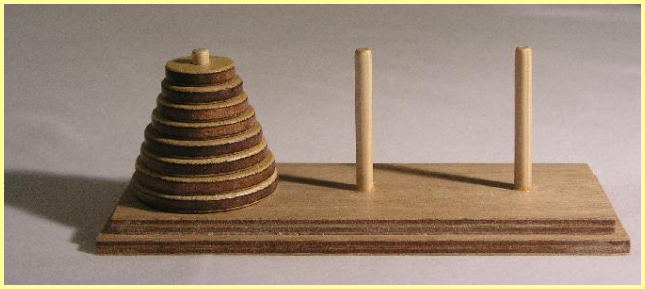
Modellezés

1. Állapottér modell

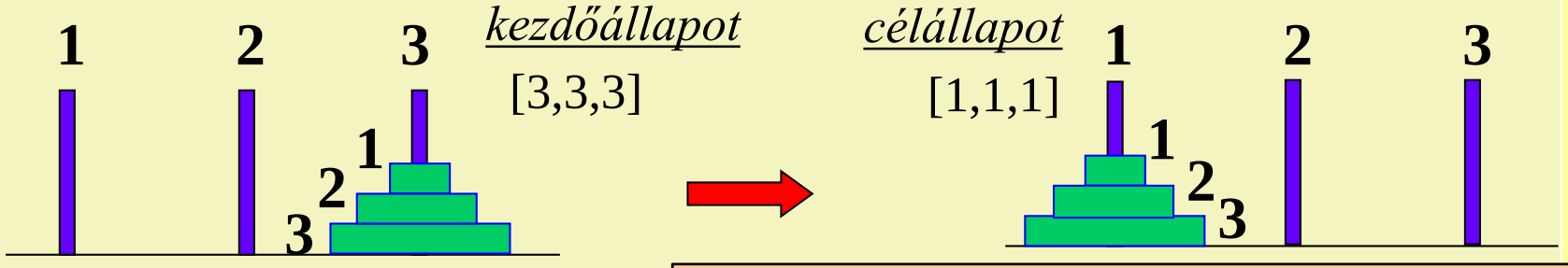
- ***Állapottér***: a probléma leírásához szükséges adatok által felvett érték-együttesek (azaz állapotok) halmaza
  - az állapot többnyire egy összetett szerkezetű érték
  - gyakran egy bővebb alaphalmazzal és egy azon értelmezett invariáns állítással definiáljuk
- ***Műveletek***: állapotból állapotba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

Állapottér modell állapot-gráfja

❑ Állapottér modell		Állapot-gráf
○ állapot	~	csúcs
○ művelet hatása egy állapotra	~	irányított él
○ művelet költsége	~	él költsége
○ 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*→*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)

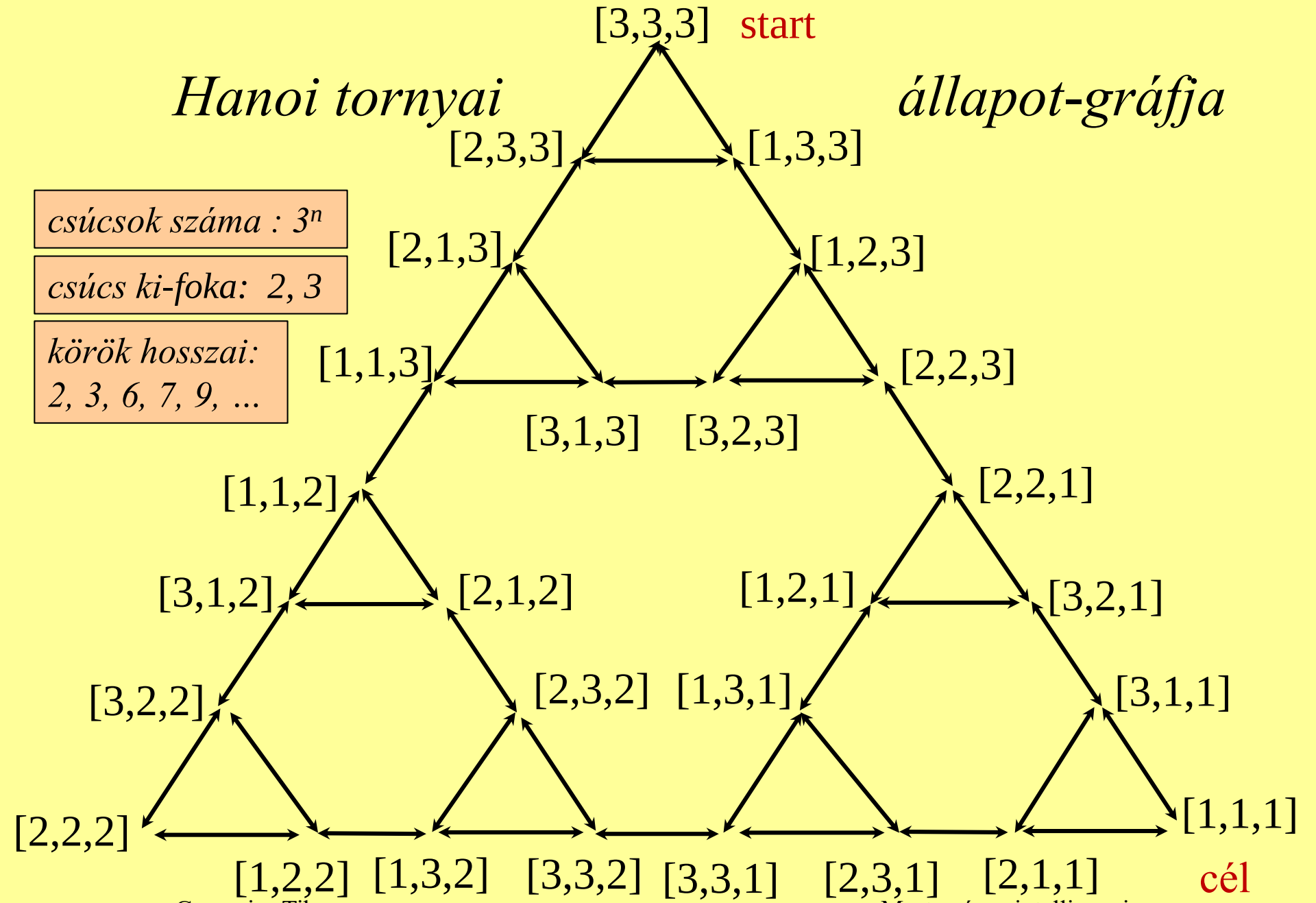
AKKOR *this*[*honnan legfelső korongja*] := *hova*

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 *állapot-gráfja*



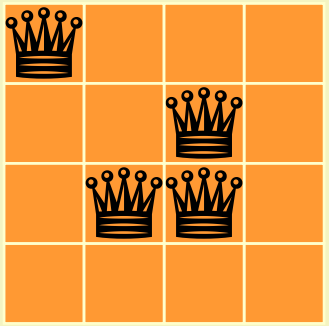
csúcsok száma : 3^n

csúcs ki-foka: 2, 3

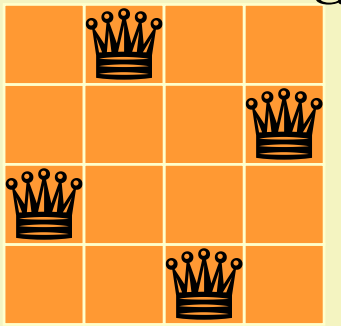
körök hosszai: 2, 3, 6, 7, 9, ...

n-királynő probléma 1.

