VEŠTAČKA INTELIGENCIJA 2020/21

REŠAVANJE PROBLEMA I TRAŽENJE

Sadržaj

I deo

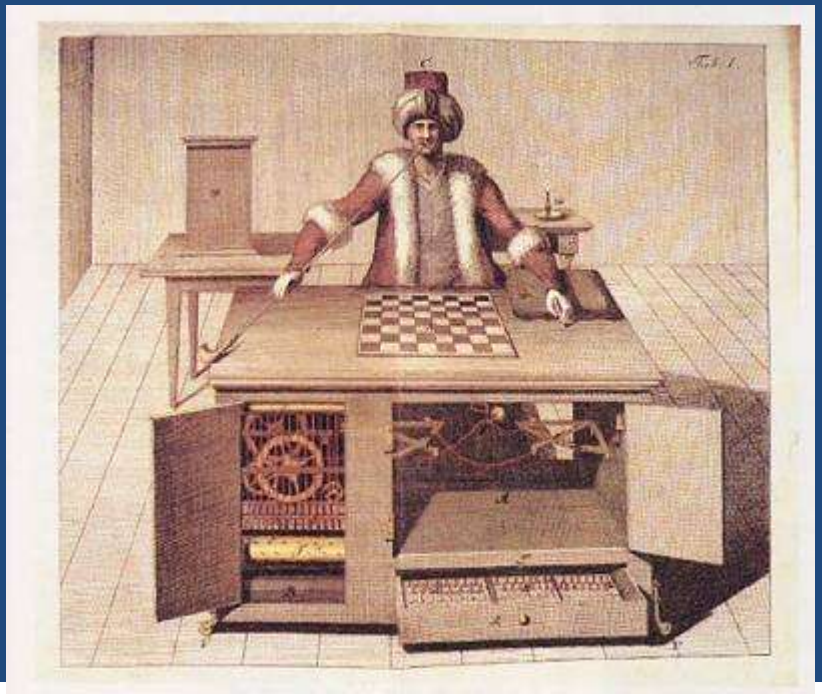
- Formulacija problema
- Slepi algoritmi za traženje

II deo

- Informisani algoritmi za traženje
- Local search algoritmi

III deo

- Algoritmi za igre



Prof. dr Leonid Stoimenov

Rešavanje problema – šta je to?

- Mi želimo da:
  - Automatizujemo rešavanje problema
- Treba nam:
  - Reprezentacija problema
  - Algoritmi koji koriste neku srategiju za rešavanje problema definisanog izabranom reprezentacijom

Tipovi problema

- **Determinističko, potpuno dostupno okruženje → tzv. *single-state* problem**
  - **Agent zna tačno u kom je stanju, rešenje je sekvenca akcija**

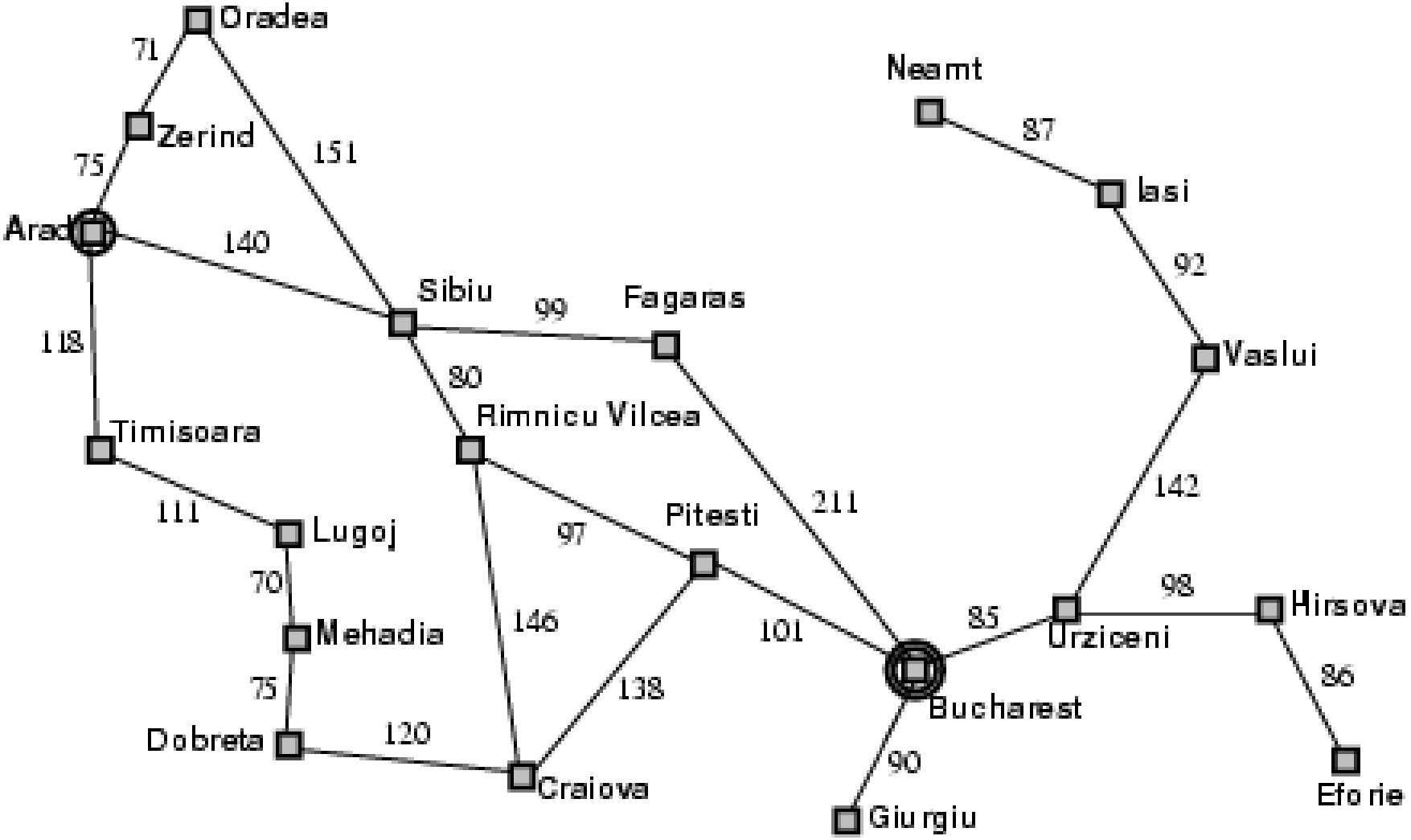
- Non-observable → sensorless problem (conformant problem)
  - Agent nema ideju gde je; rešenje je sekvenca
- Nondeterministic and/or partially observable → contingency problem
  - Percepcijom se obezbeđuju agentu nove informacije o tekućem stanju
- Unknown state space → exploration problem

Agent koji rešava problem (problem-solving agent)

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
    static: seq, an action sequence, initially empty
            state, some description of the current world state
            goal, a goal, initially null
            problem, a problem formulation

