

3. Gráfkeresés

- ❑ A gráfkeresés olyan KR, amelynek
 - globális munkaterülete: **startcsúcsból kiinduló már feltárt útvai** 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**
 - vezérlési stratégiája: a **legkedvezőbb csúcs kiterjesztésére** törekszik, és ehhez egy kiértékelő függvényt használ.

3.1. Általános gráfkereső algoritmus

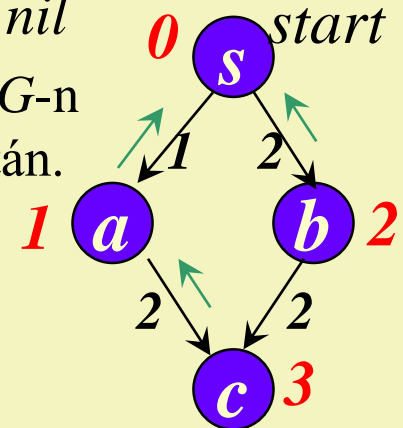
Jelölések:

- 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

Gráfkeresés függvényei

□ $\pi: N \rightarrow N$ *szülőre visszamutató pointer*

- $\pi(m) = m$ csúcs már 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.

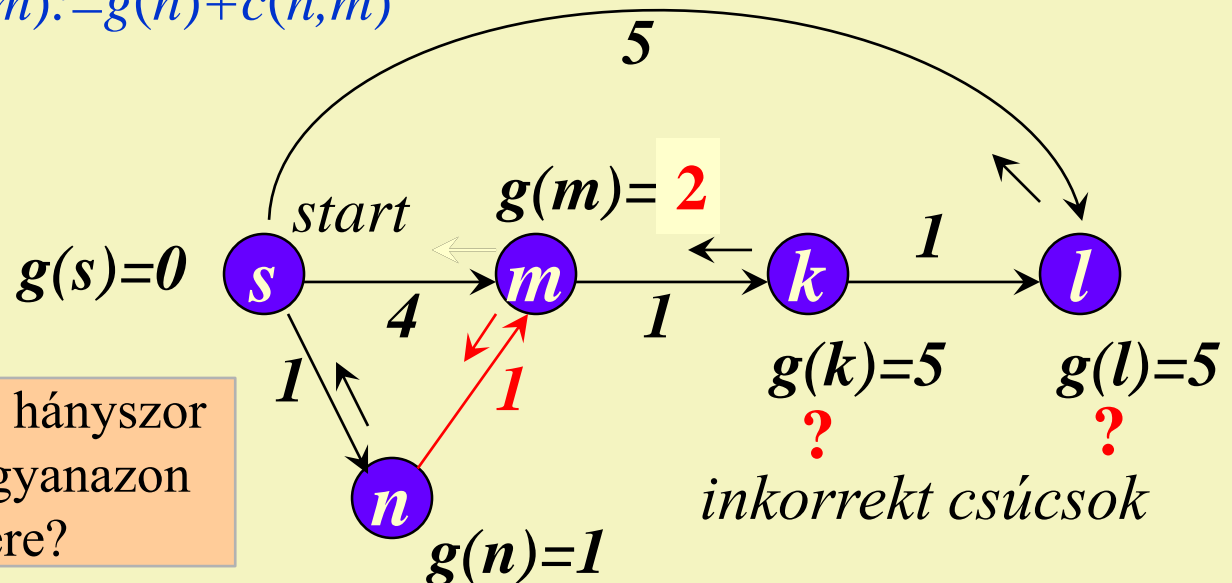
$m \in G$ csúcs *korrekt*, ha $g(m)$ és $\pi(m)$ *konzisztens*: $g(m) = c^{\pi}(start, m)$,
és $\pi(m)$ *optimális*: $c^{\pi}(start, m) = \min_{\alpha \in \{start \rightarrow m\} \cap G} c^\alpha(start, m)$
 G korrekt, ha minden csúcsa korrekt.

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

ADAT := kezdeti érték

```
while  $\neg$  terminálási feltétel(ADAT) loop  
    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. $n := \min_f(OPEN)$
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**

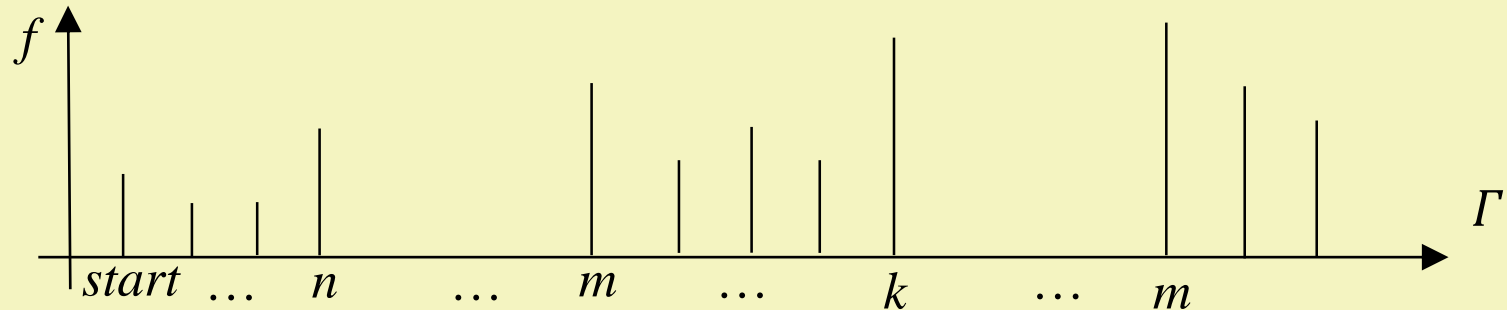
$\Gamma(akt) \sim$ akt gyermekei
 $\pi(akt) \sim$ akt egy szülője

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.

Gráfkeresés működési grafikonja

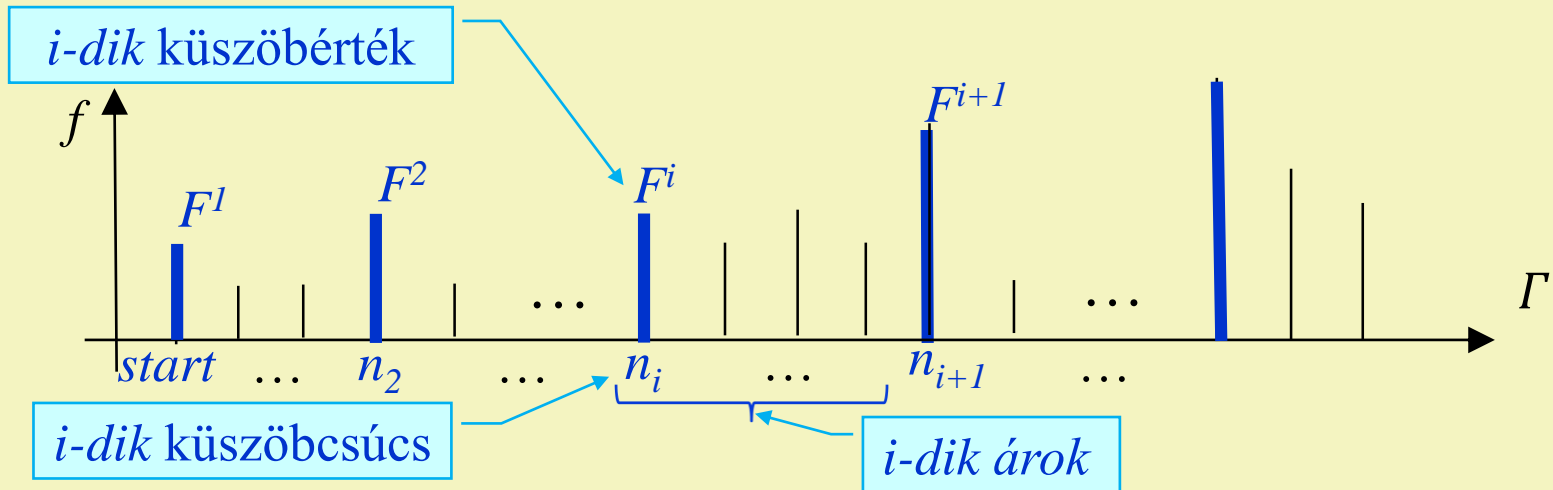


- Soroljuk fel a kiterjesztett csúcsokat kiterjesztésük sorrendjében (ugyanaz a csúcs többször is szerepelhet, hiszen többször is kiterjesztődhet) a kiterjesztésükkor mért f kiértékelő függvényértékükkel.

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 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 optimálisságát és konzisztenciáját.

Mikor lesz a kereső gráf korrekt csökkenő kiértékelő függvény mellett?



- ❑ Válasszuk ki az értékekből azt az F^i ($i=1,2,\dots$) monoton növekedő részsorozatot, amely a legelső értékkel kezdődik, majd mindig a legközelebbi nem kisebb értékkel folytatódik.
- ❑ Csökkenő kiértékelő függvény használata mellett a GK
 - kereső gráfja korrekt lesz valahányszor küszöbcsúcsot terjeszt ki
 - soha nem terjeszt ki inkorrekt csúcsot

3.2. Nevezetes gráfkereső algoritmusok

- Most az f kiértékelő függvény megválasztása következik.

