

## 2. Visszalépéses keresés



# *Visszalépéses keresés*

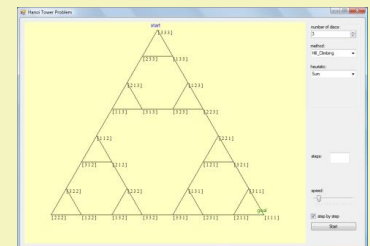
- ❑ A visszalépéses keresés egy olyan KR, amely
  - globális munkaterülete:
    - egy **út** a startcsúcsból az aktuális csúcsba (az útról leágazó még ki nem próbált élekkel együtt)
      - kezdetben a startcsúcsot tartalmazó nulla hosszúságú út
      - terminálás célcsúcs elérésekor vagy a startcsúcsból 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*

- ❑ **zsákutca**: az aktuális csúcsból (azaz az aktuális út végpontjából) nem vezet tovább él
- ❑ **zsákutca torkolat**: az aktuális csúcsból kivezető utak nem vezetnek 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 egy csúcsból kivezető élek vizsgálatára
  - **vágó szabállyal**: megjelöli egy csúcs azon kivezető éleit, amelyeket nem érdemes megvizsgálni
- ❑ Ezek a szabályok lehetnek
  - modellfüggő vezérlési stratégiák (a probléma modelljének sajátosságaiból származó ötlet)
  - heurisztikák (a megoldandó problémától származó információra támaszkodó ötlet)



# Első változat: VL1

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

ADAT := *kezdeti érték*

```
while  $\neg$ terminálási feltétel(ADAT) loop  
    SELECT SZ FROM alkalmazható szabályok  
    ADAT := SZ(ADAT)  
endloop
```

*VL1*

$A \sim$  élek

$A^* \sim$  véges élsorozat

$N \sim$  csúcsok

**Recursive procedure** *VL1*(*akt* :  $N$ ) **return** ( $A^*$ ; *hiba*)

1.     **if** *cél*(*akt*) **then** **return**(*nil*) **endif**

2.     **for**  $\forall \acute{u}j \in \Gamma(\acute{a}kt)$  **loop**

$\Gamma(\acute{a}kt) \sim$  *akt gyermekei*

3.         *megoldás* := *VL1*(*új*)

4.         **if** *megoldás*  $\neq$  *hiba* **then**

5.             **return**(*fűz*((*akt*,*új*), *megoldás*) **endif**

6.     **endloop**

7.     **return**(*hiba*)

