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

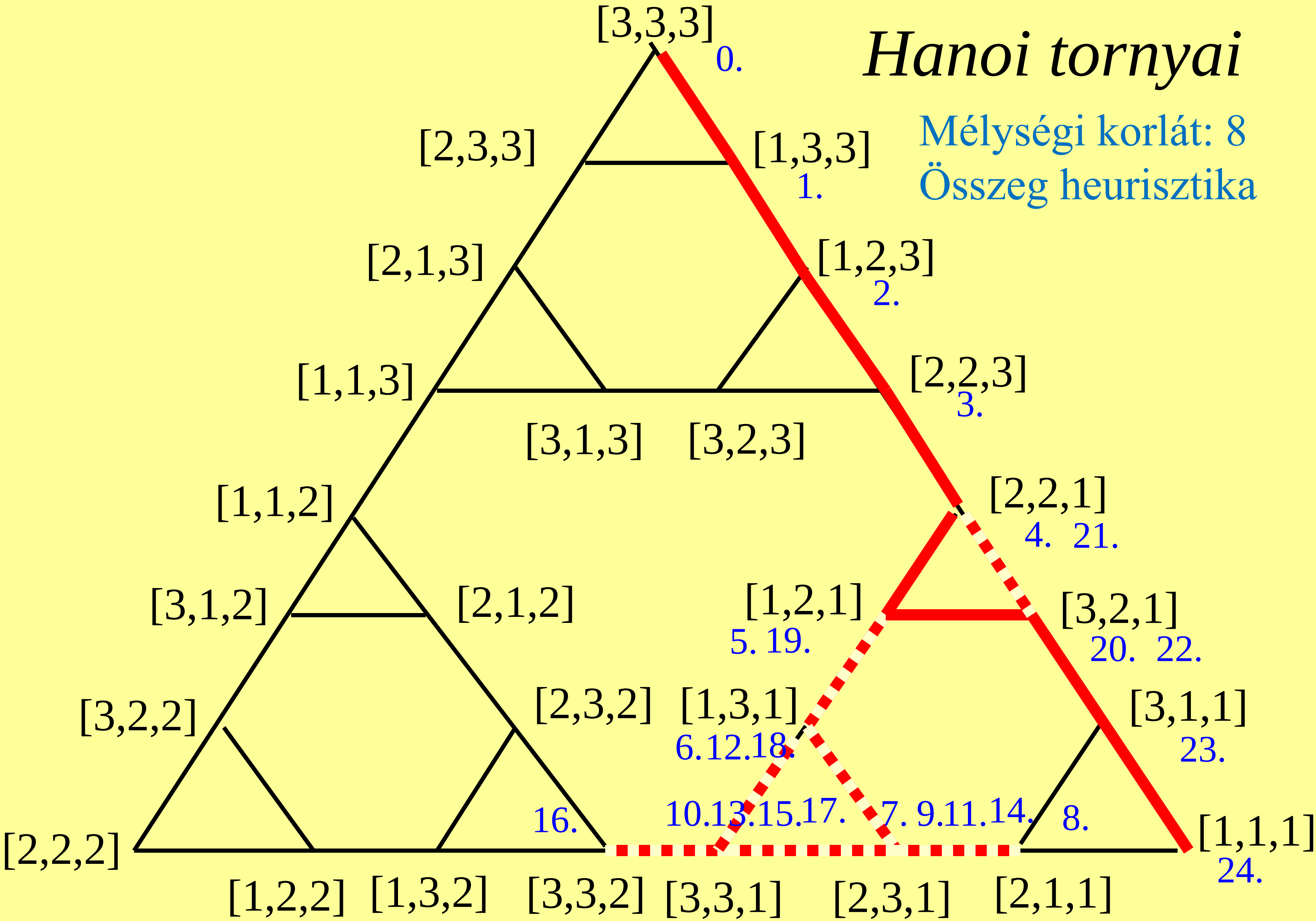
- A legvégső eset, amikor a visszalépést kell választani:
  - zsákutca: az aktuális csúcsból (azaz az aktuális út végpontjából) nem vezet tovább él
  - zsákutca torkolat: az aktuális csúcsból kivezető utak nem vezettek célba
  - kör: az aktuális csúcs szerepel már korábban is az aktuális úton
  - mélységi 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 mindig terminál, és ha létezik megoldás, akkor talál egyet.***

  UI: egy adott startcsúcsból kiinduló útból véges sok van.

