Prof.dr.sc. Bojana Dalbelo Bašić

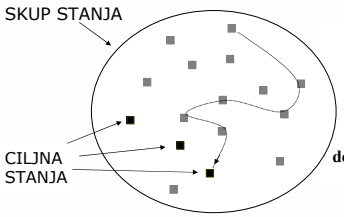
Fakultet elektrotehnike i računarstva
Zavod za elekroniku, mikroelektroniku, računalne i inteligentne sustave

www.zemris.fer.hr/~bojana
bojana.dalbelo@fer.hr

Pretraživanje prostora stanja

PRETRAŽIVANJE PROSTORA STANJA

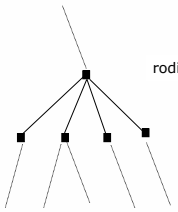
- Pretraživanje je tipična operacija za inteligentne sustave
- Tipičan UI problem:



Rješavanje problema - **Pretraživanje prostora stanja dok se ne dođe do ciljnog stanja**

PRETRAŽIVANJE PROSTORA STANJA

- Skup stanja je organiziran u strukturu **grafa** ili **stabla** pa je pretraživanje pronalaženje puta kroz graf ili stablo
- Pretraživanje - kretanje od čvora do čvora nakon širenja i generiranja susjednih veza



širenje čvora (*engl. expanding the node*) – postupak generiranja sve djece nekog čvora

PODJELA STRATEGIJA PRETRAŽIVANJA

- Prema informacijama koje se koriste tijekom pretraživanja
- Informacije koje se koriste tijekom pretraživanja

1. mogu se odnositi na:
   - cijeli prostor stanja
   - samo na neka stanja
2. mogu biti dostupna:
   - unaprijed
   - samo nakon širenja čvora

PODJELA STRATEGIJA PRETRAŽIVANJA

- U najgorem slučaju raspoloživa informacija je ona u kojoj možemo razlikovati ciljna stanja od ostalih.
- Strategije pretraživanja dijele se u dvije osnovne grupe:

1. **Slijepo pretraživanje** (*engl. blind, uninformed search*)
2. **Usmjereno pretraživanje** (*engl. directed, informed search*)

SLIJEPO PRETRAŽIVANJE

Slijepo pretraživanje jest retraživanje u kojem su jedine raspoložive informacije:

1. početno stanje
2. dozvoljene operacije nad stanjima
3. test na rješenje problema

- Slijepo pretraživanje napreduje sustavno kroz prostor pretražujući čvorove u nekom prethodno definiranom redoslijedu ili birajući ih slučajno

PRETRAŽIVANJE U ŠIRINU (*engl. breadth-first search*)

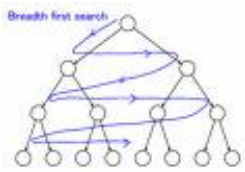
- Pretraživanje u kojem se ispituju svi čvorovi na određenoj razini prije odlaska u slijedeću razinu
- Struktura liste koristi se za čuvanje otvorenih (generiranih) i još ne istraženih čvorova
- Redoslijed kojim se čvorovi pohranjuju u listu za ispitivanje i uklanjanje određuje tip pretraživanja!

PRETRAŽIVANJE U ŠIRINU

Algoritam pretraživanja u širinu

1. Pohrani početni čvor na listu
2. AKO je lista prazna ONDA vrati pogrešku i STANI
3. AKO je prvi element liste ciljni čvor ONDA vrati nađeno uspješno rješenje i STANI
4. **Inače, ukloni prvi čvor na listi, razvi ga i svu njegovu djecu stavi na kraj liste bilo kojim redom**
5. Vrati se na korak 2

PRETRAŽIVANJE U ŠIRINU



Vremenska složenost i prostorna složenost: **O(b^d)**, gdje je **b** faktor grananja, a **d** dubina

Eksponencijalna složenost je najveći nedostatak pretraživanja u širinu.

PRETRAŽIVANJE U DUBINU (*engl. depth-first search*)

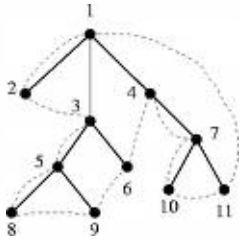
- Pretražuje poniranjem niz stablo što je brže moguće tako da uvijek generira djecu zadnje proširenog čvora dok ne dođe do rješenja (ili se pređe zadanu granicu dubine)
- Algoritam pretraživanja u dubinu razlikuje se od onog u širinu po redoslijedu kojim se otvoreni čvorovi stavljaju na listu – novo stvorena djeca stavljaju se na početak liste (tako da se ona prva ispituju).