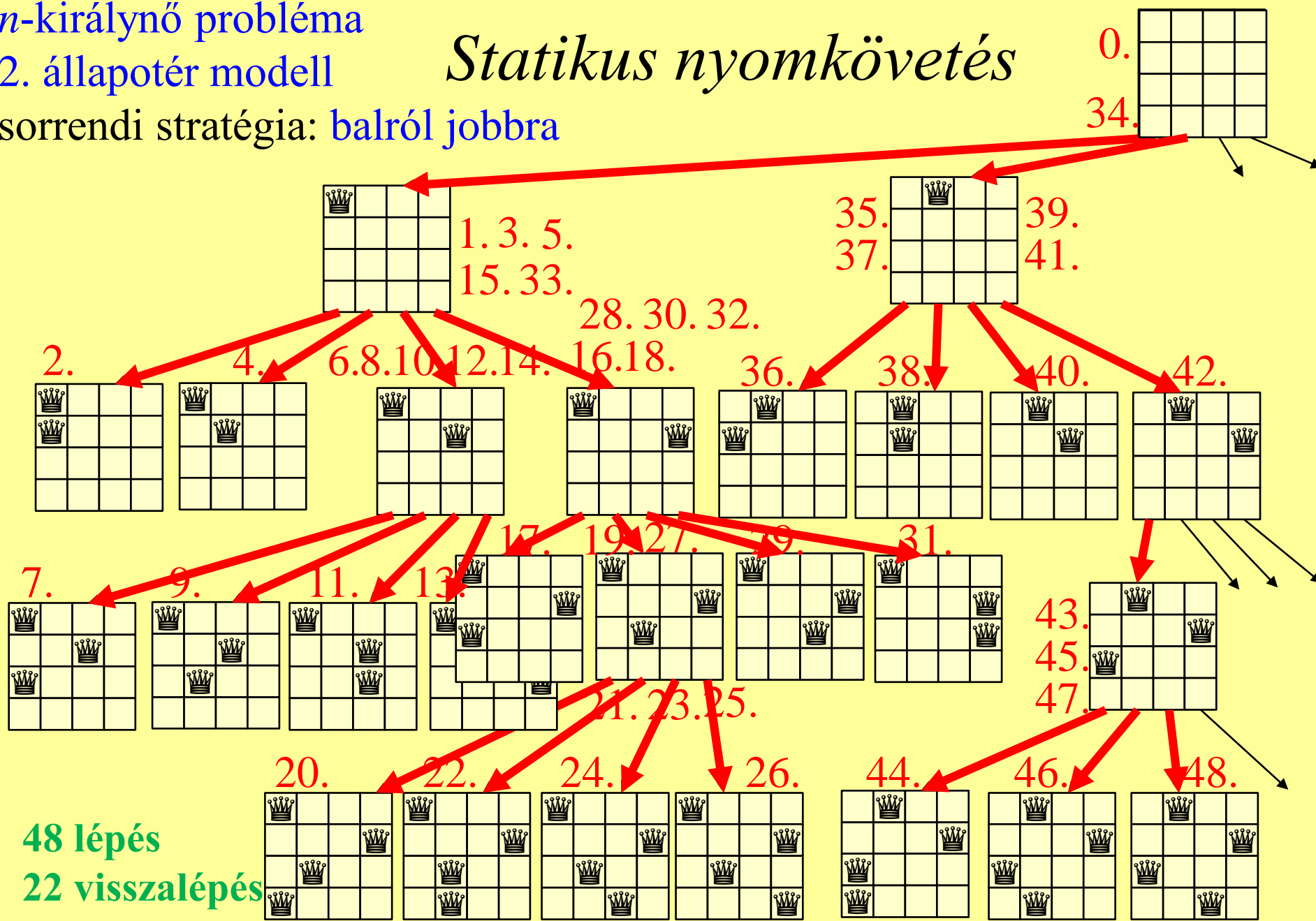
**end**

*n*-királynő probléma

2. állapotér modell

sorrendi stratégia: balról jobbra

*Statikus nyomkövetés*



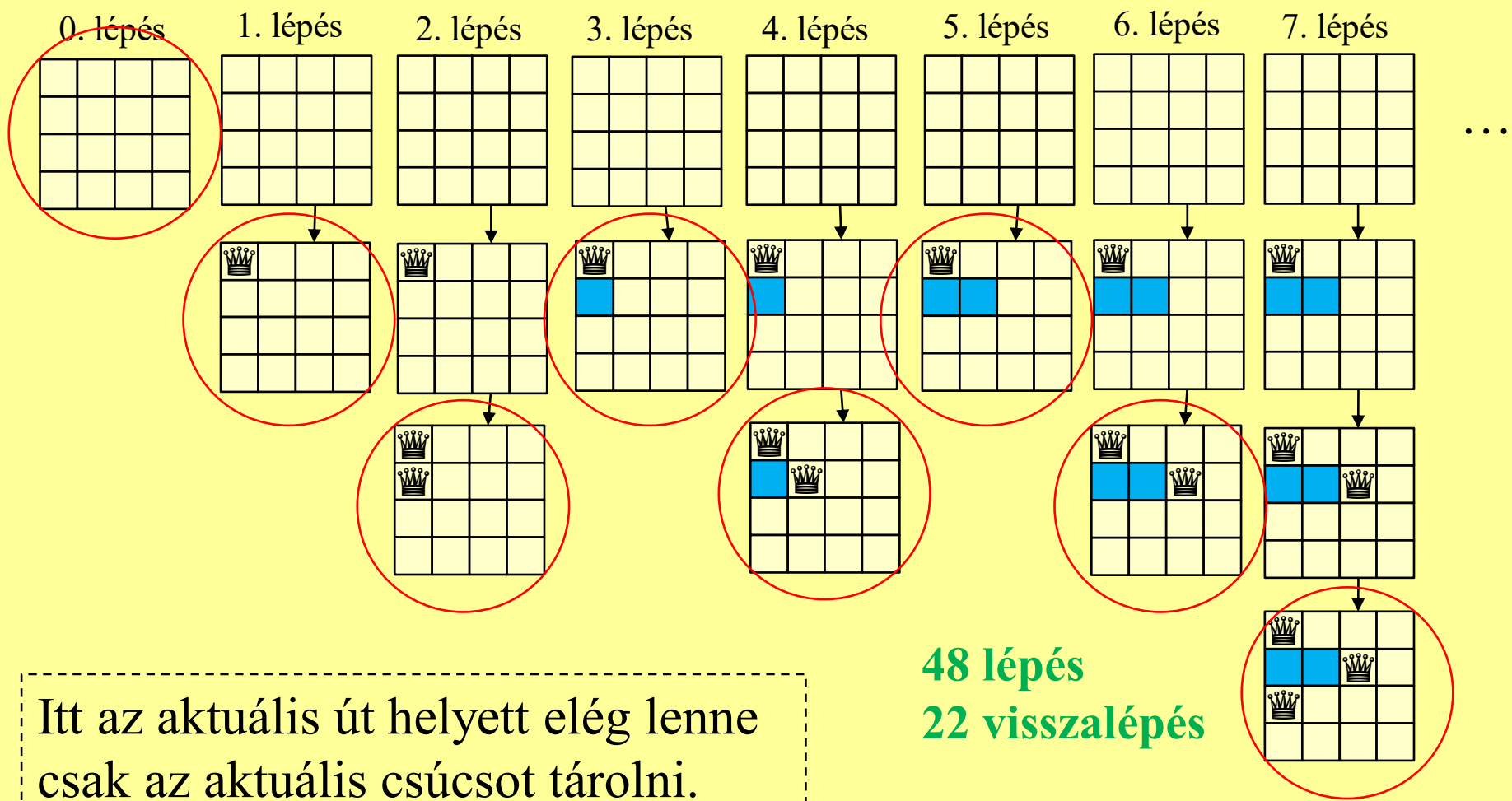
48 lépés  
22 visszalépés

*n*-királynő probléma

2. állapotér modell

sorrendi stratégia: balról jobbra

*Dinamikus nyomkövetés*






# Sorrendi heurisztikák az $n$ -királynő problémára

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

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

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

