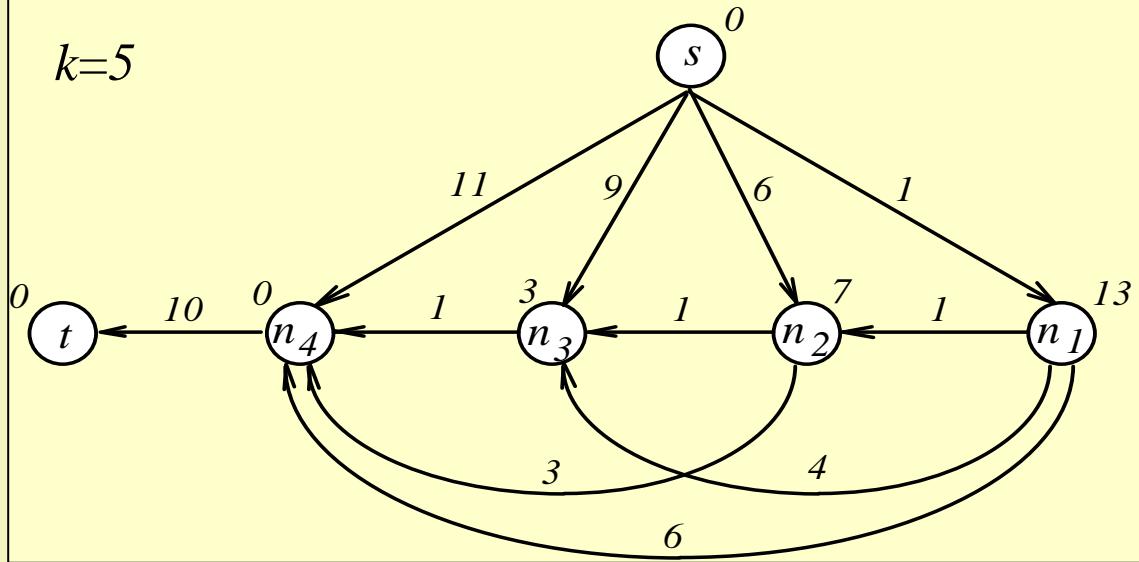
3.3.2. A futási idő elemzése

- 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.

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

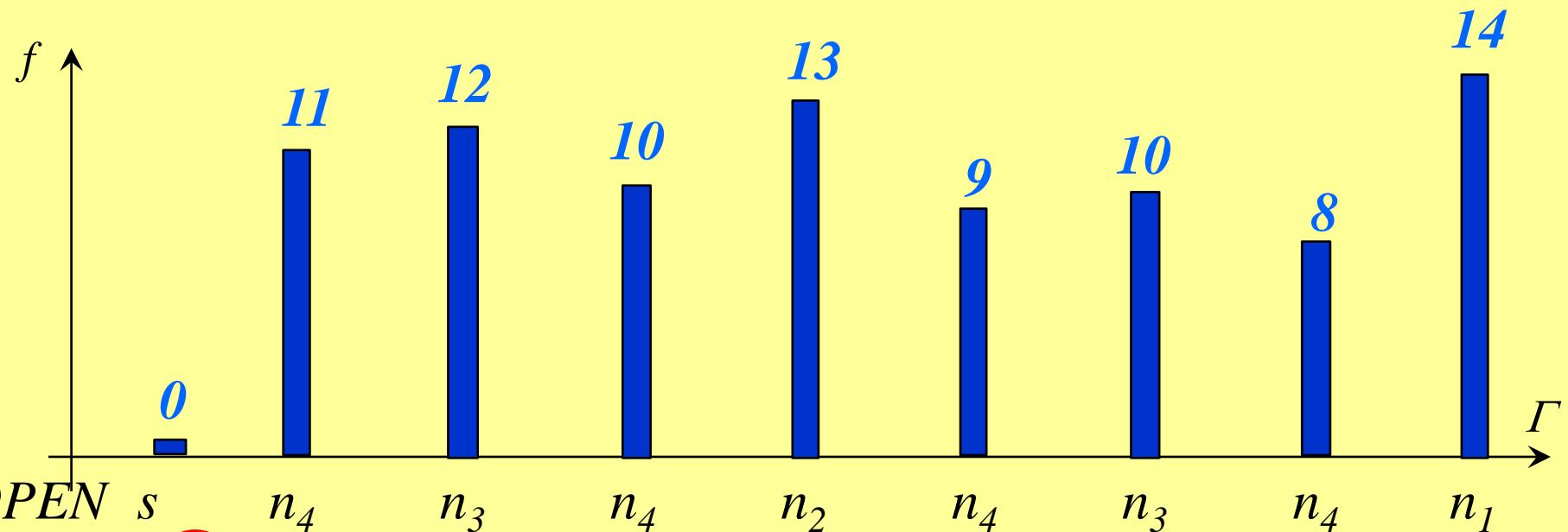
$$A = \{(n_i, n_j) \mid 0 \leq i < j < k\} \cup \{(n_{k-1}, t)\}$$

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

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

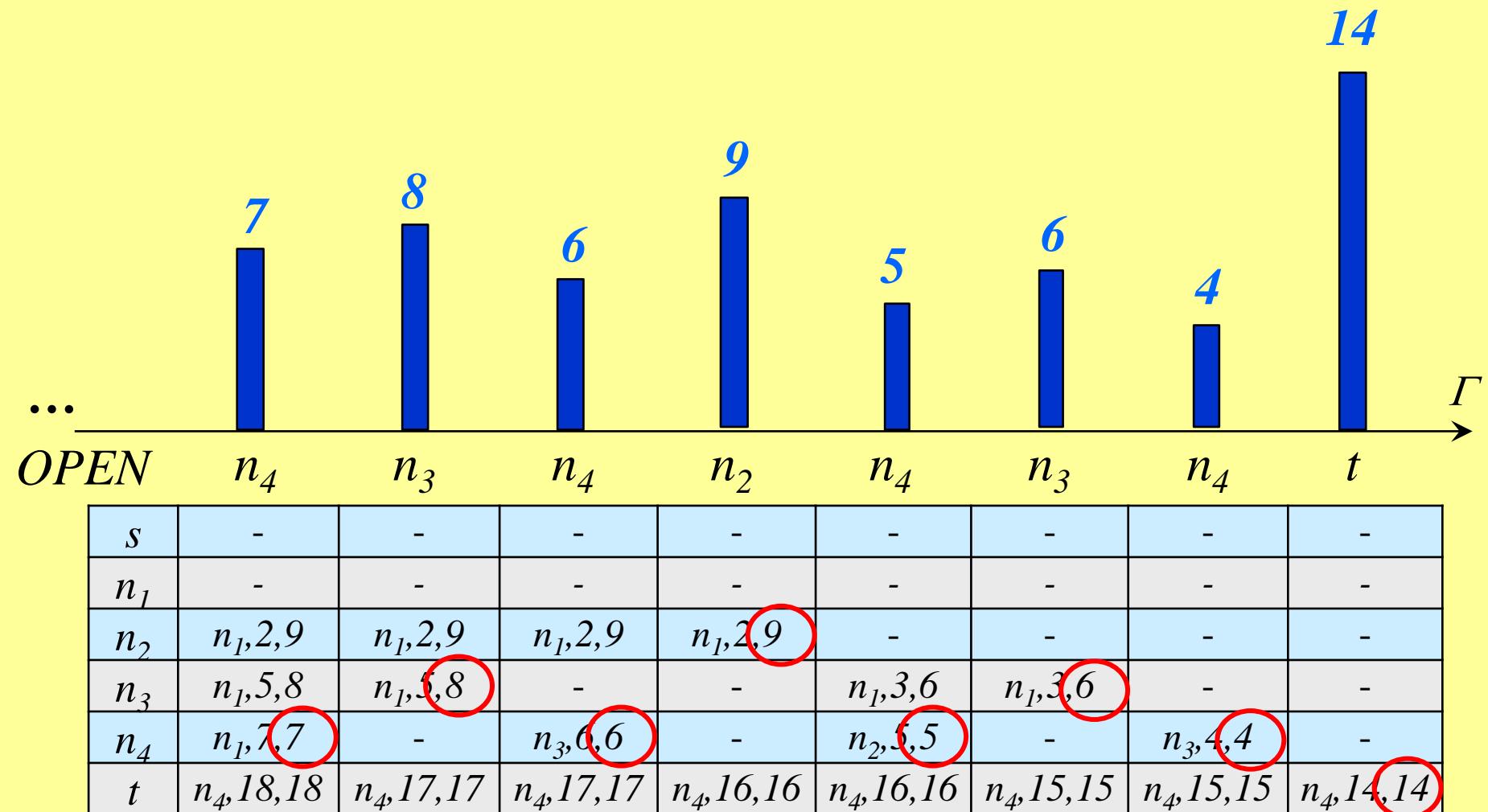
$$c(n_{k-1}, t) = h(n_1) - k + 2$$

Működési grafikon

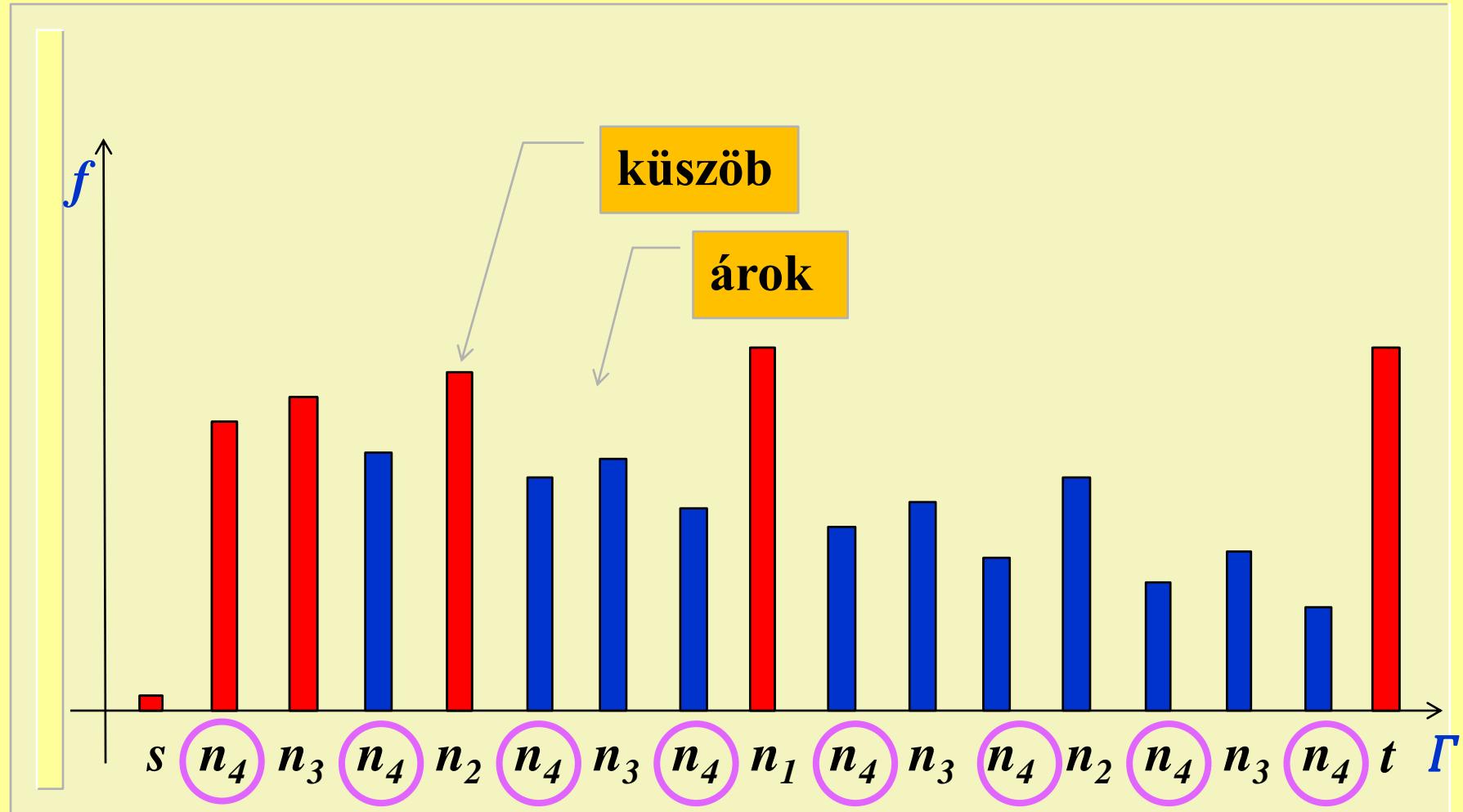


s	$nil, 0, 0$	-	-	-	-	-	-	-	-
n_1	-	$s, 1, 14$	$s, 1, 14$	$s, 1, 14$	$s, 1, 14$	$s, 1, 14$	$s, 1, 14$	$s, 1, 14$	$s, 1, 14$
n_2	-	$s, 6, 13$	$s, 6, 13$	$s, 6, 13$	$s, 6, 13$	-	-	-	-
n_3	-	$s, 9, 12$	$s, 9, 12$	-	-	$n_2, 7, 10$	$n_2, 7, 10$	-	-
n_4	-	$s, 11, 11$	-	$n_3, 10, 10$	-	$n_2, 9, 9$	-	$n_2, 8,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

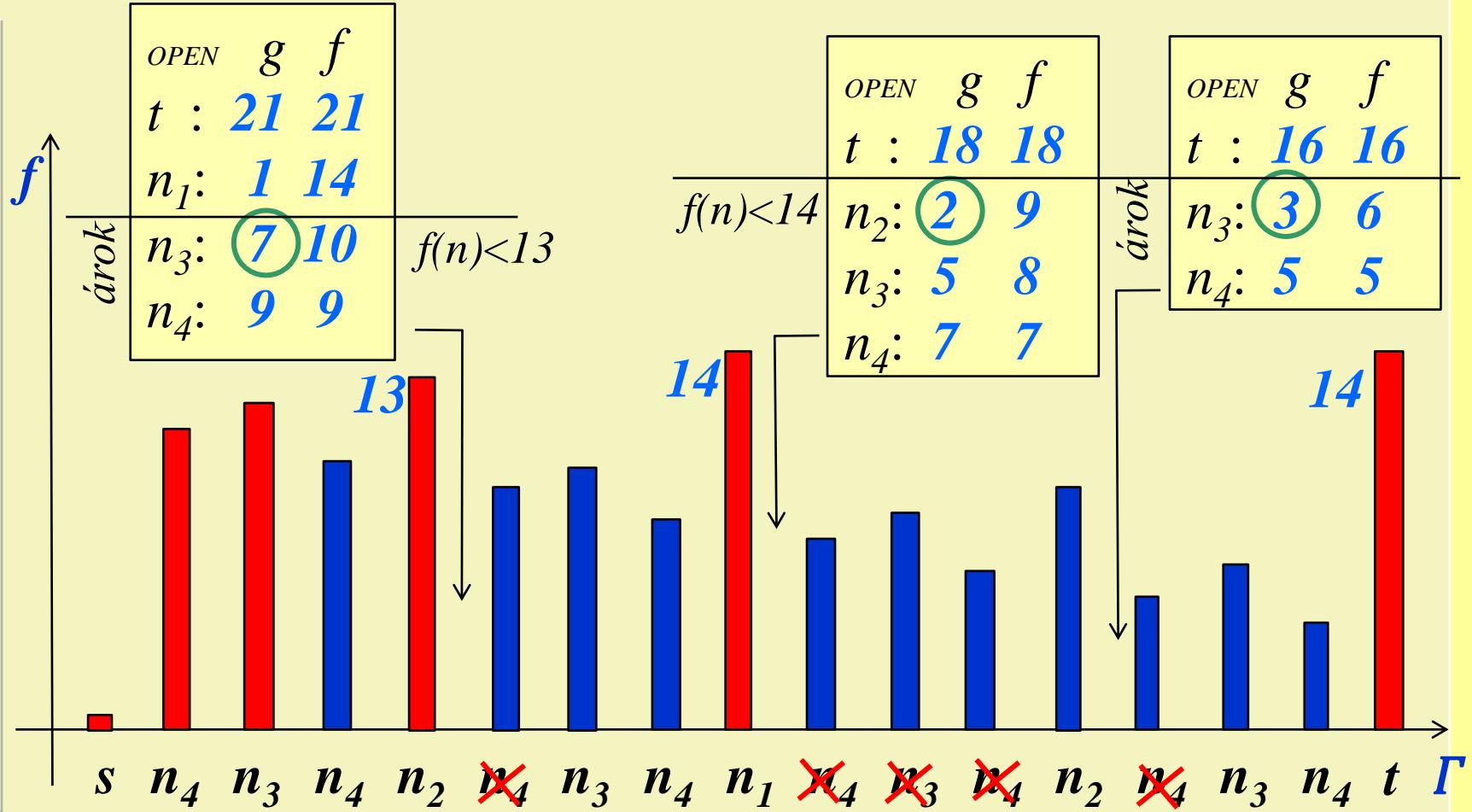
Működési grafikon



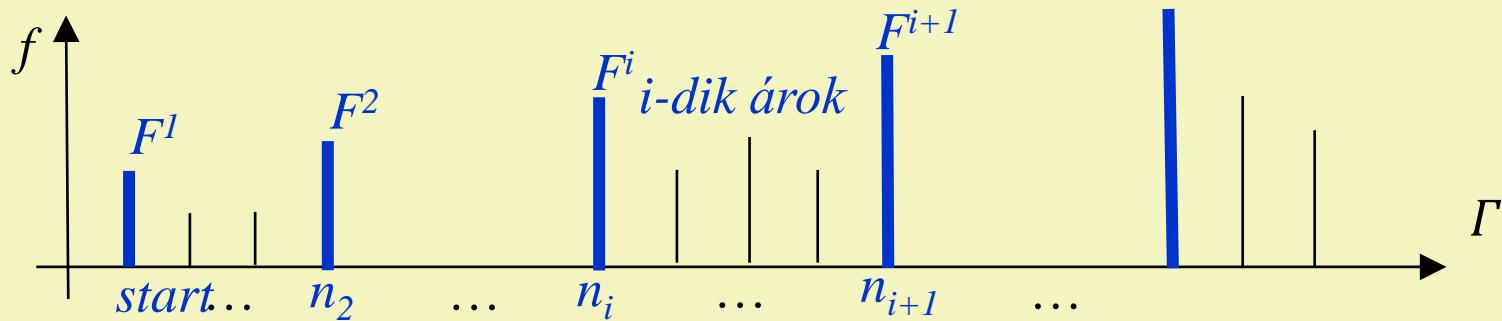
Árkon belüli kiterjesztések száma az A algoritmusnál*



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 (belso)** 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 belső 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 $n := \arg \min_g(m \in \text{OPEN} \mid f(m) < F)$
 else $n := \arg \min_f(\text{OPEN}); F := f(n)$
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

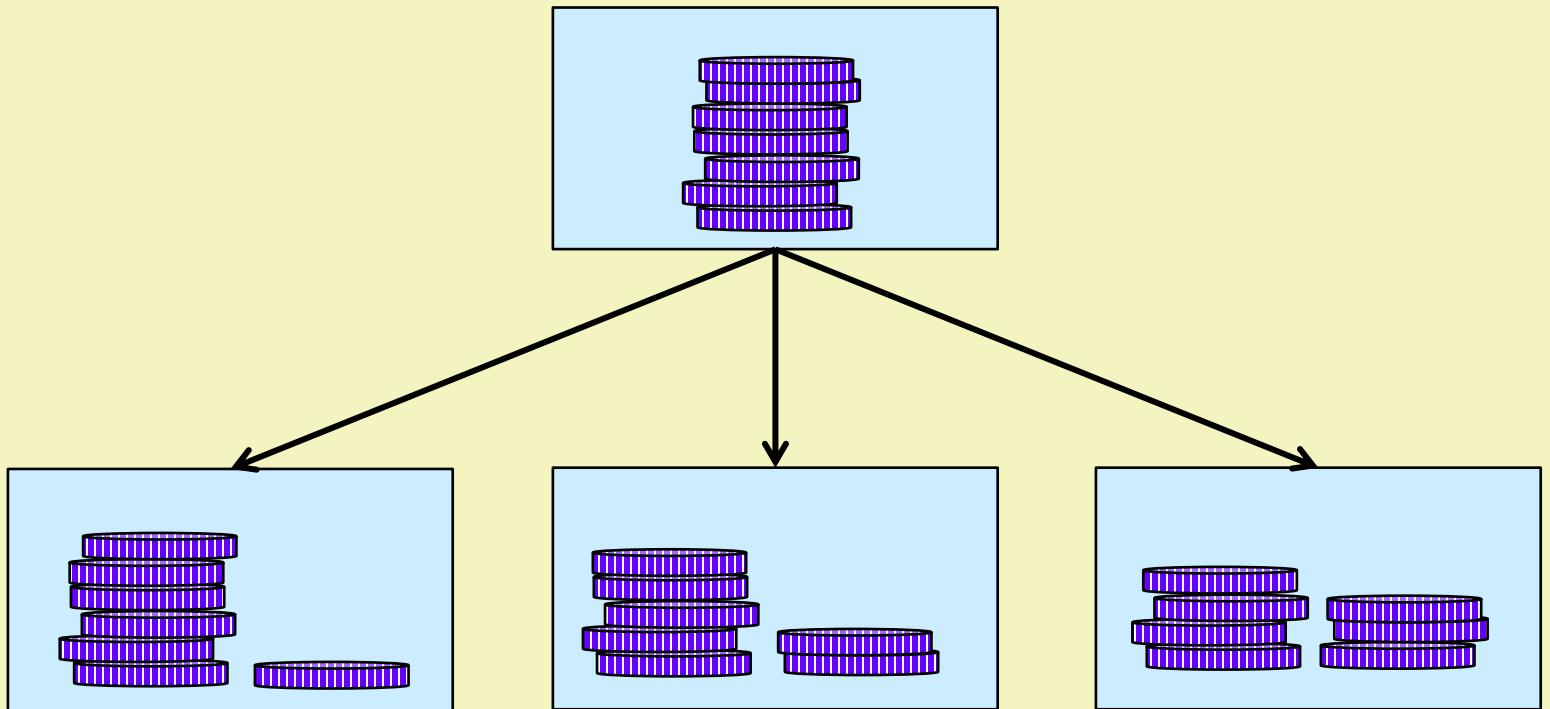
Kétszemélyes, teljes információjú, véges, determinisztikus, zéró összegű játékok

- ❑ 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)

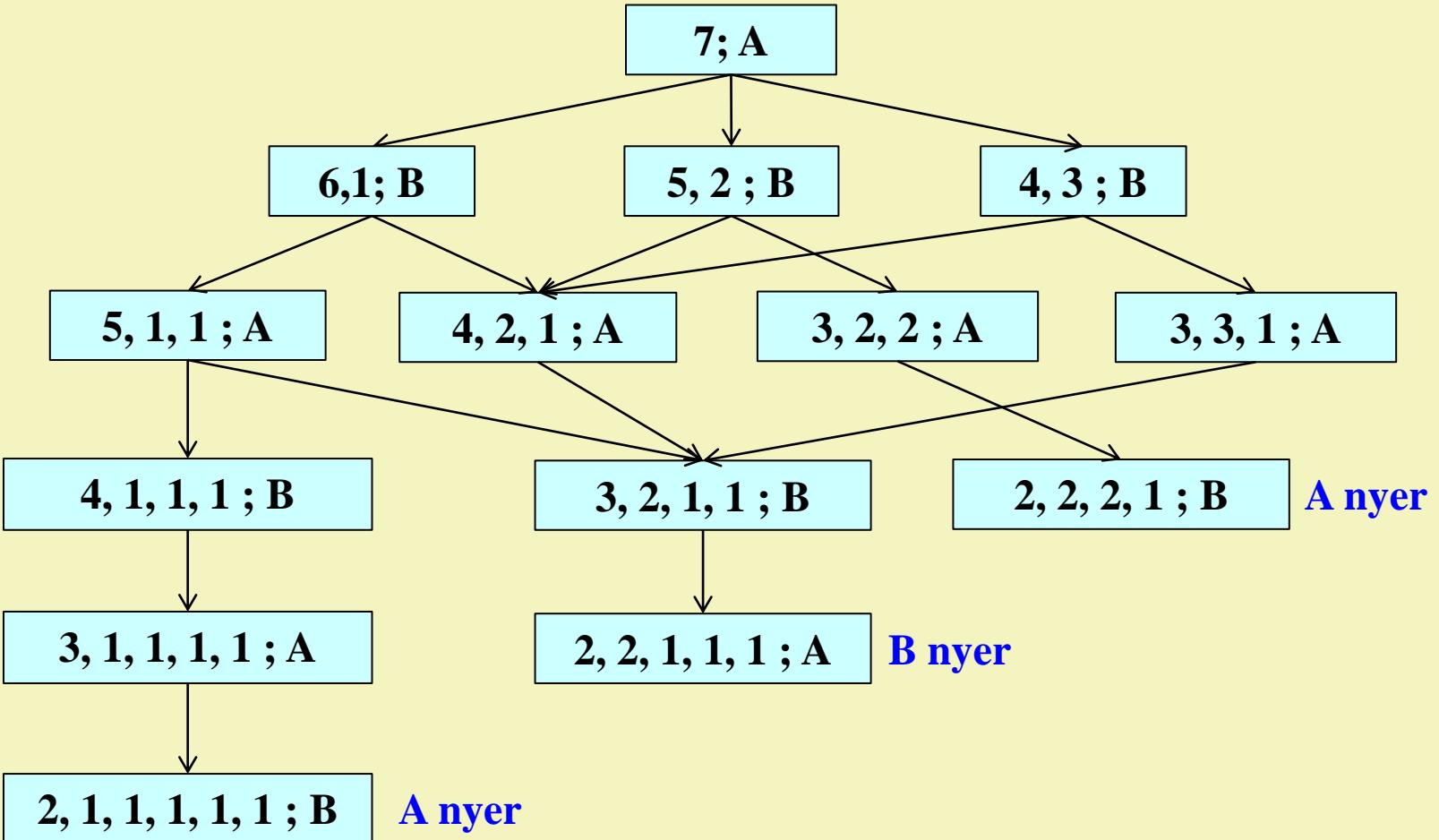
Állapottér modell

- 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

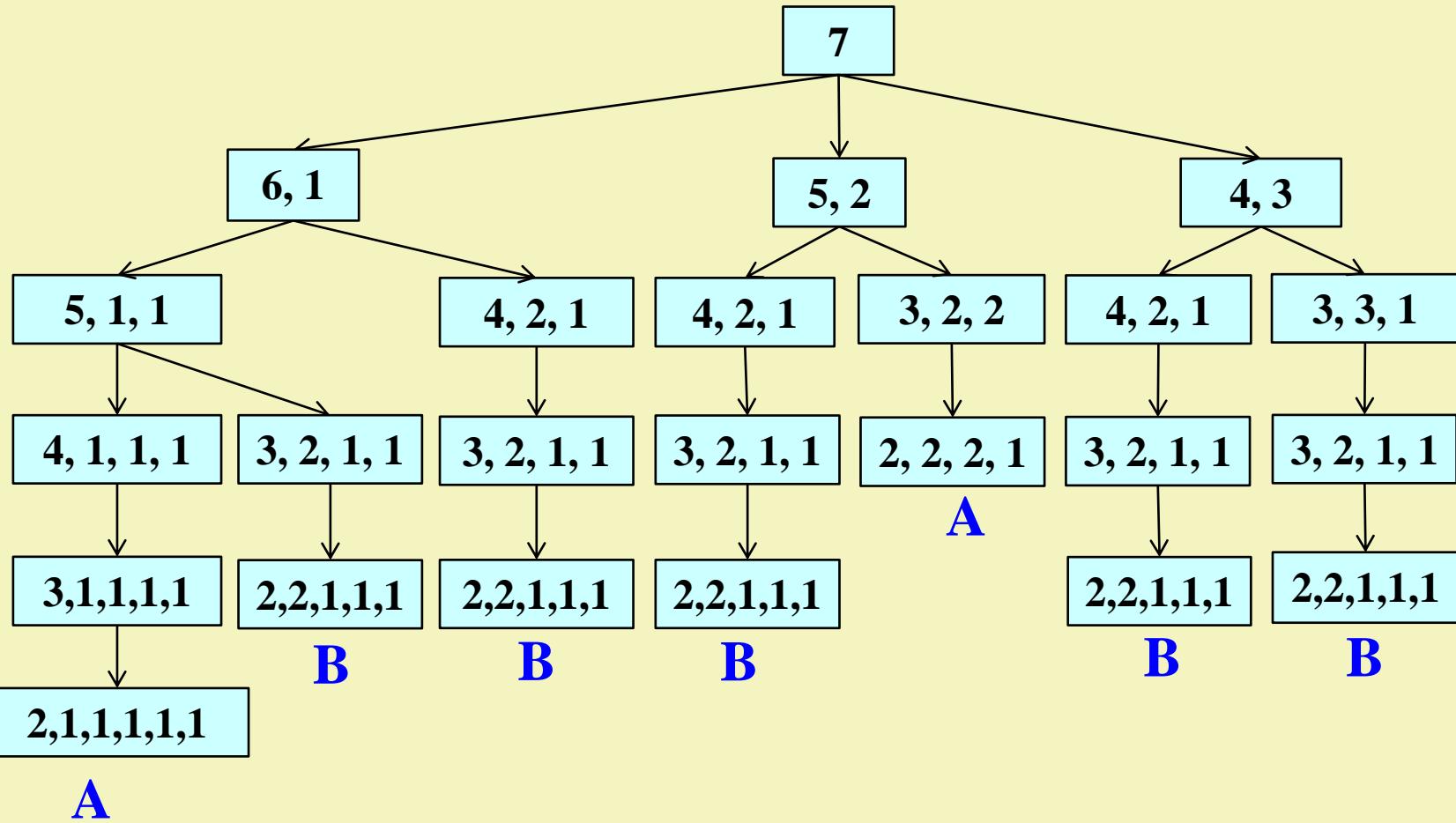
B

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

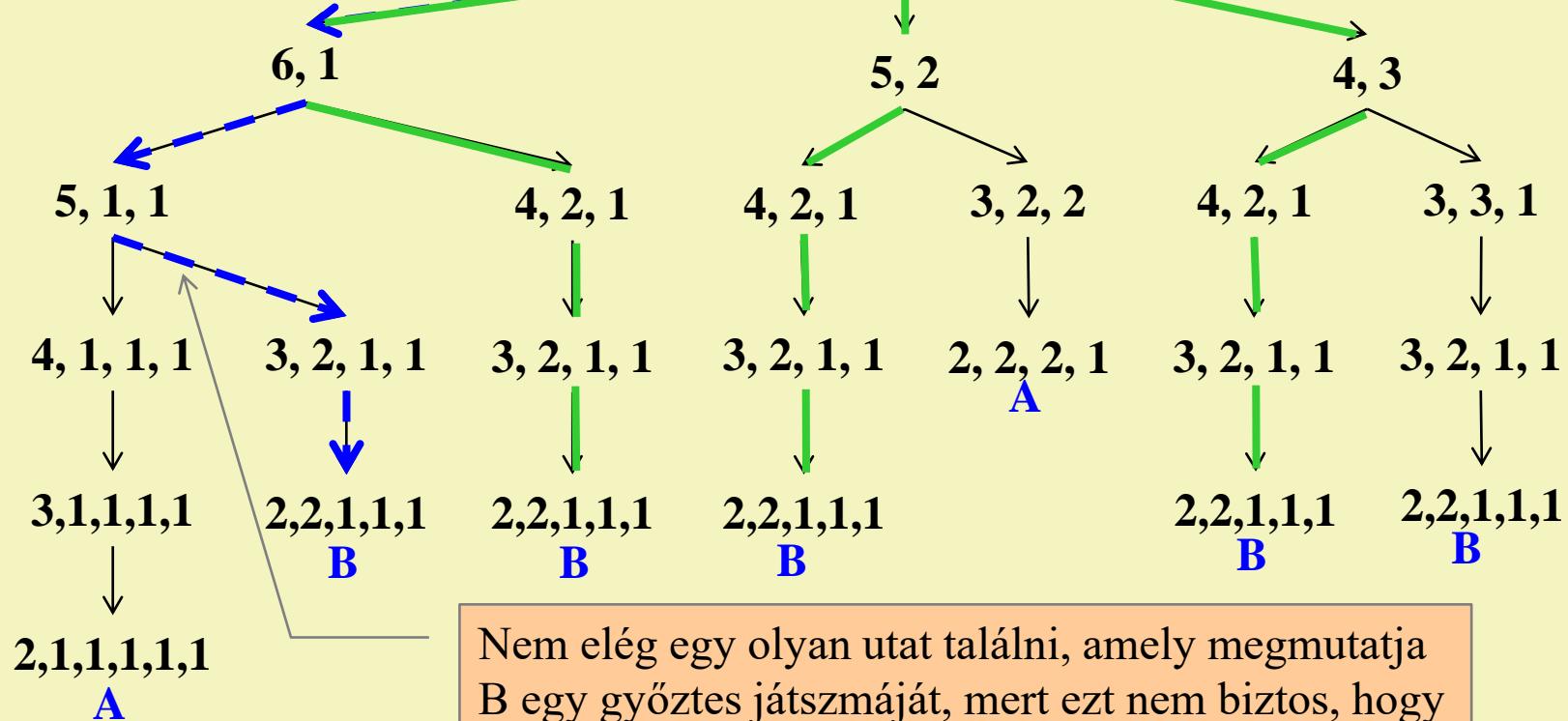
A

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.

