

Uvod u umjetnu inteligenciju

2. Pretraživanje prostora stanja

prof. dr. sc. Jan Šnajder
izv. prof. dr. sc. Marko Čupić
prof. dr. sc. Bojana Dalbelo Bašić

Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Ak. god. 2022./2023.



Creative Commons Imenovanje–Nekomercijalno–Bez prerada 3.0

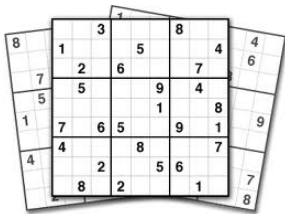
v3.8

Motivacija

- Mnogo se analitičkih problema može riješiti pretraživanjem prostora stanja
- Krenuvši od **početnog stanja** problema, pokušavamo pronaći **ciljno stanje**
- **Slijed akcija** koje nas vode do ciljnog stanja predstavljaju rješenje problema
- Problem predstavlja **velik broj stanja** te velik broj mogućih izbora
- Pretraživanje zato mora biti sustavno



Tipični problemi...



Formalan opis problema

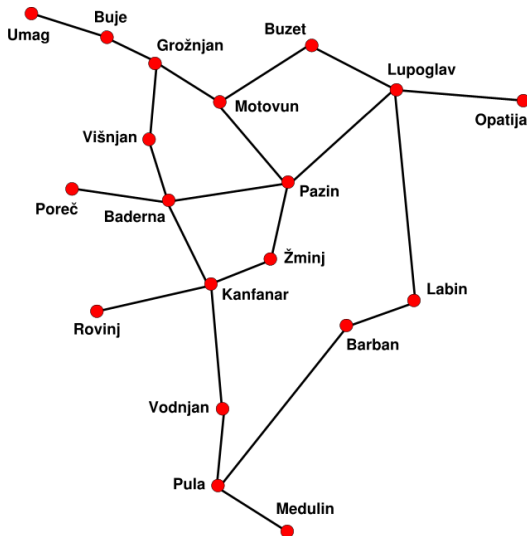
- Neka je S skup stanja (prostor stanja)
- Problem se sastoji od početnog stanja, prijelaza između stanja i ciljnog (ciljnih) stanja

Problem pretraživanja

$problem = (s_0, succ, goal)$

- 1 $s_0 \in S$ je **početno stanje**
 - 2 $succ : S \rightarrow \wp(S)$ je **funkcija sljedbenika** koja definira prijelaze između stanja
 - 3 $goal : S \rightarrow \{\top, \perp\}$ je **ispitni predikat** istinit samo za ciljna stanja
- Funkcija sljedbenika može se definirati implicitno pomoću skupa **operatora** (različitim operatorima prelazi se u različita stanja)

Primjer: Putovanje kroz Istru



Kako iz Pule do Buzeta?

$problem = (s_0, succ, goal)$

$s_0 = Pula$

$succ(Pula) =$
 $\{Barban, Medulin, Vodnjan\}$

$succ(Vodnjan) =$
 $\{Kanfanar, Pula\}$

\vdots

$goal(Buzet) = \top$

$goal(Motovun) = \perp$

$goal(Pula) = \perp$

\vdots

Zašto Buzet?



Divovska fritada s tartufima

Primjer: Slagalica 3×3

početno stanje:

8		7
6	5	4
3	2	1

ciljno stanje:

1	2	3
4	5	6
7	8	

Koji potezi vode do rješenja?

$problem = (s_0, succ, goal)$

$$s_0 = \begin{array}{|c|c|c|} \hline 8 & & 7 \\ \hline 6 & 5 & 4 \\ \hline 3 & 2 & 1 \\ \hline \end{array}$$

$$succ\left(\begin{array}{|c|c|c|} \hline 8 & & 7 \\ \hline 6 & 5 & 4 \\ \hline 3 & 2 & 1 \\ \hline \end{array}\right) = \left\{ \begin{array}{|c|c|c|} \hline & 8 & 7 \\ \hline 6 & 5 & 4 \\ \hline 3 & 2 & 1 \\ \hline \end{array}, \begin{array}{|c|c|c|} \hline 8 & 7 & \\ \hline 6 & 5 & 4 \\ \hline 3 & 2 & 1 \\ \hline \end{array}, \begin{array}{|c|c|c|} \hline 8 & 5 & 7 \\ \hline 6 & & 4 \\ \hline 3 & 2 & 1 \\ \hline \end{array} \right\}$$

