Modellezés

1. Állapottér modell

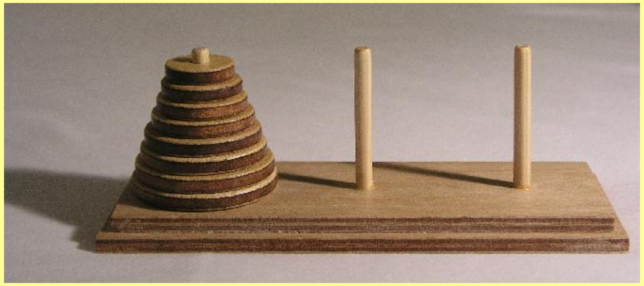
- ***Állapottér***: a probléma leírásához szükséges adatok által felvett érték-együttesek (azaz állapotok) halmaza
  - az állapot többnyire egy összetett szerkezetű érték
  - gyakran egy bővebb alaphalmazzal és egy azon értelmezett invariáns állítással definiáljuk
- ***Műveletek***: állapotból állapotba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

Állapottér modell gráfreprezentációja

Állapottér modell	Állapot-gráf	
◦ állapot	~	csúcs
◦ művelet hatása egy állapotra	~	irányított él
• egy állapotra véges sok művelet alkalmazható (véges kifokú gráf)		
◦ művelet költsége	~	él költsége
• van a műveltek költségének alsó pozitív korlátja (δ)		
◦ kezdő állapot	~	startcsúcs
◦ célállapot	~	célcsúcs

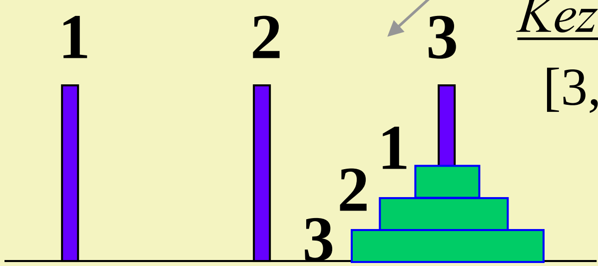
- Gráf-reprezentáció: állapot-gráf, startcsúcs, célcsúcsok

◦ műveletsorozat hatása	~	irányított út
◦ megoldás	~	ir. út a startcsúcsból egy célcsúcsba, vagy maga egy célcsúcs



Hanoi tornyai probléma

állapot ~ korongok egy elhelyezkedése

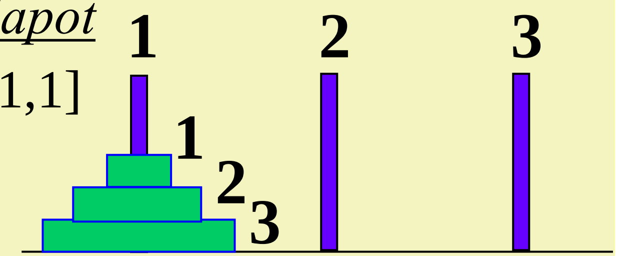


<u>*Kezdőállapot*</u>

[3,3,3]

<u>*Célállapot*</u>

[1,1,1]



<u>*Állapottér*</u>: $\boldsymbol{AT} = \{1,2,3\}^n$

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.

<u>*Művelet*</u>: ***Rak***(*honnan*, *hova*)***:AT→AT***

HA a *honnan* és *hova* rudak <u>léteznek</u> és <u>nem azonosak</u>, és <u>van korong</u> a *honnan* rúdon, és a *hova* rúd vagy <u>üres</u> vagy a legfelső korongja <u>nagyobb</u>, mint a *honnan* rúd legfelső (mozgatandó) korongja