PRETRAŽIVANJE U DUBINU

Algoritam pretraživanja u dubinu

1. Pohrani početni čvor na listu
2. AKO je lista prazna ONDA vrati pogrešku i STANI
3. AKO je prvi element liste ciljni čvor ONDA vrati nađeno uspješno rješenje i STANI
4. **Inače, ukloni prvi čvor na listi, razvi ga i svu njegovu djecu stavi na početak liste bilo kojim redoslijedom**
5. Vrati se na korak 2

PRETRAŽIVANJE U DUBINU



PRETRAŽIVANJE U DUBINU

- Vremenska kompleksnost **O(b^d)**
- Prostorna kompleksnost **O(bd)** (čuva se samo put od početnog do tekućeg čvora)
- Pogodniji je od pretraživanja u širinu kada stablo pretraživanja ima puno ciljnih čvorova
- Postoji problem određivanja granične dubine

ITERATIVNO PRETRAŽIVANJE U DUBINU

- Pretraživanje u dubinu koje u prvom koraku ide do dubine 1, a ako rješenje nije nađeno odbacuju se čvorovi i započinje ponovno pretraživanje u drugoj iteraciji do dubine 2
- Ako rješenje i tada nije nađeno započinje ponovno pretraživanje u dubinu do dubine 3 itd. slijedno do neke granične dubine n
- Vremenska složenost **O(b^d)**
- Prostorna složenost **O(bd)**

ITERATIVNO PRETRAŽIVANJE U DUBINU

- Algoritam je u jednoj iteraciji identičan algoritmu za pretraživanje u dubinu. Taj se algoritam prekida kada je postignuta dubina d, ako cilj nije pronađen
- Uklanjaju se svi čvorovi iz reda, d se povećava za jedan i postupak ponovno započinje

DVOSMJERNO PRETRAŽIVANJE

- Moguće je ako problem:

1. ima jedno ciljno stanje koje je dano eksplicitno i
2. sve operacije na čvorovima (stanjima, objektima) imaju inverze

Primjer: igra sa 8 numeriranih polja

- Pretražuje se istodobno u dva smjera:
  - unaprijed počevši od početnog stanja
  - unatrag počevši od cilja

DVOSMJERNO PRETRAŽIVANJE

- Program pohranjuje generirana stanja oba smjera dok se ne nađe zajednički čvor (stanje)
- Sve tri metode pretraživanja mogu se kombinirati u dvosmjerno pretraživanje
- Vremenska i prostorna složenost je **O($b^{d/2}$)** za dvosmjerno iterativno pretraživanje u dubinu

USMJERENO PRETRAŽIVANJE

- Što je više podataka dostupno postupku pretraživanja to je pretraživanje efikasnije
- **Heurističke informacije**
- Informacije o naravi stanja, o cijeni prijelaza iz jednog sanja u drugo, o osobinama cilja itd. mogu se oblikovati u heurističku evaluacijsku funkciju koja zavisi od čvora **n** i od cilja **g**
- **Heuristika** – iskustveno pravilo tj. tehnika prosuđivanja koja može pomoći, ali ne osigurava pronalaženje cilja.
- Heuristike pomažu u smanjenju složenosti problema od eksponencijalne na polinomijalnu.

USMJERENO PRETRAŽIVANJE

Primjer: 8 - slagalica - heuristika može biti izbor poteza koji vodi u stanje koje ima najmanji broj različitih pozicija kvadratića u tekućem i ciljnom stanju

Primjer: problem trgovačkog putnika – jednostavna heuristika je izbor najbližeg neposjećenog čvora. Ne osigurava optimalno rješenje, ali reducira vremensku kompleksnost na $O(n^2)$

METODA USPONA NA VRH

- Metoda uspona na vrh je poput metode pretraživanja u dubinu s tim da se širi onaj čvor koji je najpogodniji prema vrijednosti heurističke funkcije, dok se sve informacije o ostalim čvorovima brišu
- *Primjer:* heuristička funkcija može biti neka mjera udaljenosti od cilja, tada je najpogodniji čvor sa minimalnom vrijednošću
- Važan dobar izbor heurističke funkcije

METODA USPONA NA VRH

Metoda uspona na vrh (*eng. Hill climbing*) - pohlepna metoda

Nedostaci metode uspona na vrh:

- *brežuljak* - **lokalni ekstrem** – slijedni čvorovi (djeca) imaju lošije vrijednosti heurističke funkcije od roditelja – rješenje: povratak unatrag
- *hrbat* – nekoliko susjednih čvorova ima veće vrijednosti nego slijedni čvorovi
- *zaravan* – svi slijedni čvorovi imaju iste vrijednosti, rješenje: slučajan skok

