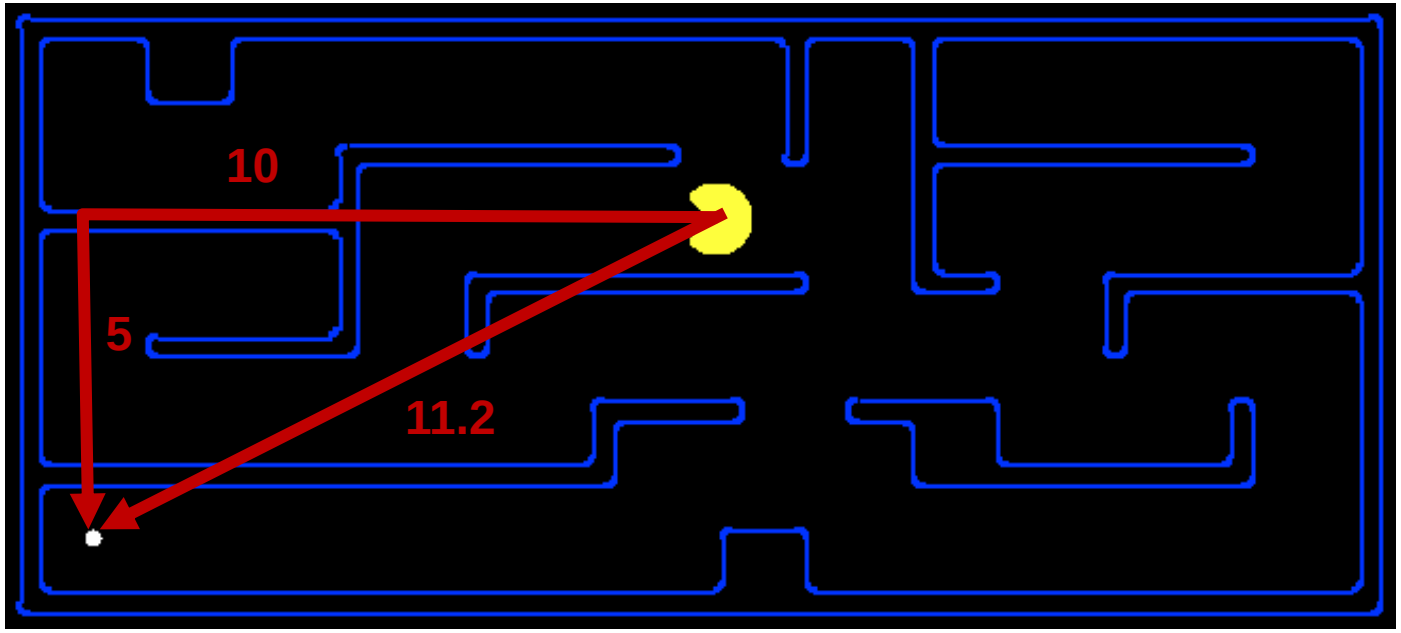
Vođene Pretrage - Informed Search

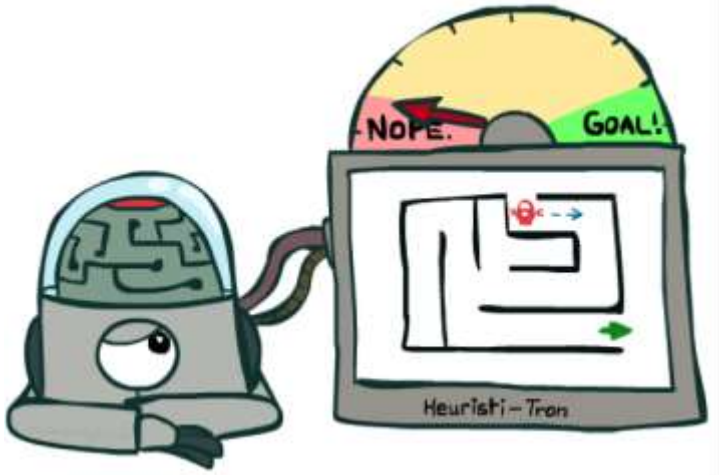
- „Slepe Pretrage“
  - DFS
  - BFS
  - UCS

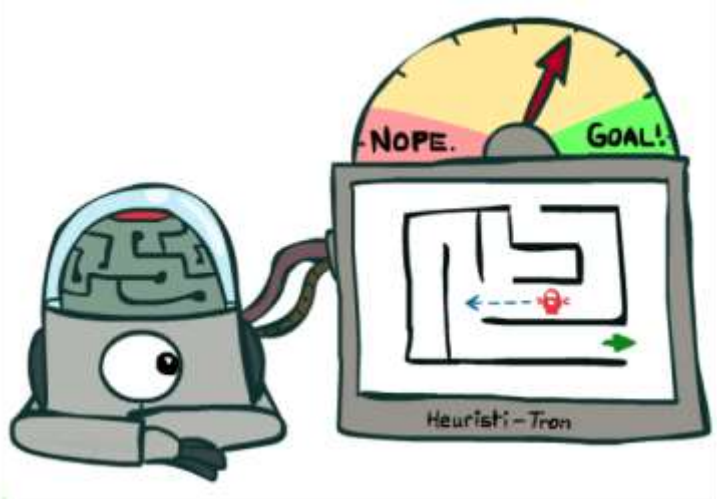
- Vođene Pretrage
  - Heuristike
  - Pohlepna Pretraga (*Greedy Search*)
  - A* Algoritam
  - Graf Pretraga

Heuristika Za Pretragu

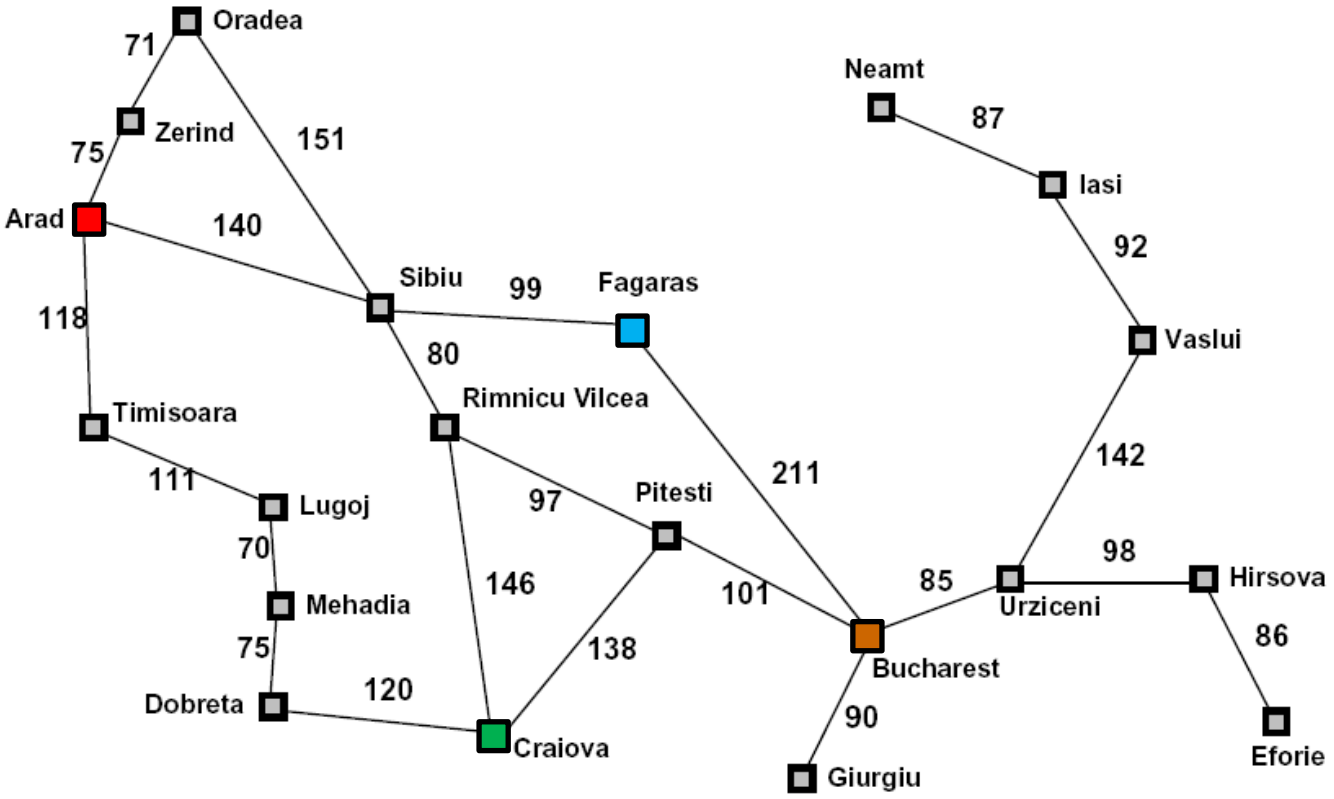
- Heuristika:
  - Funkcija koja procenjuje koliko je neko stanje daleko od cilja.
  - Kreira se za svaki problem (ili klasu problema) posebno.
  - Pronalaženje puta?
  - Primer: Manhattan udaljenost, Euklidska udaljenost







Primer: Heuristička Funkcija



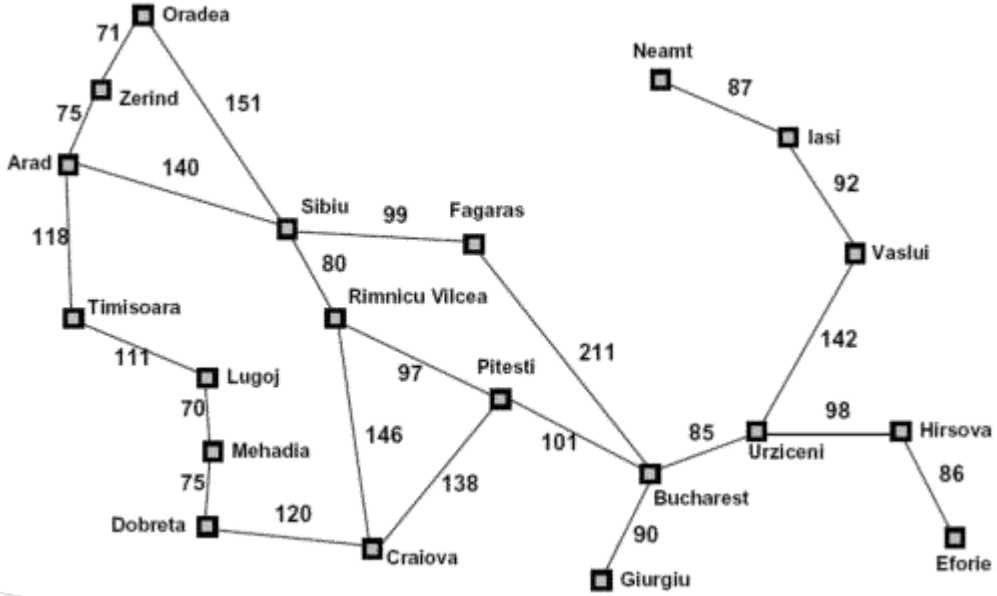
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

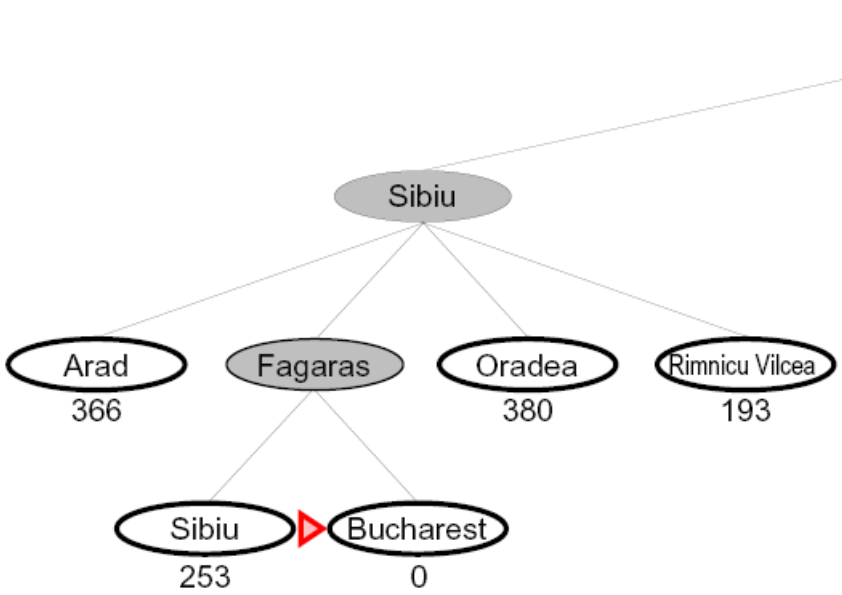
h(x)

