

3. Gráfkeresés

- A gráfkeresés olyan KR, amelynek
 - globális munkaterülete: a reprezentációs gráf **startcsúcsból kiinduló már feltárt útjait** tárolja (tehát egy **részgráfot**), és külön az egyes utak végeit, a **nyílt csúcsokat**
 - kiinduló értéke: a startcsúcs,
 - terminálási feltétel: megjelenik egy célcsúcs vagy megakad az algoritmus.
 - keresés szabálya: egyik útvégi csúcs **kiterjesztése**
 - vezérlés stratégiája: a **legkedvezőbb csúcs kiterjesztésére** törekszik

3.1. Általános gráfkereső algoritmus

Jelölések:

- keresőgráf (G) : a reprezentációs gráf eddig bejárt és eltárolt része
- nyílt csúcsok halmaza ($NYÍLT$) : kiterjesztésre várakozó csúcsok, amelyeknek gyerekeit még nem vagy nem eléggé jól ismerjük
- kiterjesztett csúcsok halmaza ($ZÁRT$) : azok a csúcsok, amelyeknek a gyerekeit már előállítottuk
- kiértékelő függvény ($f: NYÍLT \rightarrow \mathbb{R}$) : kiválasztja a megfelelő nyílt csúcsot kiterjesztésre

ADAT := kezdeti érték

while \neg terminálási feltétel(ADAT) **loop**

 SELECT SZ FROM alkalmazható szabályok

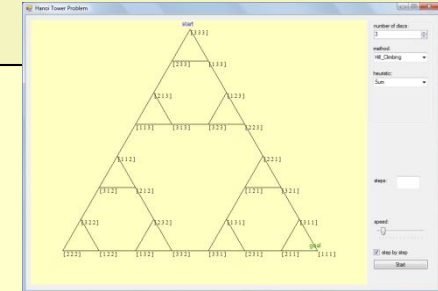
 ADAT := SZ(ADAT)

endloop

Kezdetleges verzió

~~Procedure GK0~~

- ~~1. $G := (\{start\}, \emptyset); NYÍLT := \{start\}$~~
- ~~2. **loop**~~
- ~~3. **if** empty($NYÍLT$) **then return** nincs megoldás~~
- ~~4. $n := \min_f(NYÍLT)$~~
- ~~5. **if** cél(n) **then return** van megoldás~~
- ~~6. $NYÍLT := NYÍLT - \{n\} \cup \Gamma(n) - \pi(n)$~~
- ~~7. $G := G \cup \{(n, m) \in A \mid m \in \Gamma(n) - \pi(n)\}$~~
- ~~8. **endloop**~~
- ~~**end**~~

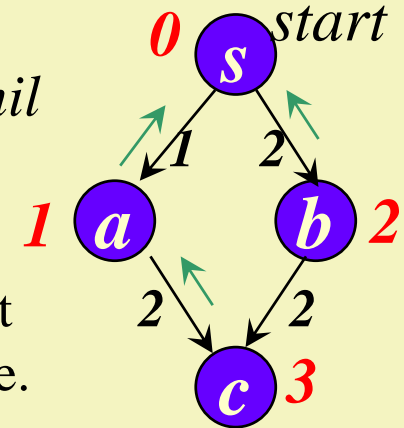


- ❑ Nem olvasható ki a megoldási út a kereső gráfból
 - Meg kell jegyezni a felfedezett utak nyomát.
- ❑ Nem garantál optimális megoldást (sőt még a megoldást sem)
 - Tároljuk el egy csúcsnál az odavezető eddig talált legjobb út költségét.
- ❑ Körökre érzékeny
 - Ha ehhez a csúcshoz egy kört tartalmazó utat találunk, akkor annak költsége drágább lesz a tárolt értéknél, hiszen δ -gráfban vagyunk.

Gráfkeresés függvényei

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

- $\pi(n) = n$ csúcs már ismert szülője, $\pi(start) = nil$
 - π egy $start$ gyökerű irányított feszítőfát jelöl ki
 G -ben: π -feszítőfa, π -út
 - Jó lenne ha a π -út optimális $start \rightarrow n$ G -beli utat jelölne ki: a π feszítőfa optimális lenne.



□ $g: N \rightarrow \mathbb{R}$ *költség függvény*

- $g(n) = c^\alpha(start, n)$ – egy már megtalált $\alpha \in \{start \rightarrow n\}$ út költsége
 - Jó lenne ha minden n -re a $g(n)$ a π -út költségét mutatná, azaz π és g konzisztens lenne.

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

A korrektség fenntartása

- 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

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

$$NYÍLT := NYÍLT \cup \{m\}$$

- 2. Ha m régi csúcs, amelyhez olcsóbb utat találtunk

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

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

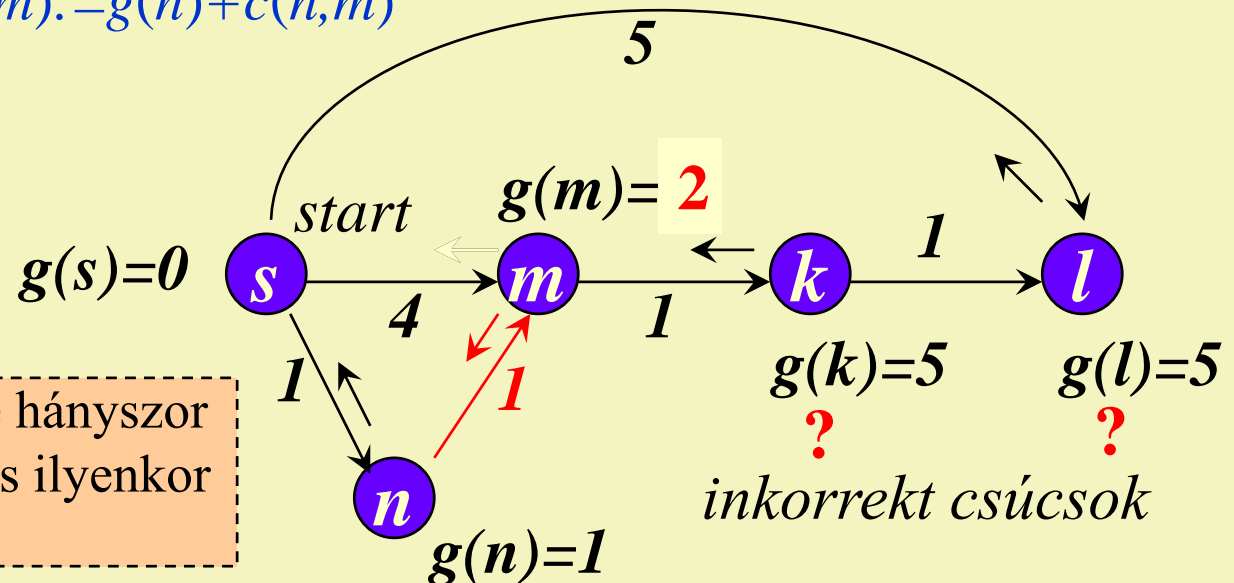
$g(n)$ értéke
ekkor csökken

- 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 tud majd egy csúcs ilyenkor kiterjesztődni?

□ 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 függvénnel, hogy ilyen történjen!
3. Semmi mást ne tegyünk, csak legyen az m csúcs újra nyílt!