- Napomena: u primjeni se traže minimalne vrijednosti heurističke funkcije, zemljopisne analogije se odnose na max vrijednosti

METODA NAJBOLJEG PRVOG

- Metoda najbolje prvog – (*eng. Best first search*)
- Za razliku od metode uspona na vrh ova metoda zadržava procjene vrijednosti heurističkih funkcija svih prethodno generiranih čvorova i izabire najbolji. Time se izbjegavaju zamke "uspona na vrh"
- Za širenje se izabire vrh između SVIH neproširenih otvorenih čvorova, bez obzira gdje se nalaze u stablu pretraživanja

METODA NAJBOLJEG PRVOG

Algoritam:

1. Pohrani početni čvor u listu
2. AKO je lista prazna ONDA vrati pogrešku i STANI
3. AKO je prvi element liste ciljni čvor ONDA vrati nađeno uspješno rješenje i STANI
4. **Ukloni prvi čvor u listi, razvi ga i izračunaj procjenu udaljenosti od cilja za svako dijete. Stavi svu djecu na listu i uredi sve elemente liste u padajućem redoslijedu**
5. Vrati se na korak 2

METODA NAJBOLJEG PRVOG

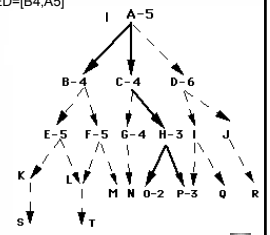
U sljedećem primjeru koristimo dvije liste:

- OPEN - čvorovi koji su stvoreni, ali nisu još prošireni, otvoreni čvorovi
- CLOSED - čvorovi koji su prošireni i čija djeca su na raspolaganju algoritmu pretraživanja, zatvoreni čvorovi

- Lista OPEN složena je po rastućoj vrijednosti funkcije heurističke funkcije
- Proširuje se onaj čvor koji je na vrhu liste, a prošireni čvor se stavlja na listu CLOSED. Djeca se stavljaju na listu OPEN koja se ponovno složi po rastućoj vrijednosti heurističke funkcije

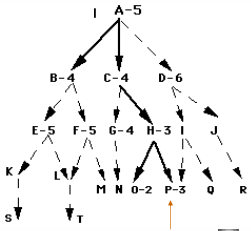
METODA NAJBOLJEG PRVOG

1. OPEN=[A5]; CLOSED=[]
2. evaluate A5; OPEN=[B4,C4,D6]; CLOSED=[A5]
3. evaluate B4; OPEN=[C4,E5,F5,D6]; CLOSED=[B4,A5]
4. evaluate C4; OPEN=[H3,G4,E5,F5,D6]; CLOSED=[C4,B4,A5]
5. evaluate H3; OPEN=[O2,P3,G4,E5,F5,D6]; CLOSED=[H3,C4,B4,A5]
6. evaluate O2; OPEN=[P3,G4,E5,F5,D6]; CLOSED=[O2,H3,C4,B4,A5]
7. evaluate P3; **CILJ!!**



METODA NAJBOLJEG PRVOG

1. Da li bi metoda uspona na vrh pronašla rješenje?
2. Uočite da heuristika nije dobra: stanje O ima manju vrijednost heurističke funkcije nego cilj P. Ova metoda se može oporaviti.



CILJ

ALGORITAM A*

- A* jest *poseban slučaj* algoritma *Najbolji prvi* koji uključuje postupak računanja udaljenosti čvora **n** od cilja!
- *Ideja*: za svaki čvor, koji je na putu od početnog do ciljnog čvora, algoritam stvara sve slijedne čvorove (djecu) i računa procjenu udaljenosti od početnog do ciljnog čvora kroz sve stvorene čvorove
- Izabire onaj slijedni čvor (dijete) za koji je ta procjena udaljenosti najmanja. Tada se stvaraju djeca odabranog čvora, računaju se procjene udaljenosti za putove kroz njih i postupak se nastavlja dok se ne pronađe rješenje