általános állapot



utófeltételnek megfelelő állapot



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

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

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

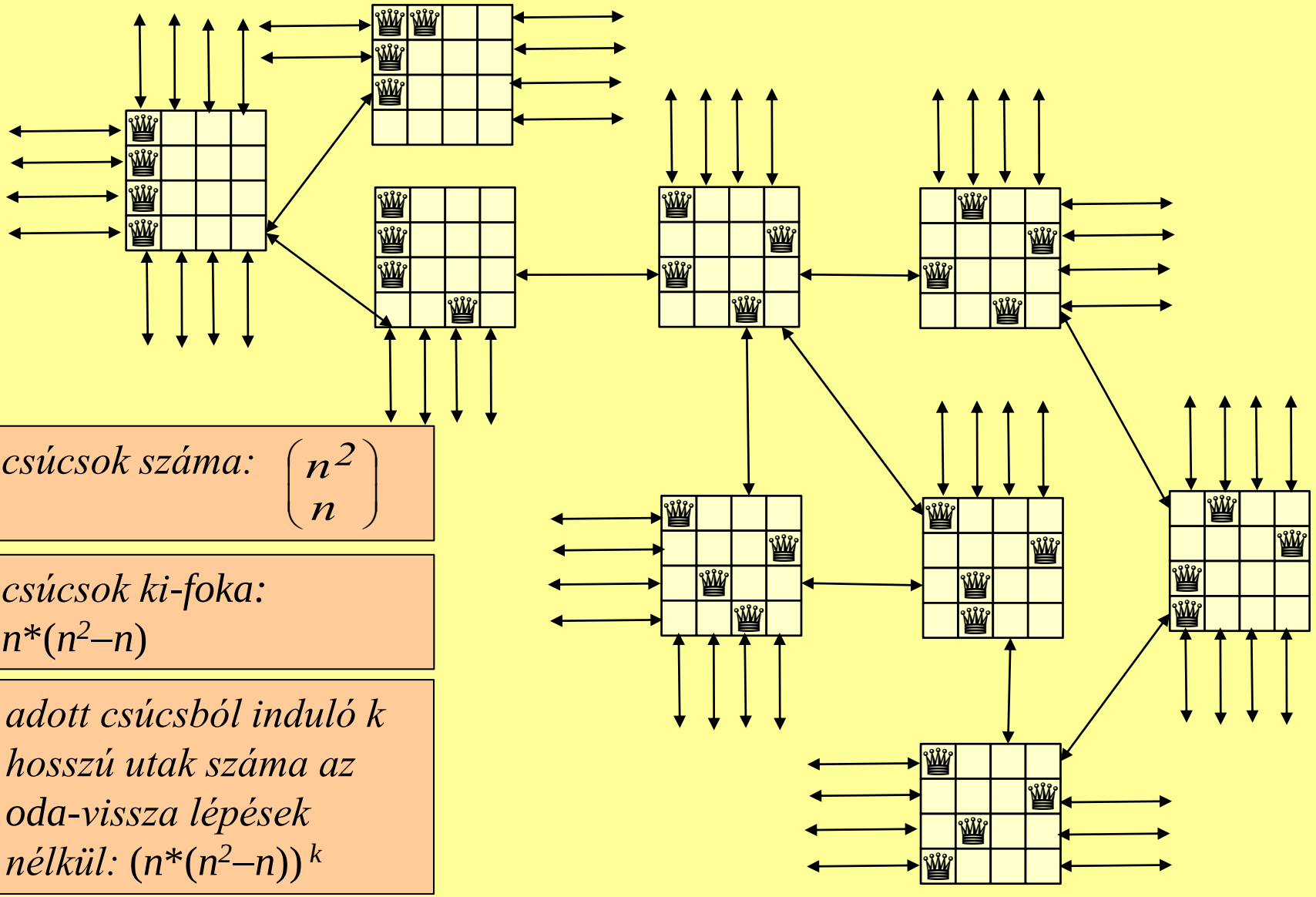
Művelet: ***Áthelyez***(x,y,u,v):**AT** → **AT** (*this*:**AT**)

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

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

csere

Állapot-gráf részlet



csúcsok száma: $\binom{n^2}{n}$

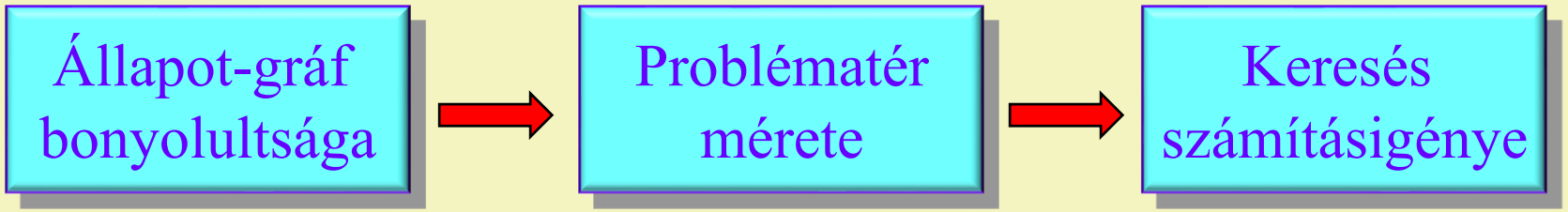
csúcsok ki-foka:
$n*(n^2-n)$

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

Állapottér vs. problématér

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

Állapot-gráf bonyolultsága



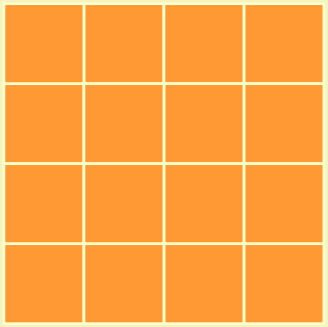
- A bonyolultság a start csúcsból kivezető utak számától függ, amely nyilván függvénye a
  - csúcsok és élek számának
  - csúcsok ki-fokának
  - körök gyakoriságának, és hosszuk sokféleségének

Csökkentsük a problématér méretét

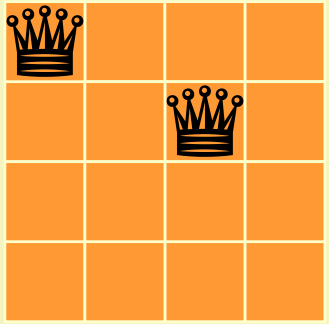
- Ugyanannak a feladatnak több modellje lehet : érdemes olyat keresni, amely kisebb problémateret jelöl ki.
  - Az *n*-királynő problématerének mérete, az előző modell szerinti a lehetséges utak száma, óriási. Adjunk jobb modellt!
  - Bővítsük az állapotteret az *n*-nél kevesebb királynőt tartalmazó állásokkal, és használjunk új műveletet : királynő-felhelyezést (kezdő állás az üres tábla).
  - Műveletek előfeltételének szigorításával csökken az 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.