Nem-informált

- mélységi (MGK)
- szélességi (SZGK)
- egyenletes (EGK)

Heurisztikus

- előre tekintő (mohó, best-first)
- A, A^*, A^c
- B, B', A^{**}

- Az úgynevezett tie-breaking rule-ok (egyenlőséget feloldó szabályok) a nem-informált gráfkereséseknél is tartalmazhatnak heurisztikát.

Nevezetes nem-informált algoritmusok

ugyanúgy mélységi stratégiát használ,
mint a visszalépéses keresés

Algoritmus	Definíció	Eredmények
<i>Mélységi gráfkeresés MGK</i>	$f = -g,$ $c(n,m) = 1$	<ul style="list-style-type: none"> • végtelen gráfokban csak mélységi korláttal garantál megoldást
<i>Szélességi gráfkeresés SZGK</i>	$f = g,$ $c(n,m) = 1$	<ul style="list-style-type: none"> • optimális (legrövidebb) megoldást ad, ha van (még végtelen δ-gráfban is) • egy csúcs kiterjesztésekor ismeri az odavezető legrövidebb utat (legfeljebb egyszer terjeszti ki)
<i>Egyenletes gráfkeresés EGK</i>	$f = g$	<ul style="list-style-type: none"> • 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 („hátralevő”) optimális út költségét.

- $h(n) \approx h^*(n)$ ($h^*:N \rightarrow \mathbb{R}$ többnyire nemismert, 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*: $h(n) \geq 0 \quad \forall n \in N$