Pohlepna Pretraga

Pohlepna Pretraga

- Razvijamo čvor sa najboljom vrednošću heuristike…





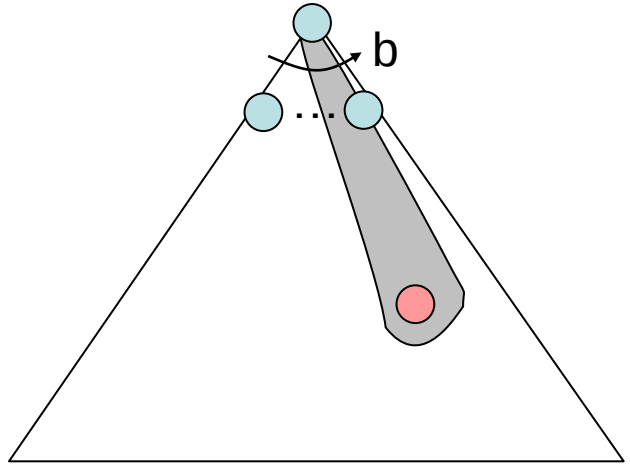
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

$h(n)$ = prvolinijska udaljenost do Bukurešta

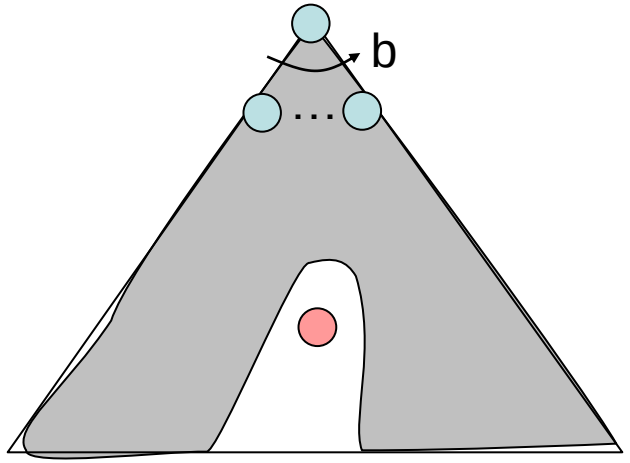
- Da li je optimalan?
  - NE. Putanja do Bukurešta koju smo dobili nije najkraća! Ar-Si-Fa-Bu = 450km, Ar-Si-Ri-Pi-Bu = 418km

Pohlepna Pretraga

- Strategija: razvija čvor za koji je procenjeno da je najbliži cilju
  - Heuristika: procena udaljenosti stanja od cilja

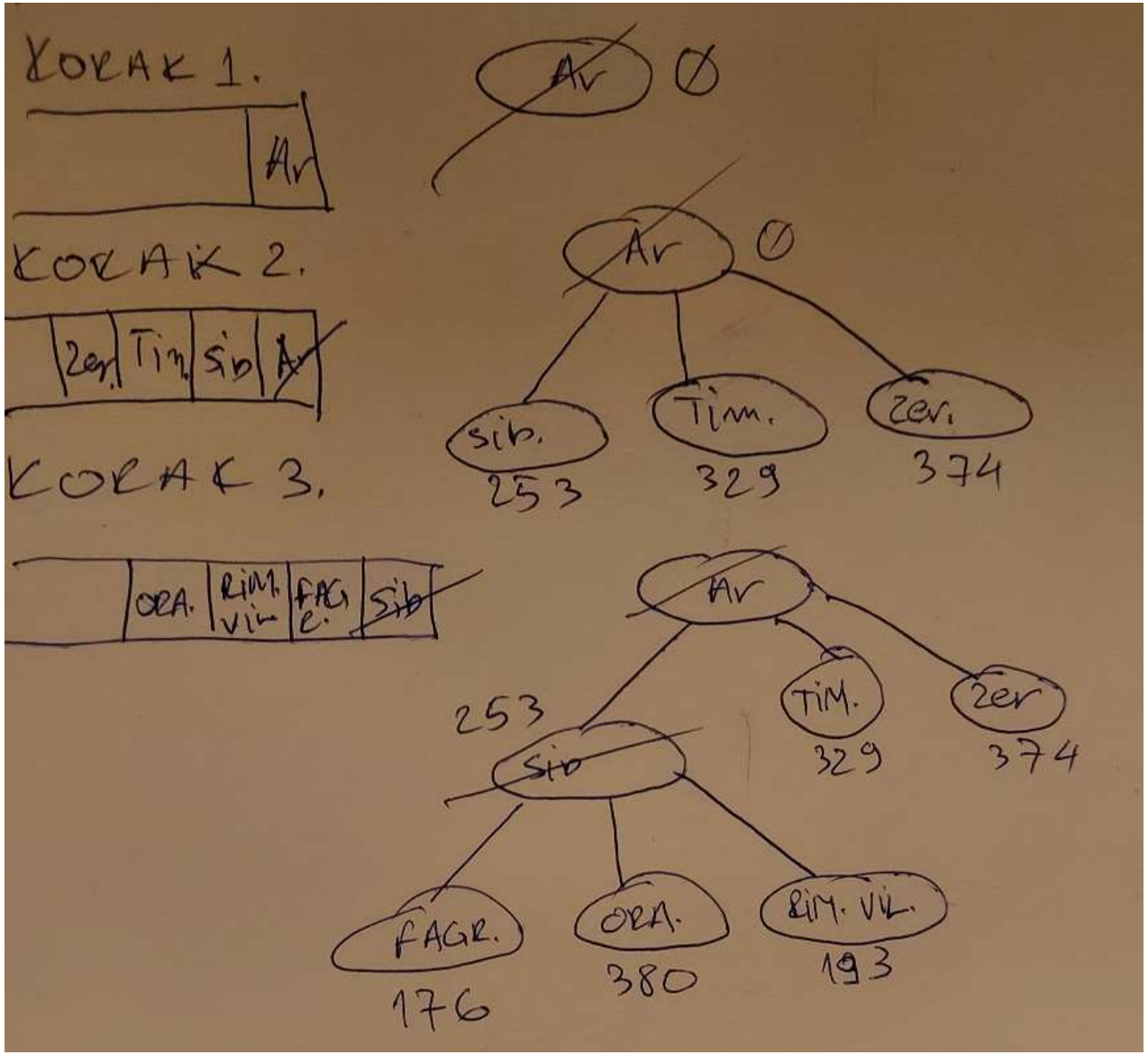


- Uobičajeno ponašanje:
  - Brzo stiže do cilja putem koji nije optimalan



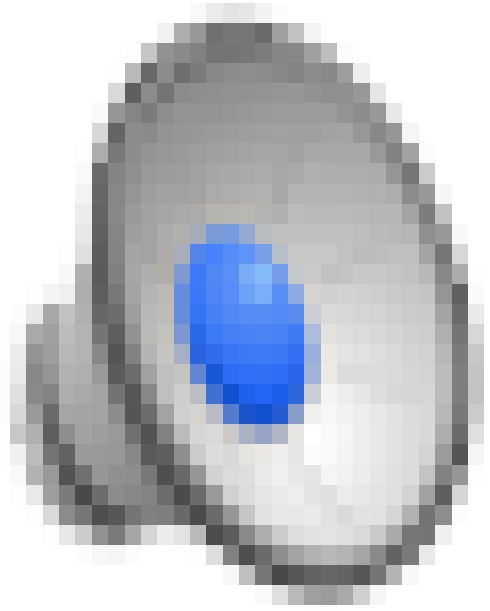
- Najgori mogući scenario (ako je heuristika loša): slično ponašanje kao DFS

Pohlepna pretraga - pojašnjenje



- Algoritam funkcioniše kao UCS.
- Umesto kumulativne cene koristi se vrednost heuristike.

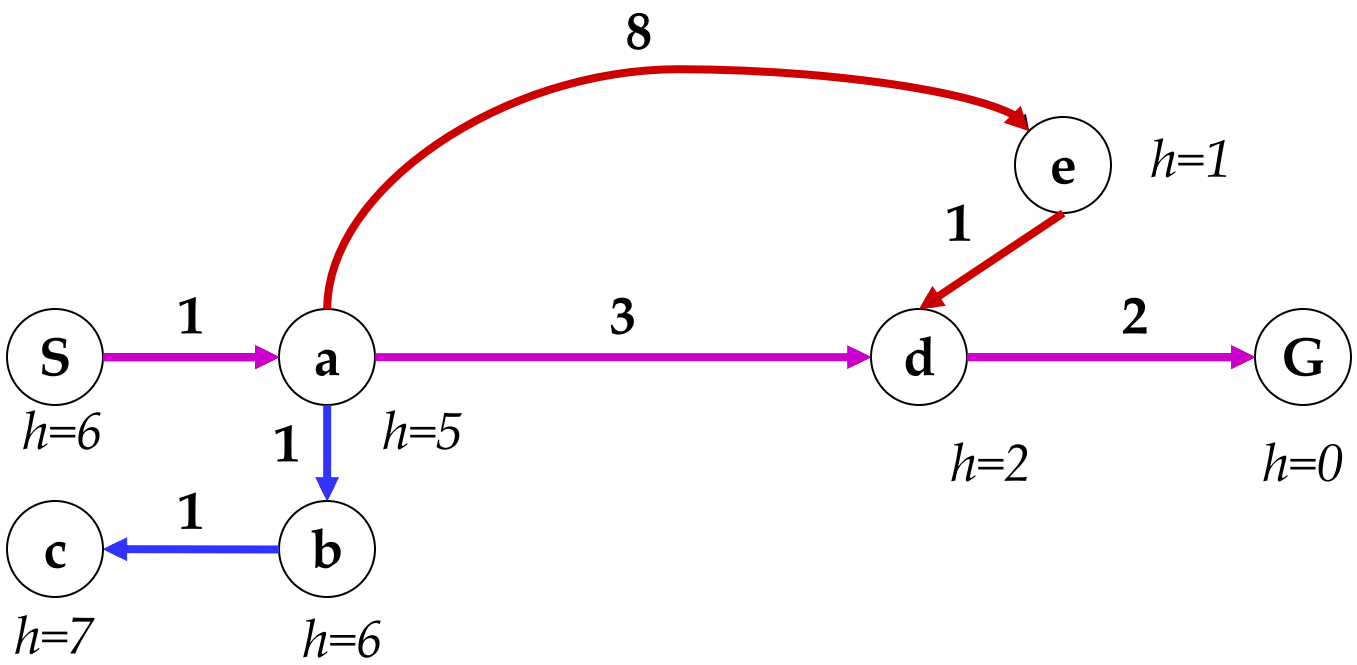
Demo – Pohlepna Pretraga Pacman – svetlija boja = kasnije razvijen čvor

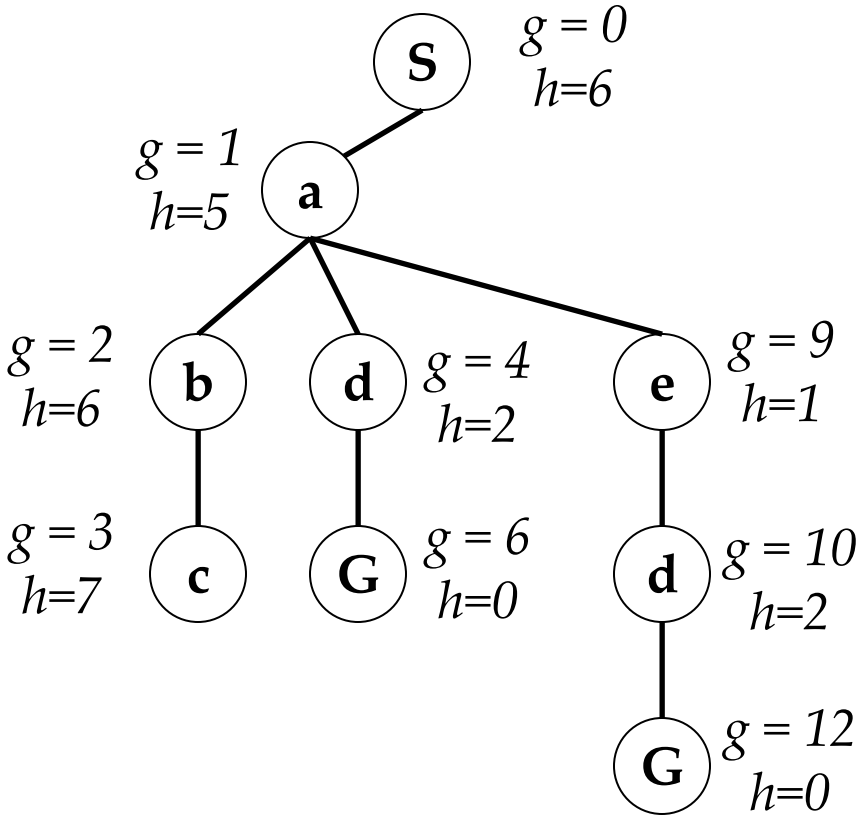


A* Algoritam

Kombinacija UCS i Pohlepne Pretrage

- UCS koristi cene akcija (puta) ili *cenu unazad* g(n)
- Pohlepna Pregraga koristi udaljenost od cilja ili *cenu unapred* h(n)



