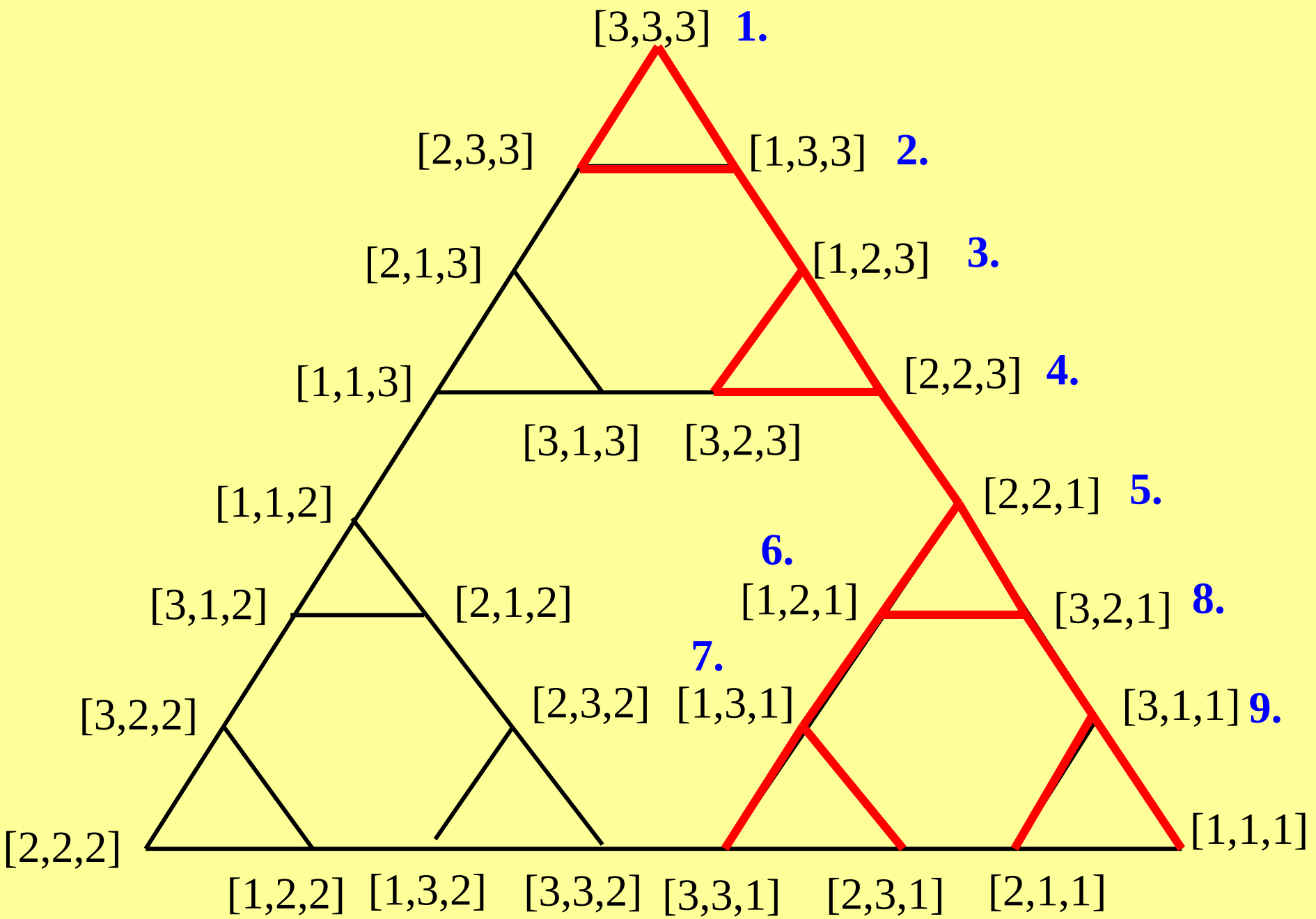
3. Gráfkeresés

- A gráfkeresés olyan KR, amelynek
  - globális munkaterülete: startcsúcsból kiinduló már feltárt útjai a reprezentációs gráfnak (keresőgráf), valamint a feltárt utak végei (nyílt csúcsok)
    - kiinduló értéke: a startcsúcs,
    - terminálási feltétel: vagy célcsúcsot 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

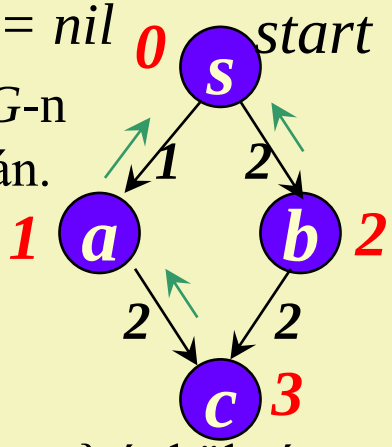
Jelölések:

- 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 eléggé jól ismerjük
- kiértékelő függvény (f: $OPEN \rightarrow \mathbb{R}$) : kiválasztja a megfelelő nyílt csúcsot kiterjesztésre


[3,3,3]
1.
[2,3,3]
[1,3,3]
2.
[2,1,3]
[1,2,3]
3.
[1,1,3]
[2,2,3]
4.
[3,1,3]
[3,2,3]
[1,1,2]
[2,2,1]
5.
6.
[3,1,2]
[2,1,2]
[1,2,1]
[3,2,1]
8.
7.
[3,2,2]
[2,3,2]
[1,3,1]
[3,1,1]
9.
[2,2,2]
[1,1,1]
[1,2,2]
[1,3,2]
[3,3,2]
[3,3,1]
[2,3,1]
[2,1,1]

Gráfkeresés függvényei

- π: $N \rightarrow N$ ***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 csúcs felfedezésekor a $\pi(m)$ a G-beli optimális $start \rightarrow m$ utat mutatná.



- g: $N \rightarrow \mathbb{R}$ ***költség függvény***
  - $g(m) = c^{\alpha}(start,m)$ − egy már megtalált $\alpha \in \{start \rightarrow n\}$ út költsége
    - Jó lenne ha egy m csúcs felfedezésekor 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 $\pi(m)$ ***konzisztens***: $g(m) = c^{\pi}(start,m)$, és $\pi(m)$ G-ben ***optimális***: $c^{\pi}(start,m) = \mathbf{min}_{\alpha \in \{start \rightarrow m\} \cap G}\, c^{\alpha}(start,m)$

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

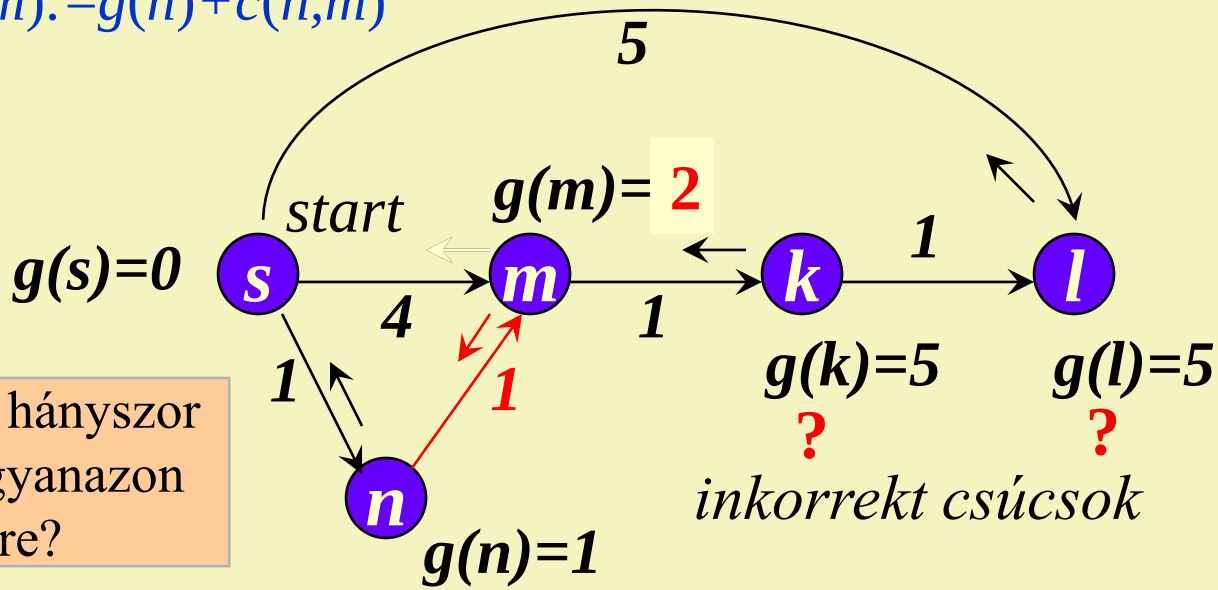
    azaz $m \in G$ és $g(n)+c(n,m)<g(m)$ akkor

    $\pi(m) := n,\ \ g(m) := g(n)+c(n,m)$
  - 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
$\pi(m):=n$, $g(m):=g(n)+c(n,m)$



Ezt választjuk. De hányszor kerülhet így sor ugyanazon csúcs kiterjesztésére?

- 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 függvénnyel, hogy ilyen történjen!
  3. Az *m* csúcsot helyezzük vissza *OPEN* halmazba!

ADAT := *kezdeti érték*
while ¬ *terminálási feltétel*(ADAT) l**oop**
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 mindig terminál.
- Ha egy véges δ-gráfban létezik megoldás, akkor a *GK* megoldás megtalálásával terminál.

3.2. Nevezetes gráfkereső algoritmusok

- Válasszunk *f* kiértékelő függvényt.