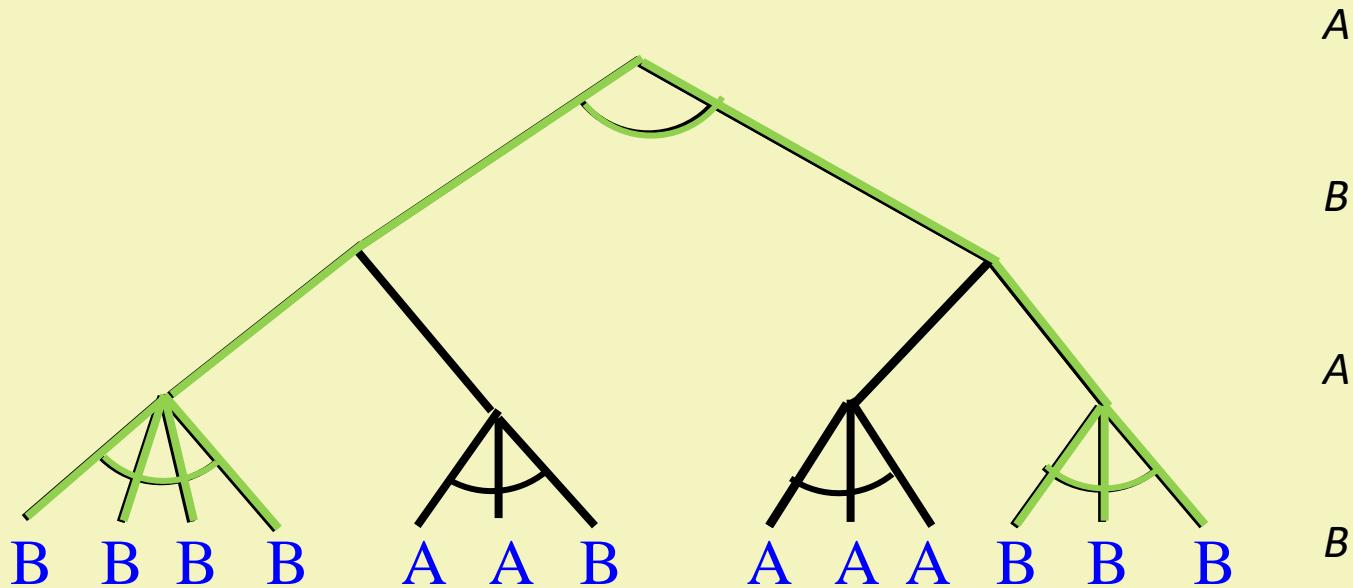
Nyerő stratégia

- 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.

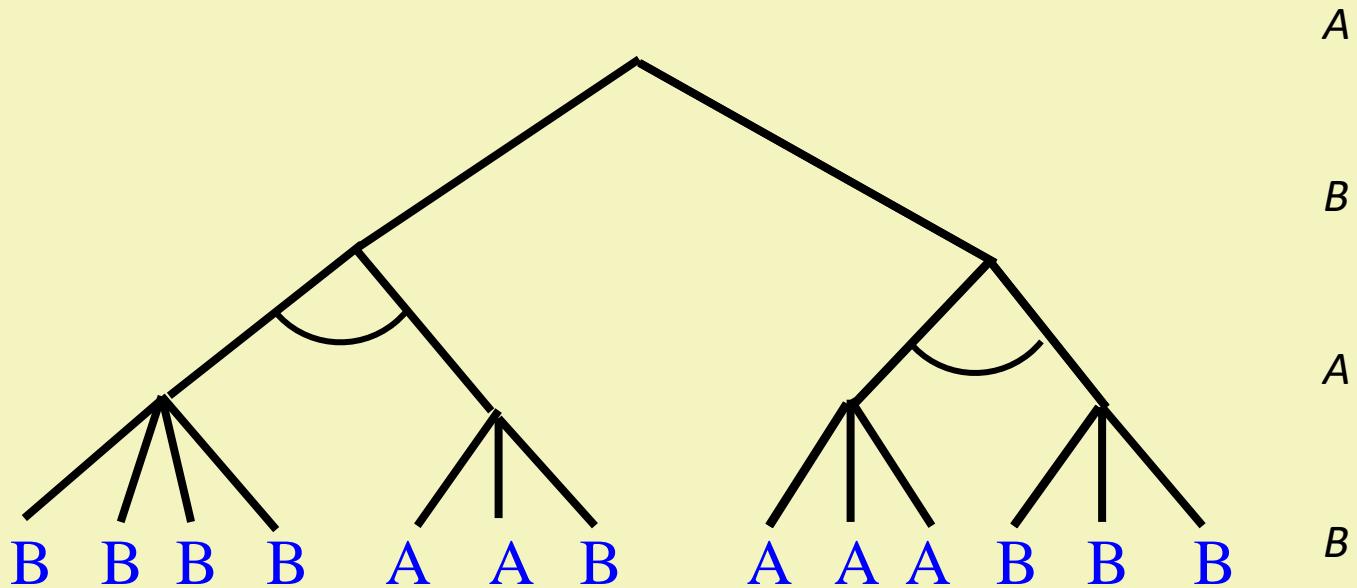
Megjegyzés

- 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 stratégiát az ÉS/VAGY játékfa azon hiper-útja mutatja, amely a gyökércsúcsból csupa nyerő 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

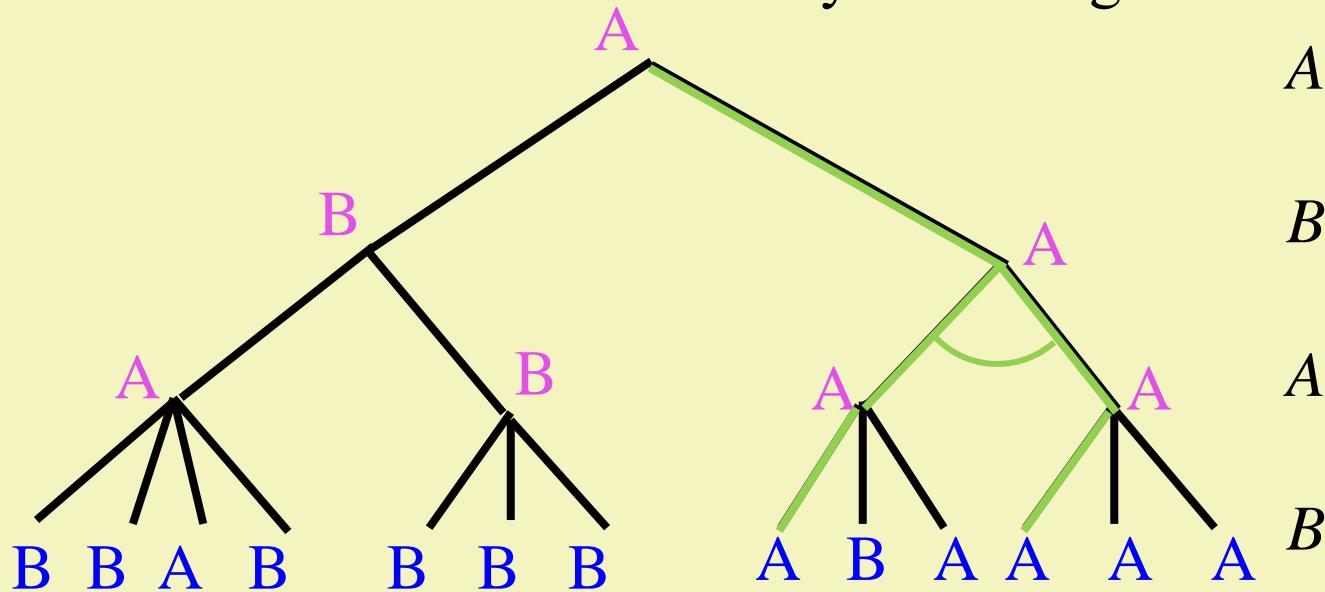


Nincs nyerő stratégia.

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.

Megjegyzés

- ❑ 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**.
- ❑ A nyerő stratégia megtalálása időigényes feladat.

sakk: 35^{90} vagy $1,76^{90} \approx 1,25 * 10^{22}$ levélcsúcs

átlagos számítógép:

$100\ 000$ csúcs/mp: $1,25 * 10^{17}$ mp ≈ 400 millió év

Deep Blue (6000 általános és 480 speciális processzor):

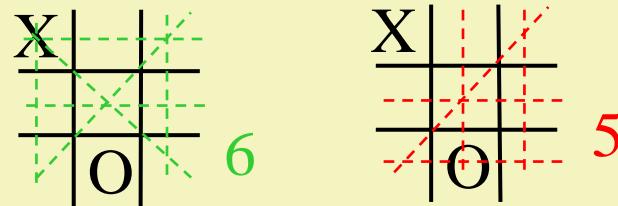
126 millió csúcs/mp: 10^{14} mp > 3 millió év

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: $f(s) = M(s) - O(s)$
 $M(s) = \text{a saját lehetséges győztes vonalaink száma}$
 $O(s) = \text{az ellenfél lehetséges győztes vonalaink száma}$

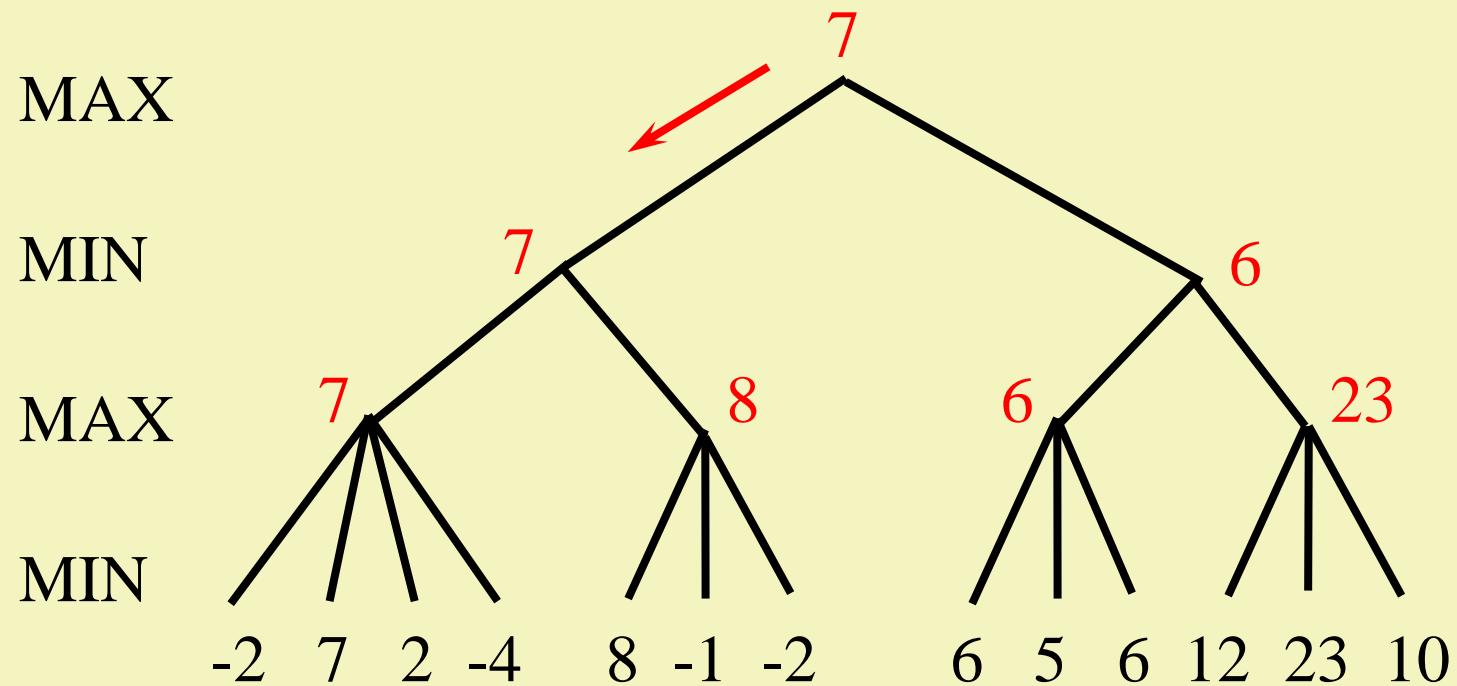


Minimax algoritmus

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.

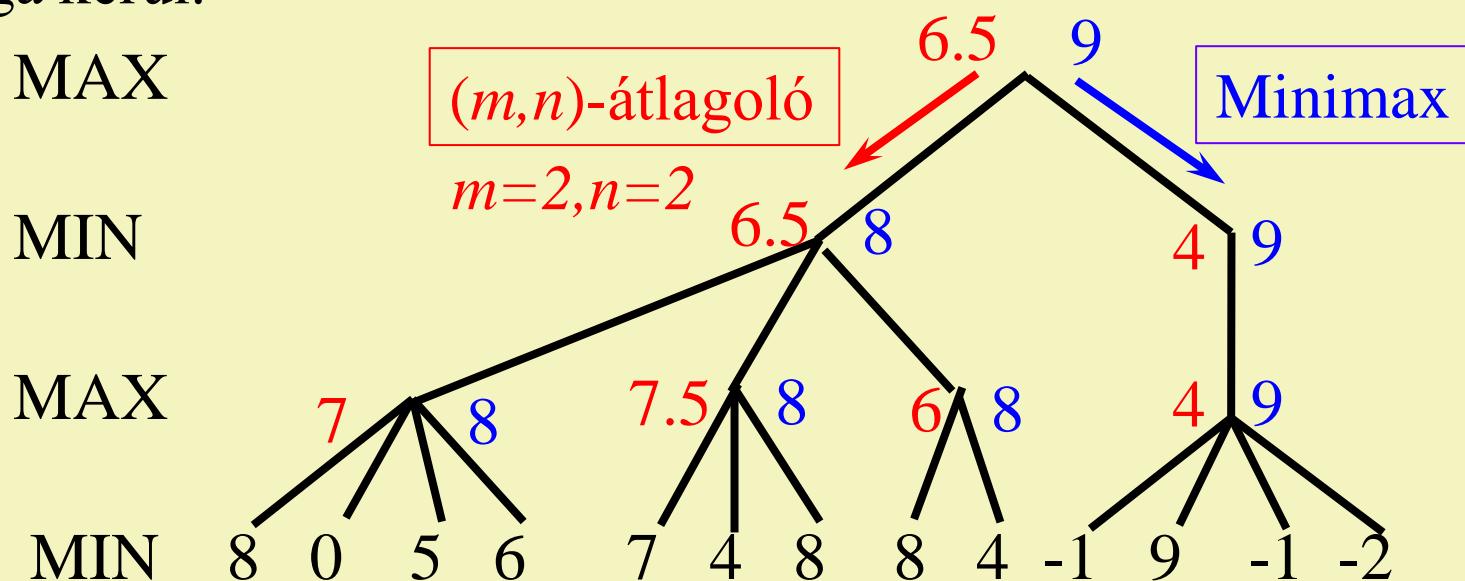


Megjegyzés

- 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

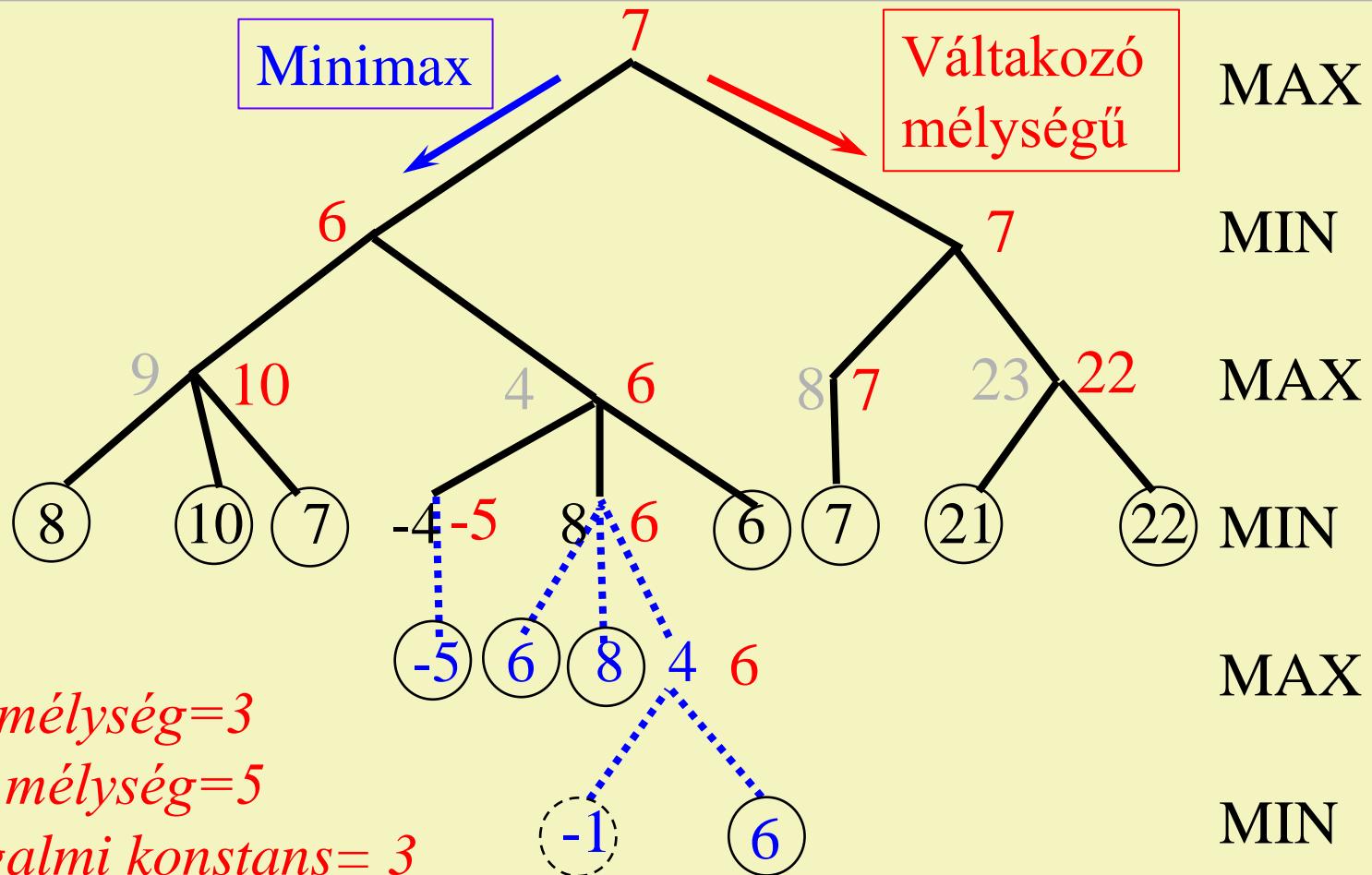
- Célja a kiértékelő függvény esetleges tévedéseinek simítása.
- MAX szintjeire az m darab legnagyobb értékű gyerek (\max_m) átlaga, a MIN-re az n darab legkisebb értékű gyerek (\min_n) átlaga kerül.



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) mindenképpen 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

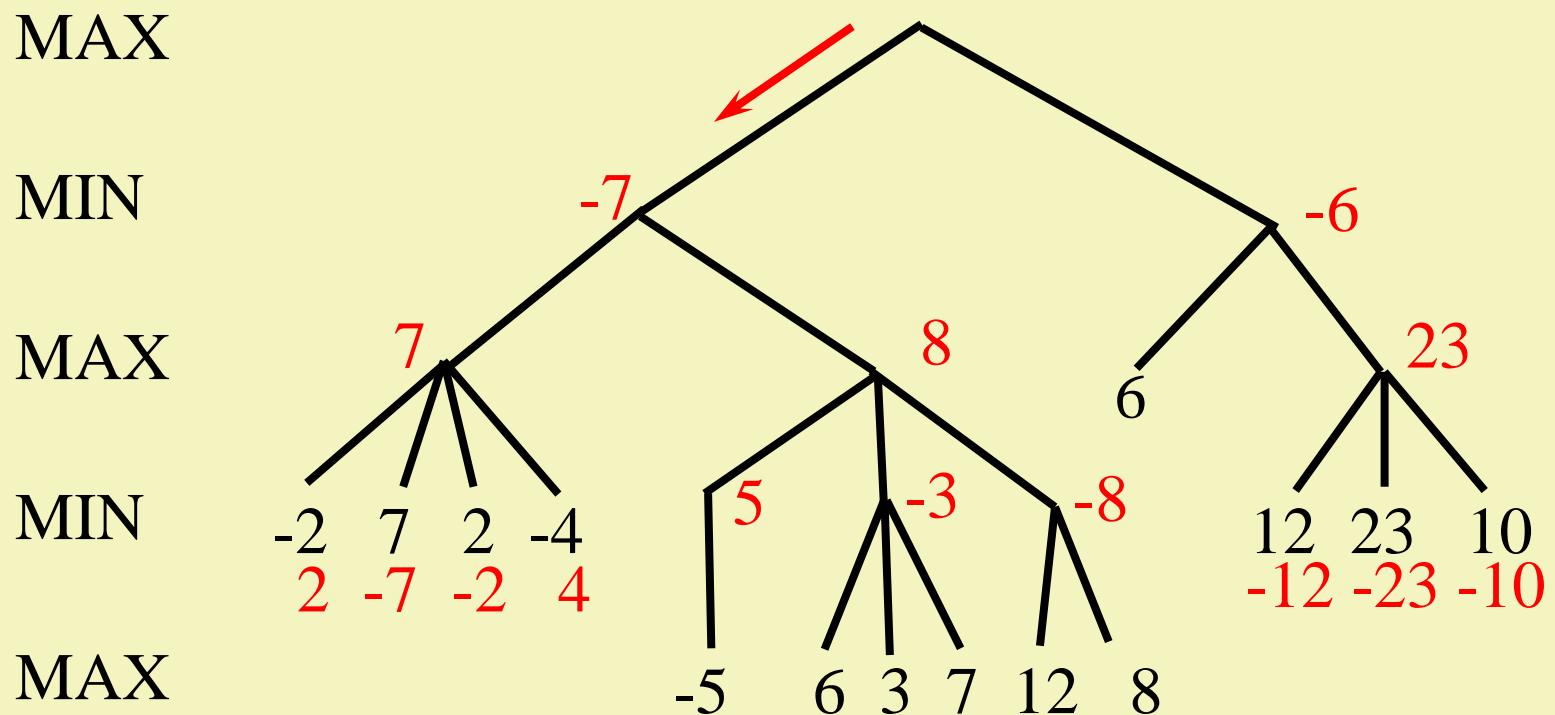
- ❑ Célja a **memória-igény** csökkentése.
- ❑ Elkülönítjük a lényeges és lényegtelen lépéseket, és csak a **lényeges lépéseknek** megfelelő részfát építjük fel.
- ❑ Ez a szétválasztás heurisztikus ismeretekre épül.

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 α érték: ennél rosszabb értékű állásba innen már nem juthatunk
 - A MIN szintjein β 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 α és β , hogy $\alpha \geq \beta$.

Példa

MAX $\alpha =$

MIN $\beta =$

MAX $\alpha =$

MIN $\beta =$

8 2

-2

7 8 4

-1

2

2

2

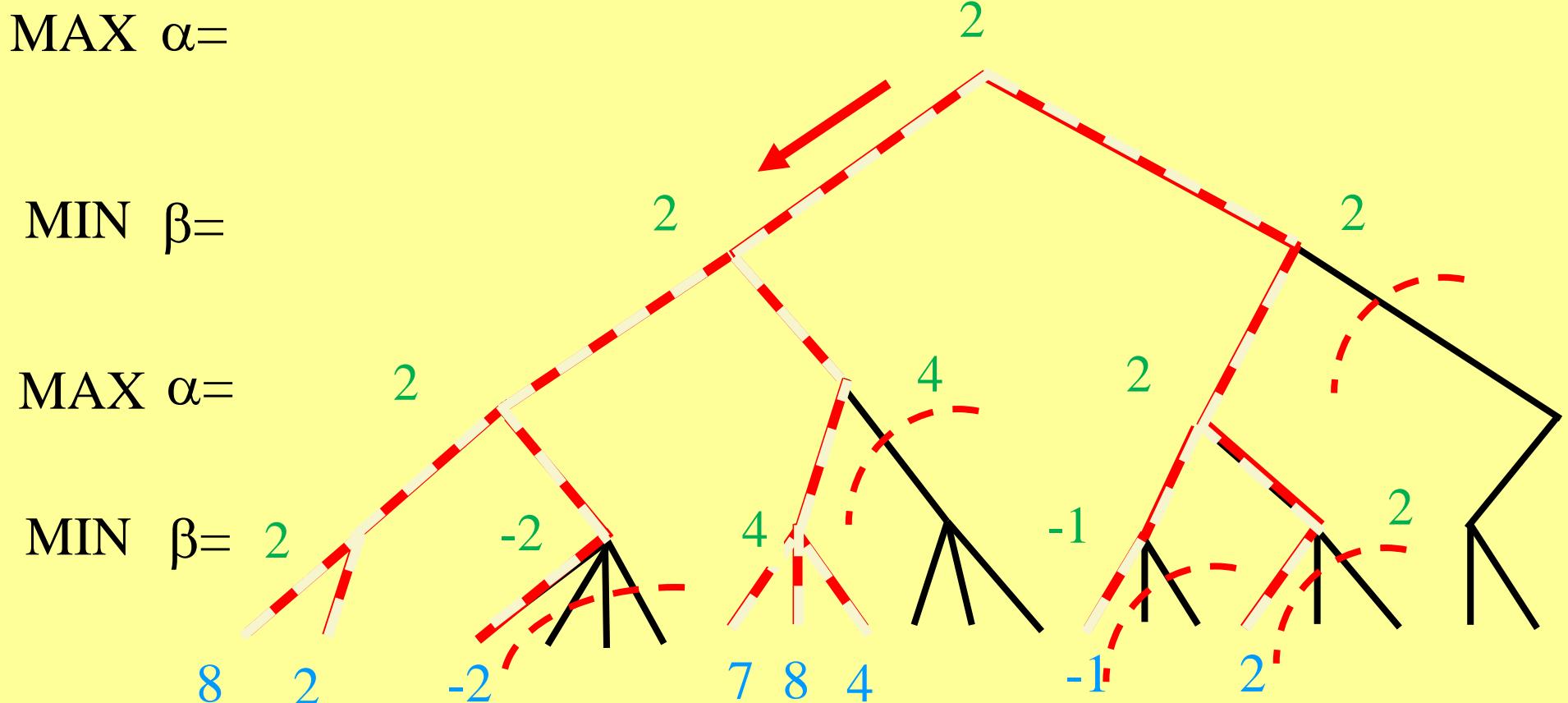
2

2

2

2

4



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**.
- Többnyire egy véletlen populációból indulunk ki, és ezt próbáljuk meg **lépésről lépésre javítani** azért, hogy abban megjelenjen egy célegyed vagy egy összeségében 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*: Kiválaszt néhány (lehetőleg rátermett) egyedet.
- *Rekombináció (keresztezés)*: A kiválasztott egyedekből, mint szülőkből olyan utódokat készít, amelyek örökölik a szülők tulajdonságait.
- *Mutáció*: Az utódok tulajdonságait kismértékben módosítja.
- *Visszahelyezés*: Új populációt alakít ki az utódokból és a régi populacióból.
- *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*)

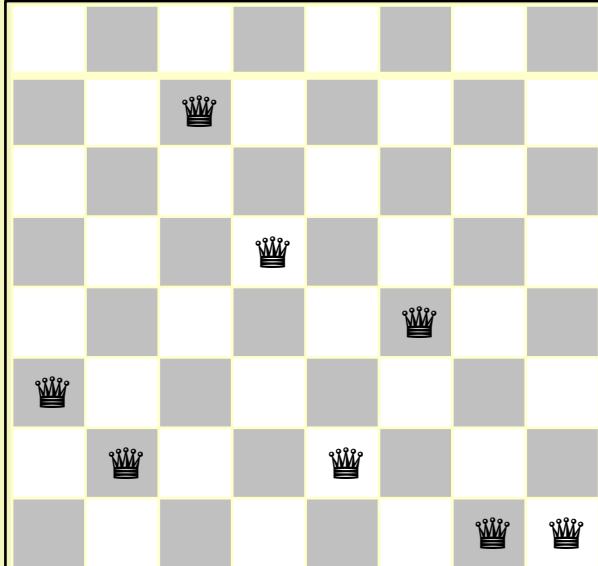
 utódok := rekombináció(szülők)

 utódok := mutáció(utódok)

populáció := visszahelyezés(*populáció*, utódok)

endloop

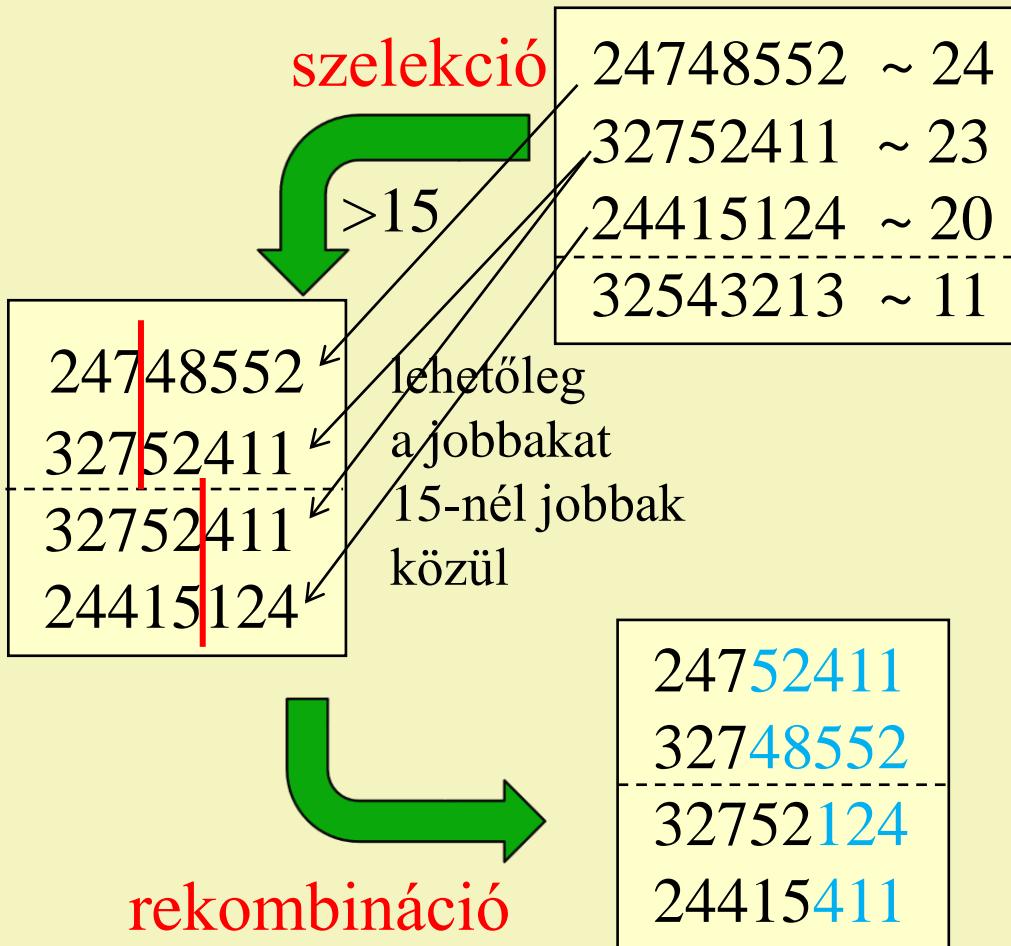
n-királynő probléma 1.