    state ← UPDATE-STATE(state, percept)
    if seq is empty then do
        goal ← FORMULATE-GOAL(state)
        problem ← FORMULATE-PROBLEM(state, goal)
        seq ← SEARCH(problem)
    action ← FIRST(seq)
    seq ← REST(seq)
    return action
```

Primer: Rumunija
Kako naći put od Arada do Bukurešta?



Primer: Rumunija

Kako naći put od Arada do Bukurešta?

- Otišli ste na odmor u Rumuniju; trenutno ste u gradu Arad.
- Sutradan imate let iz grada Bucharest
- Kako stići od Arad-a do Bucharest-a?
- Formulište cilj:
  - Stići u Bucharest
- Formulišite problem:
  - stanja: različiti gradovi
  - akcije: vožnja između gradova
- Traženje rešenja
  - Sekvenca gradova, napr. Arad, Sibiu, Fagaras, Bucharest

Selekcija (izbor) prostora stanja (Formulacija problema)

- Realni svet je jako složen
  - → Prostor stanja mora da bude neka njegova apstrakcija za rešavanje problema
- (Apstraktno) **stanje** = skup realnih stanja
- (Apstraktna) **akcija** = kompleksna kombinacija realnih akcija
  - napr., "Arad → Zerind" predstavlja kompleksni skup mogućih ruta, prevoznika, odmorišta i sl.
- Da bi se garantovala realizabilnost, neko realno stanje "in Arad“ mora da odgovara nekom real state "in Zerind"
- (Apstraktno) **rešenje** =
  - Skup realnih puteva koji su rešenje u realnom svetu
- Svaka apstraktna akcija treba da bude jednostavija u odnosu na originalni problem

(apstraktna) Reprezentacija problema

- Generalno, treba nam:
  - **Prostor stanja:**

    problem je podeljen na skup koraka (**akcija**) koji vode od **inicijalnog** stanja do **ciljnog stanja**
- **Stanje** je reprezentacija elemenata problema u datom trenutku.
  - Problem se difiniše preko elemenata problema i njihovih međusobnih veza.
  - Za svaku instancu rezolucije problema, ti elementi imaju svoje deskriptore (kako ih izabrati?) i relacije.

Stanja

- **Stanje** je reprezentacija elemenata problema u **datom trenutku.**

- Dva specijalna stanja se definišu:
  - **Inicijalno (početno) stanje** (početak problema)
  - **Finalno (ciljno) stanje** (očekivano ciljno stanje)

Modifikacija stanja: funkcije sledbenika

- Funkcija sledbenka je potrebna da bi se obezbedila **promena stanja**.
- Funkcija sledbenka je **opis mogućih akcija**, odnosno **skup operacija**.
- Radi se o funkciji koja vrši transformaciju reprezentacije stanja, odnosno promenu jednog stanja u drugo.
- Funkcija sledbenka definiše veze između stanja.
- Reprezentacija funkcije sledbenka :
  - Uslovi primenljivosti
  - Funkcija transformacije

Prostor stanja

- **Prostor stanja** je skup svih stanja koji se mogu dostići iz polaznog stanja.
- Prostor stanja ima **formu grafa** u kome su čvorovi stanja, a potezi akcije odnosno funkcije sledbenika.
- **Put** u prostoru stanja je sekvenca stanja povezanih sekvencom akcija.
- **Rešenje problema** je **jedan put** u grafu/prostoru stanja.

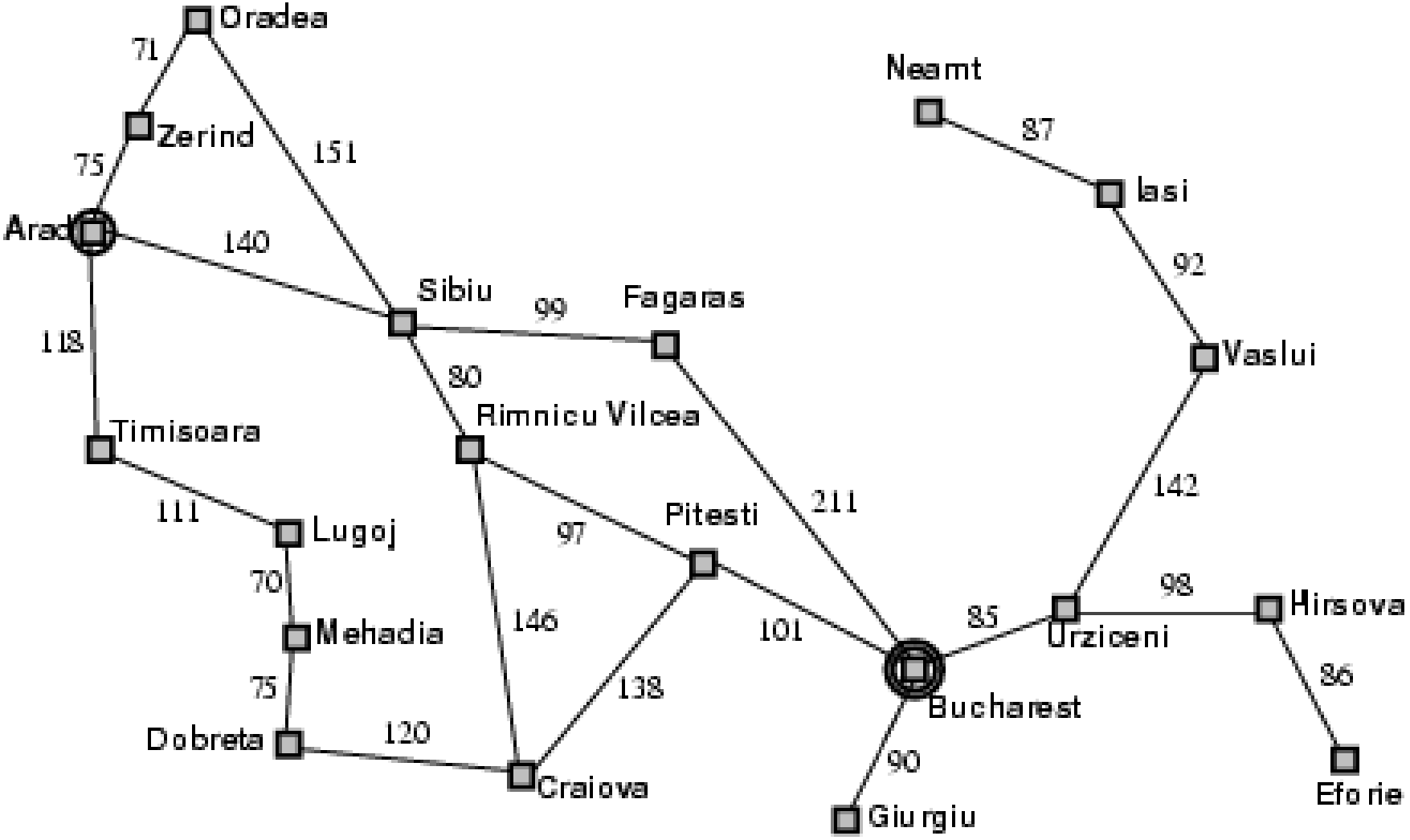
Rešenje problema

- **Rešenje** u prostoru stanja je put počev od početnog stanja do ciljnog stanja, ili ponekad, samo ciljno stanje.
- **Put/cena rešenja**: funkcija koja dodeljuje numeričku vrednost svakom putu, i odnosi se na cenu primene operatora na stanja.
- Rešenje se kvalitativno meri cenom puta, tako da **optimalno rešenje** se odnosi na put sa najnižom cenom.