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

	x	x	x
x	x	3	2
x		x	
x			x

# Heurisztikák az $n$ -királynő problémára

**diagonális + bal-jobb:**

8 lépés  
2 visszalépés

4	♔	3	4
	4	4	♔
♔	4	4	3
4		♔	4

**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. állapotér modell

sorrendi stratégia: balról jobbra

VL1

*heurisztika nélkül*

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

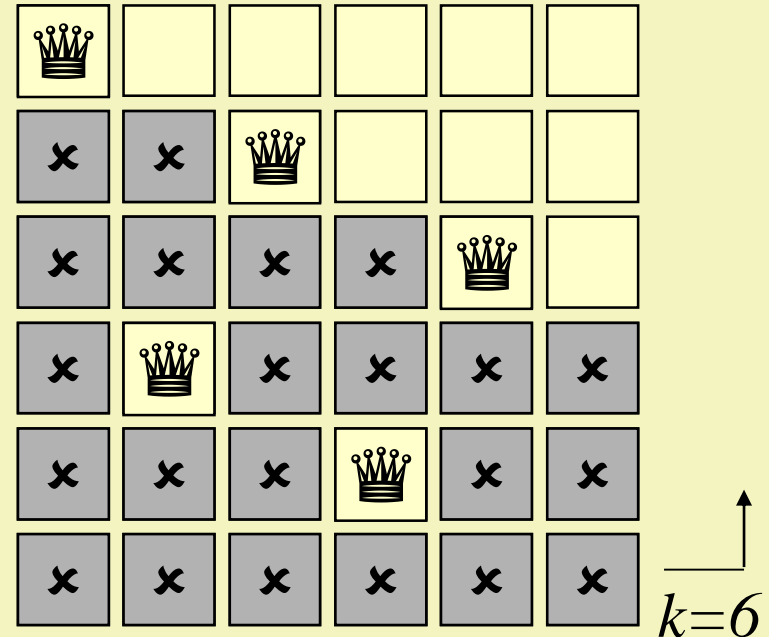
A  $k$ -adik királynő elhelyezése után a hátralevő üres sorokból töröljük az ütésbe került szabad mezőket.