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

this:***AT*** az aktuális állapot

Implementáció

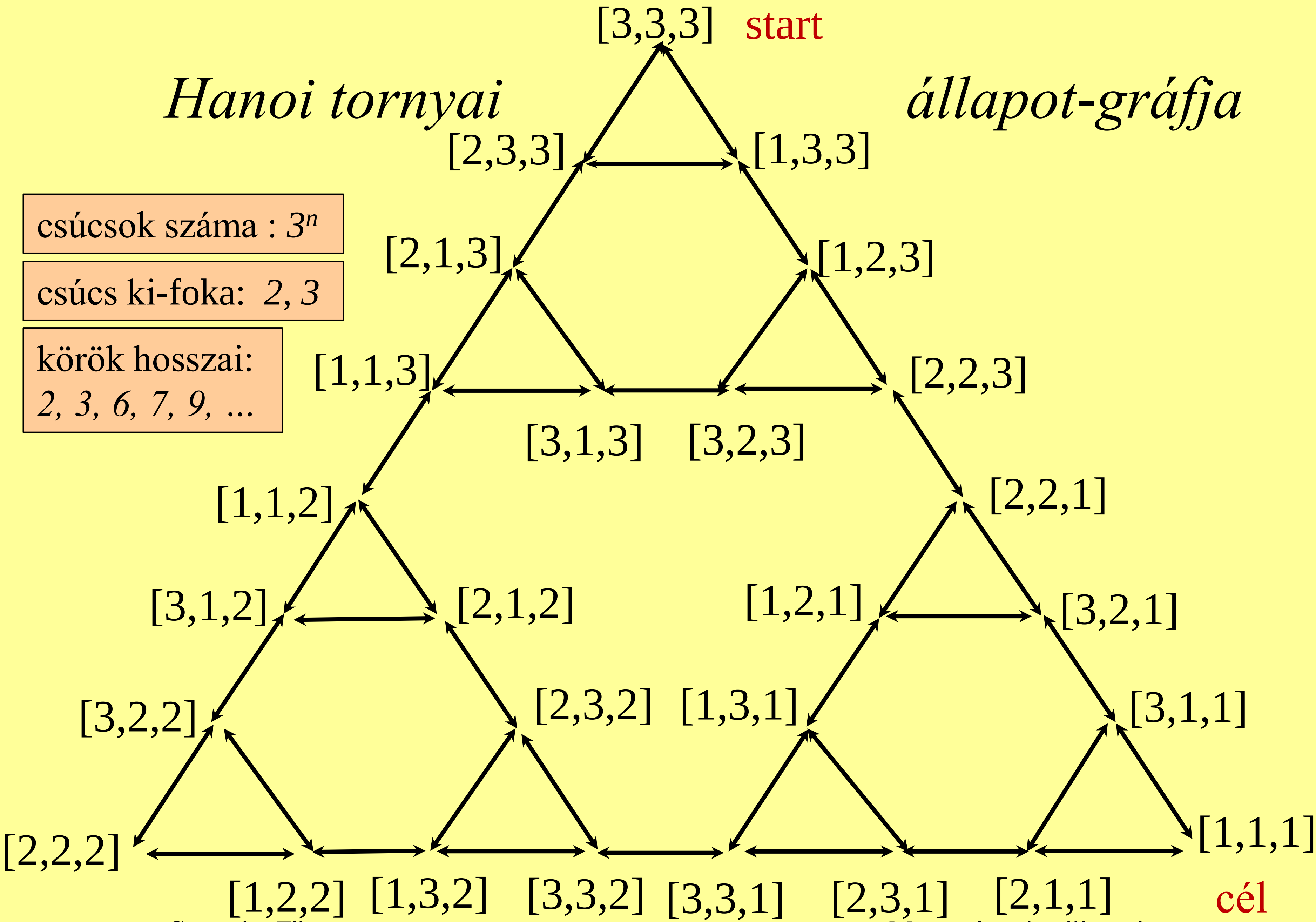
```
template <int  n = 3>
class Hanoi {
      int  _a[n];             // its elements are between 1 and  3
public :
      bool move (int from, int to) {
            if ((from<1 || from>3  || to<1 || to>3) || (from==to)) return false;
            bool  l1; int  i;    //  l1 ~ 'from' is not empty, i ~ 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 *állapot-gráfja*

csúcsok száma : 3^n

csúcs ki-foka: *2, 3*

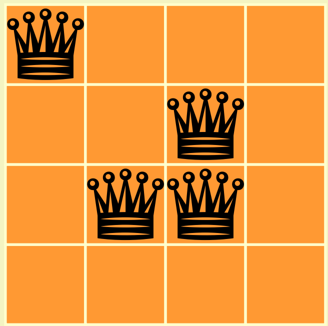
körök hosszai:
2, 3, 6, 7, 9, …



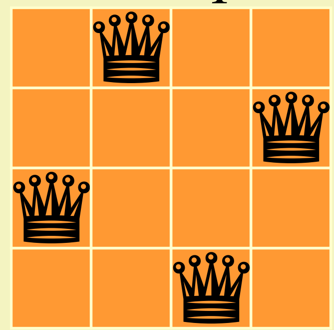
n-királynő probléma 1.

állapot ~ királynők egy elrendezése

általános állapot



Célállapot



Nem ismert, egy feltétel ellenőrzésével dönthető el

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

Állapottér: ***AT*** = {♛, _ }$^{n \times n}$

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

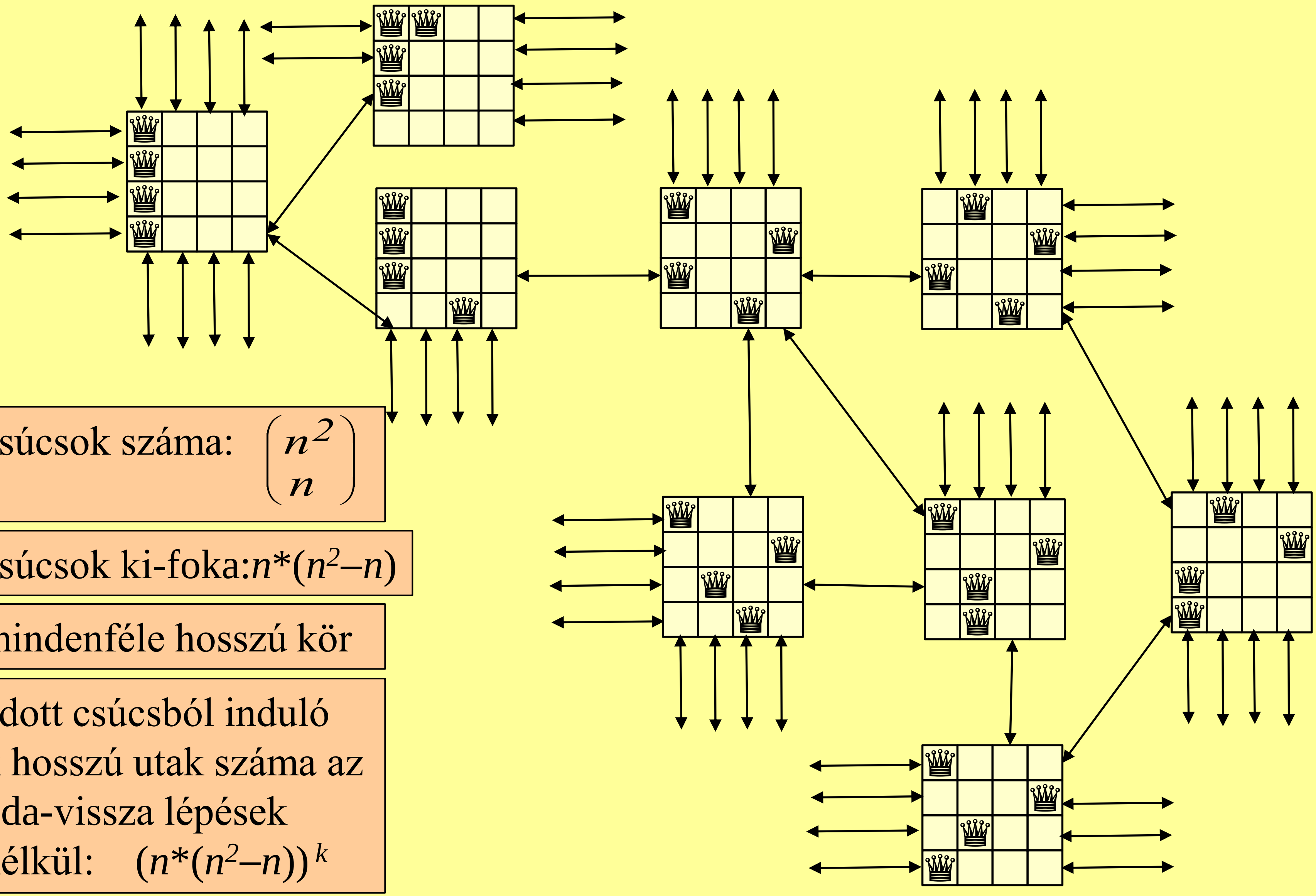
Művelet: ***Áthelyez***(*x,y,u,v*):***AT*** → ***AT*** (*this*:***AT***)

HA $1 \leqq x,y,u,v \leqq n$ és *this*[*x,y*]=♛ és *this*[*u,v*]=_

AKKOR *this*[*x,y*] ↔ *this*[*u,v*]

csere

Állapot-gráf részlet



csúcsok száma: $\binom{n^2}{n}$

csúcsok ki-foka: $n*(n^2-n)$

mindenféle hosszú kör