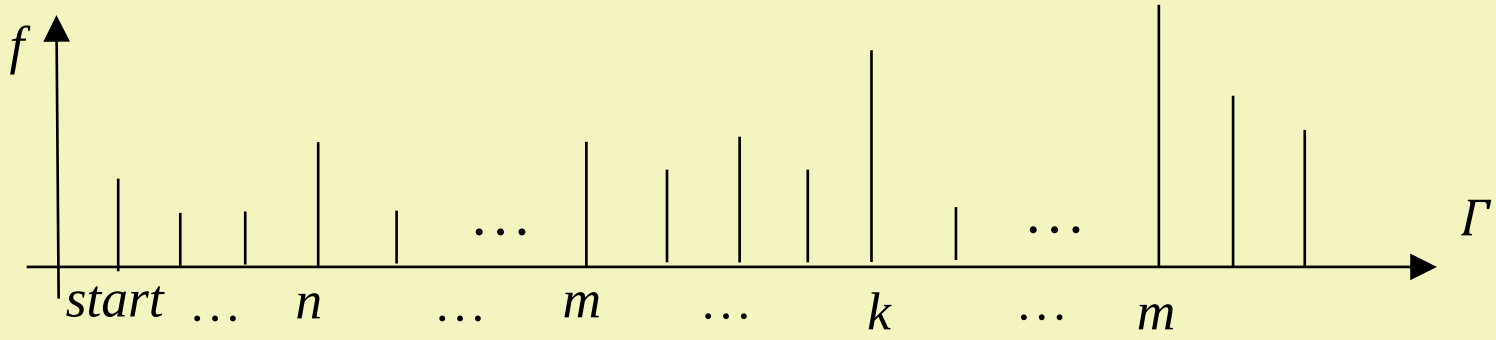
Nem-informált	Heurisztikus
mélységi (MGK)	előre tekintő (mohó, best-first)
szélességi (SZGK)	A, A^*, A^c
egyenletes (EGK)	B, B', A^{**}

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

Csökkenő kiértékelő függvény

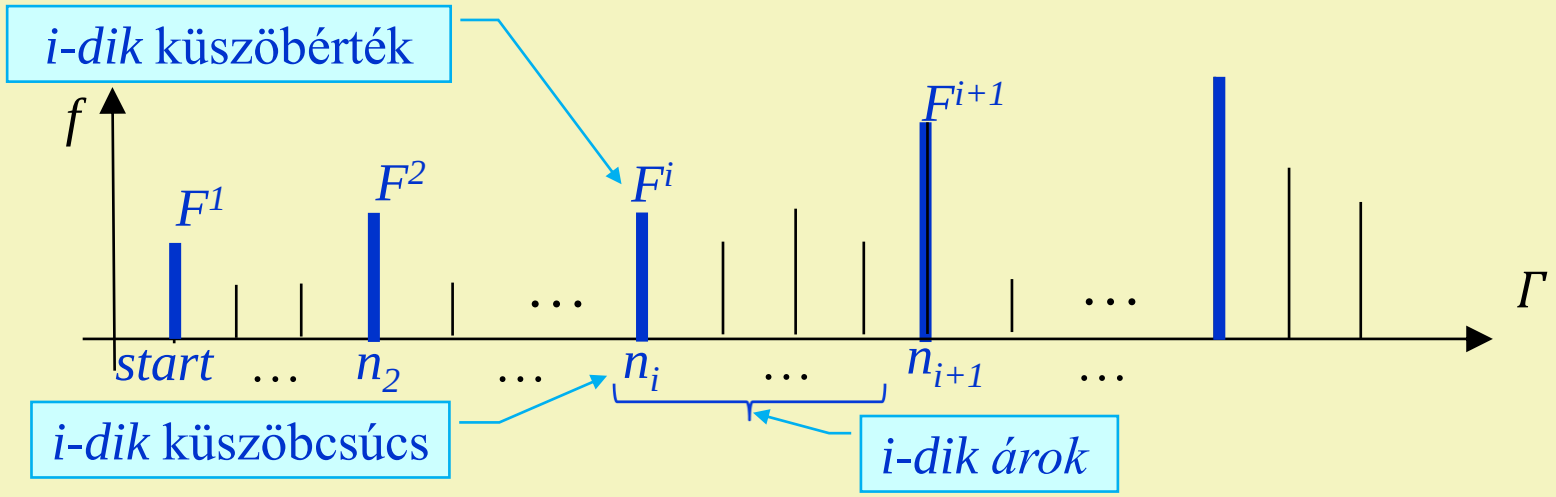
- Egy *GK* kiértékelő függvénye ***csökkenő***, amennyiben a egy csúcsra adott értéke az algoritmus működése során nem növekszik, viszont mindig csökken, valahányszor a csúcshoz a korábbinál olcsóbb utat találunk.
  - Például a *g* költségfüggvény önmagában ilyen.
- Csökkenő kiértékelő függvény mellett 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 ($i=1,2,\ldots$) monoton növekedő részsorozatot, amely a legelső értékkel kezdődik, majd mindig a legközelebbi 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	$f = -g$, $c(n,m) = 1$	• végtelen gráfokban csak mélységi korláttal garantál megoldást
Szélességi gráfkeresés SZGK	$f = g$, $c(n,m) = 1$	• 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	$f = g$	• optimális (legolcsóbb) megoldást ad, ha van (még végtelen δ-gráfban is) • egy csúcs kiterjesztésekor ismeri az odavezető legolcsóbb utat (legfeljebb egyszer terjeszt ki)