- *Megengedhető* (admissible): $h(n) \leq h^*(n) \quad \forall n \in N$
- *Monoton megszorítás*: $h(n) - h(m) \leq c(n, m) \quad \forall (n, m) \in A$
(következetes)

□ Megjegyzés:

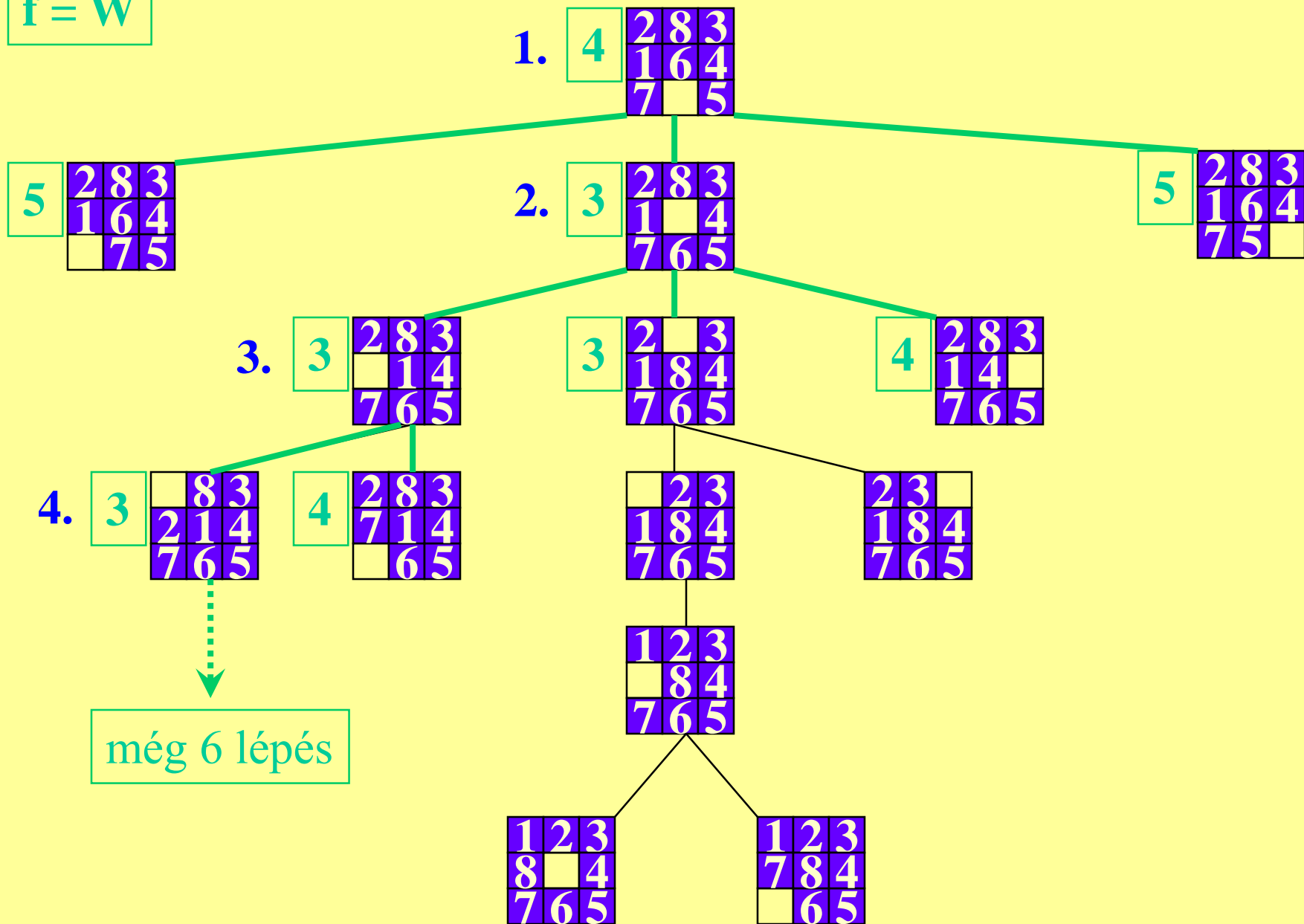
- 8-kirakó : W és P mindhárom tulajdonsággal bír.
- h monoton + h célban nulla $\Rightarrow h$ megengedhető
- Zéró függvény mindhárom tulajdonsággal bír.

Nevezetes heurisztikus algoritmusok

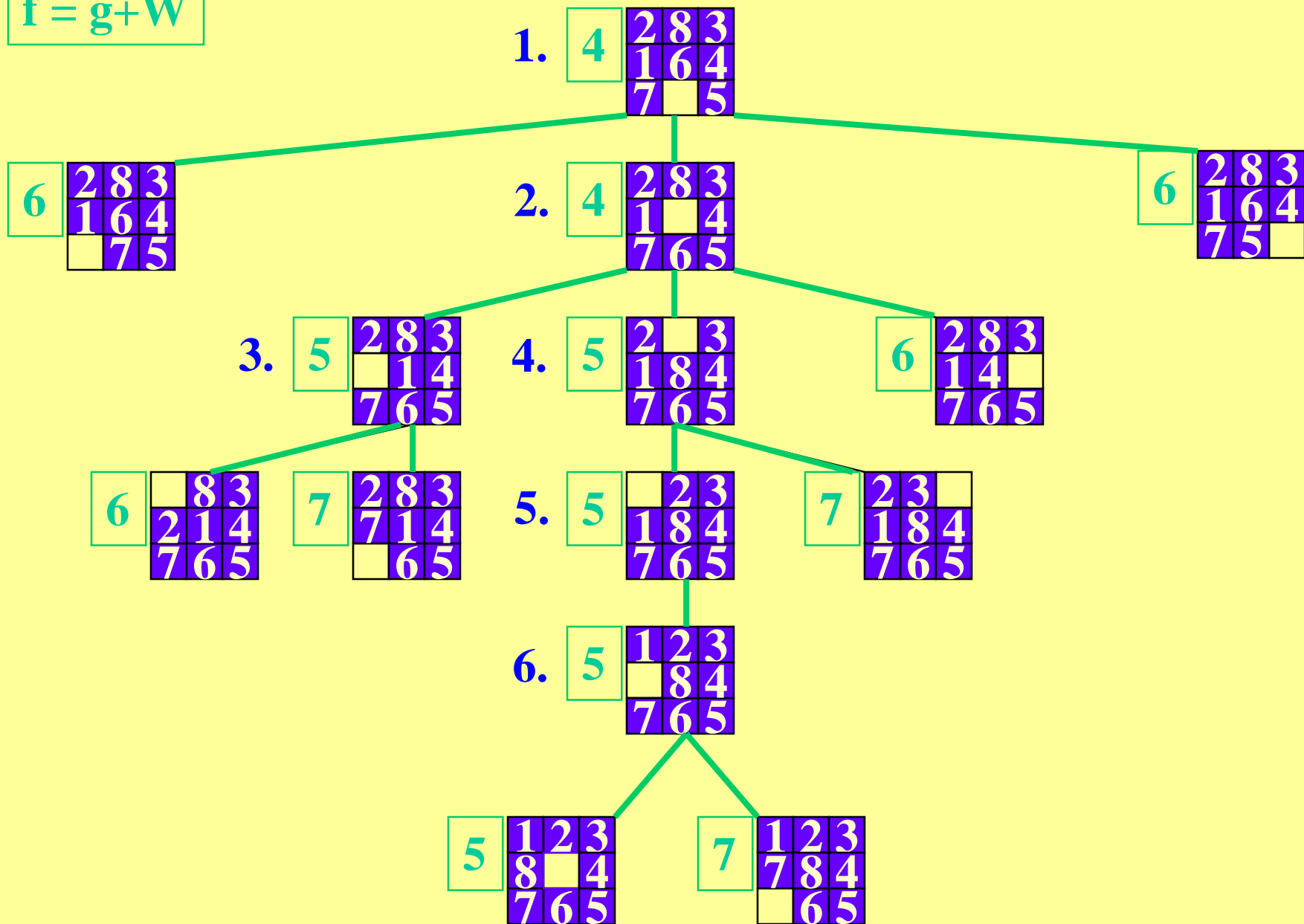
Algoritmus	Definíció	Eredmények
<i>Előre tekintő gráfkeresés</i>	$f = h$	nincs említendő extra tulajdonsága
<i>A algoritmus</i>	$f = g+h$ és $h \geq 0$	<ul style="list-style-type: none"> • megoldást ad, ha van megoldás (még végtelen δ-gráfban is)
<i>A* algoritmus</i>	$f = g+h$ és $h \geq 0$ és $h \leq h^*$	<ul style="list-style-type: none"> • optimális megoldást ad, ha van (még végtelen δ-gráfban is)
<i>A^c algoritmus</i>	$f = g+h$ és $h \geq 0$ és $h \leq h^*$ és $h(n) - h(m) \leq c(n, m)$	<ul style="list-style-type: none"> • optimális megoldást ad, ha van (még végtelen δ-gráfban is) • egy csúcs kiterjesztésekor ismeri az odavezető legolcsóbb utat (legfeljebb egyszer terjeszt ki)