- Rekurzív algoritmussal szokták implementálni
  - Indítás: *megoldás* := *VL1*(*startcsúcs*)

ADAT := *kezdeti érték*
while ¬*terminálási feltétel*(ADAT) **loop**
SELECT SZ FROM *alkalmazható szabályok*
ADAT := SZ(ADAT)
endloop

VL1

A ~ élek halmaza
A^* ~ véges élsorozatok halmaza

N ~ csúcsok halmaza

Recursive procedure *VL1*(*akt* : *N*) **return** (A^*; *hiba*)
1. **if** *cél*(*akt*) **then** **return**(*nil*) **endif**
2. **for** $\forall$*új* $\in \Gamma$(*akt*) **loop**
3. *megoldás* := *VL1*(*új*)
4. **if** *megoldás* $\neq$ *hiba* **then**
5. **return**(*fűz*((*akt*,*új*), *megoldás*) **endif**
6. **endloop**
7. **return**(*hiba*)

end

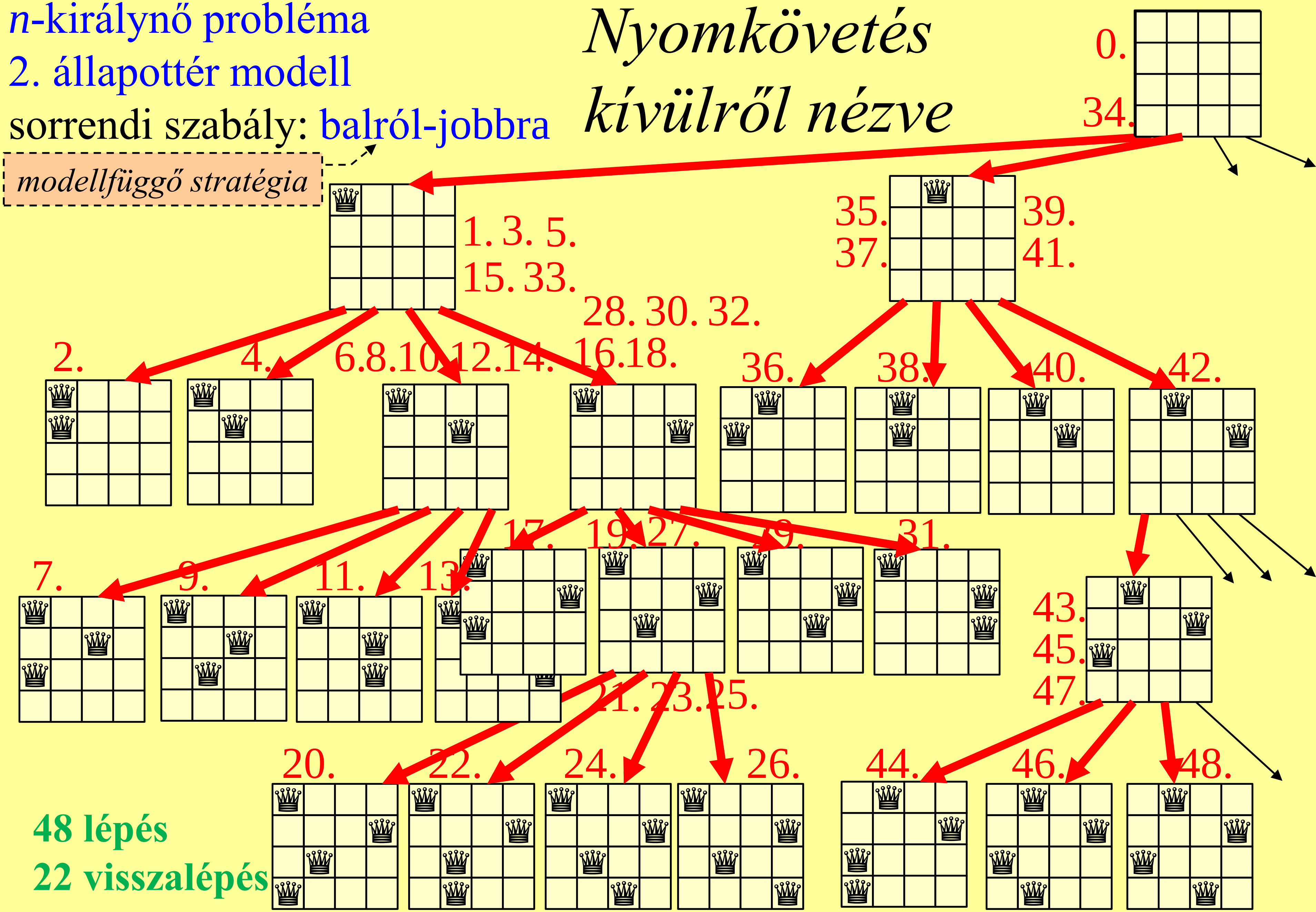
n-királynő probléma

2. állapottér modell

sorrendi szabály: balról-jobbra

modellfüggő stratégia

Nyomkövetés kívülről nézve

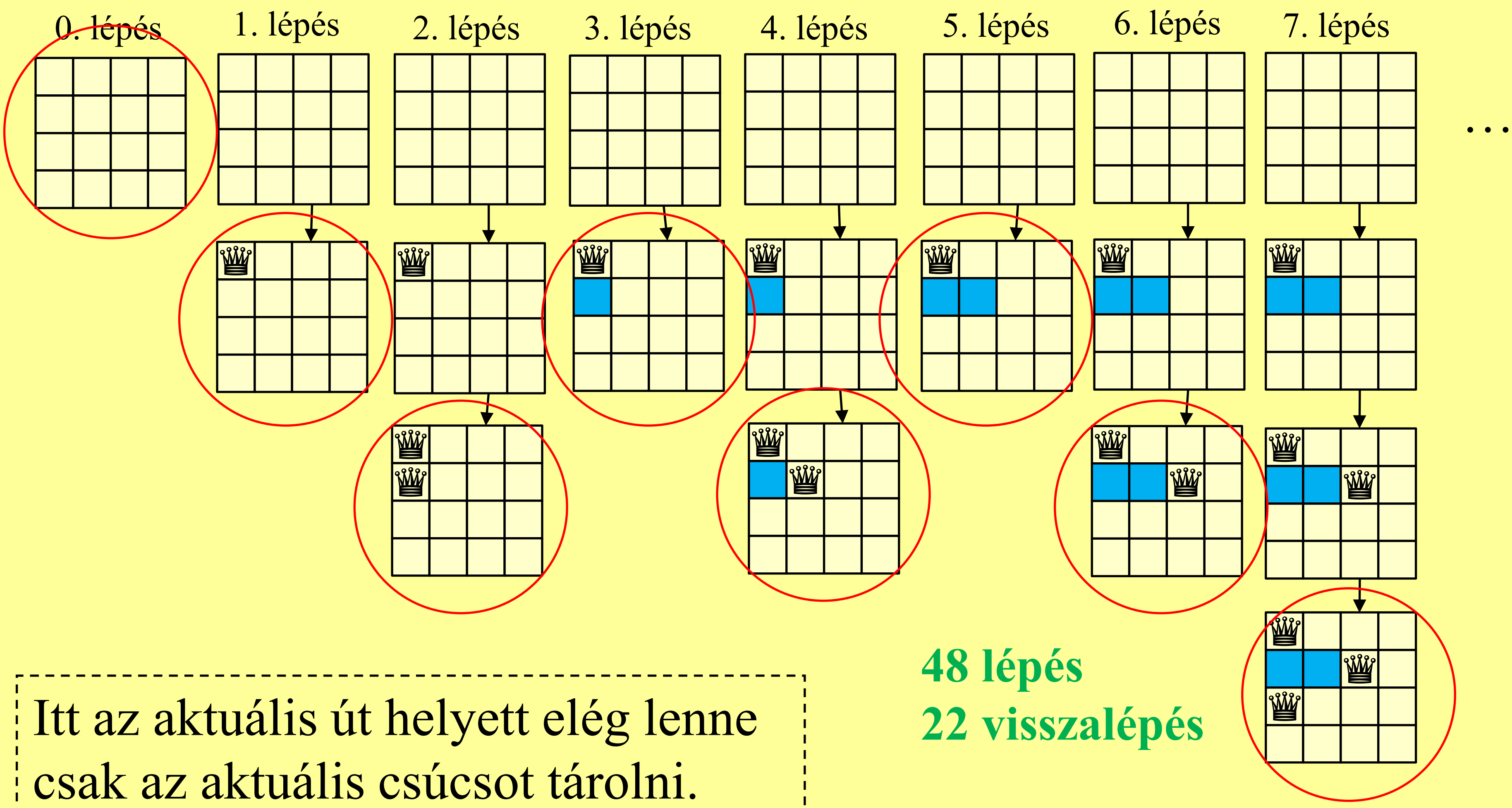


48 lépés

22 visszalépés

n-királynő probléma
2. állapottér modell
sorrendi szabály: balról-jobbra

Nyomkövetés belülről nézve



Itt az aktuális út helyett elég lenne csak az aktuális csúcsot tárolni.