adott csúcsból induló k hosszú utak száma az oda-vissza lépések nélkül: $(n*(n^2-n))^k$

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 a problémára adott válasz egy célállapot (királynő-elrendezés), habár ezt egy alkalmas művelet-sorozattal érhetjük el, azaz végsősoron 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.

Reprezentációs gráf bonyolultsága → Problématér mérete → Keresés számításigénye

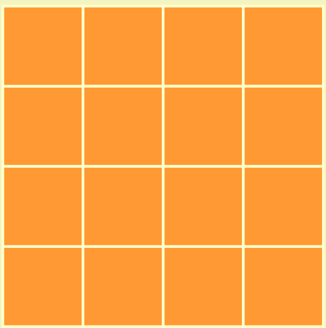
- A bonyolultság a start csúcsból kivezető utak számától függ, amely nyilván függvénye a
  - csúcsok és élek számának
  - csúcsok ki-fokának
  - körök gyakoriságának, és hosszuk sokféleségének

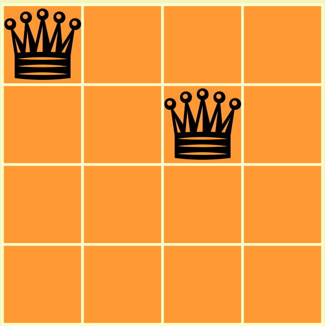
Csökkentsük a problématér méretét

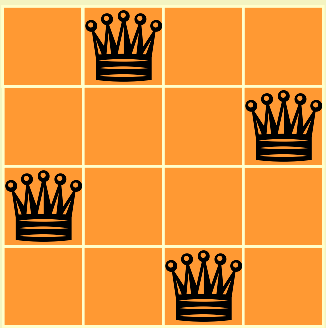
- Ugyanannak a feladatnak több modellje lehet : érdemes olyat keresni, amely kisebb problémateret jelöl ki.
  - Az *n*-királynő probléma modelljénél a problématér mérete (a lehetséges utak száma) óriási. Adjunk jobb modellt!
  - Bővítsük az állapotteret az *n*-nél kevesebb királynőt tartalmazó állásokkal,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

n-királynő probléma 2.

Kezdőállapot: *Közbülső állapot:* *Célállapot* :







nincs már üres sor és nincs ütés

Állapottér: ***AT*** = { ♛, _ }$^{n \times n}$

invariáns: az első néhány sor egy-egy királynőt tartalmaz

Művelet: **Helyez**(oszlop): ***AT*** → ***AT*** (*this*:***AT***)