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

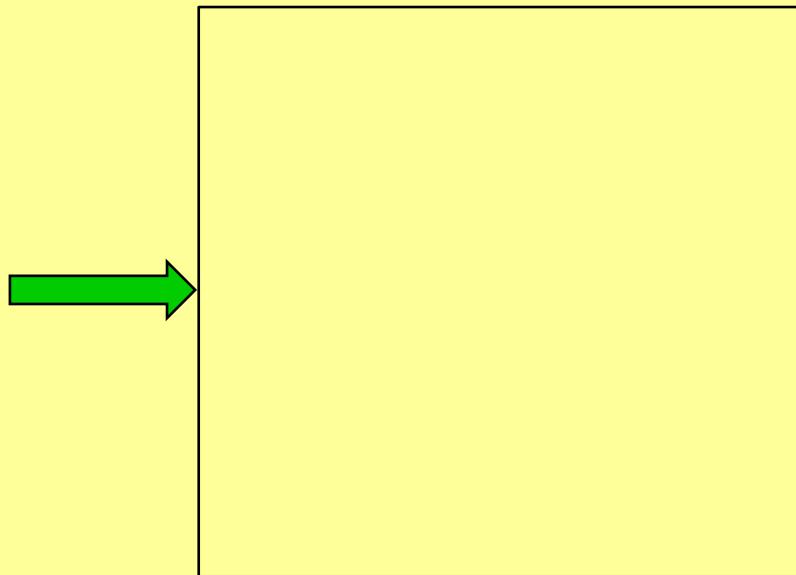
Evolúciós ciklus



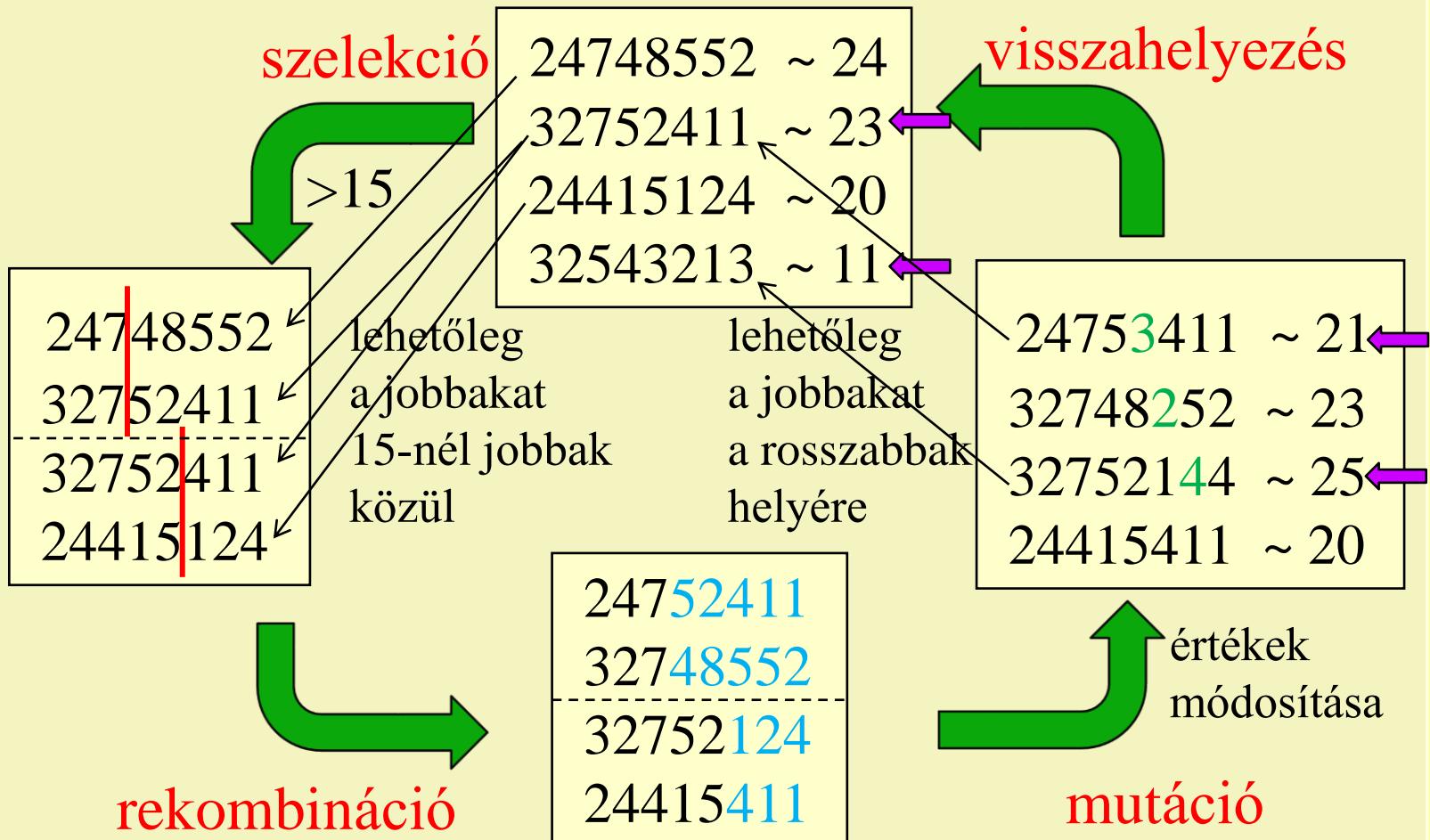
Keresztezés

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

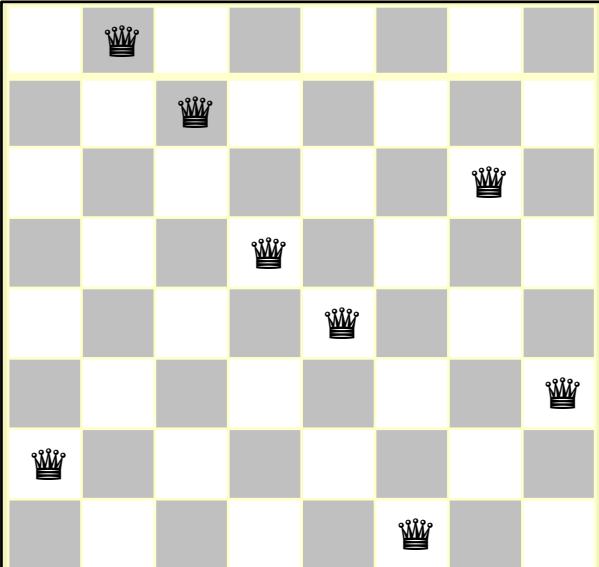
3	2	7	5	2	4	1	1
---	---	---	---	---	---	---	---



Evolúciós ciklus



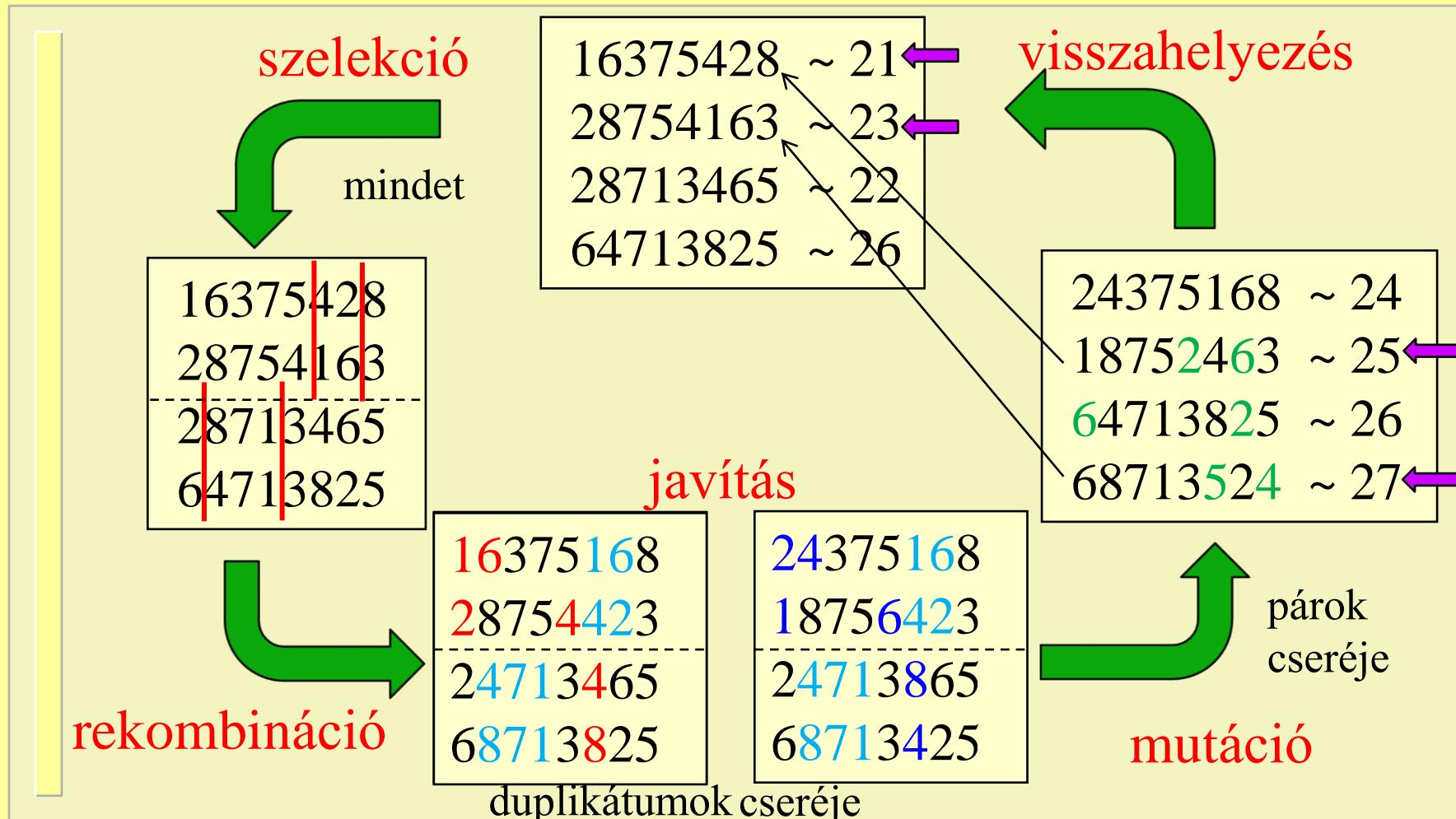
n-királynő probléma 2.



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: ütében nem levő királynő párok száma

Evolúciós ciklus



Kielégíthetőségi probléma (SAT)

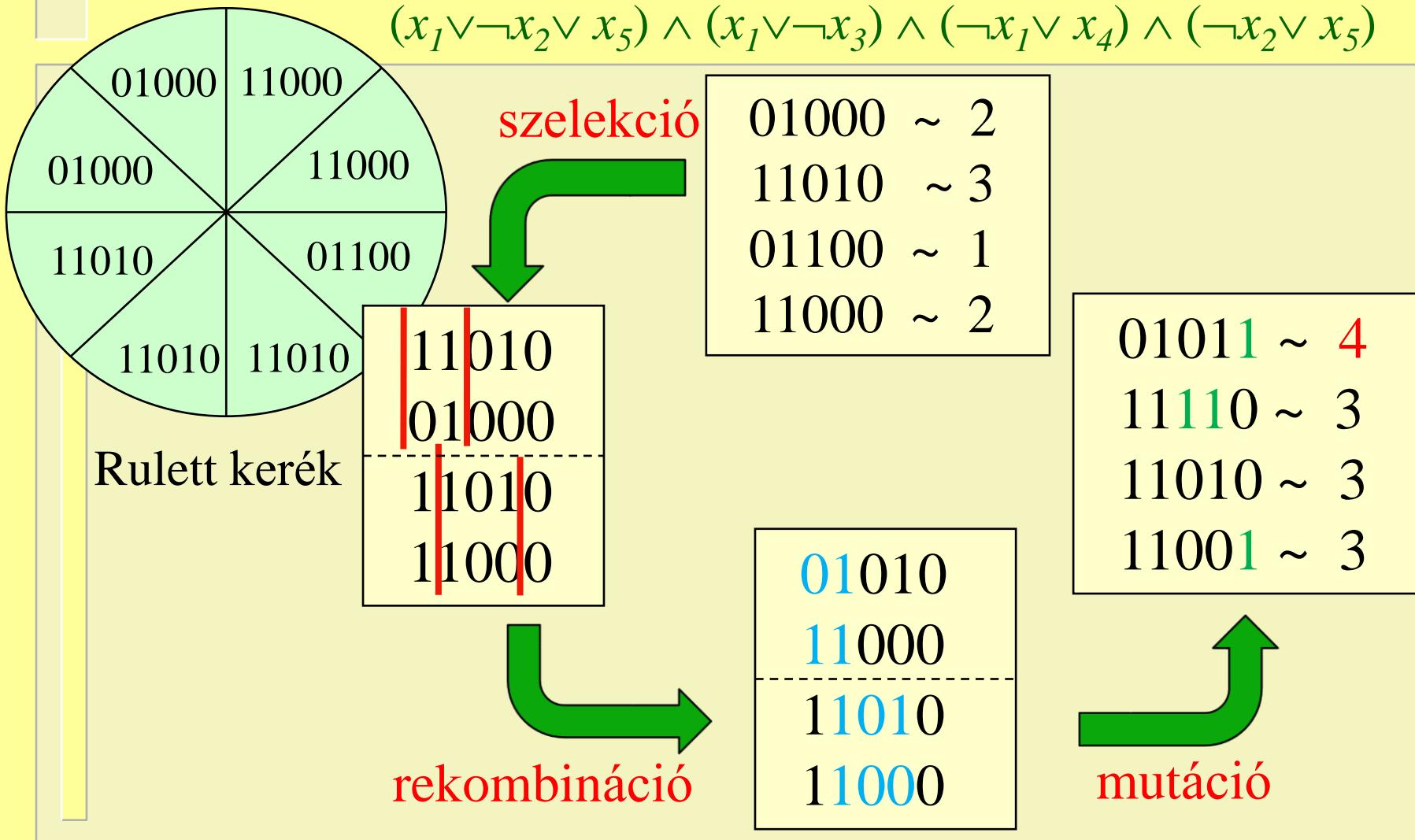
Adott egy n változós Boolean formula KNF alakban. A változók milyen igazság kiértékelése mellett lesz formula igaz?

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

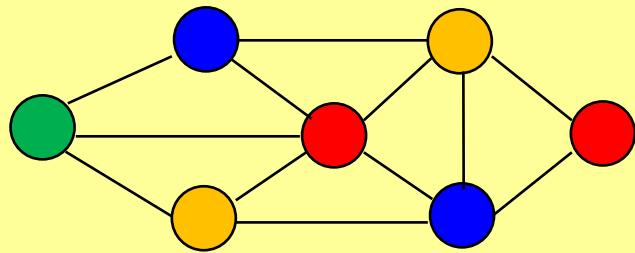


Evolúciós algoritmus tervezése

- 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 tulajdonságként **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.

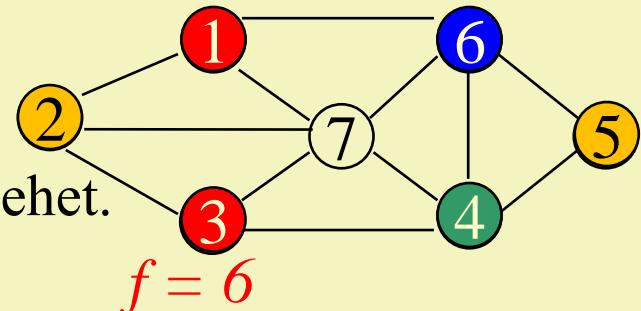
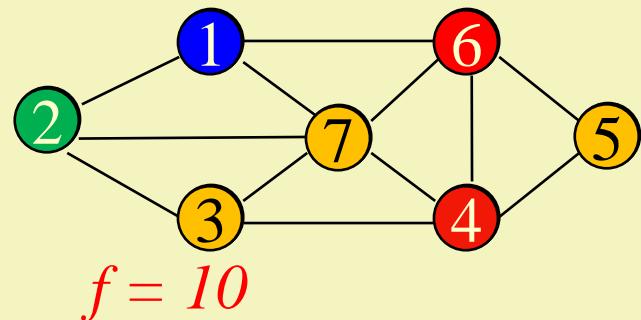
Az $x[i]$ az i -dik csúcs színe.

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>	K	P	<i>Javaslat:</i>	K
	<i>Ő:</i>	O	O		
 - 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

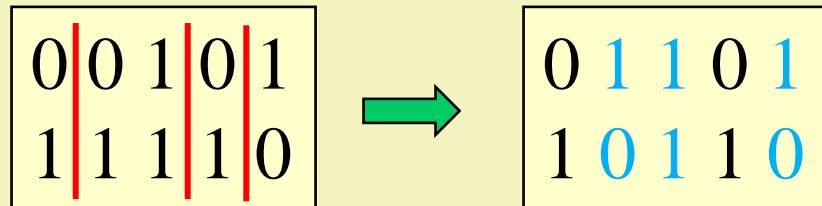
- A feladata az, hogy adott szülő-egyedekből olyan utódokat hozzon létre, amelyek a szüleik tulajdonságait "öröklik".
 - **Keresztezés**: véletlen kiválasztott pozíción jelcsoportok (gének) vagy jelek cseréje
 - **Rekombináció**: a szülő egyedek megfelelő jeleinek kombinálásával kapjuk az utód megfelelő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

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

Keresztezés

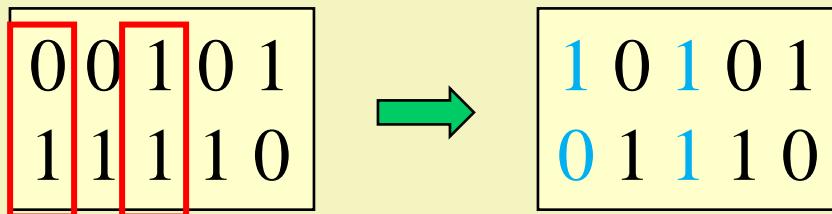
❑ Egy- illetve többpontos keresztezés

- 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

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

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

duplikátumok keresése
és párba állítása

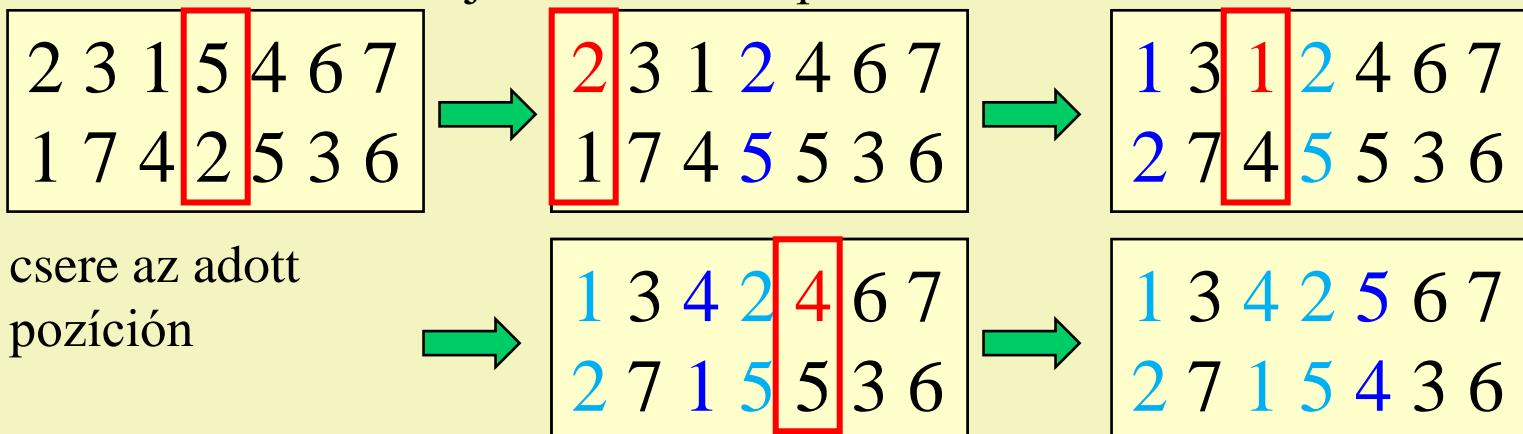
duplikátumpárok
cseréje

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.

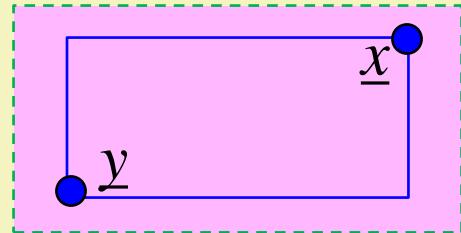
duplikátum keresése a felső utóban,
majd csere azon a pozíción is



Rekombináció vektorokra

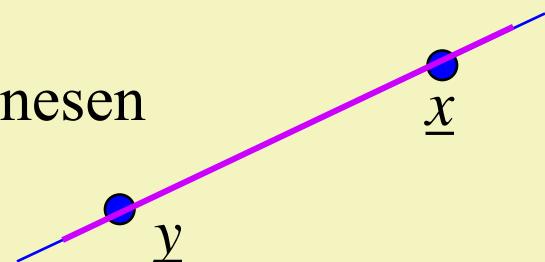
❑ Köztes rekombináció

- 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:
 - $\forall i=1 \dots n : z_i = 1 - x_i \text{ if } random[0..1] < p$
- 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

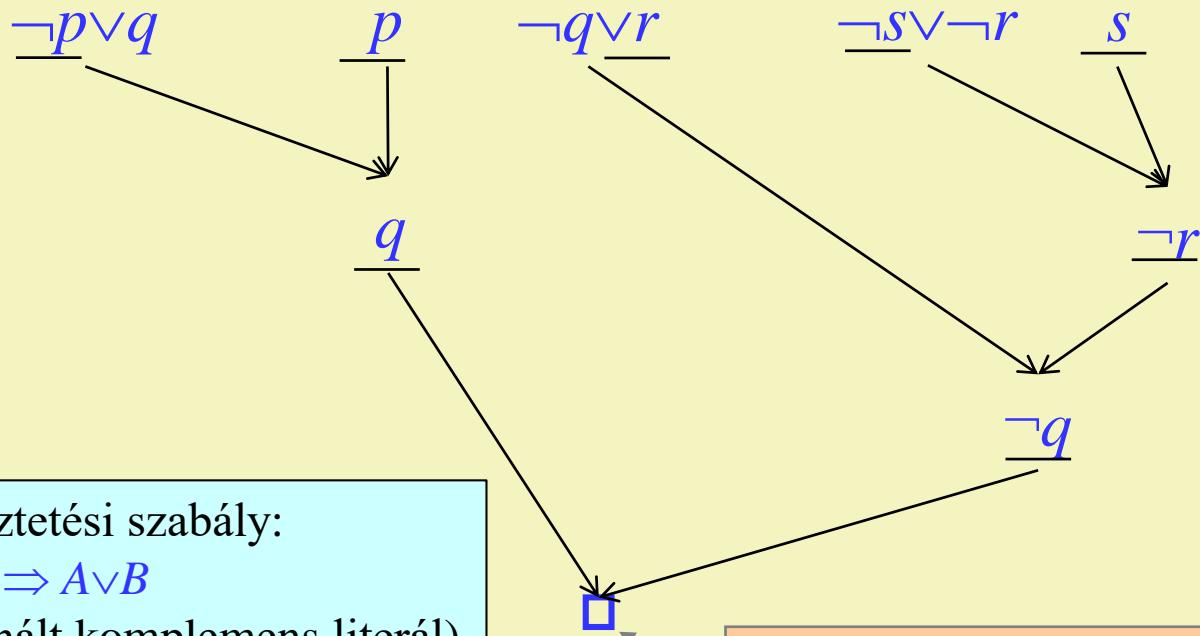
KNF: klózok között ‘és’ művelet
klóz: literálok között ‘vagy’ művelet
literál: ítéletváltozó vagy annak negáltja

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

Rezolúció = indirekt bizonyítás

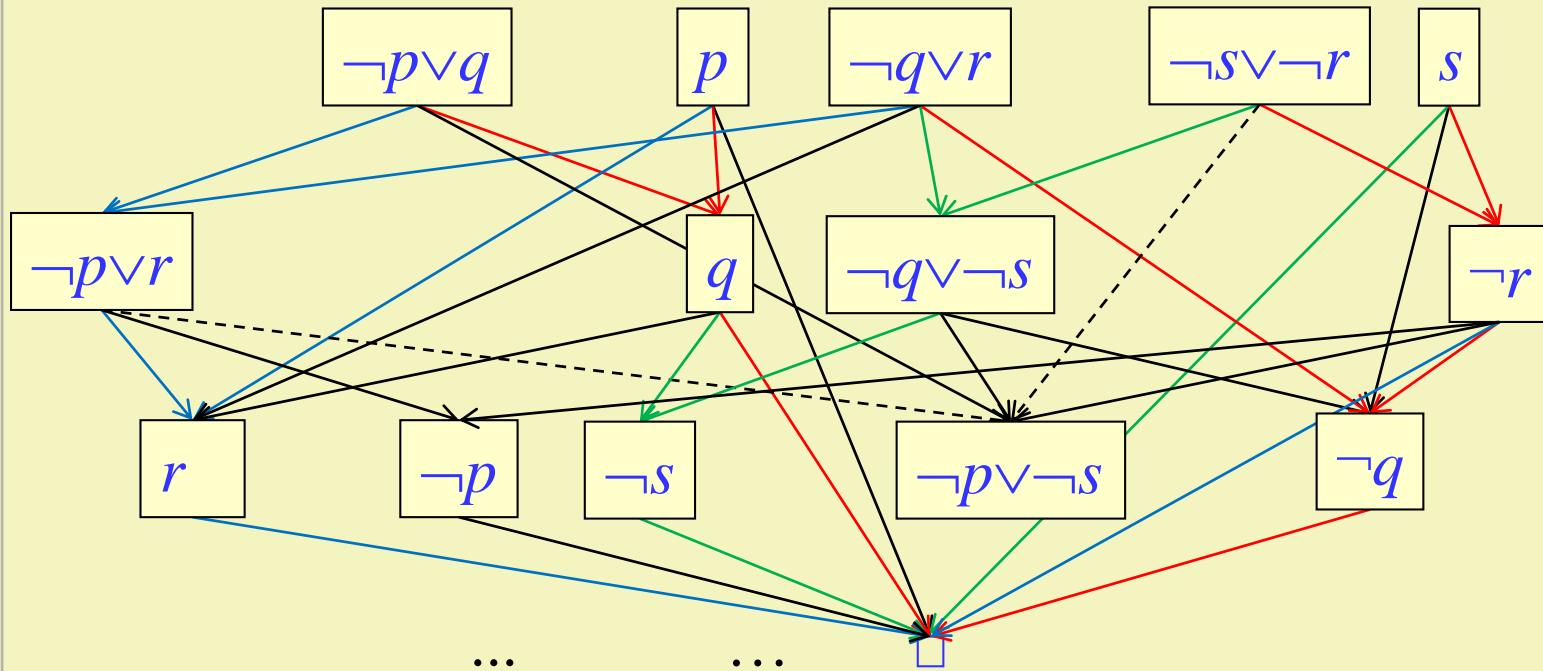
- Tekintsük a klózok halmazát és tegyük fel indirekt módon, hogy van olyan interpretáció, amikor **mindegyik klóz igaz**.
- Ekkor például p is, és $\neg p \vee q$ is *igaz*. Ha azonban p *igaz*, akkor $\neg p$ *hamis*, és ekkor a $\neg p \vee q$ csak úgy lehet *igaz*, ha a q is *igaz*.
- A q – amely 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



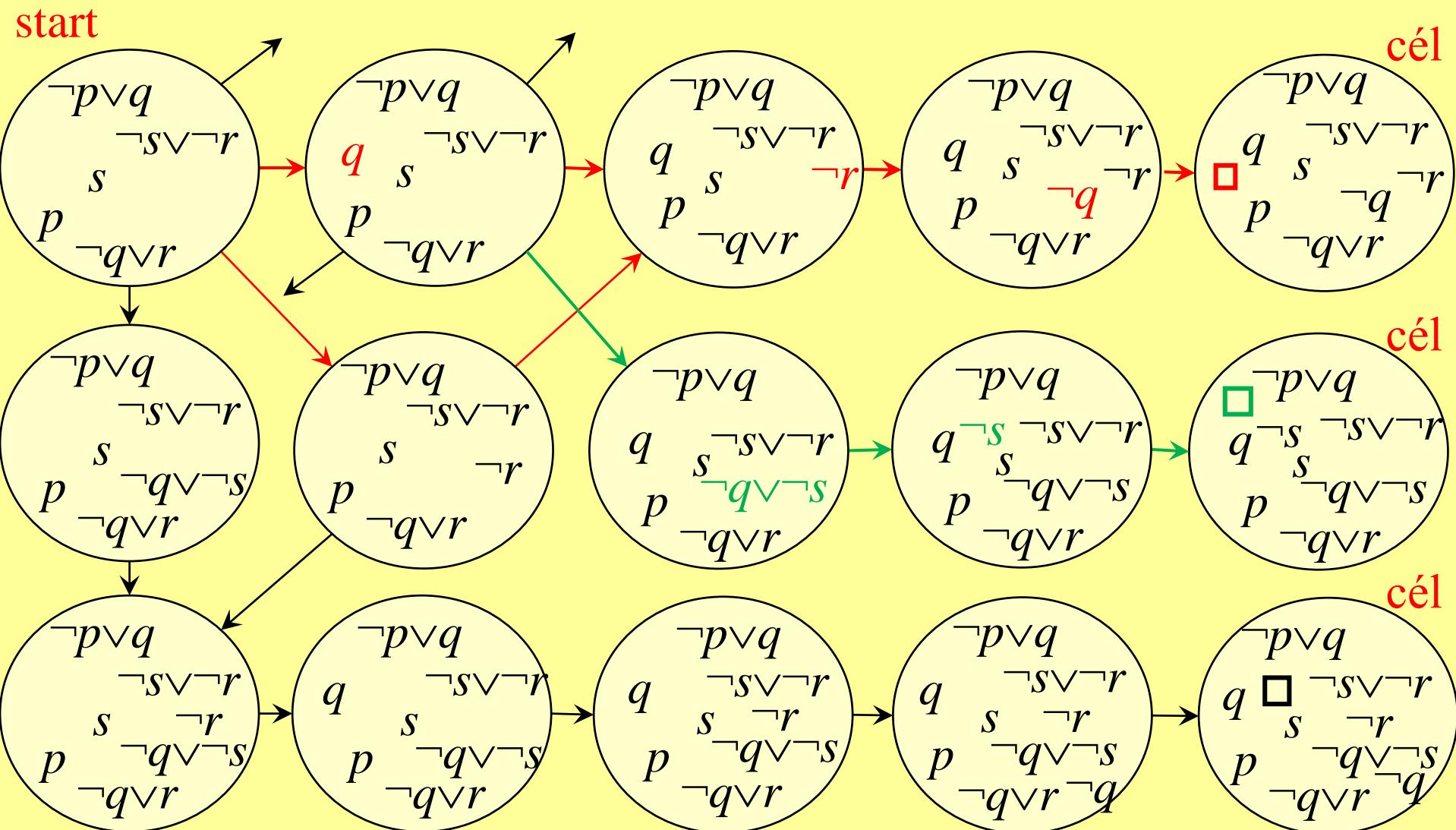
Tehát ha süt a nap, akkor Péter nem marad otthon.

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 $\neg a \vee b$

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

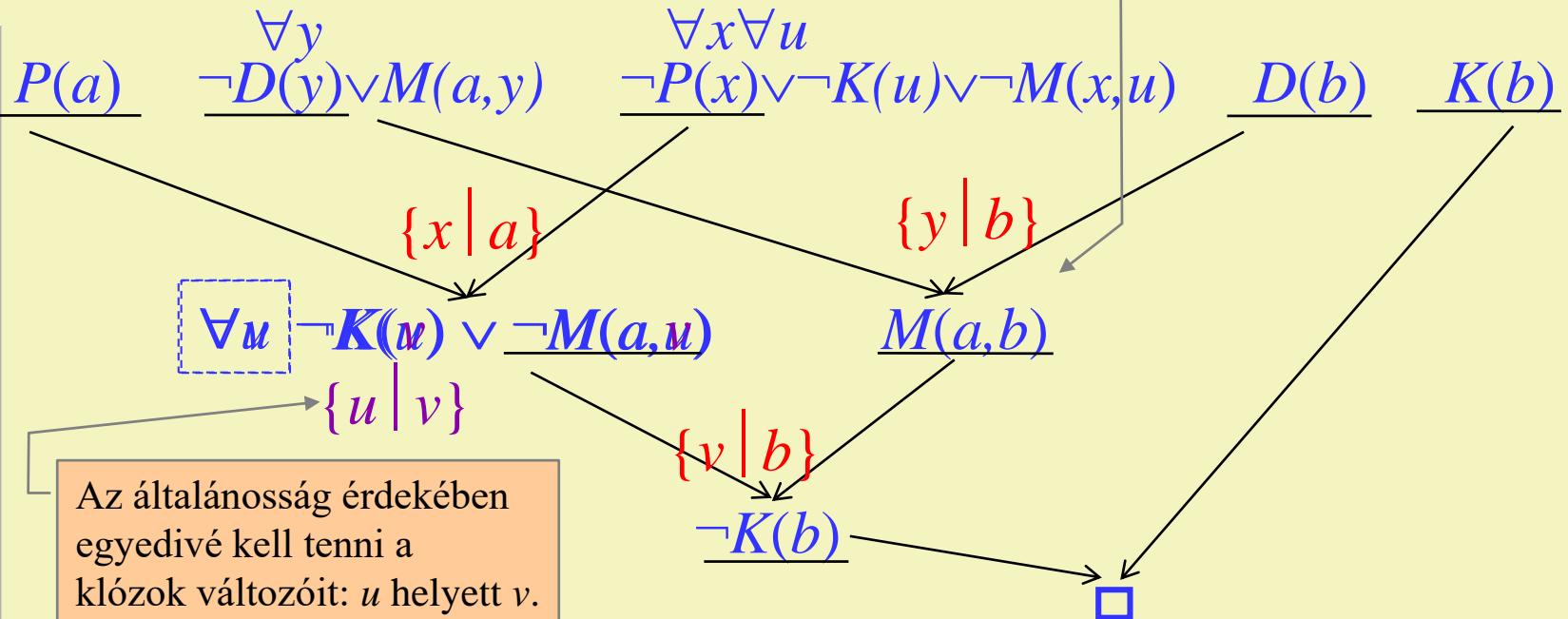
$D(b)$, $Q(b)$

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:

$C_1 = P(t_{11}, \dots, t_{1n}) \vee \dots \vee P(t_{r1}, \dots, t_{rn}) \vee C_1'$; $C_2 = \neg P(u_{11}, \dots, u_{1n}) \vee \dots \vee \neg P(u_{s1}, \dots, u_{sn}) \vee C_2'$

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

ha $P(t_{11}, \dots, t_{1n}), \dots, P(t_{r1}, \dots, t_{rn}), P(u_{11}, \dots, u_{1n}), \dots, P(u_{s1}, \dots, u_{sn})$ egyesíthetők a δ 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 klóz formája

1. $KLÓZOK := A_1, A_2, \dots, A_n$ és $\neg B$ formulák klózai
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

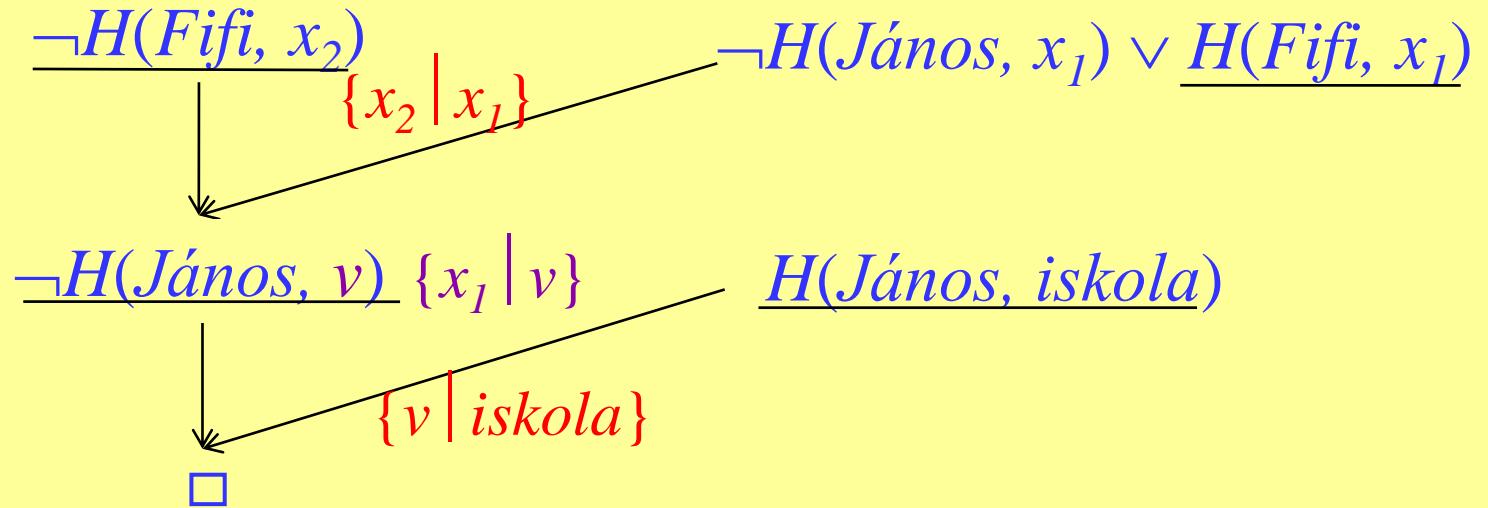
$\forall x[H(János,x) \rightarrow H(Fifi,x)]$

$H(János, \text{iskola})$

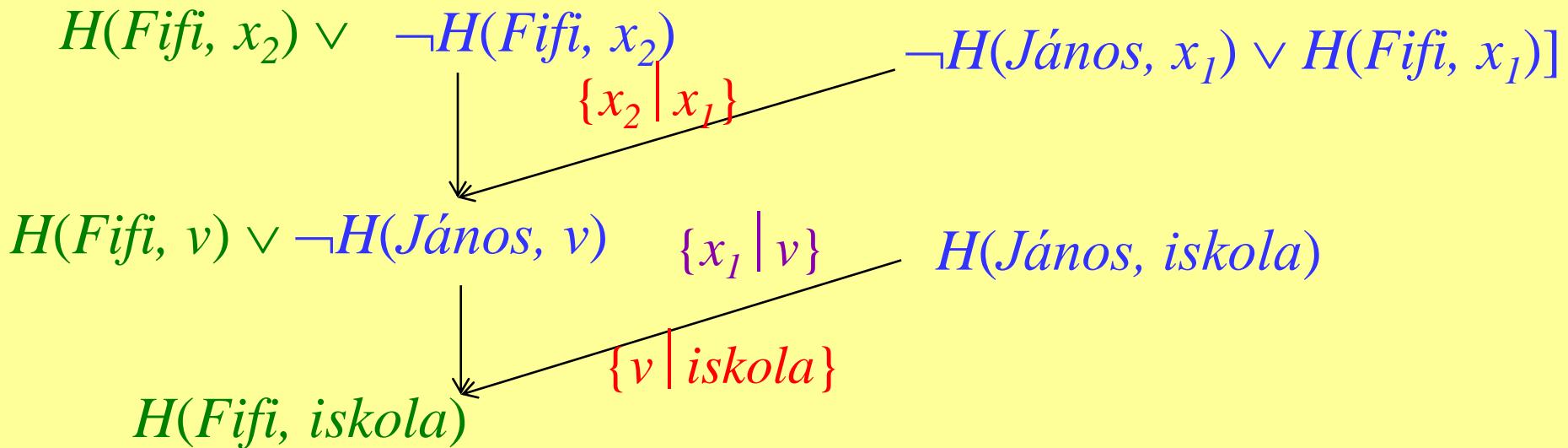
Először lássuk be, hogy létezik-e Fifi számára hely a világban?