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

*Töröl*( $i, k$ )

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

VL1: *if*  $D_k = \emptyset$  *then* visszalép



# Forward Checking

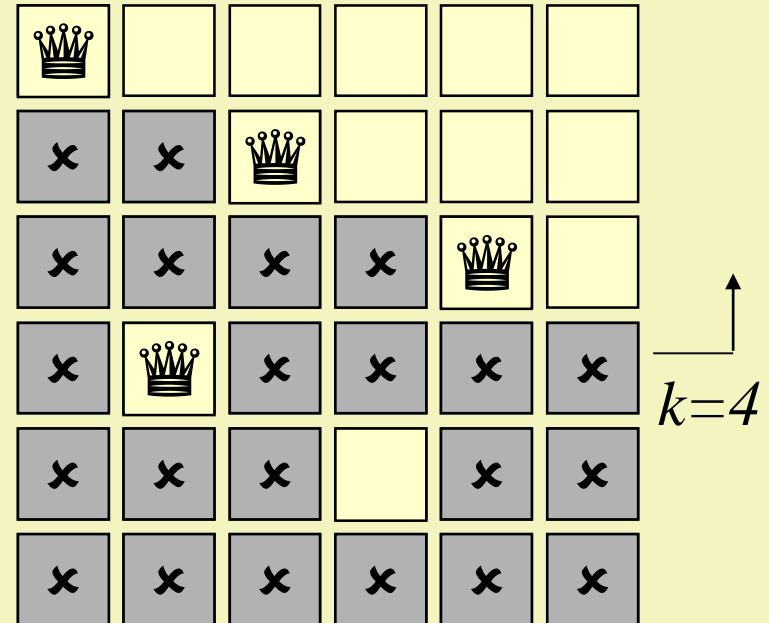
**FC algoritmus:**

*VL1*

+

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

**then** *visszalép*



$D_6 = \emptyset$

# Partial Look Forward

**PLF algoritmus:**

*VL1*

+

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

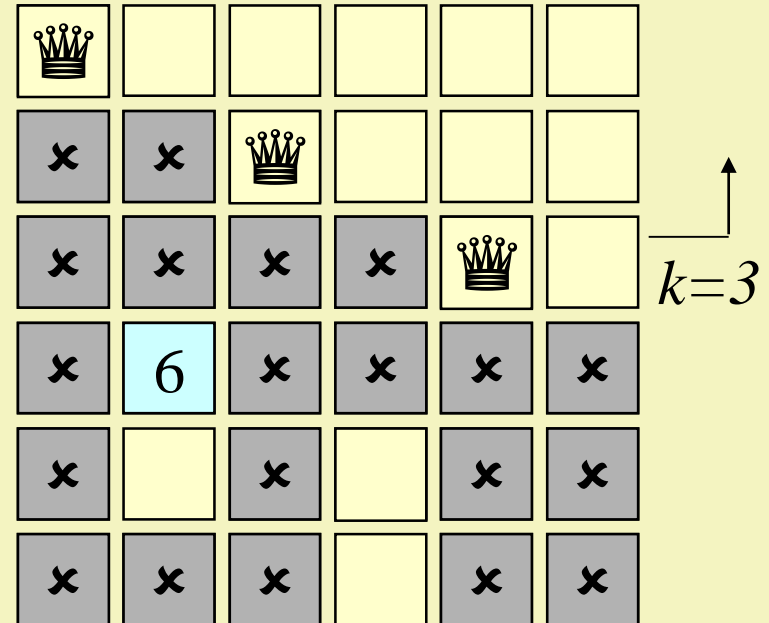
**for**  $j=i+1 \dots n$  **loop** ( $i < j$ )