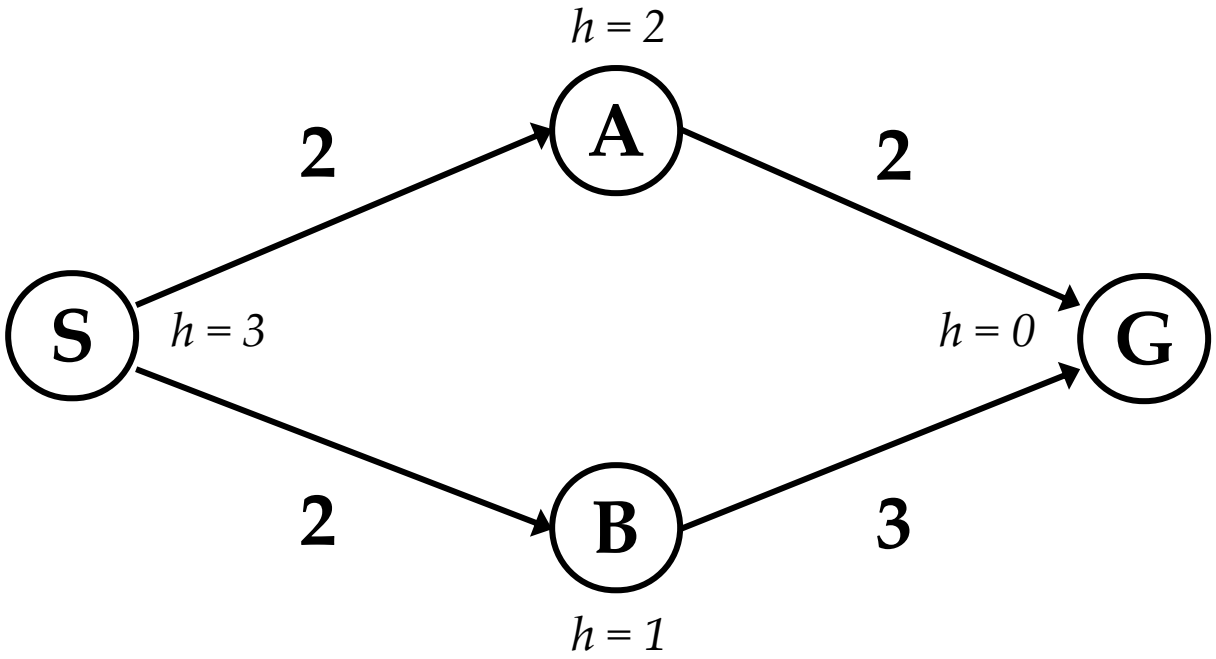


- A* Algoritam koristi zbir: f(n) = g(n) + h(n)

Primer: Teg Grenager

Kada zaustaviti A* algoritam?

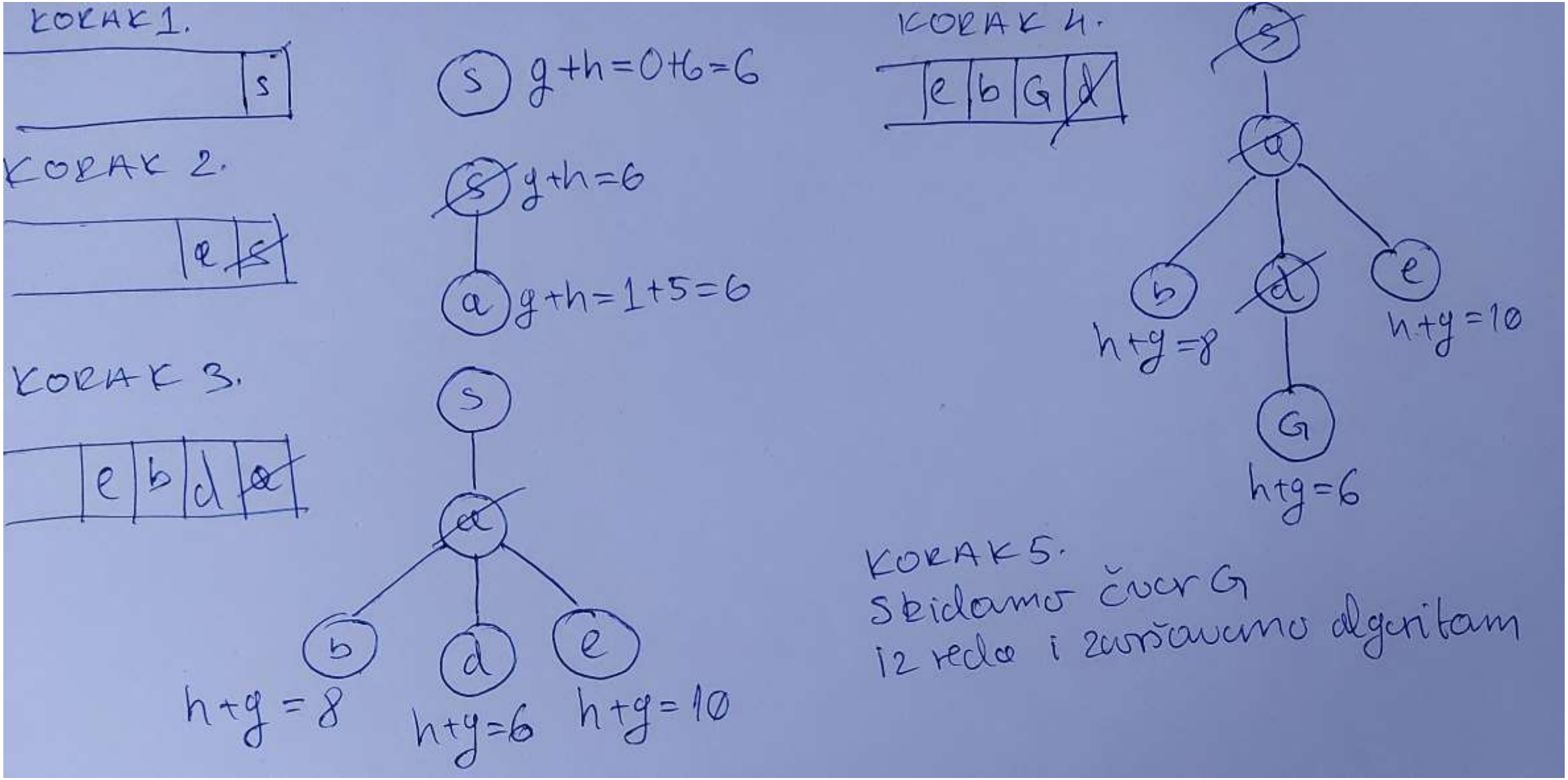
- Kada stavimo cilj u strukturu?



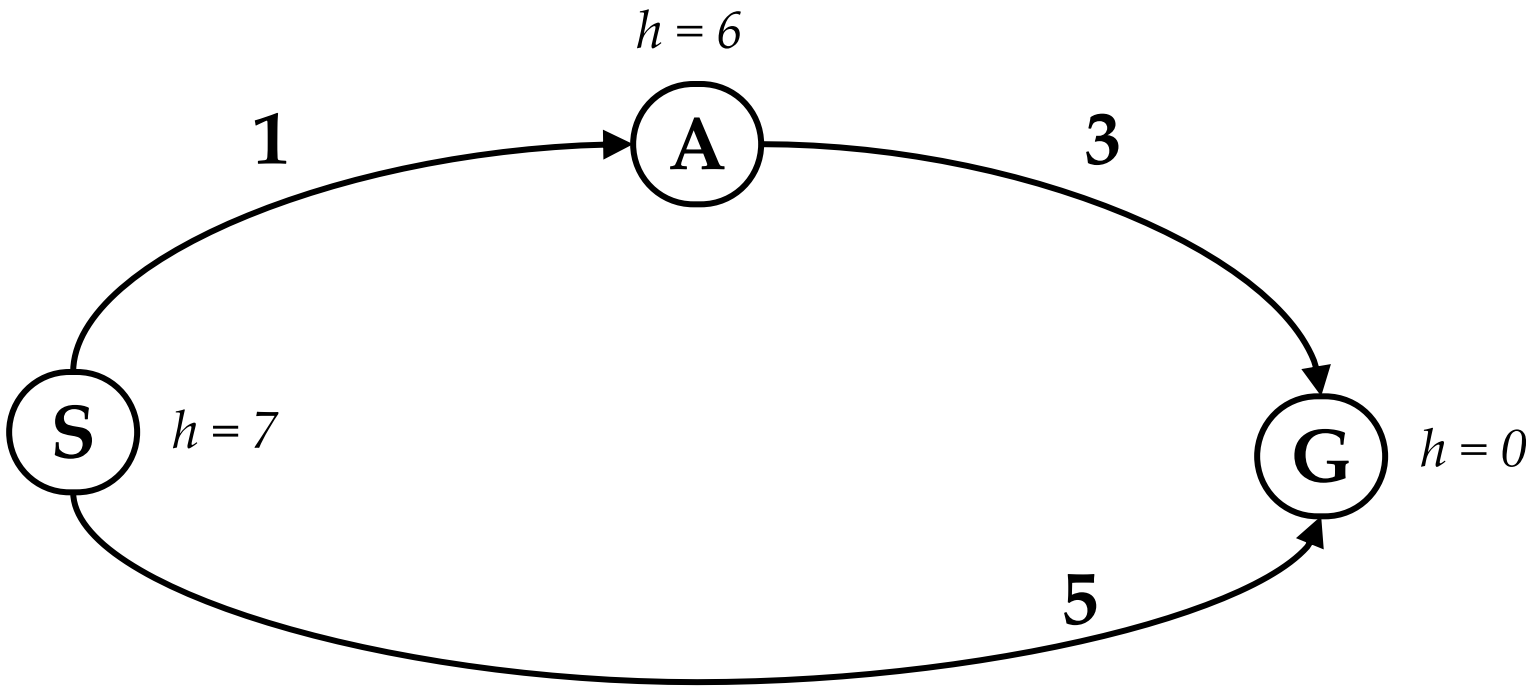
	g	h	+
~~S~~	~~0~~	~~3~~	~~3~~
~~S->A~~	~~2~~	~~2~~	~~4~~
~~S->B~~	~~2~~	~~1~~	~~3~~
S->B->G	5	0	5
S->A->G	4	0	4

- Ne: tek kad izvučemo cilj iz strukture.

A* - pojašnjenje



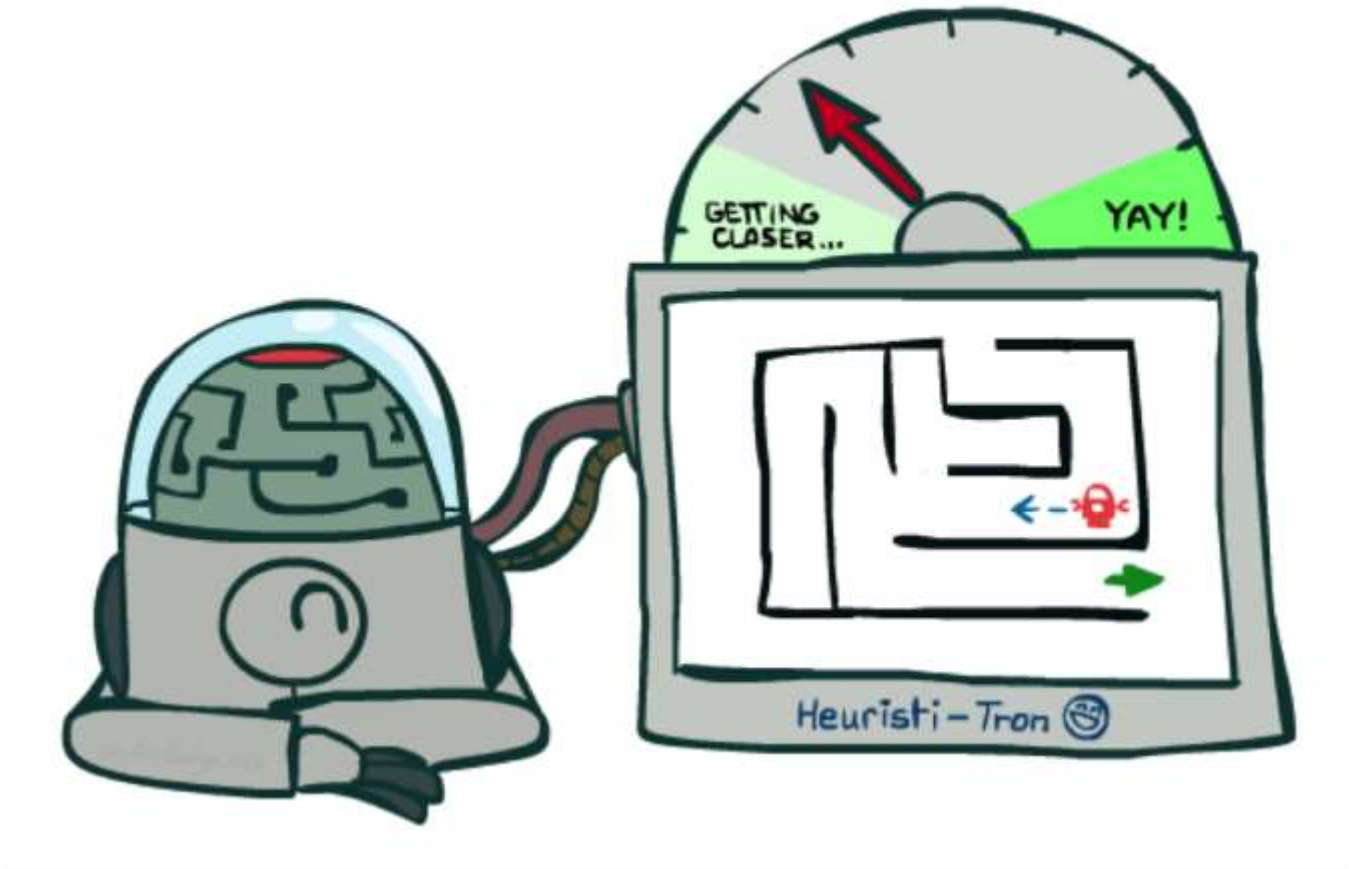
Da li je A* optimalan?