HA *1*≦ *oszlop* ≦*n* és a *this*-beli soron következő üres sor ≦ *n*
és nincs ütés a *this*-ben

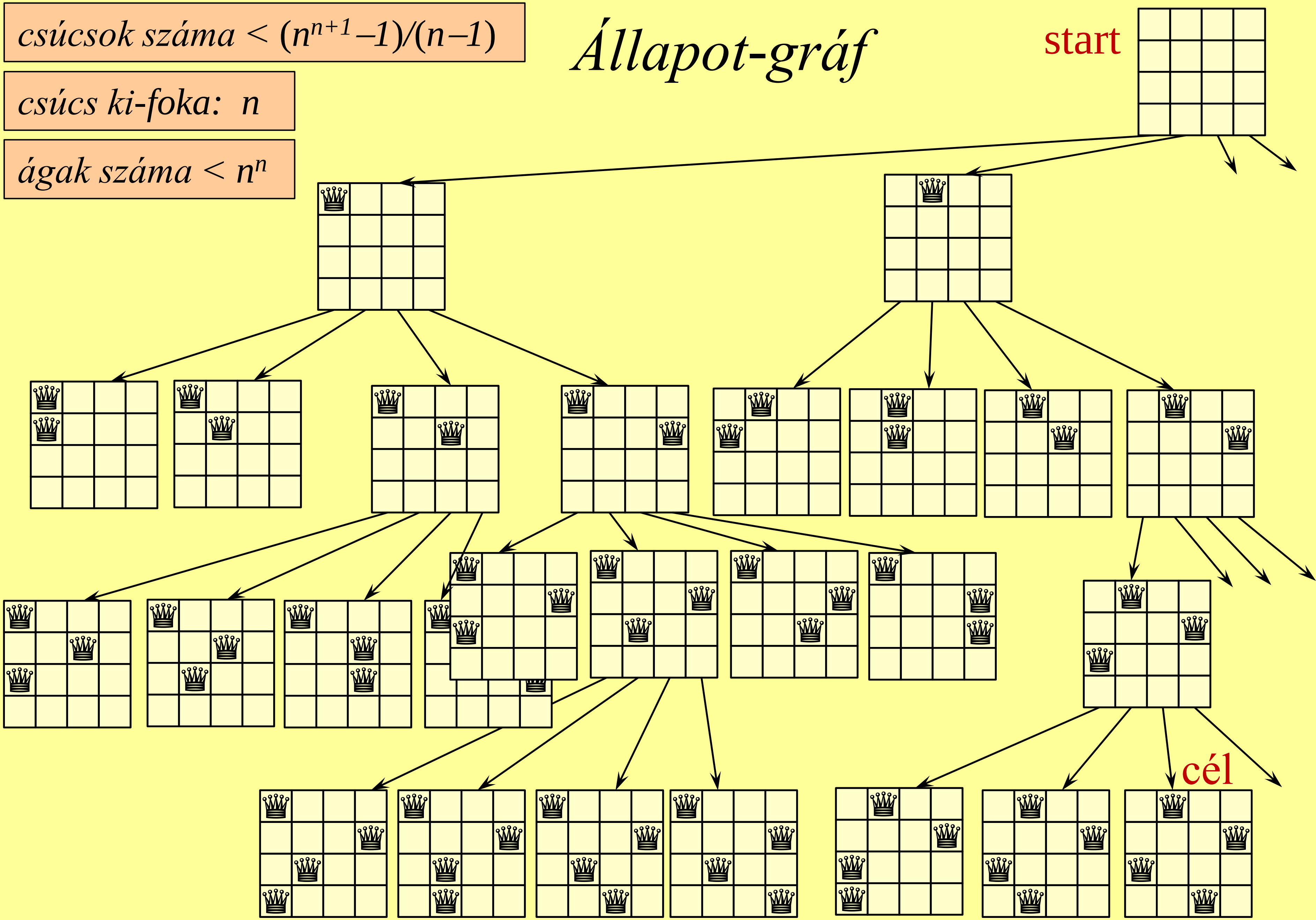
AKKOR *this*[a *this*-beli soron következő üres sor,*oszlop*] := ♛

Állapot-gráf

csúcsok száma < $(n^{n+1}-1)/(n-1)$

csúcs ki-foka: n

ágak száma < n^n



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

Közbülső állapot:

♛	×	×	×
×	×	♛	×
×	×	×	×
×		×	×

köv_sor = 3

Célállapot :

×	♛	×	×
×	×	×	♛
♛	×	×	×
×	×	♛	×

köv_sor = 5

Állapottér: **AT** = *rec*(*t* :{♛, ×, _ }$^{n\times n}$, *köv_sor* : ℕ)

invariáns: *köv_sor*≦*n*+1,

az első *köv_sor*–1 darab sor egy-egy királynőt tartalmaz,

királynők nem ütik egymást,

jelölés : × 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** → **AT**   (this:**AT**)

HA 1≦ oszlop ≦n és this.köv_sor ≦ n
és this.t[this.köv_sor,oszlop]=_

előfeltétel számítás-igénye: konstans

AKKOR

this.t[this.köv_sor,oszlop] := ♛

∀i∈[this.kövsor+1 .. n] :

this.t[i, oszlop]:=✖

hatás számítás-igénye: lineáris

ha (i ≤ n+this.köv_sor–oszlop) akkor this.t[i, i–this.köv_sor+oszlop]:=✖

ha (i ≤ this.köv_sor+oszlop–1) akkor this.t[i, this.köv_sor+oszlop–i]:=✖

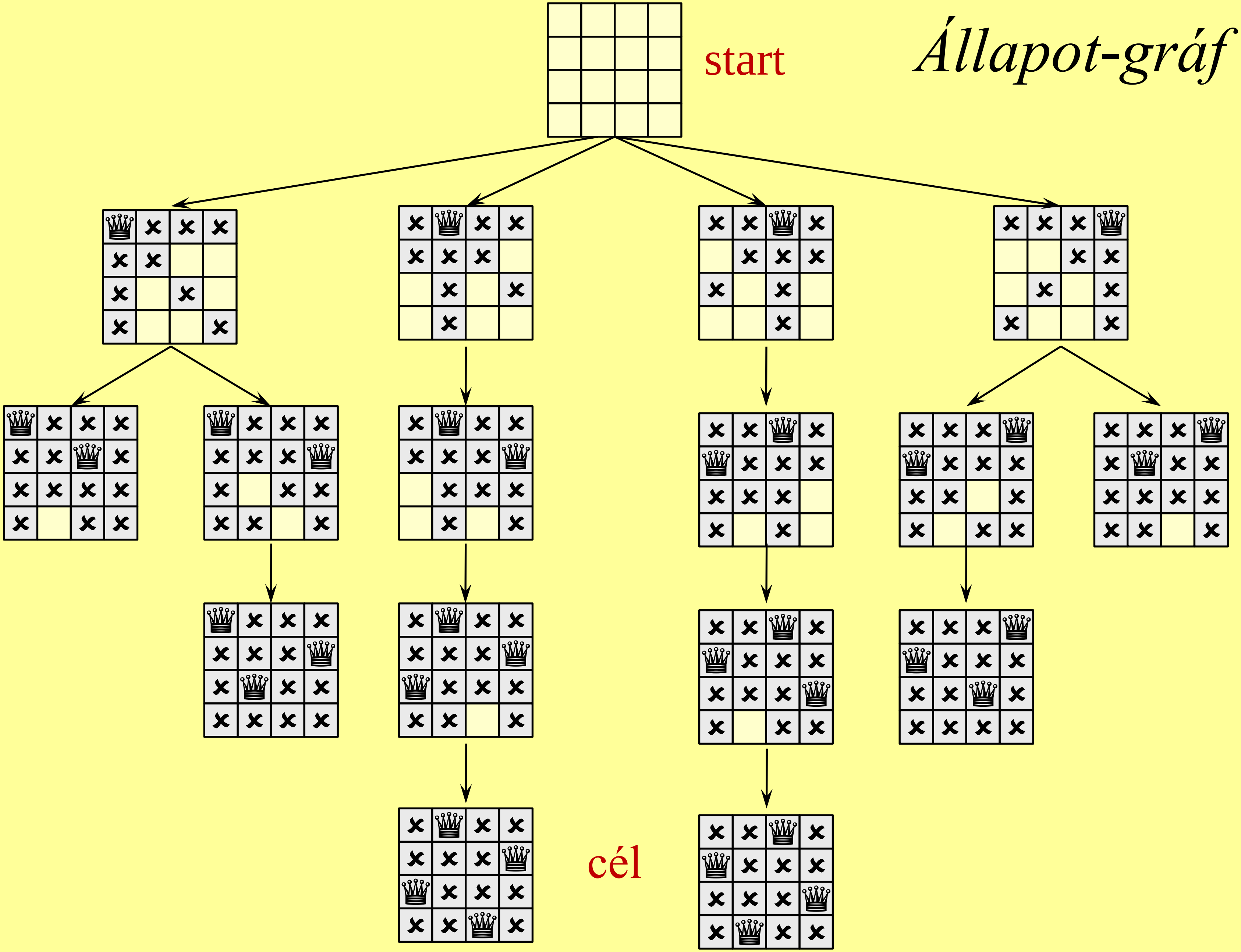
this.köv_sor := this.köv_sor+1

Kezdőállapot: this.t egy üres mátrix, this.köv_sor:=1

Célállapot: this.köv_sor>n

célfeltétel nagyon egyszerű lett