egyenletes gráfkeresés:
 $f = g + 0$

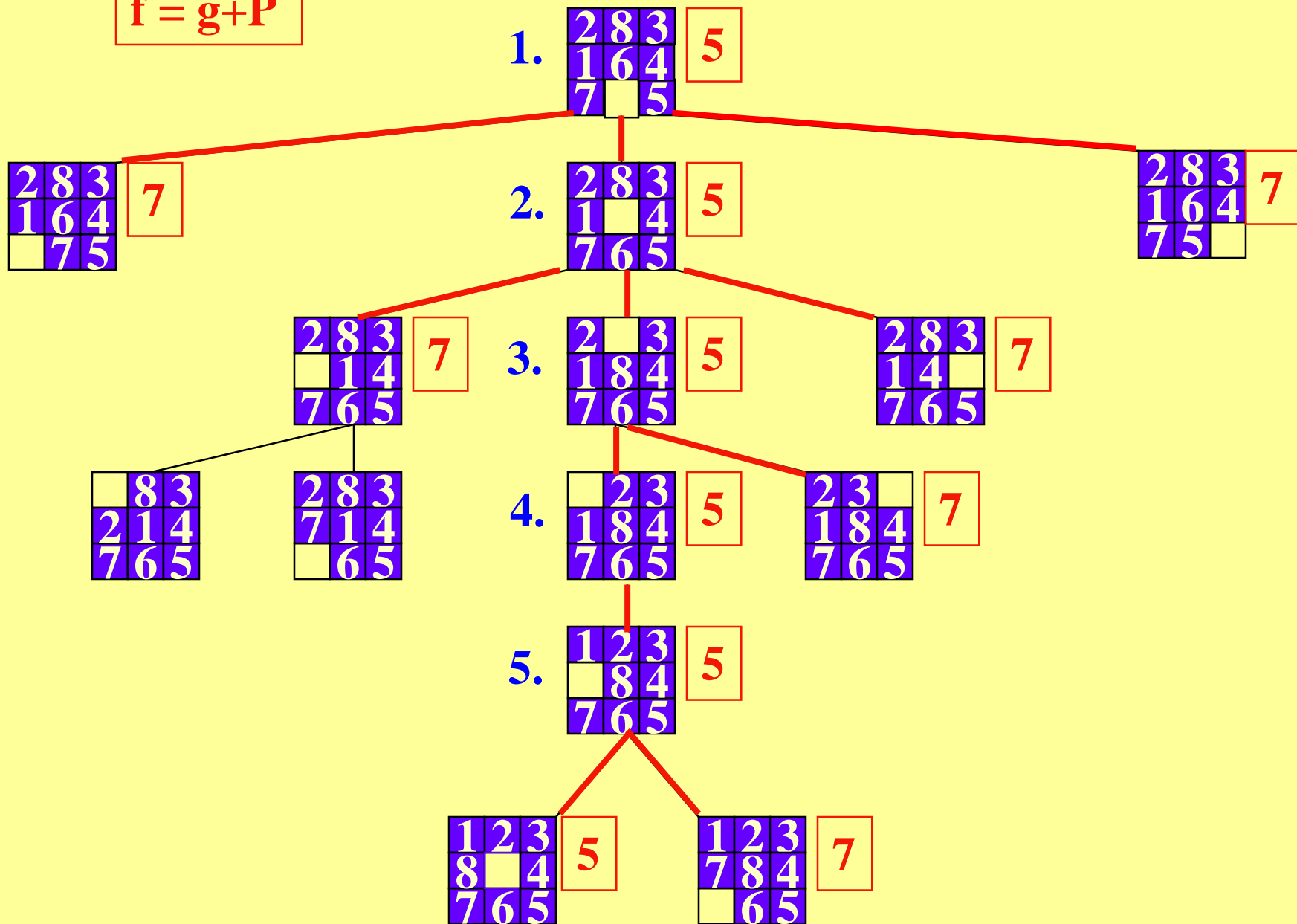
$$f = W$$



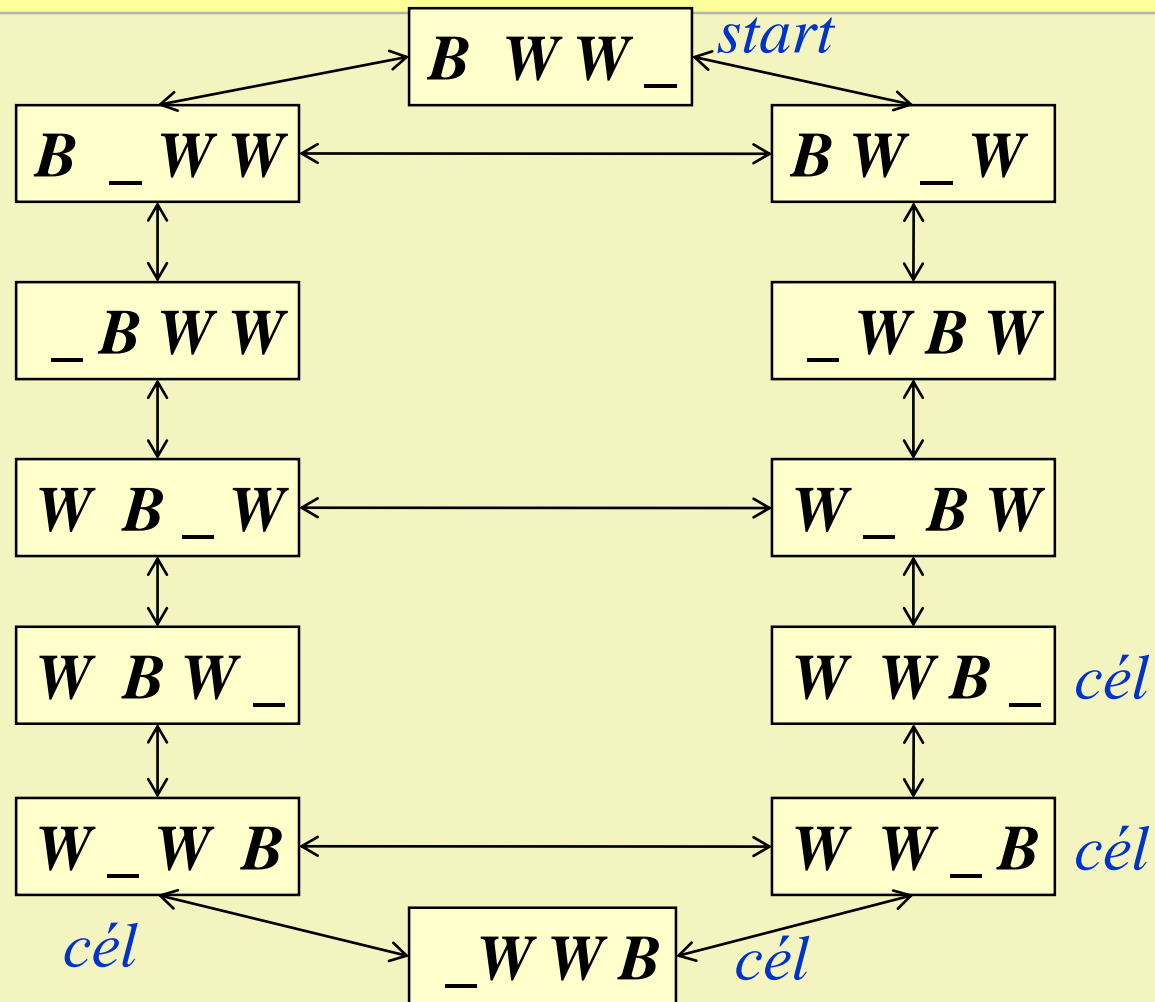
$$f = g + W$$



$$f = g + P$$

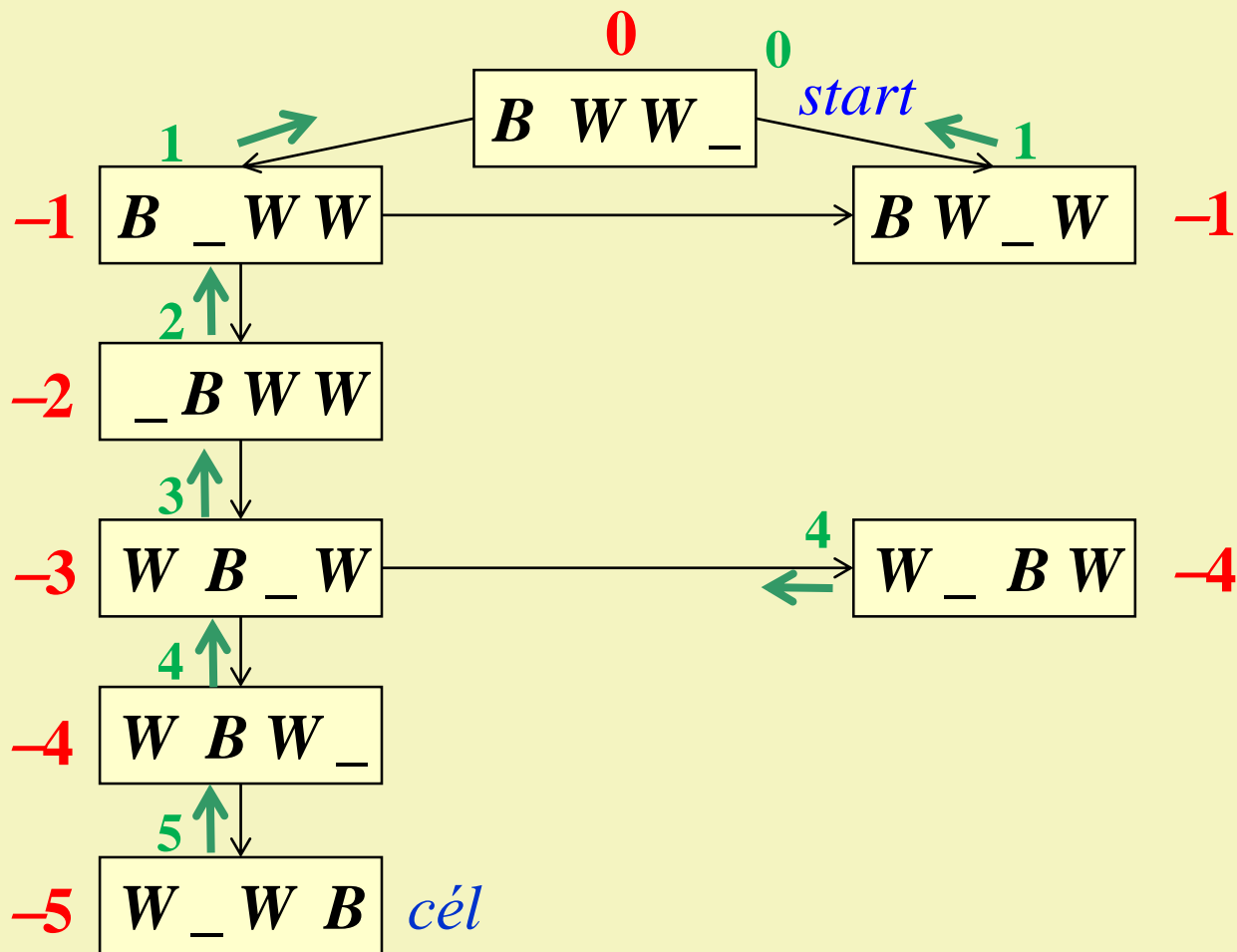


Fekete-fehér kirakó állapot gráfja



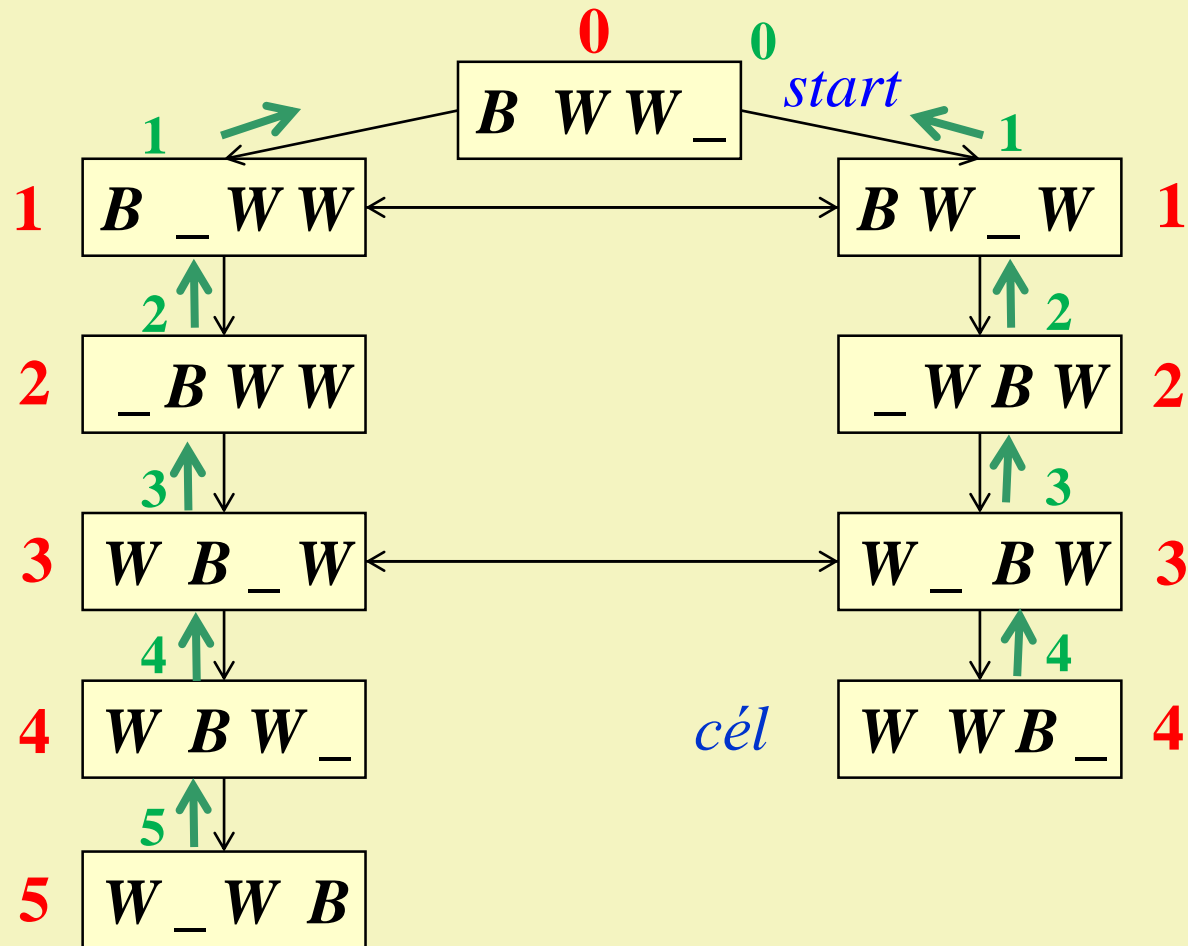
Mélységi gráfkeresés

$$f = -g$$



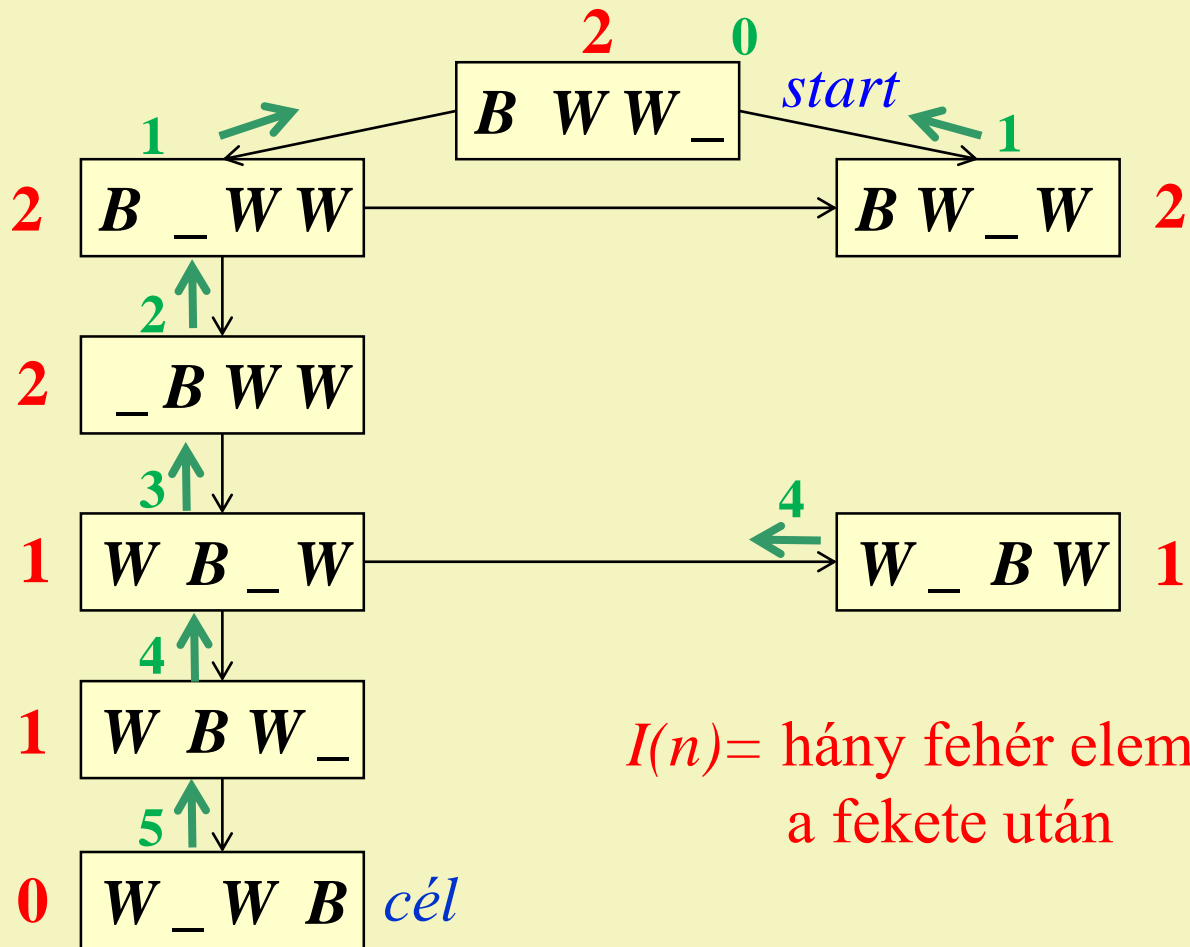
Szélességi gráfkeresés

$$f = g$$



Előre tekintő gráfkeresés

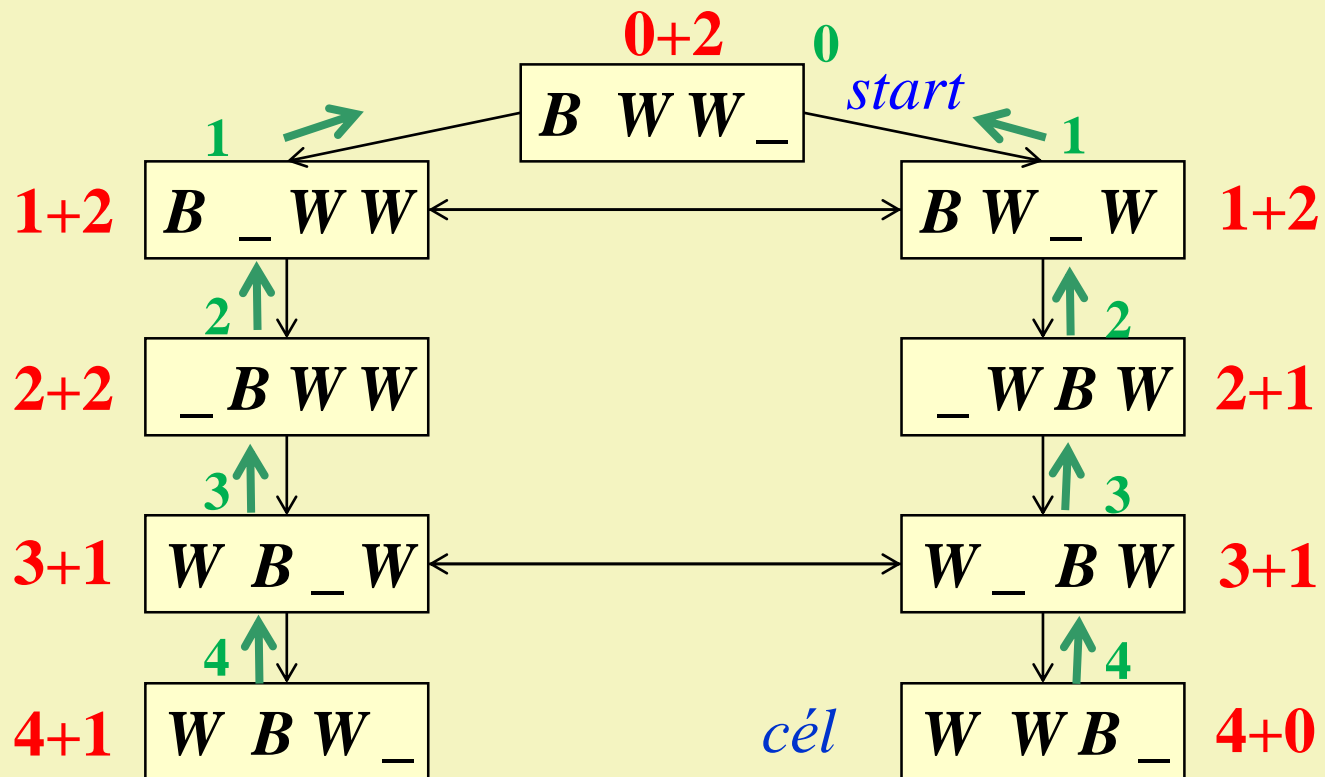
$f = I$



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

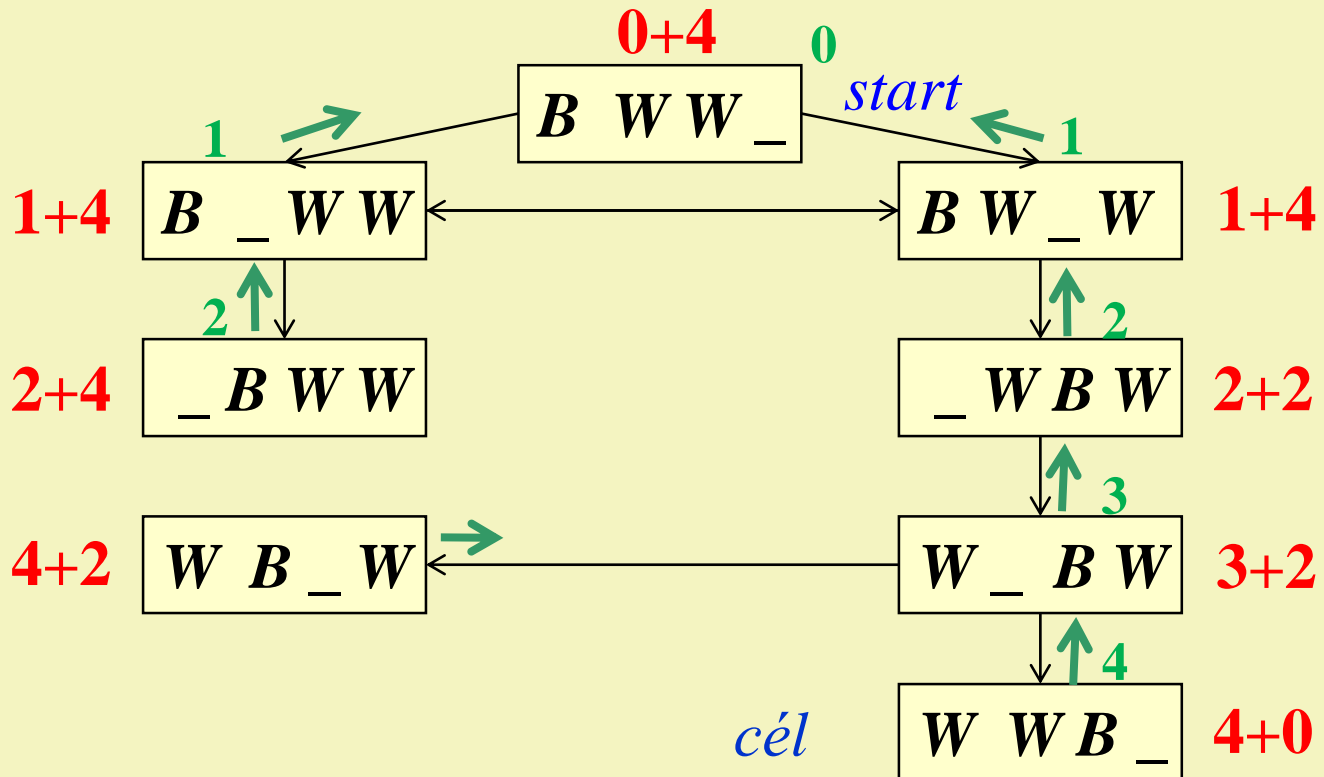
A algoritmus

$$f = g + l$$



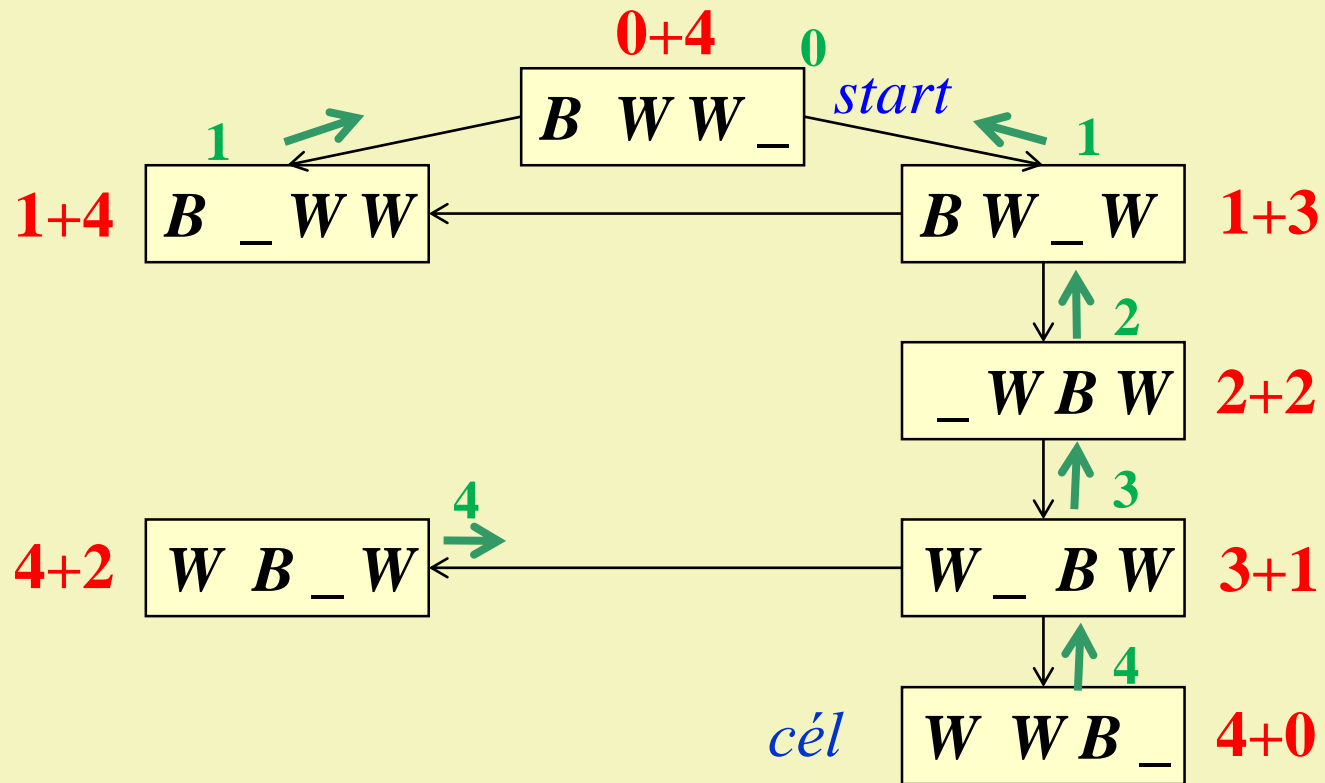
A algoritmus

$$f = g + 2 * I$$



A algoritmus

$$f = g + 2 * I - (1 \text{ ha van } BW_ \text{ vagy } _BW)$$



Elemzés

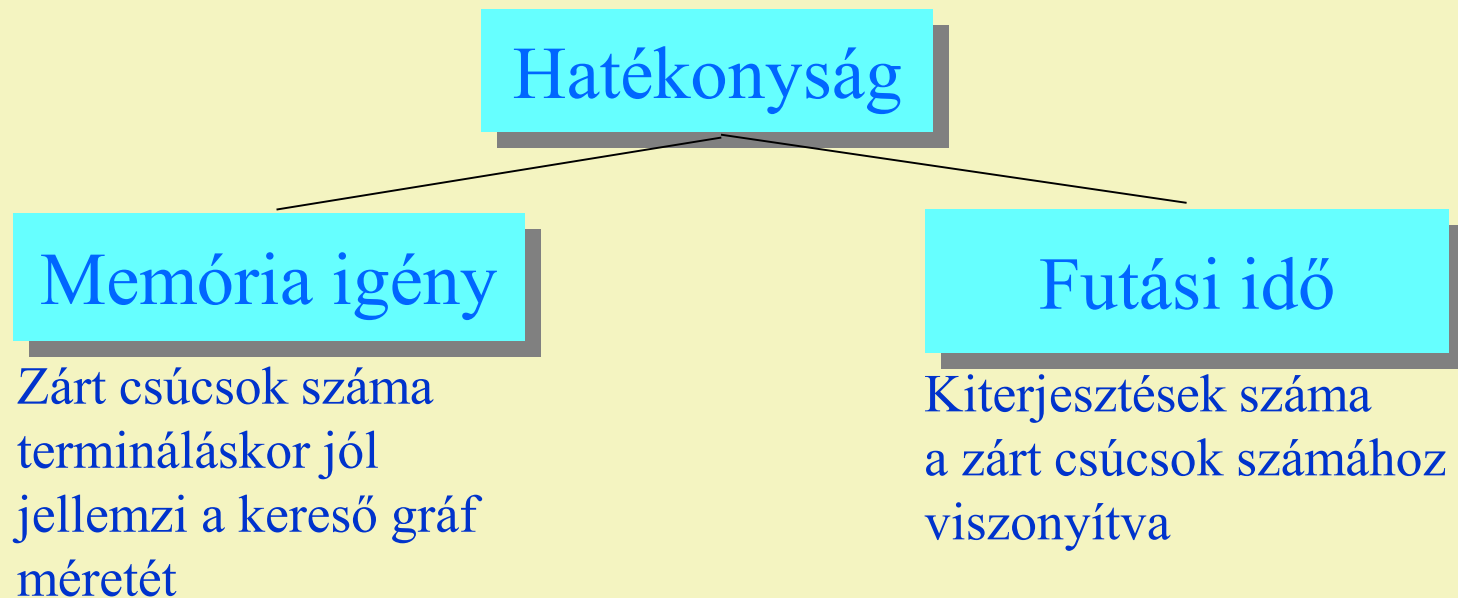
A^c alg

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

3.3. A^* algoritmus hatékonysága



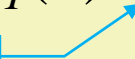
A hatékonyságot a **megengedhető feladatokon** vizsgáljuk, amelyeknek van megoldása és ismert egy megengedhető heurisztikája, tehát az A^* algoritmus optimális megoldást talál hozzájuk.