	g	h	+
S	0	7	7
S->A	1	6	7
S->G	5	0	5

- Zašto nismo našli najkraći put?
- Prava cena do lošeg čvora (G) < procenjene cene za dobar čvor (A)
- Naše procene moraju da budu manje od prave cene!

Dopustive Heuristike (*Admissible Heuristics*)



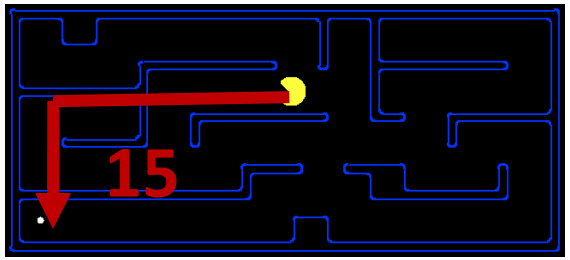
Dopustive Heuristika

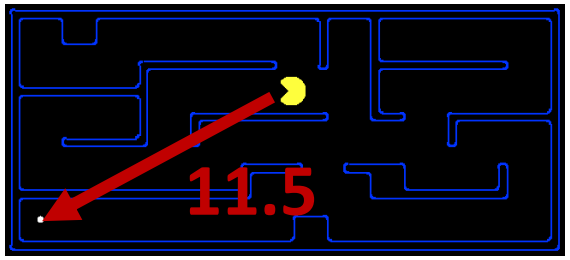
- Heuristika *h* je *dopustiva* (optimistična) ako:

$$0 \leq h(n) \leq h^*(n)$$

gde je $h^*(n)$ prava cena do najbližeg cilja.

- Primeri:



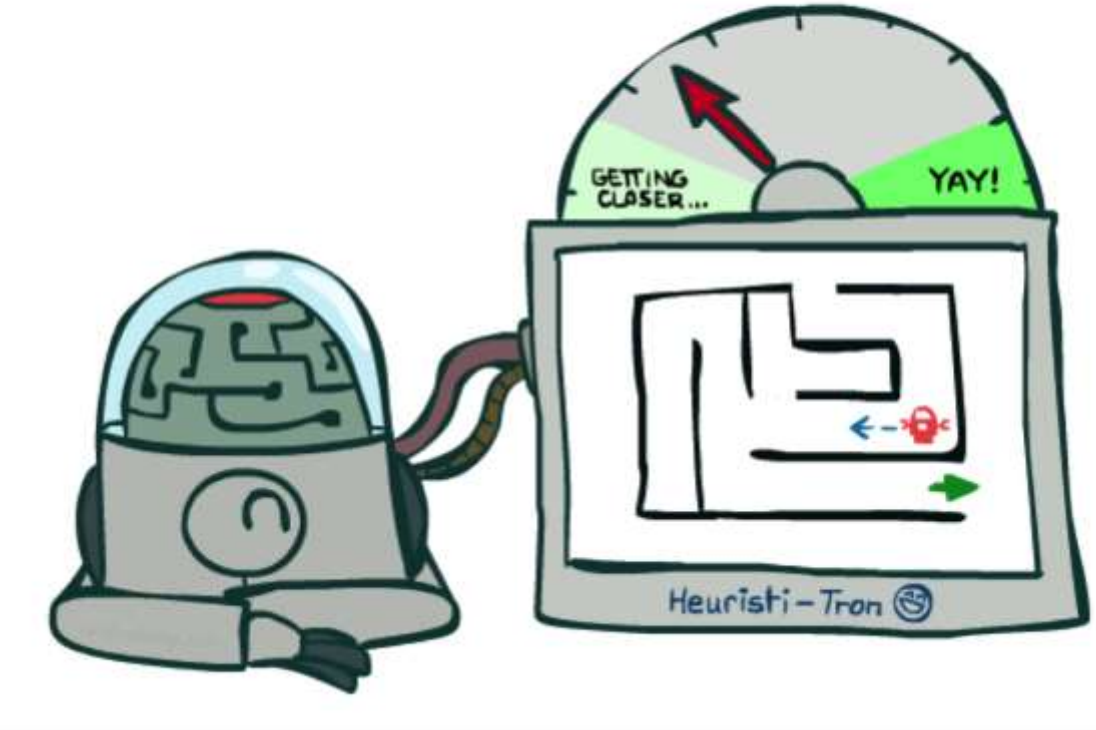


0.0

- Najviše posla kod A* je u pronalaženju dopustive heuristike.

Optimalnost A*

A* je optimalan ako koristimo dopustivu heuristiku!



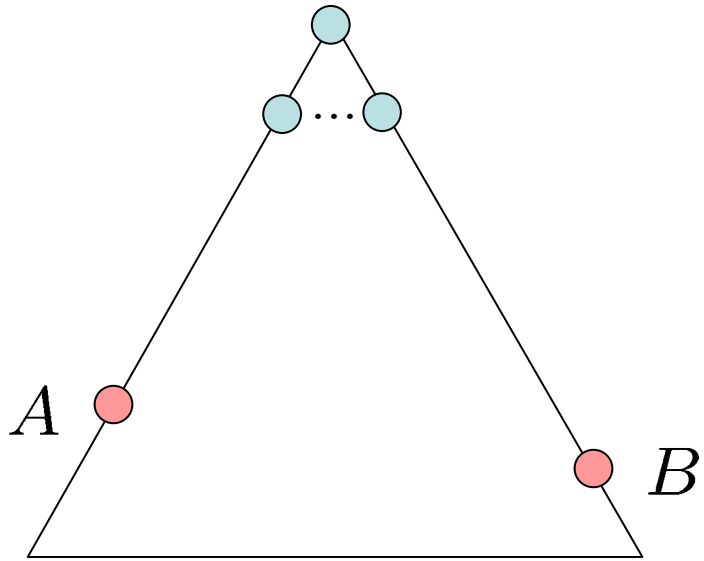
Optimalnost A* - dokaz

Pretpostavke:

- A je optimalan ciljni čvor
- B je neoptimalan ciljni čvor
- h je dopustiva

Tvrđenje:

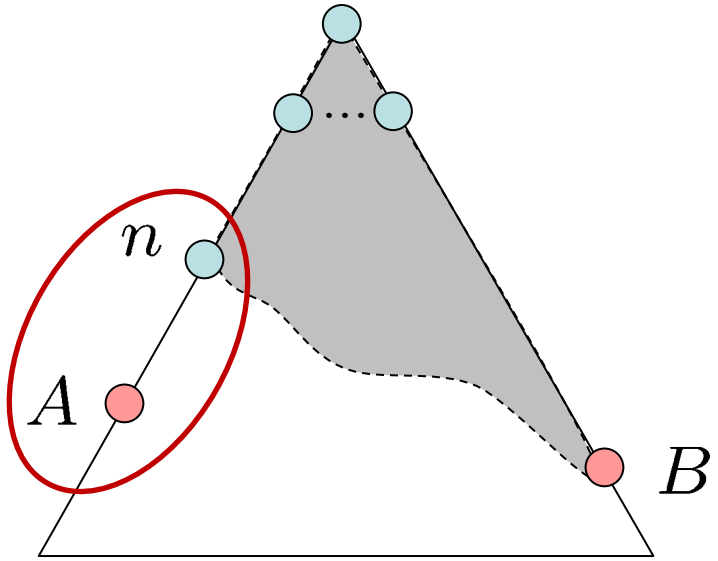
- A će biti razvijen (posećen) pre B



Optimalnost A* - dokaz

Dokaz:

- Neka je B u strukturi za razvijanje
- Neki predak *n* čvora A je takođe u strukturi (*n* može biti i sam čvor A!)
  - Ako to ne važi onda je A već razvijen i optimalno rešenje je pronađeno
- Tvrđenje: *n* će biti razvijen pre B
  1. f(n) je manje ili jednako od f(A)



f(n)=g(n)+h(n)
f(A)=g(A)+h(A)=g(A)
jer je h=0 za ciljne čvorove ako je h dopustiva (potcenjuje pravu cenu koja je 0)

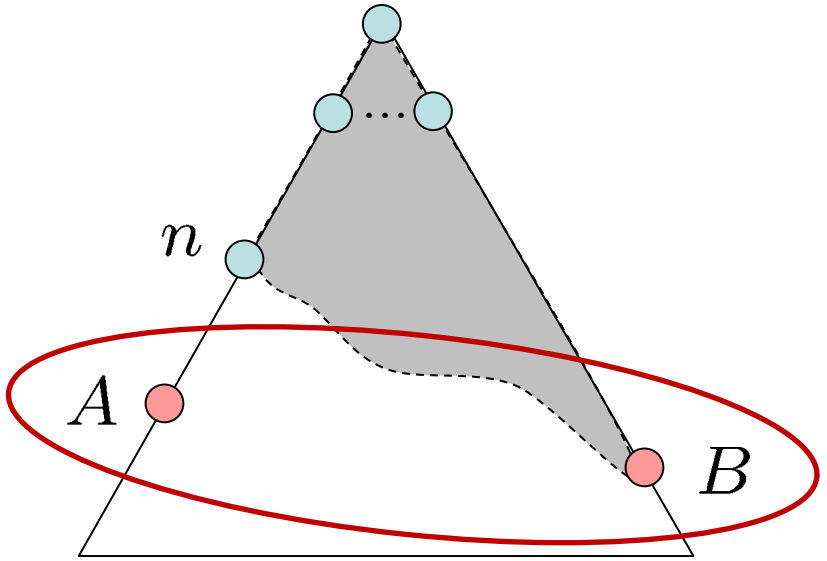
Po definiciji dopustive heursitike mora da važi: h(n)≤g(A)-g(n)
jer dopustiva heurstika potcenjuje pravu cenu od nekog čvora (u ovom slučaju n) do ciljnog čvora (u ovom slučaju A). Prava cena je g(A)-g(n).

f(n)=g(n)+h(n)≤g(n)+g(A)-g(n)=g(A) pa je:
f(n)≤ f(A) jer je f(A)=g(A).

Optimalnost A* - dokaz

Dokaz:

- Neka je B u strukturi za razvijanje
- Neki predak *n* čvora A je takođe u strukturi (n može biti i sam čvor A!)
- Tvrđenje: *n* će biti razvijen pre B
  1. f(n) je manje ili jednako od f(A)
  2. f(A) je manje od f(B)



g(A)<g(B) jer je B neoptimalan ciljni čvor.