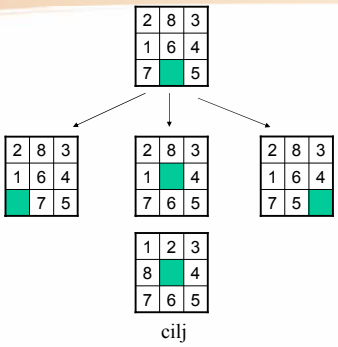
ALGORITAM A*

- Procjena udaljenosti od cilja:

$$f^*(n) = g^*(n) + h^*(n)$$

- g*(n) – procjena udaljenosti (cijene) od početnog čvora do čvora n
- h*(n) - procjena udaljenosti (cijene) od čvora n do ciljnog čvora
- Znak * označava procjenu stvarnih vrijednost funkcija f, g, h. Kada se radi o pretraživanju stabla (a ne grafa), vrijedi da je g*=g

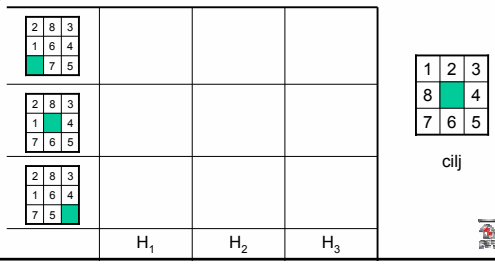
ALGORITAM A*



Početno stanje, prvi potezi, i rješenje 8-slagalice

ALGORITAM A*

- Moguće su različite heuristike:
- H_1 – broj pločica izvan svog mjesta
- H_2 – suma udaljenosti pločica izvan svog mjesta
- H_3 – broj susjednih pločica zamijenjenih mjesta pomnožen s dva



ALGORITAM A*

- Moguće su različite heuristike:
- H_1 – broj pločica izvan svog mjesta
- H_2 – suma udaljenosti pločica izvan svog mjesta
- H_3 – broj susjednih pločica zamijenjenih mjesta pomnožen s dva

2 8 3 / 1 6 4 / _ 7 5	5	6	0
2 8 3 / 1 _ 4 / 7 6 5	3	4	0
2 8 3 / 1 6 4 / 7 5 _	5	6	0
	H_1	H_2	H_3

1 2 3 / 8 _ 4 / 7 6 5

cilj

ALGORITAM A*

Algoritam A* koristi dvije liste:

- OPEN - čvorovi koji su stvoreni, ali nisu još prošireni
- CLOSED - čvorovi koji su prošireni i čija djeca su na raspolaganju algoritmu pretraživanja

- Lista OPEN složena je po rastućoj vrijednosti funkcije f*
- Proširuje se onaj čvor koji je na vrhu liste (najmanji f*), a prošireni čvor se stavlja na listu CLOSED. Djeca se stavljaju na listu OPEN koja se ponovno složi po rastućoj vrijednosti funkcije f*

ALGORITAM A*

1. Stavi početni čvor na listu OPEN
2. Ako je lista OPEN prazna onda se zaustavi i vrati pogrešku
3. Ukloni sa liste OPEN čvor **n** koji ima najmanju vrijednost f*(**n**). Ako je čvor ciljni vrati uspješan završetak

   Inače,
4. Proširi čvor **n**, stvarajući svu njegovu djecu **n'.** Za svako dijete n' izračunaj vrijednost f(**n'**), a čvor **n** stavi na listu CLOSED

   Za svako dijete **n'**, koje nije na listi OPEN ili CLOSED učini:
   - pridijeli mu izračunatu vrijednost f(**n'**)
   - pridijeli mu pokazivač unazad (engl. *back-pointer*) na čvor **n**
   - stavi ga na listu OPEN

ALGORITAM A*

(nastavak)

5. Za svako dijete **n'**, koje je na listi OPEN ili CLOSED učini:
   - pridijeli mu manju od vrijednosti f*(**n'**) izračunatu u ovom koraku ili prije
   - ako je **n'** bio na listi CLOSED i vrijednost f*(**n'**) mu je promijenjena (smanjena), ukloni ga i stavi ga na listu OPEN te ažuriraj njegov pokazivač unatrag
6. Vrati se na korak 2