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 -nél, ha $CLOSED_X \subseteq CLOSED_Y$
 - b. X jobb Y -nél, ha $CLOSED_X \subset CLOSED_Y$
- ❑ Ezek alapján összevethető
 1. két eltérő heurisztikájú A^* algoritmus ugyanazon a feladaton, azaz a két heurisztika.
 2. két útkereső algoritmus, például az A^* algoritmus és egy másik – szintén optimális megoldást garantáló – 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 $h_1(n) < h_2(n)$.

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


- 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 $h_1 \leq h_2$ teljesül, esetleg nem is minden csúcsra.
 - *Példák:*
 - 8-as tologató: $0 \leq W \leq P (\leq F)$
 - Fekete-fehér: $I \leq M (\leq 2 \cdot I)$
- ❑ Minél jobban (közelebbbről) becsli (ha lehet, alulról) a heurisztika a h^* -ot, várhatóan annál kisebb lesz a memória igénye.

15-kirakó

<i>$f =$</i>	<i>$g+0$</i>	<i>$g+W$</i>	<i>$g+P$</i>
6 lépéses megoldás	117	7	6
13 lépéses megoldás	32389	119	13
21 lépéses megoldás	<i>n.a.</i>	3343	145
30 lépéses megoldás	<i>n.a.</i>	<i>n.a.</i>	1137