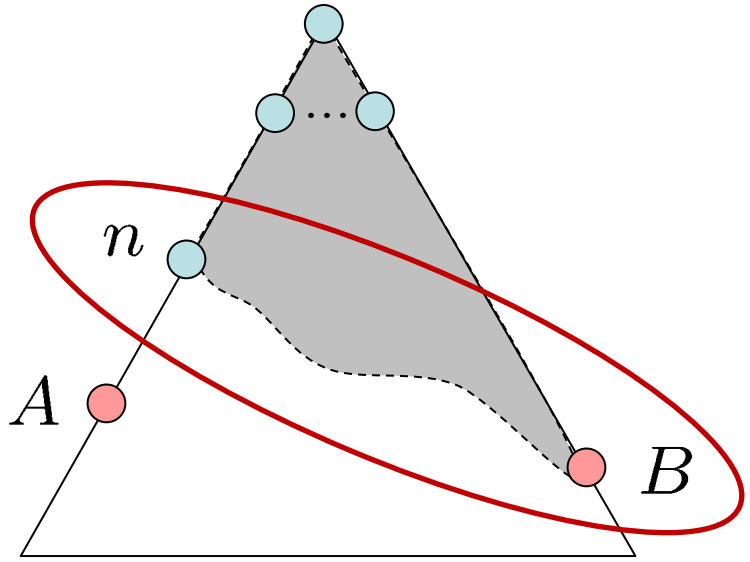
f(B)=g(B)+h(B)=g(B) jer je h=0 za ciljne čvorove
f(A)=g(A)+h(A)=g(A) jer je h=0 za ciljne čvorove

Pošto je g(A)<g(B) onda važi f(A)<f(B)

Optimalnost A* - dokaz

Dokaz:

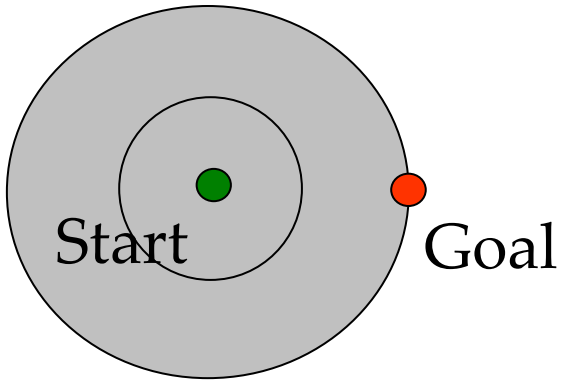
- Neka je B u strukturi za razvijanje
- Neki predak *n* čvora A je takođe u strukturi (n može biti i sam čvor A!)
- Tvrđenje: *n* će biti razvijen pre B
  1. f(n) je manje ili jednako od f(A)
  2. f(A) je manje od f(B)
  3. n će biti razvijen pre B
- Svi preci A će biti razvijeni pre B
- A će biti razvijen pre B
- A* pretraga je optimalna



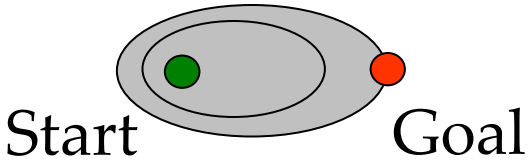
$$f(n) \leq f(A) < f(B)$$

UCS vs. A* - konture

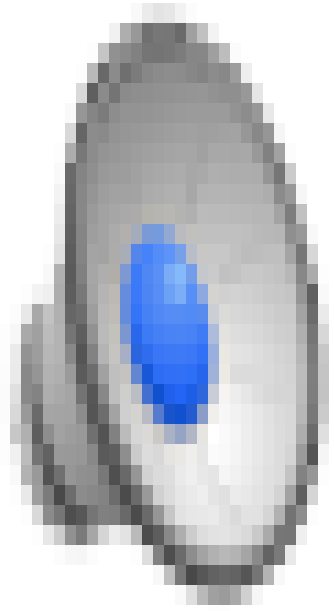
- UCS razvija jednako u svim pravcima.



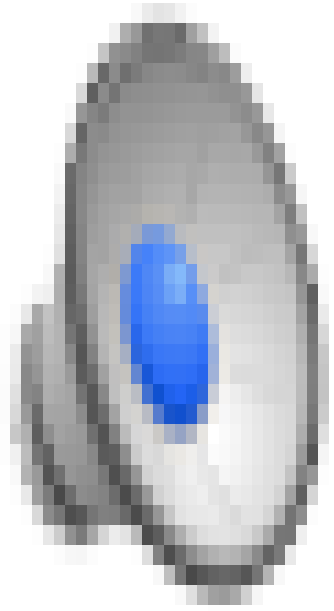
- A* uglavnom razvija prema cilju, ali ne ako će to „ugroziti“ optimalnost.



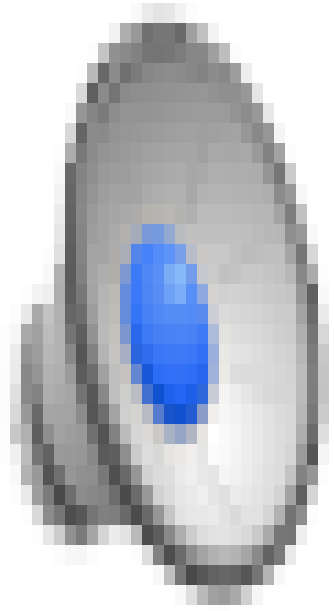
Demo – UCS konture, jednake cene



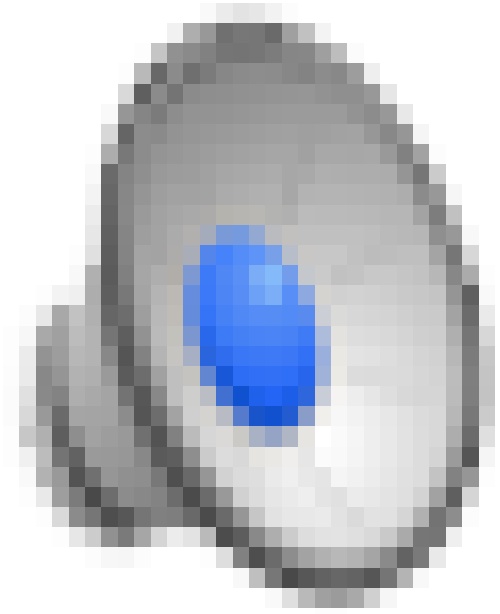
Demo – Pohlepna Pretraga, jednake cene



Demo – A* konture, jednake cene



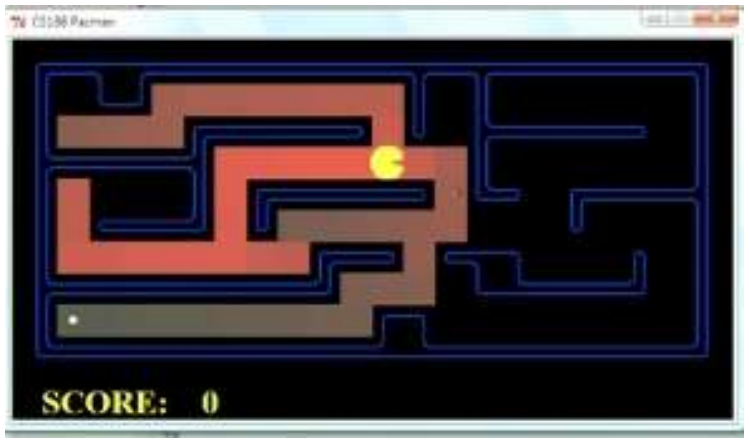
Demo – A* konture, Pacman



Poređenje

Pohlepna Pretraga

UCS



A*

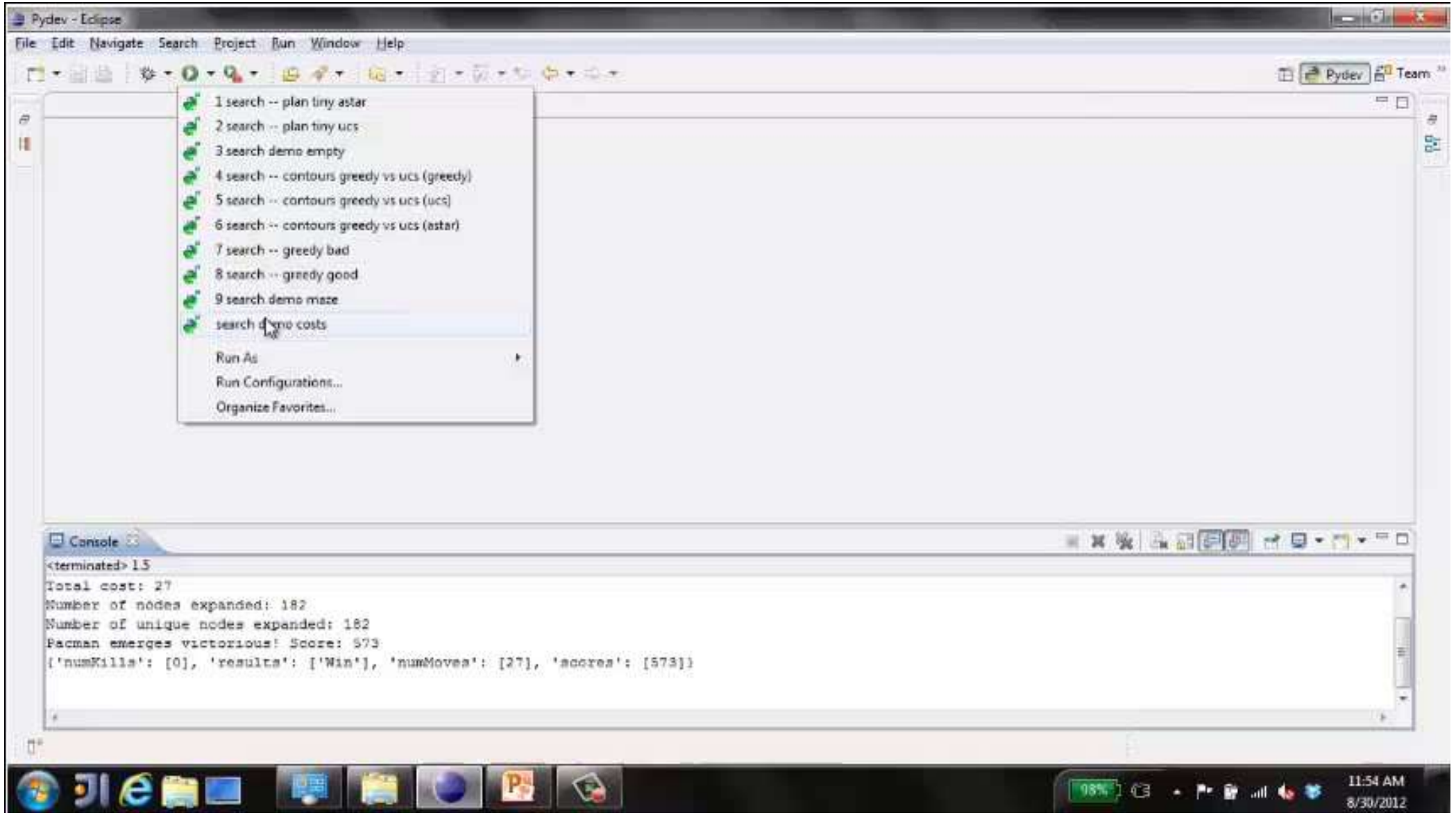
A* Primene


APM: 487
342
0
15/17
Terran SCV
Terran Marine
Terran SCV
Terran SCV
Terran SCV
Terran SCV
Terran SCV
Terran SCV
Terran SCV
Terran SCV
G attack: false G defense: false
[3663:WorkerLoiterAndHarassController16] INFO - Scared of a visible Terran Marine
LOITER_AND_HARASS
MOVE_TO_CANARY
MENU