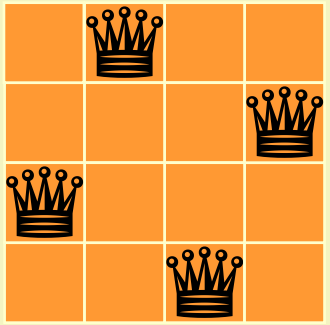
kezdőállapot



közbülső állapot



célállapot



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

nincs már üres sor és nincs ütés

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

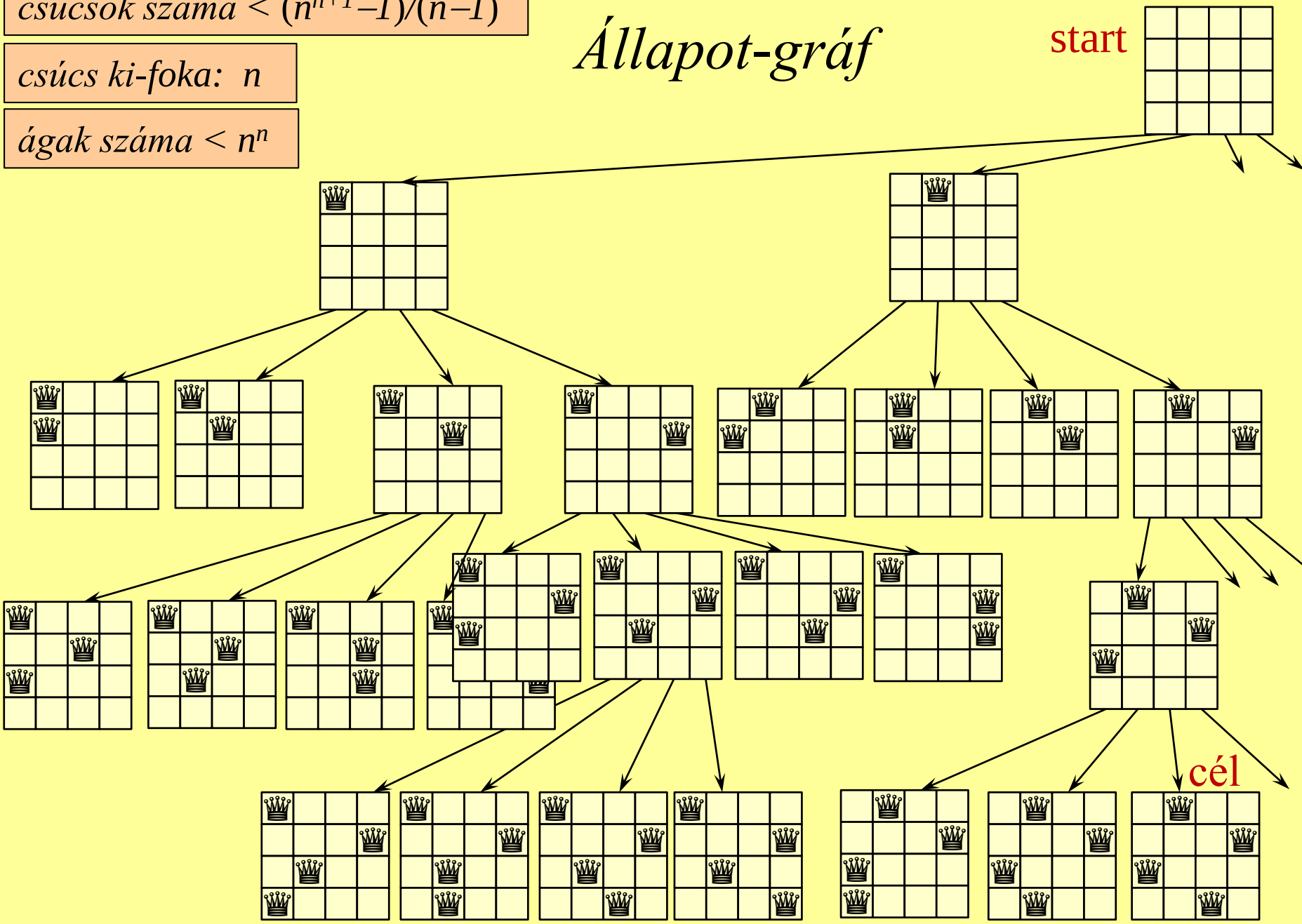
AKKOR *this*[a *this*-beli soron következő üres sor,*oszlop*] := ♕

csúcsok száma $< (n^{n+1}-1)/(n-1)$

csúcs ki-foka: n

ágak száma $< n^n$

Állapot-gráf

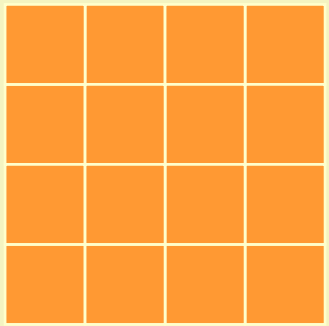


Művelet végrehajtásának hatékonysága

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

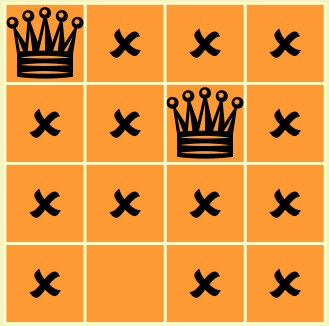
n-királynő probléma 3.

kezdőállapot:



köv_sor = 1

közbülső állapot:



köv_sor = 3

célállapot :

köv_sor = 5

Állapottér: **AT** = $rec(t : \{♕, ×, \_\}^{n \times n}, köv\_sor : \mathbb{N})$

invariáns: $köv\_sor \leqq n+1$,
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 $1 \leqq oszlop \leqq n$ és $this.köv\_sor \leqq n$

és $this.t[this.köv\_sor, oszlop] = \_$

AKKOR

$this.t[this.köv\_sor, oszlop] := ♕$

$\forall i \in [this.kövsor+1 \,..\, n]$:

$this.t[i, oszlop] := ✖$

ha $(i \leq n+this.köv\_sor-oszlop)$ akkor $this.t[i, i-this.köv\_sor+oszlop] := ✖$

ha $(i \leq this.köv\_sor+oszlop-1)$ akkor $this.t[i, this.köv\_sor+oszlop-i] := ✖$