$\exists xH(Fifi,x)$

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:

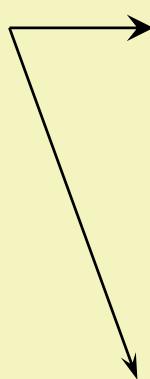
$$A, \ \neg C \vee \neg A, \ \neg A \vee B, \ B \vee D, \ \neg B$$

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



tények

konkrét ismeret
implikáció nélküli
formulákban

szabályok

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:

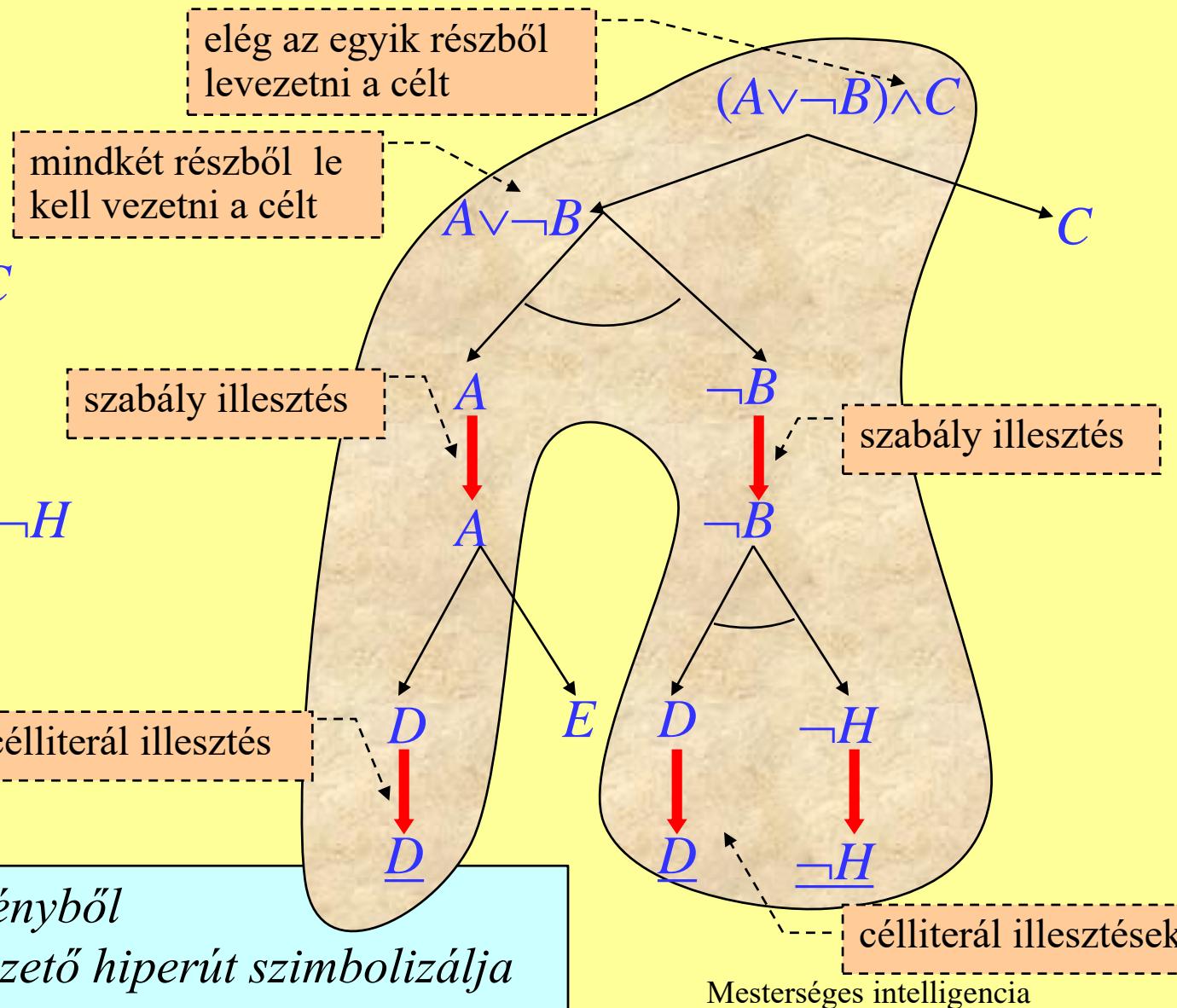
$$A \rightarrow D \wedge E$$

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

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:

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

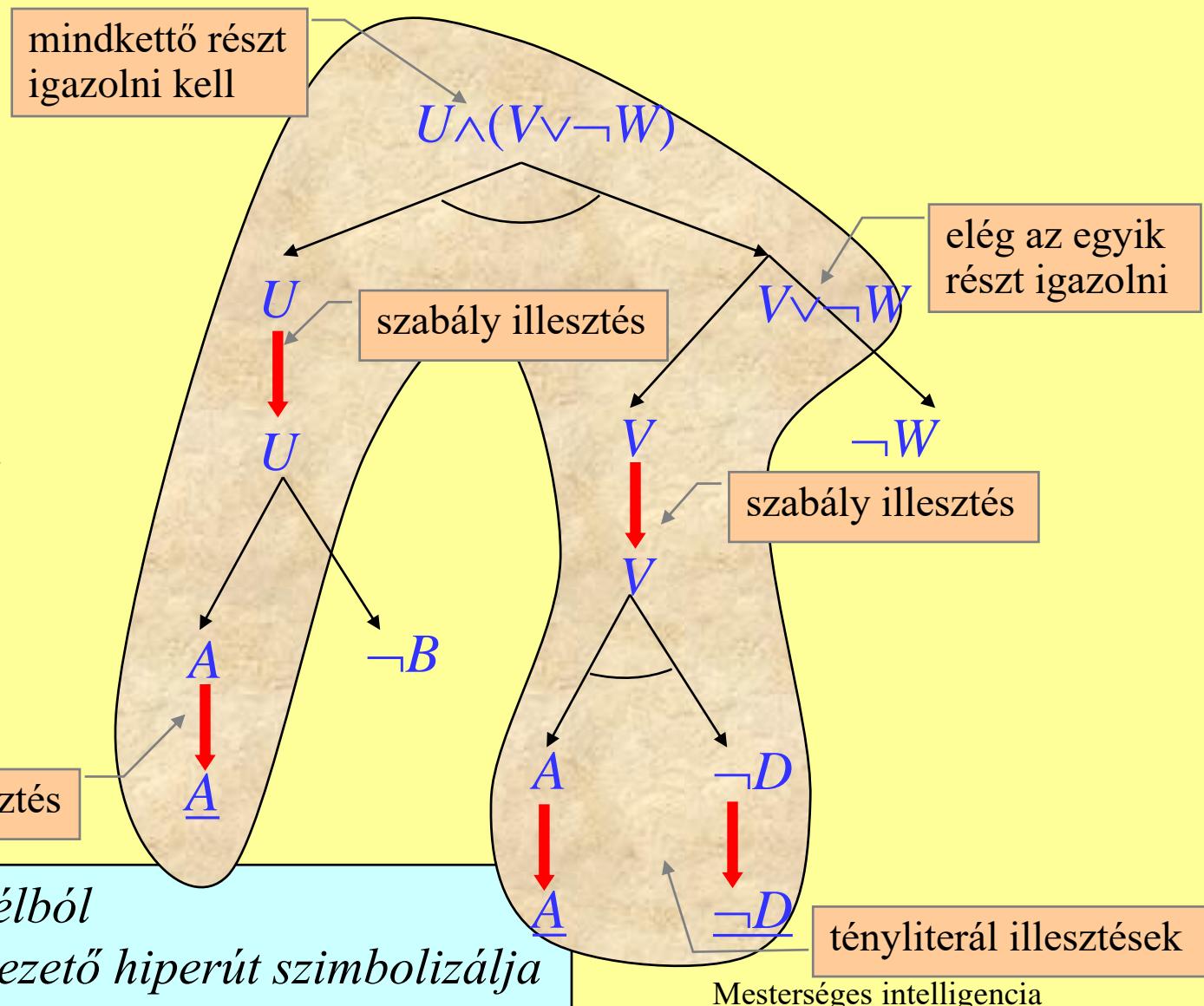
Szabályok:

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

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

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

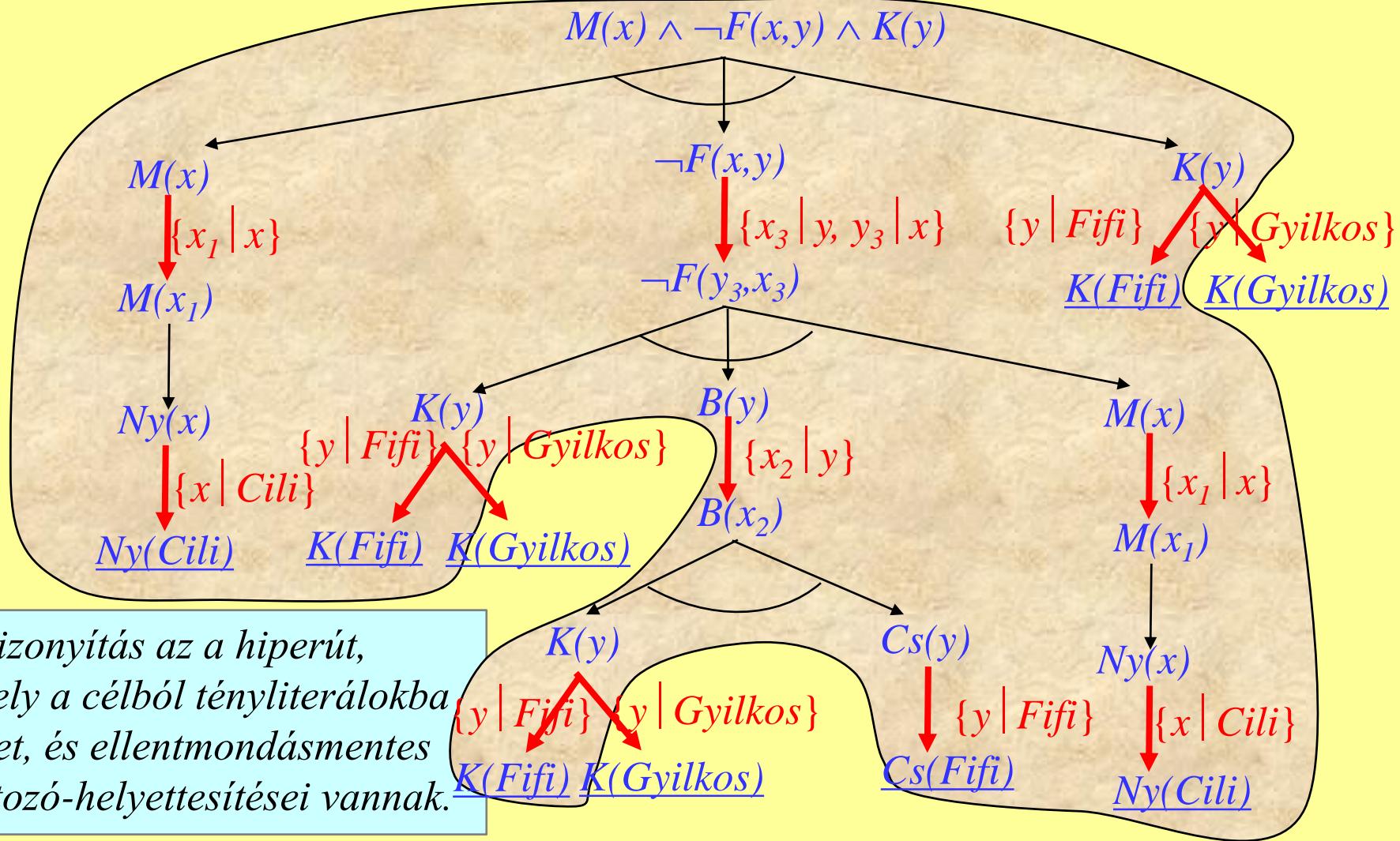
$K(x_2) \wedge Cs(x_2) \rightarrow B(x_2)$

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

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



Bizonytalanságkezelés

Bizonytalanság forrásai

- Hiányzó adat mellett történő következtetés
 - Mi lehet a páciens betegsége?

- **Bizonytalan adatra** épülő következtetés

objektív

Pontatlan műszerek pontatlan leolvasása: $80\text{ }^{\circ}\text{C} \pm 2\text{ }^{\circ}\text{C}$

szubjektív

- Következmény bizonytalansága:

- Mennyi az esélye, hogy egy sárga bőrű páciens hepatitiszes, ha ismerjük a *sárga bőrű hepatitiszesek / sárga bőrű betegek* arányát?

- Elmosódott jelentésű állítások:

- A nadrag erősen szennyezett

szubjektív

objektív

- Ellentmondó adatokból vagy ellentmondó következtetésekből származtatott következmény

Bizonytalanságkezelés alapkérdései

- Hogyan **reprezentáljuk** a bizonytalanságot?
 - Az ismeretekhez numerikus vagy szimbolikus értéket rendelünk
- Hogyan **kombináljuk** a bizonytalanságot?
 - A logikai műveletek mentén komponált összetett ismeret bizonytalanságát a komponensek bizonytalanságából számoljuk.
- Hogyan **következtessünk** bizonytalan információból?
 - Mennyire (milyen mértékben) bizonytalan az a következmény, amelyre bizonytalan ismeretekből indulva bizonytalan következtetési szabállyal következtünk?

1. Klasszikus valószínűség számítás

- A központi kérdés az, hogy egy bizonytalan $B \rightarrow A$ szabály alapján milyen bizonyossággal állítható az, hogy ha B igaz, akkor A is?
- Ugyanez másképpen is megfogalmazható: mi az A esemény bekövetkezésének valószínűsége, amikor a B esemény bekövetkezik.

Feltételes valószínűség:

$$p(A | B) = \frac{p(A \wedge B)}{p(B)} \quad \text{ha } p(B) > 0$$

Állítás = esemény

- A továbbiakban az állítások mindenkor egy esemény bekövetkezéséről szólnak majd, így azokat (diszkrét) valószínűségi változók segítségével fogalmazhatjuk meg.
 - $X_i=x_i$ esemény esetén
 - X_i a diszkrét valószínűségi változó,
 - x_i a változó értéke.
- Speciális jelölés:
 - Amikor az X_i értéke csak *igaz* vagy *hamis* lehet, akkor használjuk
 - $X_i=igaz$ helyett X_i ,
 - $X_i=hamis$ helyett $\neg X_i$

Megjegyzés

- ❑ Egy adott problémakör (eseményrendszer) összes feltételes valószínűségét az események együttes valószínűsségi eloszlásának ismeretében könnyen kiszámolhatjuk.
- ❑ De a gyakorlatban az együttes valószínűsségi eloszlás
 - többnyire nem ismert explicit módon
 - túl sok apriori adat tárolását igényelné (a memória igény exponenciálisan nő az elemi események számával növelésével)
- ❑ Ezért egy feltételes valószínűség közvetlen kiszámolásához különféle elkerülő technikákat alkalmazunk.

Bayes téTEL kÜLÖNFÉLE ALAKJAI

a) Klasszikus

$$p(B \mid A) = \frac{p(A \mid B) \cdot p(B)}{p(A)}$$

b) Háttértudás (*E*) mellett

$$p(B \mid A, E) = \frac{p(A \mid B, E) \cdot p(B \mid E)}{p(A \mid E)}$$

c) Általánosított (B_1, \dots, B_n teljes és független)

$$p(B_i \mid A) = \frac{p(A \mid B_i) \cdot p(B_i)}{\sum_k p(A \mid B_k) \cdot p(B_k)}$$

Szuvas-e egy fog, ha lyukas és fáj?

Russel-Norvig: AI

□ $p(\text{szuvas} \mid \text{lyukas}, \text{fáj}) = ?$

apriori ismeretek:

- $p(\text{szuvas})=0.65$
- $p(\text{fáj} \mid \text{szuvas})=0.5$
- $p(\text{fáj} \mid \neg \text{szuvas})=0.1$
- $p(\text{lyukas} \mid \text{szuvas})=0.95$
- $p(\text{lyukas} \mid \neg \text{szuvas})=0.01$

$$p(\text{szuvas} \mid \text{lyukas}, \text{fáj}) = ?$$

Példa folytatása (Bayes tételek alkalmazása)

- Ha a klasszikus Bayes tételt alkalmazzuk, akkor hamar elakadunk, mert csak a $p(\text{sz})$ -t ismerjük.

$$p(\text{sz} \mid \text{ly}, \text{f}) = \frac{p(\text{f}, \text{ly} \mid \text{sz}) \cdot p(\text{sz})}{p(\text{f}, \text{ly})}$$

- Keressünk más utat! (Bayes-i frissítés módszere)
 - Először a háttér tudás melletti Bayes tételt alkalmazzuk a fáj eseményre, mint háttértényre,
 - És az ehhez szükséges $p(\text{sz} \mid \text{f})$ -re a közönséges Bayes tételt írjuk fel.

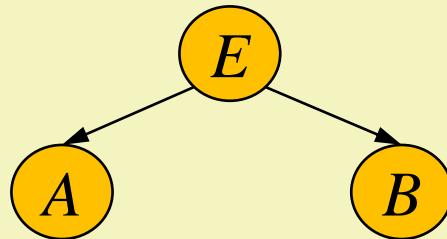
$$p(\text{sz} \mid \text{ly}, \text{f}) = \frac{p(\text{ly} \mid \text{sz}, \text{f}) \cdot p(\text{sz} \mid \text{f})}{p(\text{ly} \mid \text{f})} = \frac{p(\text{ly} \mid \text{f}, \text{sz}) \cdot p(\text{f} \mid \text{sz}) \cdot p(\text{sz})}{p(\text{ly} \mid \text{f}) \cdot p(\text{f})}$$

Feltételes függetlenség

- Közönséges függetlenség: $p(A,B) = p(A) \cdot p(B)$
- Az A és a B események feltételesen függetlenek az E eseményre nézve (nincs közöttük közvetlen függőségi kapcsolat, csak az E -n keresztül), ha
$$p(A,B | E) = p(A | E) \cdot p(B | E)$$
- Az A és a B feltételesen függetlenek az E -re nézve, akkor
$$p(A | B, E) = p(A | E) \quad \text{illetve} \quad p(B | A, E) = p(B | E)$$

Feltételes függetlenség esetei

- Az A és a B feltételesen függetlenek az E -re nézve:
 - A is, B is függ az E -től, de más kapcsolat nincs köztük
 - A -tól függ az E , és E -től függ a B , de más kapcsolat nincs köztük
 - B -től függ az E , és E -től függ a A , de más kapcsolat nincs köztük



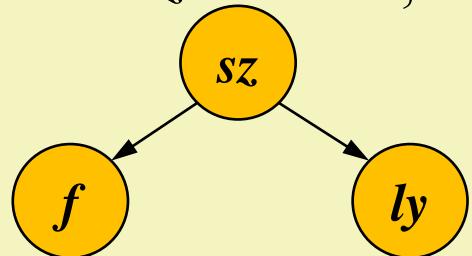
$$p(\text{szuvas} \mid \text{lyukas}, \text{fáj}) = ?$$

Példa folytatása (feltételes függetlenség kihasználása)

- A lyukas fogat és a fogfájást a szuvasodás kapcsolja össze, mindenki következménye a szuvasodásnak, ettől eltekintve függetlenek: Az *ly* **feltételesen független** az *f*-től az *sz*-re nézve, azaz $p(\text{ly} \mid f, \text{sz}) = p(\text{ly} \mid \text{sz})$

- Hozzáolvasva ezt az eddigiekhez:

$$p(\text{sz} \mid \text{ly}, \text{f}) = \frac{p(\text{ly} \mid \text{sz}) \cdot p(\text{f} \mid \text{sz}) \cdot p(\text{sz})}{p(\text{ly} \mid \text{f}) \cdot p(\text{f})}$$



- Már csak a nevezőbeli valószínűségeket nem ismerjük. Az apriori tudásunk alapján a számlálóbeli valószínűségeket akkor is ismernénk, ha ott az *sz* helyére $\neg\text{sz}$ -t írnánk. Ilyenkor alkalmazhatjuk a **normalizálás** technikáját.

Normalizálás

- Amikor egy eseménynek és az ellentetjének a valószínűségét ugyanazon, de ismeretlen együtthatóval számoljuk ki más valószínűségekből:

$$- \quad p(A) = \alpha \cdot u \qquad \qquad \qquad \text{és} \qquad \qquad p(\neg A) = \alpha \cdot v$$

- ❑ akkor az együttható könnyen meghatározható:

$$I = p(A) + p(\neg A) = \alpha \cdot [u + v]$$

$$\alpha = l/[u + v]$$

$$p(\text{szuvas} \mid \text{lyukas}, \text{fáj}) = ?$$

Példa folytatása (normalizálás)

- ugyanaz sz-re és $\neg\text{sz}$ -re:

$$p(\text{sz} \mid \text{ly}, \text{f}) = \frac{p(\text{ly} \mid \text{sz}) \cdot p(\text{f} \mid \text{sz}) \cdot p(\text{sz})}{p(\text{ly} \mid \text{f}) \cdot p(\text{f})} = \alpha \cdot p(\text{ly} \mid \text{sz}) \cdot p(\text{f} \mid \text{sz}) \cdot p(\text{sz})$$

$$p(\neg\text{sz} \mid \text{ly}, \text{f}) = \frac{p(\text{ly} \mid \neg\text{sz}) \cdot p(\text{f} \mid \neg\text{sz}) \cdot p(\neg\text{sz})}{p(\text{ly} \mid \text{f}) \cdot p(\text{f})} = \alpha \cdot p(\text{ly} \mid \neg\text{sz}) \cdot p(\text{f} \mid \neg\text{sz}) \cdot p(\neg\text{sz})$$

- összeg:

$$1 = \alpha \cdot [p(\text{ly} \mid \text{sz}) \cdot p(\text{f} \mid \text{sz}) \cdot p(\text{sz}) + p(\text{ly} \mid \neg\text{sz}) \cdot p(\text{f} \mid \neg\text{sz}) \cdot p(\neg\text{sz})]$$

- együttható:

$$\alpha = 1 / [p(\text{ly} \mid \text{sz}) \cdot p(\text{f} \mid \text{sz}) \cdot p(\text{sz}) + p(\text{ly} \mid \neg\text{sz}) \cdot p(\text{f} \mid \neg\text{sz}) \cdot p(\neg\text{sz})]$$

$$p(\text{szuvas} \mid \text{lyukas}, \text{fáj}) = ?$$

Példa befejezése

apriori ismeretek:

$$p(\text{ly} \mid \text{sz}) = 0.7$$

$$p(\text{ly} \mid \neg\text{sz}) = 0.01$$

$$p(\text{f} \mid \text{sz}) = 0.5$$

$$p(\text{f} \mid \neg\text{sz}) = 0.1$$

$$p(\text{sz}) = 0.65$$

Bayes-i frissítés és a feltételes függetlenség felhasználása miatt:

$$p(\text{sz} \mid \text{ly}, \text{f}) = \alpha \cdot p(\text{ly} \mid \text{sz}) \cdot p(\text{f} \mid \text{sz}) \cdot p(\text{sz}) = \alpha \cdot 0.7 \cdot 0.5 \cdot 0.65$$

normalizálás:

$$\begin{aligned}\alpha &= 1 / [p(\text{ly} \mid \text{sz}) \cdot p(\text{f} \mid \text{sz}) \cdot p(\text{sz}) + p(\text{ly} \mid \neg\text{sz}) \cdot p(\text{f} \mid \neg\text{sz}) \cdot p(\neg\text{sz})] \\ &= 1 / [0.7 \cdot 0.5 \cdot 0.65 + 0.01 \cdot 0.1 \cdot 0.35] = 4.38885\end{aligned}$$

eredmény:

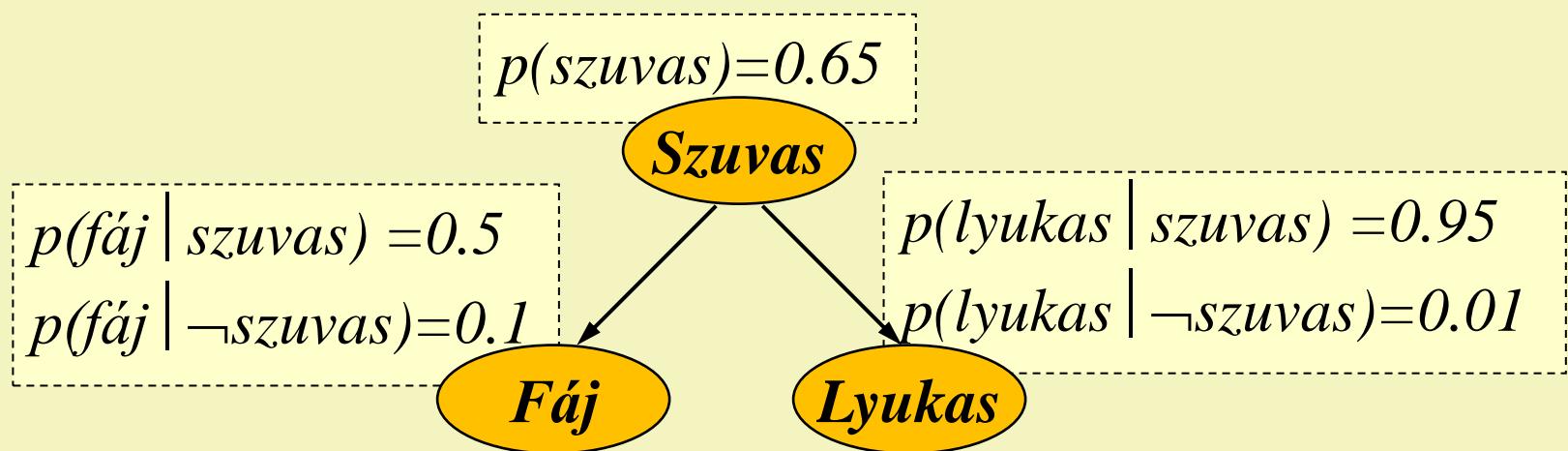
$$p(\text{sz} \mid \text{ly}, \text{f}) = 4.38885 \cdot 0.2275 = 0.99846$$

Bayes modell értékelése

- ❑ Az apriori valószínűségekhez nehéz hozzájutni.
- ❑ Még a bevetett trükkök ellenére is sok apriori valószínűséget kell beszerezni és tárolni hozzá.
- ❑ A következtetés túl ötletszerűnek tűnik, nehéz algoritmizálni.
- ❑ Matematikailag jól megalapozott, de igen számításigényes, és magyarázatadásra nem alkalmas.
- ❑ A modell új ismeretekkel nehezen bővíthető. Nem elég ugyanis egy új esemény és a vele kapcsolatos feltételes események valószínűségeit megadni, ilyenkor a korábbi valószínűségi értékeket is felül kell bírálni.

2. Bayes (valószínűségi) hálók

- Az előző példa megoldásánál alkalmazott módszert általánosíthatnánk, ha a minimálisan szükséges apriori valószínűségeket úgy tárolnánk (**tömör reprezentáció**), hogy a feltételes függetlenségek felismerése egyértelmű és automatizálható legyen.

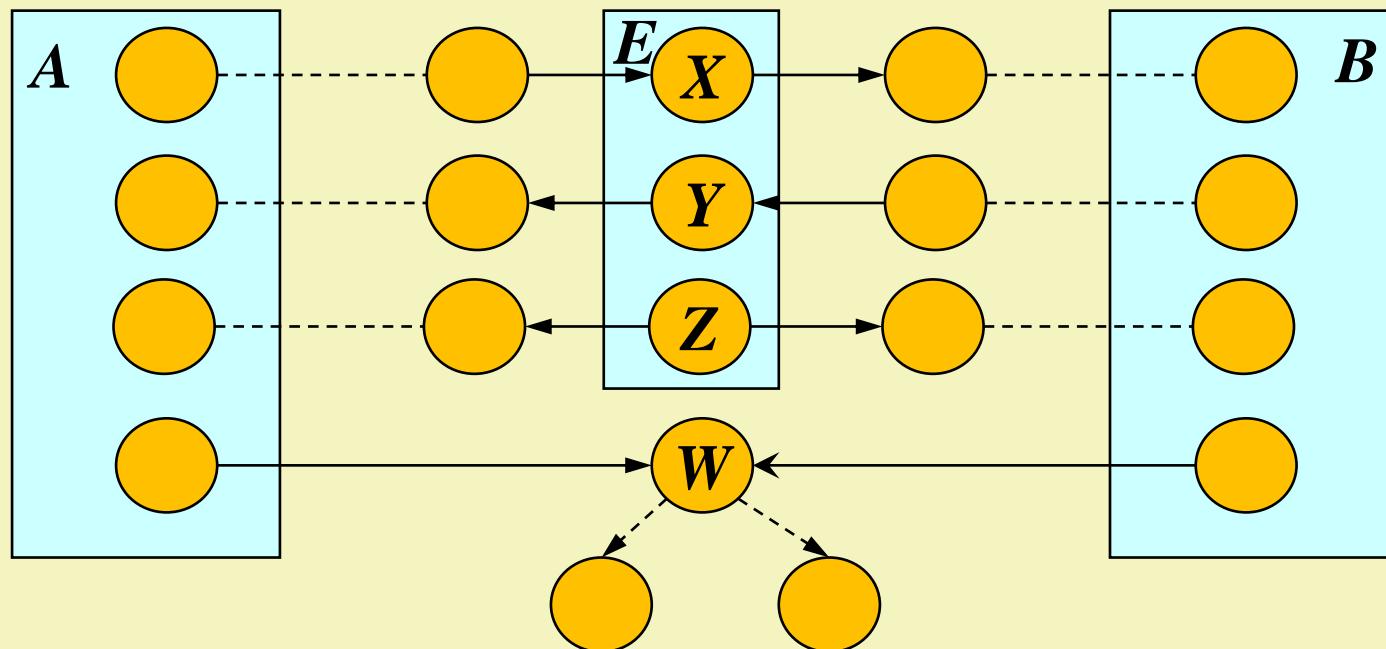


Reprezentáció Bayes hálóval

- Tekintsük a tárgyprobléma valószínűségi változóit.
- Feleltessük meg a változókat egy **körmentes irányított gráf** csúcsainak.
- Ábrázoljuk az irányított élekkel a változók közötti közvetlen **ok-okozati összefüggéseket** (ez által implicit módon rögzítjük a feltételes függetlenségeket is).
- Adjuk meg az csúcsok **feltételes valószínűségi tábláit** (FVT):
 $p(X_i=x_i \mid \text{szülő}(X_i)=x_{i1}, \dots, x_{ik})$
ahol a $\text{szülő}(X_i)$ az X_i változó csúcsának szülőcsúcsaihoz rendelt X_{i1}, \dots, X_{ik} változók együttesét jelöli.