*Szűr*( $i, j$ )

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

**then** *visszalép*

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



# Look Forward

**LF algoritmus:**

*VL1*

+

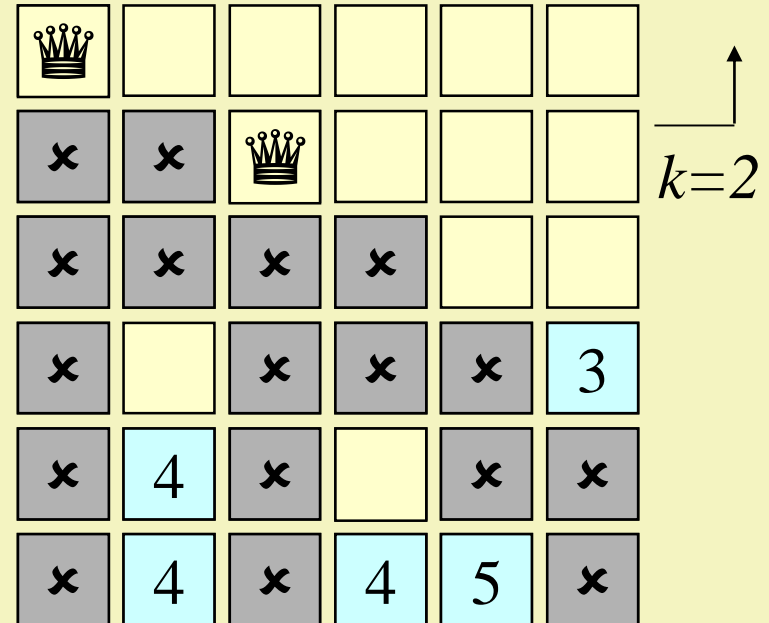
**for**  $i=k+1 .. n$  **loop**

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

*Szűr*( $i, j$ )

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

**then** *visszalép*



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

$i = 5, j = 4$

$i = 6, j = 4$

$i = 6, j = 5$

# Az $n$ -királynő probléma új reprezentációs modellje

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

# *Bináris korlát-kielégítési modell*

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



# *Modellfüggő vezérlési stratégia*

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

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

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

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

# Második változat: VL2

- ❑ A visszalépéses algoritmus második változata az, amikor a visszalépés feltételei közül mindet beépítjük a kereső rendszerbe.
- ❑ Bebizonyítható: *A VL2 &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 (VL2) adjuk meg
  - Indítás:  $megoldás := VL2(<startcsúcs>)$

ADAT := *kezdeti érték*

**while**  $\neg$ *terminálási feltétel*(ADAT) **loop**  
     SELECT SZ FROM *alkalmazható szabályok*  
     ADAT := SZ(ADAT)  
**endloop**

**Recursive procedure**  $VL2(\acute{u}t : N^*)$  **return** ( $A^*$ ; *hiba*)

1.       $akt := utolsó\_csúcs(\acute{u}t)$
2.      **if**  $cél(akt)$  **then return**(*nil*) **endif**
3.      **if**  $hossza(\acute{u}t) \geq korlát$  **then return**(*hiba*) **endif**
4.      **if**  $akt \in maradék(\acute{u}t)$  **then return**(*hiba*) **endif**
5.      **for**  $\forall \acute{u}j \in \Gamma(akt) - \pi(akt)$  **loop**
6.           $megoldás := VL2(fűz(\acute{u}t, \acute{u}j))$
7.          **if**  $megoldás \neq hiba$  **then**
8.              **return**( $fűz((akt, \acute{u}j), megoldás)$ ) **endif**
9.      **endloop**
10.     **return**(*hiba*)