ADAT := kezdeti érték

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

Általános gráfkereső algoritmus

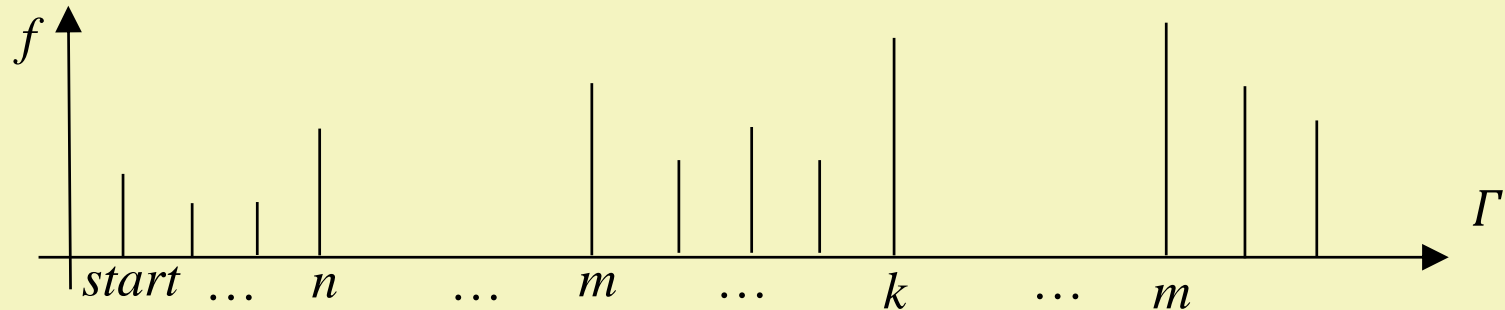
1. $G := (\{start\}, \emptyset); NYÍLT := \{start\}; g(start) := 0; \pi(start) := nil$
2. **loop**
3. **if** *empty*(NYÍLT) **then return** *nincs megoldás*
4. $n := \min_f(NYÍLT)$
5. **if** *cél*(n) **then return** *megoldás*
6. $NYÍLT := NYÍLT - \{n\}$
7. **for** $\forall m \in \Gamma(n) - \pi(n)$ **loop**
8. **if** $m \notin G$ or $g(n) + c(n, m) < g(m)$ **then**
9. $\pi(m) := n; g(m) := g(n) + c(n, m); NYÍLT := NYÍLT \cup \{m\}$
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.

Gráfkeresés működési grafikonja

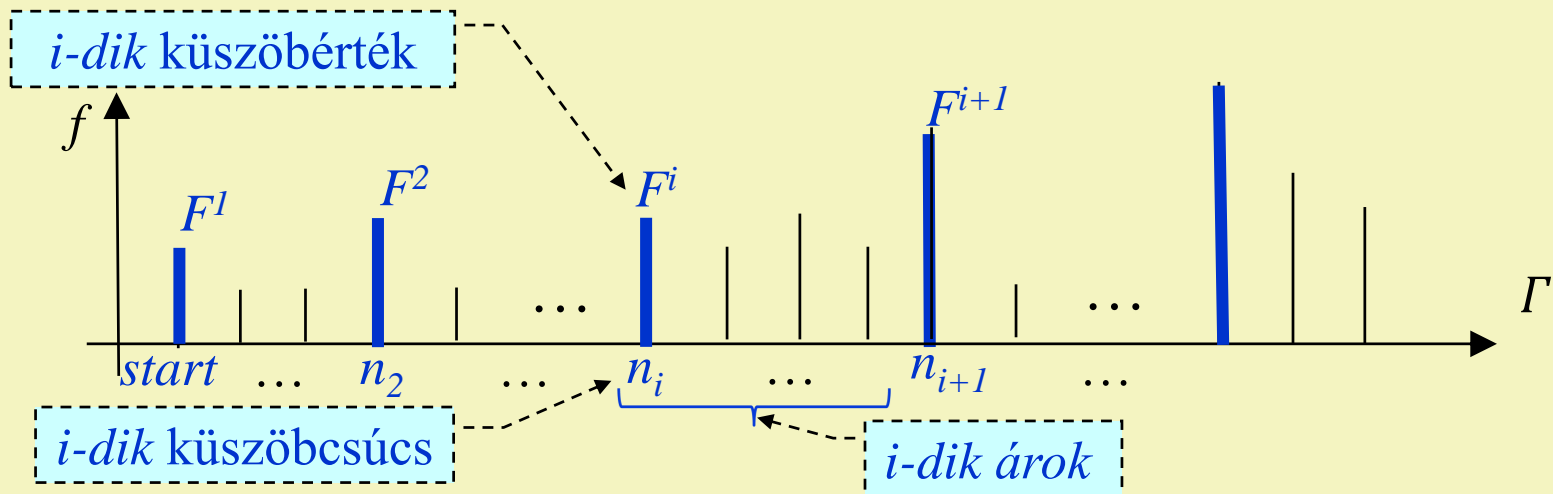


- Soroljuk fel a kiterjesztett csúcsokat kiterjesztésük sorrendjében (ugyanaz a csúcs többször is szerepelhet, hiszen többször is kiterjesztődhet) a kiterjesztésükkor mért f kiértékelő függvényértékükkel.

Csökkenő kiértékelő függvény

- ❑ Egy *GK* kiértékelő függvénye **csökkenő**, amennyiben a egy csúcs kiértékelő függvény értéke az algoritmus működése során nem növekszik, viszont mindig csökken, valahányszor a korábbinál olcsóbb utat találunk hozzá.
- ❑ Csökkenő kiértékelő függvény mellett a ***GK* 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
- A^{**}, B

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

Nevezetes nem-informált algoritmusok

Kapcsolat a visszalépéses kereséssel

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 adja, ha van (még végtelen δ -gráfokban is) egy csúcsot legfeljebb egyszer terjeszt ki
Egyenletes gráfkeresés (EGK)	$f = g$	optimális (legolcsóbb) megoldást adja, ha van (még végtelen δ -gráfokban is) egy csúcsot legfeljebb egyszer terjeszt ki

Kapcsolat Dijkstra algoritmussal

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.
- $h(n) \approx \min_{t \in T} c^*(n, t) = c^*(n, T) = h^*(n)$
 $(h^*: N \rightarrow \mathbb{R})$
- Ez egy az eddiginél szigorúbb definíciója a heurisztikának.
- Példák:
 - 8-kirakó : W, P
 - 0 (zéró függvény)?

hátralevő optimális költség

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)