Feltételes függetlenség felismerése Bayes hálóban

- ❑ Legyenek A , B és E összetett (több csúcs) események.
- ❑ Az A és B feltételesen független az E -re nézve, ha minden A és B -beli csúcs közti irányítatlan útvonalra az alábbi 4 eset valamelyike teljesül:



Bayes háló kifejező ereje

- Az együttes valószínűségi eloszlás (a lánc-szabály alapján)

$$\begin{aligned} p(X_1=x_1, \dots, X_n=x_n) &= \\ &= p(X_n=x_n \mid X_1=x_1, \dots, X_{n-1}=x_{n-1}) \cdot p(X_1=x_1, \dots, X_{n-1}=x_{n-1}) = \\ &= \dots = \prod_{i=1 \dots n} p(X_i=x_i \mid X_1=x_1, \dots, X_{i-1}=x_{i-1}) \end{aligned}$$

- Sorszámozzuk meg úgy a változókat, hogy ha $i>j$, akkor X_i -ből ne vezessen irányított út X_j -be: ekkor $\forall i: szüлő(X_i) \subseteq \{X_1, \dots, X_{i-1}\}$, és ekkor a feltételes függetlenség miatt

$$p(X_i=x_i \mid X_1=x_1, \dots, X_{i-1}=x_{i-1}) = p(X_i \mid szüлő(X_i)=x_{i1}, \dots, x_{ik})$$

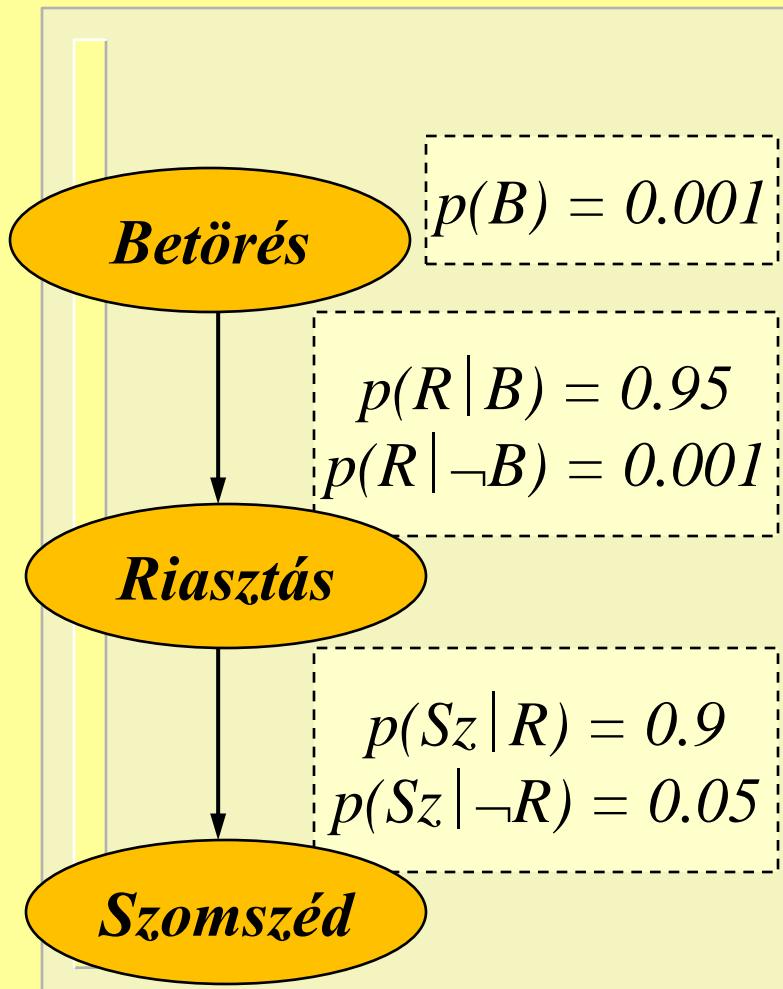
- Az adott tárgykör együttes valószínűségi eloszlása tehát a Bayes háló FVT-iból közvetlenül megkapható.

$$p(X_1=x_1, \dots, X_n=x_n) = \prod_{i=1 \dots n} p(X_i=x_i \mid szüлő(X_i)=x_{i1}, \dots, x_{ik})$$

Bayes hálók tervezése

- ❑ Határozzuk meg a tárgytartományt leíró változók halmazát, majd meghatározott sorrendben dolgozzuk fel őket:
 1. Válasszunk ki olyat, amely kizárolag a már hálóhoz csatolt változóktól függ, és új csúcsként vegyük fel azt a hálóba
 2. A hálóbeli változóknak vegyük azt a minimális halmazát, amelyek közvetlenül hatnak az új változóra. Rajzoljuk be ezeket a függőségeket reprezentáló éleket.
 3. Töltsük ki az új csúcs FVT-jét.
 4. GOTO 1.

*A szomszédunk telefonált, hogy szól a betörés-riasztónk a lakásunkban.
Betörtek volna hozzáink? Russel-Norvig: AI*



$$\begin{aligned}
 p(B | Sz) &= p(Sz | B) \cdot p(B) / p(Sz) && \text{Bayes tétel} \\
 &= \alpha \cdot p(Sz | B) \cdot p(B) && \text{normalizálás} \\
 &= \alpha \cdot [p(Sz, R | B) + p(Sz, \neg R | B)] \cdot p(B) && \text{telj fgl rsz} \\
 &= \alpha \cdot [p(Sz | R, B) \cdot p(R | B) + p(Sz | \neg R, B) \cdot p(\neg R | B)] \cdot p(B) && \text{lánc szabály} \\
 &= \alpha \cdot [p(Sz | R) \cdot p(R | B) + p(Sz | \neg R) \cdot p(\neg R | B)] \cdot p(B) && \text{felt. fgl.} \\
 &= \alpha \cdot 0.0008575 \\
 p(\neg B | Sz) &= \alpha \cdot 0.0507991 \\
 \alpha &= 19.3585 && \text{normalizálás vége} \\
 p(B | Sz) &= 0.0166
 \end{aligned}$$

Következtetés Bayes hálókban

- Célja egy feltételes valószínűség meghatározása a Bayes módszerre alapuló számítással (Bayes tételek, normalizálás, felbontás teljes fgl. eseményrendszerre, lánc-szabály, feltételes fgl.)
- Egy feltételes valószínűség kiszámolására egy (rekurzív) algoritmus készíthető, amelynek számításigénye erősen függ a háló bonyolultságától.
- Egyszeresen kötött hálókra ([fa-gráfokra](#)), ahol az irányítást figyelmen kívül hagyva két csúcs között nincsenek alternatív irányítatlan útvonalak, van lineáris futási idejű algoritmus.
- Többszörösen kötött hálók esetén különféle redukáló módszereket alkalmazhatunk.

*Példa kétszeresen kötött
Bayes hálóra
Russel-Norvig: AI*

$L=$	i	h
$E\bar{E}=i$	0.0	1.0
$E\bar{E}=h$	0.9	0.1

Esős évszak

$E=$	i	h
$\bar{E}\bar{E}=i$	0.8	0.2
$\bar{E}\bar{E}=h$	0.1	0.9

Eső

$VP=$	i	h
$L+E=ii$	0.95	0.05
$L+E=ih$	0.9	0.1
$L+E=hi$	0.8	0.2
$L+E=hh$	0.1	0.9

Vizes pázsit

Következtetés többszörösen kötött hálókban

Összevonásos eljárások

- Változók (csúcsok) összevonásával fa-gráfot kapunk, amelyben meg kell határozni az összevont csúcsok FVT-it.

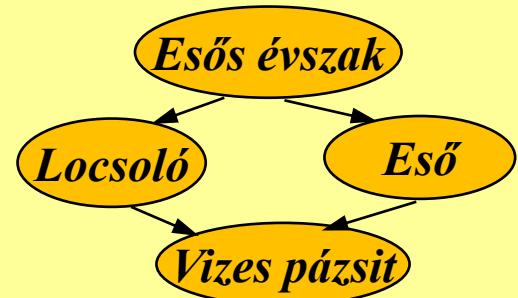
Vágóhalmaz feltételezésen alapuló eljárások

- Változók (csúcsok) elhagyásával annyi azonos szerkezetű fa-gráfot kapunk, ahányféléképpen az elhagyott változók értékét rögzíthetjük. Egy-egy fa-gráf súlya az a valószínűség, amely mellett az elhagyott változók a fa-gráfban rögzített értékeiket felveszik. A fa-gráfok FVT-it újra kell számolni. A válasz az egyes (esetleg csak a valószínűbb) fa-gráfokból kiszámolt eredmények súlyozott átlaga lesz.

Sztochasztikus szimulációs eljárások

- A háló valószínűségi értékekeit figyelembe véve példákat generálunk. A válasz a jó példáknak az összes példához vett relatív gyakorisága.

a) Összevonás



$E\bar{E}=i$	0.5
$E\acute{E}=h$	0.5

Esős évszak

$L=$	i	h
$E\acute{E}=i$	0.0	1.0
$E\acute{E}=h$	0.9	0.1

$L+E=$	ii	ih	hi	hh
$E\acute{E}=i$	0	0	0.8	0.2
$E\acute{E}=h$	0.09	0.81	0.01	0.09

Locsoló+Eső

$E=$	i	h
$E\acute{E}=i$	0.8	0.2
$E\acute{E}=h$	0.1	0.9

$VP=$	i	h
$L+E=ii$	0.95	0.05
$L+E=ih$	0.9	0.1
$L+E=hi$	0.8	0.2
$L+E=hh$	0.1	0.9

Vizes pázsit

Esős évszak

Locsoló

Eső

Vizes pázsit

b) Vágóhalmaz feltételezés

$E\acute{E}=i$	0.5
$E\acute{E}=h$	0.5

Esős évszak

$L=i$	0.0
$L=h$	1.0

igen
0.5

$E=i$	0.8
$E=h$	0.2

$L=i$	0.9
$L=h$	0.1

$E=i$	0.1
$E=h$	0.9

Locsoló

Eső

Locsoló

Eső

Vizes pázsit

$L=$	i	h
$E\acute{E}=i$	0.0	1.0
$E\acute{E}=h$	0.9	0.1

$VP=$	i	h
$L+E=ii$	0.95	0.05
$L+E=ih$	0.9	0.1
$L+E=hi$	0.8	0.2
$L+E=hh$	0.1	0.9

$E=$	i	h
$E\acute{E}=i$	0.8	0.2
$E\acute{E}=h$	0.1	0.9

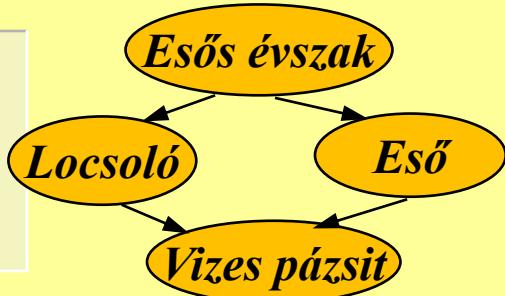
Adott pontosságú vágóhalmaz feltételezés

- ❑ Nem szükséges az összes fa-gráfra kiszámolni a keresett feltételes valószínűséget.
- ❑ Sokszor elég csak a legvalószínűbb hálókra súlyozott átlagot számolni, mert már ez is jól közelítheti a pontos választ.
 - A számolási hiba a ki nem értékelt hálók valószínűségeinek összege.

c) Sztochasztikus szimuláció

$$p(A \mid B) = \frac{p(A, B)}{p(B)} = \frac{\text{Jó hasznos példák száma}}{\text{Összes hasznos példa száma}}$$

- A példa hasznos, ha a feltételt (B) teljesíti
- Annak érdekében, hogy csak hasznos példát generáljunk, a feltételt (B) alkotó tény változók értékét rögzítjük, de az így generált példát azzal a valószínűsséggel súlyozzuk, amely mellett ezek a tény változók a számukra kijelölt értékeket felveszik. A relatív gyakoriságot a példák így súlyozott darabszáma alapján számítjuk.



Egy hasznos példa előállítása a $p(Vizes\ pázsit \mid Eső)$ számára

- $E\acute{E}=Random(0.5)$ mert $p(E\acute{E})=0.5$
 - TF: $E\acute{E}=hamis.$
- $L=Random(0.9)$ mert $p(L \mid \neg E\acute{E})=0.9$
 - TF: $L=igaz.$
- E tényváltozó, értéke igaz, és $p(E \mid \neg E\acute{E})=0.2$
 - Ezért $E=igaz (0.2)$
 - Ez garantálja a példa hasznosságát
 - Ez a példa súlya
- $VP=Random(0.95)$ mert $p(VP \mid E,L)=0.95$
 - TF: $VP=igaz.$
 - Ez tehát egy jó hasznos példa

Bayes hálók tanulása

- ❑ Adott háló-struktúrában az **FVT tanulása** példákból nyert relatív gyakorisági értékek számolásával valósítható meg.
 - Probléma: ha a példák hiányosak, azaz nem ismerjük, hogy egy példában bizonyos változó milyen értéket vesz fel.
- ❑ A **háló szerkezetének tanulása** során metrikát definiálunk a feladat és az azt leíró háló „távolságára” és ez alapján keressük a legjobban illeszkedő struktúrát.

Bayes hálók értékelése

- ❑ Kevesebb a priori valószínűséget kell benne tárolni ahhoz képest, ha az együttes valószínűségi eloszlásfüggvényt akarnánk ábrázolni.
- ❑ Egyszerűen bővíthető anélkül, hogy eddigi valószínűségeket újra kellene gondolni.
- ❑ A következtetés felhasználható magyarázatadásra.
- ❑ Matematikailag jól megalapozott, de – az erőfeszítéseink ellenére is – igen számításigényes.

3. Heurisztikus technikák

- „Betörés-riasztó-szomszéd” probléma:
 - szabályok:
 - ha a szomszéd hallani véli a riasztót akkor szól a riasztónk*
 $Sz \rightarrow R \text{ (0.9)}$
 - ha szól a riasztónk akkor betörtek hozzáink*
 $R \rightarrow B \text{ (0.95)}$
 - tény: *a szomszéd telefonál, hogy hallja a riasztót*
 Sz
- Betörtek-e hozzáink?
 - $Sz, Sz \rightarrow R \Rightarrow R ; R, R \rightarrow B \Rightarrow B$
 - Új következtetési elv: $T(p), T \rightarrow K(q) \Rightarrow K(p \cdot q)$
 - $Sz(1) \Rightarrow R(0.9) \Rightarrow B(0.855)$

Ismert heurisztikus technológiák

- ❑ MYCIN bizonytalanság kezelési technikája
- ❑ Dempster-Shafer elmélet
- ❑ Fuzzy következtetés



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 tanulással meg lehet adni a feladat
 - modelljét (logikai formulák, valószínűségi hálók)
 - megoldó algoritmusát (genetikus programozás, mély hálók)
 - heurisztikáját (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 $L(\Theta)$ hiba már elég kicsi (ettől reméljük, hogy $f(\Theta, x) \approx \varphi(x)$)

$$L(\Theta) = \frac{1}{N} \sum_{n=1}^N \ell \left(\underbrace{f(\Theta, x_n)}_{t_n}, \underbrace{y_n}_{\text{számított kimenet}} \right) \quad \begin{array}{|c|} \hline \text{elvárt kimenet} \\ \hline \end{array}$$

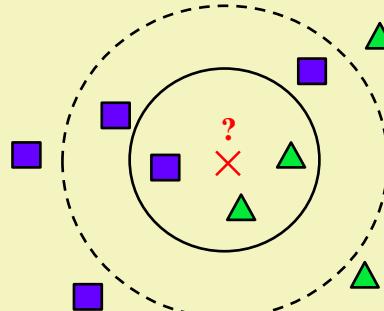
- $\ell : Y \times Y \rightarrow \mathbb{R}$ hibafüggvény (ha Y m dimenziós)
 - $\ell(t_n, y_n)$ lehet például $\|t_n - y_n\|_1$, $\|t_n - y_n\|_2^2$, vagy $-\sum_{j=1..m} y_{nj} \cdot \log t_{nj}$.

Megjegyzés

- ❑ Fontos, hogy az $f(\Theta, x)$ kiszámítása gyors legyen; nem baj, ha a megfelelő Θ megtalálása lassú, hiszen ezt a tanító minták segítségével előre számoljuk ki.
- ❑ A Θ 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 ℓ megfelelőek (ehhez tapasztalat, sok próbálkozás kell)
 - Θ közel esik a paraméter globális optimumához
- ❑ A Θ megtalálása egy nem-konvex optimalizálási feladat: a Θ globális optimumának megtalálása egy 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

- ❑ Egy $x \in X$ bemenethez annak a K darab mintának az outputja alapján számol kimenetet (pl. átlagolással, vagy többségi szavazással), amely minták inputja a legközelebb esik az x -hez.



A módszer és értékelése

sort(x, P, K) : a $P = \{x_n \mid n=1..N\}$ tanító minták bemeneteiből képzett, az x -től vett távolság alapján növekvő sorozat első K eleme.

$$f(\Theta, x) = \sum_{n=1..N} \frac{\mathbb{I}(x_n \in sort(x, P, K))}{K} \cdot y_n$$

ha egy állítás igaz, akkor 1-et ad, különben 0-t

$$f(\Theta, x) = \arg \max_{n=1..N} \frac{\sum_{\substack{x_i \in sort(x, P, K) \\ \varphi(x_i) = y_n}} 1}{\varphi(x_i) = y_n}$$

- a Θ paraméter a minták és a $K \in \mathbb{N}$ szám együttese
- 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 költséges;

az f kiszámítása, azaz a minták sorba rendezése erőforrásigényes

1.2. Döntési fa

- Tegyük fel, hogy az $x \in X$ bemeneteknek ugyanazon tulajdonságait (adott attribútumainak értékeit) ismerjük, azaz egy bemenetet **attribútum-érték párok halmazával jellemezhetünk**.
- 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
 - **levelei egy-egy lehetséges kimeneti értéket** mutatnak
- Egy x bemenet az attribútum-érték párai alapján egyértelműen leképezhető a döntési fa egyik levelére, amelyik a bemenethez tartozó kimenetet adja meg.

Példa: Elfogadjuk-e a megajánlott vizsgajegyet?

❑ Minták:

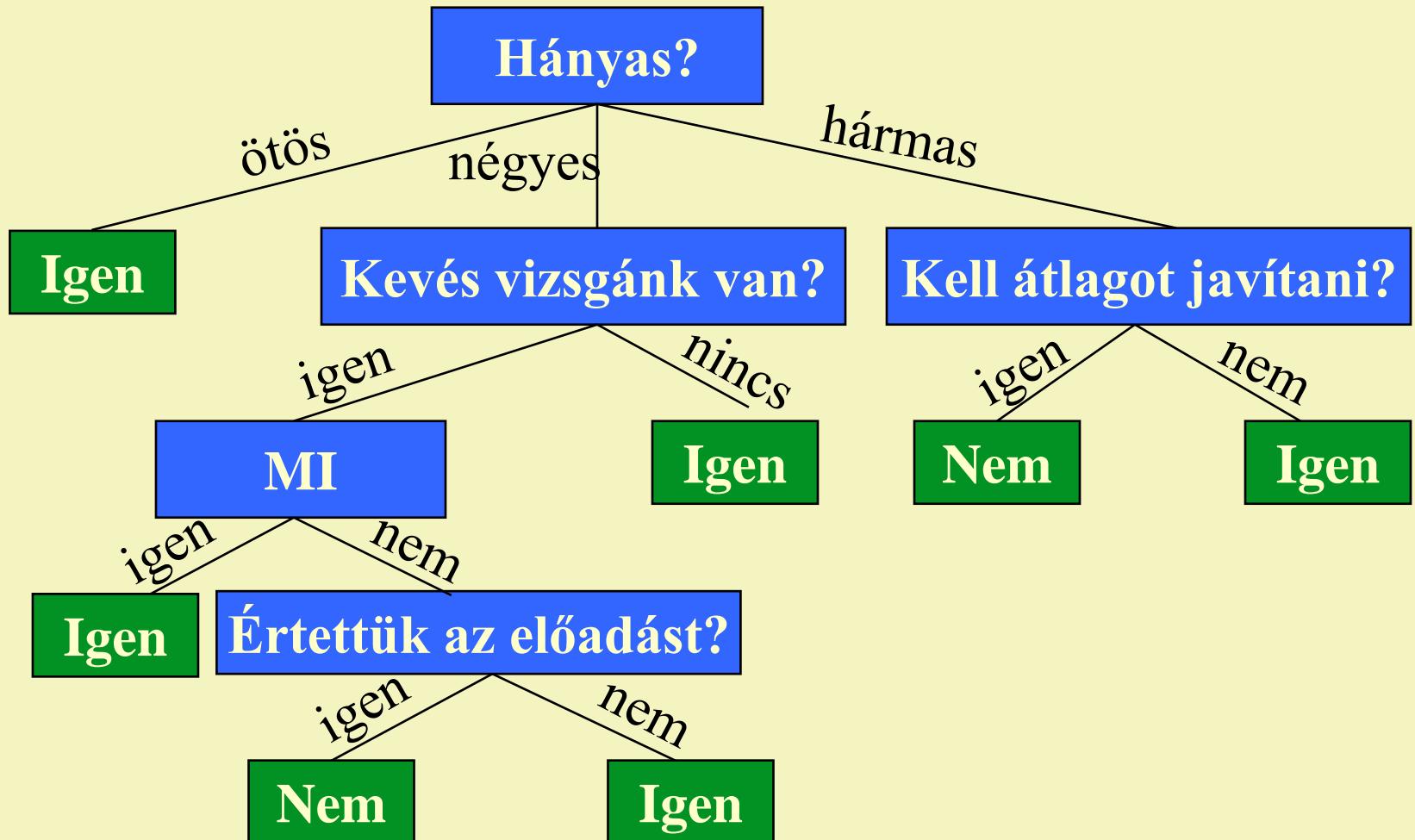
- Ha az ötös, akkor feltétlenül.
- Ha négyes és kevés vizsgánk van és értettük az előadást, akkor nem; feltéve, hogy a tárgy nem a Mesterséges intelligencia.
- Ha hármas és az átlagot kell javítanunk, akkor nem.



Attribútumok és lehetséges értékeik:

- hányast ajánlottak meg (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)

Példa: Elfogadjuk-e a megajánlott vizsgajegyet?



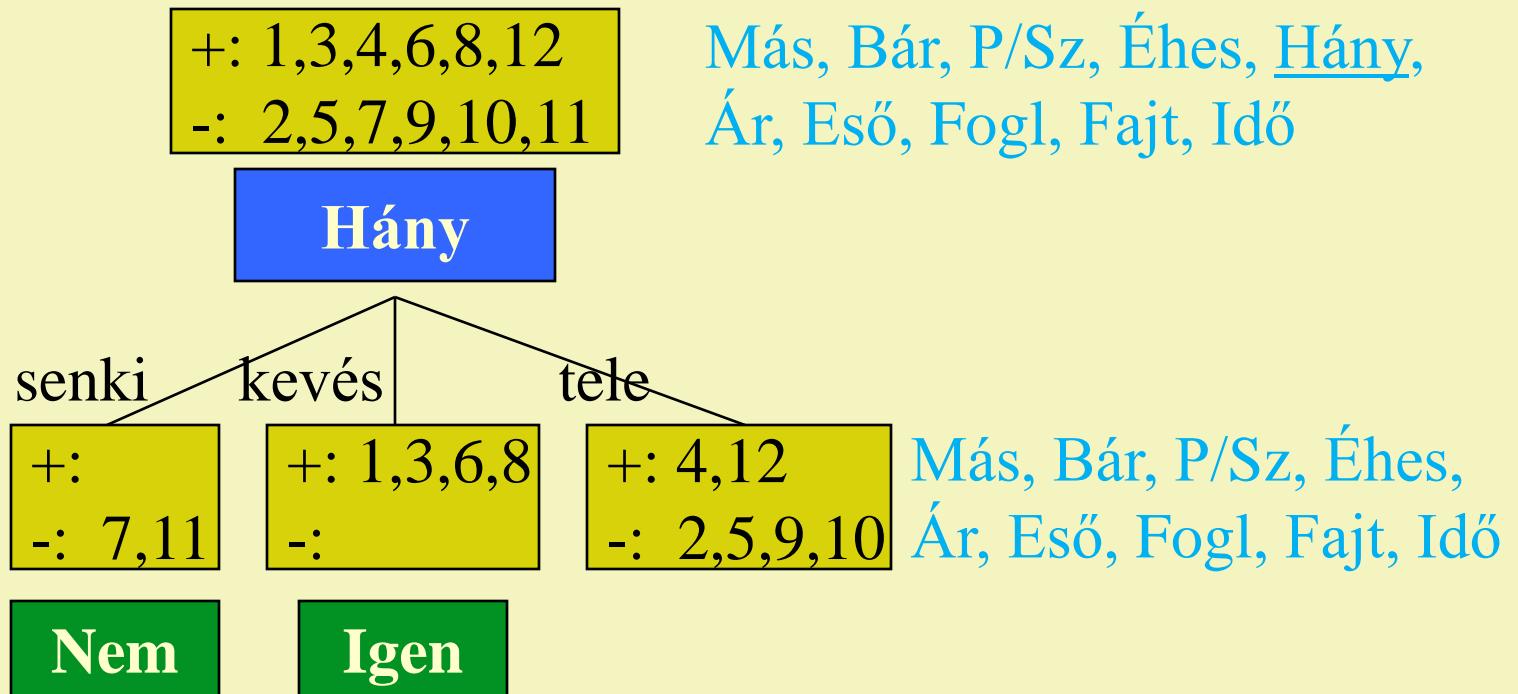
Döntési fa építése

- A döntési fát tanító minták segítségével építjük fel. Ennek során
 - minden **csúcshoz hozzárendeljük azon tanító mintákat**, amelyek a csúcshoz vezető ág attribútum-érték párlával rendelkeznek.
 - egy csúcs úgy válik **belső csúccsá**, hogy egy – a hozzávezető ágon még nem szereplő – attribútummal címkézzük fel.
 - egy csúcsot véglegesen **levélcsúcsnak** nyilváníthatunk, ha
 - nincsenek tanító mintái, vagy mind hasonló kimenetű, vagy a csúcshoz vezető ágon már minden attribútum szerepel.
 - a **levélcsúcs értéke** a hozzá tartozó tanító minták kimeneteinek átlaga vagy a leggyakoribb kimenete lesz.
(Ha nem tartoznak minták a levélcsúcshoz, vagy nem egyértelmű, melyik a leggyakoribb kimenet, akkor a szülőcsúcsának mintái alapján számolunk.)

Étterem probléma (Russel-Norvig)

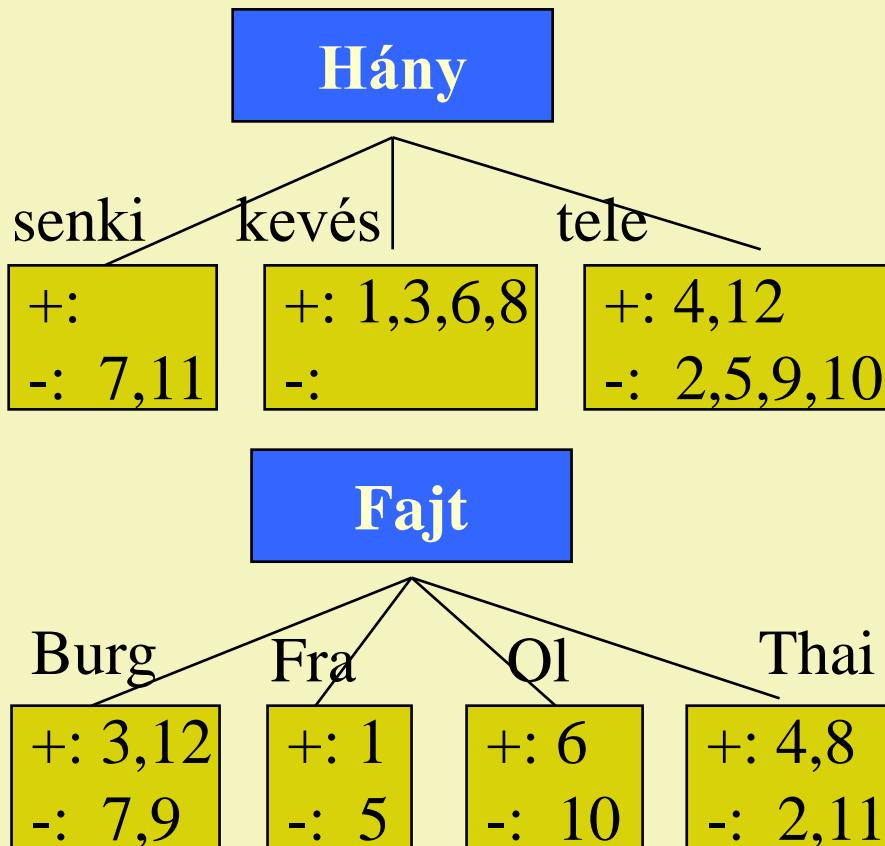
Pl.	Más	Bár	P/Sz	Éhes	Hány	Ár	Eső	Fogl	Fajt	Idő	Marad
1	I	N	N	I	kevés	drá	N	I	Fra	10	I
2	I	N	N	I	tele	olcs	N	N	Tha	60	N
3	N	I	N	N	kevés	olcs	N	N	Bur	10	I
4	I	N	I	I	tele	olcs	N	N	Tha	30	I
5	I	N	I	N	tele	drá	N	I	Fra	sok	N
6	N	I	N	I	kevés	köz	I	I	Ol	10	I
7	N	I	N	N	senki	olcs	I	N	Bur	10	N
8	N	N	N	I	kevés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



Ugyanazon problémához több döntési fa is megadható.
A lehető legkisebb döntési fa megtalálása egy NP-teljes probléma.

Alternatív lépések



A fa építésének első lépésében a Hány attribútum választása tűnik a legjobbnak, mert ekkor csak egy olyan új csúcs lesz, amely még nem levél, és annak mintái nem fele-fele arányban igen-nem értékek.

Heurisztika

- Egy döntési fa annál kisebb (annál hamarabb lehet benne egy tetszőleges bemenethez illeszkedő levélcsúcsot találni), minél rövidebbek az ágai.
- Ennek érdekében a fa építése során minél hamarabb levélcsúcsokat próbálunk meg képezni. Ehhez az kell, hogy egy csúcs tanító mintáinak kimenetei között kicsi legyen az eltérés (a 2-es norma), vagy minél kevésbé legyenek a kimentek változatosak (entrópia).
- Egy csúcshoz tehát úgy érdemes attribútumot választani, hogy megkeressük, melyik attribútum mellett kapunk a gyerekcsúcsoknál összességében legkisebb eltérésű kimenetekkel rendelkező tanító minta halmazokat.

Információ tartalom (Entrópia)

- ❑ P -beli minták információtartalma (entrópiája):

$$E(P) = E(p_1, \dots, p_n) = - \sum_{i=1}^n p_i \log_2 p_i$$

- ahol a P -ben n féle eltérő értékkel rendelkező minta van, amelyek P -beli gyakoriságainak aránya $p_1 : \dots : p_n$, és $p_1 + \dots + p_n = 1$

- ❑ Az éttermes problémában a mintáknak mindössze kétféle (pozitív vagy negatív) kimenete lehet. Ekkor

$$E(P) = E(p^+, p^-) = -p^+ \log_2 p^+ - p^- \log_2 p^-$$

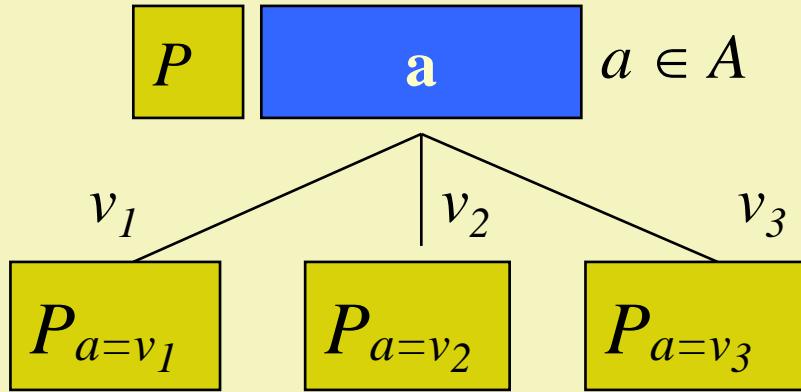
- ahol p^+ a P -beli pozitív, p^- a negatív minták aránya ($p^+ + p^- = 1$)

- 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: $E(P) = E(0/3, 3/3) = 0$

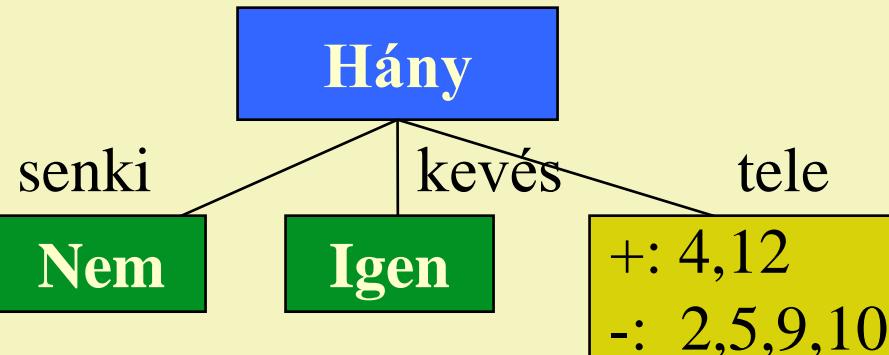
Információs előny számítása



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

- 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.



$$E(\{2,4,5,9,10,12\}) = \\ = E(2/6,4/6) = 0.92$$

Más, Bár, P/Sz, Éhes,
Ár, Eső, Fogl, Fajt, Idő

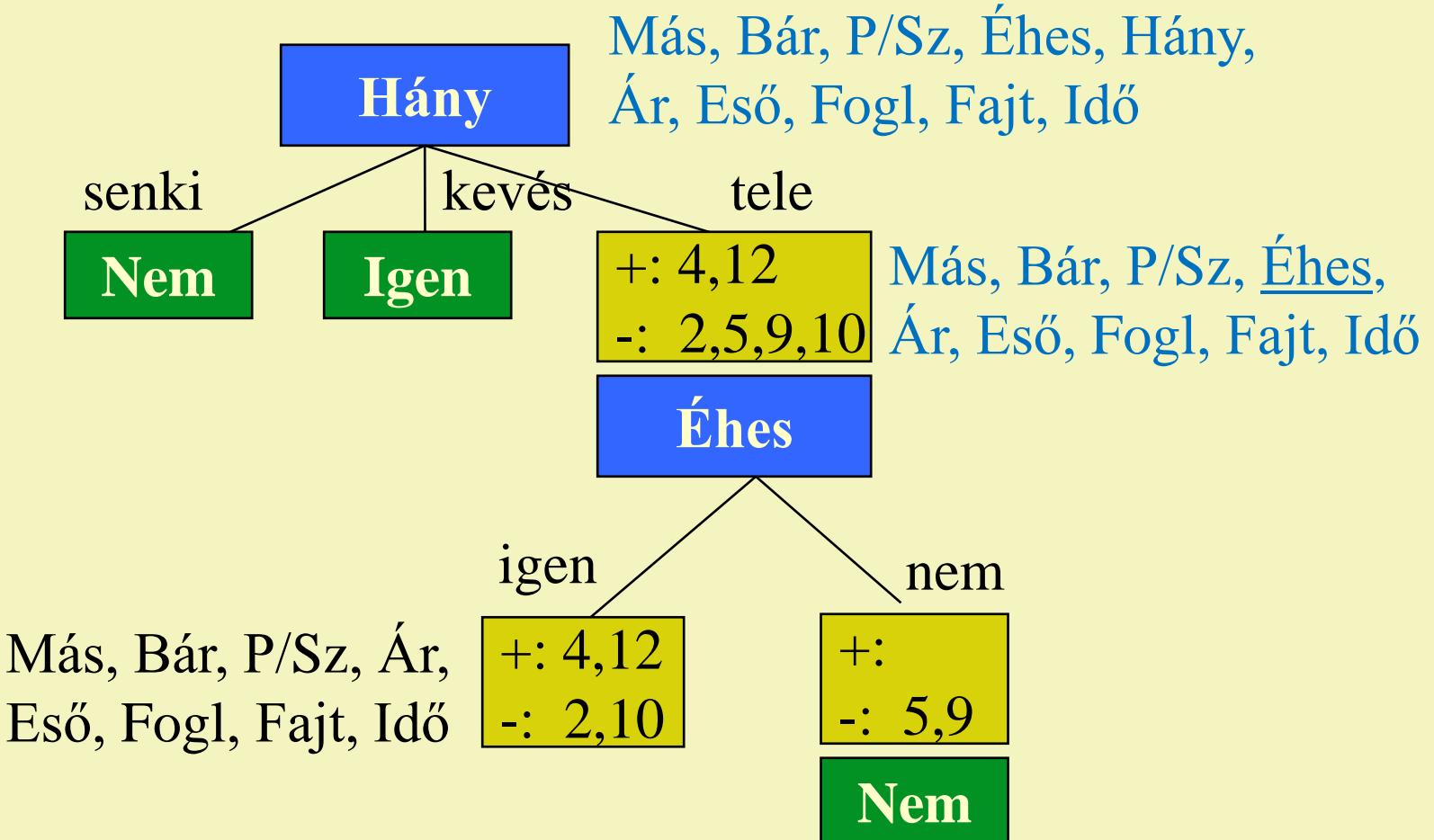
- Ha a fa építés folytatásakor a *Más* attribútumot választjuk, akkor a minták 1:5 arányban ketté válnak: {9} (*Más=hamis*), és {2, 4, 5, 10, 12} (*Más=igaz*),
 - $E(\{9\}) = E(0/1, 1/1) = 0$
 - $E(\{2,4,5,10,12\}) = E(2/5, 3/5) = 0.97$
- Az információs előny: $C(\{2,4,5,9,10,12\}, \text{Más}) = E(\{2,4,5,9,10,12\})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

Egy csúcs attribútumának kiválasztása 2.