Állapot-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(mátrix* :$\{0..8\}^{3\times3}$, *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*** → ***AT***

koordinátánkénti összeadás

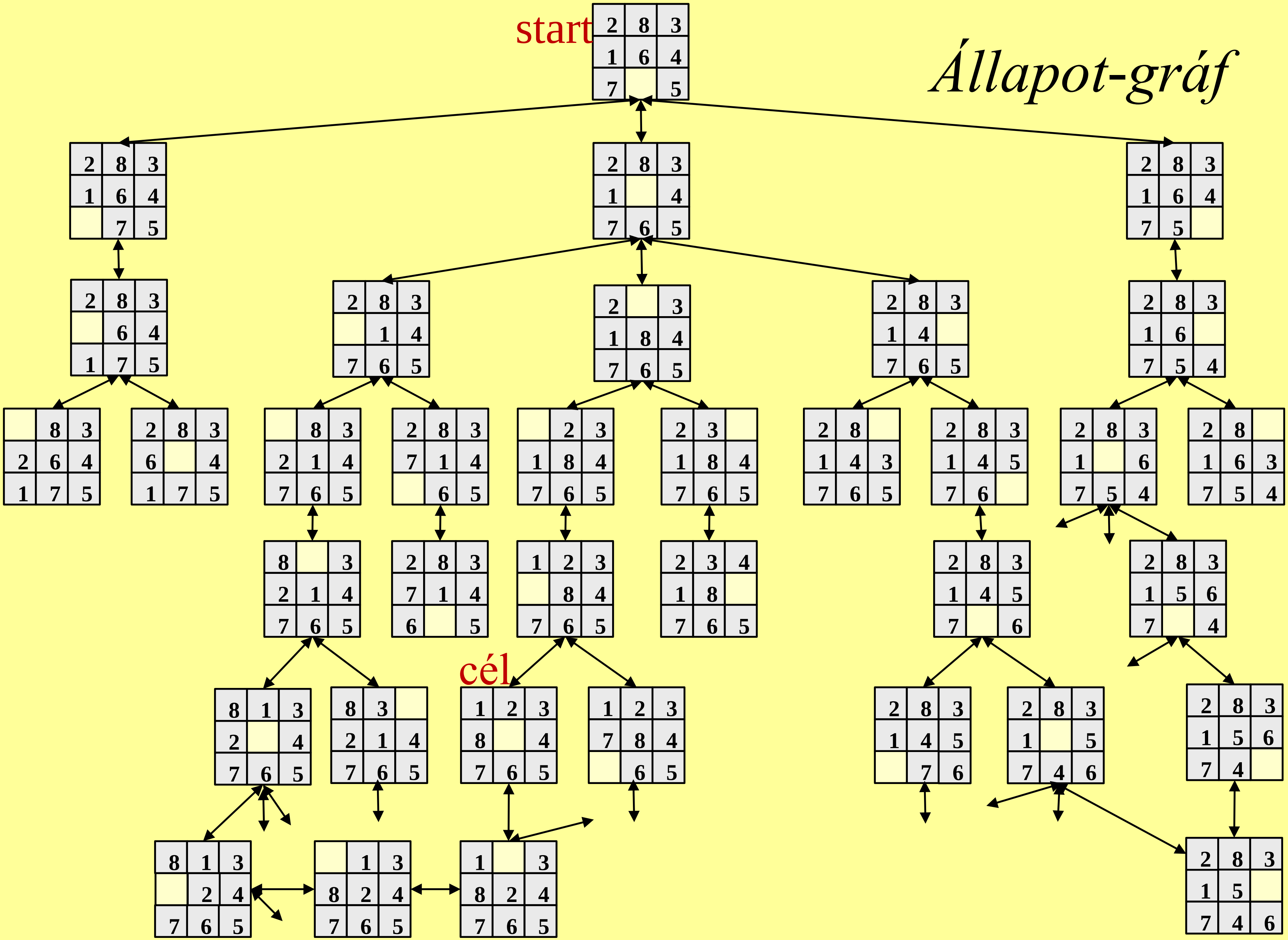
HA *irány* ∈ {(*0*,-*1*),(-*1*,*0*),(*0*,*1*),(*1*,*0*)} és

(*1*,*1*) ≦ *this.üres*+*irány* ≦ (*3*,*3*) (*this*: ***AT***)

AKKOR *this.mátrix*[*this.üres*] ↔ *this.mátrix*[*this.üres* +*irány*]

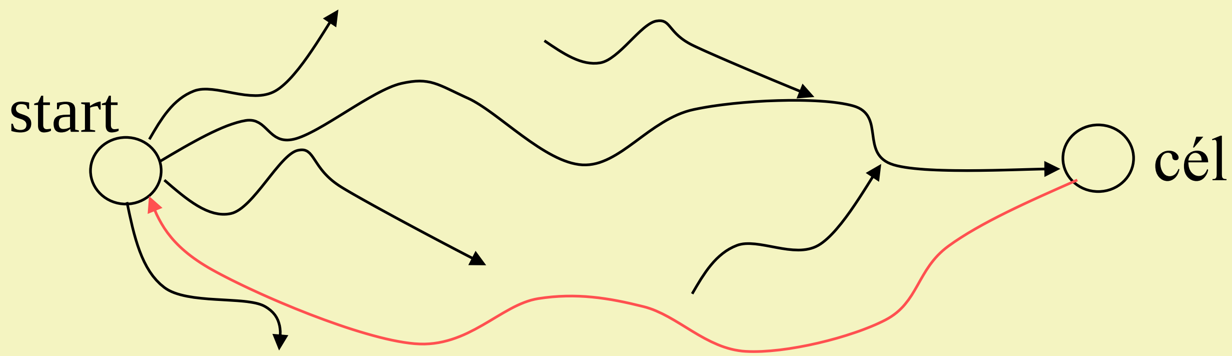
this.üres := *this.üres* +*irány*

Állapot-gráf



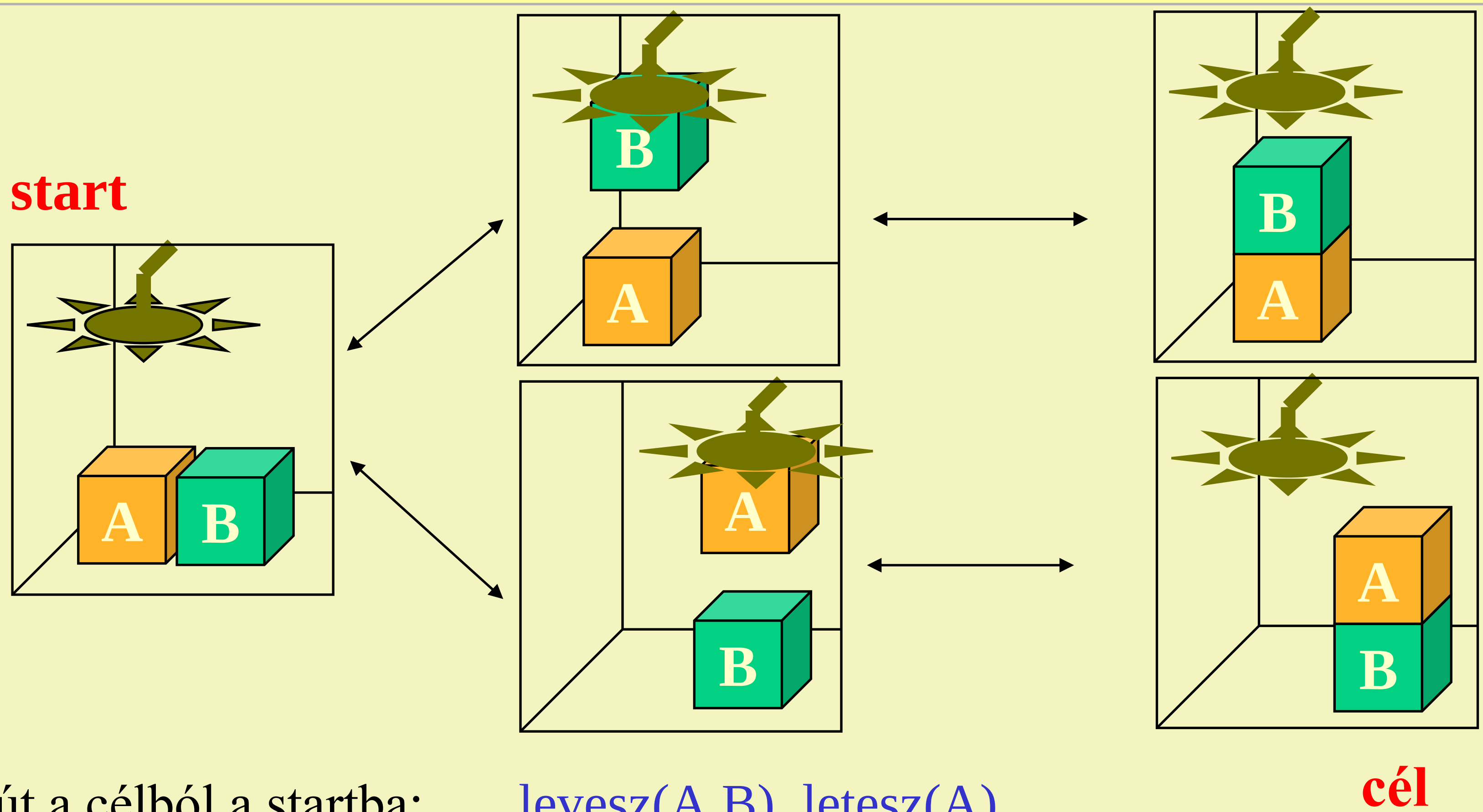
2. Visszafelé haladó keresés

- Ha a megoldás megtalálása a cél felől nézve egyszerűbb, mint a start felől nézve (mert a problématér 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



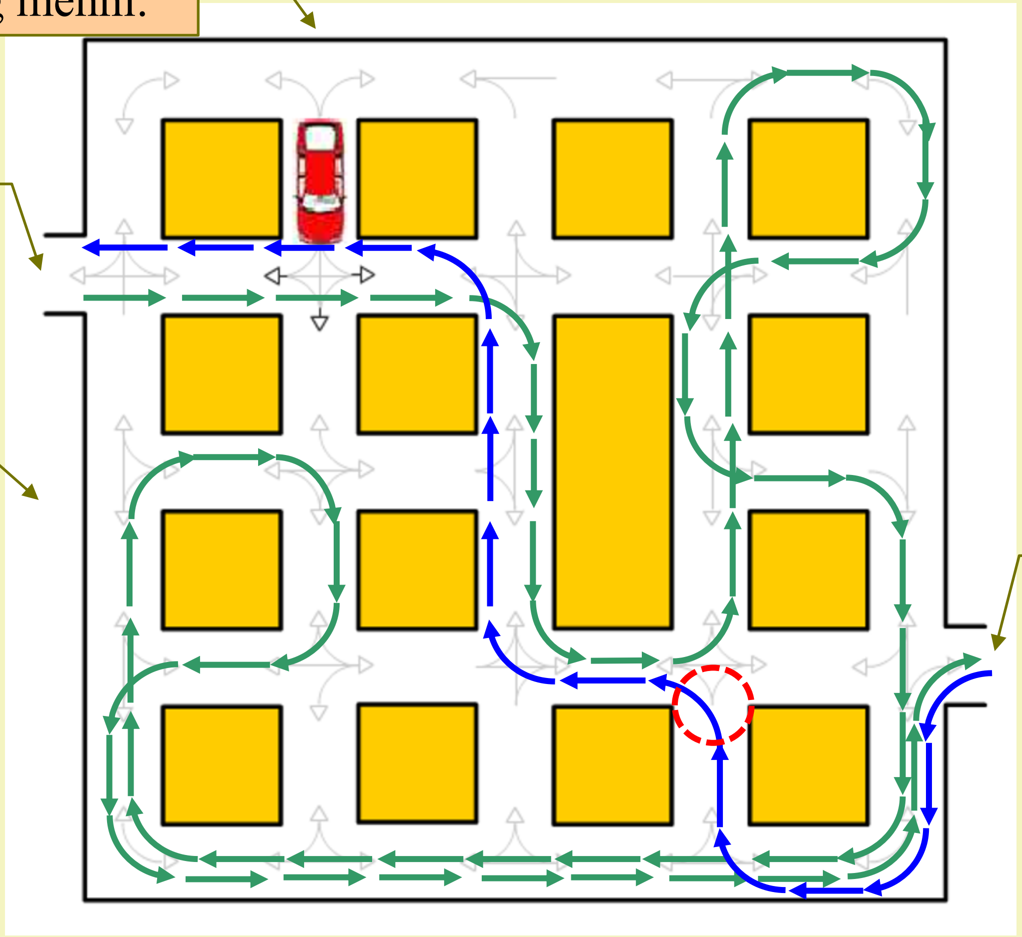
út a célból a startba: levesz(A,B), letesz(A)
megoldás: felvesz(A), rátesz(A,B)

Útvonal keresés városban

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

start

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



cél

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!

<u>*Állapottér*</u>: ***AT***= *map*(*key*:Keys, *value*:ℕ) ahol Keys={5,3,2 }

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

this[5]=5, this[3]=0, this[2]=0

<u>*Kezdőállapot*</u>: [5, 0, 0]

this[2]=1

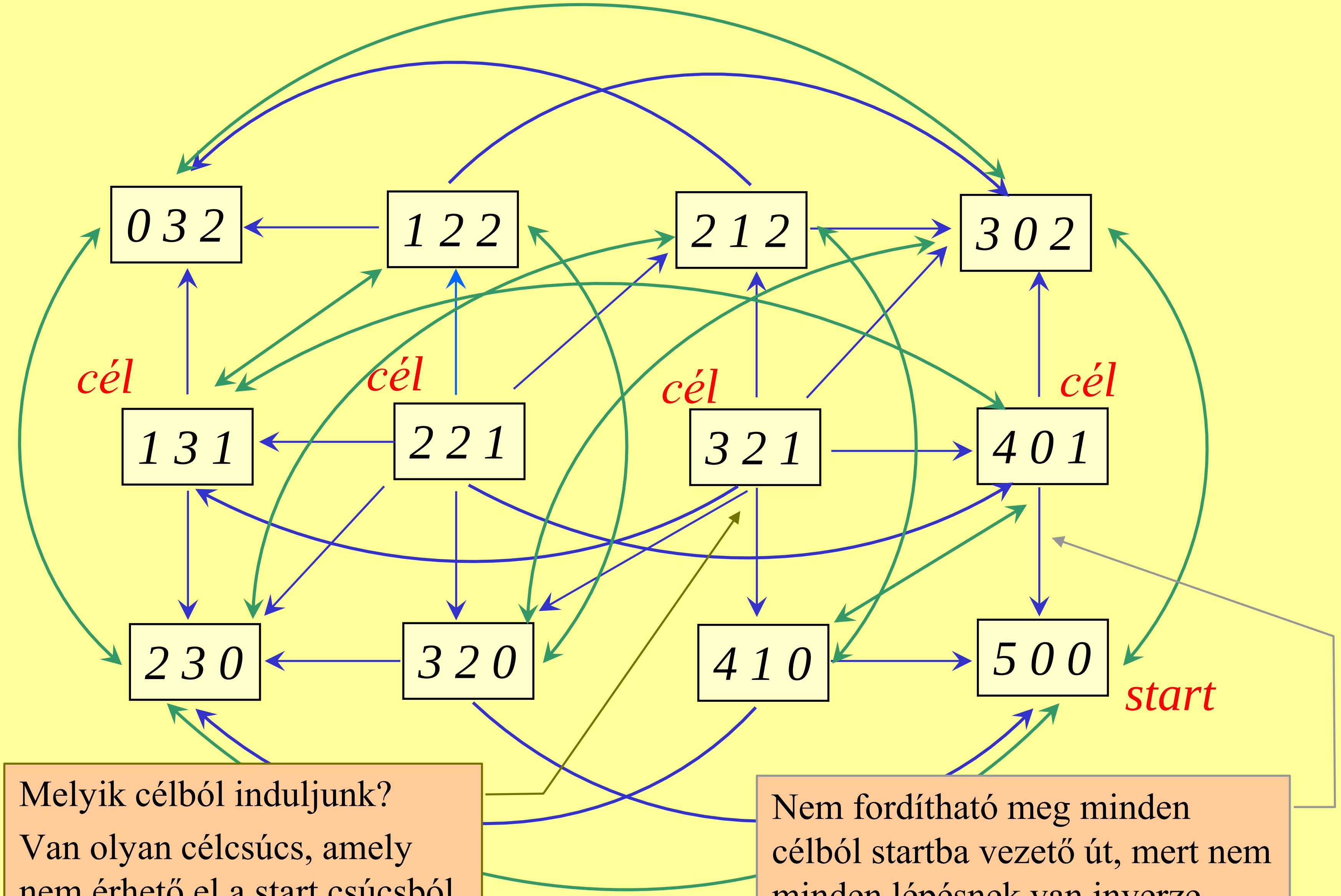
<u>*Célállapot*</u>: [*x*, *y*, 1]