48 lépés
22 visszalépés

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

VL1

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

D_i ={*i*-dik sor szabad mezői}

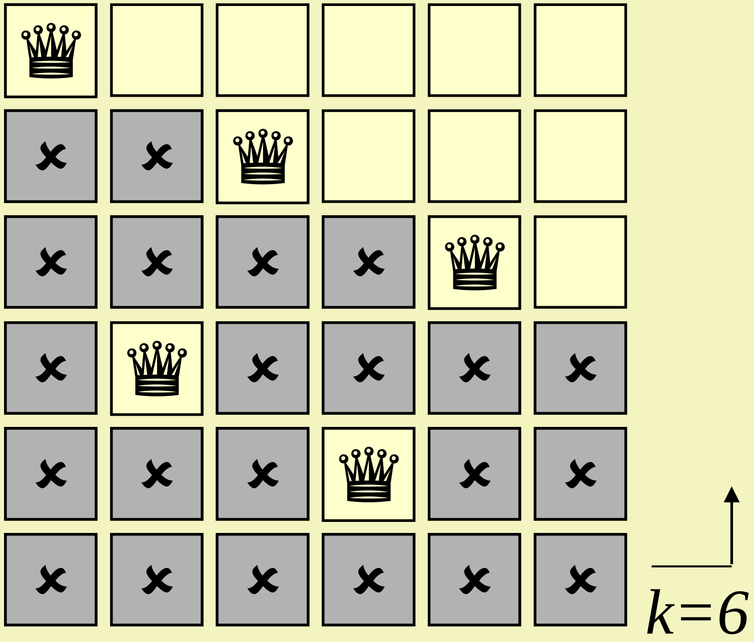
Töröl(*i*,*k*) : törli D_i azon mezőit, amelyeket a *k*-dik sor királynője üt.

k-dik lépésben végzett törlések:

for *i*=*k*+*1* .. *n* **loop**
 Töröl(i,k)
endloop

+

if D_k = ∅ **then** visszalép



Forward Checking

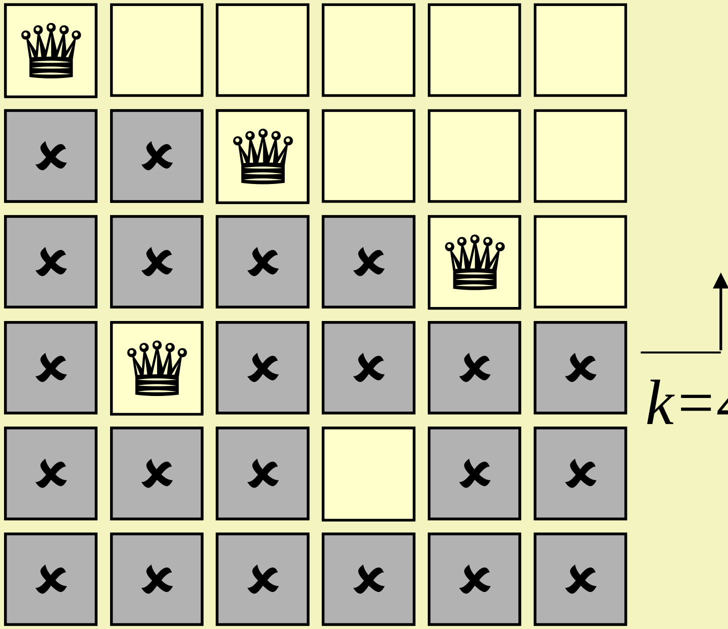
***FC* algoritmus**:*VL1*+vágó szabály

k-dik lépésben végzett törlések

+

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

then *visszalép*



$k=4$

$D_6 = \emptyset$

Partial Look Forward

PLF **algoritmus**:*VL1*+vágó szabály

k-dik lépésben végzett törlések

\+

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

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

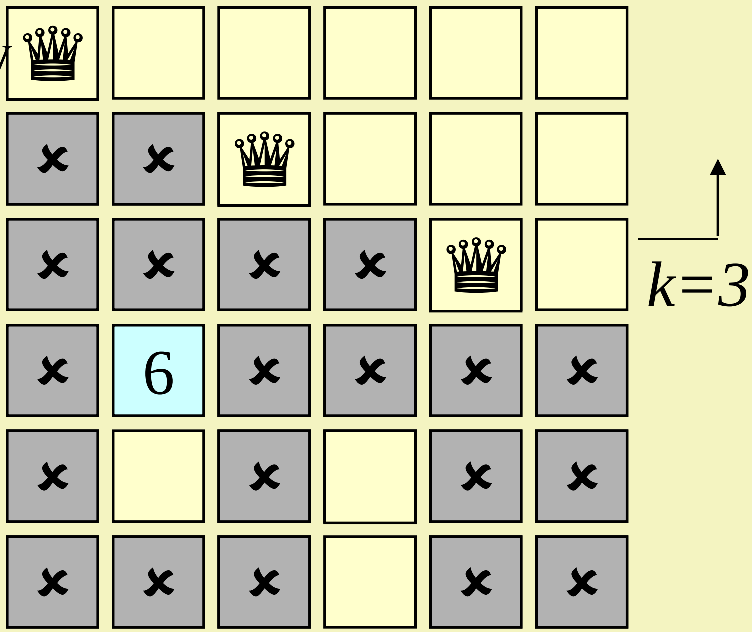
    Szűr(i, j)   ($i<j$)

  endloop

endloop

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

then *visszalép*



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

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

Look Forward

***LF* algoritmus**:*VL1*+vágó szabály

k-dik lépésben végzett törlések

\+

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

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

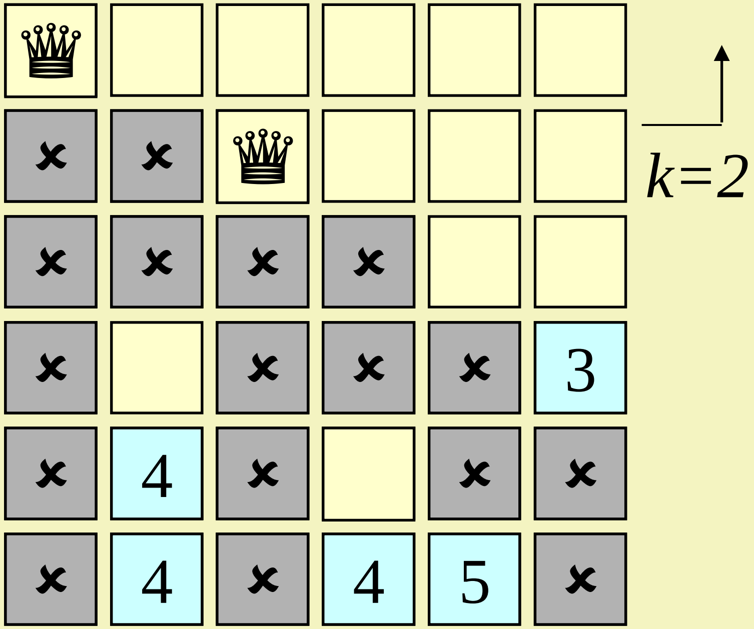
Szűr(i, j)

endloop

endloop

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

then *visszalép*



$i = 4,\ j = 3$

$i = 5,\ j = 4$

$i = 6,\ j = 4$

$i = 6,\ j = 5$

$D_6 = \emptyset$

Az n-királynő probléma újabb modellje

- A vágó szabályok bemutatása közben rátaláltunk az *n*-királynő probléma egy új modelljére:
  - Tekintsük a $D_1 , \ldots , D_n$ halmazokat, ahol D_i az *i*-dik sor szabad mezőinek oszlopszámai, kezdetben $D_i =\{1 \ldots n\}$.
  - Keressük azt az $(x_1 ,\ldots, x_n) \in D_1 \times \ldots \times D_n$ 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 visszalépésre 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 korlátkielégítési modell bináris korlátjaival fogalmazhatók meg anélkül, hogy a korlátok jelentését ismernünk kellene:

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

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

- 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 töltsük ki.