$this.köv\_sor := this.köv\_sor+1$

előfeltétel számítás-igénye: konstans

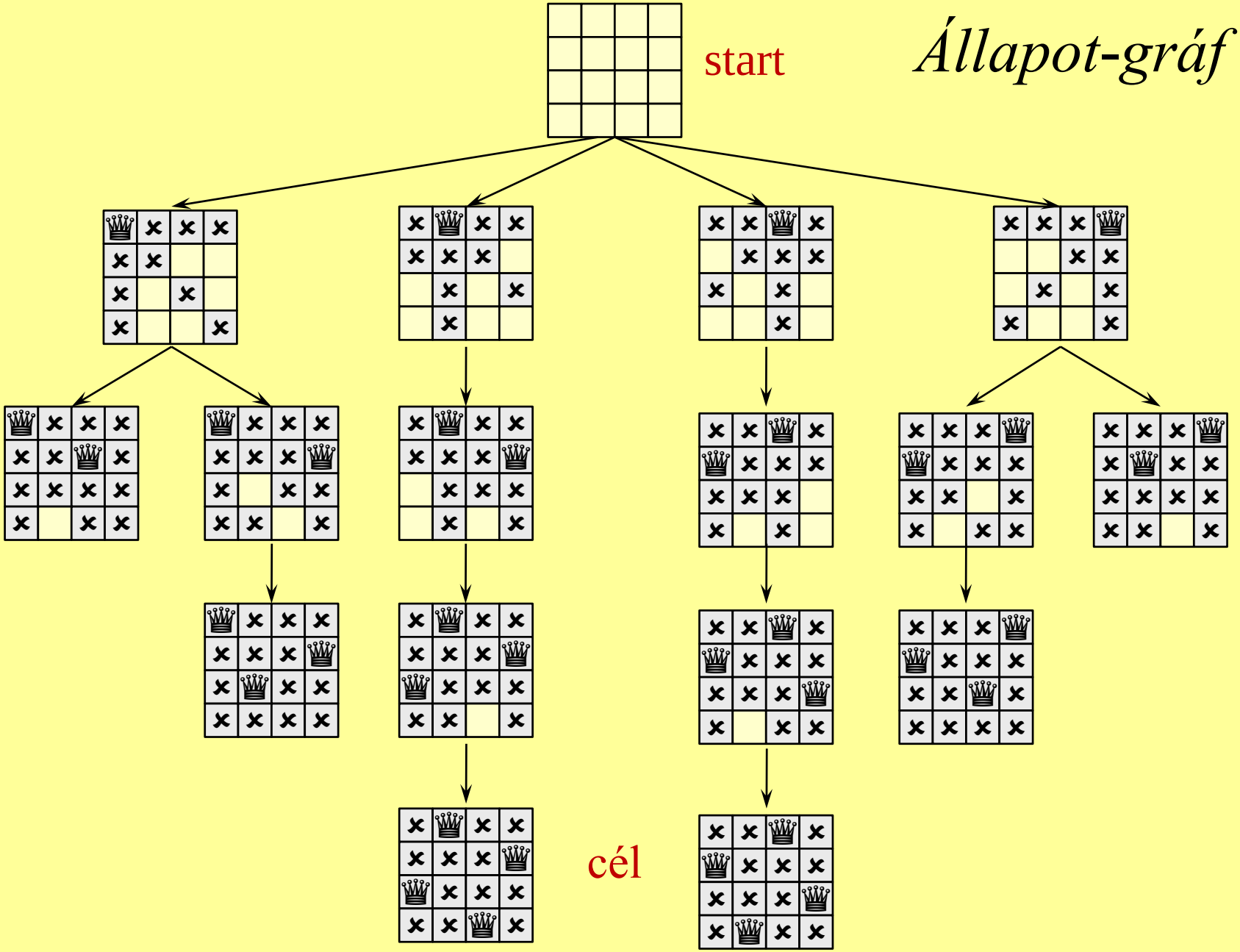
hatás számítás-igénye: lineáris

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

célállapot: $this.köv\_sor > n$

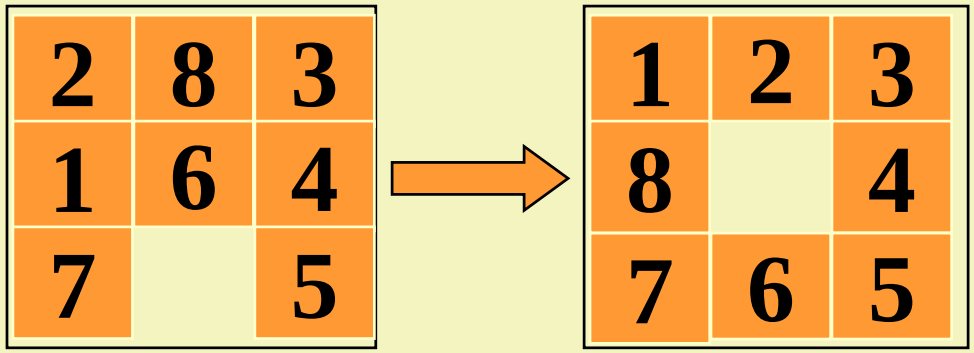
célfeltétel nagyon egyszerű lett

Állapot-gráf



Tologató játék (8-as, 15-ös)