<u>*Művelet*</u>: ***Tölt***(i,j): ***AT*** → ***AT*** (*this* : ***AT***)

HA $i,j \in \text{Keys}$ 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



Melyik célból induljunk?
Van olyan célcsúcs, amely nem érhető el a start csúcsból.

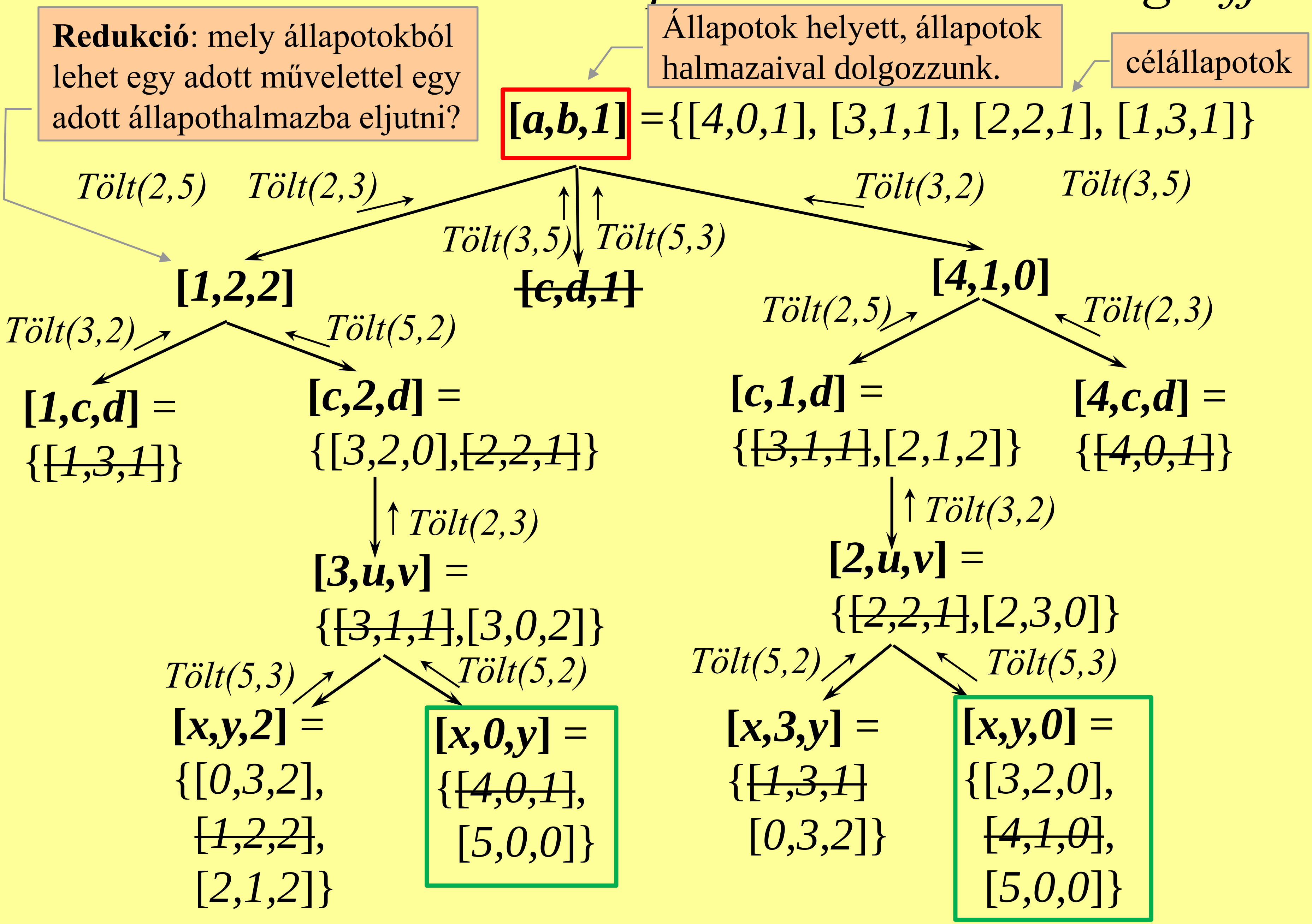
Nem fordítható meg minden célból startba vezető út, mert nem minden lépésnek van inverze.

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



invariáns: $\Sigma_{i\,\in \mathrm{Keys}}\, this[i] = 5$ és $\forall i \in \mathrm{Keys}: this[i] \le i$

Tölt(i,j): ***AT*** $\to$ ***AT***

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

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

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

Kancsó probléma redukciós operátora

$R_{\boldsymbol{Tölt}(i,j)} : 2^{AT} \longrightarrow 2^{AT}$ ahol $i,j \in \mathrm{Keys}$ és $i \neq j$

állapothalmazból állapothalmazba

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

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

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

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

$\Sigma_{i\in \mathrm{Keys}}\, a[i] = 5$ és $\forall i \in \mathrm{Keys} : a[i] \leqq i$ és

az állapottér modell invariánsa

$min(a[i],\, j - a[j]) > 0$ és

$\mathcal{D}_{\boldsymbol{Tölt}(i,j)}$-nek a *this*-re (itt *a*-ra) vonatkozó része