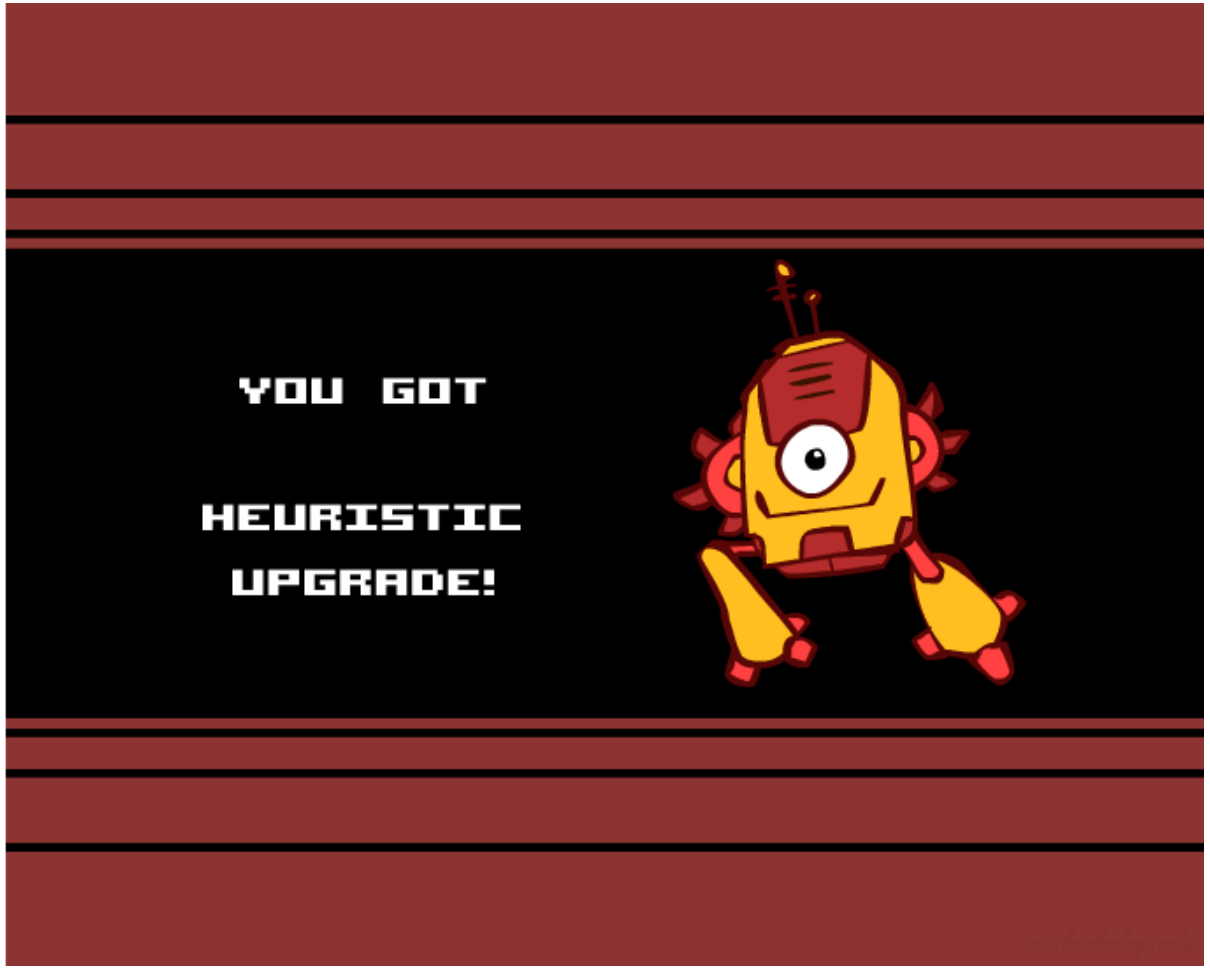
A* Primene

- Video igre
- Problemi rutiranja (traženja puta)
- Planiranje resursa
- Planiranje pokreta robota
- NLP
- Mašinski prevod
- Prepoznavanje govora
- …

Demo – duboka i plitka voda – koji je koji alg.?



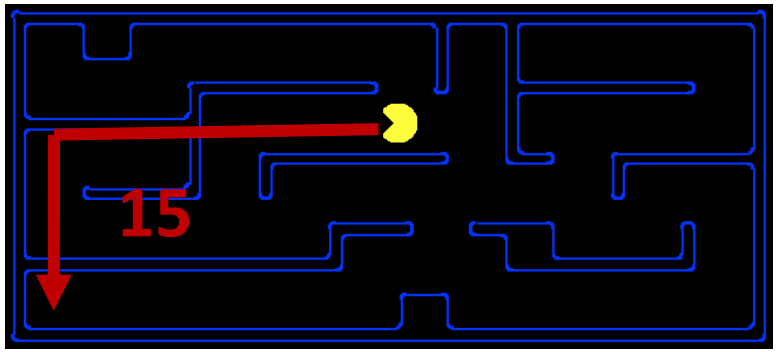
Kreiranja Heuristika



Kreiranja Dopustivih Heuristika

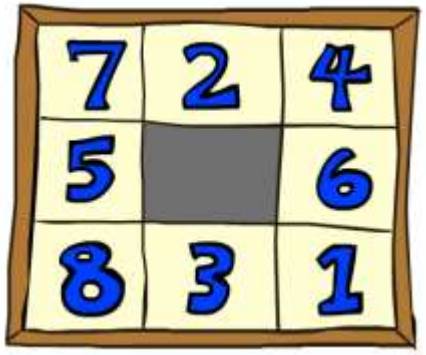
- Najviše posla kod rešavanja težih problema pretraga na optimalan način je pronalaženje dopustive heuristike.

- Često su dopustive heuristike rešenja *relaksiranih problema,* kod kojih „imamo“ akcije koje inače nemamo (npr. dijagonalno kretanje).

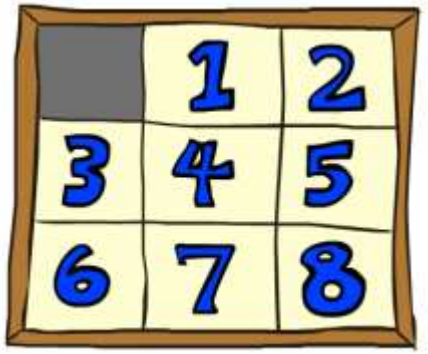


Primer: Slagalica

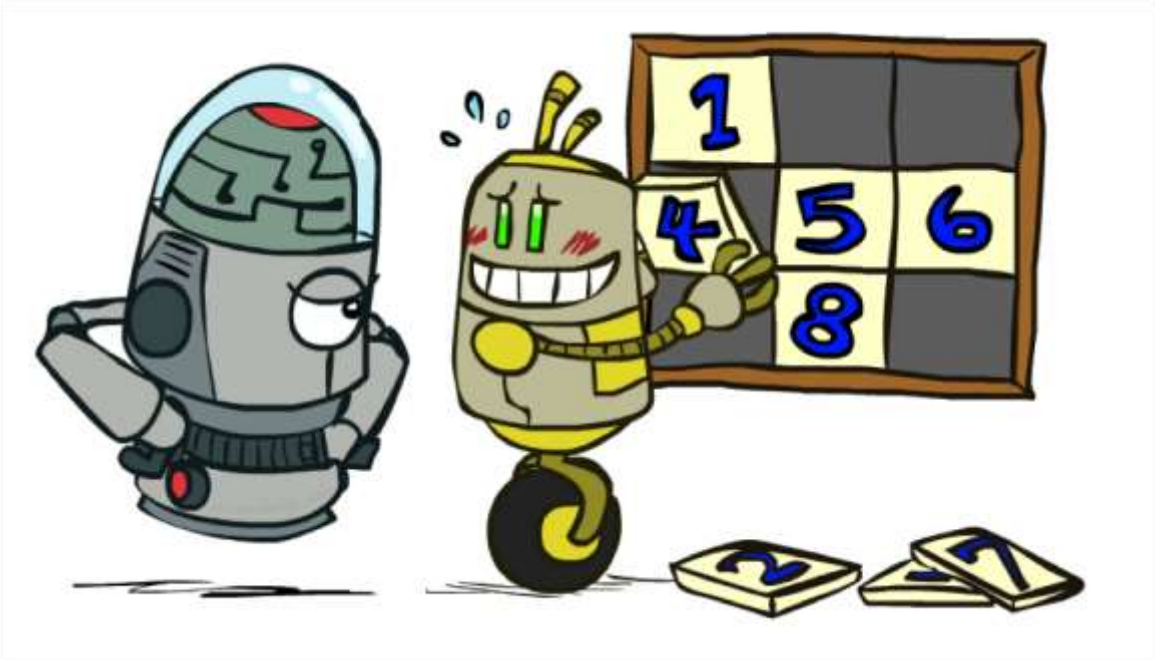
- Heuristika: Broj delova van svog mesta (TILES)
- Zašto je dopustiva?
- h(start) = 8
- Ovo je heuristika *relaksiranog problema*



Start State



Goal State



	Prosečan broj čvorova razvijen kada optimalno rešenje ima:		
	…4 koraka	…8 koraka	…12 koraka
UCS	112	6,300	3.6×10^6
TILES	13	39	227

Statistics from Andrew Moore

Slagalica

- Šta bi bilo ako bi znali i koristili stvarnu cenu kao heuristiku?
  - Da li bi bila dopustiva?
  - Da li bi razvijali manje čvorova?
  - U čemu je problem?

- A*: trampa (*trade-off*) između kvaliteta i kompleksnosti heuristike
  - Ako je heuristika jako dobra brže razvijamo manje čvorova, ali su takve herustike obično kompleksne i troše puno vremena.

Dominantnost Heuristika

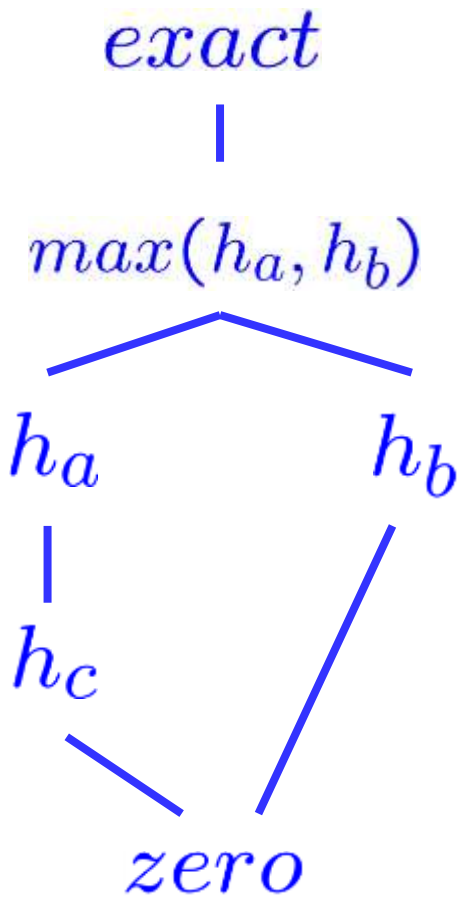
- Dominantnost: $h_a \geq h_c$ ako:

$$\forall n : h_a(n) \geq h_c(n)$$

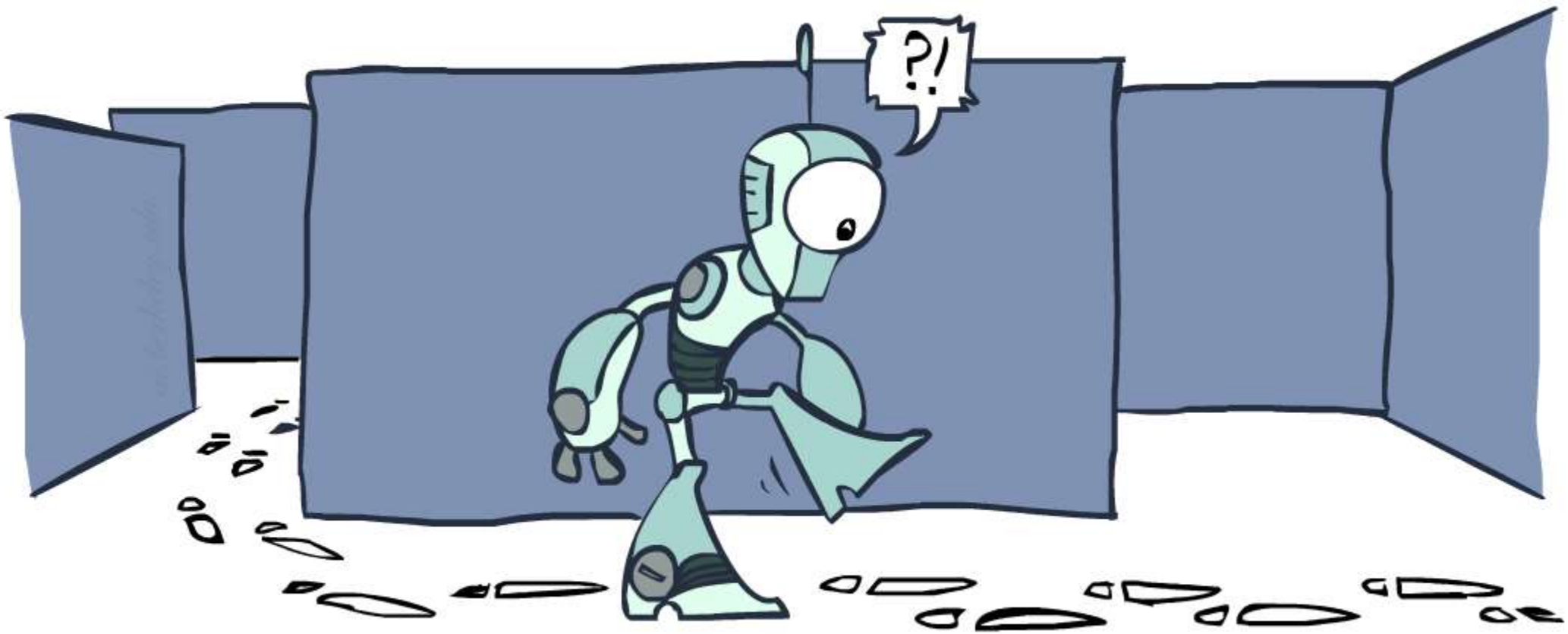
- Heuristike prate strukturu na slici:
  - Max dopustivih heuristika je dopustva heuristika

$$h(n) = max(h_a(n), h_b(n))$$

- Trivijalne heuristike
  - Na dnu imamo nula heursitiku
  - Na vrhu je stvarna cena
  - Šta time dobijamo?
  - Često je dobro rešenje pronaći puno trivijalnih heuristika za neki problem i onda uzeti njihov maksimum kao konačnu heuristiku.