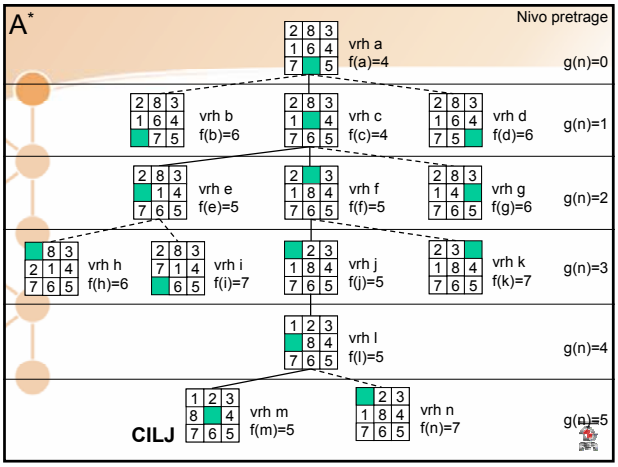
ALGORITAM A*

Primjer 8-slagalica

$$f^*(m)=g(m)+h^*(m)$$

Razina pretrage

Koliko kvadratića nije na svom mjestu



Primjer: 8-slagalica

1. OPEN=[a4]; CLOSED=[]
2. OPEN=[c4,b6,d6]; CLOSED=[a4]
3. OPEN=[e5,f5,g6,b6,d6]; CLOSED=[a4,c4]
4. OPEN=[f5,h6,g6,b6,d6,i7]; CLOSED=[a4,c4,e5]
5. OPEN=[j5,h6,g6,b6,d6,k7,i7]; CLOSED=[a4,c4,e5,f5]
6. OPEN=[l5,h6,g6,b6,d6,k7,i7]; CLOSED=[a4,c4,e5,f5,j5]
7. OPEN=[m5,h6,g6,b6,d6,n7,k7,i7];
   CLOSED=[a4,c4,e5,f5,j5,l5]
8. RJEŠENJE , **m je cilj**

SVOJSTVA HEURISTIČKIH ALGORITAMA ZA PRETRAŽIVANJE

- **Kako procijenjujemo ponašanja heurističkih algoritama?**

Može nas zanimati sposobnost algoritma da pronađe optimalan put (planiranje puta autonomnih robota u opasnoj okolini)

- **Svojstvo potpunosti** (*engl. completeness condition*) - Algoritam je potpun ako pronalazi uvijek kada postoji rješenje
- **Svojstvo dosežljivosti** (*engl. admissibility condition*) - Algoritam je dosežljiv ako osigurava nalaženje optimalnog rješenja uvijek kada rješenje postoji
- **Primjer: A***

SVOJSTVA HEURISTIČKIH ALGORITAMA ZA PRETRAŽIVANJE

- **Svojstvo dominacije** (*engl. dominance property*) – Neka su A_1^* i A_2^* dva dosežljiva algoritma sa heurističkim funkcijama h_1^* i h_2^*.

  Algoritam A_2^* je **obavješteniji** od algoritma A_1^* ako je $h_1^*(n) < h_2^*(n)$ za svaki čvor n. Još se kaže da A_2^* **dominira** nad A_1^*

  - A* za kojeg je h*=0 postaje algoritam ...

SVOJSTVA HEURISTIČKIH ALGORITAMA ZA PRETRAŽIVANJE

- **Svojstvo dominacije** (*engl. dominance property*) – Neka su A_1^* i A_2^* dva dosežljiva algoritma sa heurističkim funkcijama h_1^* i h_2^*.

  Algoritam A_2^* je **obavješteniji** od algoritma A_1^* ako je $h_1^*(n) < h_2^*(n)$ za svaki čvor n. Još se kaže da A_2^* **dominira** nad A_1^*

  Primjer

  Algoritam "pretraživanja u širinu" može se opisati kao A* za koji je h*(n) = 0 tj. f*(n) = g*(n) + 0, pa je svaki A* algoritam obavješteniji od "pretraživanja u širinu"

ALGORITAM A*

Primjer:

$h_1(n)=0$ za svaki n

$h_2(n)$ – broj pločica koje nisu na mjestu

h*

$$h_1 \leqq h_2 \leq h^*$$

•h_1 i h_2 nalaze optimalne puteve, ali h_2 evaluira manje stanja u postupku traženja rješenja

•h_2 je informiranija od h_1

ALGORITAM A*

Primjer: heuristika koja računa sumu udaljenosti od mjesta na kojem se pločica treba nalaziti je informiranija od heuristike broja pločica koje nisu na mjestu

Općenito: Informiraniji algoritam pretražuje manji prostor!

Oprez!
Uzeti u obzir složenost izračunavanja heuristike!

Primjer: šah

SVOJSTVA HEURISTIČKIH ALGORITAMA ZA PRETRAŽIVANJE

Svojstvo optimalnosti (*engl. optimality condition*)
Algoritam je optimalan na klasi algoritama ako dominira svim elementima klase

Vrijedi:

- **Algoritam A* je potpun i dosežljiv**

Efikasnost A* zavisi od toga koliko dobro h* aproksimira h i koja je "cijena" računanja funkcije f*

Zadatak:

Odredite i obrazložite sva spomenuta svojstva svih do sada spomenutih algoritama pretraživanja prostora stanja.