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

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

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

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

Probléma-redukciós modell

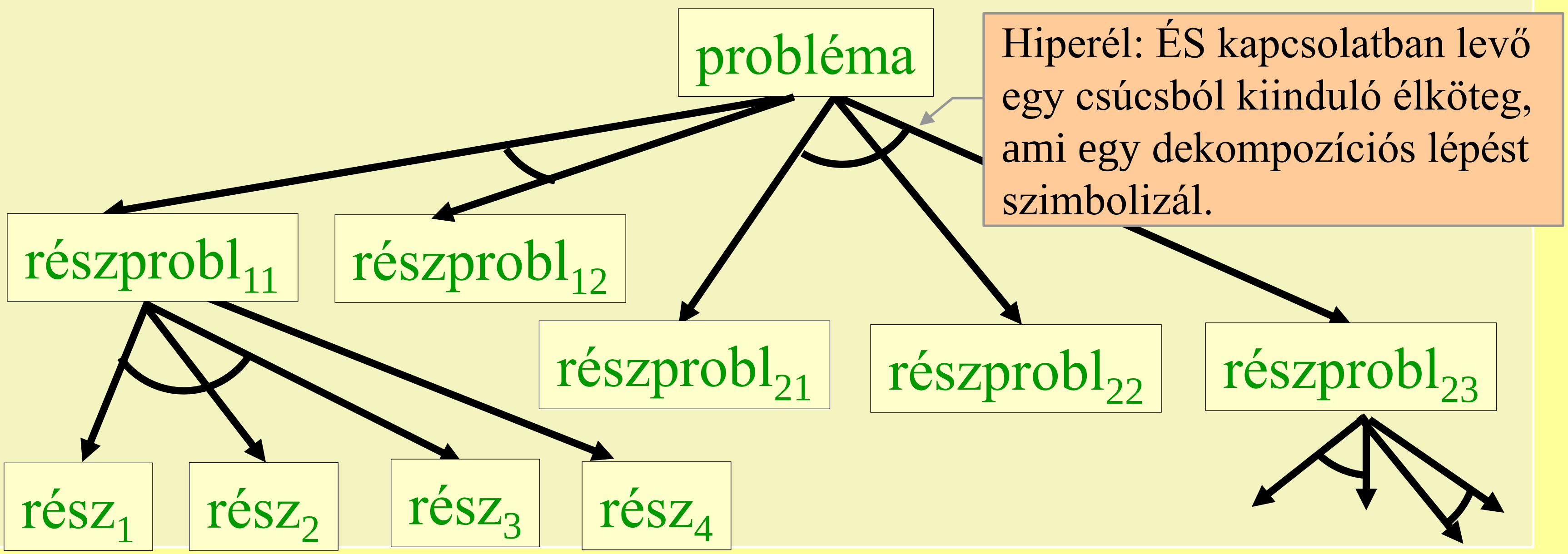
- Adott a probléma valamelyik állapottér modellje.
  (állapottér: AT, invariáns: $Inv{:}AT \longrightarrow \mathbb{L}$)
- Az állapottér modell minden $M{:}AT \longrightarrow AT$ műveletéhez generálunk egy olyan $R_M : 2^{AT} \longrightarrow 2^{AT}$ leképezést (redukciós operátor), hogy az egy B állapot-halmazhoz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

Megjegyzés

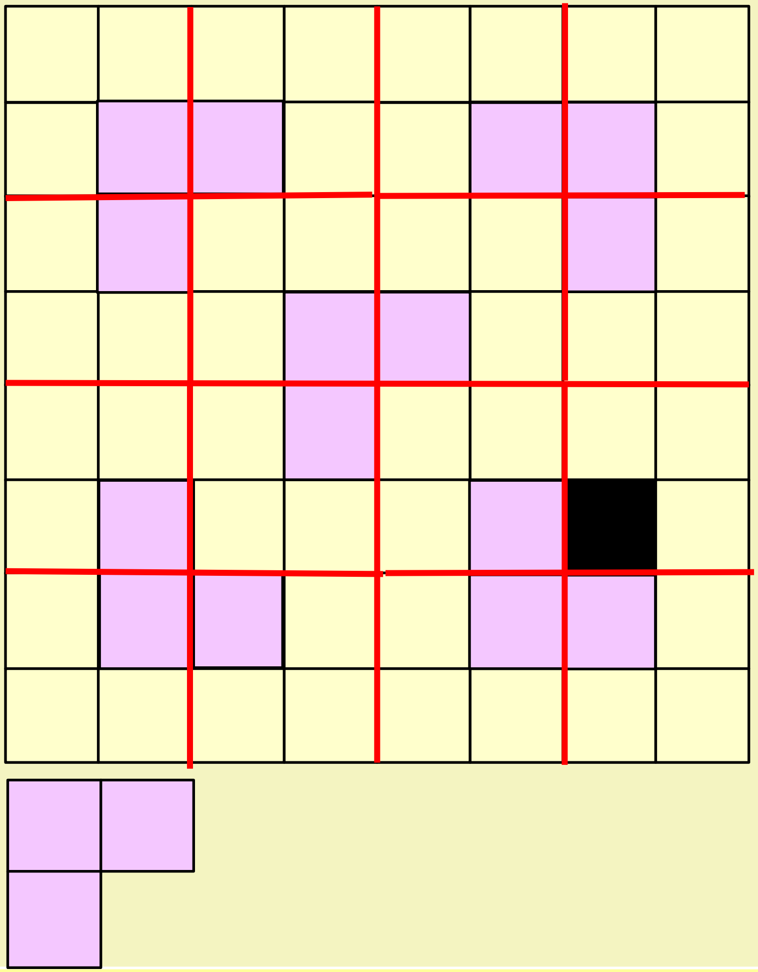
- A feladat olyan redukciós operátor-sorozat megtalálása, amely célhalmazból kezdőhalmazba vezet.
- A megoldás ezen sorozat redukciós operátorait generáló műveletek fordított sorrendben kiolvasott sorozata lesz.
- A probléma-redukció is modellezhető egy irányított gráffal, ahol a csúcsok állapot-halmazokat, 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.

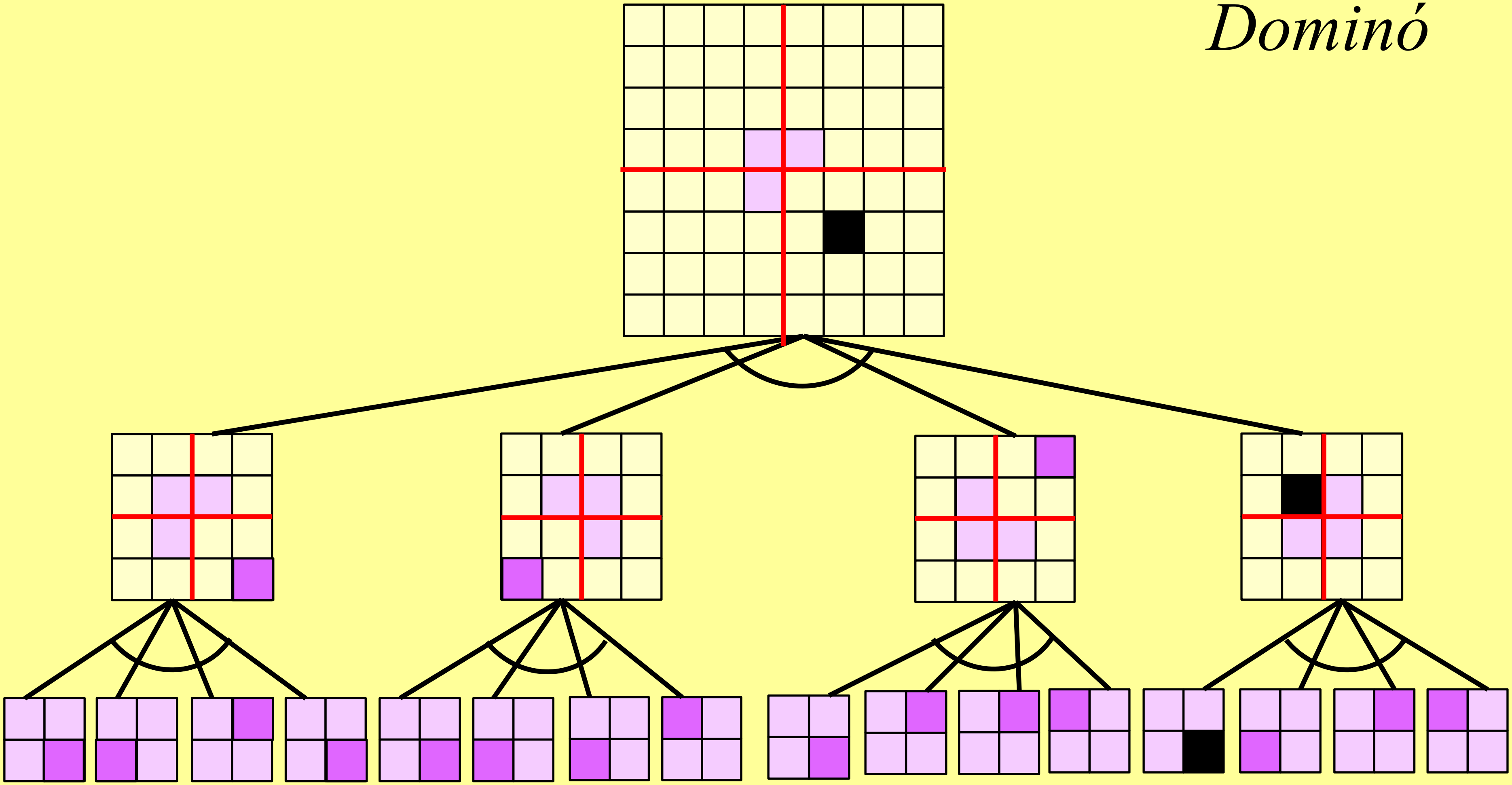


Dominó



- **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

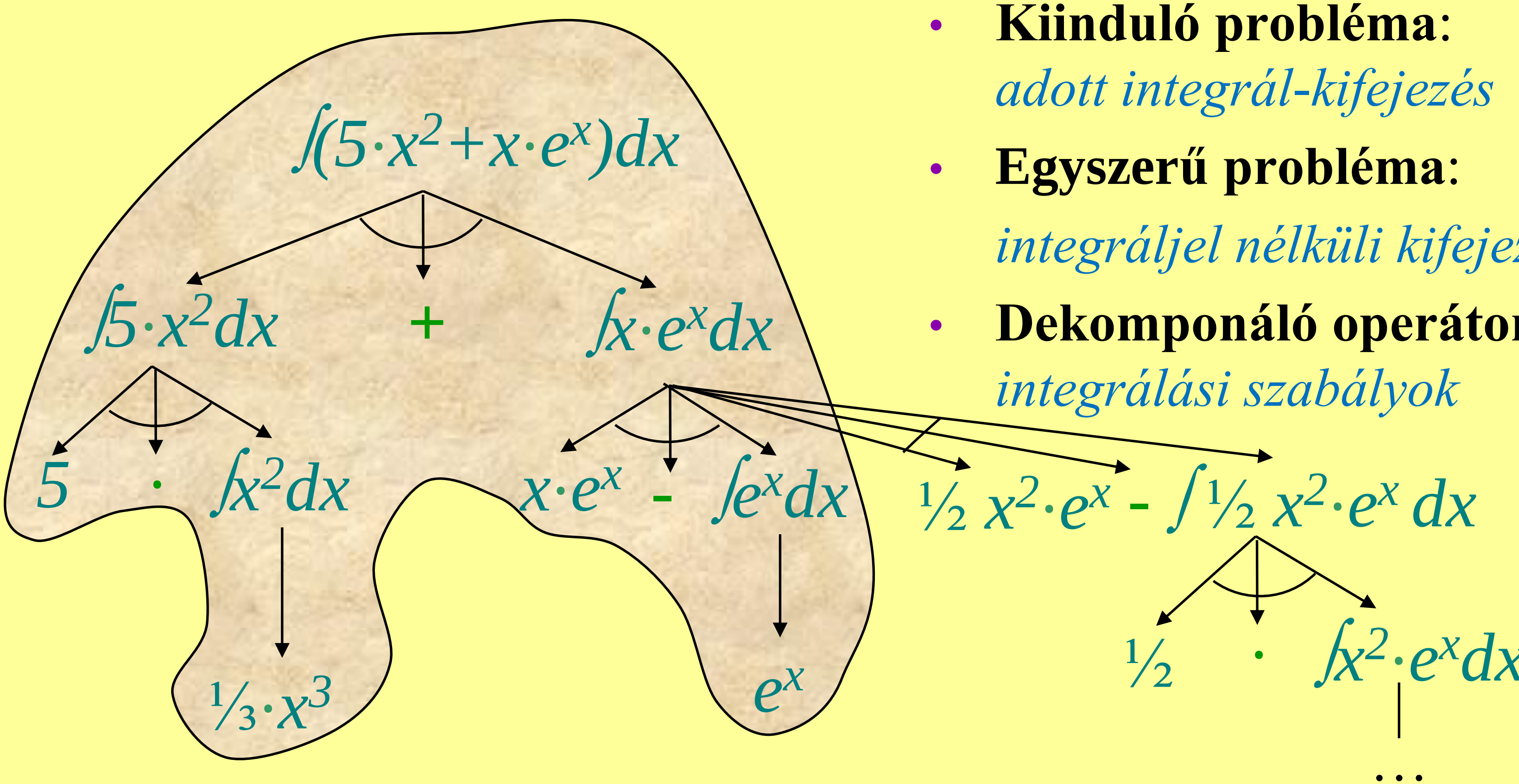
Dominó



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.

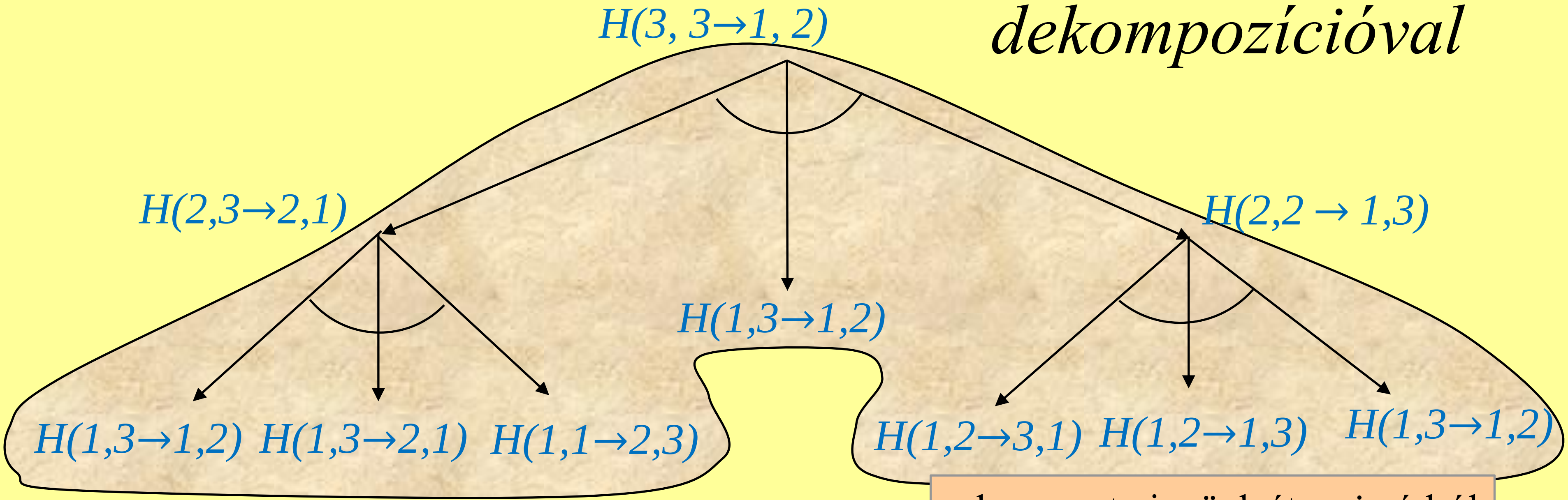