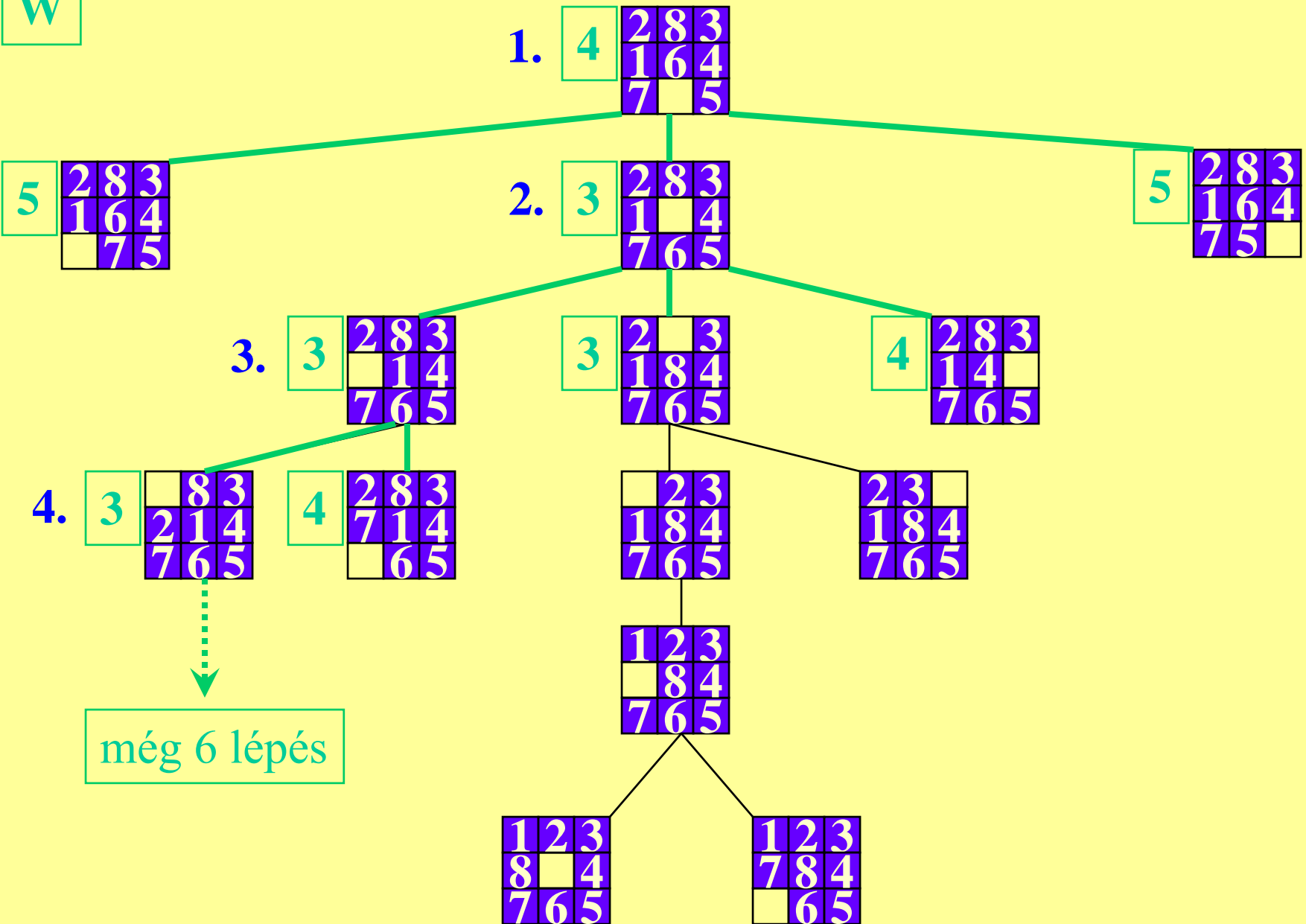
□ Megjegyzés:

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

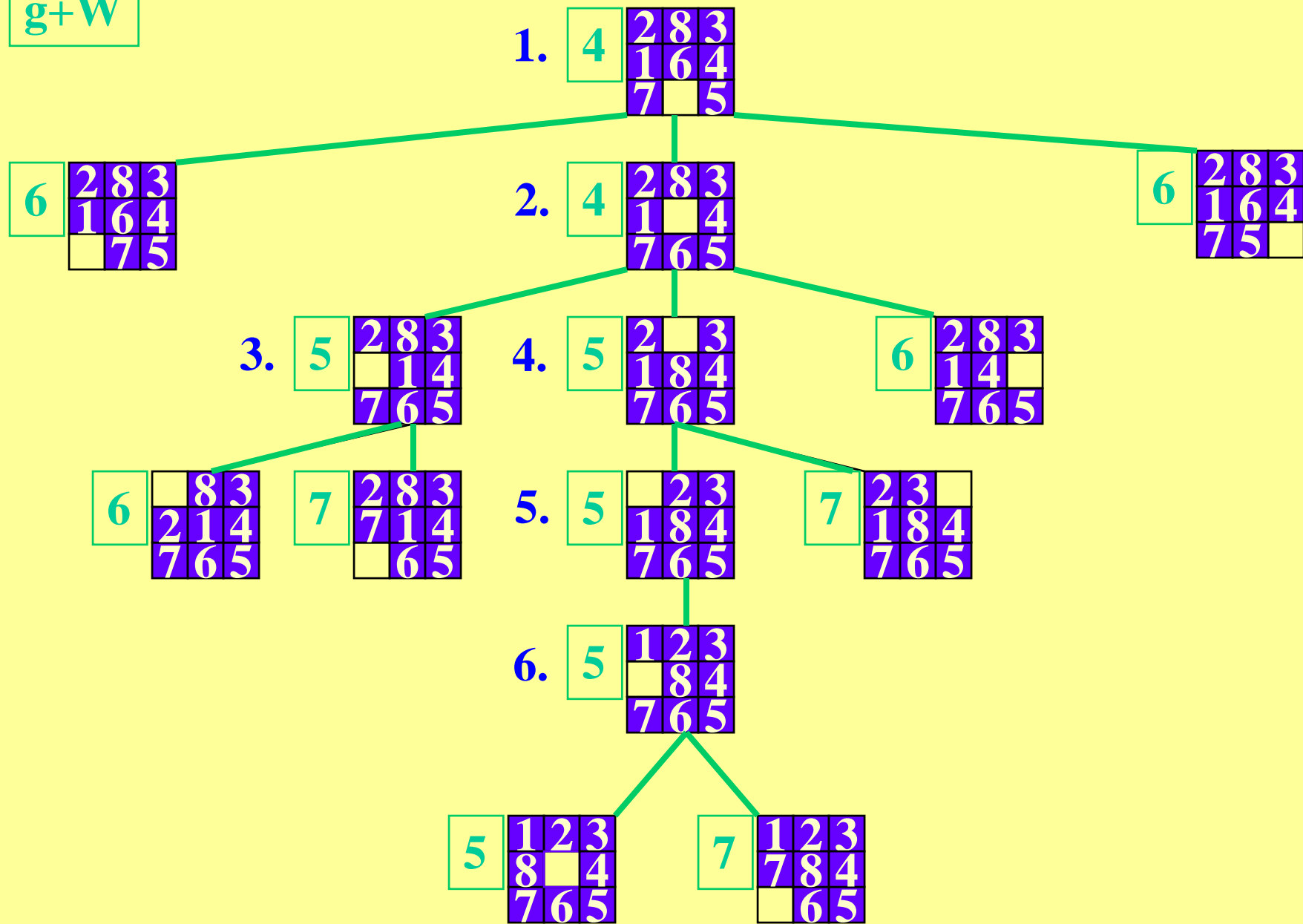
Nevezetes heurisztikus algoritmusok

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 csúcsot legfeljebb egyszer terjeszt ki