\vdots

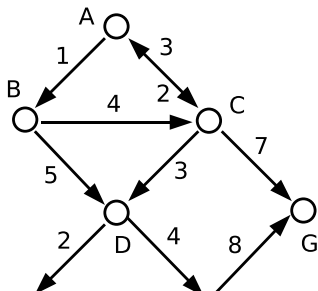
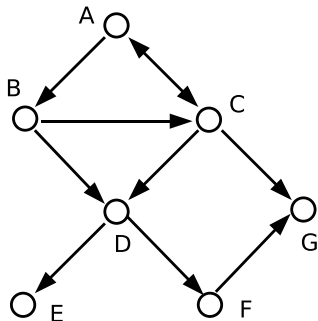
$$goal\left(\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ \hline 7 & 8 & \\ \hline \end{array}\right) = \top$$

$$goal\left(\begin{array}{|c|c|c|} \hline 8 & & 7 \\ \hline 6 & 5 & 4 \\ \hline 3 & 2 & 1 \\ \hline \end{array}\right) = \perp$$

$$goal\left(\begin{array}{|c|c|c|} \hline & 8 & 7 \\ \hline 6 & 5 & 4 \\ \hline 3 & 2 & 1 \\ \hline \end{array}\right) = \perp$$

\vdots

Ideja pretraživanja



- Pretraživanje prostora stanja svodi se na pretraživanje **usmjerenog grafa** (digrafa)
- Vrhovi grafa = stanja
lukovi = prijelazi između stanja
- Graf može biti zadan eksplicitno ili implicitno
- Graf može imati cikluse
- Ako definiramo cijene prijelaza, onda je to **usmjeren težinski graf** (težinski digraf)

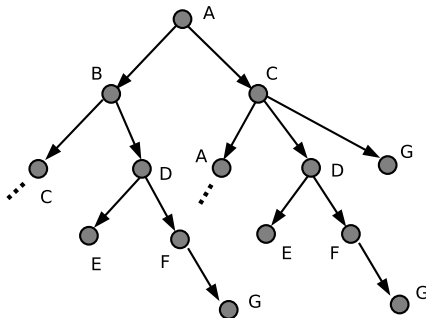
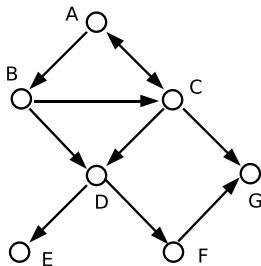
Stablo pretraživanja

- Pretraživanjem usmjerenog grafa postepeno nastaje **stablo pretraživanja**
- Stablo nastaje tako da pojedine čvorove **proširujemo**: pomoću funkcije sljedbenika (odnosno operatora) generiramo sve sljedbenike nekog čvora
- **Otvoreni čvorovi** ili **fronta**: čvorovi koji su generirani, ali još nisu prošireni
- **Zatvoreni čvorovi**: čvorovi koji su već prošireni

Strategija pretraživanja

Redoslijed kojim proširujemo čvorove određuje **strategiju pretraživanja**. Različiti redoslijedi daju različite strategije.

Prostor stanja vs. stablo pretraživanja



- Stablo pretraživanja **nastaje** pretraživanjem prostora stanja
 - Stablo pretraživanja može biti beskonačno čak i onda kada je prostor stanja konačan
- NB:** prostor stanja ima cikluse \Rightarrow stablo pretraživanja je beskonačno

Stanje vs. čvor

- Čvor n je podatkovna struktura koja sačinjava stablo pretraživanja
- **Čvor pohranjuje stanje**, ali i još neke dodatne podatke:

Podatkovna struktura čvora

$$n = (s, d)$$

s – stanje

d – dubina čvora u stablu

$$\text{state}(n) = s, \text{ depth}(n) = d$$

$$\text{initial}(s) = (s, 0)$$

Opći algoritam pretraživanja

Opći algoritam pretraživanja

```
function search( $s_0$ , succ, goal)
   $open \leftarrow [\text{initial}(s_0)]$ 
  while  $open \neq []$  do
     $n \leftarrow \text{removeHead}(open)$ 
    if goal(state( $n$ )) then return  $n$ 
    for  $m \in \text{expand}(n, \text{succ})$  do
      insert( $m, open$ )
  return fail
```

- $\text{removeHead}(l)$ – skida prvi element neprazne liste l
- $\text{expand}(n, \text{succ})$ – proširuje čvor n uporabom funkcije sljedbenika succ
- $\text{insert}(n, l)$ – umeće čvor n u listu l

Proširivanje čvora

- Proširivanje čvora treba ažurirati sve komponente čvora:

Proširivanje čvora

```
function expand( $n$ , succ)  
  return { ( $s$ , depth( $n$ ) + 1) |  $s \in \text{succ}(\text{state}(n))$  }
```

- Funkcija će biti složenija kada u čvor budemo pohranjivali dodatne podatke (npr. pokazivač na roditeljski čvor)

Usporedba problema i algoritama

Karakteristike problema:

- $|S|$ – **broj stanja**
- b – **faktor grananja** stabla pretraživanja
- d – **dubina optimalnog rješenja** u stablu pretraživanja
- m – **maksimalna dubina** stabla pretraživanja (moguće ∞)

Svojstva algoritama:

- 1 **Potpunost** (engl. *completeness*) – algoritam je potpun akko pronalazi rješenje uvijek kada ono postoji
- 2 **Optimalnost** (engl. *optimality, admissibility*) – algoritam je optimalan akko pronalazi optimalno rješenje (ono s najmanjom cijenom)
- 3 **Vremenska složenost** (broj generiranih čvorova)
- 4 **Prostorna složenost** (broj pohranjenih čvorova)

Rasprava: problem slagalice 3×3

Razmislite o problemu slagalice 3×3 kao o problemu pretraživanja prostora stanja. Željeli bismo okarakterizirati složenost tog problema.