**end**

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

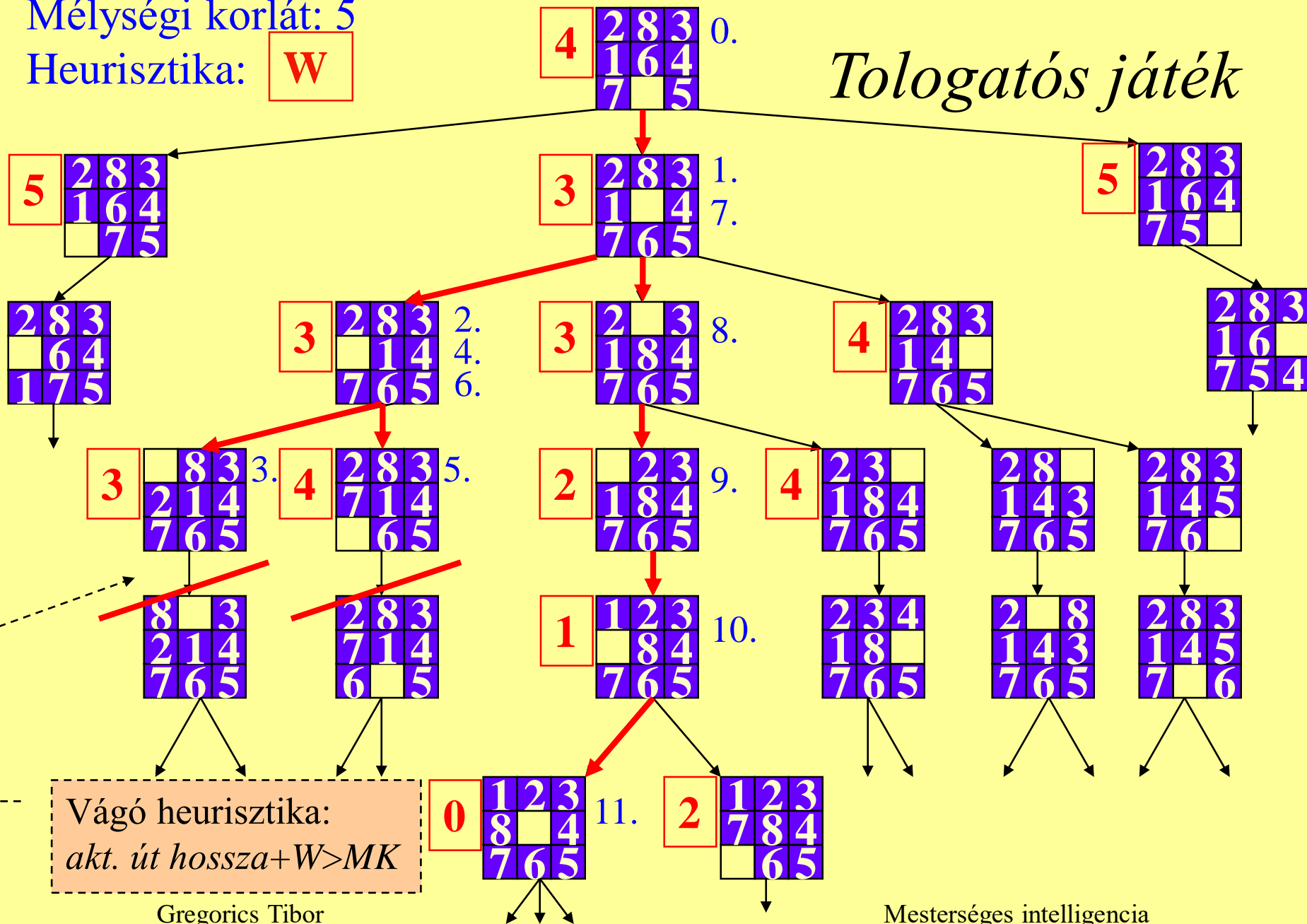
# *Mélységi korlát szerepe*

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

Mélységi korlát: 5

Heurisztika: **W**