kezdőállapot: tetszőleges



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**

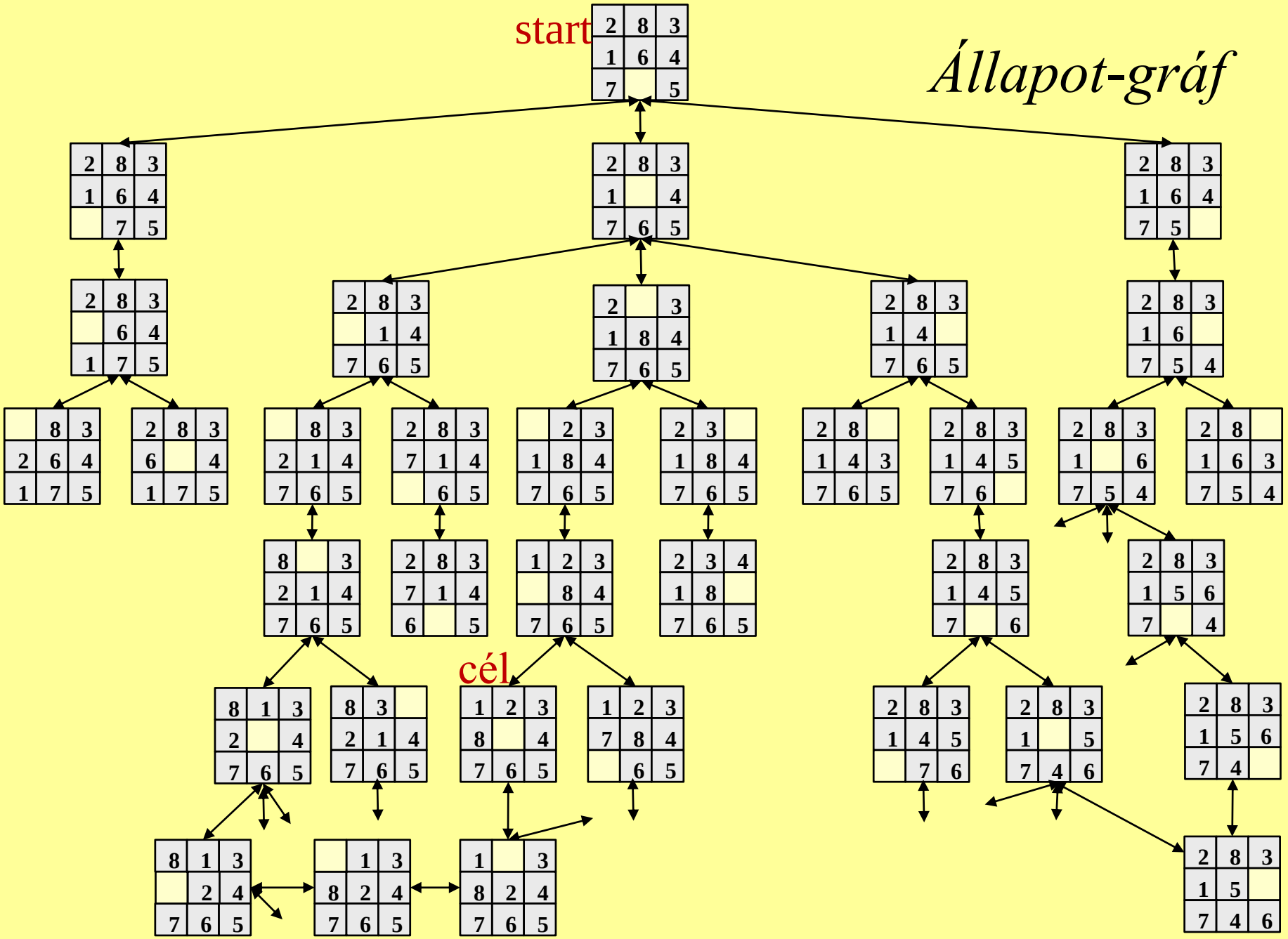
koordinátánkénti összeadás

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*]