34 lépéses megoldás	<i>n.a.</i>	<i>n.a.</i>	3971

Különböző gráfkereső algoritmusok memória igényének összehasonlítása

- ❑ Célunk megmutatni azt, hogy az A^* algoritmus memória igénye nem rossz más, hasonló eredményű gráfkereső algoritmusok memória igényéhez képest.
- ❑ **Megengedhetőnek** nevezzük azt az gráfkereső algoritmust, amely megengedhető heurisztikájú útkeresési problémákra optimális megoldást talál, ha van megoldás.
- ❑ Példák:
 - EGK : $f(n) = g(n) + 0$
 - A (A^*) algoritmus : $f(n) = g(n) + h(n)$
 - A^{**} algoritmus: $f(n) = \max_{m \in start \rightarrow n} (g(m) + h(m))$ és a célcsúcs előnyben

Bizonyítható eredmények

- A^* algoritmus lehet rosszabb más megengedhető algoritmusnál egy adott megengedhető feladaton. De
 - Bármelyik megengedhető algoritmus is lehet rosszabb más megengedhető algoritmusnál egy adott megengedhető feladaton.
 - A^* soha nem rosszabb a többi megengedhető algoritmusnál a monoton megszorításos heurisztikájú megengedhető feladatokon.
 - Az A^* -nál nincs jobb megengedhető algoritmus az olyan feladatokon, ahol van olyan optimális megoldási út, amelynek csúcsaira a célcúcs kivételével $h < h^*$ áll fenn).