Második változat: VL2

- A visszalépéses algoritmus második változata az, amikor a visszalépés feltételei közül mindet 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*>)

VL2

ADAT := *kezdeti érték*
while ¬*terminálási feltétel*(ADAT) **loop**
    SELECT SZ FROM *alkalmazható szabályok*
    ADAT := SZ(ADAT)
endloop

Recursive procedure *VL2*(*út* : N^*) **return** (A^*; *hiba*)
    akt := *utolsó_csúcs*(*út*)
    if *cél*(*akt*) **then** **return**(*nil*) **endif**
    if *hossza*(*út*) $\geq$ *korlát* **then** **return**(*hiba*) **endif**
    if *akt* $\in$ *maradék*(*út*) **then** **return**(*hiba*) **endif**
    for $\forall$*új* $\in \Gamma$(*akt*) $-\ \pi$(*akt*) **loop**
        megoldás := *VL2*(*fűz*(*út*, *új*))
        if *megoldás* $\neq$ *hiba* **then**
            return(*fűz*((*akt*,*új*),*megoldás*)) **endif**
    endloop
    return(*hiba*)
end

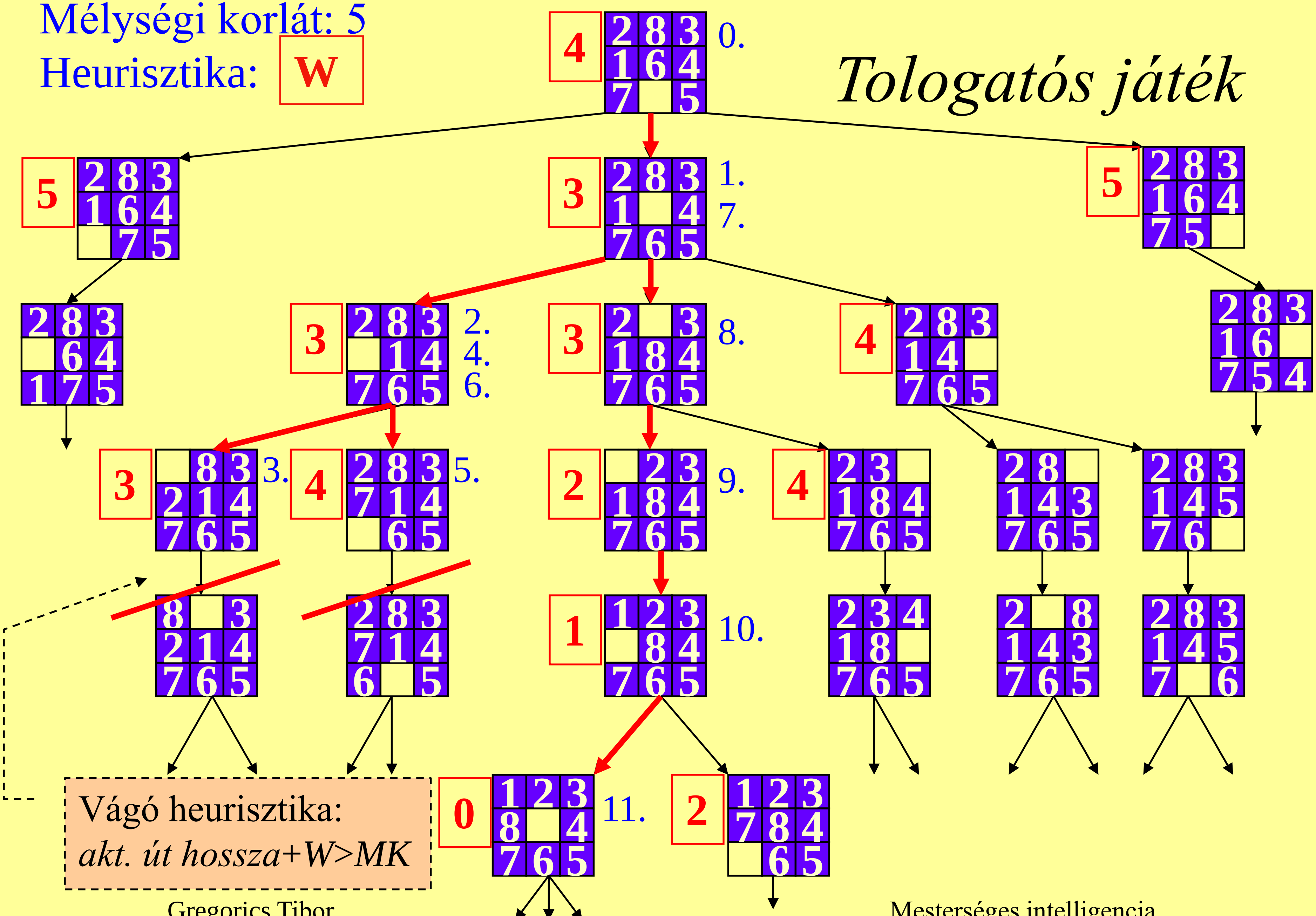
Mélységi korlát szerepe

- A mélységi korlát ellenőrzése önmagában is biztosítja a terminálást körfigyelés nélkül.
  - Ilyenkor nem kell a rekurzív hívásnál a teljes aktuális utat átadni : elég az aktuális csúcsot, annak szülőjét (ez kettő hosszú körök kiszűréséhez kell), és az aktuális út hosszát.
  - Hatékonyság nő: nem kell utakat tárolni (csökkent a memória igény), nem végzünk körfigyelést (csökken egy lépés futási ideje), viszont ha a mélységi korlátnál rövidebb körök 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



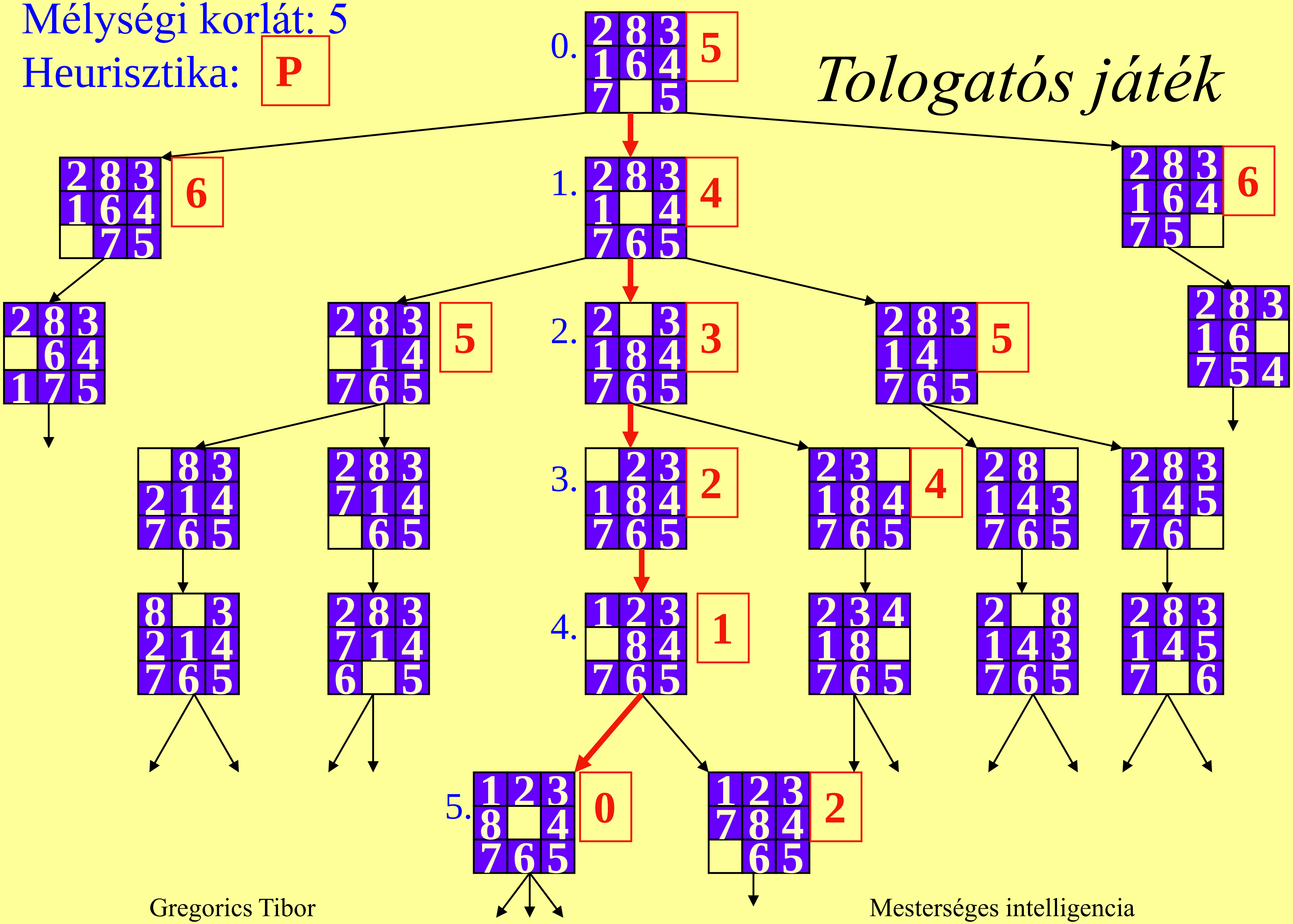
Vágó heurisztika:

akt. út hossza+W>MK

Mélységi korlát: 5

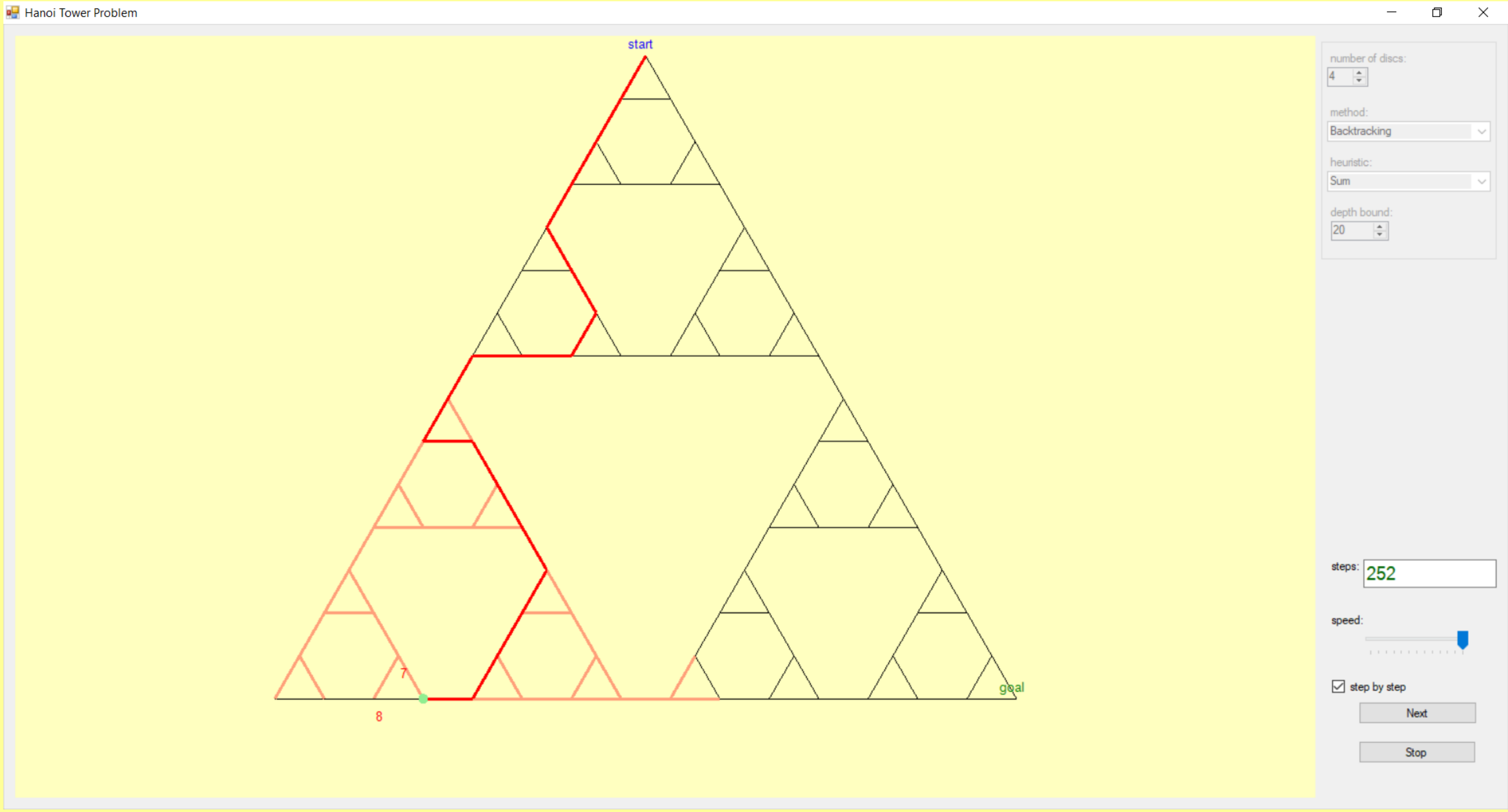
Heurisztika: P

Tologatós játék



Értékelés

- ELŐNYÖK
  - mindig 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
start
goal
7
8
number of discs:
4
method:
Backtracking
heuristic:
Sum
depth bound:
20
steps:
252
speed:
step by step
Next
Stop