- Rešenja: bilo koje, optimalno, sva rešenja. Cena zavisi od problema i tipa rešenja koje se očekuje.

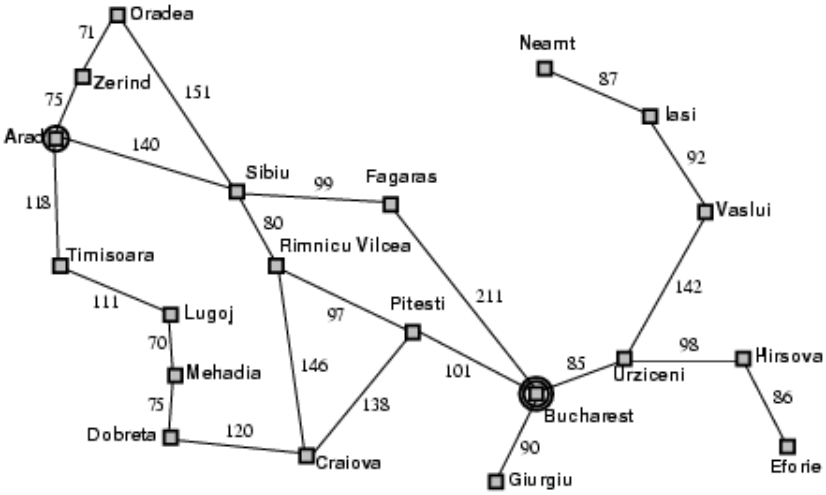
Komponente za opis problema

- Komponente:
  - Prostor stanja (definisan eksplicitno ili implicitno)
  - Inicijalno stanje
  - Ciljno stanje (ili uslovi koje treba zadovoljiti)
  - Dostupne/dozvoljene akcije (operatori koji menjaju stanja)
  - Ograničenja (napr. cena)
  - Elementi koji se odnose na znanje o domenu koje je relevantno za konkretan problem
  - Tip rešena:
    - Sekvenca operatora ili ciljno stanje
    - Neko, optimalno (potrebna definicija cene puta), sva

Primer: Rumunija
Kako naći put od Arada do Bukurešta?



Formulacija problema



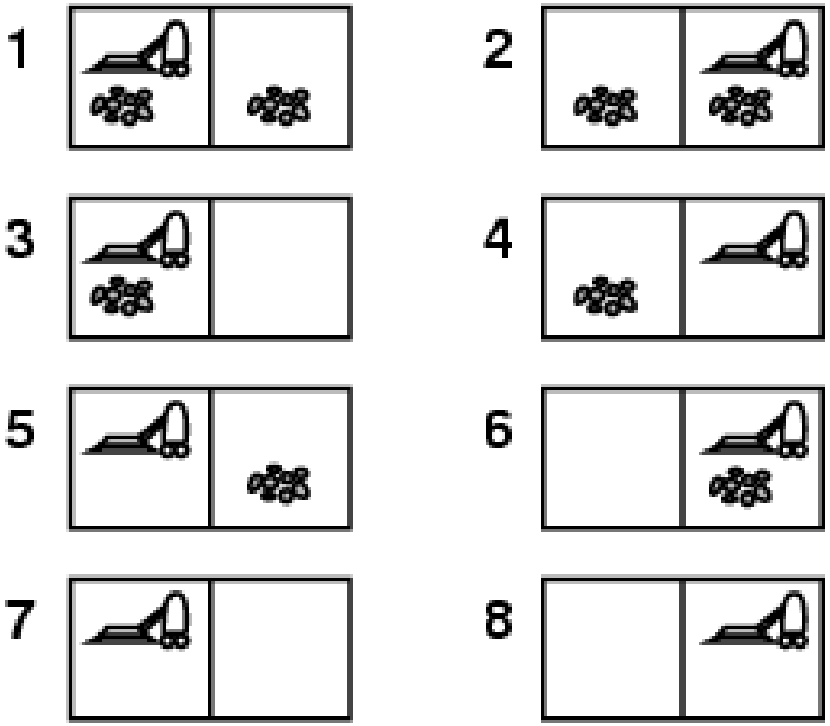
Problem se definiše preko:

- inicijalnog stanja napr., "Arad"
- akcija ili funkcije sledbenika $S(x)$ = skup parova akcija–stanje
  - Primer: $S(Arad) = \{<Arad \rightarrow Zerind, Zerind>, \ldots\}$
- test na ciljno stanje,

  primer: x = "Bucharest",...
- cena puta
  - Primer: zbir rastojanja, broj izvršenih akcija itd.
  - $c(x,a,y)$ je cena koraka, uz pretpostavku ≥ 0
- **rešenje** je sekvenca akcija počev od incijalnog stanja do ciljnog stanja

Primer: vacuum world
Moguća stanja i operatori

Stanja (formani opis?):



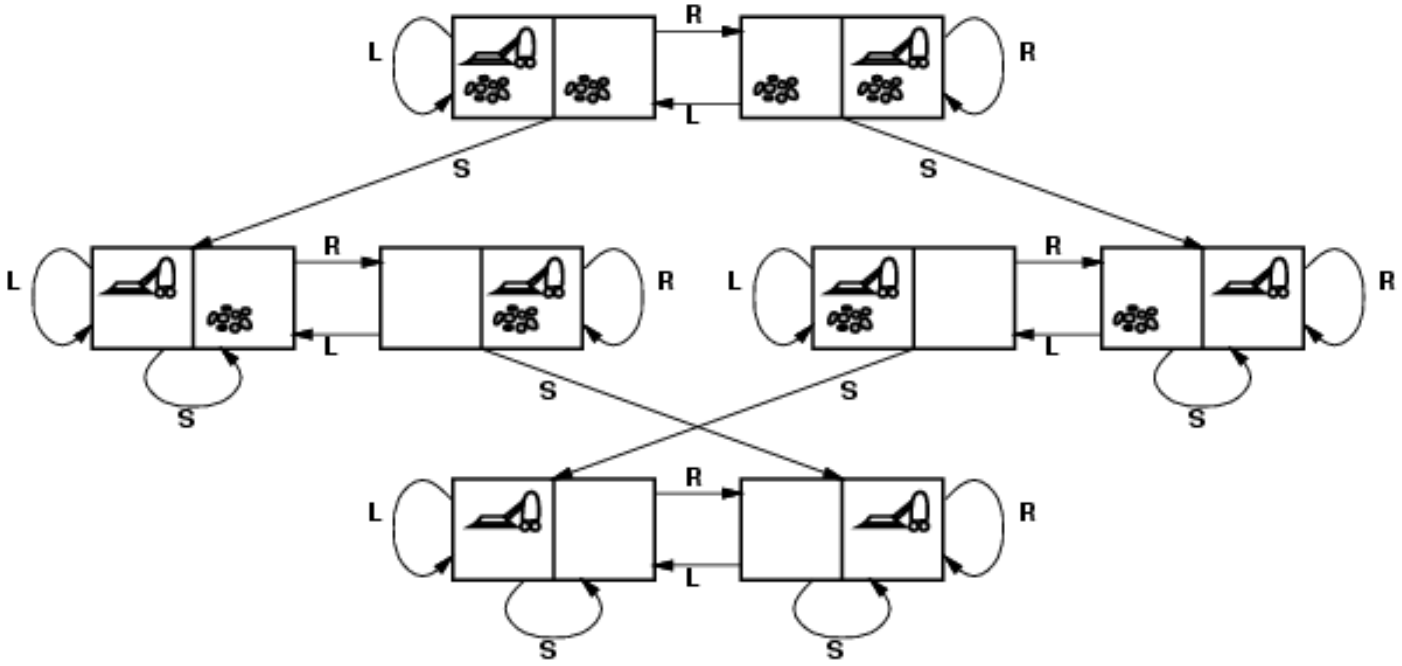
Operatori:

L – LEVO

D – DESNO

U – USISAJ

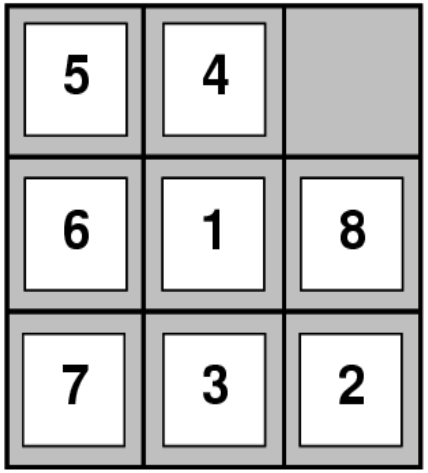
Primer: vacuum world
Prostor traženja

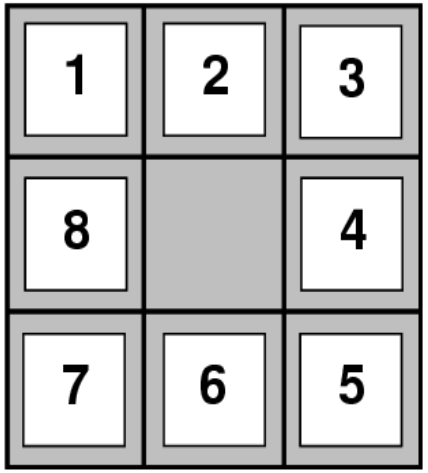


Stanja? *integer*: smeće i lokacija
Poč. stanje? Bilo koje, od 1 do 8
Akcije? ***L****evo*, ***D****esno*, ***U****sisaj*
Test ciljnog stanja? *Čisto,* tj nema smeća na svim lokacijama
Cena puta? 1 po akciji

Primer: 8-puzzle, formulacija 1

- Prostor stanja
  - Svako stanje je polje 3x3 sa brojevima
  - Lokacija pločica
- Početno stanje: stanje 1
- Ciljno stanje: stanje 2
- Operatori:
  - levo, desno, gore, dole, koje se primenjuju na bilo koji element polja (ili NIL)
- Algoritam vraća niz operatora (put)
- Cena: 1 po potezu



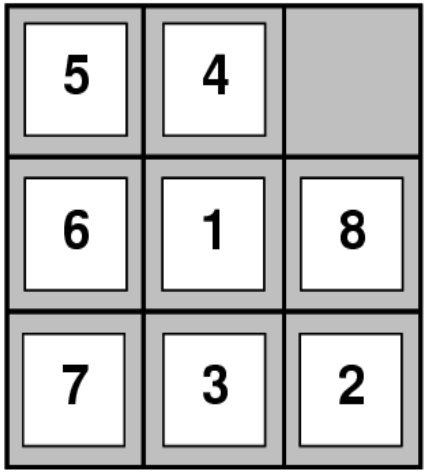


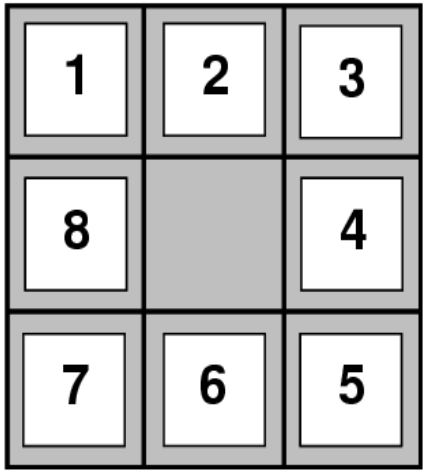
Problemi sa formulacijom 1

- Koliko operatora moze da se primeni na početno stanje?
- Kako se odredjuju sledbenici početnog stanja?
- Bolje je ako se operatori mogu primeniti u većini slučajeva

8-puzzle: formulacija 2

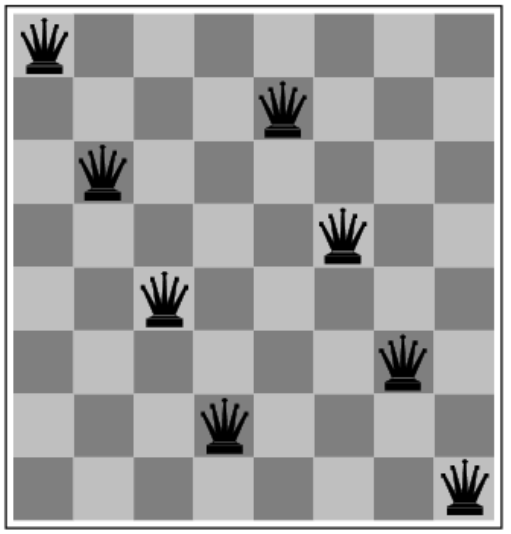
- Isti operatori, ali se primenjuju na **praznu poziciju**
- Operatori tada ne zahtevaju argumente
- Algoritam i dalje vraća putanju





Problem 8 kraljica: formulacija 1

- Prostor stanja:
  - polje 8x8, svaki element može sadržati <= n kraljica, pozicija otvoren/napadnut.
- Početno: prazna tabla
- Ciljno stanje: raspoređene kraljice
- Operatori:
  - 8 operatora (po jedan za svaki red)
  - Kraljica se smesta u prvo slobodno (nenapadnuto) polje



Problem 8 kraljica: formulacija 2

- Počni sa slučajno razmeštenim kraljicama
- Razmestati kraljice dok nema konflikata
- Stanje: vektor od 8 elemenata (vrste)
- Operatori: pomeraju napadnute kraljice u drugi red u istoj koloni

Tipični problemi

- traženje puta (route finding)
- Najkraci put koji obilazi zadata mesta (travelling salesperson)
- VLSI layout
- Navigacija za robote
- Planiranje letova za avio kompaniju