- Uparite se sa susjedom
- Pokušajte odrediti ukupan broj stanja $|S|$
- Koji je minimalan a koji maksimalan faktor grananja?
- Izračunajte prosječan faktor grananja.
- Napišite svoje odgovore

Strategije pretraživanja

Dvije osnovne vrste strategija pretraživanja:

- **Slijepo pretraživanje** (engl. *blind, uninformed search*)
- **Usmjereno pretraživanje** (engl. *directed, informed, heuristic search*)

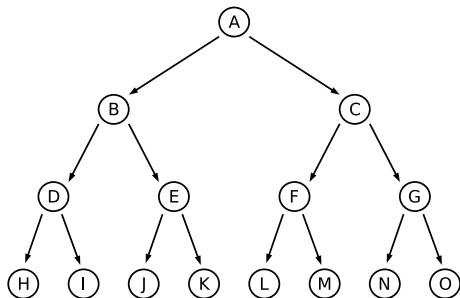
Danas govorimo samo o slijepom pretraživanju.

Slijepo pretraživanje

- 1 Pretraživanje u širinu (engl. *breadth-first search*, BFS)
- 2 Pretraživanje s jednolikom cijenom (engl. *uniform-cost search*)
- 3 Pretraživanje u dubinu (engl. *depth-first search*, DFS)
- 4 Ograničeno pretraživanje u dubinu
- 5 Iterativno pretraživanje u dubinu

Pretraživanje u širinu

- Jednostavna slijepa strategija pretraživanja
- Nakon proširenja korijenskog čvora, proširuju se sva njegova djeca, zatim sva djeca djece, itd.
- Općenito, čvorovi na dubini d proširuju se tek nakon što se prošire svi čvorovi na razini $d - 1$, tj. pretražujemo **razinu po razinu**



A, B, C, D, E, F, G, H, ...

Pretraživanje u širinu – izvedba

- Ovakvu strategiju ostvarit ćemo ako generirane čvorove uvijek **dodajemo na kraj** liste otvorenih čvorova

Pretraživanje u širinu

```
function breadthFirstSearch( $s_0$ , succ, goal)
   $open \leftarrow [initial(s_0)]$ 
  while  $open \neq []$  do
     $n \leftarrow removeHead(open)$ 
    if goal(state( $n$ )) then return  $n$ 
    for  $m \in expand(n, succ)$  do
      insertBack( $m, open$ )
  return fail
```

- Lista otvorenih čvorova zapravo je **red** (engl. *queue*)

Pretraživanje u širinu – primjer izvođenja

- 0 $open = [(Pula, 0)]$
- 1 $expand(Pula, 0) = \{(Vodnjan, 1), (Barban, 1), (Medulin, 1)\}$
 $open = [(Vodnjan, 1), (Barban, 1), (Medulin, 1)]$
- 2 $expand(Vodnjan, 1) = \{(Kanfanar, 2), (Pula, 2)\}$
 $open = [(Barban, 1), (Medulin, 1), (Kanfanar, 2), (Pula, 2)]$
- 3 $expand(Barban, 1) = \{(Labin, 2), (Pula, 2)\}$
 $open = [(Medulin, 1), (Kanfanar, 2), (Pula, 2), (Labin, 2), (Pula, 2)]$
- 4 $expand(Medulin, 1) = \{(Pula, 2)\}$
 $open = [(Kanfanar, 2), (Pula, 2), (Labin, 2), (Pula, 2), (Pula, 2)]$
- 5 $expand(Kanfanar, 2) =$
 $\{(Baderna, 3), (Rovinj, 3), (Vodnjan, 3), (Zminj, 3)\}$
 $open = [(Pula, 2), (Labin, 2), (Pula, 2), (Pula, 2), (Baderna, 3), \dots]$
 \vdots

Pretraživanje u širinu – svojstva

- Pretraživanje u širinu je **potpuno** i **optimalno**
- U svakom koraku proširuje se najbliži čvor, pa je strategija optimalna (uz pretpostavku da je cijena prijelaza konstantna)
- **Vremenska složenost:**
$$1 + b + b^2 + b^3 + \dots + b^d + (b^{d+1} - b) = \mathcal{O}(b^{d+1})$$

(na zadnjoj razini generiraju se sljedbenici svih čvorova osim ciljnog)
- **Prostorna složenost:** $\mathcal{O}(b^{d+1})$
- Eksponencijalna složenost (pogotovo prostorna) glavni je nedostatak pretraživanja u širinu
- Npr. $b = 4, d = 16, 10 B/\text{čvor} \rightarrow 43 GB$
- Primjenjivo samo na male probleme