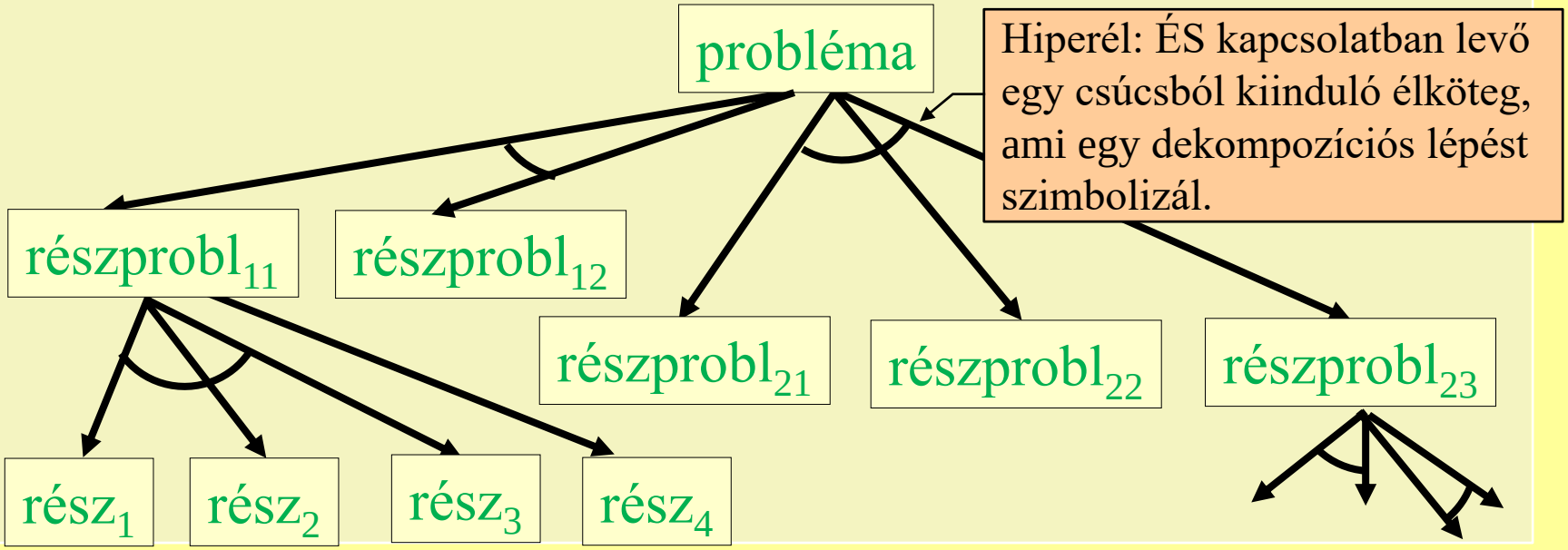
this.üres := *this.üres* +*irány*

Állapot-gráf

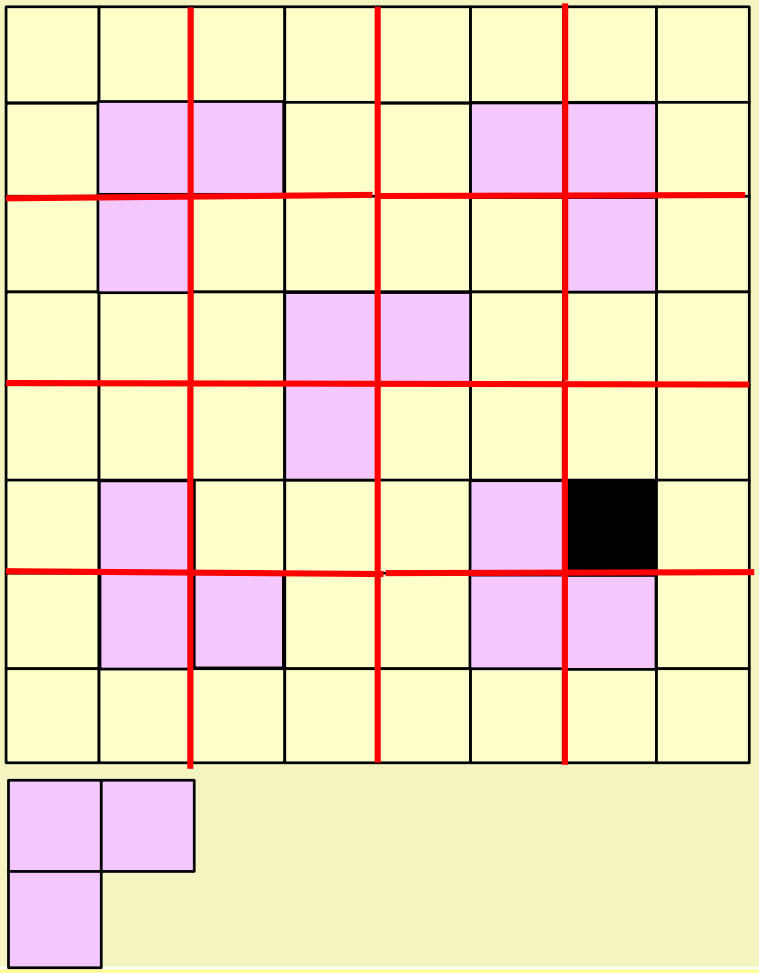


2. Probléma dekompozíció

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

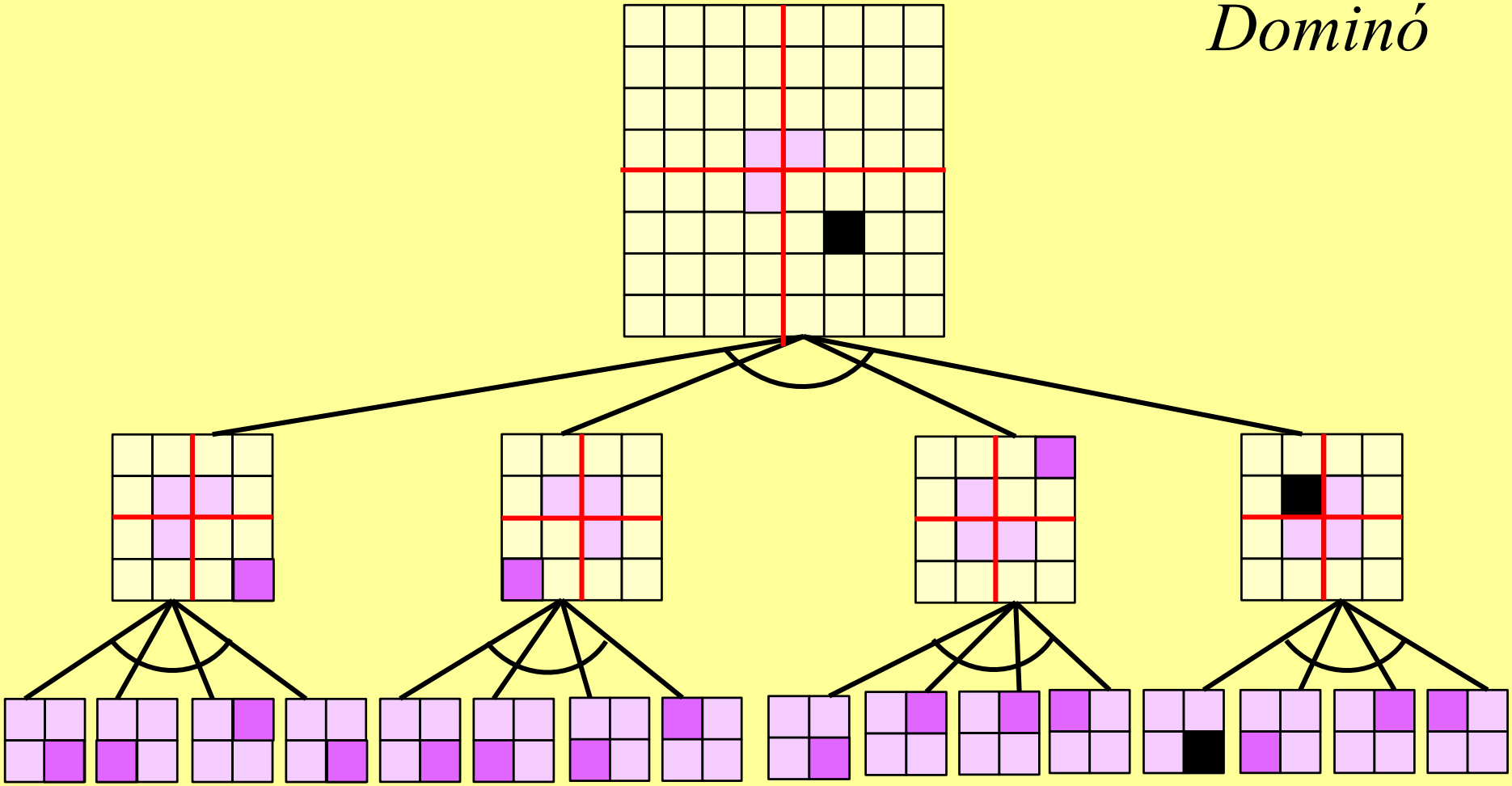


Dominó



- **Probléma általános leírása**:
  $2^n \times 2^n$-es tábla egy foglalt mezővel
- **Kiinduló probléma**:
  8×8-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