- ...

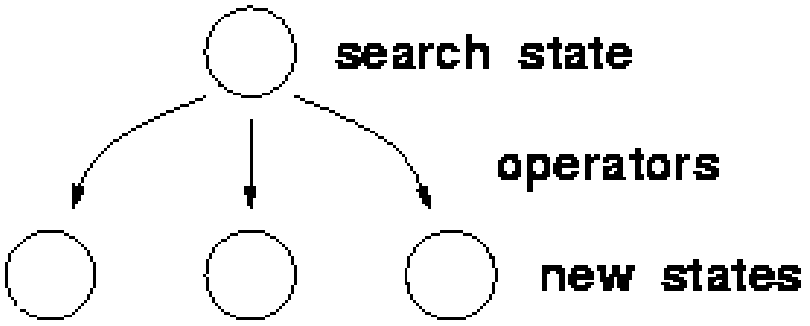
Rešavanje problema traženjem

Stablo traženja

Algoritmi za traženje

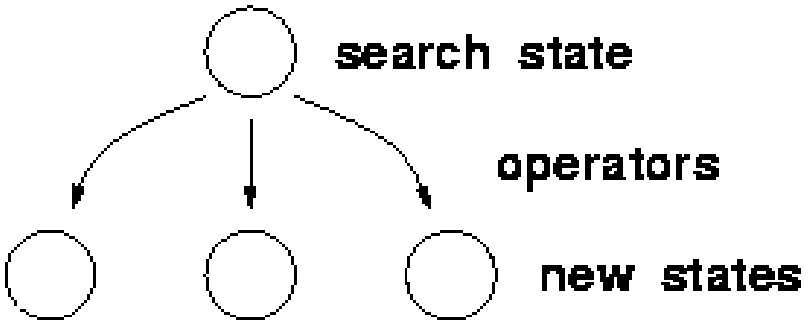
Rešavanje problema kao traženje (pregled)

- ***Stanje*** – diskretno stanje «sveta»
- ***Prostor stanja***: - skup svih stanja problema
- ***Početno stanje***
- ***Ciljno stanje*** (ili stanja!!)
- ***Operatori***: - prelaz iz stanja u stanje; funkcije koje odredjuju sledeće stanje na osnovu tekućeg stanja (ili NIL)



Rešavanje problema kao traženje (2)

- **Sledbenici** (potomci): sva stanja u koja se može preći iz tekućeg stanja
- **Traženje sledbenika:** primena svih operatora na tekuće stanje da bi se dobili potomci
- ***Uslovi/Test za ciljno stanje*:** predikat koji vraća T ili F za zadato stanje



Rešavanje problema kao traženje (3)

- ***Cena puta*** (opciono): suma cena individualnih operatora
- **Heuristika** – informacija o tome zbog čega je izbor nekog čvora (stanja) bolji od izbora ostalih

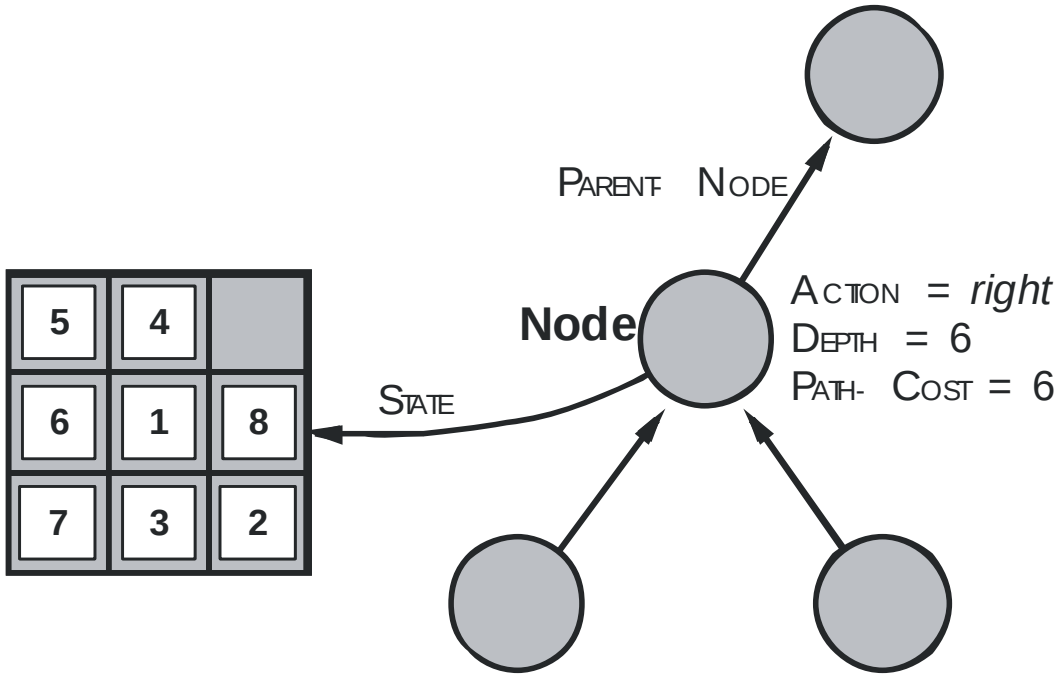
- **Rešavanje problema** – odrediti niz operatora koji prevodi sistem iz početnog stanja u ciljno stanje; put do cilja (**rešenje problema**) je niz čvorova u grafu koje treba obići da bi se došlo do cilja.

Implementacija:
Stanja / Čvorovi u prostoru stanja

Problem: izbor (apstraktnog) stanja

Stanje je (reprezentacija za) fizička(u) konfiguracija(u)

Čvor je struktura podataka koja je deo stabla tražanja, uključuje stanje, roditeljski čvor, akciju, cenu puta $g(x)$, dubinu



Traženje

- Polazno stanje
- Operatori
- Sledbenici
- Krajnje stanje (cilj)
- Test
- Cena puta

Algoritam za traženje

Obilazak prostora stanja

•Pronalazi jednu putanju (ili sve putanje) od početnog do ciljnog stanja

• Izračunava cenu primene niza operatora (put) koji vodi od početnog stanja to krajnjeg stanja

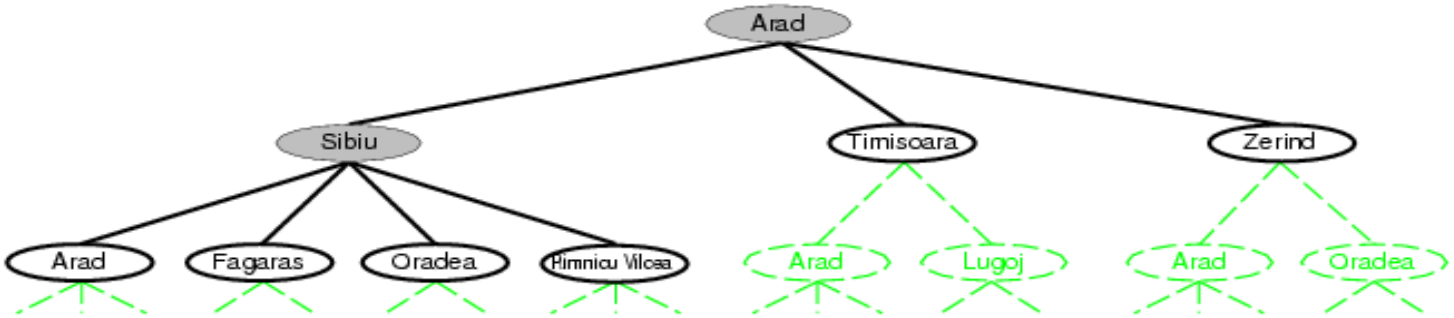
• Neki algoritmi nalaze put sa minimalnom cenom

Stablo traženja

- Osnovna ideja:
  - *offline*, simulirani obilazak prostora stanja generisanjem sledbenika za već posećena stanja
- Stablo traženja je efikasan način da se predstavi kako algoritam traženja ispituje prostor traženja.
- Dinamički se kreira, počev od početnog stanja

Važno: prostor traženja i stablo traženja se razlikuju!

Primer stabla traženja



function TREE-SEARCH(*problem, strategy*) **returns** a solution, or failure

initialize the search tree using the initial state of *problem*

loop do

if there are no candidates for expansion **then return** failure

choose a leaf node for expansion according to *strategy*

if the node contains a goal state **then return** the corresponding solution

else expand the node and add the resulting nodes to the search tree

Strategije

- **Strategija** je definisana izborom čvorova za ekspanziju
- Evaluacija strategija
  - **Kompletnost**: da li nalazi rešenje?
  - **Cena** – vreme i memorija
  - **Optimalnost** – da li nalazi rešenje koje najmanje košta
- Cena tj vremenska i prostorna (memorijska) komplekstnost se mogu meriti u zavisnosti od:
  - ***b***: maksimalne vrednosti faktora grananja stabla traženja
  - ***d***: dubine najjeftinijeg rešenja
  - ***m***: maksimalna dubina prostora traženja (može biti ∞)

Implementacija (generalizacija): Opšti algoritam traženja