hasonlít Dijkstra legrövidebb utak algoritmusára

Heurisztika a gráfkereséseknél

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

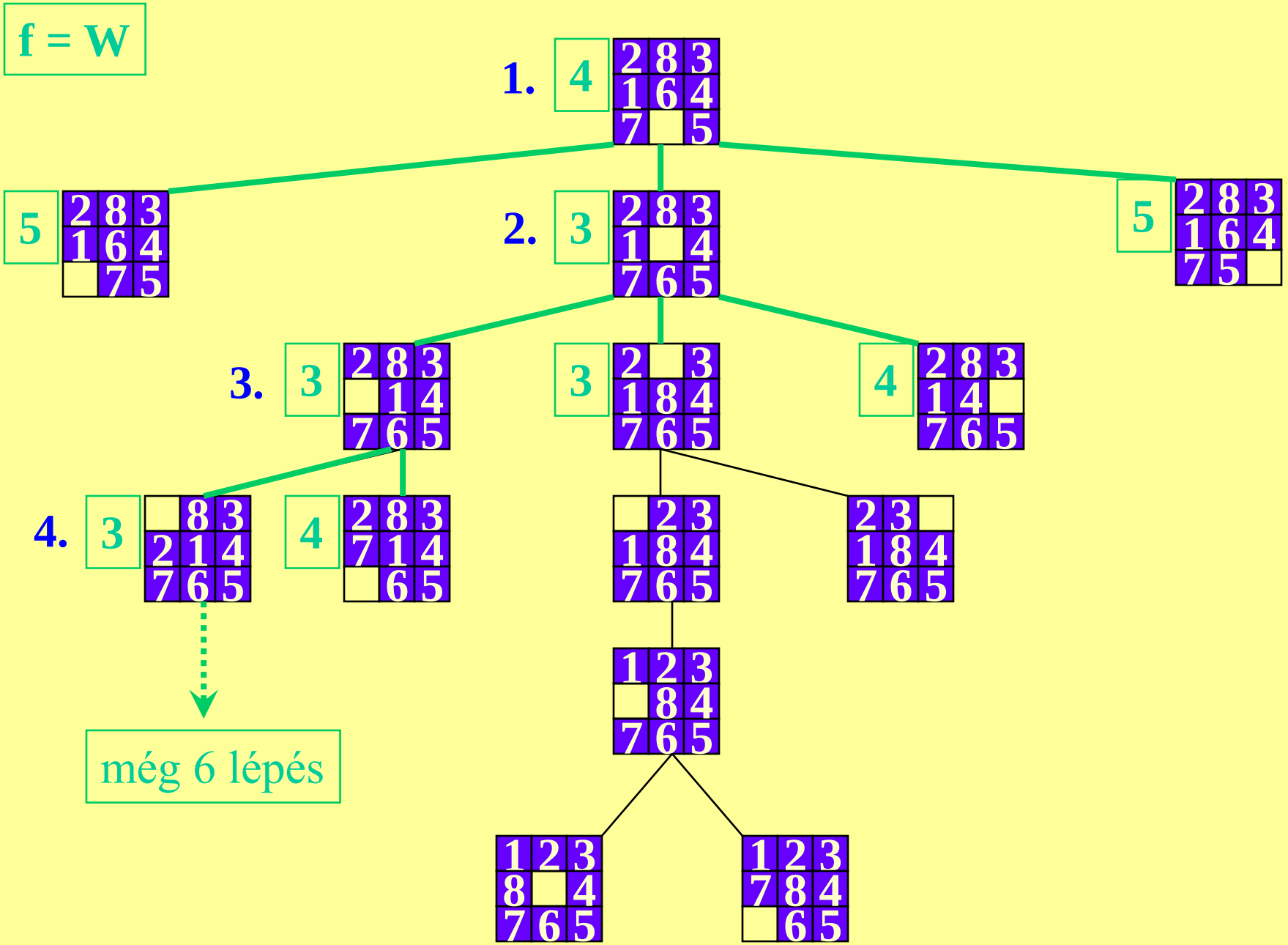
Heurisztikus függvények tulajdonságai

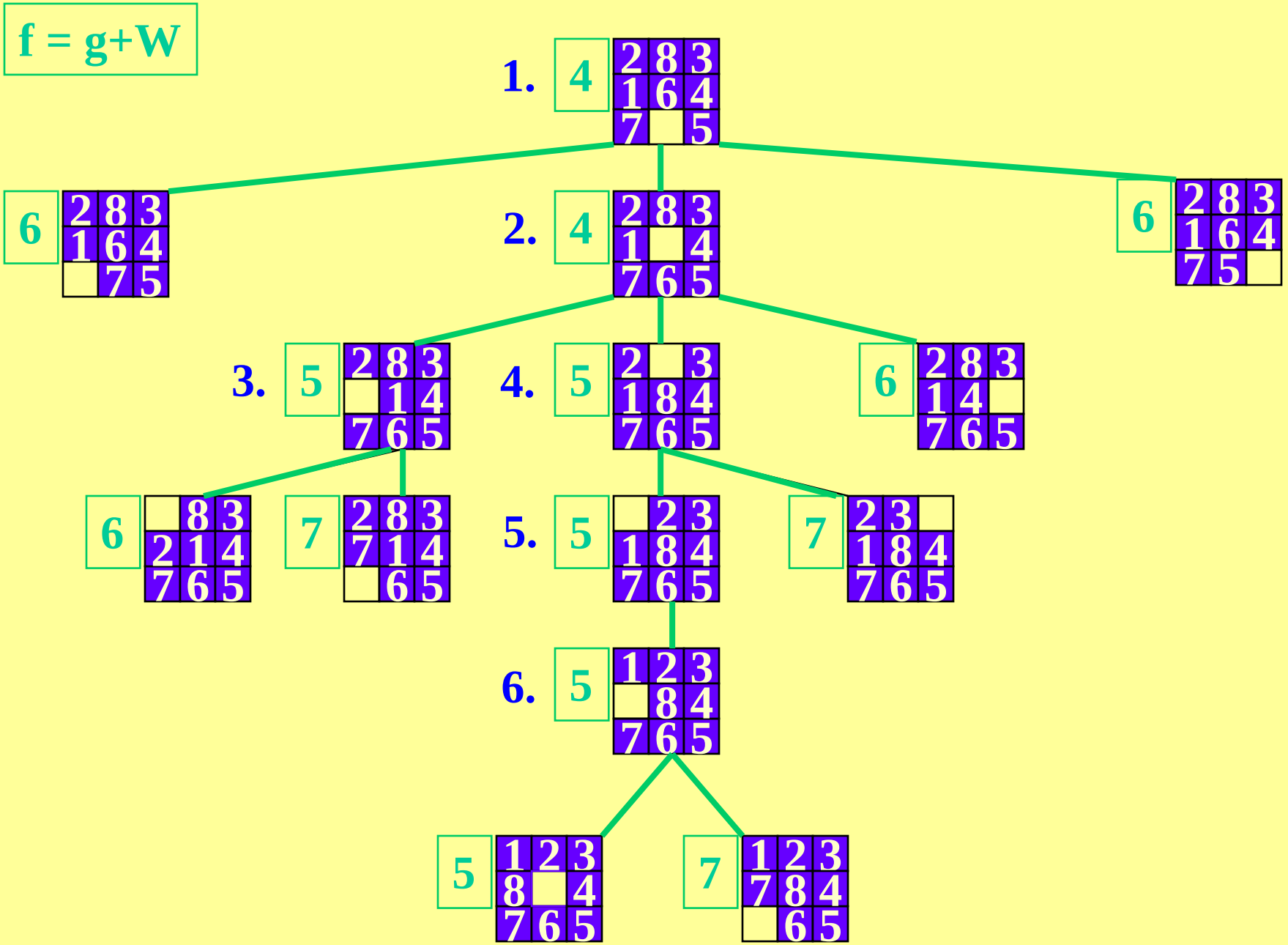
- Nevezetes tulajdonságok:
  - ***Nem-negatív***: $h(n) \geqq 0$ $\forall n \in N$
  - ***Megengedhető*** (admissible): $h(n) \leqq h^*(n)$ $\forall n \in N$
  - ***Monoton megszorítás***: $h(n) - h(m) \leqq c(n,m)$ $\forall (n,m) \in A$ (következetes)
- Megjegyzés:
  - *8*-kirakó : *W* és *P* mindhárom tulajdonsággal bír.
  - *h* monoton + *h* célban nulla ⇒ *h* megengedhető
  - *Zéró* függvény mindhárom tulajdonsággal bír.

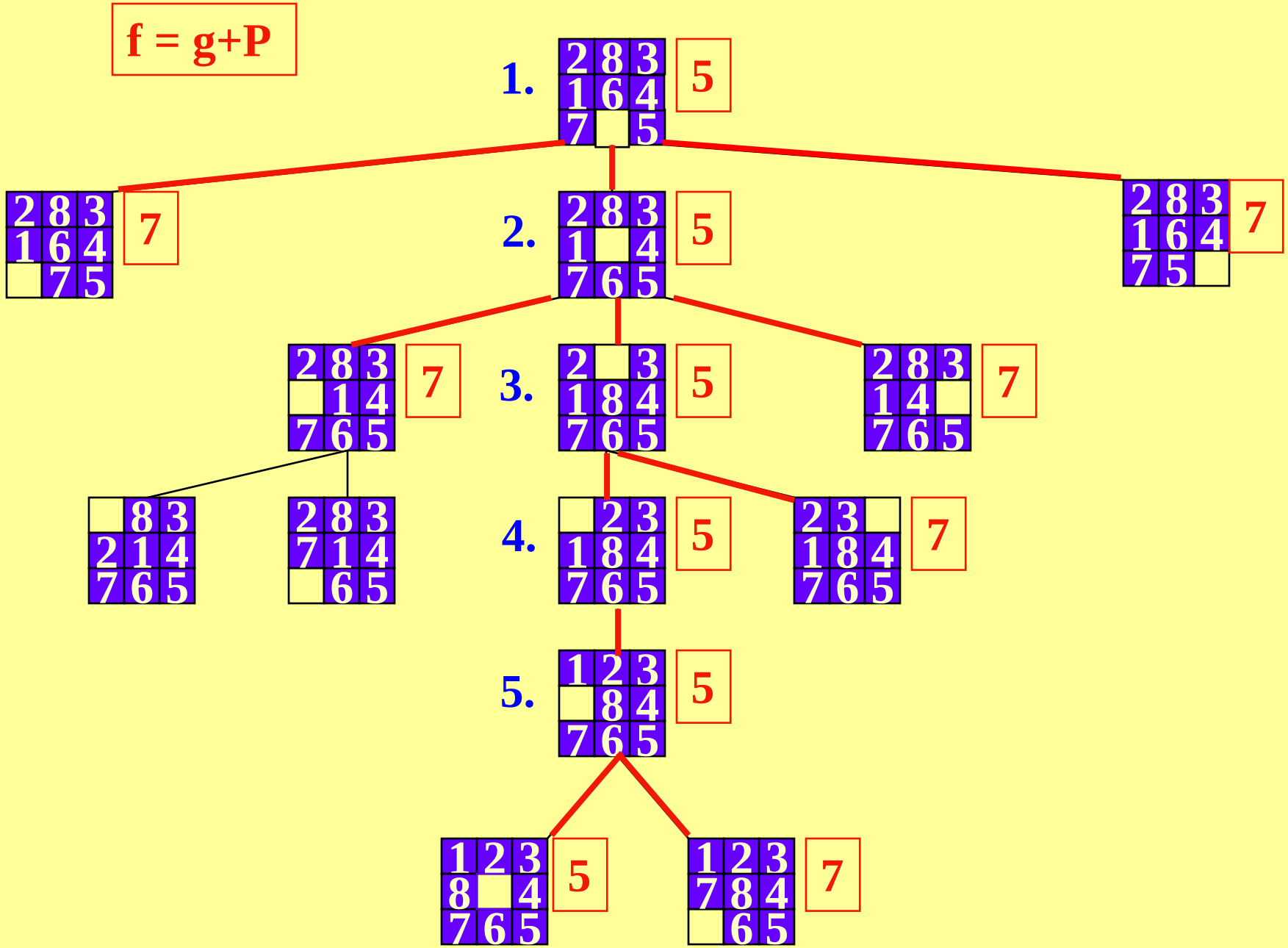
Nevezetes heurisztikus algoritmusok

Algoritmus	**Definíció**	**Eredmények**
Előre tekintő gráfkeresés	$f = h$	nincs említhető extra tulajdonsága
A algoritmus	$f = g+h$ *és* $h \geqq 0$	• megoldást ad, ha van megoldás (még végtelen δ-gráfban is)
A algoritmus*	$f = g+h$ *és* $h \geqq 0$ *és* $h \leqq h^*$	• optimális megoldást ad, ha van (még végtelen δ-gráfban is)
A^c algoritmus	$f = g+h$ *és* $h \geqq 0$ *és* $h \leqq h^*$ *és* $h(n)-h(m) \leqq c(n,m)$	• optimális megoldást ad, ha van (még végtelen δ-gráfban is) • egy csúcs kiterjesztésekor ismeri az odavezető legolcsóbb utat (legfeljebb egyszer terjeszt ki)