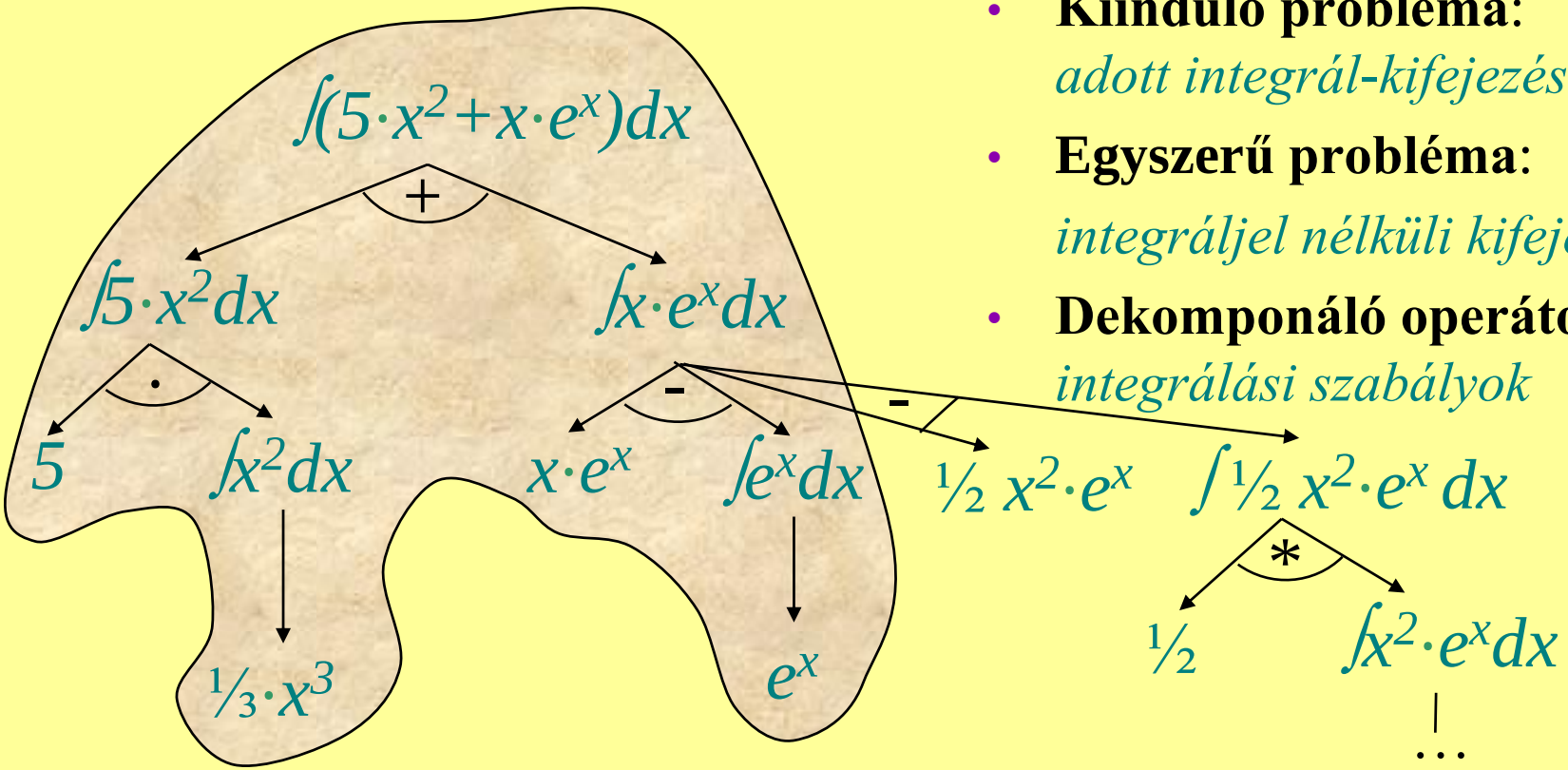
Dominó



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

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

Integrálszámítás

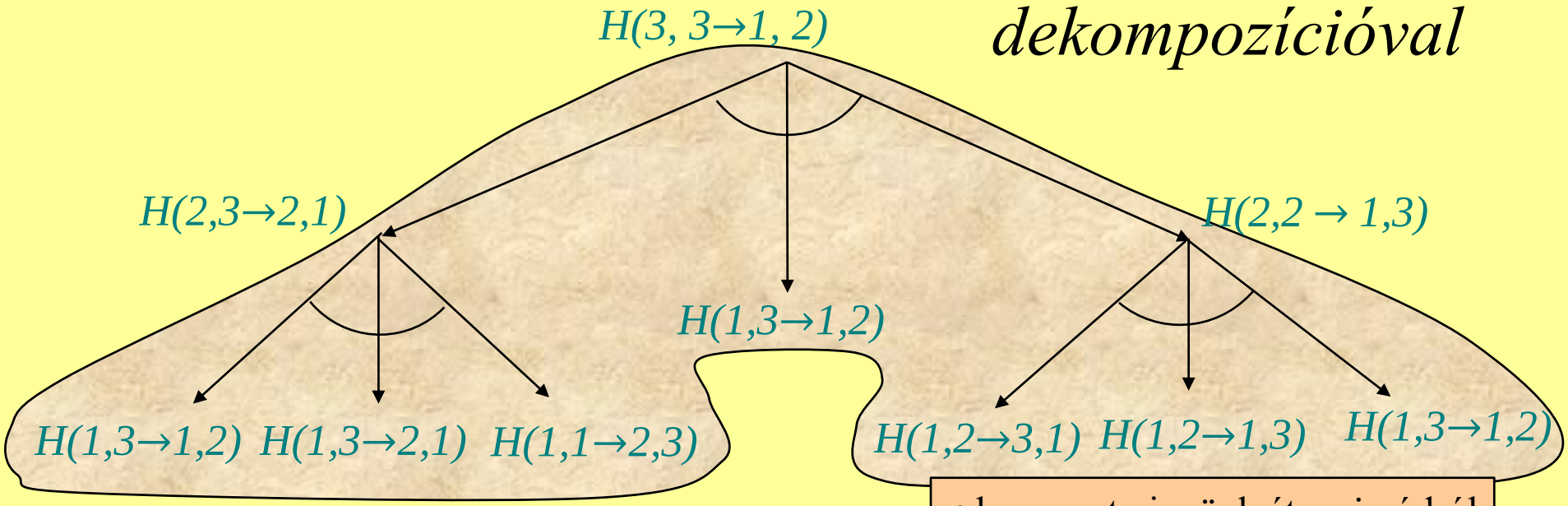


- **Probléma általános leírása**: *szintaktikusan helyes kifejezés*
- **Kiinduló probléma**: *adott integrál-kifejezés*
- **Egyszerű probléma**: *integráljel nélküli kifejezés*
- **Dekomponáló operátorok**: *integrálási szabályok*

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

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

Hanoi tornyai probléma megoldása dekompozícióval



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

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

Megoldás gráf: kiinduló problémát egyszerű problémákra visszavezető fa

Megoldás: a részfa leveleit kell balról jobbra haladva összeolvasni

Dekompozíciós modellezés fogalma

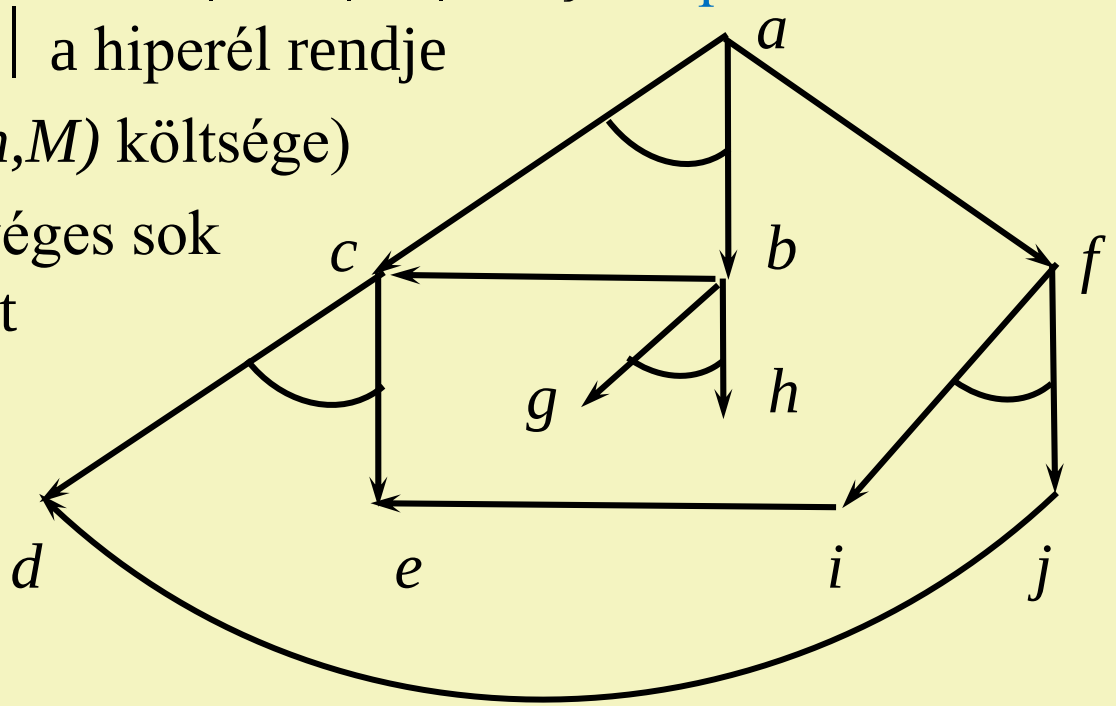
- A modellhez meg kell adnunk:
  - a feladat részproblémáinak általános leírását,
  - a kiinduló problémát,
  - az egyszerű problémákat, amelyekről könnyen eldönthető, hogy megoldhatók-e vagy sem, és
  - a dekomponáló műveleteket:
    - D: *probléma* $\rightarrow$ *probléma*$^+$ és
      $D(p)=<p_1,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