Integrálszámítás

- **Probléma általános leírása**: *szintaktikusan helyes kifejezés, vagy egy műveleti jel*
- **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 összeolvasni ahhoz, hogy az eredeti integrálkifejezés primitív függvényét megkapjuk.

Hanoi tornyai probléma megoldása dekompozícióval



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

n korongot vigyünk át az *i*. rúdról a *j*. rúdra a *k*. rúd segítségével

Kiinduló probléma: $H(3, 3 \to 1, 2)$

Egyszerű probléma: $H(1, i \to j, k)$

eldönthető, hogy megoldható-e

Dekomponálás: $H(n, i \to j, k) \leadsto \langle H(n-1, i \to k, j), H(1, i \to j, k), H(n-1, k \to j, i) \rangle$

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: *probléma* $\rightarrow$ *probléma*$^+$ és
      $D(p)=\langle p_1, \ldots , p_n\rangle$

Probléma dekompozíció ÉS/VAGY gráffal

- Dekompozíciós modell ÉS/VAGY gráf
  - részprobléma ~ csúcs
  - dekomponáló művelet hatása egy problémára ~ irányított hiperél
    - egy problémára véges sok művelet alkalmazható (véges kifokú gráf)
  - művelet költsége ~ hiperél költsége
    - van a műveltek költségének alsó pozitív korlátja (δ)
  - kiinduló probléma ~ startcsúcs
  - megoldható probléma ~ célcsúcs
- Gráf-reprezentáció: ÉS/VAGY gráf, startcsúcs, célcsúcsok
  - dekompozíciós folyam ~ hiperút
  - megoldás ~ megoldás-gráfból olvasható ki