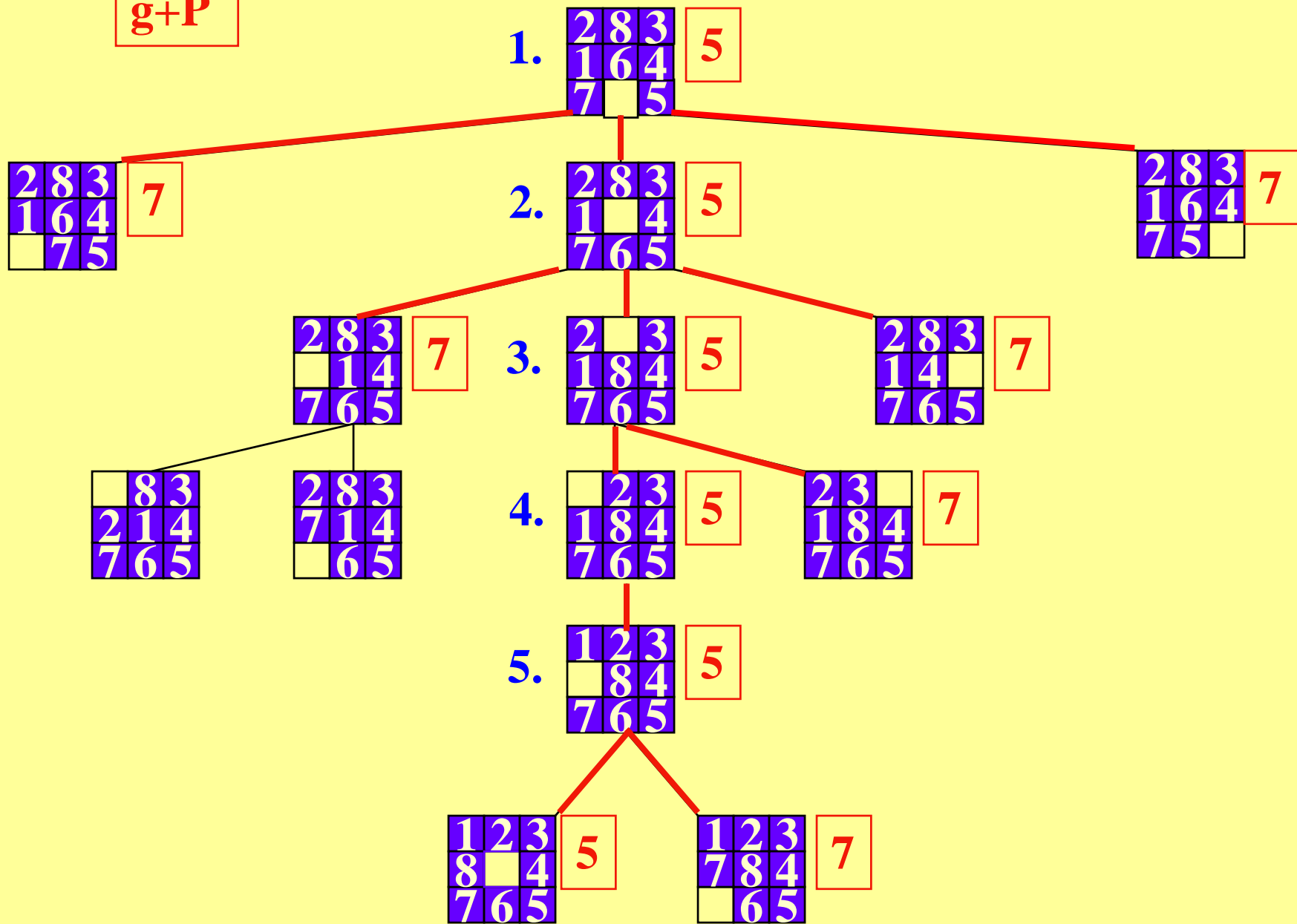
egyenletes gráfkeresés: $f = g + 0$



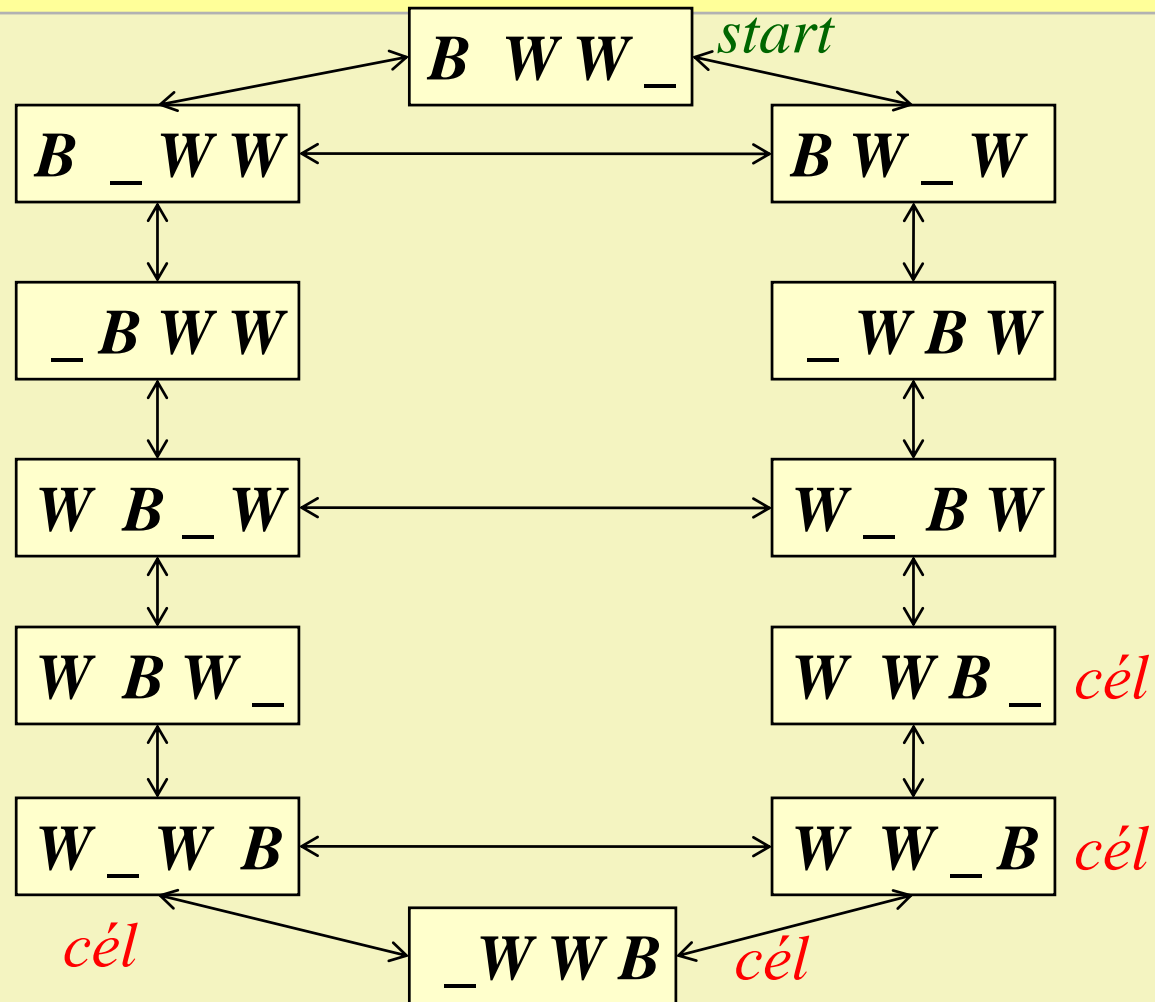
$g+W$



g+P

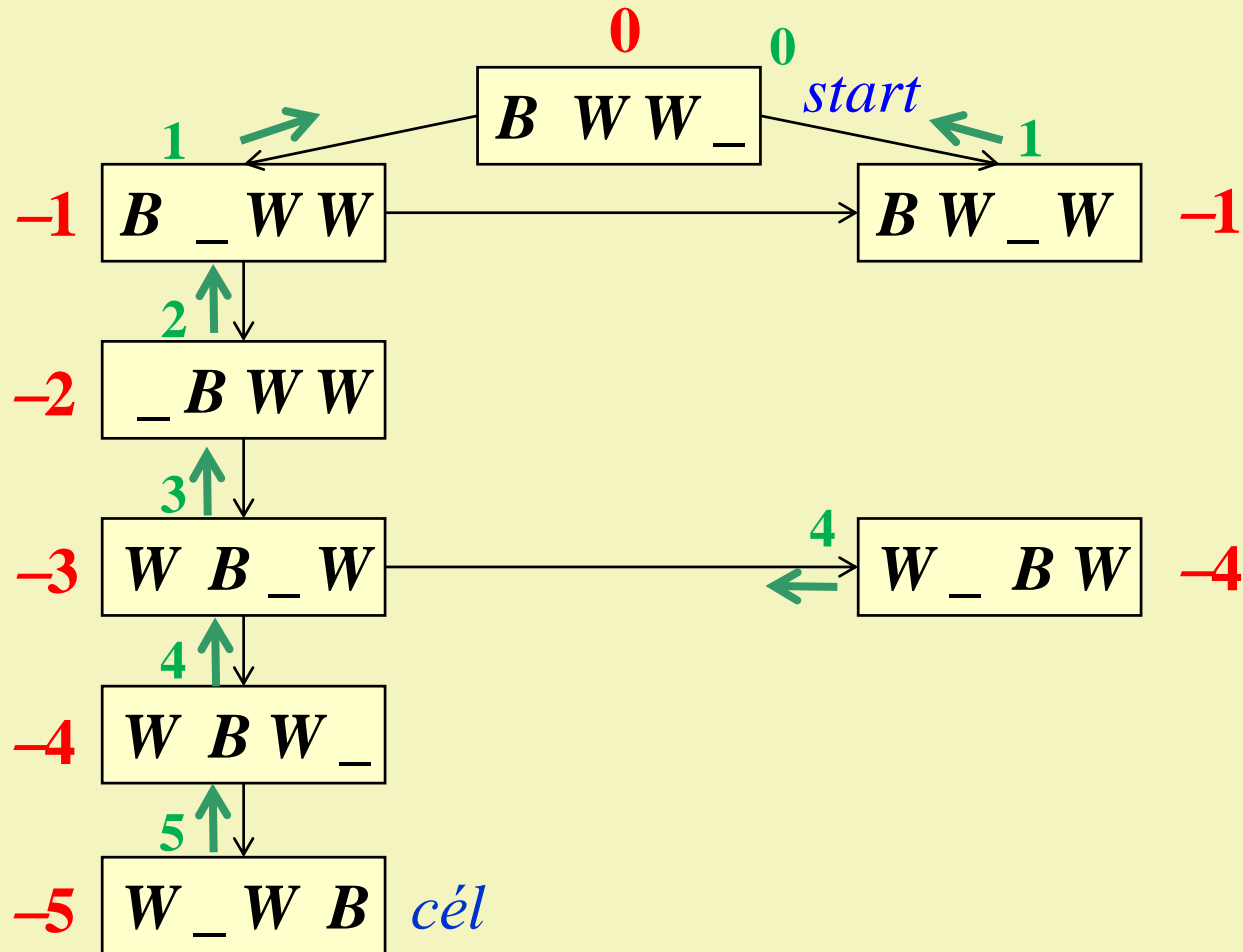


Fekete-fehér kirakó állapot gráfja



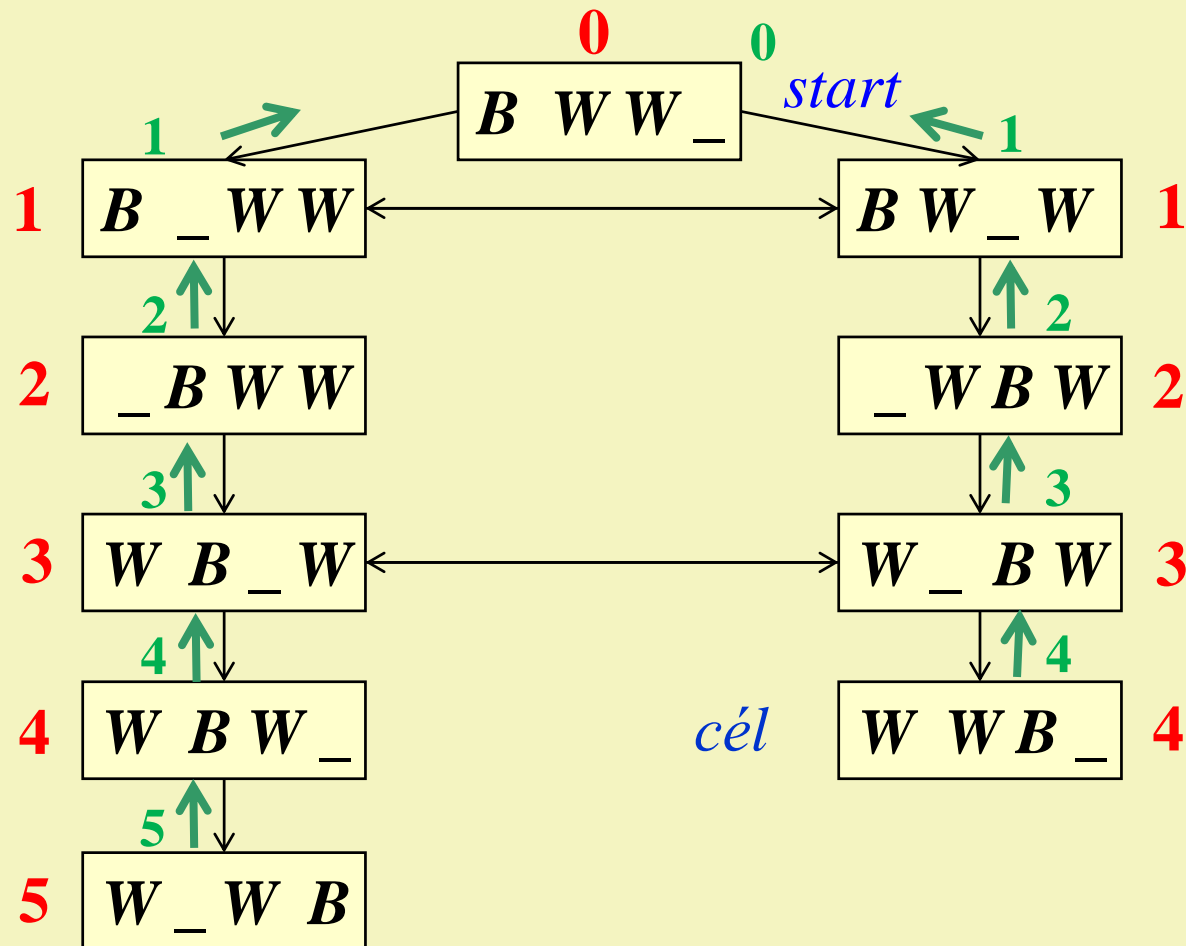
Mélységi gráfkeresés

$$f = -g$$