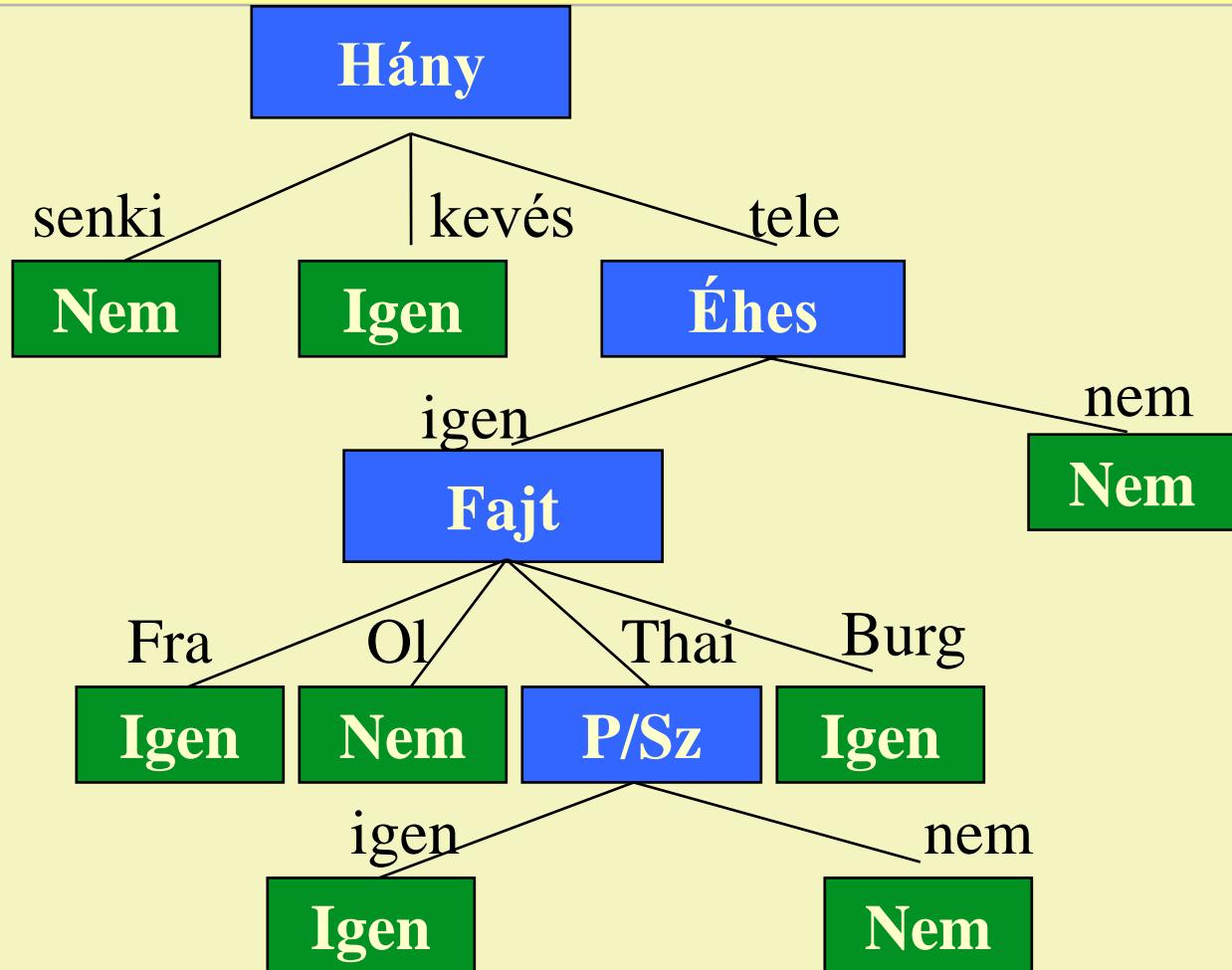
$$C(\{2,4,5,9,10,12\},a) = 0.92 -$$

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
<u>Éhes:</u>	<u>$4/6 E(2/4,2/4)+2/6 E(0/2,2/2)=$</u>	<u>0.67</u>
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:	$2/6 E(1/2,1/2)+1/6 E(0/1,1/1)+1/6 E(0/1,1/1)+2/6 E(1/2,1/2)=0.67$	0.67
<u>Idő:</u>	<u>$0/6 E(0,0)+2/6 E(1/2,1/2)+2/6 E(1/2,1/2)+2/6 E(0/2,2/2)=$</u>	<u>0.67</u>

További lépések



Étterem probléma döntési fája



Készítsünk algoritmust

- ❑ Egy fokozatosan épülő döntési fában minden csúcsnál eltároljuk
 - a csúcshoz tartozó (csúcshoz vezető ág attribútum-érték párjaival rendelkező) **tanító mintákat**
 - a csúcsnál még választható (a csúcshoz vezető út csúcsainak címkéiben nem szereplő) **attribútumokat**
- ❑ Egy csúcs lehet
 - attribútummal **címkézett belső csúcs**, amelyekből kivezető éleket az attribútum lehetséges értékei címkézik
 - **kiértékelт vagy értékkel nem rendelkező levélcsúcsok**
- ❑ minden lépésben egy értékkel még nem rendelkező levélcsúcsról kell eldöntenи, hogy kaphat-e értéket vagy legyen-e belső csúcs.

Algoritmus

- ❑ Kezdetben a fa egyetlen címkézettlen csúcsból áll (ez lesz majd a gyökér), amelyhez az összes mintát és attribútumot rendeljük.
- ❑ Veszünk a fából egy értékkel még nem rendelkező levélcsúcsot, amíg van ilyen:
 1. Ha a csúcsnak nincsenek mintái ($P = \emptyset$), akkor a szülőcsúcsának mintái alapján kiszámoljuk a csúcs értékét.
 2. Ha a csúcshoz tartozó minták kimenete nem nagyon tér el egymástól, akkor ezekből kiszámoljuk a csúcs értékét.
 3. Ha a csúcsnak nincsenek választható attribútumai ($A = \emptyset$), akkor a csúcs mintái alapján kiszámoljuk a csúcs értékét.
 4. Egyébként a választható attribútumok közül a legkedvezőbbnek látszával megcímkézzük az adott csúcsot, és generáljuk a gyerekeit az azokhoz tartozó mintákkal és választható attribútumokkal együtt.

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 a kocka színe és a dobás 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 döntési fában az $x \in X$ bemenetre kiszámolt levélcsúcs értéke:

$$f(\Theta, x) = \sum_{n=1..N} \frac{\mathbb{I}(x_n \in P(x))}{|P(x)|} \cdot y_n$$

$P(x)$ az x -re kiszámolt levélcsúcs tanító mintái bemeneteinek halmaza

$$f(\Theta, x) = \arg \max_{n=1..N} \sum_{\substack{x_i \in P(x) \\ \varphi(x_i) = y_n}} \mathbf{1}$$

- Θ a döntési fa, amely optimalizálása annak mohó felépítése.

előny: jól értelmezhető (a mintákra tökéletes eredményt ad, és 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: az optimális döntési fa építése NP-teljes, a bemutatott mohó módszerrel csak lokálisan optimális döntési fához jutunk

1.3. Véletlen erdő

- K darab döntési fát építünk a tanító minták alapján úgy, hogy egy-egy fa építéséhez a tanító mintáknak is ($P_k \subseteq P$), és az attribútumoknak is ($A_k \subseteq A$) csak egy-egy véletlen kiválasztott részhalmazát használjuk fel. Így kapjuk a véletlen erdőt.
- 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 kimenetei alapján becsüljük az x kimenetét. Ez lehet az egyes fák
 - adott levelei értékéből képzett átlag, vagy
 - adott leveleinek értékei közül a leggyakoribb

Tanulás véletlen erdővel

- Egy x bemenethez tartozó kimenetet a minták kimeneteinek súlyozott áлага, 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:

$$f(\Theta, x) = \sum_{k=1}^K \frac{\text{value}(x, \text{decision_tree}(A_k, P_k))}{K}$$

ami a P_k és az A_k alapján felépített döntési fából számolható ki x -re

$$f(\Theta, x) = \arg \max_{k=1}^K \sum_{i=1}^K \mathbb{I}(\text{value}(x, \text{decision_tree}(A_k, P_k)) = \text{value}(x, \text{decision_tree}(A_i, P_i)))$$

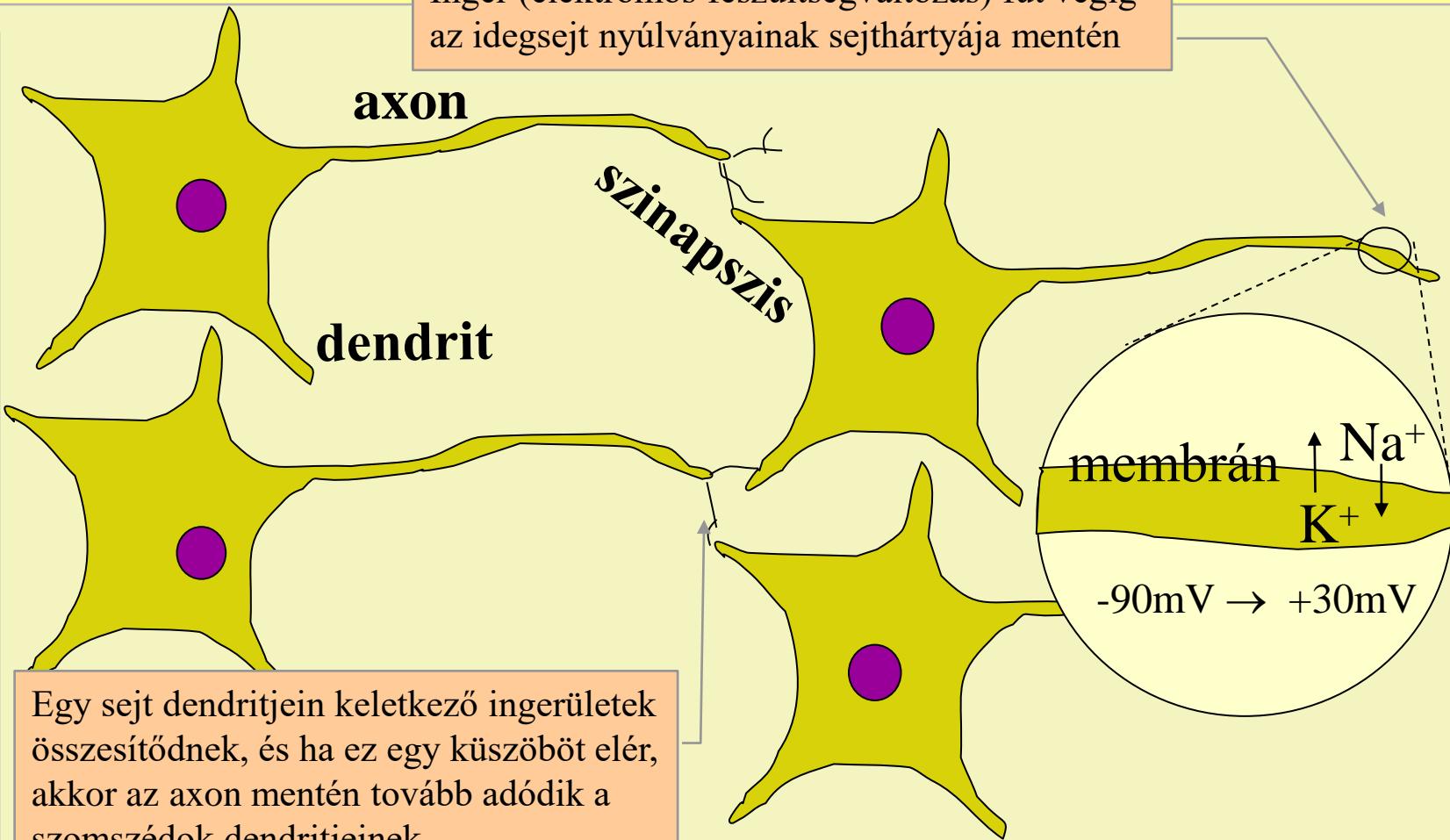
- Θ maga a véletlen erdő, optimalizálása az erdő felépítése

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

3. Mesterséges neuronhálók

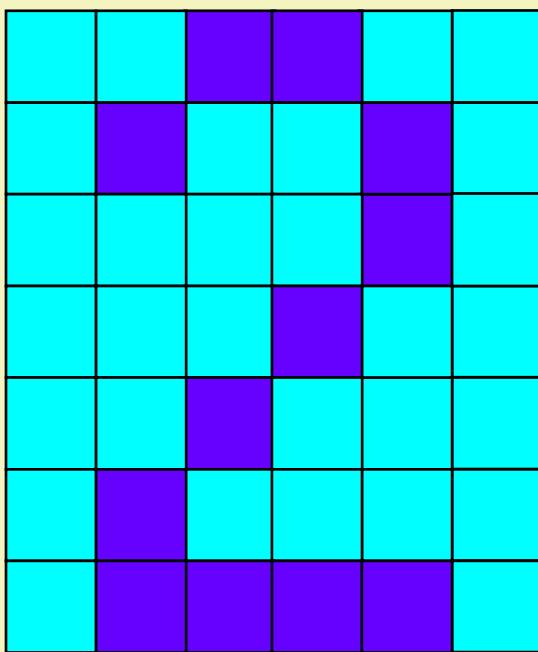
Inger (elektromos feszültségváltozás) fut végig az idegsejt nyúlványainak sejthártyája mentén



Számjegy felismerés egy mesterséges neuronhálózata

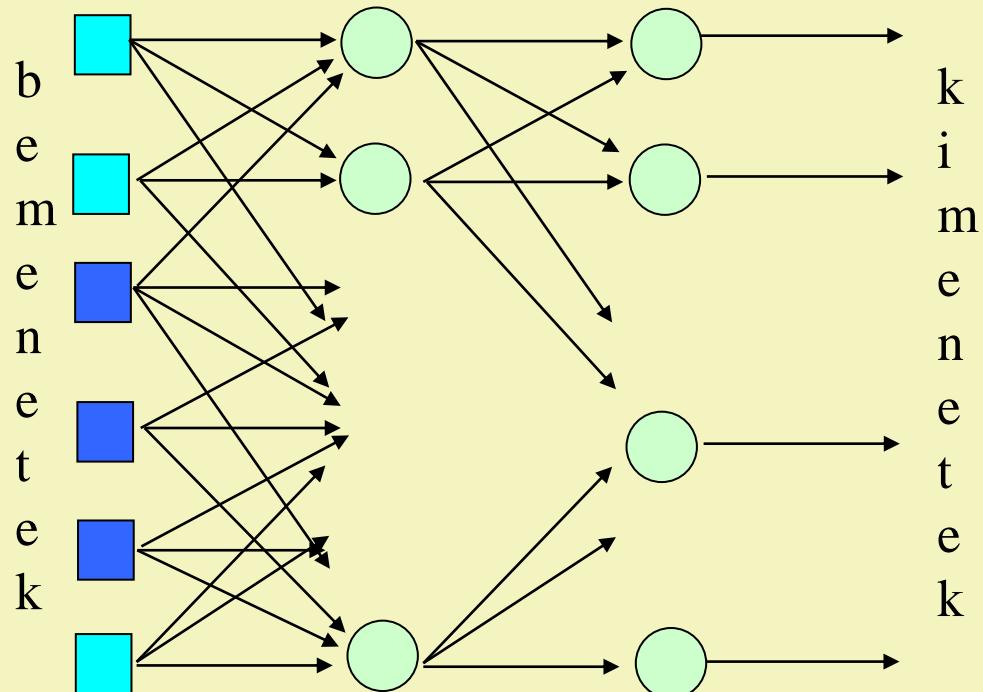
Bemeneti értékek száma: 42

Minden pixel generál
egy bemeneti értéket.



Kimenetek száma: 10

Minden számjegyhez tartozik egy neuron a
kimeneti rétegben, amely akkor tüzel, ha az
adott számjegyet véli felismerni a rendszer.



Mesterséges neuronhálók alkotóelemei

Mesterséges neuron

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

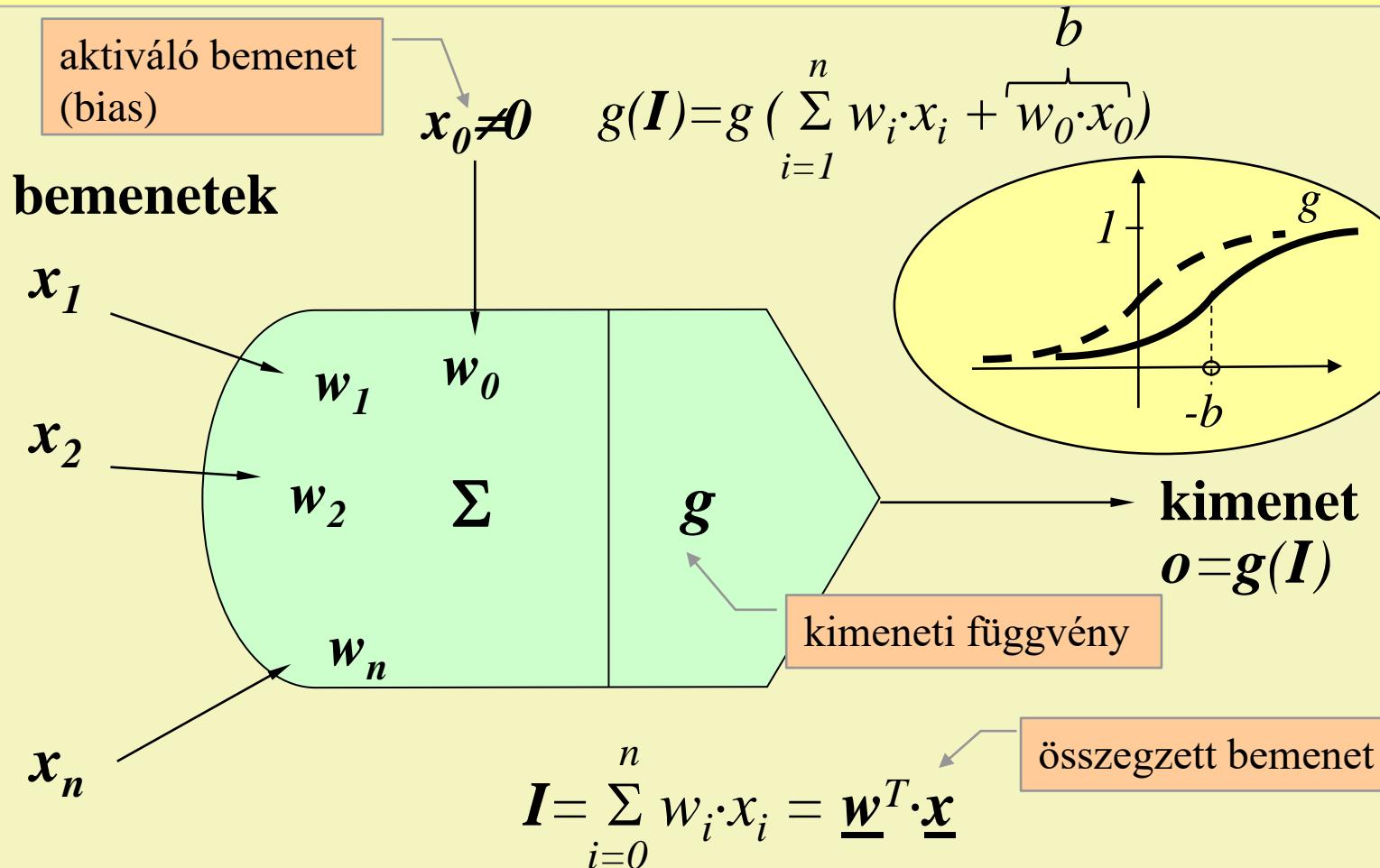
Hálózati topológia

- 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

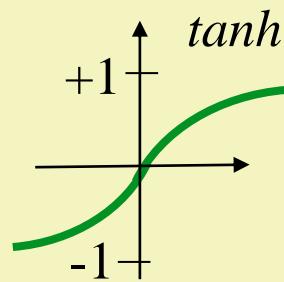
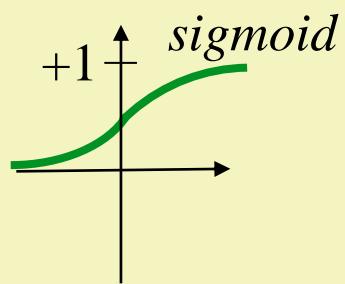
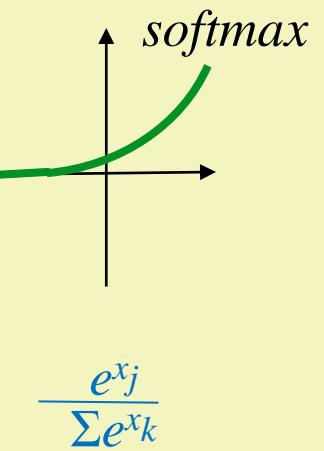
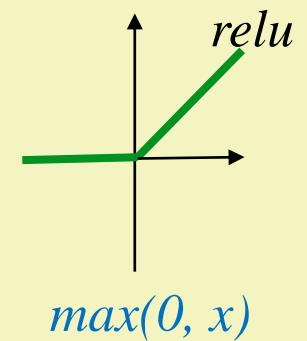
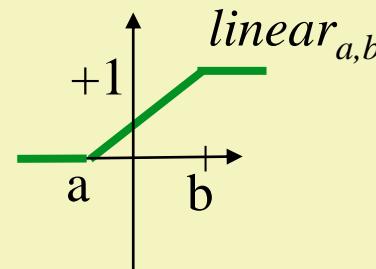
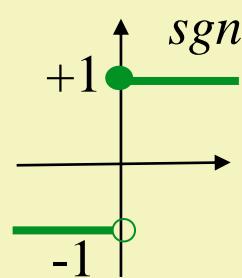
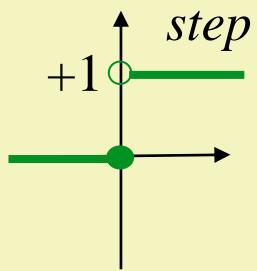
Tanulási szabály

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

Általánosított perceptron

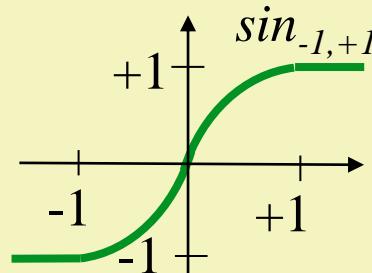


Kimeneti függvények



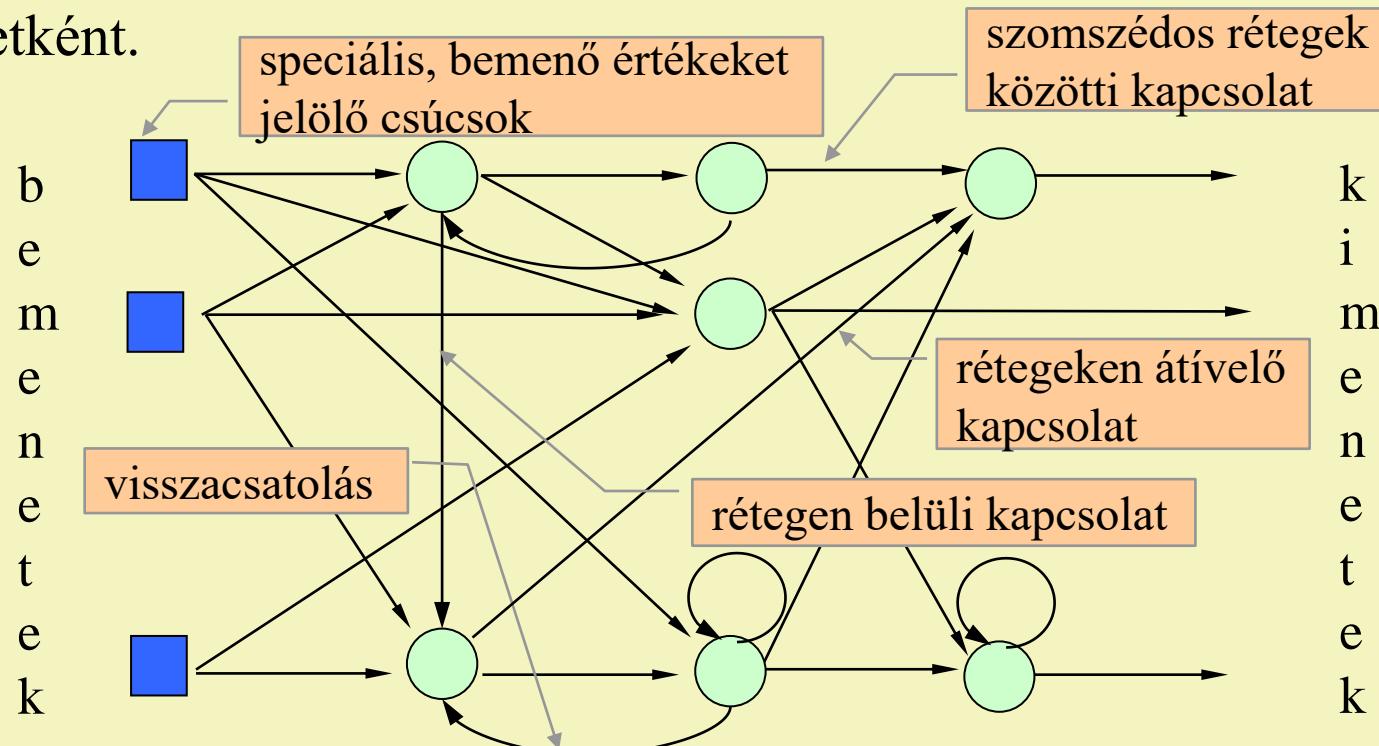
$$\frac{1}{1+e^{-x}}$$

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

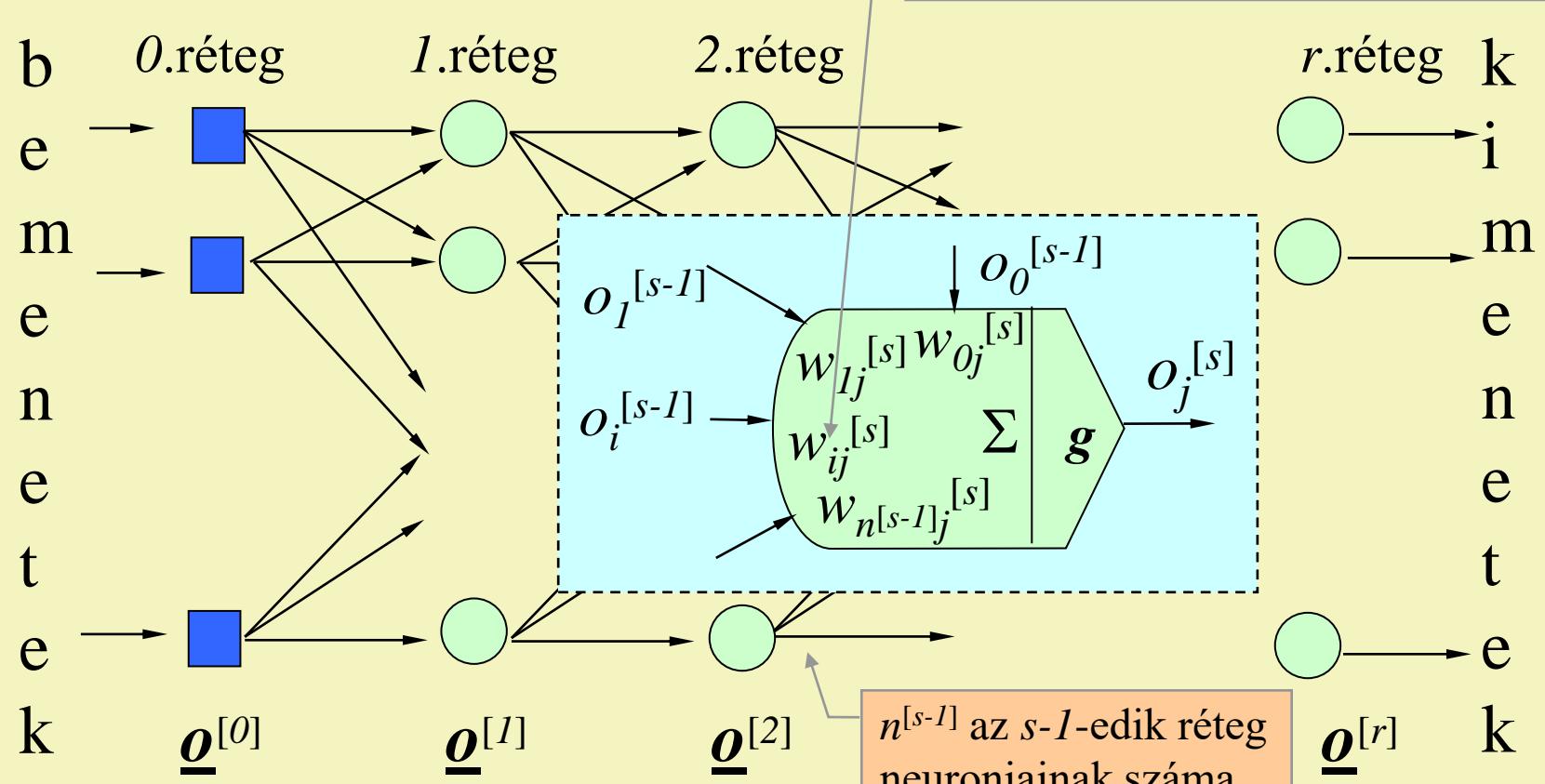


Hálózati topológia

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.

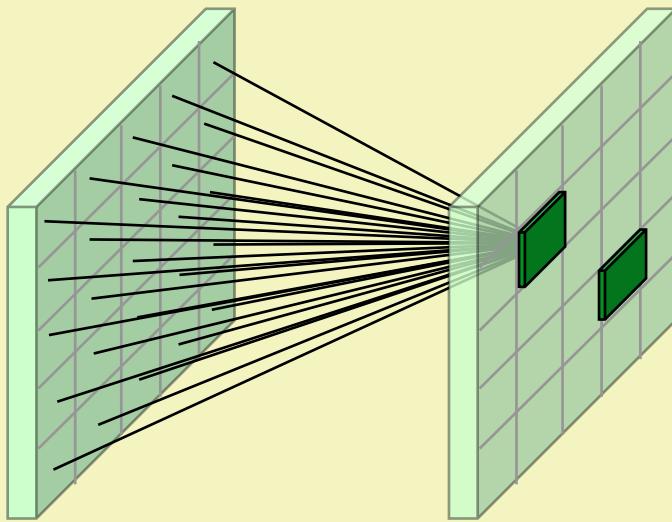


Többrétegű előrecsatolt hálózat (feed forward MLP)



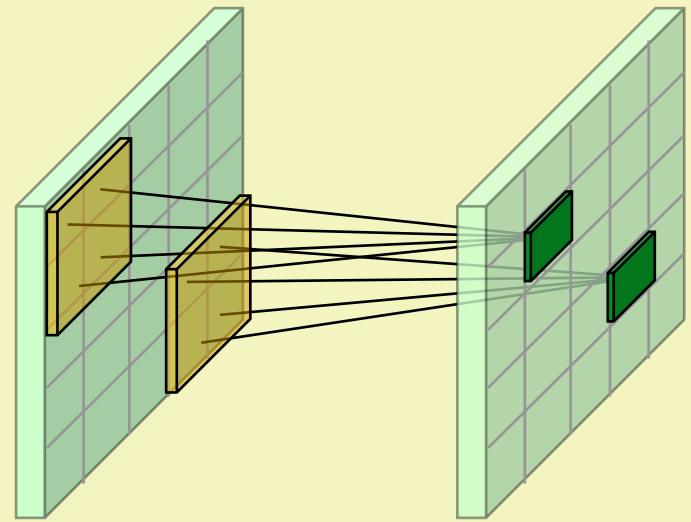
Konvolúciós neuron hálózat

Teljesen összekötött (sűrű)
fully connected neural net



Pl.: 1000×1000 -es első réteg esetén
a második réteg egy neuronjában
 10^6 darab súlyt kell tárolni.