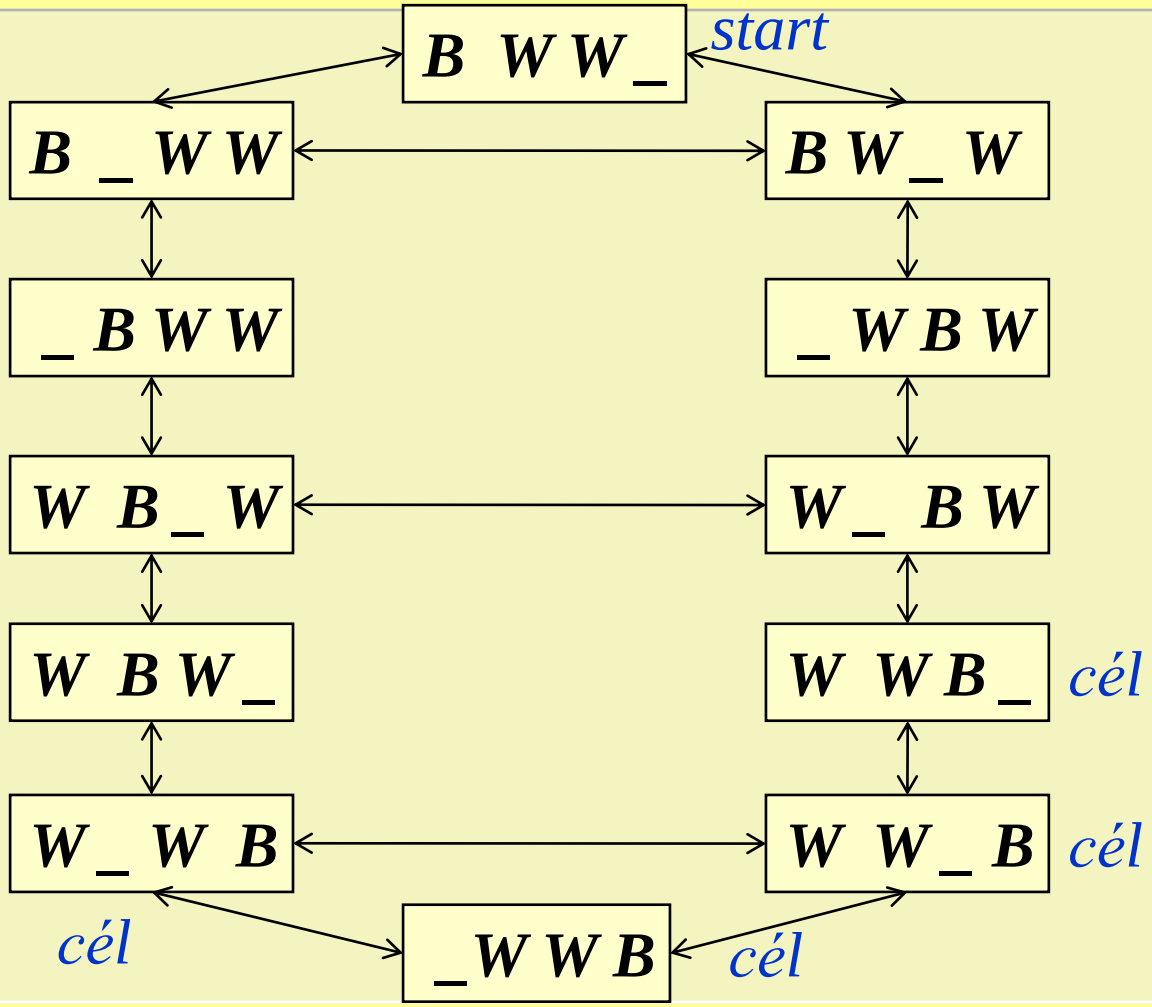
egyenletes gráfkeresés:
$f = g+0$


f = W
1.
4
2 8 3
1 6 4
7 5
5
2 8 3
1 6 4
7 5
2.
3
2 8 3
1 4
7 6 5
5
2 8 3
1 6 4
7 5
3.
3
2 8 3
1 4
7 6 5
3
2 3
1 8 4
7 6 5
4
2 8 3
1 4
7 6 5
4.
3
8 3
2 1 4
7 6 5
4
2 8 3
7 1 4
6 5
2 3
1 8 4
7 6 5
2 3
1 8 4
7 6 5
1 2 3
8 4
7 6 5
még 6 lépés
1 2 3
8 4
7 6 5
1 2 3
7 8 4
6 5


f = g+W
1.
4
2 8 3
1 6 4
7 5
6
2 8 3
1 6 4
7 5
2.
4
2 8 3
1 4
7 6 5
6
2 8 3
1 6 4
7 5
3.
5
2 8 3
1 4
7 6 5
4.
5
2 3
1 8 4
7 6 5
6
2 8 3
1 4
7 6 5
6
8 3
2 1 4
7 6 5
7
2 8 3
7 1 4
6 5
5.
5
2 3
1 8 4
7 6 5
7
2 3
1 8 4
7 6 5
6.
5
1 2 3
8 4
7 6 5
5
1 2 3
8 4
7 6 5
7
1 2 3
7 8 4
6 5

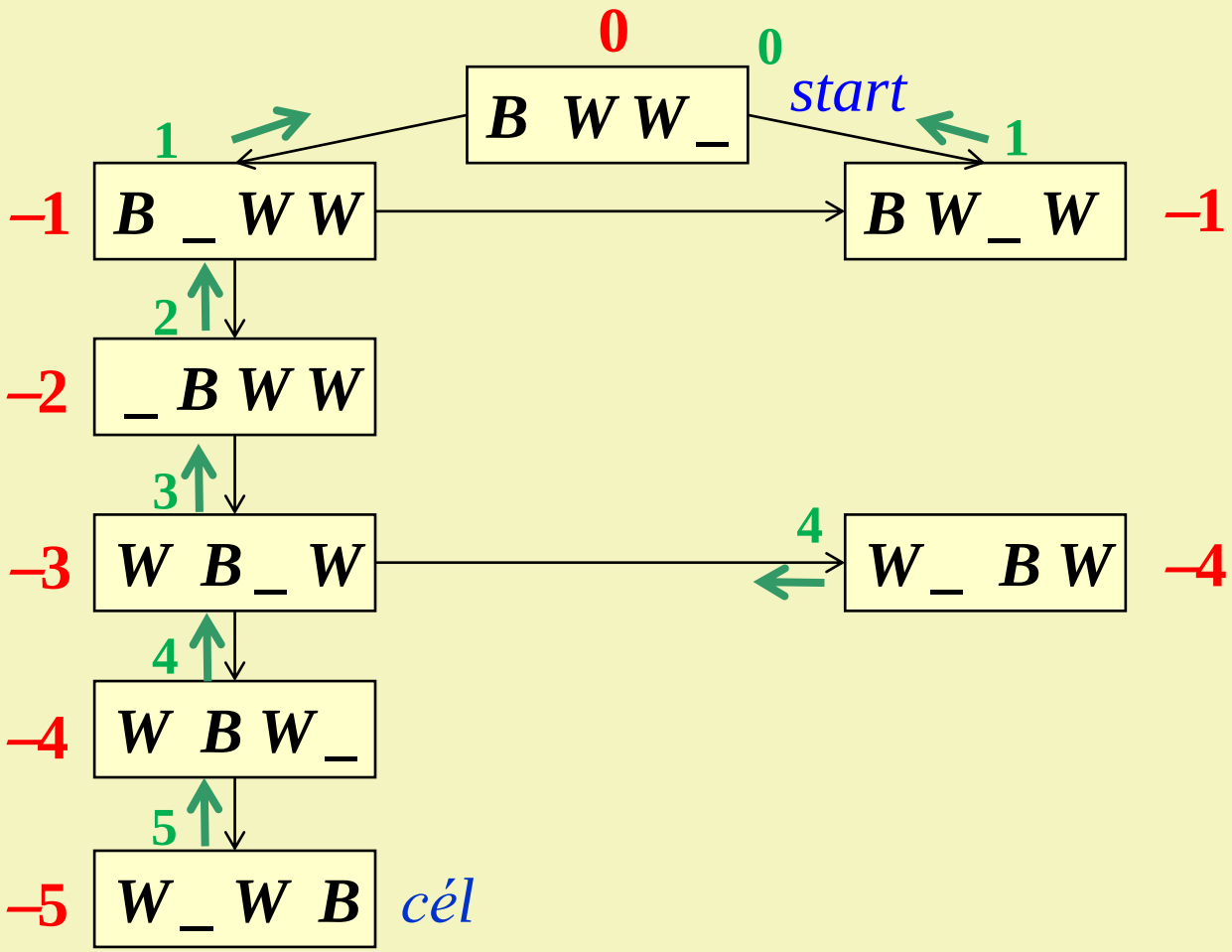

f = g+P
1.
2.
3.
4.
5.