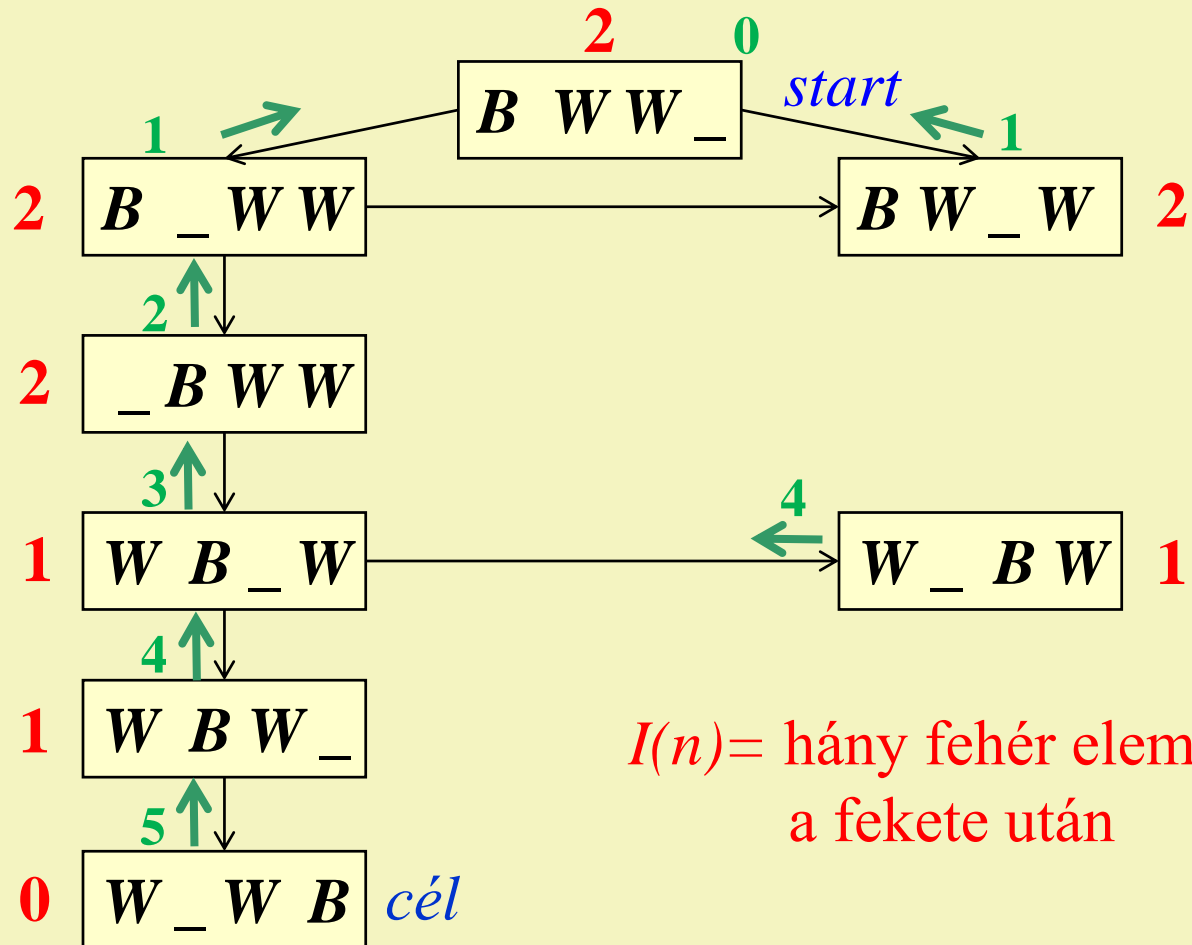
Szélességi gráfkeresés

$$f = g$$



Előre tekintő gráfkeresés

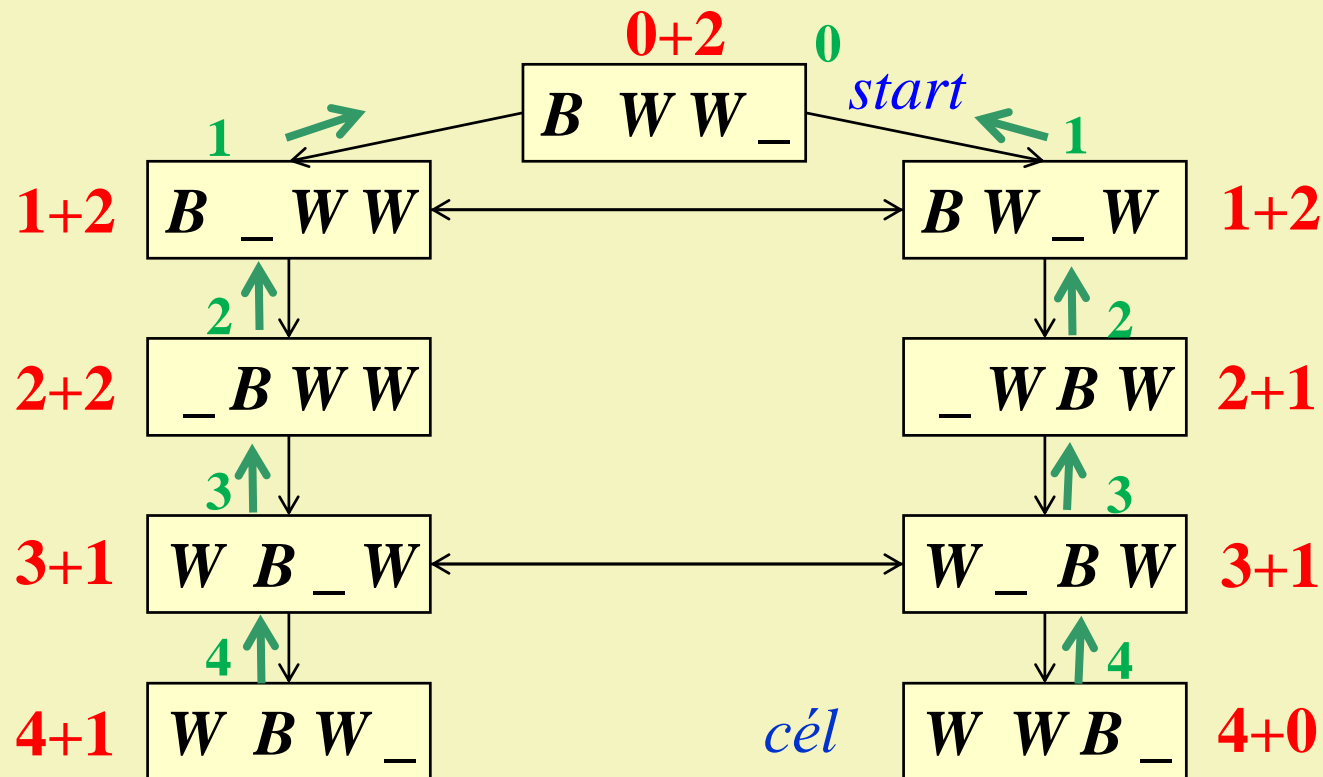
$f = I$



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

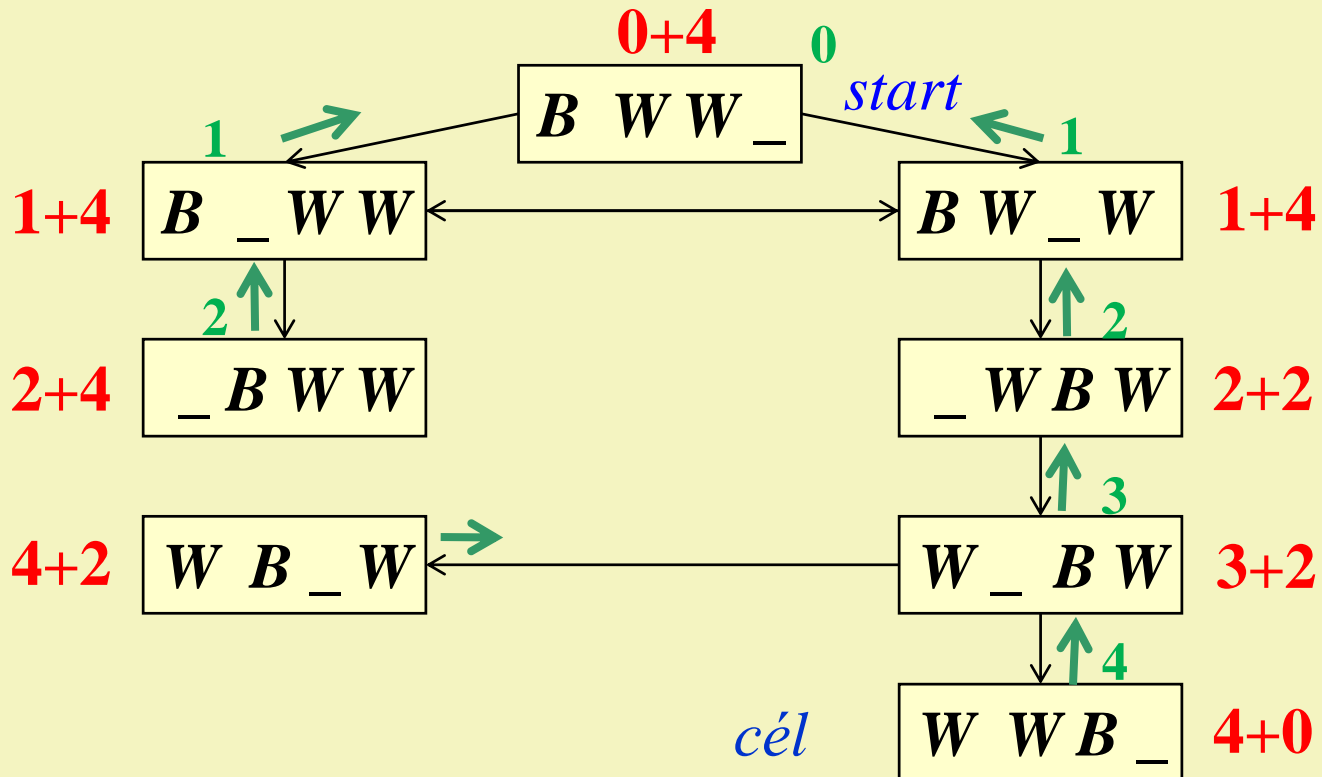
A algoritmus

$$f = g + l$$