Podsjetnik: Asimptotska složenost algoritma

- Asimptotska složenost funkcije: ponašanje funkcije $f(n)$ kada $n \rightarrow \infty$ izražena pomoću jednostavnijih funkcija

Notacija “Veliko-O”

$$\mathcal{O}(g(n)) = \{f(n) \mid \exists c, n_0 \geq 0 \text{ takvi da} \\ \forall n \geq n_0. 0 \leq f(n) \leq c \cdot g(n)\}$$

- Konvencija: umjesto $f(n) \in \mathcal{O}(g(n))$ pišemo $f(n) = \mathcal{O}(g(n))$
- Gornja ograda složenosti (složenost u najgorem slučaju)
- Donja ograda nije definirana, pa npr. $n = \mathcal{O}(n), n = \mathcal{O}(n^2), \dots$ (u principu nas zanima ona najmanja ograda)
- $\Theta(g(n))$ definira i gornju i donju ogradu (engl. *tight bounds*)

Cijene prijelaza

- Ako operacije (prijelazi između stanja) nisu jednake cijene, funkciju sljedećeg stanja modificiramo tako da ona za svakog sljedbenika vraća i cijenu prijelaza:

$$\text{succ} : S \rightarrow \wp(S \times \mathbb{R}^+)$$

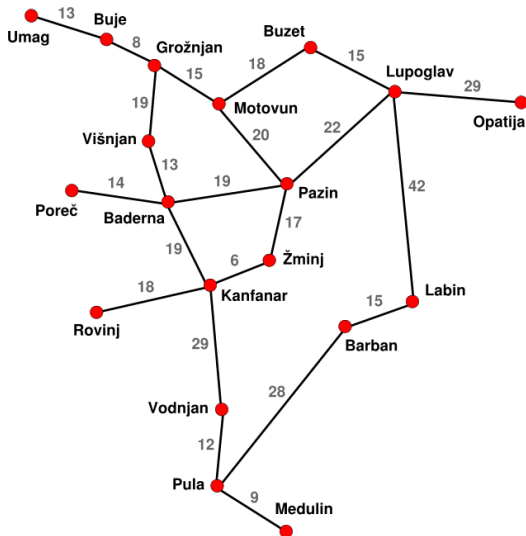
- U čvoru više ne pohranjujemo dubinu nego **ukupnu cijenu puta** do tog čvora:

$$n = (s, c), \quad g(n) = c$$

- Funkciju proširenja čvora moramo također modificirati tako da ažurira cijenu puta do čvora:

```
function expand( $n$ , succ)  
  return { ( $s$ ,  $g(n) + c$ ) | ( $s$ ,  $c$ )  $\in$  succ(state( $n$ )) }
```

Primjer: Putovanje kroz Istru



Kako iz Pule do Buzeta?

$problem = (s_0, succ, goal)$

$s_0 = Pula$

$succ(Pula) =$
 $\{(Barban, 28), (Medulin, 9),$
 $(Vodnjan, 12)\}$

$succ(Vodnjan) =$
 $\{(Kanfanar, 29), (Pula, 12)\}$

\vdots

$goal(Buzet) = \top$

$goal(Motovun) = \perp$

$goal(Pula) = \perp$

\vdots

Pretraživanje s jednolikom cijenom

- Kao i pretraživanje u širinu, no u obzir uzimamo cijenu prijelaza

Pretraživanje s jednolikom cijenom

```
function uniformCostSearch( $s_0$ , succ, goal)
   $open \leftarrow [initial(s_0)]$ 
  while  $open \neq []$  do
     $n \leftarrow removeHead(open)$ 
    if goal(state( $n$ )) then return  $n$ 
    for  $m \in expand(n, succ)$  do
      insertSortedBy( $g, m, open$ )
  return fail
```

- **insertSortedBy**(f, n, l) – umeće čvor n u listu l sortiranu uzlazno prema vrijednosti $f(n)$
- Lista $open$ funkcionira kao **prioritetni red**

Pretraživanje s jednolikom cijenom – primjer izvođenja

- ① $open = [(Pula, 0)]$
- ① $expand(Pula, 0) = \{(Vodnjan, 12), (Barban, 28), (Medulin, 9)\}$
 $open = [(Medulin, 9), (Vodnjan, 12), (Barban, 28)]$
- ② $expand(Medulin, 9) = \{(Pula, 18)\}$
 $open = [(Vodnjan, 12), (Pula, 18), (Barban, 28)]$
- ③ $expand(Vodnjan, 12) = \{(Kanfanar, 41), (Pula, 24)\}$
 $open = [(Pula, 18), (Pula, 24), (Barban, 28), (Kanfanar, 41)]$
- ④ $expand(Pula, 18) = \{(Vodnjan, 30), (Barban, 46), (Medulin, 27)\}$
 $open = [(Pula, 24), (Medulin, 27), (Barban, 28), (Vodnjan, 30), \dots]$
 \vdots

Q: Hoće li ovaj algoritam pronaći rješenje (Buzet)?