Fekete-fehér kirakó állapot gráfja



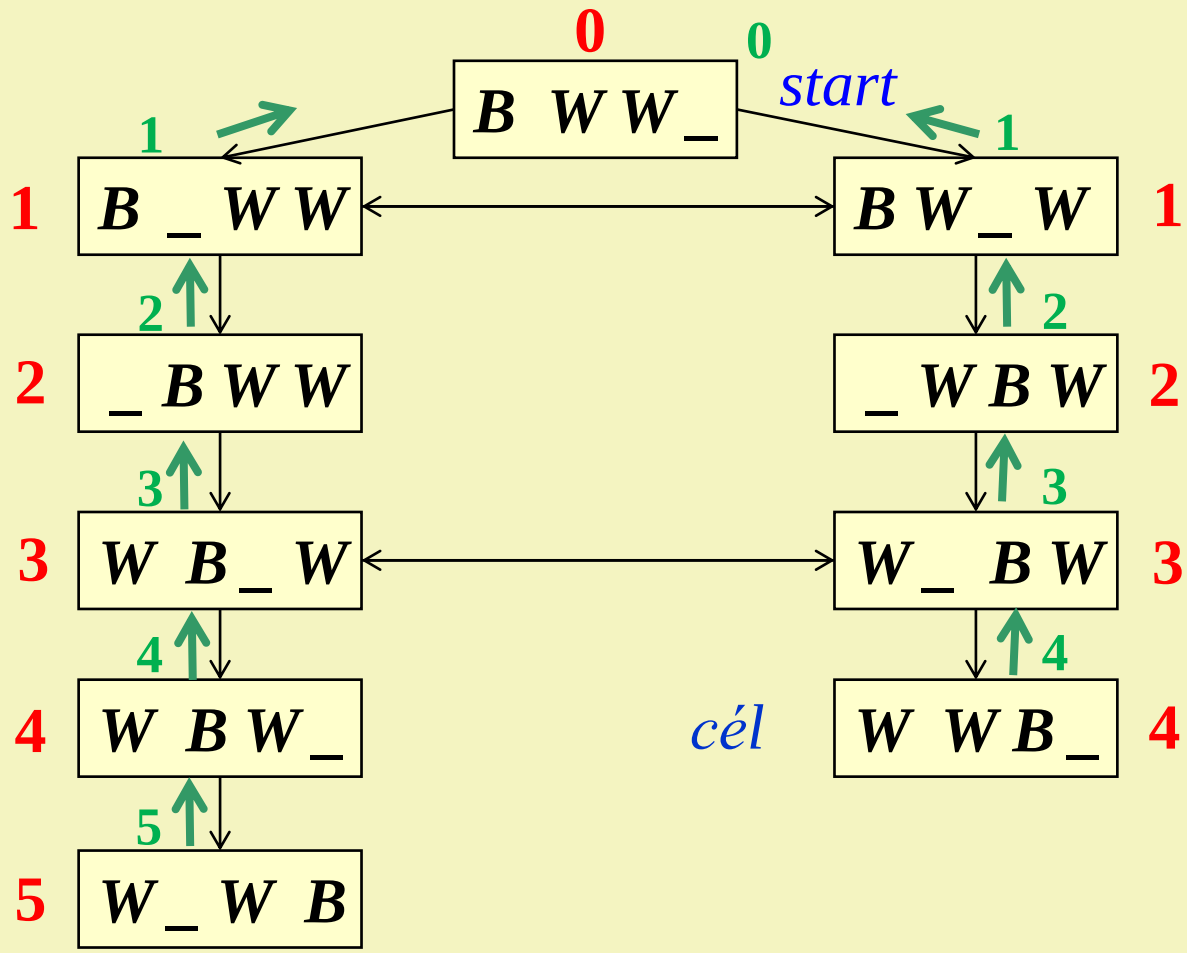
Mélységi gráfkeresés

$f = -g$



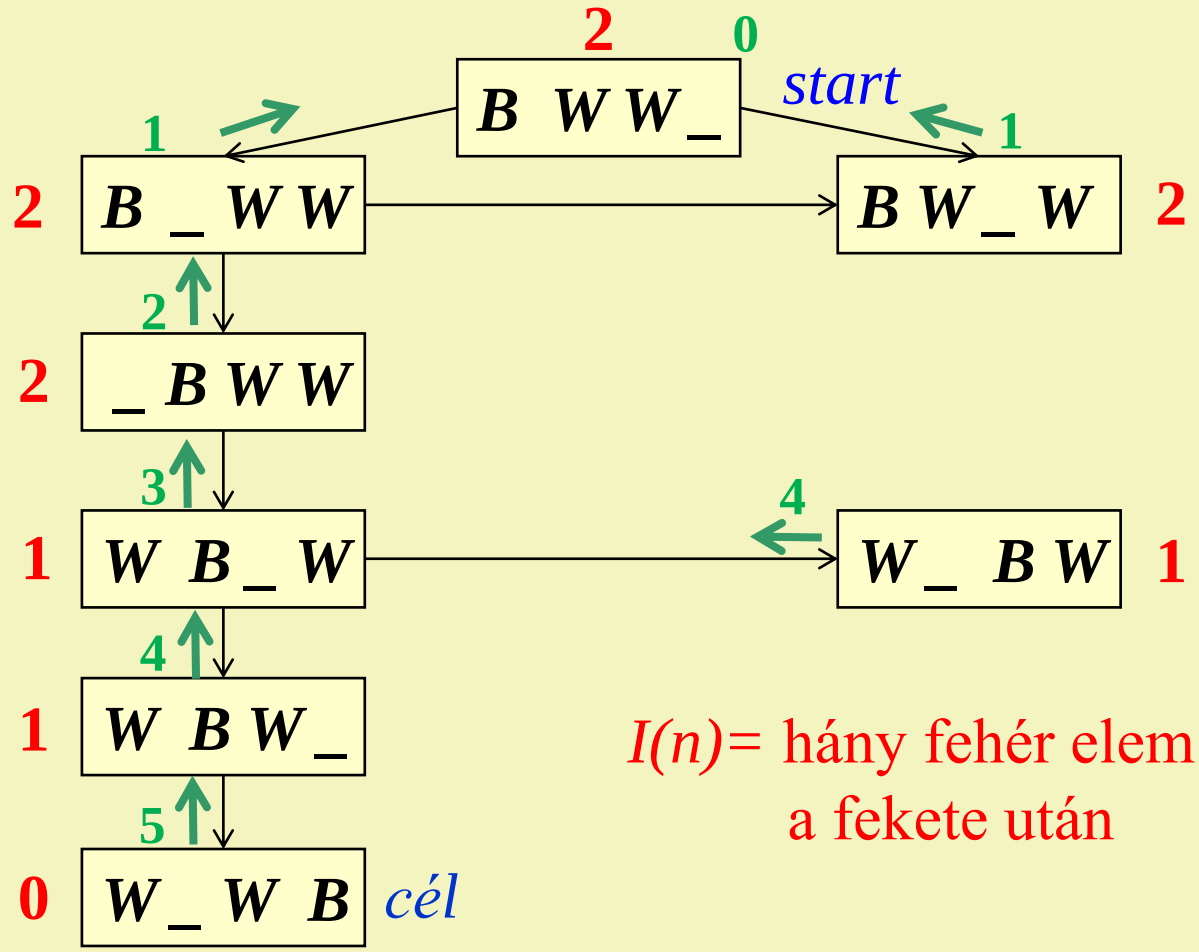
Szélességi gráfkeresés

$f = g$



Előre tekintő gráfkeresés

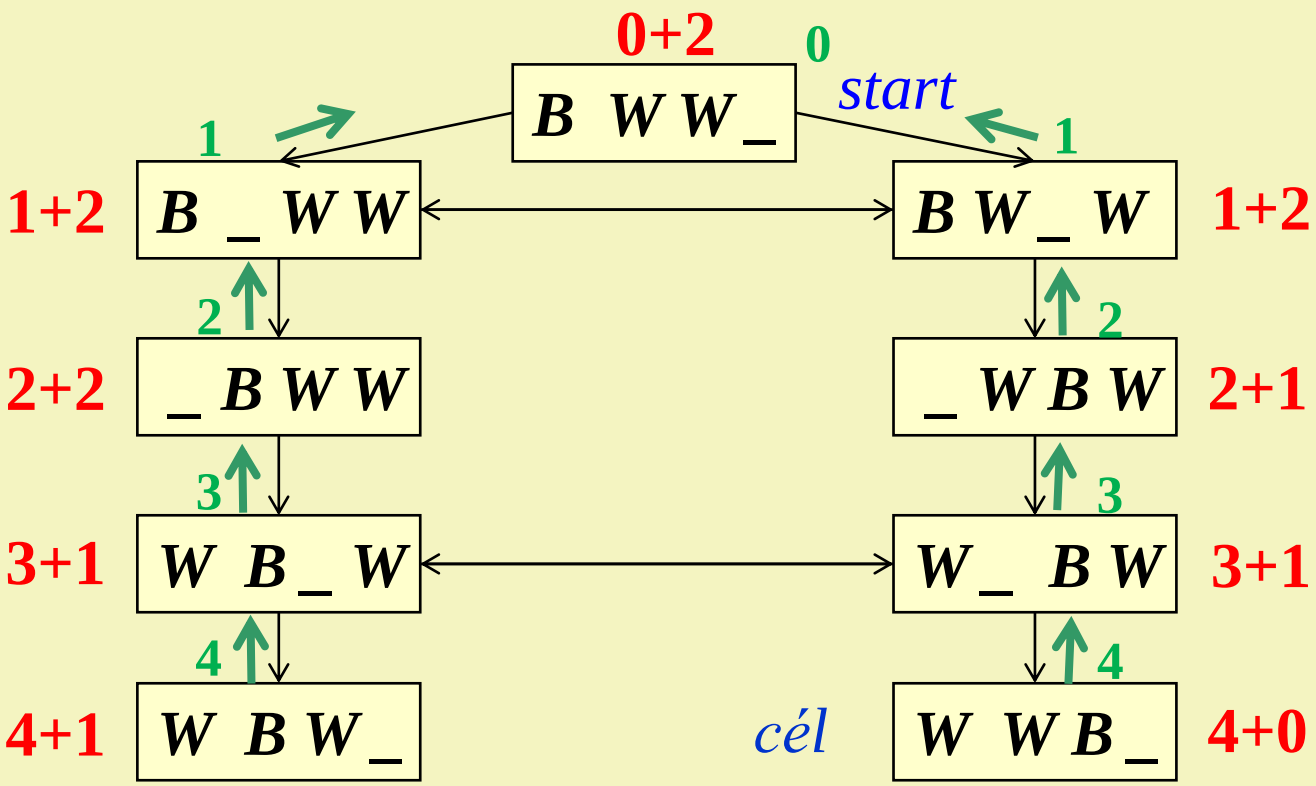
$f = I$



I(n)= hány fehér elem áll a fekete után

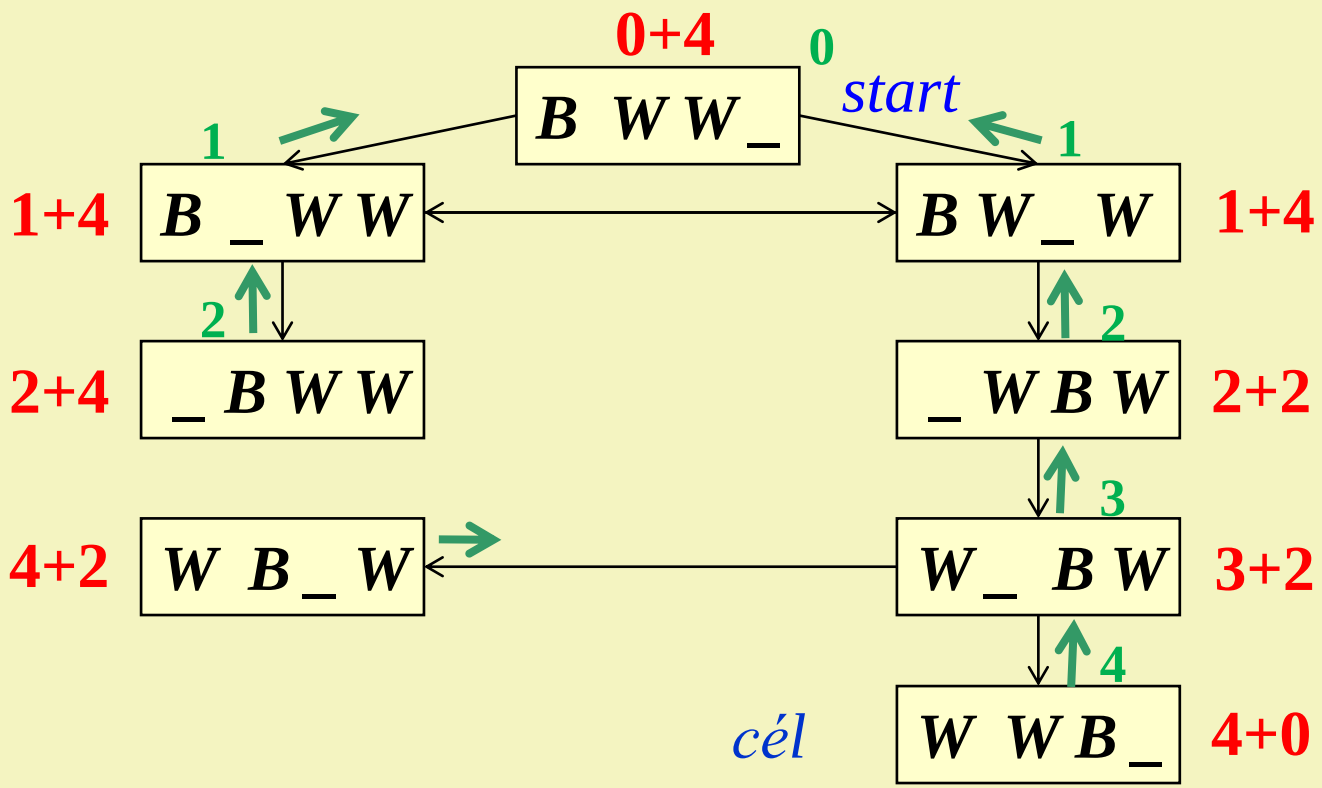
A algoritmus

$f = g + I$



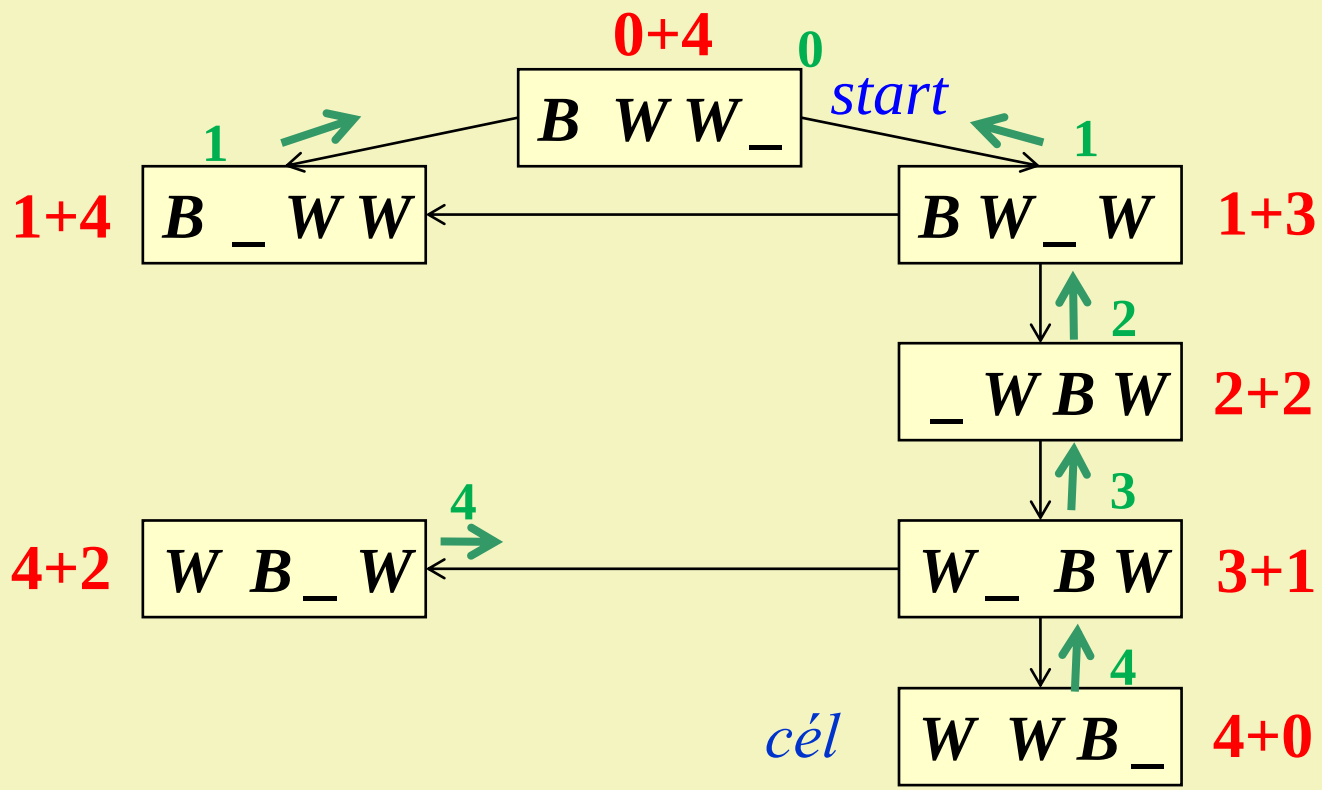
A algoritmus

$f = g + 2*I$



A algoritmus