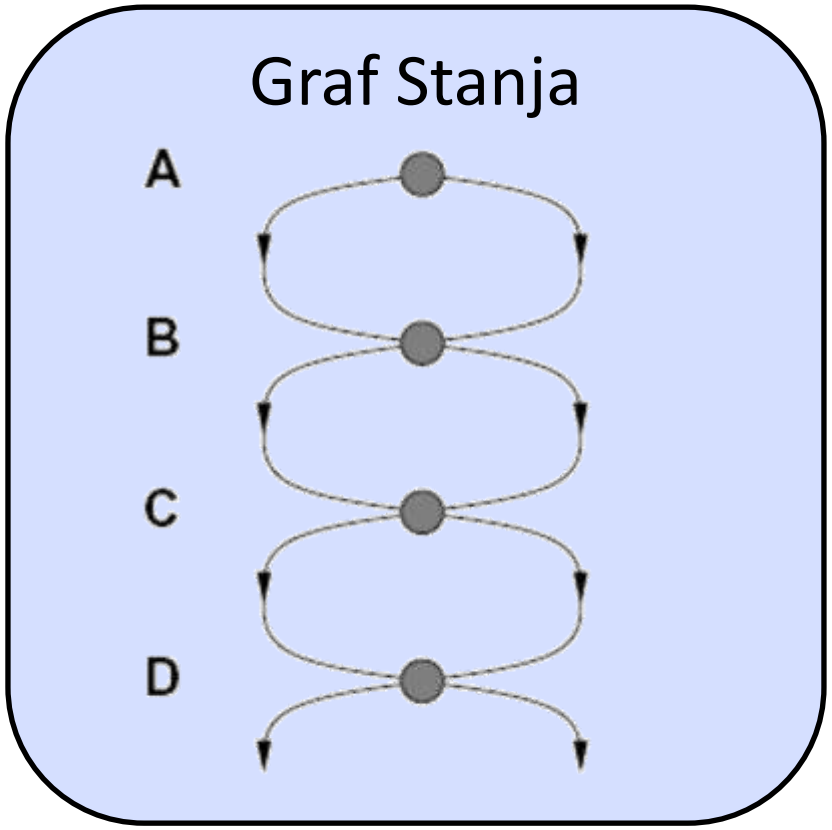


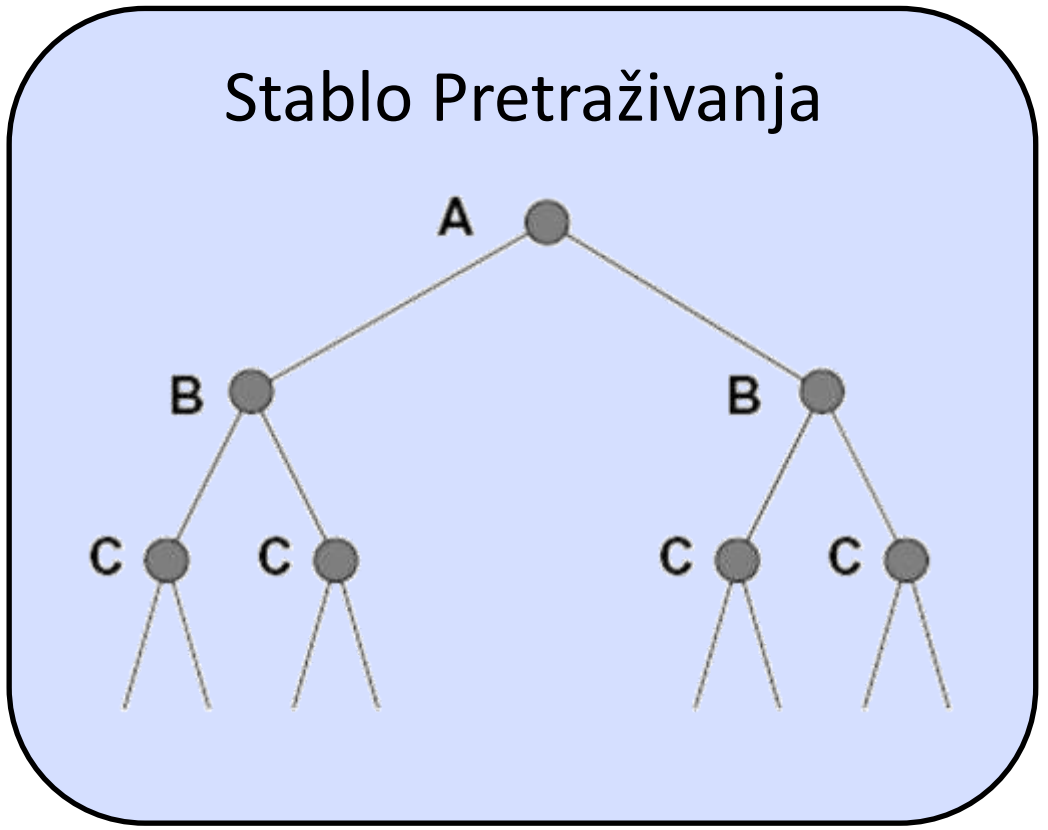
Graf Pretraga



Stablo Pretraživanja: Dodatan Posao!

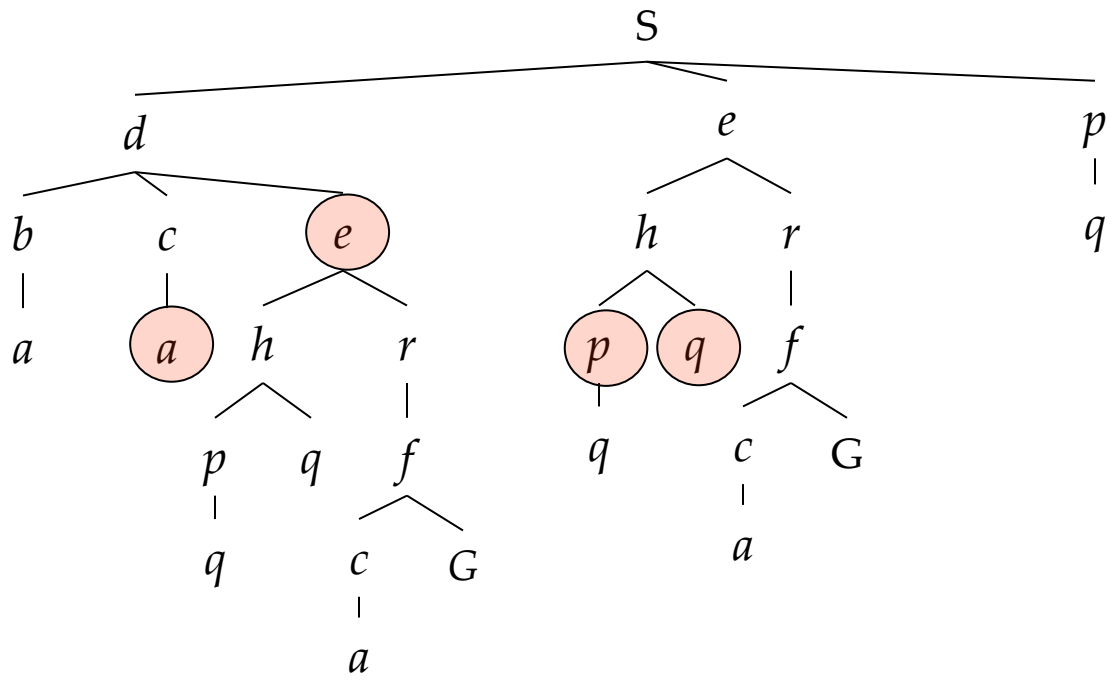
- Ako ne uspemo da detektujemo petlje u grafu tj. ponavljanje stanja, razvijaćemo eksponecijalno mnogo više čvorova nego što treba.





Graf Pretraga

- Kod BFS, na primer, ne bi trebalo da razvijamo označene čvorove (zašto?)

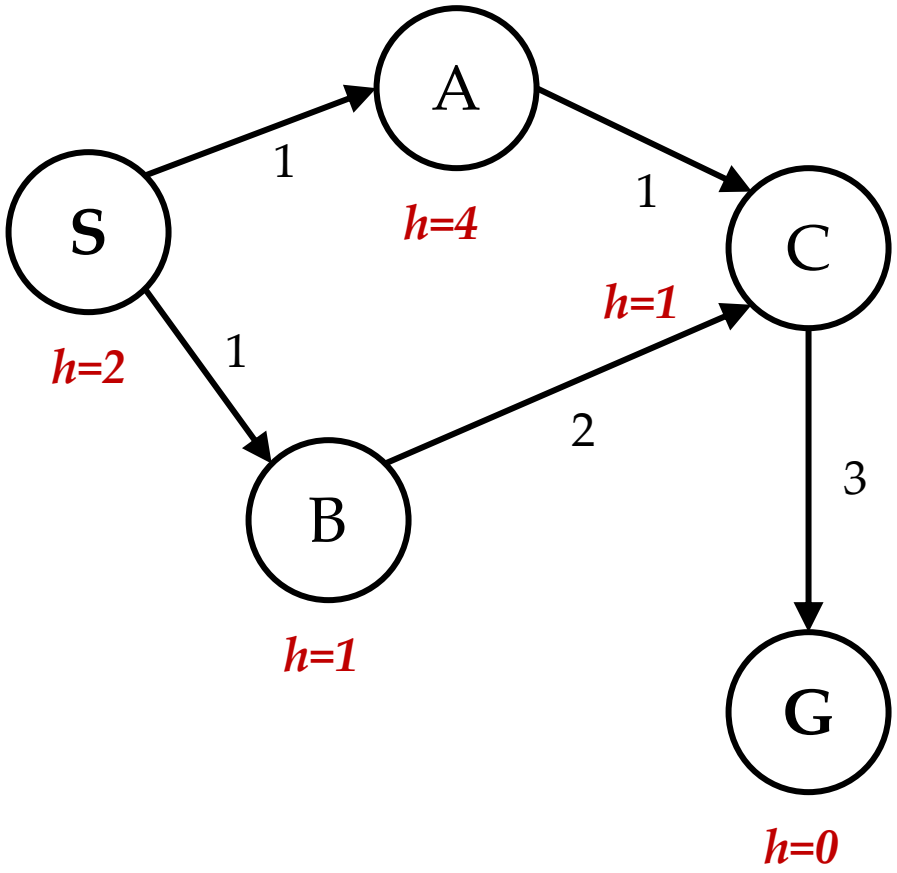


Graf Pretraga

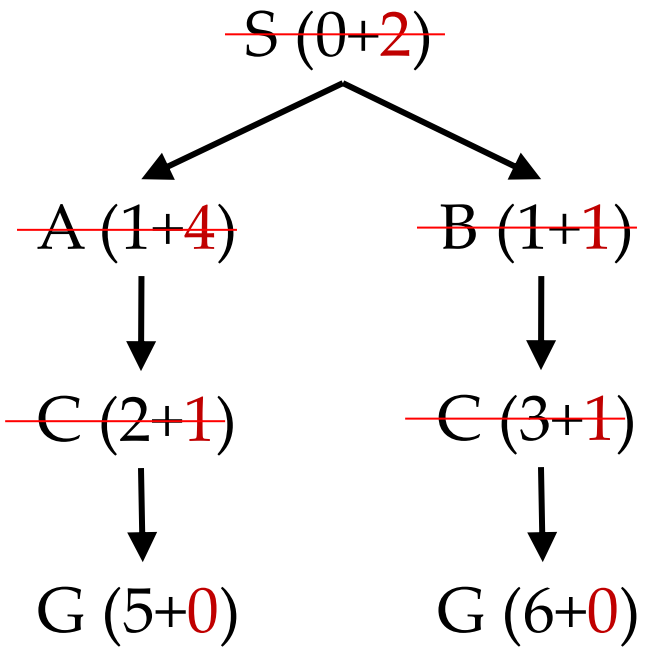
- Ideja: nikad ne razvijamo stanje dva puta
- Implementacija:
  - Stablo Pretraživanja + skup do sada razvijenih čvorova (“zatvoreni skup”)
  - Razvijamo stablo čvor po čvor, ali…
  - Pre nego što razvijemo čvor proverimo da li se nalazi u zatvorenom skupu
  - Ako da, onda ga preskačemo, inače ga razvijamo i dodajemo u zatvoreni skup
- Važno: zatvoreni skup treba da bude implementiran kao skup (ili heš tabela, rečnik...), a ne lista
- Da li dodatak Graf Pretrage kvari kompletnost?
- Da li kvari optimalnost?

A* Graf Pretraga, Problem?