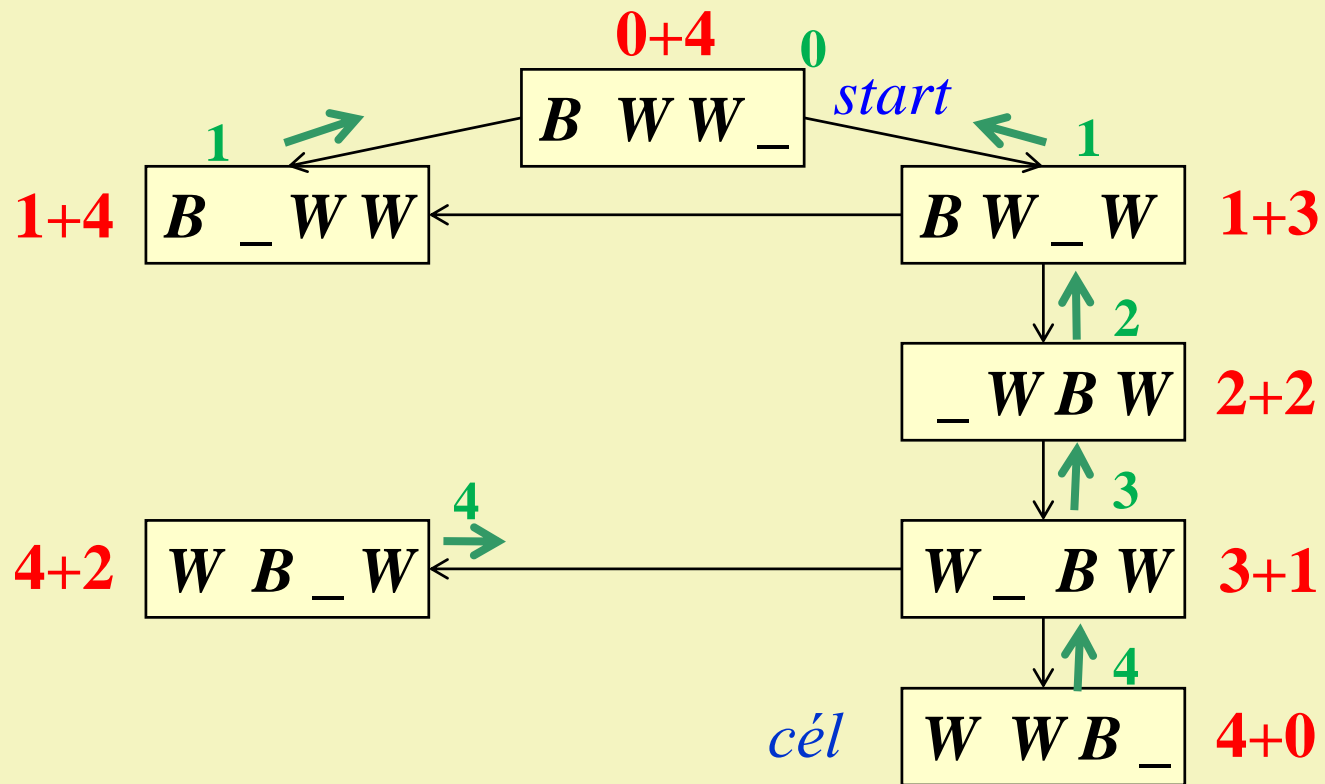
A algoritmus

$$f = g + 2 * I$$



A algoritmus

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



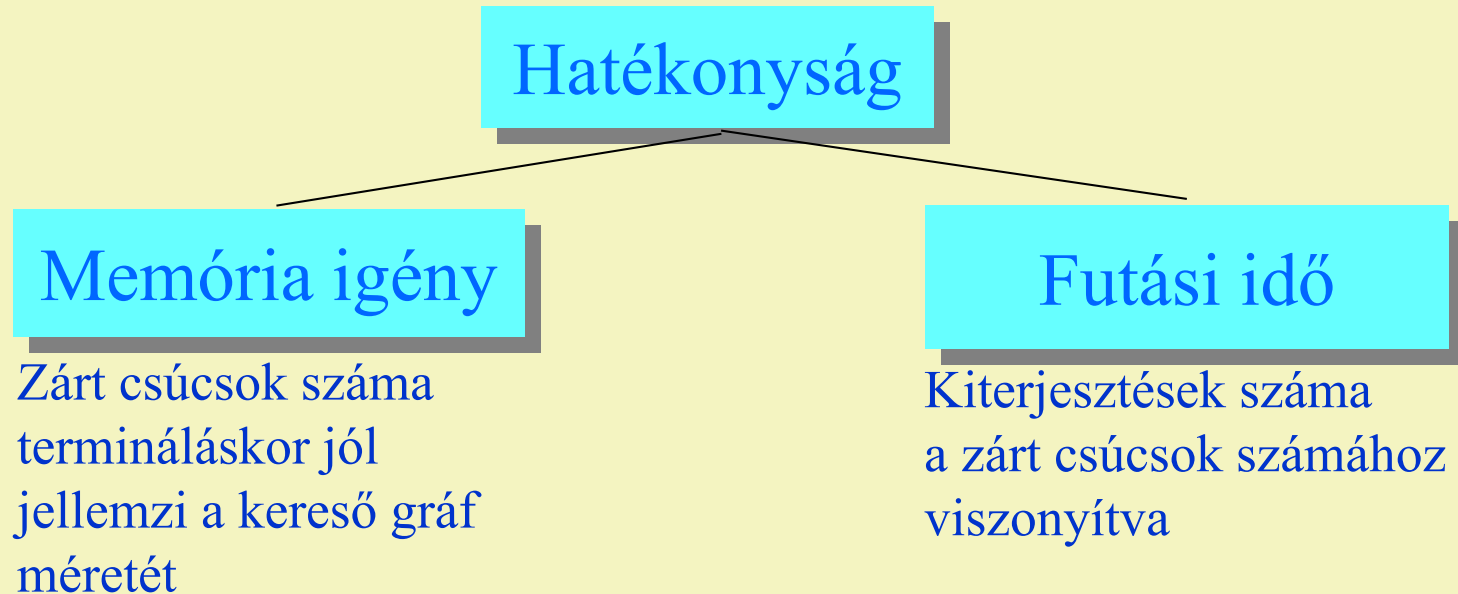
f	Alg	mo	G	Γ
$-g$	MGK	5	8	5
g	SZGK	4	10	8
l	Előre tekintő	5	8	5
$g+l$	A alg	4	9	7
$g+2*l$	A alg	4	8	6
$g+2*l-1(ha...)$	A alg	4	7	5

A^c alg

A^c alg

A^c alg

3.3. A^* algoritmus hatékonysága



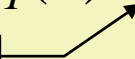
A hatékonyságot a **megengedhető feladatokon** vizsgáljuk, amelyeknek van megoldása és ismert egy megengedhető heurisztikája, tehát az A^* algoritmus optimális megoldást talál hozzájuk.