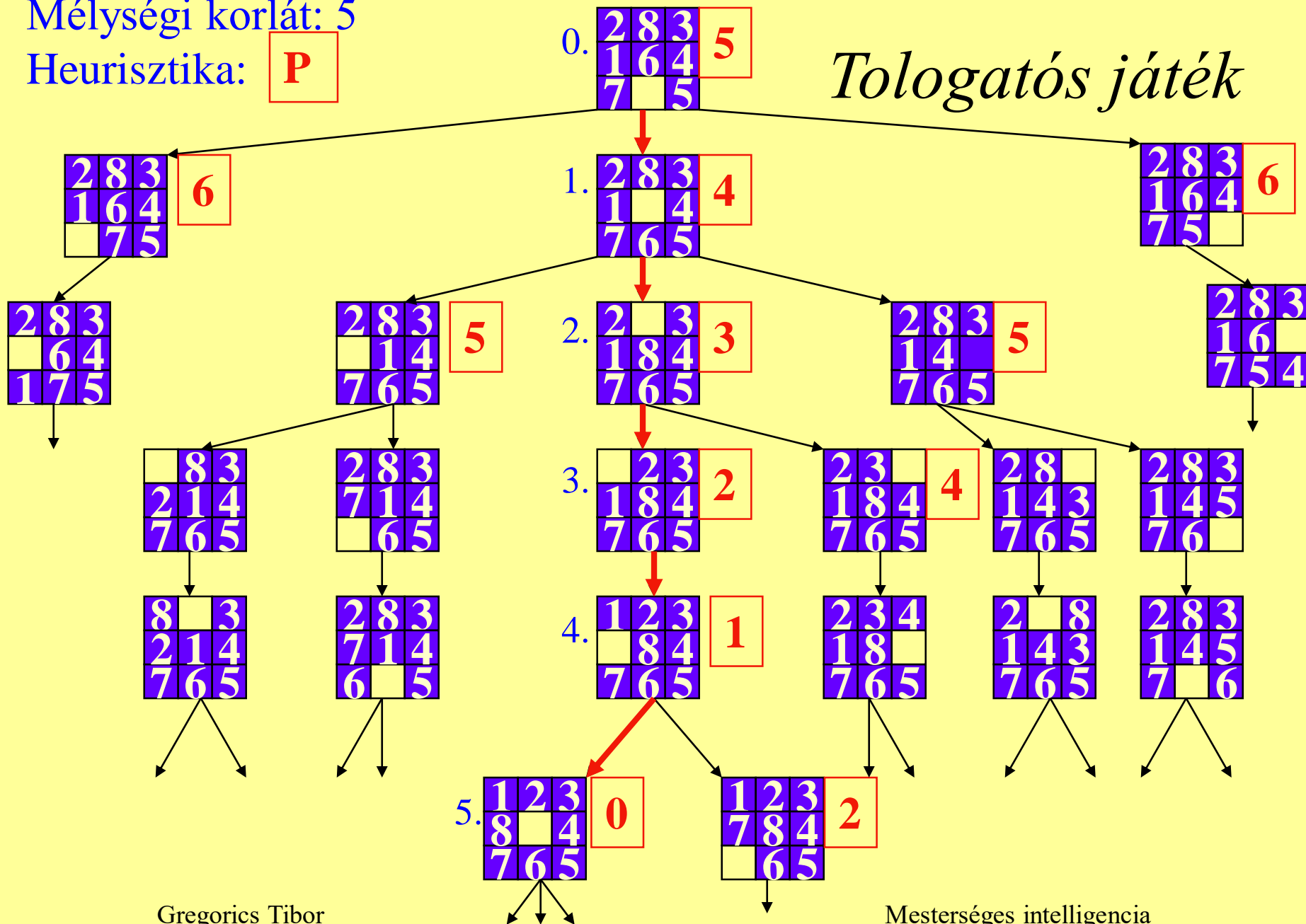
# Tologatós játék



Mélységi korlát: 5

Heurisztika: **P**

# *Tologatós játék*



## □ ELŐNYÖK

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

## □ HÁTRÁNYOK

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