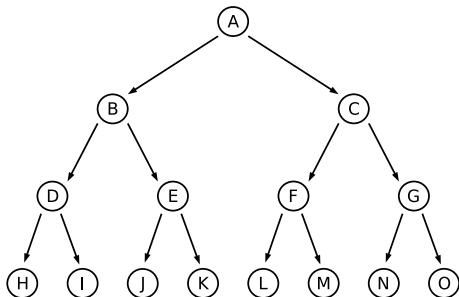
Q: Hoće li pronaći put do Buzeta s najmanjom cijenom?

Pretraživanje s jednolikom cijenom – svojstva

- Algoritam je **potpun** i **optimalan**
- Ako je C^* optimalna cijena do cilja, a ε minimalna cijena prijelaza, dubina stabla do ciljnog čvora je $d = \lfloor C^*/\varepsilon \rfloor$
- Vremenska i prostorna složenost: $\mathcal{O}(b^{1+\lfloor C^*/\varepsilon \rfloor})$

Pretraživanje u dubinu

- Pretraživanje u dubinu uvijek prvo proširuje najdublji čvor u stablu pretraživanja
- Postupak se vraća na pliće razine tek kada dosegne listove (stanja koja nemaju sljedbenika)



A, B, D, H, I, E, J, K, C, ...

Pretraživanje u dubinu – izvedba

- Strategiju pretraživanja u dubinu ostvarit ćemo ako generirane čvorove **dodajemo na početak** liste *open*

Pretraživanje u dubinu

```
function depthFirstSearch( $s_0$ , succ, goal)
   $open \leftarrow [\text{initial}(s_0)]$ 
  while  $open \neq []$  do
     $n \leftarrow \text{removeHead}(open)$ 
    if goal(state( $n$ )) then return  $n$ 
    for  $m \in \text{expand}(n, \text{succ})$  do
      insertFront( $m, open$ )
  return fail
```

- Lista otvorenih čvorova zapravo je **stog**

Pretraživanje u dubinu – primjer izvođenja

- 0 $open = [(Pula, 0)]$
- 1 $expand(Pula, 0) = \{(Vodnjan, 1), (Barban, 1), (Medulin, 1)\}$
 $open = [(Vodnjan, 1), (Barban, 1), (Medulin, 1)]$
- 2 $expand(Vodnjan, 1) = \{(Kanfanar, 2), (Pula, 2)\}$
 $open = [(Kanfanar, 2), (Pula, 2), (Barban, 1), (Medulin, 1)]$
- 3 $expand(Kanfanar, 2) =$
 $\{(Baderna, 3), (Rovinj, 3), (Vodnjan, 3), (Zminj, 3)\}$
 $open = [(Baderna, 3), (Rovinj, 3), (Vodnjan, 3), (Zminj, 3), (Pula, 2), \dots]$
- 4 $expand(Baderna, 3) = \{(Porec, 4), (Visnjan, 4), (Pazin, 4), (Kanfanar, 4)\}$
 $open =$
 $[(Porec, 4), (Visnjan, 4), (Pazin, 4), (Kanfanar, 4), (Baderna, 3), \dots]$
 \vdots

Q: Je li ovo jedini mogući tijek izvođenja?

Pretraživanje u dubinu – svojstva

- Pretraživanje u dubinu manje je memorijski zahtjevno
- **Prostorna složenost:** $\mathcal{O}(bm)$, gdje je m maksimalna dubina stabla
- **Vremenska složenost:** $\mathcal{O}(b^m)$
(nepovoljno, ako $m \gg d$)
- **Potpunost:** ne, jer može zaglaviti u beskonačnoj petlji
- **Optimalnost:** ne, jer ne pretražuje razinu po razinu
- Pretraživanje u dubinu treba izbjegavati kod stabla pretraživanja čija je maksimalna dubina velika ili beskonačna

Pretraživanje u dubinu – rekurzivna izvedba

- Listu *open* možemo izbjeći:

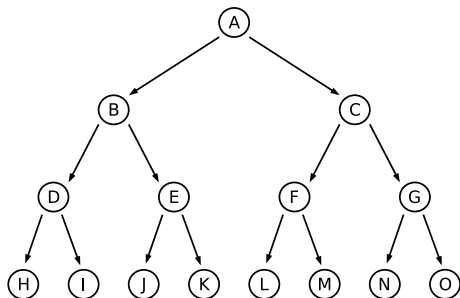
Pretraživanje u dubinu (rekurzivna izvedba)

```
function depthFirstSearch(s, succ, goal)
  if goal(s) then return s
  for m ∈ succ(s) do
    r ← depthFirstSearch(m, succ, goal)
    if r ≠ fail then return r
  return fail
```

- Umjesto eksplicitne liste *open* koristi se sistemski stog
- Uz lijenu (nestriktnu) evaluaciju skupa succ(*s*): prostorna složenost je $\mathcal{O}(m)$ umjesto $\mathcal{O}(bm)$