```
function TREE-SEARCH( problem, fringe) returns 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 SOLUTION(node)
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

```
function EXPAND( node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN[problem](STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```

Osnovni algoritmi za traženje
(u zavisnosti od strategije)

- **Neinformisani (slepi) algoritmi**

  (depth-first search, breadth-first search, …

  Ne koriste nikakvo znanje o domenu

- **Informisani (heuristicki) algoritmi** (A*)

  Koriste heuristike da pronađu najbolje rešenje

Neinformisani algoritmi traženja

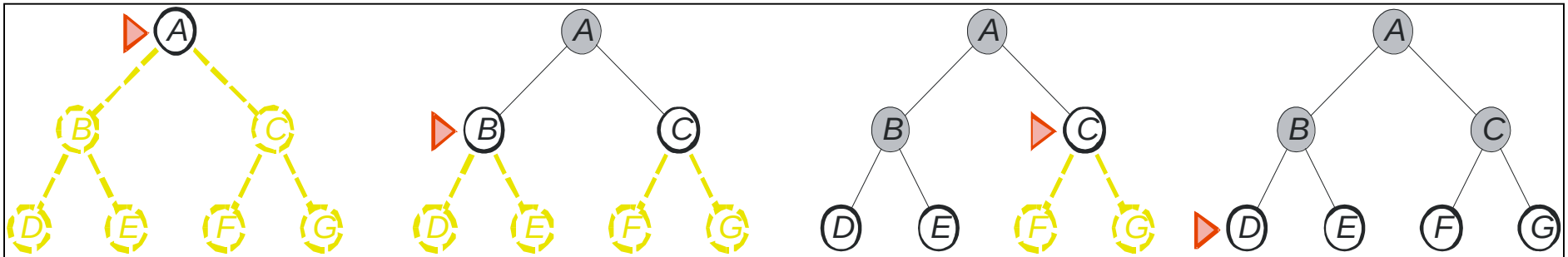
- **Breadth-first search** – traženje po širini
- **Uniform-cost search** – traženje sa uniformnom cenom
- **Depth-first search** – traženje po dubini
- **Depth-limited search** – ograničavanje traženja po dubini
- **Iterative deepening search**

Traženje po širini

- *Breadth-first search*
- Čvorovi se obilaze s leva udesno
- Da bi se izbegle petlje, ne generisu se sledbenici koji su jednaki roditelju tekuceg čvora (postoje i drugi načini da se izbegnu petlje)
- Nalazi najplići cilj
- Kompletan i optimalan ako je cena puta neopadajuca funkcija dubine čvora
- **Implementacija: red (FIFO)**

Algoritam traženja po širini

1. Formirati listu čvorova koja inicijalno sadrži samo startni čvor.
2. Dok se lista čvorova ne isprazni ili se ne dođe do ciljnog čvora, proveriti da li je prvi element liste ciljni čvor
   a) Ako je prvi element liste ciljni čvor, ne raditi ništa.
   b) Ako prvi element liste nije ciljni čvor, ukloniti ga iz liste i dodati sve njegove sledbenike iz stabla pretrage (ako ih ima i ako nisu već posećeni) na kraj liste.
3. Ako je pronađen ciljni čvor, pretraga je uspešno završena; u suprotnom pretraga je neuspešna.



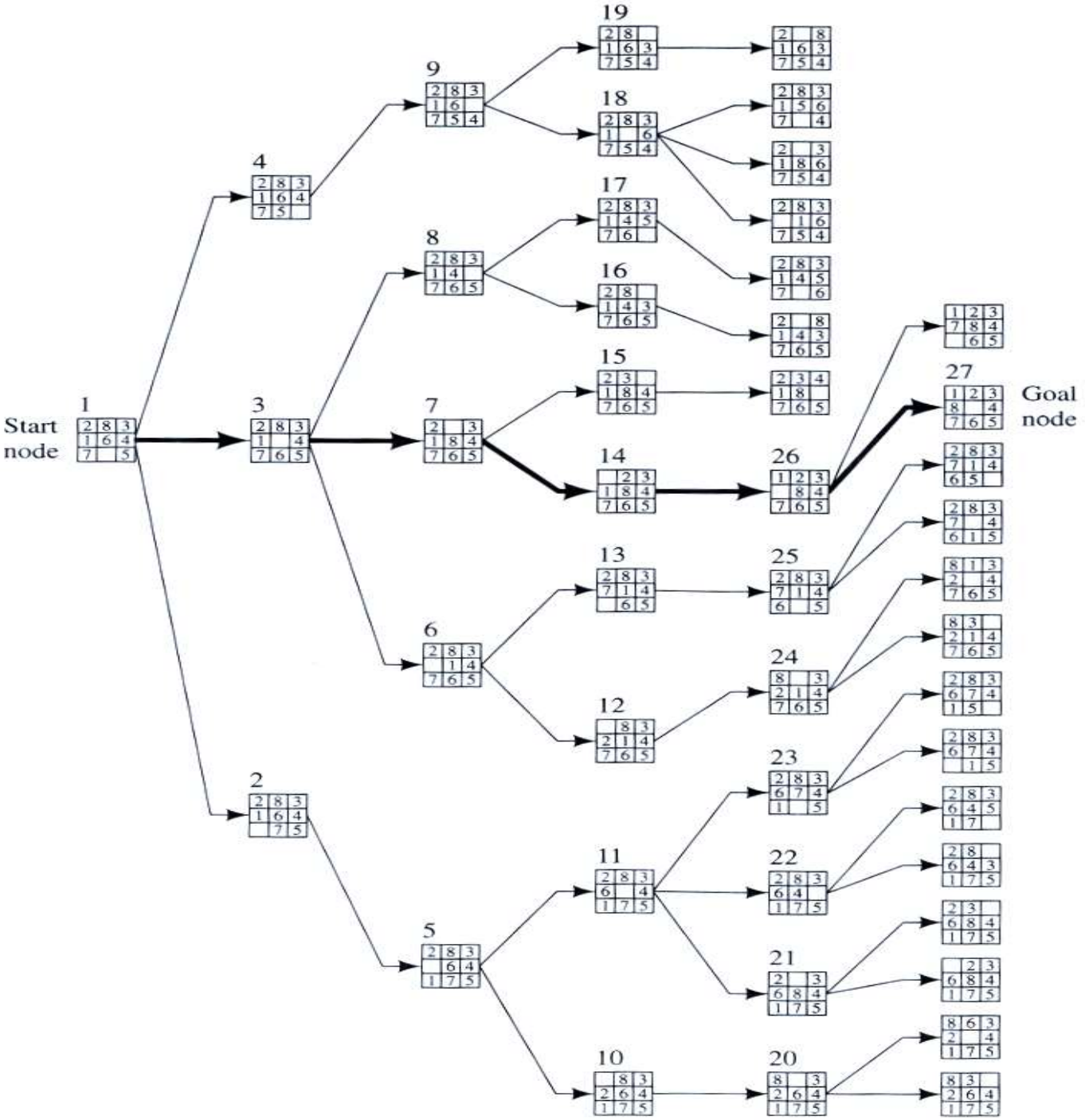