3.3.1. A memória igény vizsgálata

- $ZÁRT_S$ ~ az S gráfkereső algoritmus által lezárt (kiterjesztett) csúcsok halmaza
- Rögzítsünk egy feladatot és két, X és Y gráfkereső algoritmust
Az adott feladatra nézve
 - a. az X nem rosszabb az Y -nél, ha $ZÁRT_X \subseteq ZÁRT_Y$
 - b. az X jobb az Y -nél, ha $ZÁRT_X \subset ZÁRT_Y$
- 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 $ZÁRT_{A_2} \subseteq ZÁRT_{A_1}$

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 ($ZÁRT_{A_2} \subset ZÁRT_{A_1}$), 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 (közelebbről) 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>
<i>6 lépéses megoldás</i>	<i>117</i>	<i>7</i>	<i>6</i>
<i>13 lépéses megoldás</i>	<i>32389</i>	<i>119</i>	<i>13</i>
<i>21 lépéses megoldás</i>	<i>n.a.</i>	<i>3343</i>	<i>145</i>
<i>30 lépéses megoldás</i>	<i>n.a.</i>	<i>n.a.</i>	<i>1137</i>
<i>34 lépéses megoldás</i>	<i>n.a.</i>	<i>n.a.</i>	<i>3971</i>

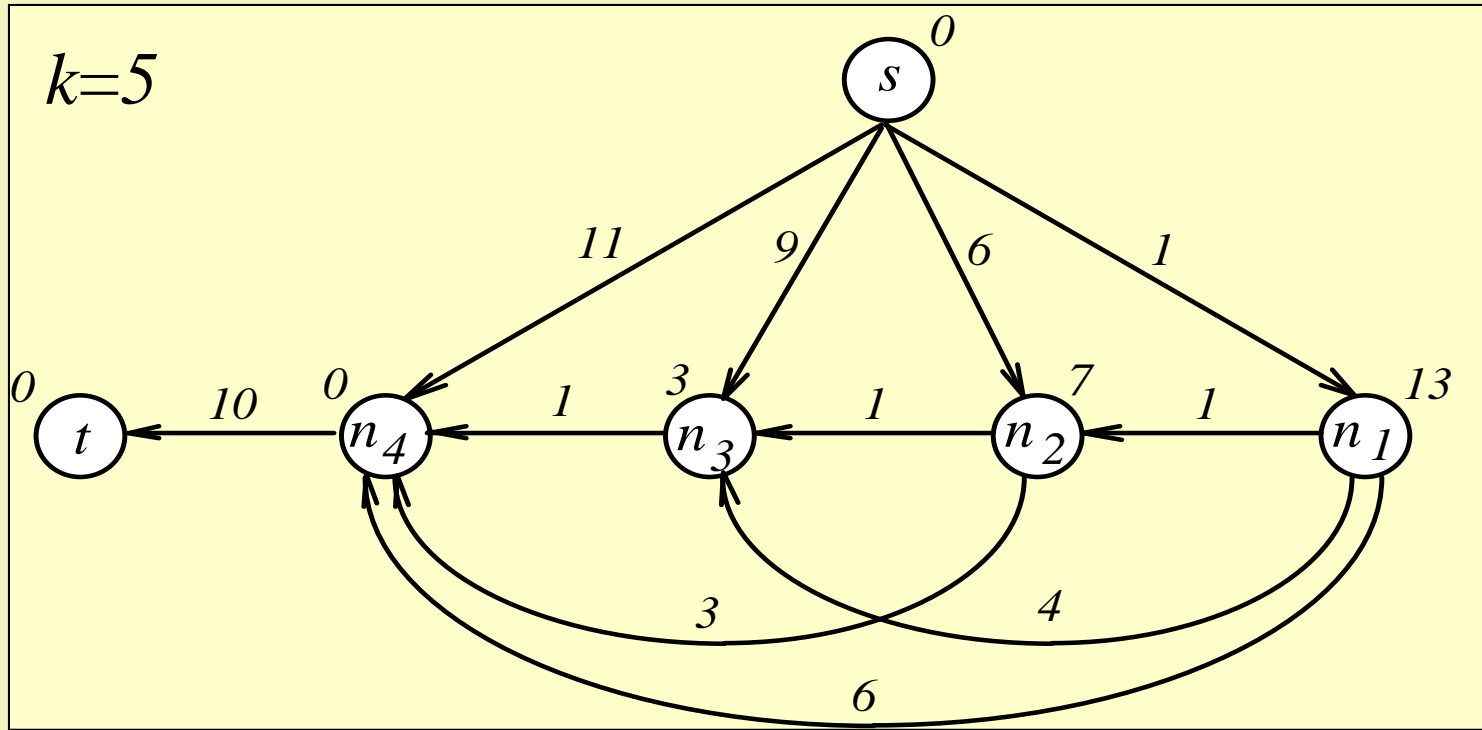
3.3.2. A futási idő elemzése

- ❑ Zárt csúcsok száma: $k = |ZÁRT|$
- ❑ 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

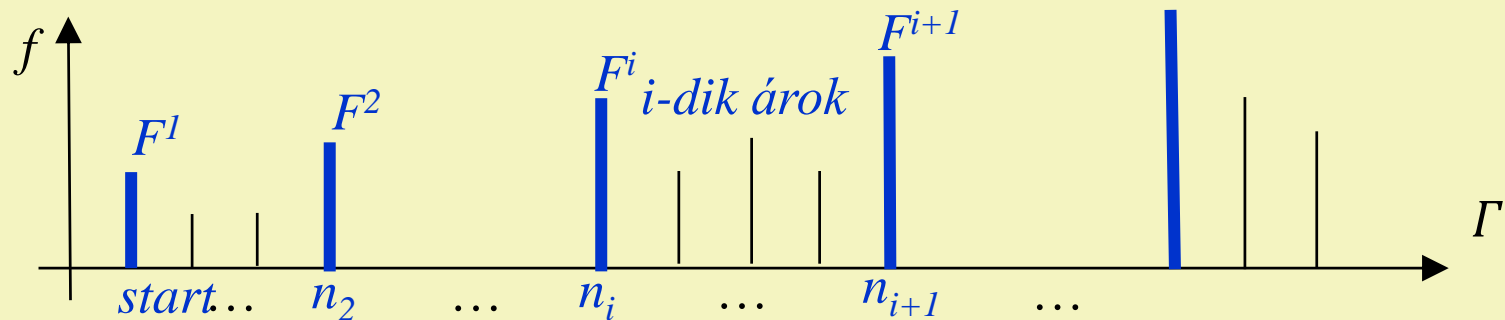
Megjegyzés

- ❑ 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



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 árkokban 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 $\min_f(NYÍLT) < F$
 then $n := \min_g(m \in NYÍLT \mid f(m) < F)$
 else $n := \min_f(NYÍLT); F := f(n)$
 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

Mohó A algoritmus

- ❑ Nincs mindig szükség az optimális megoldásra.
 - Ilyenkor a mohó A^* algoritmus is használható, amely rögtön megáll, ha célsúcs jelenik meg a *NYÍLT*-ban.
- ❑ A mohó A^* algoritmus csak a megoldás megtalálását garantálja. De belátható
 - Ha h megengedhető és $\forall t \in T: \forall (n,t) \in A: h(n) + \alpha \geq c(n,t)$, akkor a talált megoldás költsége: $g(t) \leq h^*(s) + \alpha$
- ❑ A mohó A^* *algoritmus* megengedhető heurisztika mellett akkor garantálja az optimális megoldást is,
 - ha $\forall t \in T: \forall (n,t) \in A: h(n) = c(n,t)$ vagy
 - ha h monoton és $\exists \alpha \geq 0: \forall t \in T: \forall (n,t) \in A: h(n) + \alpha = c(n,t)$