*f = g + 2*I – (*1 ha van *BW_* vagy *_BW)*



Elemzés

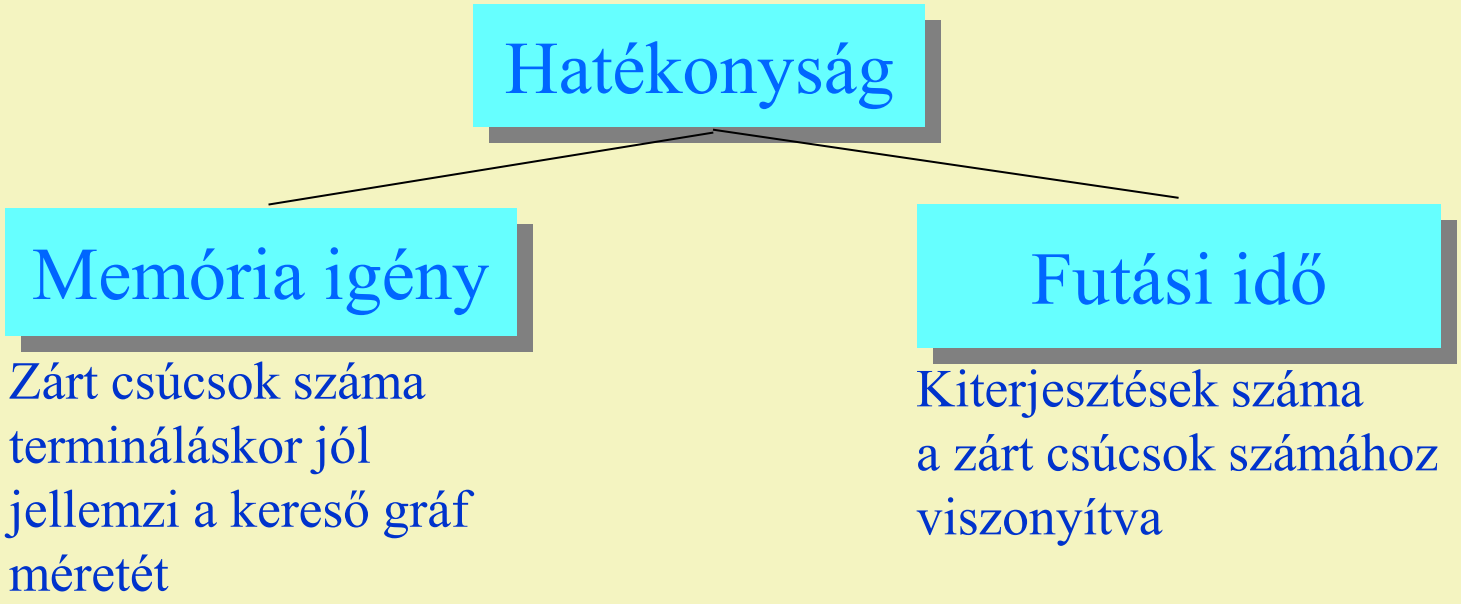
f	Alg	mo	G	Γ
-g	*MGK*	5	8	5
g	*SZGK*	4	10	8
l	*Előre tekintő*	5	8	5
g+l	*A alg*	4	9	7
*g+2*l*	*A alg*	4	8	6
*g+2*l–1(ha…)*	*A alg*	4	7	5

A^c alg

A^c alg

A^c alg

3.3. *A* algoritmus* hatékonysága



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 $CLOSED_X \subset CLOSED_Y$

- 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 $h_1(n) < h_2(n)$.