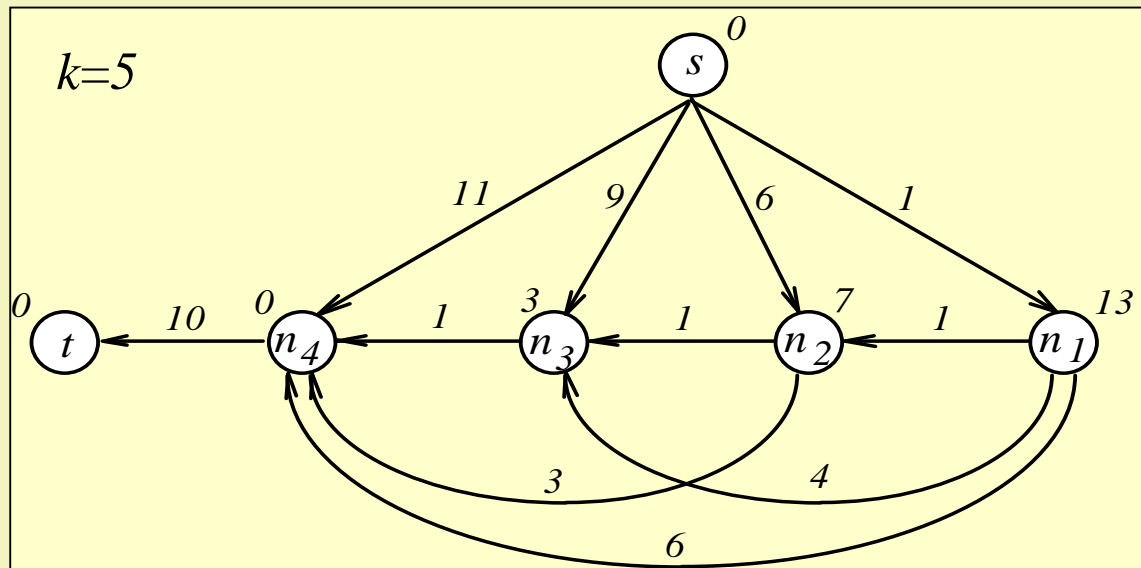
3.3.2. A futási idő elemzése

- ❑ 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 $k_1 \ll k_2$.

Martelli példája



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

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

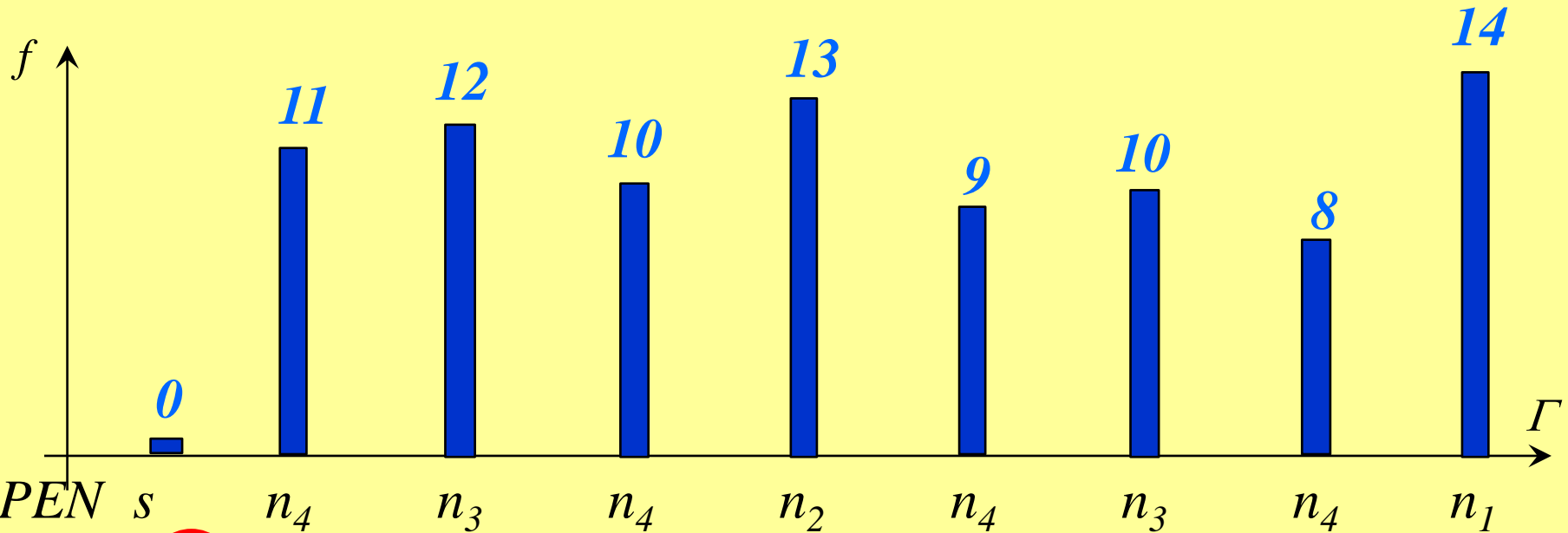
$A = \{(n_i, n_j) \mid 0 \leq i < j < k\} \cup \{(n_{k-1}, t)\}$

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

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

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

Működési grafikon



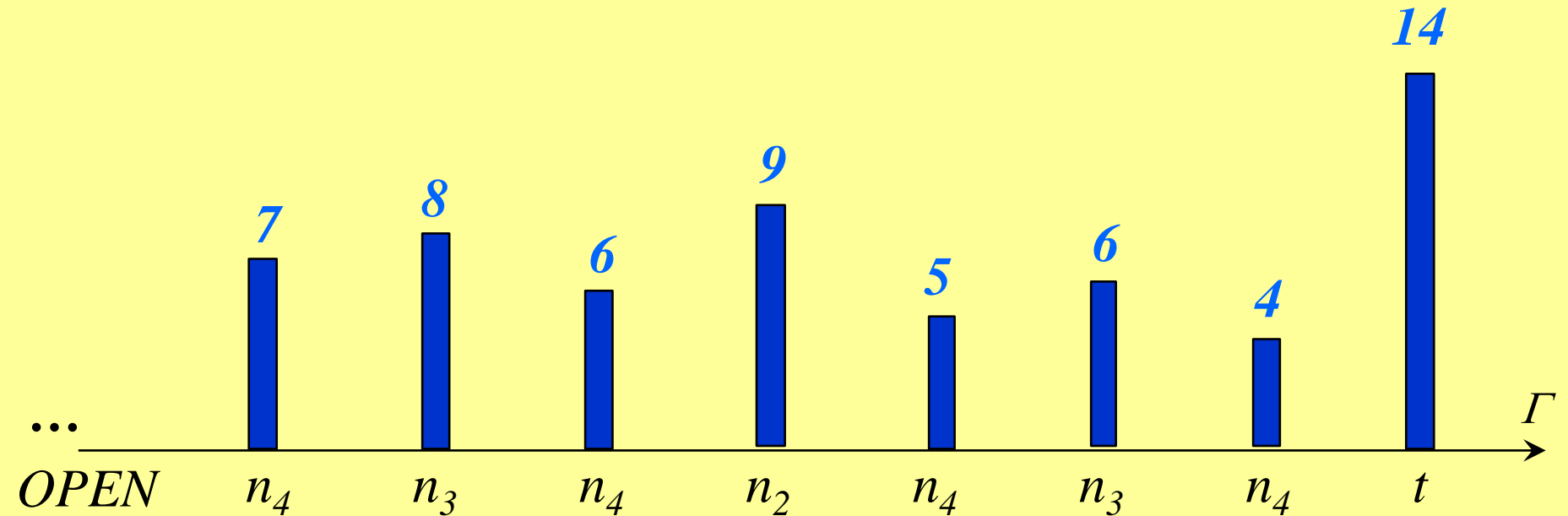
s	$nil, 0, 0$	-	-	-	-	-	-	-	-
n_1	-	$s, 1, 14$	$s, 1, 14$	$s, 1, 14$	$s, 1, 14$	$s, 1, 14$	$s, 1, 14$	$s, 1, 14$	$s, 1, 14$
n_2	-	$s, 6, 13$	$s, 6, 13$	$s, 6, 13$	$s, 6, 13$	-	-	-	-
n_3	-	$s, 9, 12$	$s, 9, 12$	-	-	$n_2, 7, 10$	$n_2, 7, 10$	-	-
n_4	-	$s, 11, 11$	-	$n_3, 10, 10$	-	$n_2, 9, 9$	-	$n_2, 8, 8$	-
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