Ograničeno pretraživanje u dubinu

- Pretražuje u dubinu, ali ne dublje od zadane granice



$k = 0$: A

$k = 1$: A, B, C

$k = 2$: A, B, D, E, C, F, G

Ograničeno pretraživanje u dubinu – izvedba

- Čvor proširujemo samo ako se u stablu pretraživanja nalazi iznad dubinskog ograničenja k :

Ograničeno pretraživanje u dubinu

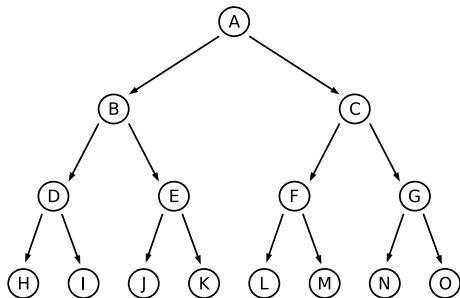
```
function depthLimitedSearch( $s_0$ , succ, goal,  $k$ )  
   $open \leftarrow [initial(s_0)]$   
  while  $open \neq []$  do  
     $n \leftarrow removeHead(open)$   
    if goal(state( $n$ )) then return  $s$   
    if depth( $n$ ) <  $k$  then  
      for  $m \in expand(n, succ)$  do  
        insertFront( $m, open$ )  
  return fail
```

Ograničeno pretraživanje u dubinu – svojstva

- **Prostorna složenost:** $\mathcal{O}(bk)$, gdje je k dubinska granica
- **Vremenska složenost:** $\mathcal{O}(b^k)$
- **Potpunost:** ne, jer može biti $d > k$
- **Optimalnost:** ne, jer ne pretražuje razinu po razinu
- Algoritam je uporabiv ako znamo dubinu rješenja d (možemo postaviti $k = |S|$)

Iterativno pretraživanje u dubinu

- Izbjegava problem izbora optimalne dubinske granice isprobavajući sve moguće vrijednosti krenuvši od dubine 0
- Kombinira prednosti pretraživanja u dubinu i pretraživanja u širinu



A, A, B, C, A, B, D, E, C, F,
G, A, B, D, H, ...

Iterativno pretraživanje u dubinu – izvedba

Iterativno pretraživanje u dubinu

```
function iterativeDeepeningSearch( $s_0$ , succ, goal)
  for  $k \leftarrow 0$  to  $\infty$  do
     $result \leftarrow$  depthLimitedSearch( $s_0$ , succ, goal,  $k$ )
    if  $result \neq fail$  then return  $result$ 
```

Iterativno pretraživanje u dubinu – svojstva

- Strategija se na prvi pogled čini neučinkovitom: više puta proširujemo iste čvorove
- U većini slučajeva to ne predstavlja problem: većina čvorova stabla nalazi se na dubljim razinama, pa ponavljanje proširivanja preostalih čvorova na višim razinama nije problematično
- **Vremenska složenost:** $\mathcal{O}(b^d)$
- **Prostorna složenost:** $\mathcal{O}(bd)$
- **Potpunost:** da, jer koristi dubinsko ograničenje i povećava ga
- **Optimalnost:** da, jer pretražuje razinu po razinu
- Iterativno pretraživanje u dubinu preporučena je strategija za probleme s velikim prostorom stanja i nepoznatom dubinom rješenja

Iterativno pretraživanje u dubinu – složenost

- Broj generiranih čvorova kod pretraživanja u širinu je

$$1 + b + b^2 + \dots + b^{d-2} + b^{d-1} + b^d + (b^{d+1} - b)$$

pa je asimptotska vremenska složenost $\mathcal{O}(b^{d+1})$

- Broj proširenih čvorova kod iterativnog pretraživanja je

$$(d+1)1 + db + (d-1)b^2 + \dots + 3b^{d-2} + 2b^{d-1} + 1b^d$$

pa je asimptotska vremenska složenost $\mathcal{O}(b^d)$

- Razlika je to manja što je veći faktor grananja
- Npr. za $b = 2$ to je 100% više čvorova
za $b = 3$ to je 50% više čvorova
za $b = 10$ to je 11% više čvorova

Vježba: pretraživanje IDS

Pitanje 2

Skup stanja: $S = \{a, b, c, d, e\}$. Prijelazi: $f(a) = \{b, c, d\}$, $f(b) = \{c\}$, $f(c) = \{a, d, e\}$, $f(d) = \{e\}$, $f(e) = \emptyset$. Početno stanje je a , ciljno stanje je e . Koji redoslijedom pretražuje algoritam IDS?