Osobine BFS

- Kompletnost? Da (ako je b konačno)
- Vreme? $1+b+b^2+b^3+\ldots+b^d+b(b^d-1) = O(b^{d+1})$
- Prostor? $O(b^{d+1})$ (čuva svaki čvor u memoriji)
- Optimalnost? Da (ako je cena = 1 po koraku)
- **Prostor** – problem (u odnosu na vreme)

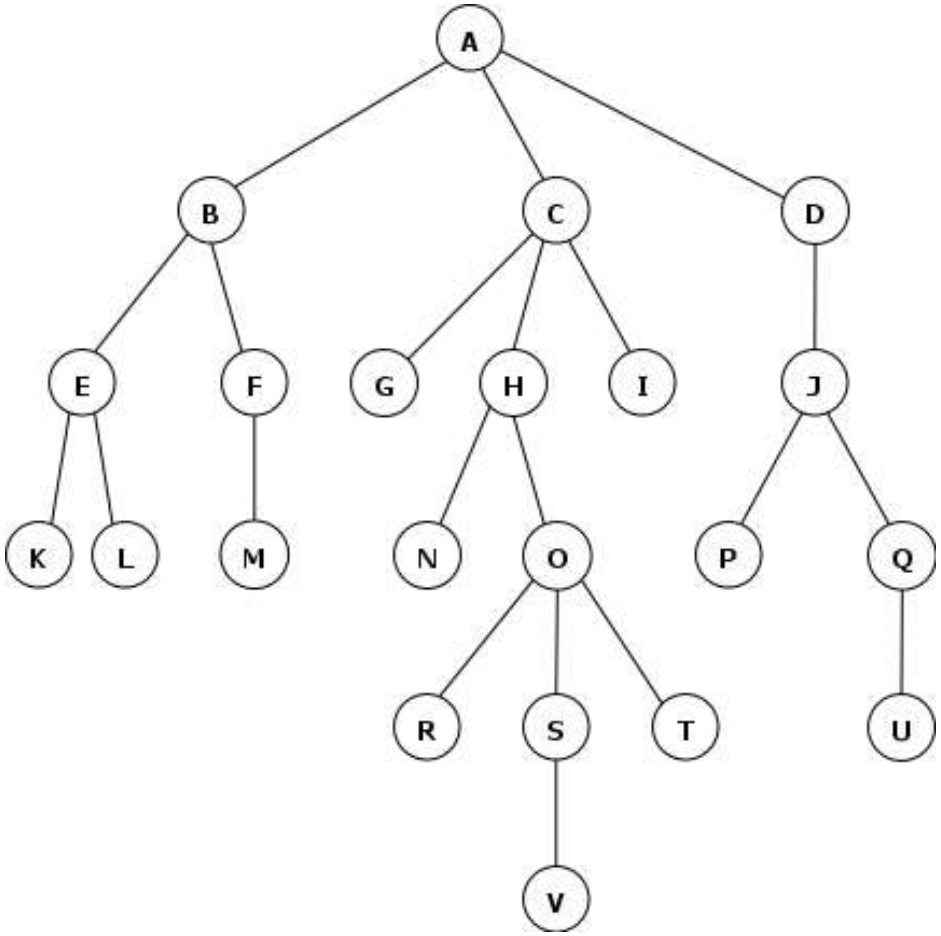
Traženje po širini – pregled

- Prednosti:
  - Uvek nalazi cilj (rešenje)
  - Dobar za plitka stabla
- Nedostaci:
  - Veliki zahtevi za memorijom
  - Može biti spor za stabla sa velikom dubinom

Primer BFS


Start node
Goal node
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

Primer: traženje po širini
Cilj: M, V ili J



- A
- B C D
- C D E F
- D E F G H I
- E F G H I J
- F G H I J K L
- G H I J K L M
- H I J K L M
- I J K L M N O
- J K L M N O
- K L M N O P Q
- L M N O P Q
- M N O P Q
- N O P Q
- O P Q
- P Q R S T
- Q R S T
- R S T U
- S T U
- T U V
- U V
- V

Varijante:
Traženje sa uniformnom cenom

- Uniform cost search
- Modifikacija traženja po širini
- Iz reda se bira čvor sa najnižom cenom g(n)
- Nalazi optimalno rešenje ako cena puta nikada ne opada sa dubinom
- **Implementacija:**

red(**FIFO**) uređen po ceni puta

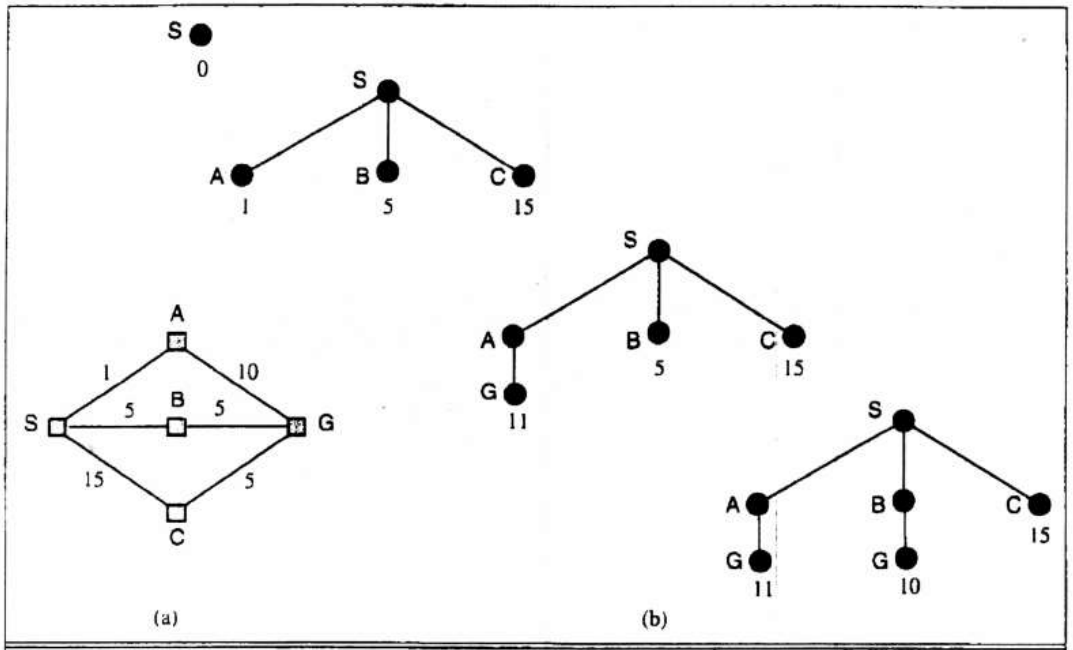
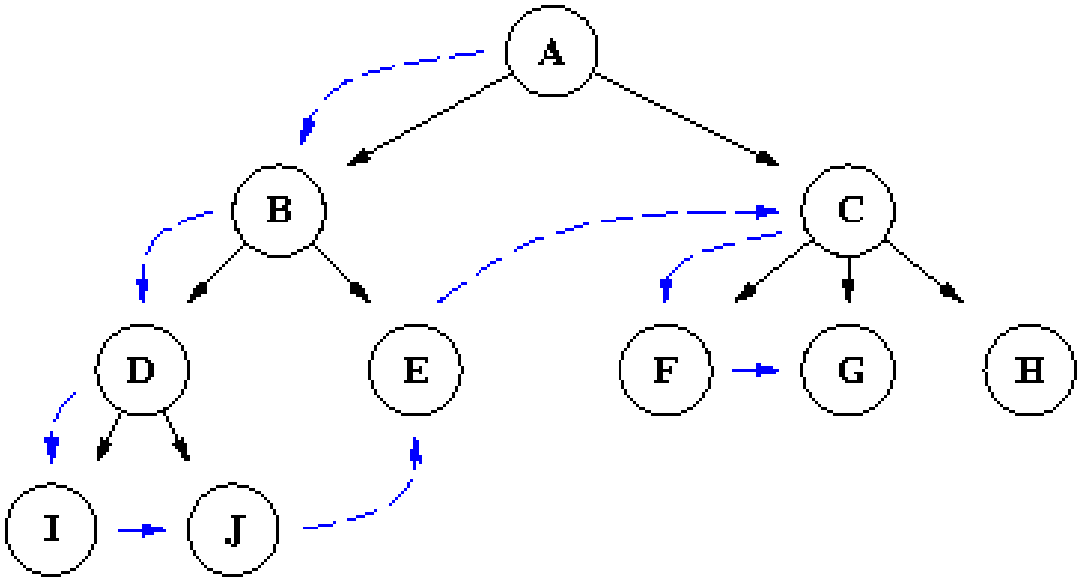


Figure 3.13 A route-finding problem. (a) The state space, showing the cost for each operator. (b) Progression of the search. Each node is labelled with $g(n)$. At the next step, the goal node with $g = 10$ will be selected.

Traženje po dubini

- Depth-first search
- Čvorovi se obilaze s leva udesno
- **Implementacija: magacin(LIFO)**



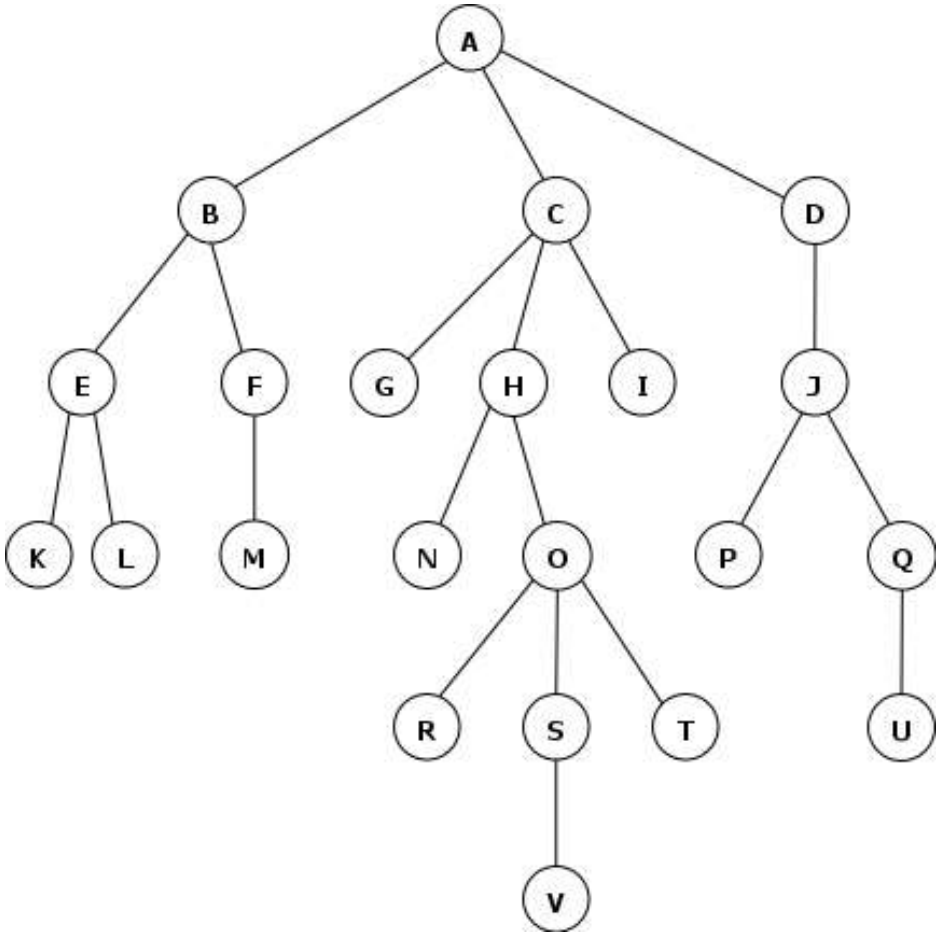
Traženje po dubini - osobine

- Kompletnost? Ne: ne nalazi rešenje u beskonačnim prostorima stanja (infinite-depth spaces), kod stanja sa petljama
  - Modifikuj ili izbegavaj stanja koja se ponavljaju!
  - → kompletan sa konačnim stanjima