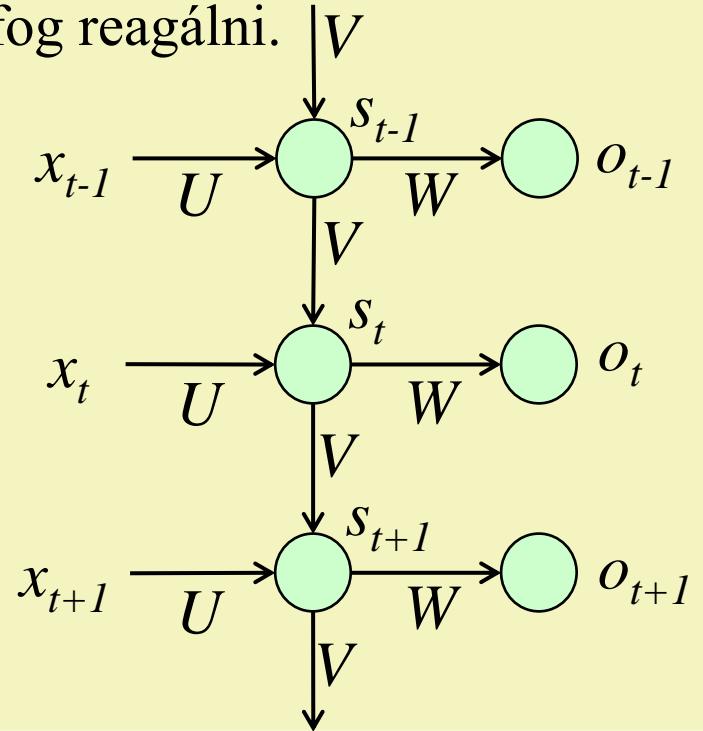
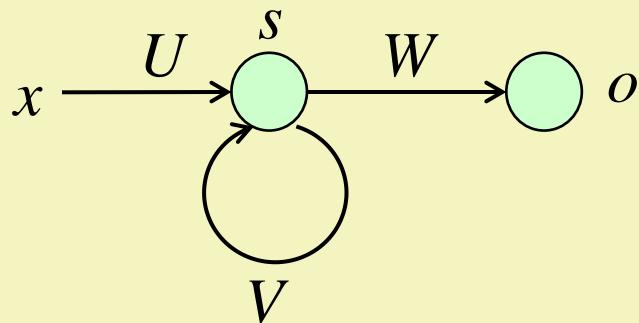
Lokálisan összekötött
convolutional neural net



Pl.: 1000×1000 -es első réteg esetén
egy 2×2 -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 számítási képlet azt utánozza, mintha a neuron emlékezne a megelőző inputokra, mivel egy új inputra a megelőző inputokra kiszámolt outputot is felhasználva fog reagálni.

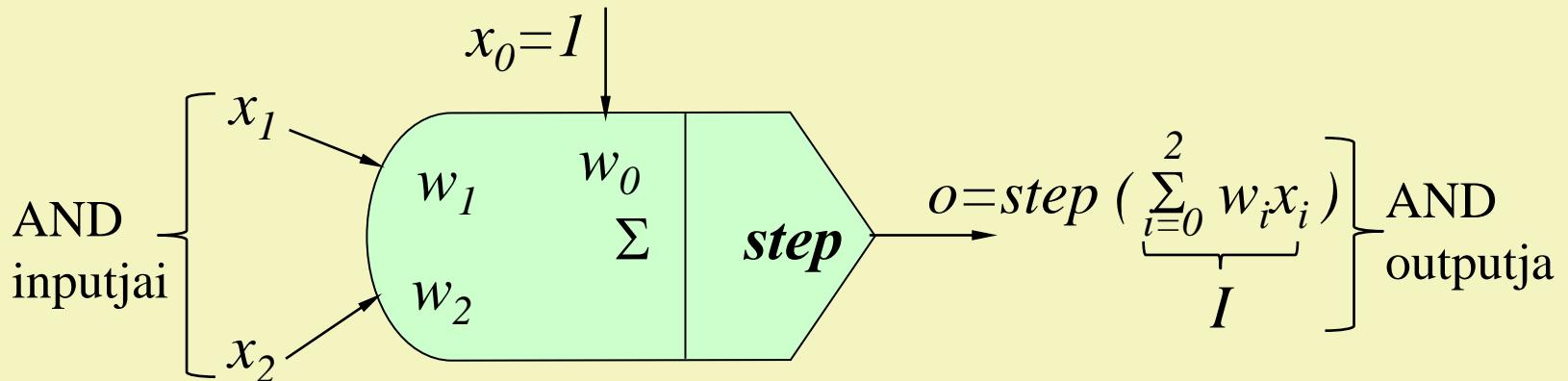


Általánosított perceptron tanulása

- ❑ 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 (törlesét) jelenti.
- ❑ Tanulás során a súlyokat fokozatosan módosítjuk ($w := w + \Delta w$).
- ❑ A Δw 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.

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 ($o \in \{0,1\}$)

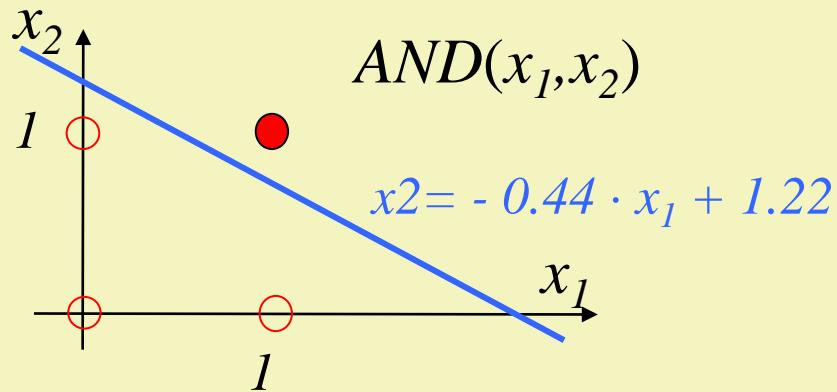
Delta szabály

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		1	-1
2.	0	1	0	-0.02	-0.02	0.08		1	-1
3.	1	1	1	-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	= 0.06	1	-1
5.	0	1	0	-0.12	-0.02	0.08		0	0
6.	1	1	1	-0.12	-0.02	0.08	= -0.06	0	1
második epoch				-0.22	0.08	0.18			
13.	1	0	0	-0.22	0.08	0	= -0.14	0	0
14.	0	1	0	-0.22	0	0.18	= -0.04	0	0
15.	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

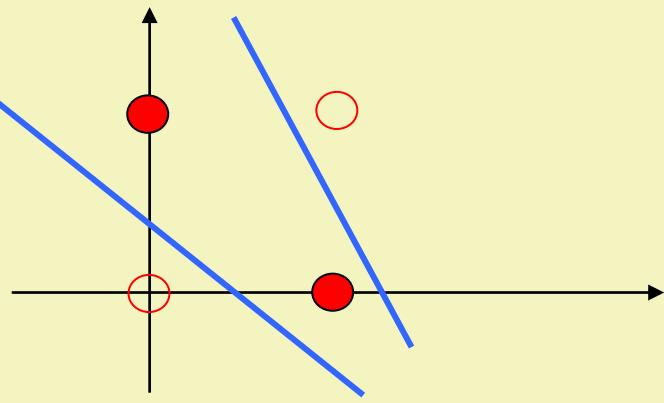
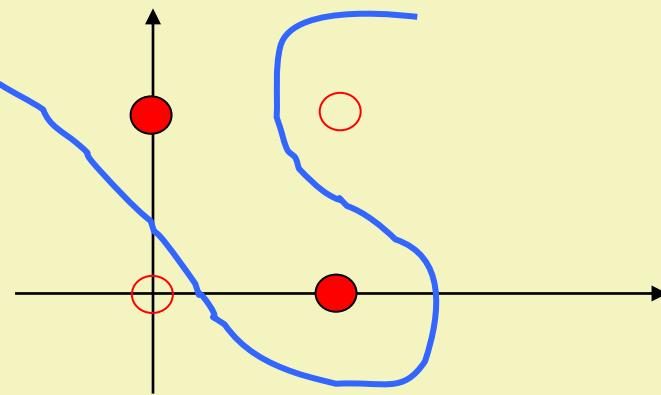
Mit tanultunk meg?

- ❑ A neuron működése azonos az $AND(x_1, x_2)$ működésével, mert az összegzett bemenet előjele csak az $(1,1)$ bementre pozitív:
 - ha $I(x_1, x_2) \leq 0$ akkor $step(I(x_1, x_2)) = 0$
 - ha $I(x_1, x_2) > 0$ akkor $step(I(x_1, x_2)) = 1$
- ❑ A w együtthatók megtanulásával azt az $I(x_1, x_2) = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2$ egyenest kaptuk meg a lehetséges bemenet-párok síkján, amely elszeparálja (osztályozza) a bemenet-párokat: egy oldalra (félsíkra) kerülnek az azonos eredményű bemenetek.

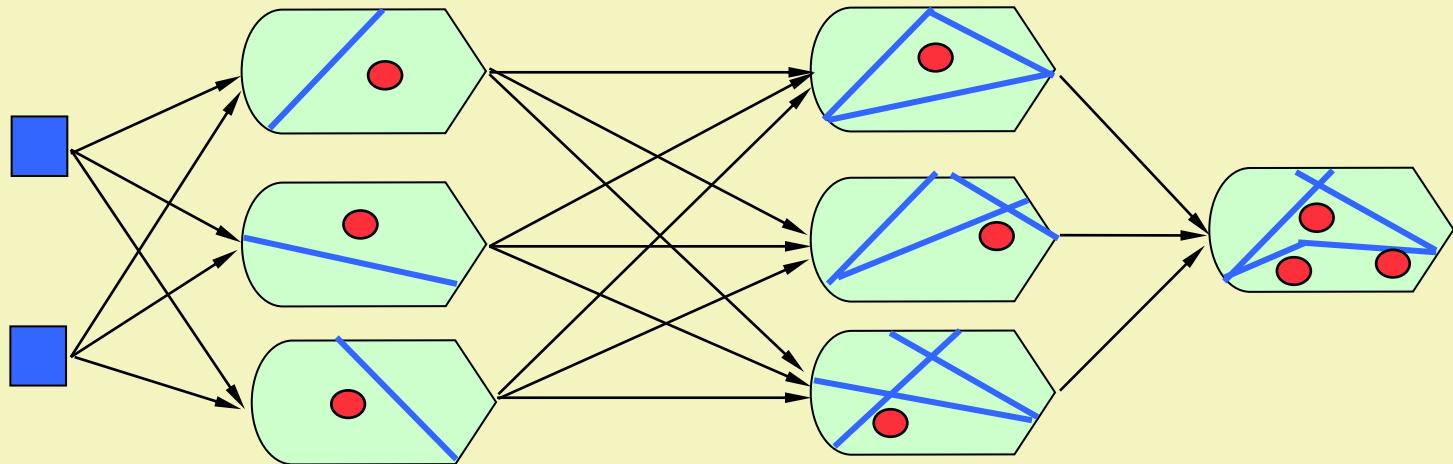


Lineáris szeparálhatóság

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



Rétegek „tudása”



- Több réteggel összetettebb problémák is megoldhatók, de mivel a *step* függvény (**nem deriválható**) egyszerű perceptronokkal eddig nem sikerült megfelelő tanuló algoritmust találni ehhez.
- De ha lecseréljük a kimeneti függvény (pl. szigmoidra), akkor már létrehozható olyan algoritmus, amellyel a háló tanítható (pl. error backpropagation).

Homogén MLP háló számítási modellje

- ❑ 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 Θ paraméter a $(w_{ij}^{[s]})$ súlyok összessége

deriválható
aktivizációs
függvény

- ❑ Egy réteg számítási modellje

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

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. A tanulás során ennek a függvénynek keressük a minimum helyét gradiens módszerrel, azaz lépésről lépére módosítjuk a súlyokat megfelelő irányban kis mértékben.

$$w_{ij}^{[s]} := w_{ij}^{[s]} - \Delta w_{ij}^{[s]}$$

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

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

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

Egy neuronra visszavetített számítási hiba

Ha $s=r$ akkor

$$e_j^{[r]} = o_j^{[r]}(1 - o_j^{[r]})(t_j - o_j^{[r]})$$

$$\Delta w_{ij}^{[r]} = \eta e_j^{[r]} o_i^{[r-1]}$$

$$e_j^{[s]} = - \frac{\partial E}{\partial I_j^{[s]}}$$

ha g a szigmoid függvény

Ha $s < r$ akkor

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

$$\Delta w_{ij}^{[s]} = \eta e_j^{[s]} o_i^{[s-1]}$$

ha g a szigmoid függvény

rekurzív szabályt kapjuk a súlyok módosítására.

Backpropagation tanuló algoritmus

1. Az \underline{x} bemeneti vektorból indulva rétegenként számoljuk a neuronok kimenetét: $o_j^{[s]}$, a kimeneti réteg kimeneteit $o_j^{[r]}$ 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 sigmoid függvény

3. Rétegenként hátulról előre haladva számoljuk a belső neuronok hibáit:

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

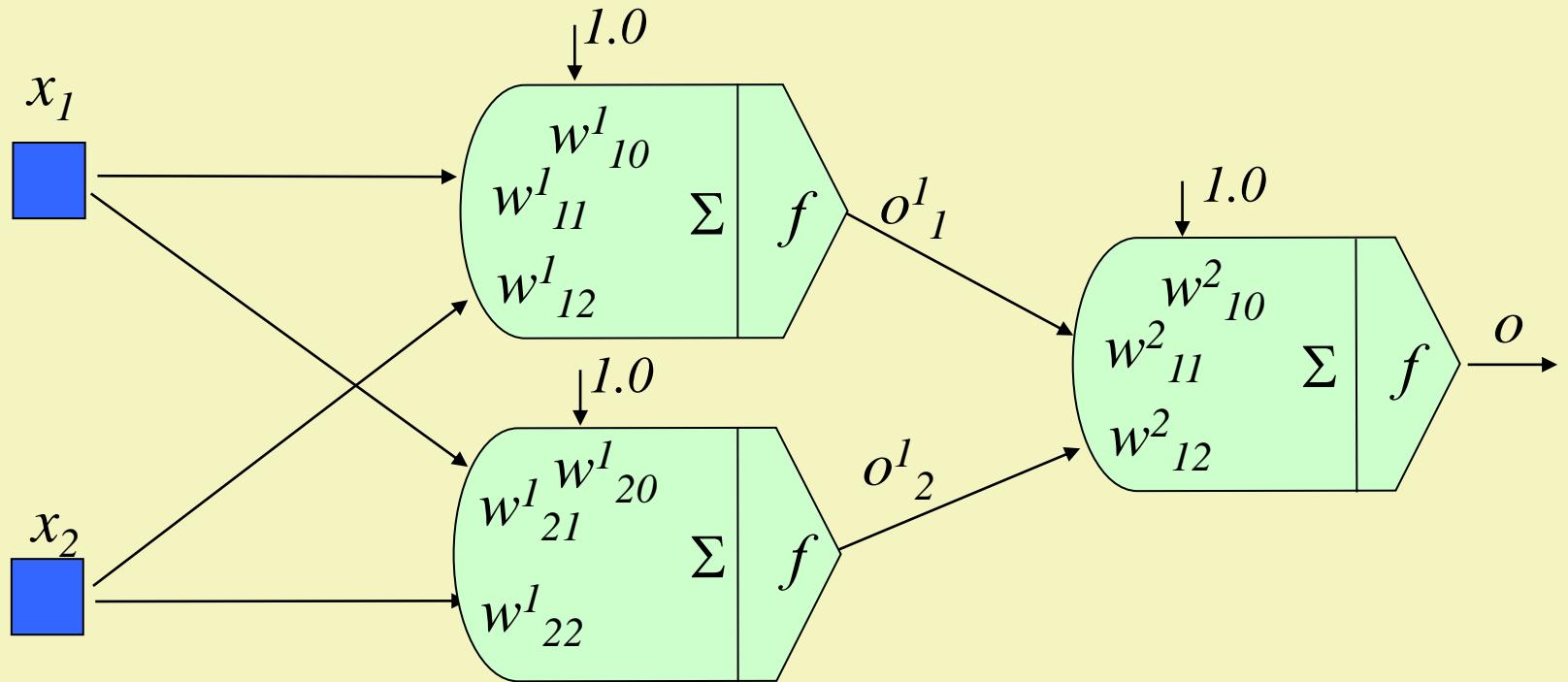
ha g a sigmoid függvény

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

XOR művelet példája

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

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



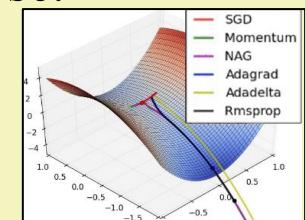
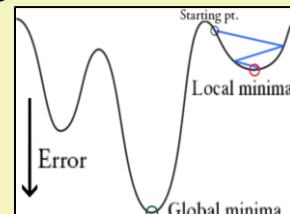
Rétegenként eltérő aktivizációs függvény

- Egy inhomogén MLP háló számítási modellje: $f:P \times X \rightarrow Y$

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

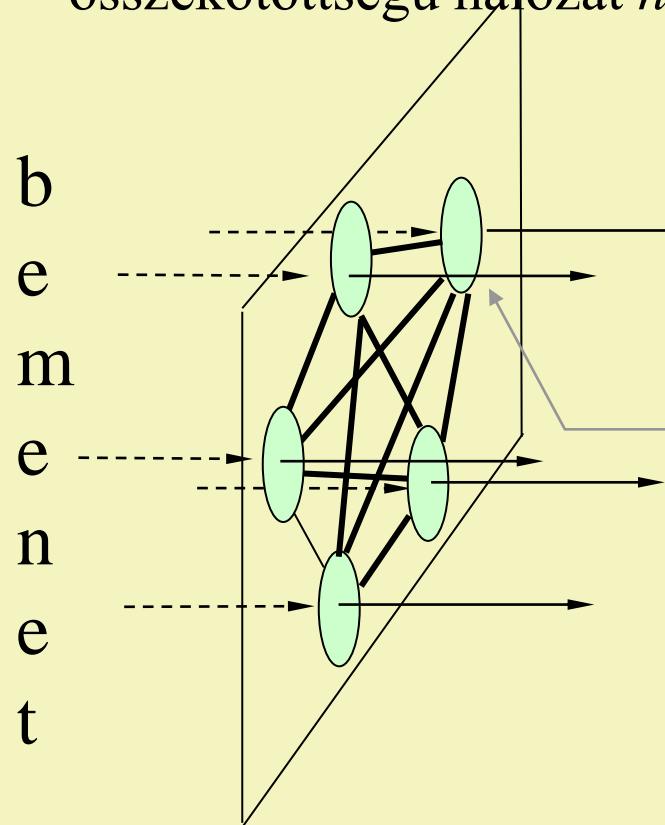
- $\Theta = \{w^{[1]}, \dots, w^{[r]}\}$ azaz a paraméterek a súlyok
- $g_s : P \times X^{s-1} \rightarrow X^s$ s -edik réteg kimeneti függvénye ($X = X^0, Y = X^r$)

- A gradiens elméleti kiszámítása nehéz, ezért erre numerikus módszereket használnak. (Keras, TensorFlow)
- További problémák:
 - A tanító minták kiválasztása nehéz (homogén minták, overfitting).
 - A hiper-paraméterek (N , η) megtanulásának kérdése.
 - A lokális minimum-, illetve a nyeregpontok problémája.

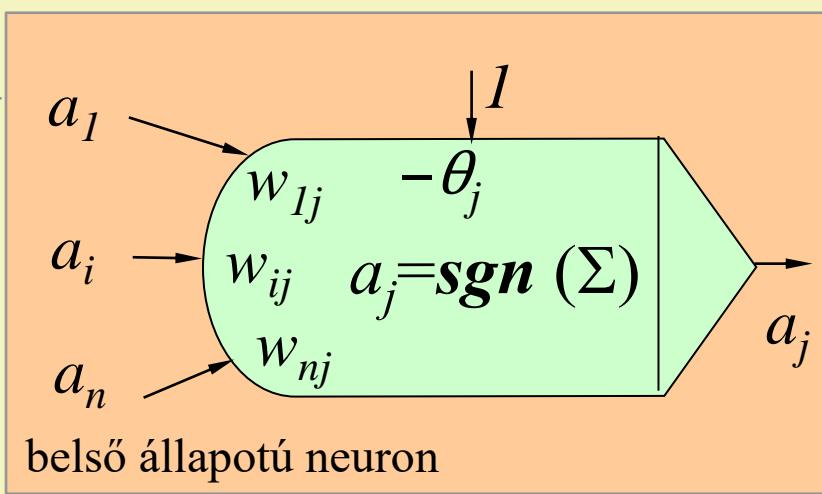


Hopfield modell

- Aszinkron működésű Hopfield topológia: egyrétegű, teljes összekötöttségű hálózat n darab +1 v. -1 állapotú neuronnal.



k
i
m
e
n
e
t



Hopfield modell működése

- ❑ Kezdetben a neuronok kívülről kapják értékül az állapotaikat. A neuronok által felvett állapot-együtttest hívjuk a háló egy konfigurációjának.
(Az összes konfiguráció alkotja a konfigurációs teret.)

- ❑ Ezután elkezdik újra számolni az állapotukat, aszinkron módon többször is, folyamatosan változtatva a hálózat konfigurációját:

$$a_j^{új} = \text{sgn}\left(\sum_{i=1}^n w_{ij} a_i - \theta_j \right)$$

(a_i az i -dik neuron állapota, w_{ij} i -dik neuronból a j -dik neuronba vezető kapcsolat súlya, a θ_j a j -dik neuron küszöbértéke)

- ❑ Végül, amikor a hálózat egy stabil konfigurációba jut (azaz az állapotok újraszámolásuk ellenére sem változnak tovább), akkor a neuronok állapotait a háló kimeneteinek tekintjük.

Hopfield hálózat felhasználása

- ❑ A modell eredetileg **asszociatív memória** megvalósítására készült. Ilyenkor a hálózat egy-egy stabil konfigurációja ad meg egy eltárolt mintát (pl. egy kép pixelpontjait).
- ❑ Amikor a minta zajos, torzított vagy hiányos változatát adjuk meg bemenetként a háló neuronjainak (kibillentve őket egy stabil konfigurációból), akkor a hálózat addig változtatja az állapotát a neuronjainak, amíg újra stabil konfigurációba nem kerül, azaz fel nem idézi az eredeti mintát (több minta esetén a leghasonlóbbat).
- ❑ A működéssel kapcsolatban megvizsgálandó kérdések:
 - Hogyan lehet (felügyelet nélkül) betanítani arra a hálózatot, hogy a tárolt minták a hálózat stabil konfigurációi legyenek?
 - Konvergál-e a hálózat a stabil konfigurációk valamelyikéhez?

Hopfield hálózat egyetlen minta tárolására

- Kell, hogy a tárolandó $\mathbf{p} \in \{+1, -1\}^n$ minta egy stabil állapotkonfigurációja legyen a hálónak, azaz $p_j = \text{sgn}(\sum_i w_{ij} \cdot p_i)$ – feltéve, hogy $\theta_j = 0$. Ez akkor teljesül, ha $w_{ij} \cong p_i \cdot p_j$.
- Az ilyen minta jelentős vonzáskörzettel bír: ekkor ugyanis minden olyan \underline{a} konfigurációból, amely komponenseinek több, mint a fele azonos a mintáéval (tehát $\sum_i p_i \cdot a_i > 0$), a háló véges lépésekben a mintához konvergál.
 - Ez abból következik, hogy amikor egy neuron újra számolja az állapotát, akkor az azonnal a minta megfelelő komponensével válik azonossá:
$$a_j^{(ij)} = \text{sgn}(\sum_i w_{ij} \cdot a_i) = \text{sgn}(\sum_i p_i \cdot p_j \cdot a_i) = \text{sgn}(p_j \cdot \sum_i p_i \cdot a_i) = p_j.$$

Hopfield hálózat több minta tárolására

- Legyen d darab tárolandó mintánk: $\mathbf{p}^k \in \{+1, -1\}^n$ ($k=1 \dots d$).
- Válasszuk a súlyoknak a $w_{ij} = 1/n \cdot \sum_k p_i^k \cdot p_j^k$ szuperpozíciókat.
Ugyanez inkrementális tanulási szabályként is felírható:
$$\Delta w_{ij} = 1/n \cdot p_i^k \cdot p_j^k$$
 . Hebb szabály
- Véletlenszerűen választott (ortogonális) minták esetén annak, hogy a minták stabil konfigurációk legyenek, akkor a legnagyobb a valószínűsége, ha $d \leq n/\log n$.
- Konvergencia: Vegyük a $-\frac{1}{2} \sum_{i,j} w_{ij} \cdot a_i \cdot a_j - \sum_i \theta_i \cdot a_i$ hibafüggvényt.
Belátható, hogy ha a súlymátrix szimmetrikus, és a diagonális elemei nem-negatívak, akkor egy konfiguráció-váltás során a függvény értékének csökkennie kell. Mivel a konfigurációk száma véges, így a hálózat véges lépésekben stabil konfigurációba fog jutni.

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

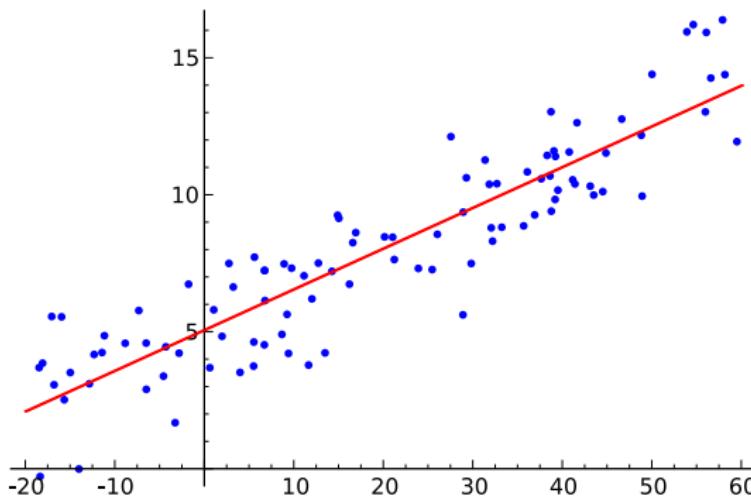
- Kovariancia, korreláció
- Főkomponens analízis

4 Autoenkóderek

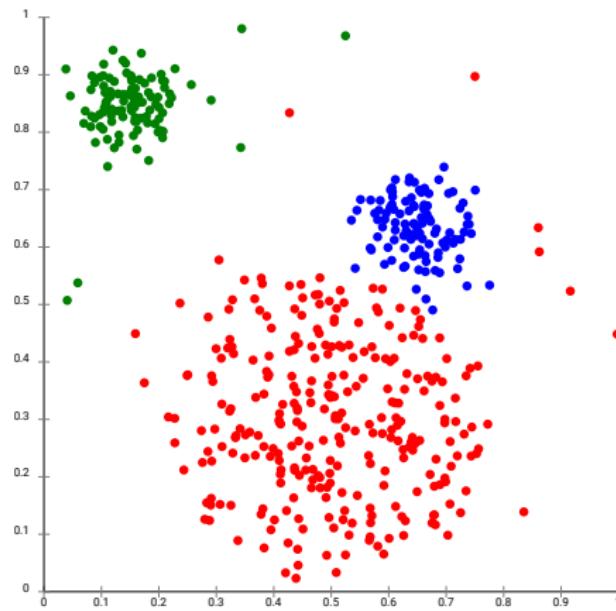
Felügyelt tanulás – osztályozás

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 8 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

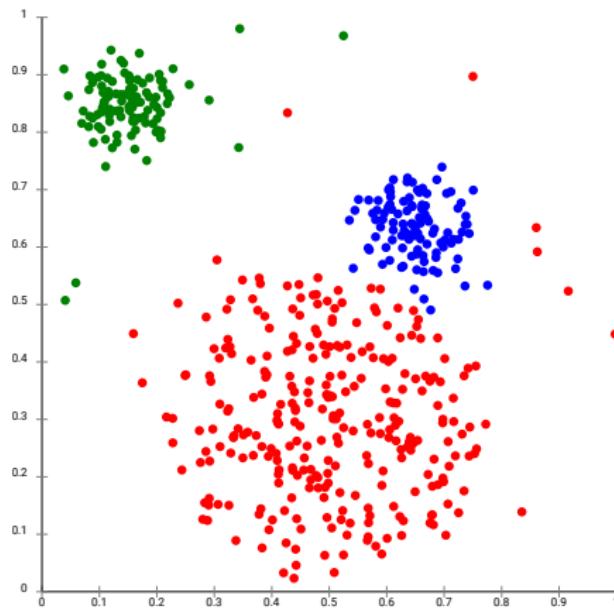
- Hard clustering – k-means
- Soft clustering – témamodellek

3 Dimenziócsökkentés

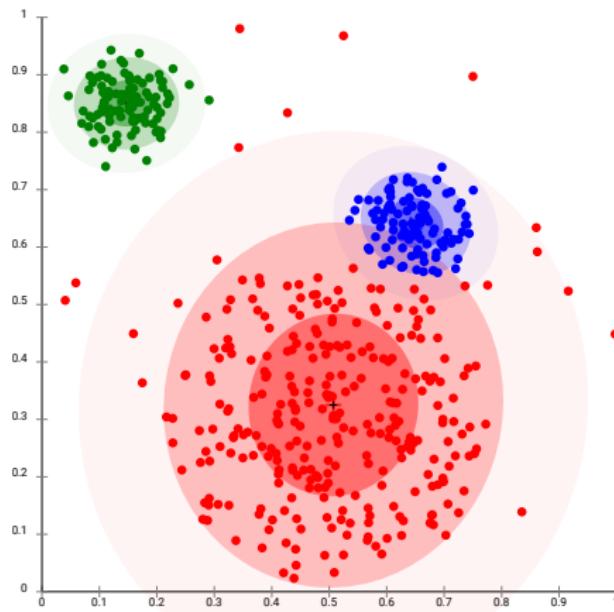
- Kovariancia, korreláció
- Főkomponens analízis

4 Autoenkóderek

Példa – eloszlás alapú klaszterezés



Példa – eloszlás alapú klaszterezés



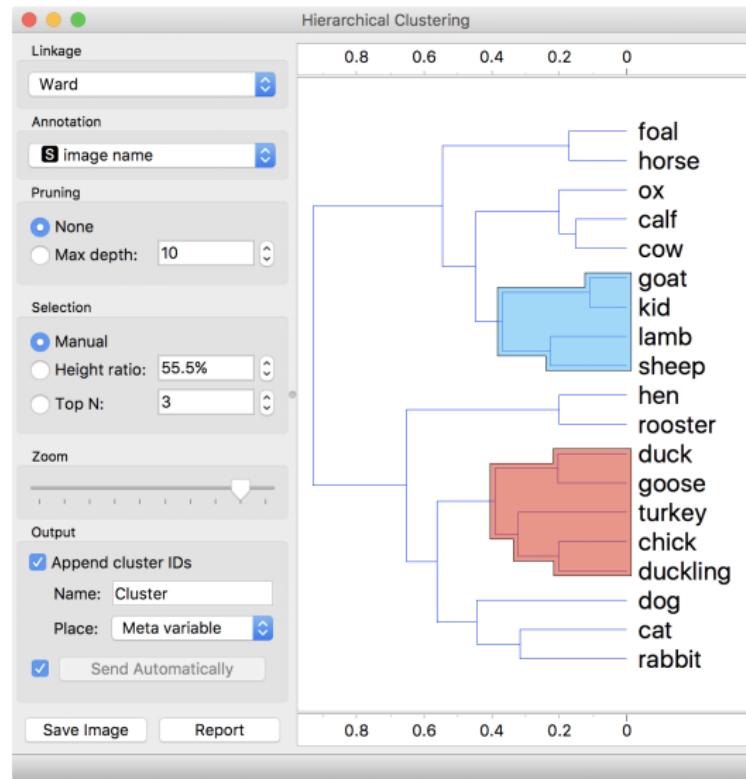
Feladat

- 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

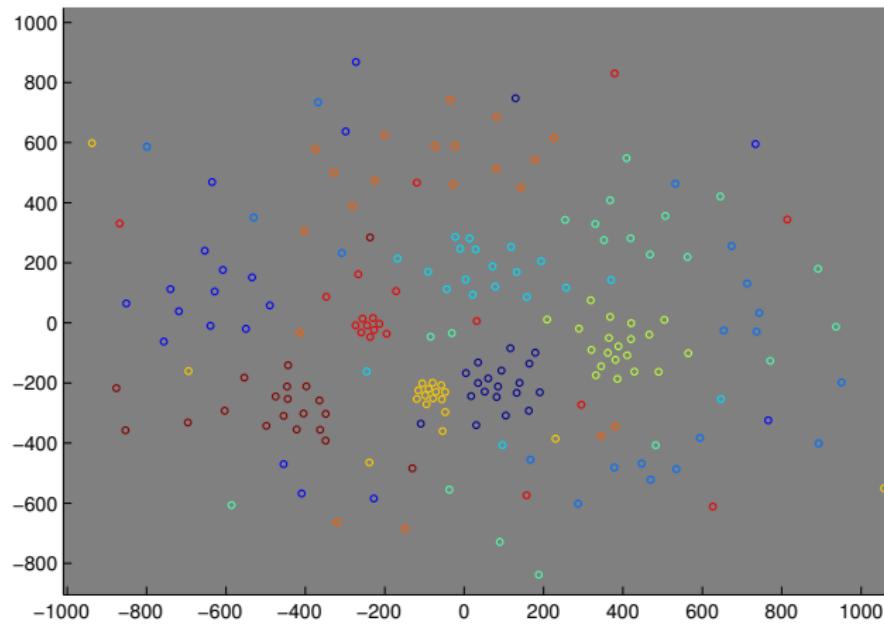
Fajtái

- 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

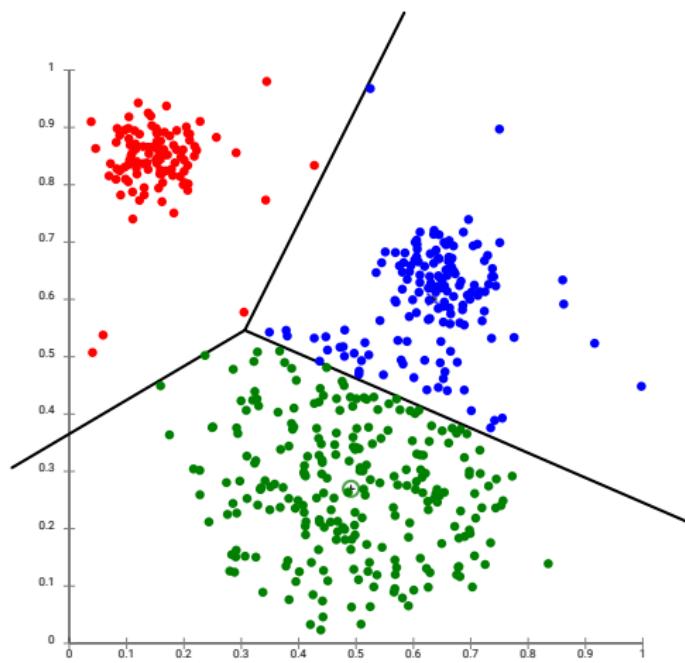
- Hard clustering – k-means
- Soft clustering – témamodellek