A dekompozíció modellezése ÉS/VAGY gráffal

- Dekompozíciós modell — ÉS/VAGY gráf
  - részprobléma ~ csúcs
  - dekomponáló művelet hatása egy problémára ~ irányított hiperél ugyanazon csúcsból induló ÉS kapcsolatban álló élek kötege
  - művelet költsége ~ hiperél költsége
  - kiinduló probléma ~ startcsúcs
  - megoldható probléma ~ célcsúcs
- Gráf-reprezentáció: ÉS/VAGY gráf, startcsúcs, célcsúcsok
  - dekompozíciós folyam ~ hiperút
  - megoldás ~ megoldás-gráf: egy hiperút startcsúcsból célcsúcsokba

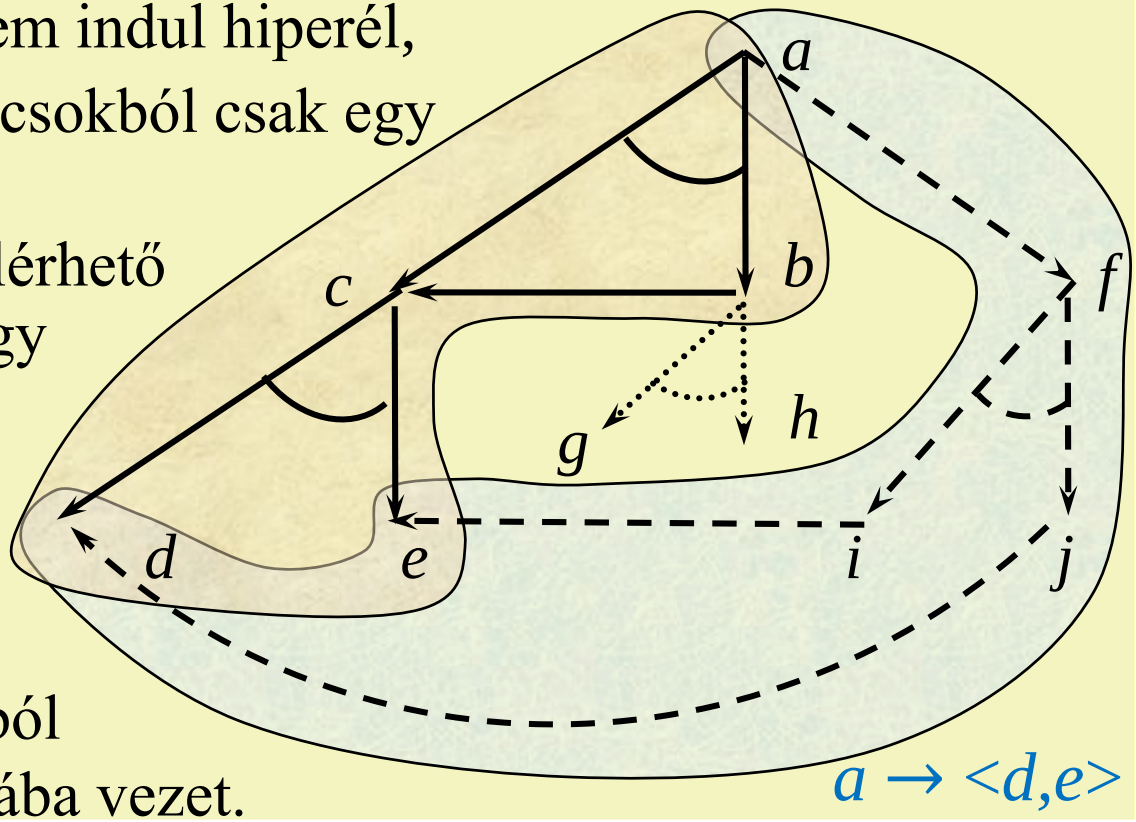
ÉS/VAGY gráfok

1. Az $R=(N,A)$ élsúlyozott irányított hipergráf, ahol az
   - N a csúcsok halmaza,
   - $A \subseteq \{ (n,M) \in N \times N^{+} \mid 0 \neq |M| < \infty \}$ a hiperélek halmaza, $|M|$ a hiperél rendje
   - ($c(n,M)$ az (n,M) költsége)
2. Egy csúcsból véges sok hiperél indulhat
3. $(0 < \delta \leq c(n,M))$



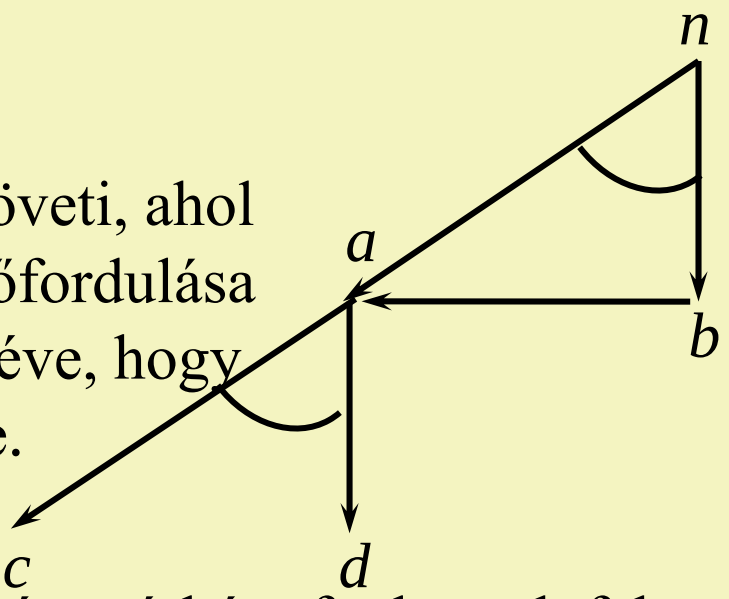
Az n csúcsból az M csúcs-sorozatba vezető irányított hiperút fogalma

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



A hiperút bejárása

- Az $n \rightarrow M$ hiperút egy bejárásán a hiperút csúcsaiból képzett sorozatoknak a felsorolását értjük:
  - az első sorozat: $<n>$
  - a C sorozatot a $C^{k \leftarrow K}$ sorozat követi, ahol a $k \in C$ csúcs ($k \notin M$) minden előfordulása helyére a K sorozatot írjuk feltéve, hogy van a hiperútnak (k,K) hiperéle.



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

$$<n> \rightarrow <a,b> \rightarrow <a,a> \rightarrow < c,d,c,d >$$

$$<n> \rightarrow <a,b> \rightarrow <c,d,b > \rightarrow <c,d,a> \rightarrow < c,d,c,d >$$

Útkeresés ÉS/VAGY gráfban

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