- Vreme? $O(b^m)$: jako loše ako je m mnogo veće od d
  - ali ako plitko, može da bude značajno brži od breadth-first
- Prostor? $O(bm)$, tj, linearno!
- Optimalnost? Ne

Traženje po dubini - pregled

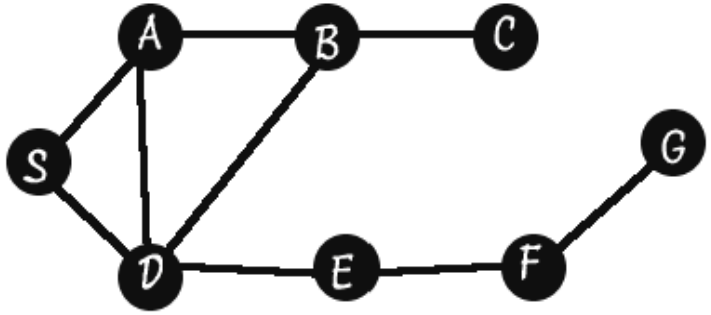
- Ne generišu se oni sledbenici koji se već pojavljuju u putu od korena do tekućeg čvora (da bi se izbegle petlje)
- Mali zahtevi za memorijom
- Ako postoje veoma dugački (ili neograničeni) putevi, ne garantuje rešenje
- Nije kompletno, ni optimalno

Primer: traženje po dubini
Cilj: M, V ili J



- A
- B C D
- E F C D
- K L F C D
- L F C D
- F C D
- M C D
- C D
- G H I D
- H I D
- N O I D
- O I D
- R S T I D
- S T I D
- V T I D
- T I D
- I D
- D
- J
- P Q
- Q
- U

Primer traženja po dubini



Stanje magacina

S

S->A, S->D

S->A->B, S->A->D, S->D (S->A->S petlja – eliminiše se)

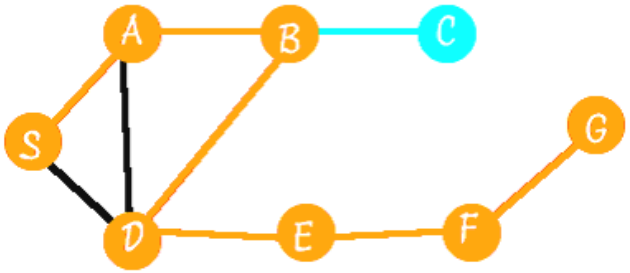
S->A->B->A (loop) **S->A->B->C,** S->A->B->D, S->A->D, S->D

S->A->B->D, S->A->D, S->D

S->A->B->D->E, S->A->D, S->D (tri petlje: S->A->B->D->S, S->A->B->D->A, S->A->B->D->B)

S->A->B->D->E->F, S->A->D, S->D

S->A->B->D->E->F->G, S->A->D, S->D



Depth-limited search

= DFS sa ograničenjem dubine traženja

(čvor je na zadatoj dubini ili nema sledbenike)

Rekurzivna implementacija:

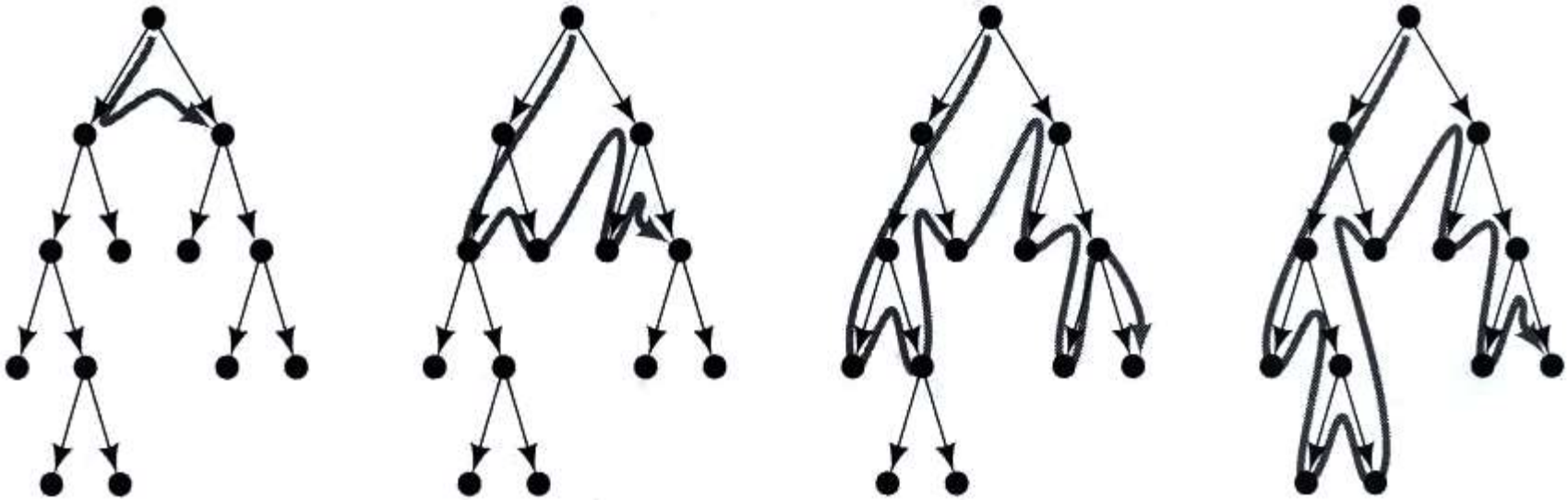
```
function DEPTH-LIMITED-SEARCH( problem, limit) returns soln/fail/cutoff
    RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
    cutoff-occurred? ← false
    if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
    else if DEPTH[node] = limit then return cutoff
    else for each successor in EXPAND(node, problem) do
        result ← RECURSIVE-DLS(successor, problem, limit)
        if result = cutoff then cutoff-occurred? ← true
        else if result ≠ failure then return result
    if cutoff-occurred? then return cutoff else return failure
```

Iterative deepening search