$h_1(n) < h_2(n) \leqq h^*(n)$

- 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 $h_1 \leqq h_2$ teljesül, esetleg nem is minden csúcsra.
  - *Példák:*
    - *8-as tologató:* $0 < W \leqq P$ ($\leqq F$)
    - *Fekete-fehér:* $I \leqq M$ ($\leqq 2 \cdot I$)
- 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:* $f(n)=\mathbf{max}_{m \in start \rightarrow n}(g(m)+h(m))$ és a célcsúcs 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

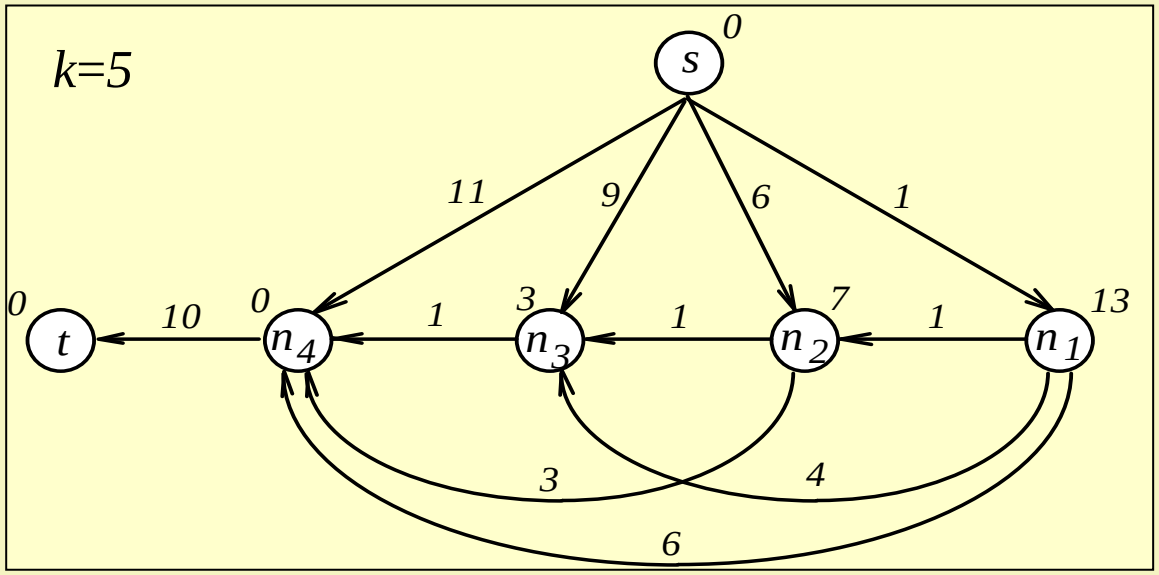
3.3.2. A futási idő elemzése

- 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 $k_1 << k_2$.

Martelli példája



Az $n_1,\ldots,n_{k-1}$ csúcsokba rendre $2^0,2^1,\ldots,2^{k-2}$ különböző út vezet. Így elvileg 2^{k-1} kiterjesztés történhet. És itt ennyi is történik.

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

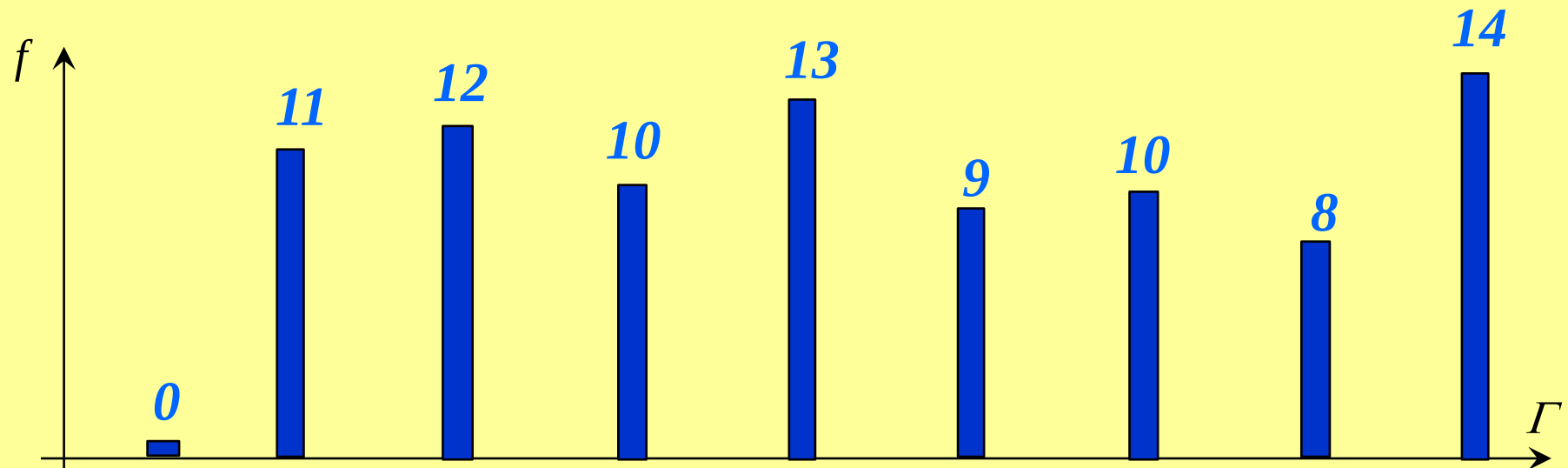
$A = \{(n_i,n_j) \mid 0 \le i < j < k\} \cup \{(n_{k-1},t)\}$