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

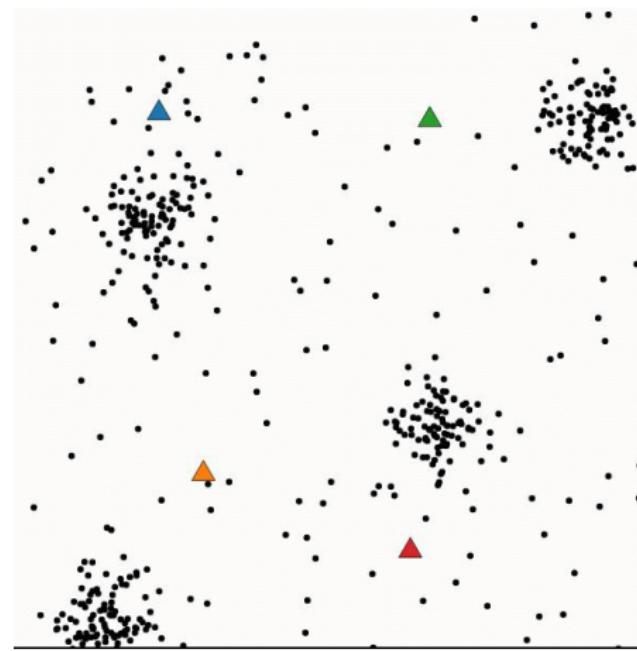
- Kezdetben adott k és az adatpontok \mathbb{R}^n -ben
- Inicializáljuk az $m_1^{(1)}, m_2^{(1)}, \dots, m_k^{(1)}$ centroidokat
 - Véletlenszerűen kiválasztunk k adatpontot, vagy
 - minden adatpontot véletlenszerűen egy klaszterbe sorolunk és kiszámoljuk a centroidokat
- Váltogatjuk a következő két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

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

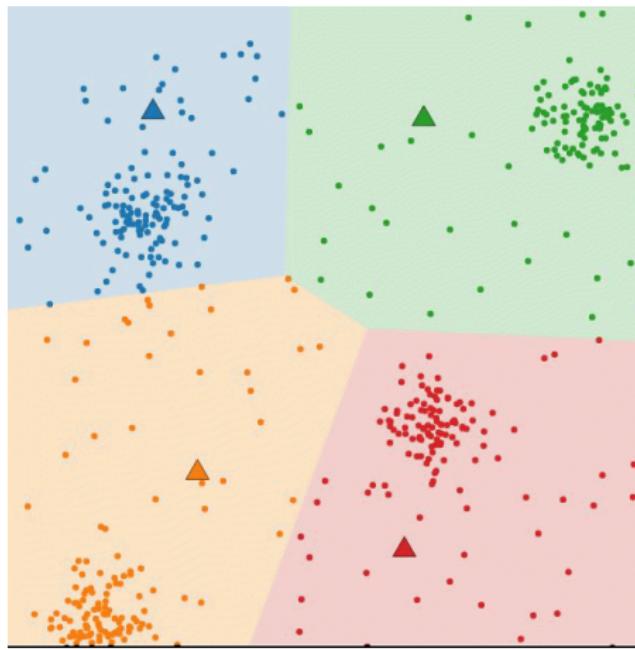
- 2 Kiszámítjuk az új centroidokat

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

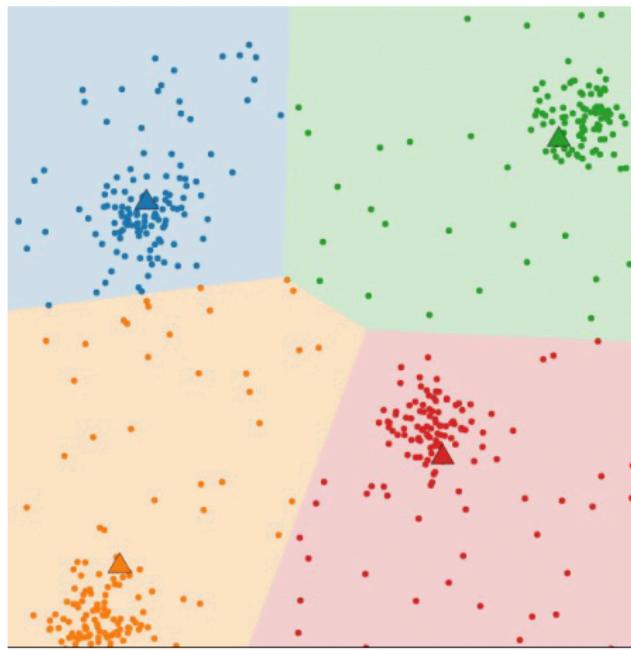
Algorithmus



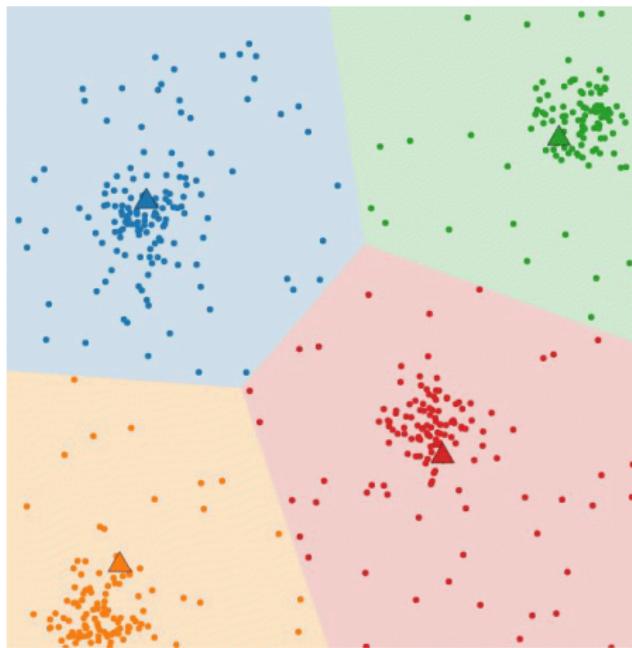
Algoritmus



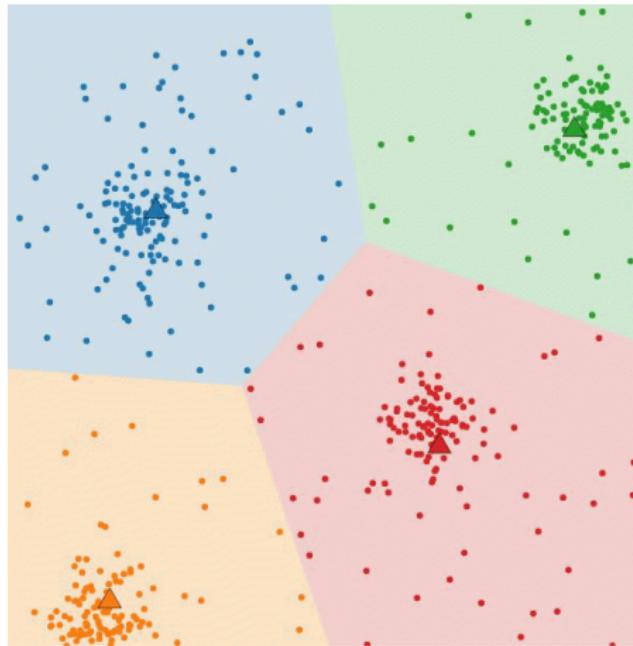
Algoritmus



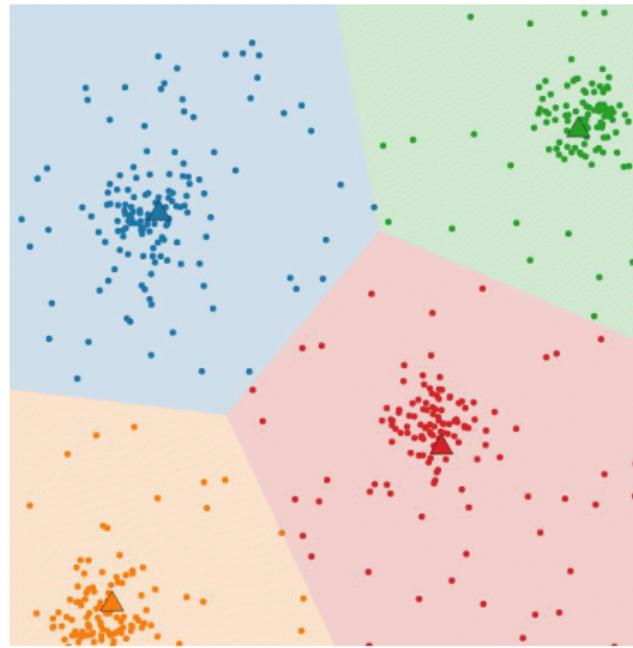
Algoritmus



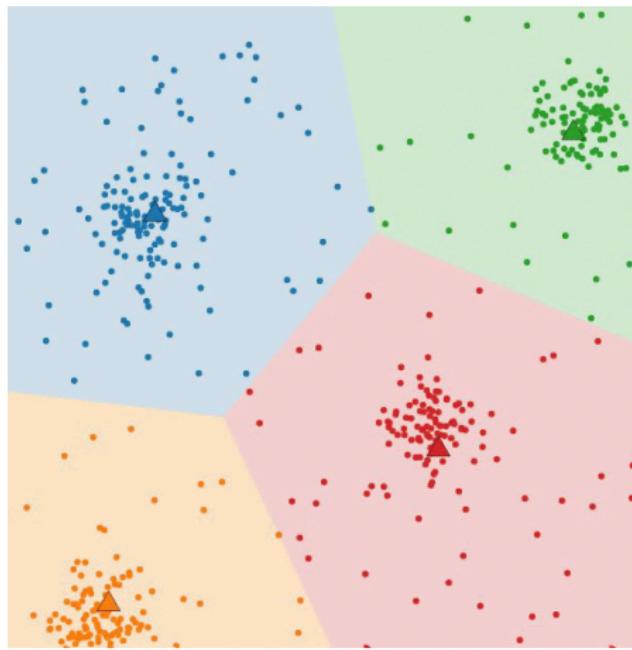
Algoritmus



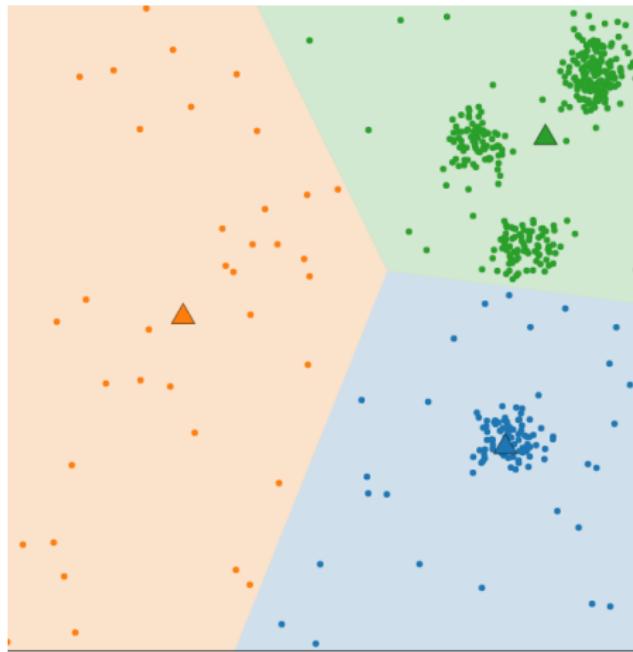
Algorithmus



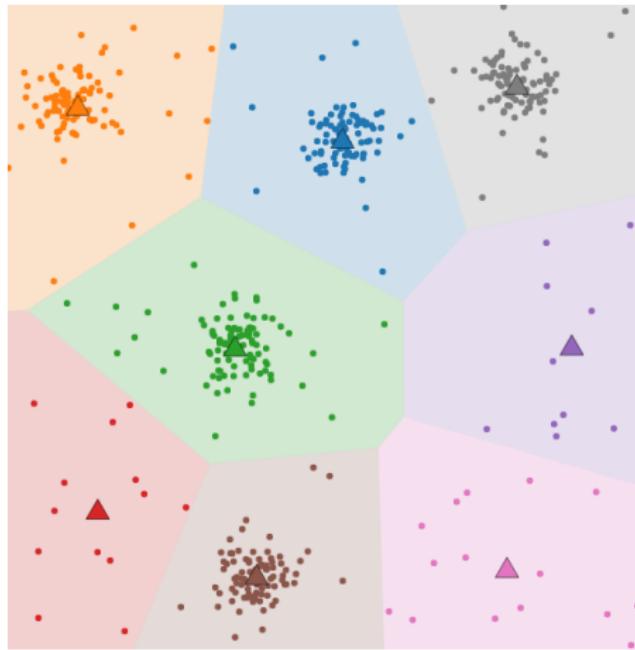
Algoritmus



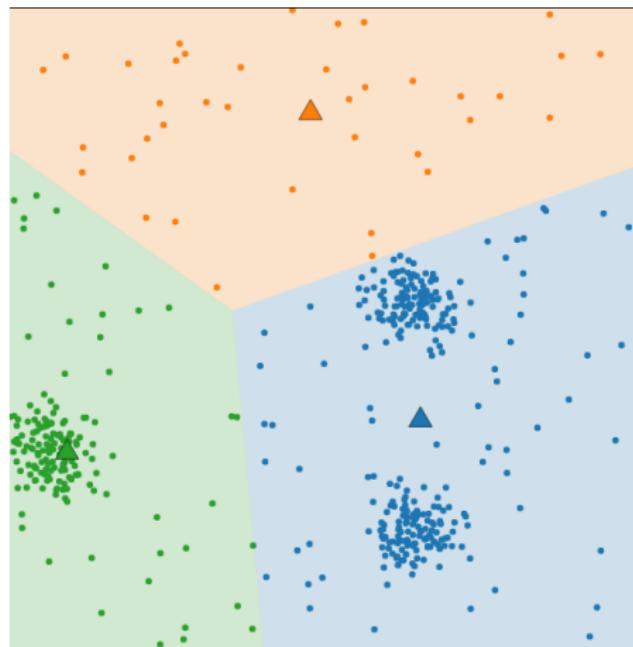
Problémák – túl kicsi k-t adunk meg



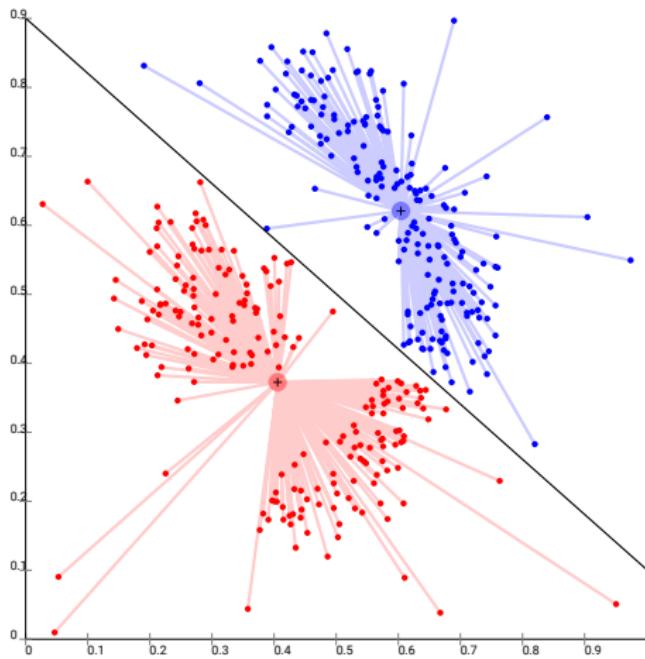
Problémák – túl nagy k-t adunk meg



Problémák – rossz inicializáció



Problémák – sűrűség alapúak a klaszterek



Python példák

- Kép kvantálás: http://scikit-learn.org/stable/auto_examples/cluster/plot_color_quantization.html
- Dokumentumok klaszterezése:
https://scikit-learn.org/0.19/auto_examples/text/document_clustering.html

Tartalom

1 Bevezetés

2 Klaszterezés

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

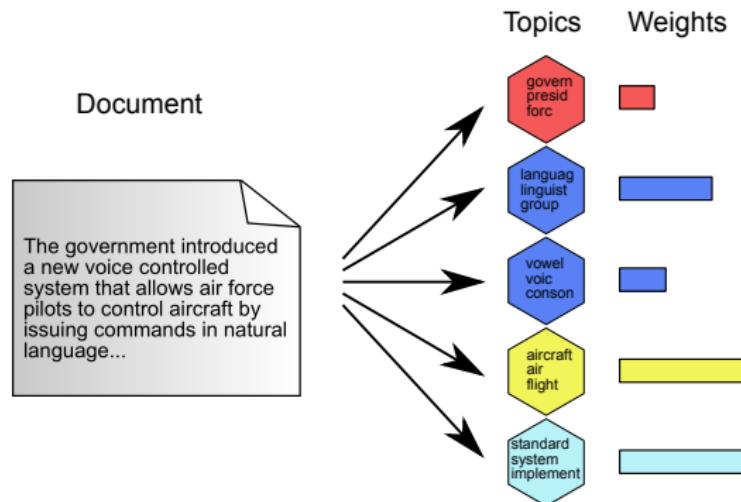
3 Dimenziócsökkentés

- Kovariancia, korreláció
- Főkomponens analízis

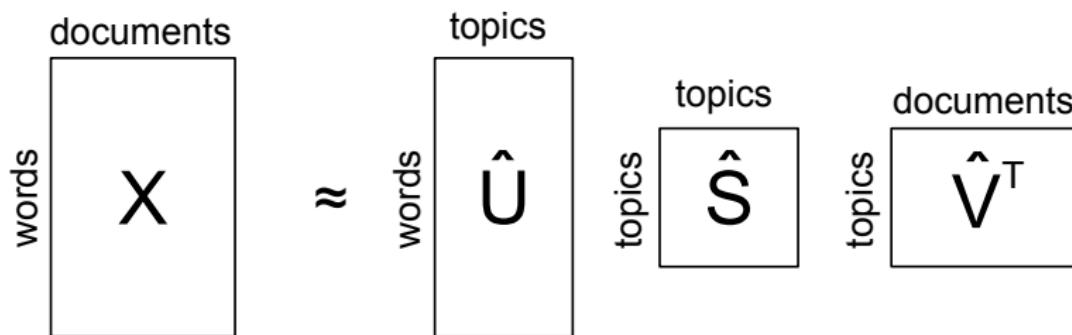
4 Autoenkóderek

Témamodellek

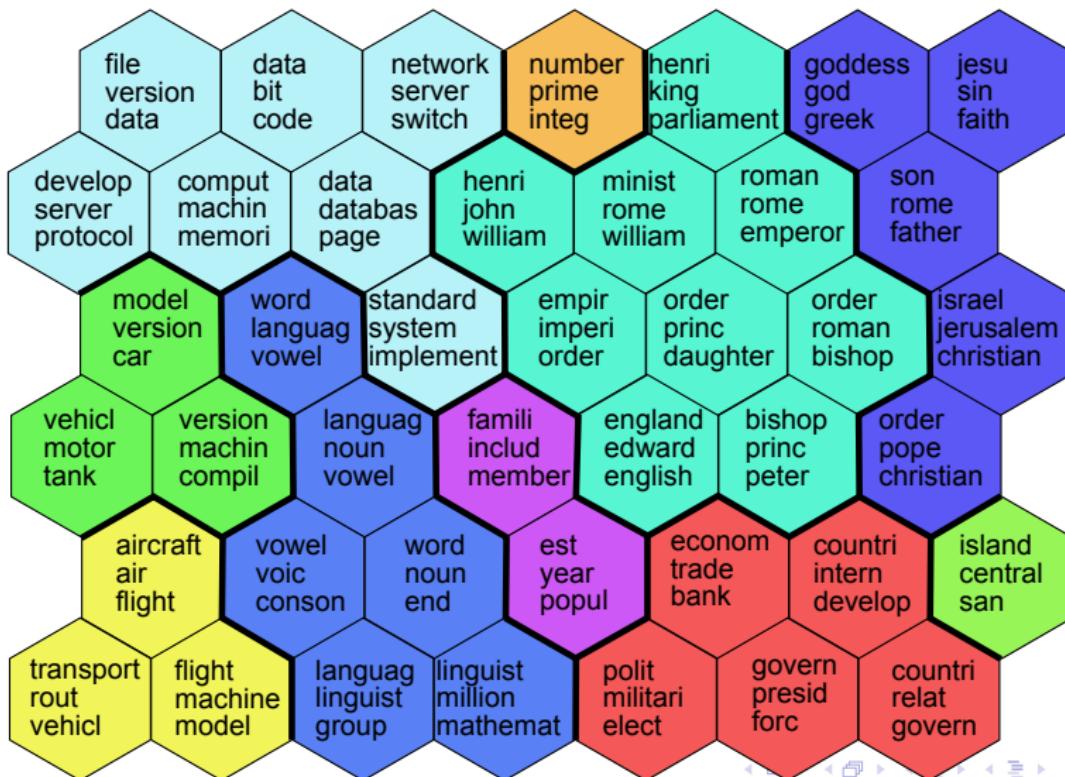
- Soft clusteringre példa: témamodellek
 - Egy dokumentum mennyire szól az egyes témákról
 - Egy témába milyen szavak tartoznak?
 - Pl. Latent Semantic Analysis (SVD)



Latent Semantic Analysis



Kiterjesztés csoportritka regularizációval



Példa: kakukktojás játék

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

Tartalom

1 Bevezetés

2 Klaszterezés

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

3 Dimenziócsökkentés

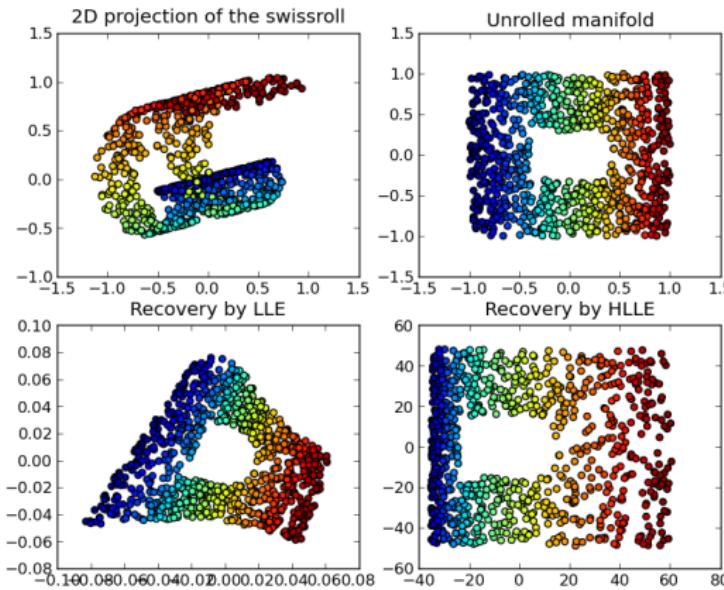
- Kovariancia, korreláció
- Főkomponens analízis

4 Autoenkóderek

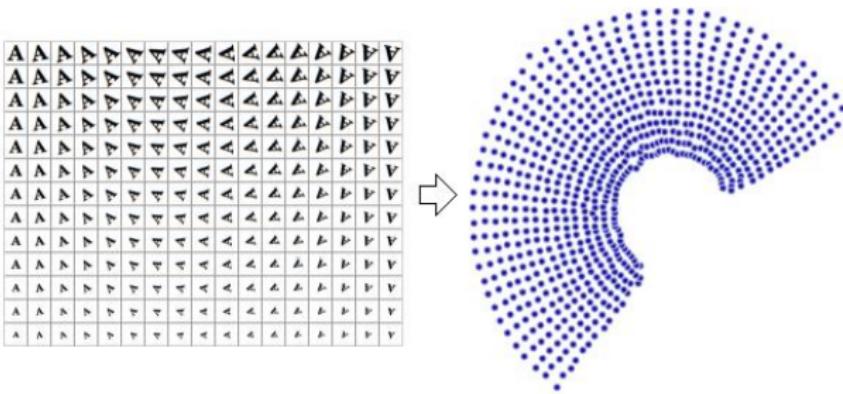
Miért dimenziócsökkentünk?

- 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

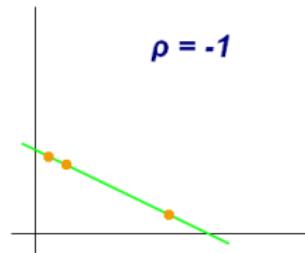
3 Dimenziócsökkentés

- Kovariancia, korreláció
- Főkomponens analízis

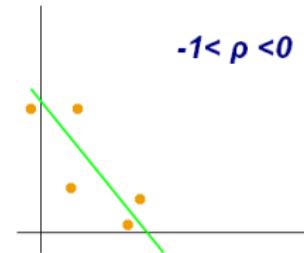
4 Autoenkóderek

Kovariancia, korreláció

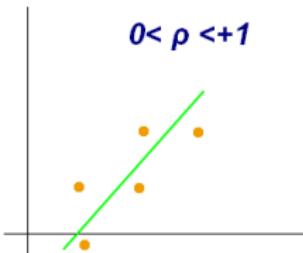
$$\rho = -1$$



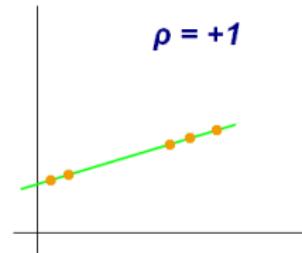
$$-1 < \rho < 0$$



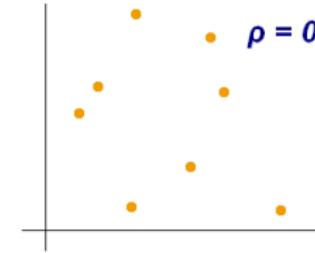
$$0 < \rho < +1$$



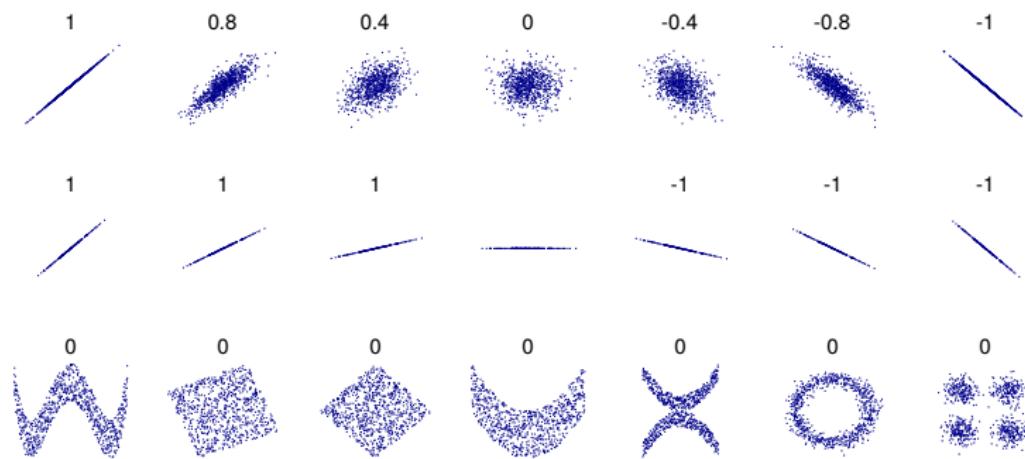
$$\rho = +1$$



$$\rho = 0$$



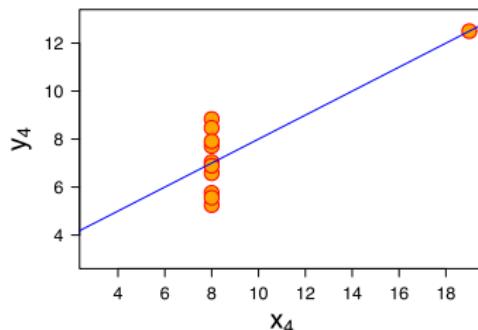
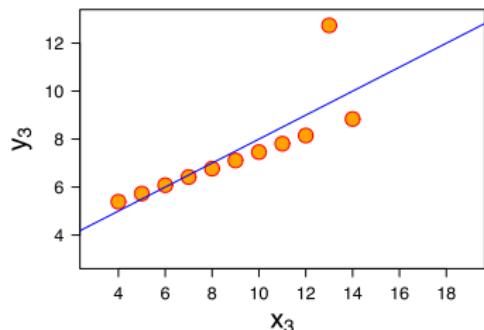
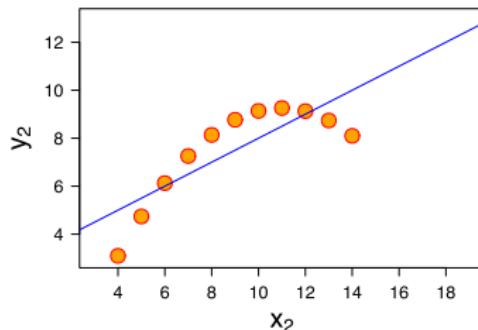
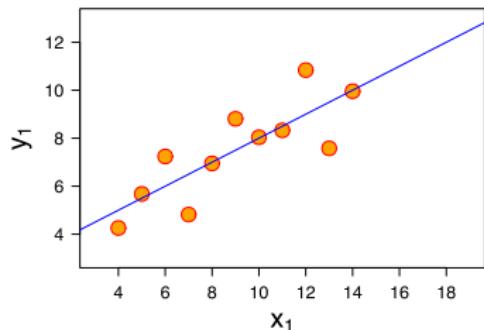
Kovariancia, korreláció



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

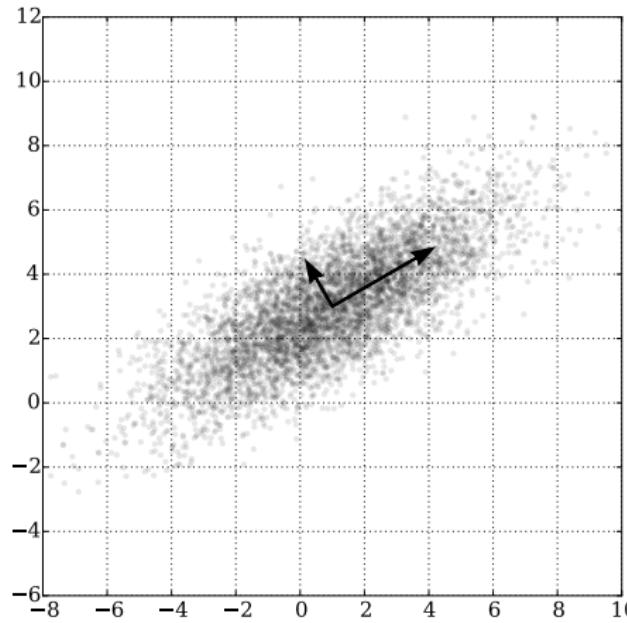
- Hard clustering – k-means
- Soft clustering – témamodellek

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:

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

- Mivel \mathbf{w} egységevektor:

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

- 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

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

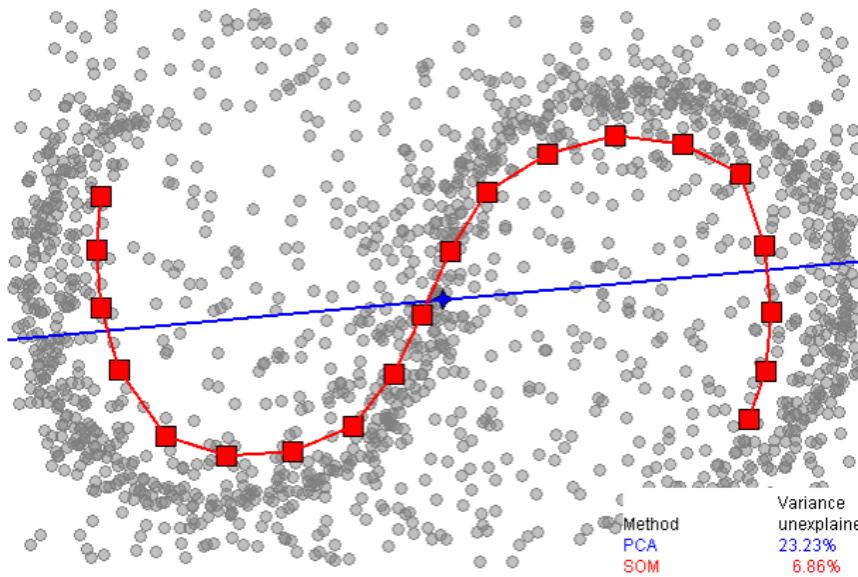
- \mathbf{W} -ben már $\mathbf{X}^T\mathbf{X}$ sajátvektorai vannak. A szinguláris értékek a sajátertékek négyzetgyökei.

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

Tartalom

1 Bevezetés

2 Klaszterezés

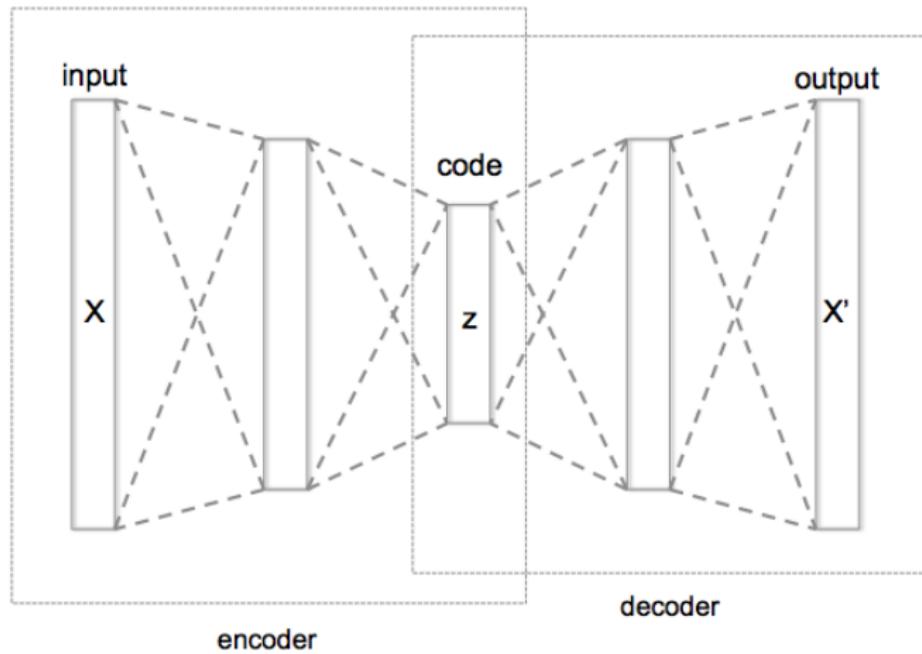
- Hard clustering – k-means
- Soft clustering – témamodellek

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!