- A $a, a, b, c, d, c, a, d, e$
- B $a, a, b, c, a, b, c, a, b, c, \dots$
- C a, a, b, c, c, a, d, e
- D $a, a, b, c, d, a, b, c, c, a, d, e$

Usporedba algoritama slijepog pretraživanja

Algoritam	Vrijeme	Prostor	Potpunost	Opt.
U širinu	$\mathcal{O}(b^{d+1})$	$\mathcal{O}(b^{d+1})$	Da	Da
Jednol. cijena	$\mathcal{O}(b^{1+\lfloor C^*/\epsilon \rfloor})$	$\mathcal{O}(b^{1+\lfloor C^*/\epsilon \rfloor})$	Da	Da
U dubinu	$\mathcal{O}(b^m)$	$\mathcal{O}(bm)$	Ne	Ne
Ogr. u dubinu	$\mathcal{O}(b^k)$	$\mathcal{O}(bk)$	Ne	Ne
Iter. u dubinu	$\mathcal{O}(b^d)$	$\mathcal{O}(bd)$	Da	Da

b – faktor grananja, d – dubina optimalnog rješenja,

m – maksimalna dubina stabla ($m \geq d$), k – dubinsko ograničenje

- Svi su algoritmi **eksponencijalne vremenske složenosti!**
- Pretraživanje u dubinu (i njegove varijante) bolje su prostorne složenosti od pretraživanja u širinu

Rekonstrukcija rješenja

- U strukturi čvora moramo pamtit i pokazivač na roditeljski čvor:

$$n = (s, d, p), \text{ parent}(n) = p$$

```
function expand( $n$ , succ)  
  return { ( $s$ , depth( $n$ ) + 1,  $n$ ) |  $s \in \text{succ}(\text{state}(n))$  }
```

- Krećemo od ciljnog čvora i pratimo pokazivače unazad:

Rekonstrukcija puta do čvora

```
function path( $n$ )  
   $p \leftarrow \text{parent}(n)$   
  if  $p = \text{null}$  then return [state( $n$ )]  
  return insertBack(state( $n$ ), path( $p$ ))
```

- Vremenska složenost je $\mathcal{O}(d)$
- **NB:** Zatvorene čvorove očit o moramo **čuvati u memoriji!**

Problem ponavljanja stanja (1)

- **Rješenje 1:** spriječiti povratak u stanje iz kojeg smo došli

Opći algoritam pretraživanja (nadopuna)

```
function search( $s_0$ , succ, goal)
   $open \leftarrow [initial(s_0)]$ 
  while  $open \neq []$  do
     $n \leftarrow removeHead(open)$ 
    if goal(state( $n$ )) then return  $n$ 
    for  $m \in expand(n)$  do
      if state( $m$ )  $\neq$  state(parent( $n$ )) then insert( $m$ ,  $open$ )
  return fail
```

- **Q:** Rješava li ovo problem ciklusa?
A: Ne općenito! (samo cikluse duljine 2, tzv. transpozicije)

Problem ponavljanja stanja (2)

- **Rješenje 2:** spriječiti nastanak puteva s ciklusima

Opći algoritam pretraživanja (nadopuna)

```
function search( $s_0$ , succ, goal)
   $open \leftarrow [initial(s_0)]$ 
  while  $open \neq []$  do
     $n \leftarrow removeHead(open)$ 
    if goal(state( $n$ )) then return  $n$ 
    for  $m \in expand(n)$  do
      if state( $m$ )  $\notin$  path( $n$ ) then insert( $m$ ,  $open$ )
  return fail
```

- Sprječava cikluse (osigurava potpunost algoritma)
- Ne sprječava ponavljanje na različitim putevima u stablu pretraživanja
- Povećava vremensku složenost za faktor $\mathcal{O}(d)$

Problem ponavljanja stanja (3)

- **Rješenje 3:** spriječiti ponavljanje bilo kojeg stanja

Opći algoritam pretraživanja sa skupom posjećenih stanja

```
function search( $s_0$ , succ, goal)
   $open \leftarrow [initial(s_0)]$ 
   $visited \leftarrow \emptyset$ 
  while  $open \neq []$  do
     $n \leftarrow removeHead(open)$ 
    if goal(state( $n$ )) then return  $n$ 
     $visited \leftarrow visited \cup \{state(n)\}$ 
    for  $m \in expand(n)$  do
      if state( $m$ )  $\notin visited$  then insert( $m$ ,  $open$ )
  return fail
```

- **NB:** Skup posjećenih stanja pohranjuje stanja, a ne čvorove

Problem ponavljanja stanja – komentari

- Korištenje skupa posjećenih stanja osigurava potpunost algoritma (sprječava da zaglavi u beskonačnoj petlji)
- Osim toga, može **smanjiti prostornu i vremensku složenost**:
Budući da se stanja ne ponavljaju, umjesto složenosti $\mathcal{O}(b^{d+1})$ imamo $\mathcal{O}(\min(b^{d+1}, b|S|))$, gdje je $|S|$ veličina prostora stanja (u praksi je često $b|S| < b^d$)
- Lista posjećenih stanja uobičajeno se implementira tablicom raspršenog adresiranja (engl. *hash table*) (omogućava provjeru “ $\text{state}(m) \notin \text{visited}$ ” u vremenu $\mathcal{O}(1)$)

Primjer: Problem misionara i kanibala (1)

Problem misionara i kanibala

Tri misionara i tri kanibala potrebno je jednim čamcem prevesti s jedne strane obale rijeke na drugu, pri čemu se niti u jednom trenutku na jednoj strani obale ne smije naći više kanibala nego misionara. Čamac može prevesti najviše dvije osobe i ne može ploviti prazan. Tražimo rješenje s **najmanjim brojem koraka**.



http://www.game.st/game_276_missionaries_and_cannibals.html

- Koji algoritam pretraživanja upotrijebiti?
- Kako prikazati problem?