Gregorics Tibor

Mesterséges intelligencia

Működési grafikon



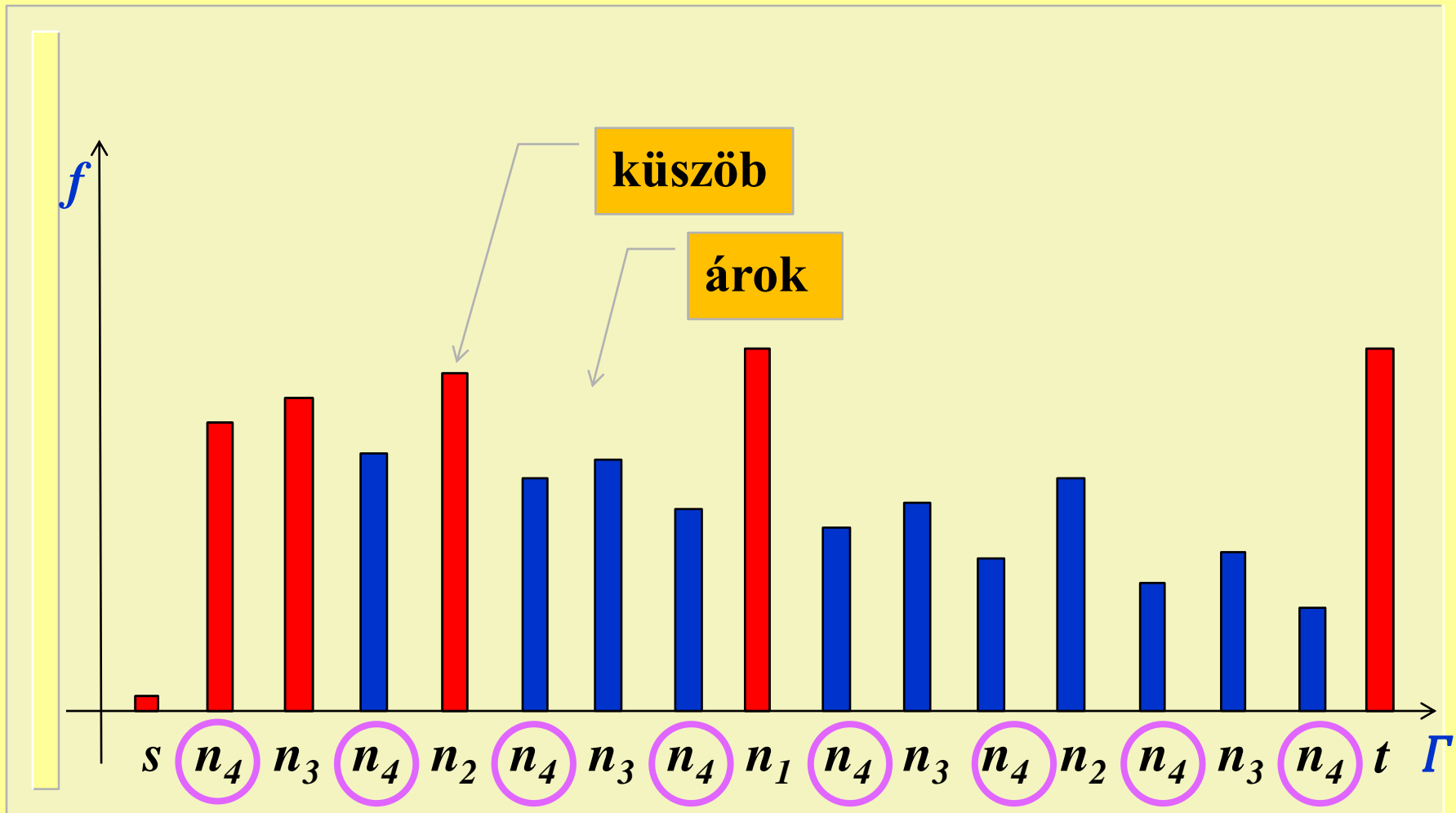
s	-	-	-	-	-	-	-	-
n_1	-	-	-	-	-	-	-	-
n_2	$n_1, 2, 9$	$n_1, 2, 9$	$n_1, 2, 9$	$n_1, 2, 9$	-	-	-	-
n_3	$n_1, 5, 8$	$n_1, 5, 8$	-	-	$n_1, 3, 6$	$n_1, 3, 6$	-	-
n_4	$n_1, 7, 7$	-	$n_3, 6, 6$	-	$n_2, 5, 5$	-	$n_3, 4, 4$	-
t	$n_4, 18, 18$	$n_4, 17, 17$	$n_4, 17, 17$	$n_4, 16, 16$	$n_4, 16, 16$	$n_4, 15, 15$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

π, g, f

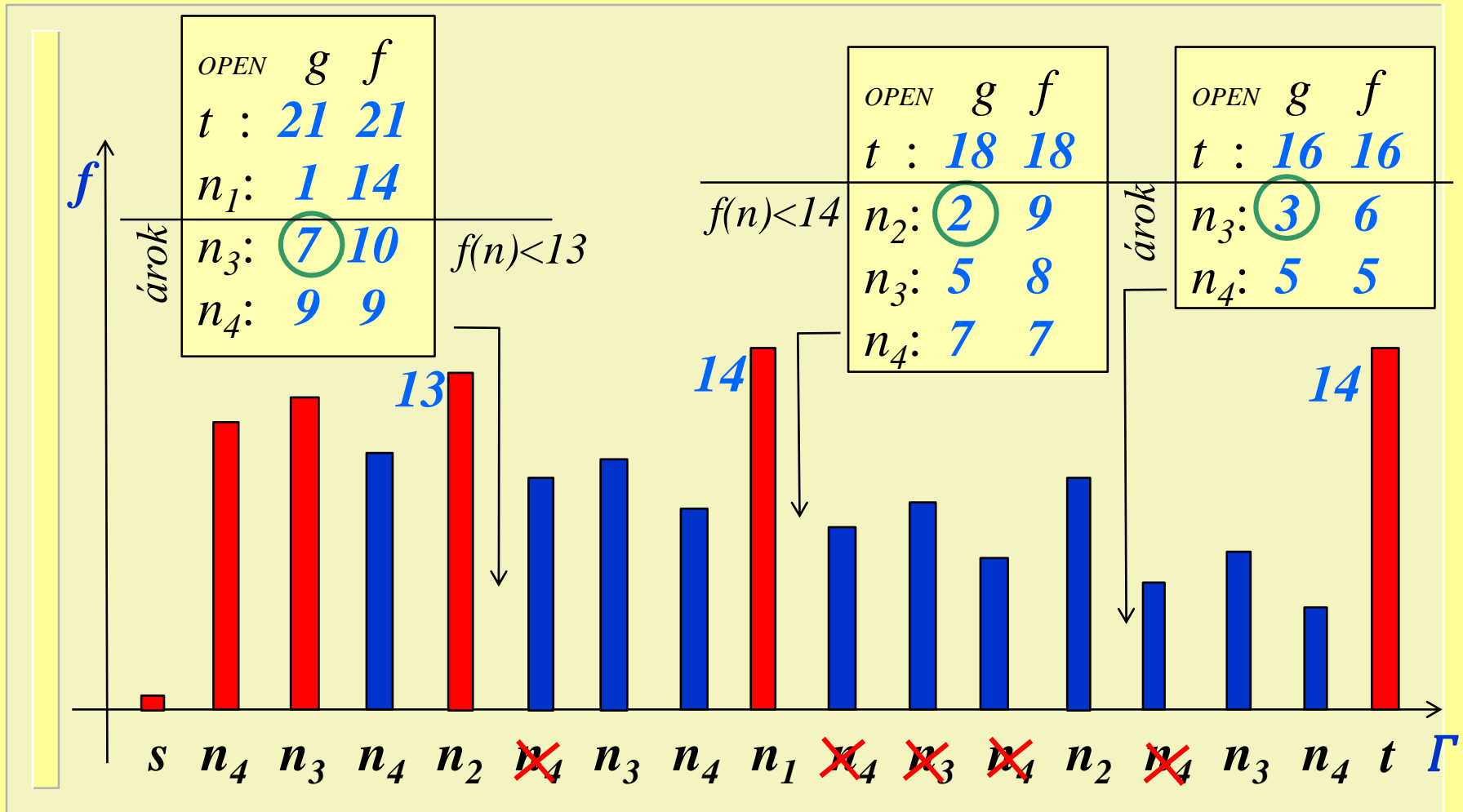
Gregorics Tibor

Mesterséges intelligencia

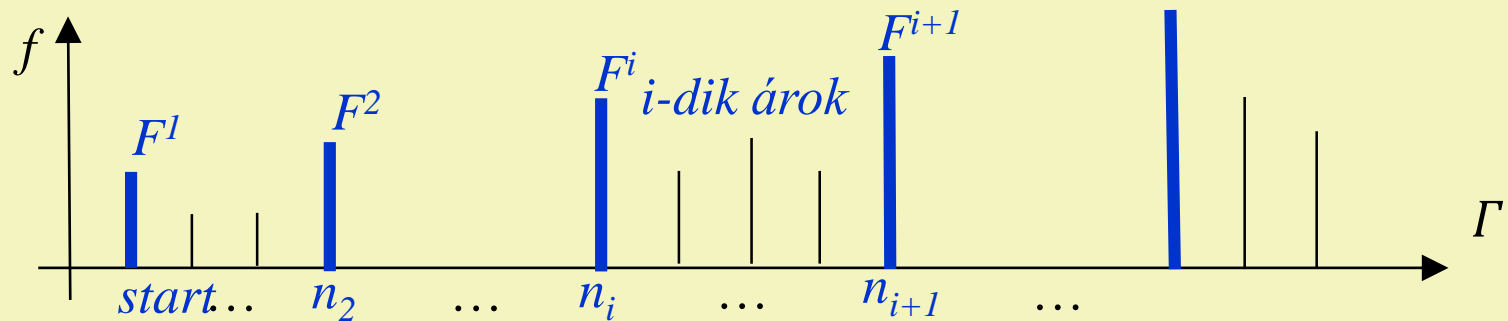
Árkon belüli kiterjesztések száma az A^ algoritmusnál*



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 árkonban 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 $\min_f(OPEN) < F$
 then $n := \min_g(m \in OPEN \mid f(m) < F)$
 else $n := \min_f(OPEN); F := f(n)$
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 monoton megszorításos élek száma nő