```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution, or failure
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
```

Primer IDS



Stages in Iterative-Deepening Search

Iterative deepening search - Osobine

- Kompletnost? Da
- Vreme? $(d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Prostor? $O(bd)$
- Optimalnost? Da, ako je cena puta = 1

Ostali algoritmi

- **Bidirekciono traženje**

  Istovremeno se polazi

  od S i od G

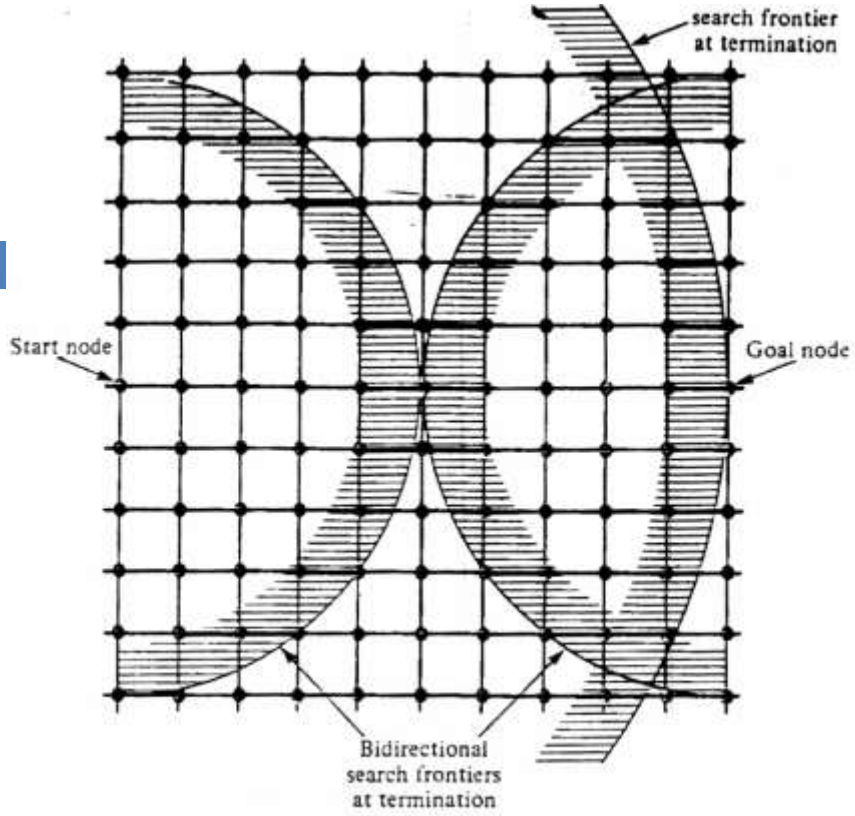


Fig. 2.10 Bidirectional and unidirectional breadth-first searches.

- **Nedeterminističko traženje**

  Varijanta BFS i/ili DFS, sa slučajnim izborom čvora koji se razvija

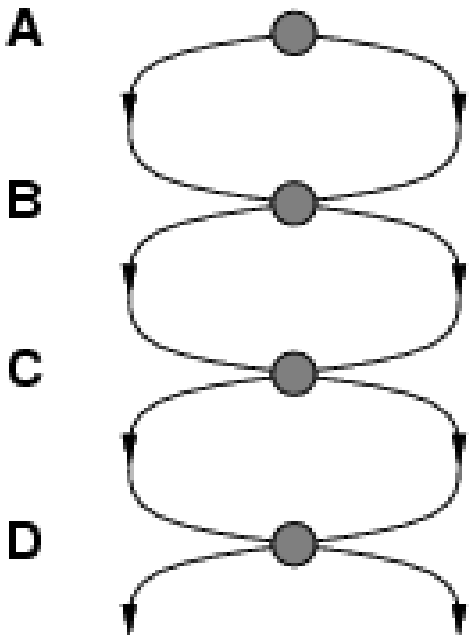
  Pokušaj rešavanja problema velikog faktora grananja i mnogo nivoa

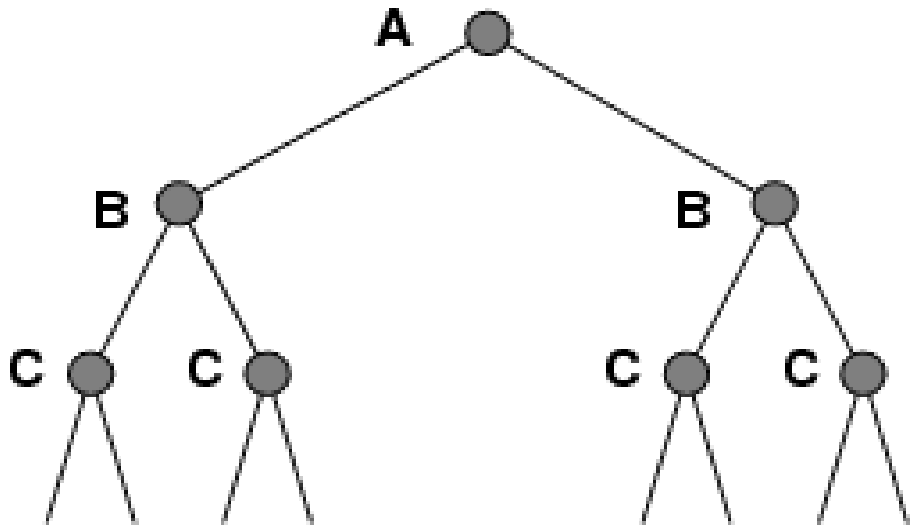
Kompleksnost neinformisanih algoritama

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

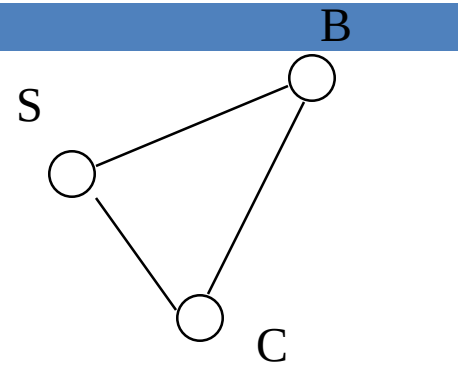
Problem kod implementacije: Stanja koja se ponavljaju

- Problem detekcije stanja koji se ponavljaju mogu da pretvore linearni problem u eksponencijalni!

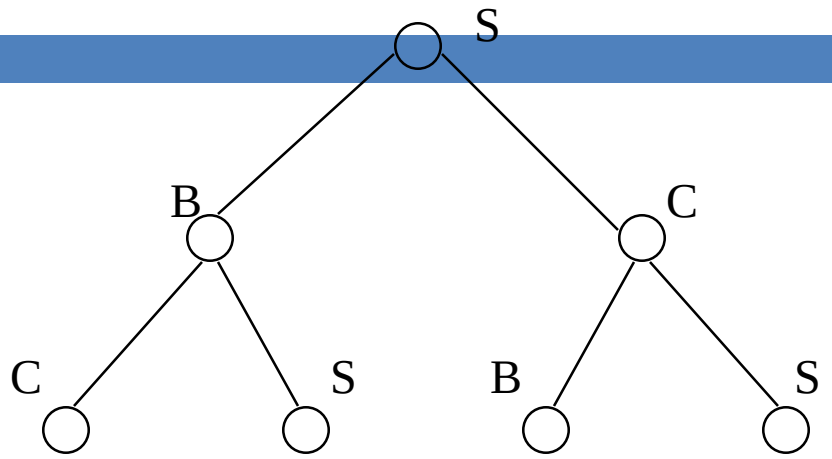




Izbegavanje stanja koja se ponavljaju



Prostor traženja



Primer stabla traženja

- Metoda 1 ← Nije optimala ali je praktična
  - Ne kreiraj put koji sadrži petlje
- Metoda 2 ← Optimalna ali memorijski neefkasna
  - Ne generiši stanje koje je već generisano
    - Moraju se znati sva moguća stanja (memorija!)
    - Primer: 8-puzzle problem, postoji 9! = 362,880 stanja

PITANJA?

Dileme?

Komentari?