Graf Stanja



Stablo Pretraživanja



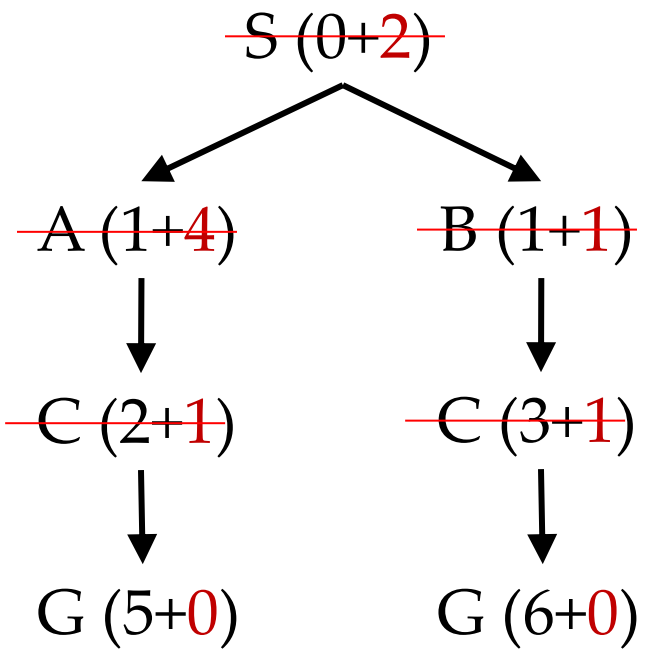
Drugo C tj. C (2+1) ne razvijamo jer smo C već razvili.
Međutim to C nas vodi do kraćeg puta S A C G

Zatvoreni Skup:S B C A

A* Graf Pretraga, Problem?

- Zašto A* nije pronašao optimalan put u ovom slučaju?
- Čvor C sa f vrednošću od 3+1, razvijen je pre čvora C sa f vrednošću od 2+1.
- To znači da smo do C prvi put stigli dužim putem nego što je moguće.
- Da li postoji način da se nekako obezbedimo da se to ne događa tj. da prvi put kada razvijemo neki čvor uvek dođemo optimalnim tj. najkraćim putem do njega?
- Postoji, ali moramo da nametnemo dodatna ograničenja na heuristiku.
- Tako dolazimo do pojma Dosledne Heuristike (sledeći slajd).

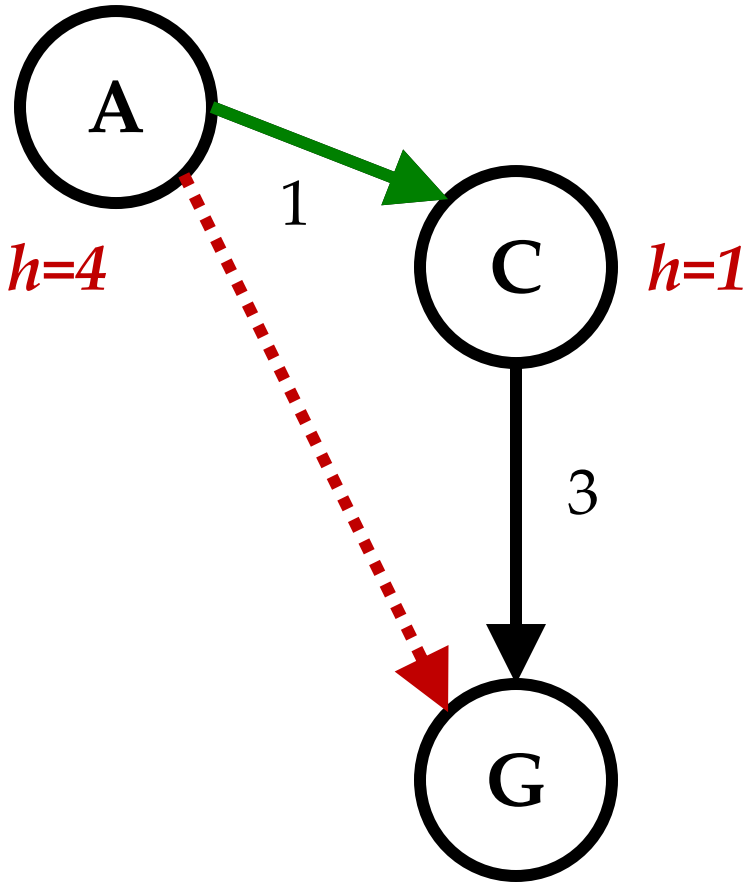
Stablo Pretraživanja



Drugo C tj. C (2+1) ne razvijamo jer smo C već razvili.
Međutim to C nas vodi do kraćeg puta S A C G

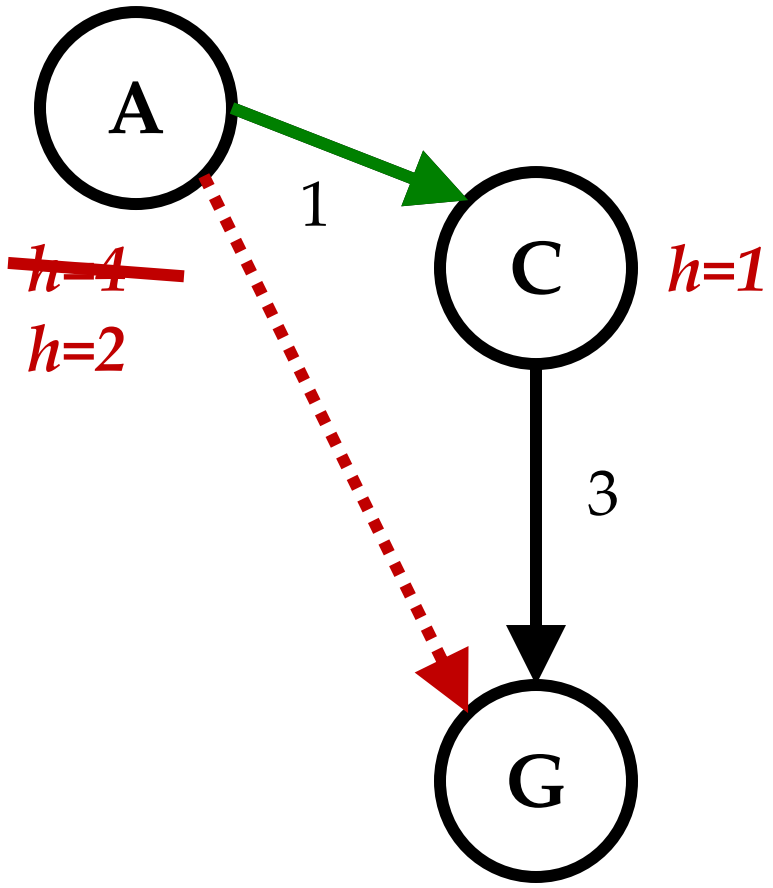
Zatvoreni Skup:S B C A

Dosledna Heuristika



- Šta je problem na slici?
- Heuristička cena grane (A,C) je h(A)-h(C) tj. koliko je heuristika opala kada smo prešli iz A u C.
- To je procena cene grane (pad heuristike). Stvarna cena grane je cena(A do C).
- Želimo da heuristika bude dosledna tj. da procena cene grane uvek bude manja ili jednaka od stvarne cene grane.
- Na primer, h(B)-h(C)=1-1=0, a stvarna cena od B do C je 2.
- Dok je h(A)-h(C)=4-1=3 a cena(A do C)=1. Dakle, heuristika nije dosledna jer je procena cene grane u nekim slučajevima manja, a u nekim veća od stvrane cene grane.

Dosledna Heuristika



- Ideja: procena cene ≤ stvarna cena
  - Dopustivost: cena od heuristike ≤ stvarna cena do cilja

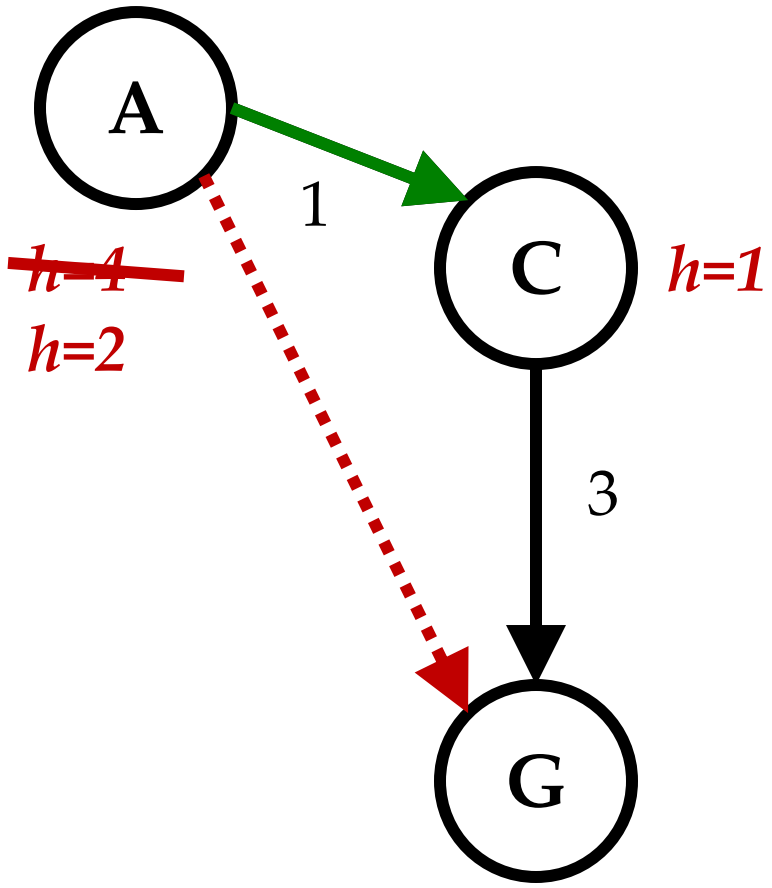
    h(A) ≤ stvarna cena A do G
  - Doslednost: heuristička cena „grane" ≤ stvarna cena „grane"

    h(A) – h(C) ≤ cena(A do C)

    h(A) ≤ cena(A do C) + h(C)
- Posledica doslednosti:
  - Vrednost f=h+g nikada ne opada na putu do cilja

    h(A) ≤ cena(A do C) + h(C)
  - A* sa graf pretragom je sad optimalan algoritam!

Dosledna Heuristika



- Ako je herustika dosledna f ne opada na putu od starta do clilja.
- Koristimo:

$$h(A) \leq \text{cena}(A \text{ do } C) + h(c)$$

- $f(C)=h(C)+g(C)=h(C)+g(A)+\text{cena}(A \text{ do } C) \geq g(A)+h(A)=f(A)$
- Dakle $f(C) \geq f(A)$ što znači da f ne opada na putu do cilja.
- Ako je herustika dosledna onda je A* graf pretraga optimalna

Dosledna Heuristika

- Ako je herustika dosledna onda je A* graf pretraga optimalna.
- Dokaz: Pretpostavimo suprotno tj. da je A* stigao do ciljnog čvora n nekim neoptimalnim putem sa cenom $g(n)$ i da postoji neki drugi optimalni put sa cenom $g^*(n)$. Znači $g^*(n)<g(n)$. Pošto put do n nije optimalan znači da A* nije razvio ceo optimalan put do n (jer da jeste onda ne bi do n stigli neoptimalnim putem), pa samim tim postoji neki čvor n' koji nismo još razvili koji je na optimalnom putu od starta do n. Zato važi $g^*(n')=g(n')$ (jer je n' na optimalnom putu) i $f(n')=g^*(n')+h(n')$.
- Pošto je n na putu posle n' i h je dosledna važi $h(n') \leq cena(n' \text{ do } n) + h(n)$.
- Tako dobijamo $f(n') = g^*(n') + h(n') \leq g^*(n') + cena(n' \text{ do } n) + h(n)$. Pošto je je g^* optimalna cena važi: $g^*(n) = g^*(n') + cena(n' \text{ do } n)$, ne postoji manja cena od starta do n od ove.
  Na početku smo rekli $g^*(n) < g(n)$, pa onda važi: $f(n') \leq g^*(n) + h(n) < g(n) + h(n) = f(n)$ tj. $f(n') < f(n)$
- Dobili smo $f(n') < f(n)$, ali to je u kontradikciji sa tim da smo prvo razvili n pa tek onda n'. Doslednost heuristike i prethodni slajd kažu da smo morali prvo da razvijemo n'. Pošto smo pretpostavli suprotno i došli do kontradikcije, važi tvrdnja da ako je herustika dosledna onda je A* graf pretraga optimalna.

Pretraga Pomoću Stabla - Pseudo-Kod

```
function TREE-SEARCH(problem, fringe) return a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        for child-node in EXPAND(STATE[node], problem) do
            fringe ← INSERT(child-node, fringe)
        end
    end
```

Graf Pretraga – Pseudo-Kod

```
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            for child-node in EXPAND(STATE[node], problem) do
                fringe ← INSERT(child-node, fringe)
            end
    end
```