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

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

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

Működési grafikon

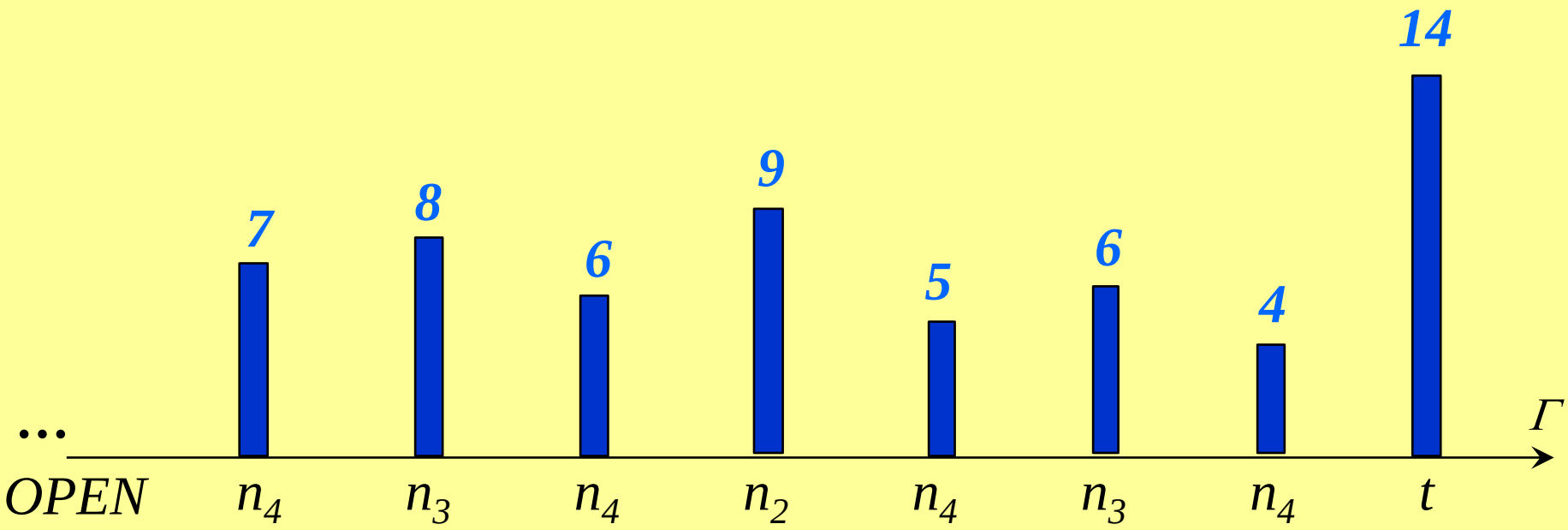


OPEN s n_4 n_3 n_4 n_2 n_4 n_3 n_4 n_1

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

π,g,f

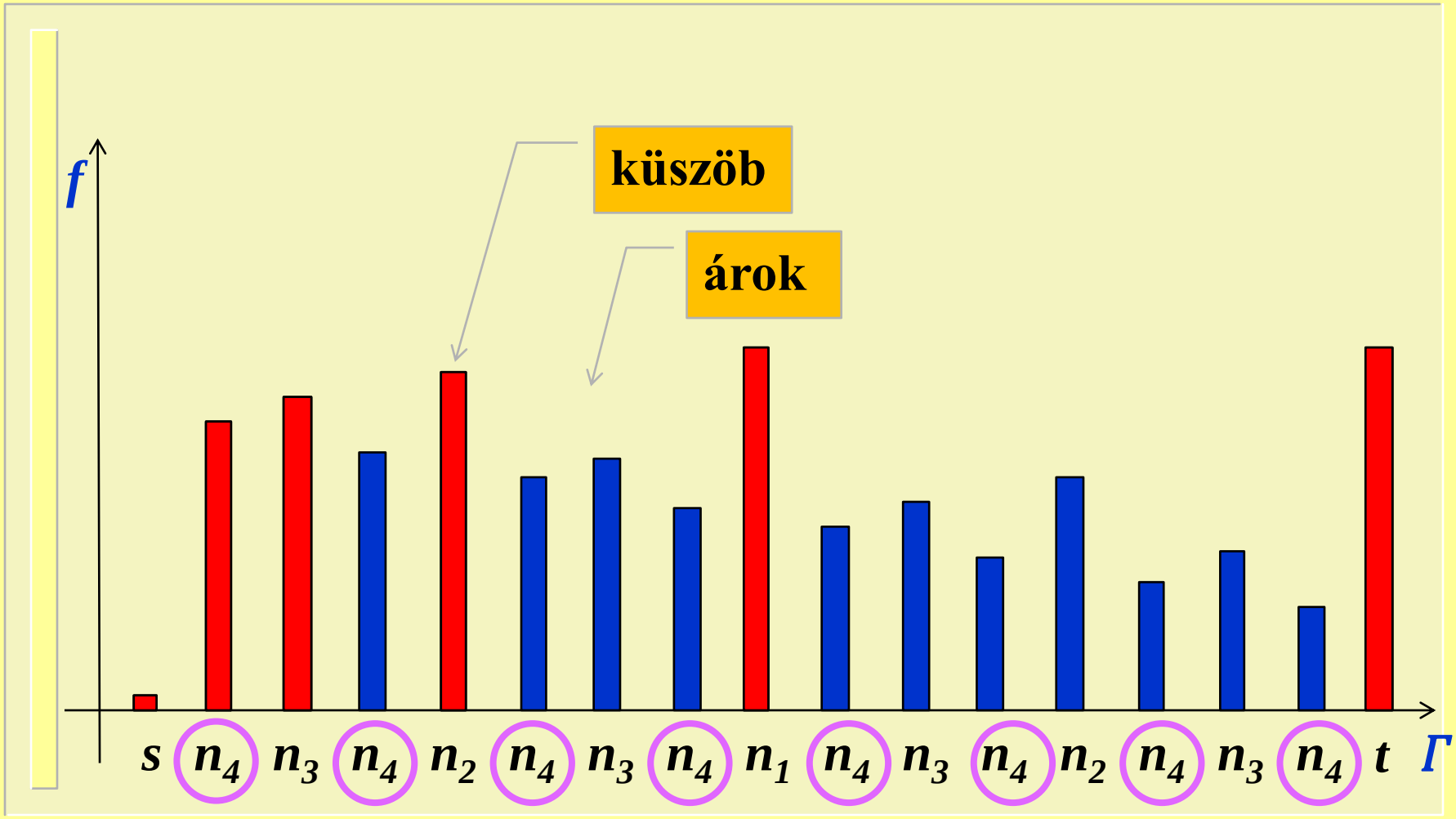
Működési grafikon



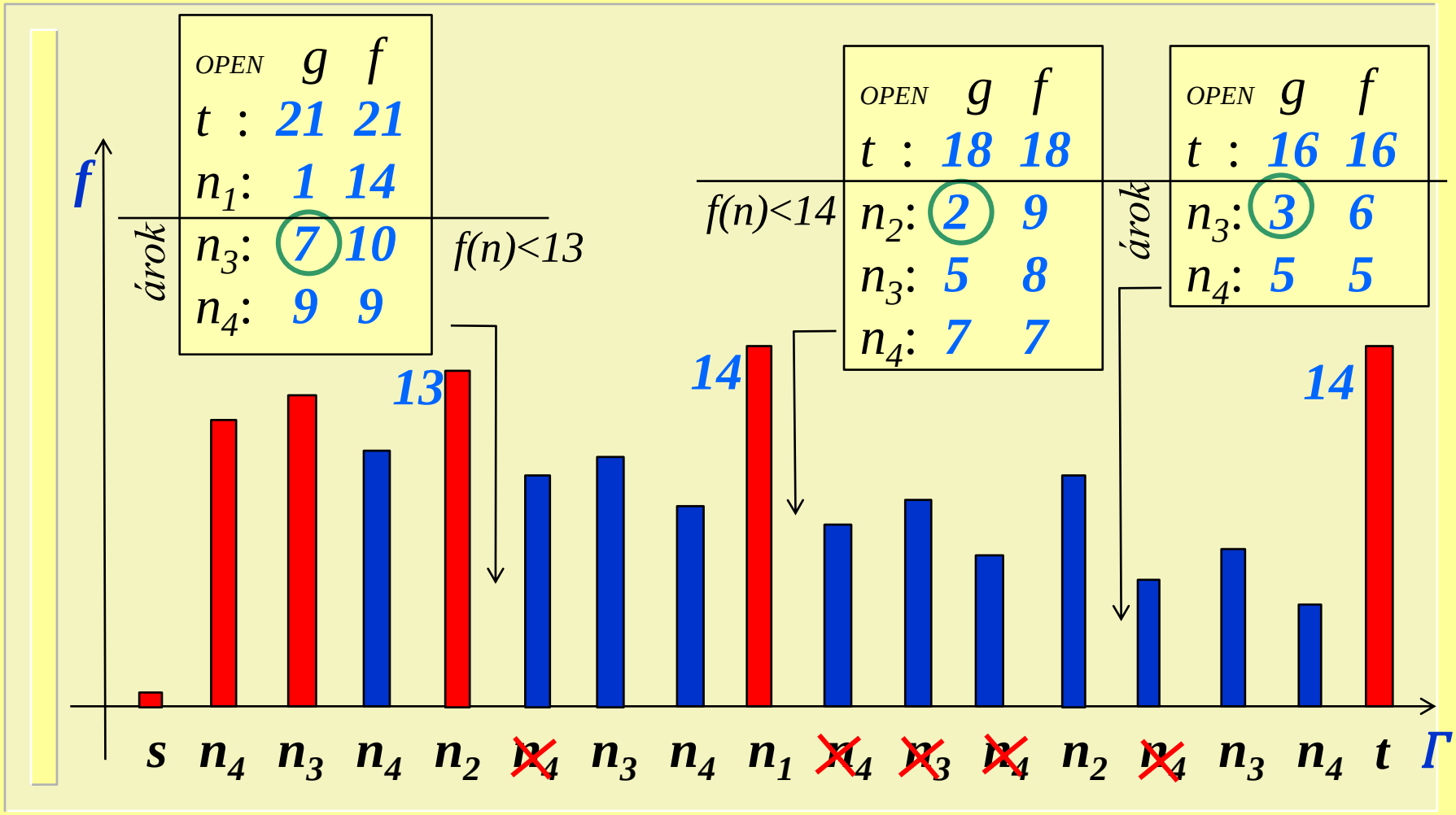
s	-	-	-	-	-	-	-	-
n_1	-	-	-	-	-	-	-	-
n_2	n_1,2,9	n_1,2,9	n_1,2,9	n_1,2,9	-	-	-	-
n_3	n_1,5,8	n_1,5,8	-	-	n_1,3,6	n_1,3,6	-	-
n_4	n_1,7,7	-	n_3,6,6	-	n_2,5,5	-	n_3,4,4	-
t	n_4,18,18	n_4,17,17	n_4,17,17	n_4,16,16	n_4,16,16	n_4,15,15	n_4,15,15	n_4,14,14

π,*g*,*f*

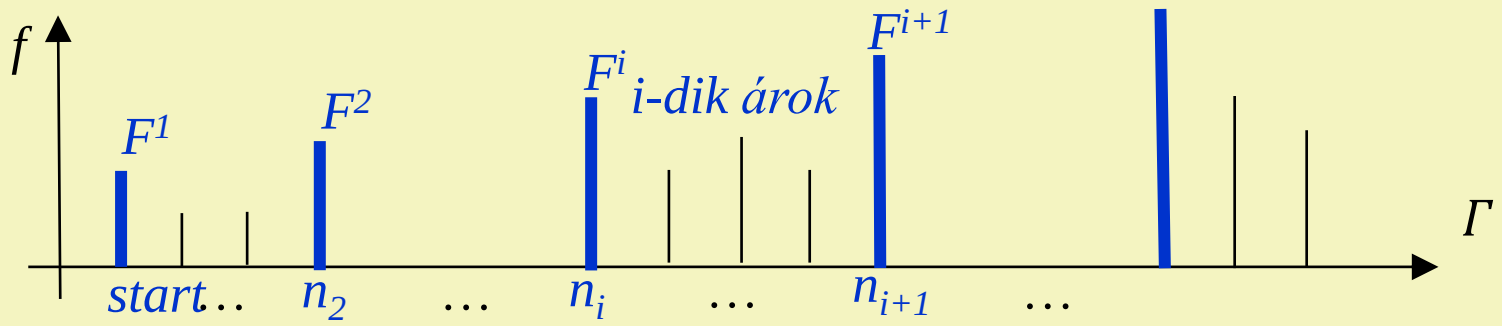
Árkon belüli kiterjesztések száma az A algoritmusnál*



Csökkentsük a kiterjesztések számát



A probléma oka és csillapítása



- Egy csúcs – még akár egy árkon belül is – többször kiterjesztődhet.
- Használjunk az árkokban egy másik, egy másodlagos (belső) 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 $\boldsymbol{min}_f(OPEN)<F$

    then $n := \boldsymbol{arg\ min}_g(m \in OPEN \mid f(m)<F)$

    else $n := \boldsymbol{arg\ min}_f(OPEN)$; $F := f(n)$

    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 kiterjesztésekor lehet küszöb.)
    - Az *i*-dik árok legfeljebb az összes addigi $i-1$ darab küszöbcsúcsot tartalmazhatja (a start csúcs nélkül).
  - Így az összes kiterjesztések száma legfeljebb $\frac{1}{2} \cdot k^2$

Heurisztika szerepe

- Milyen a jó heurisztika?
  - megengedhető: $h(n) \leq h^*(n)$
    - Bár nincs mindig 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