Rješenje (Haskell):

[http://www.fer.unizg.hr/_download/repository/misionari\[1\].hs](http://www.fer.unizg.hr/_download/repository/misionari[1].hs)

Primjer: Problem misionara i kanibala (2)

$problem = (s_0, succ, goal)$

① $s_0 = (3, 3, L)$

- ▶ broj misionara na lijevoj obali, $\{0, 1, 2, 3\}$
- ▶ broj kanibala na lijevoj obali, $\{0, 1, 2, 3\}$
- ▶ pozicija čamca, $\{L, R\}$

② $succ(m, c, b) = \{s \mid s \in Succs, safe(s)\}$

$$Moves = \{(1, 1), (2, 0), (0, 2), (1, 0), (0, 1)\}$$

$$Succs = \{f(\Delta m, \Delta c) \mid (\Delta m, \Delta c) \in Moves\}$$

$$f(\Delta m, \Delta c) = \begin{cases} (m - \Delta m, c - \Delta c, R) & \text{ako } b = L \\ (m + \Delta m, c + \Delta c, L) & \text{inače} \end{cases}$$

$$safe(m, c, b) = (m = 0) \vee (m = 3) \vee (m = c)$$

③ $goal(m, c, b) = \begin{cases} \top & \text{ako } (m = c = 0) \wedge (b = R) \\ \perp & \text{inače} \end{cases}$

Primjer: Problem misionara i kanibala (3)

- Jesmo li opisali sve što je bitno za problem?
- Jesmo li apstrahirali sve što je nebitno za problem?
- Generiramo li sve moguće poteze?
- Jesu li potezi koje generiramo legalni?
- Generiramo li neželjena stanja?
- Bi li bilo pametnije provjeru ispravnosti stanja ugraditi u ispitni predikat *goal*?
- Trebamo li paziti na ponavljanje stanja?
- Kakve su karakteristike problema?
 - ▶ $|S| = ?$
 - ▶ $b = ?$
 - ▶ $d = ?$
 - ▶ $m = ?$
- Je li ovo težak problem?

Laboratorijski zadatak: Problem ljubomornih muževa

Napišite program koji će **pretraživanjem u širinu** pronaći optimalno rješenje problema ljubomornih muževa.

Problem je opisan na sljedeći način: tri muža i njihove supruge potrebno je jednim čamcem prevesti s jedne strane obale rijeke na drugu. Niti u jednom trenutku ne smije se dogoditi da neka od suprugica bude u prisustvu drugoga muža, a da njezin muž nije također prisutan. Čamac može prevesti najviše dvije osobe i ne može ploviti prazan.

Optimalno rješenje je ono s najmanjim brojem koraka. Program treba ispisati rješenje u obliku niza operatora i stanja koja dovode do ciljnog stanja.

Q: Kako definirati skup S i funkciju succ ?

Laboratorijski zadatak: Igra brojki

Napišite program koji koristi **iterativno pretraživanje u dubinu** kako bi riješio problem igre brojki.

Cilj igre je, krenuvši od šest zadanih cijelih brojeva, pronaći aritmetički postupak kojim se izvodi neki slučajno generirani ciljani broj. Aritmetičke operacije koje se pritom smiju koristiti jesu zbrajanje, oduzimanje, množenje i dijeljenje bez ostatka. Svaki se broj može iskoristiti samo jednom ili niti jednom.

Početnih šest brojeva neka zadaje korisnik, a ciljani broj neka se generira slučajno iz intervala od 100 do 999.

Ako izraz kojim se izvodi točan broj nije pronađen, treba pronaći izraz koji izvodi broj najbliži traženome.

Q: Kako definirati skup S i funkciju `succ`?

Eksperimentiranje

- Implementacija slijepih algoritama pretraživanja (by Marko Čupić)
 - ▶ <http://java.zemris.fer.hr/nastava/ui/>
- Možete se igrati s formuliranjem prethodnih zadataka kao problema pretraživanja prostora stanja:
 - ▶ Prikaz stanja
 - ▶ Generiranje sljedećih stanja (tj., funkcija *Succ*)
 - ▶ Ciljni predikat

Sažetak

- Mnogi UI problemi mogu se riješiti pretraživanjem prostora stanja
- Problemi se razlikuju po **broju stanja**, **faktoru grananja** i **dubini**
- Poželjna svojstva algoritama pretraživanja su **potpunost**, **optimalnost** i **mala prostorna složenost**
- Svi algoritmi pretraživanja nažalost imaju **eksponencijalnu vremensku složenost**
- Kod velikog broja stanja preporuča se koristiti **iterativno pretraživanje u dubinu**
- Treba voditi računa o ciklusima (ponavljanju stanja)



Sljedeća tema: Heurističko pretraživanje