OSNOVE UMETNE INTELIGENCE

2022/23

strojno učenje

nenadzorovano učenje

preiskovanje

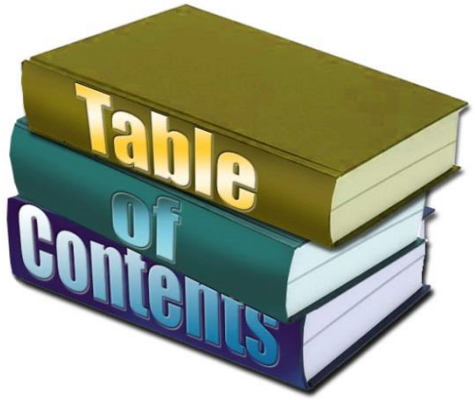
cilji, primeri problemov

neinformirani algoritmi

Pridobljeno znanje s prejšnjih predavanj

- **strojno učenje**
  - **metoda k najbližjih sosedov**
    - leno učenje
    - merjenje razdalje (razdalja Minkowskega - evklidska, manhattanska, hammingova razdalja)
    - izbira ustreznega k (prešibko ali pretirano prilagajanje)
    - normalizacija, prekletstvo dimenzionalnosti
  - **lokalno utežena regresija**
    - uteževanje označb z razdaljo
    - uporaba jedra, nastavitev širine jedra
  - **regresijska drevesa**
    - srednja kvadratna napaka (MSE) kot mera nečistoče
  - **nenadzorovano učenje**
    - hierarhično gručenje (združevalno, delilno): dendrogram, rezanje dendrograma, merjenje razdalj med primeri, med gručami in med gručo-primerom

Pregled



I. strojno učenje

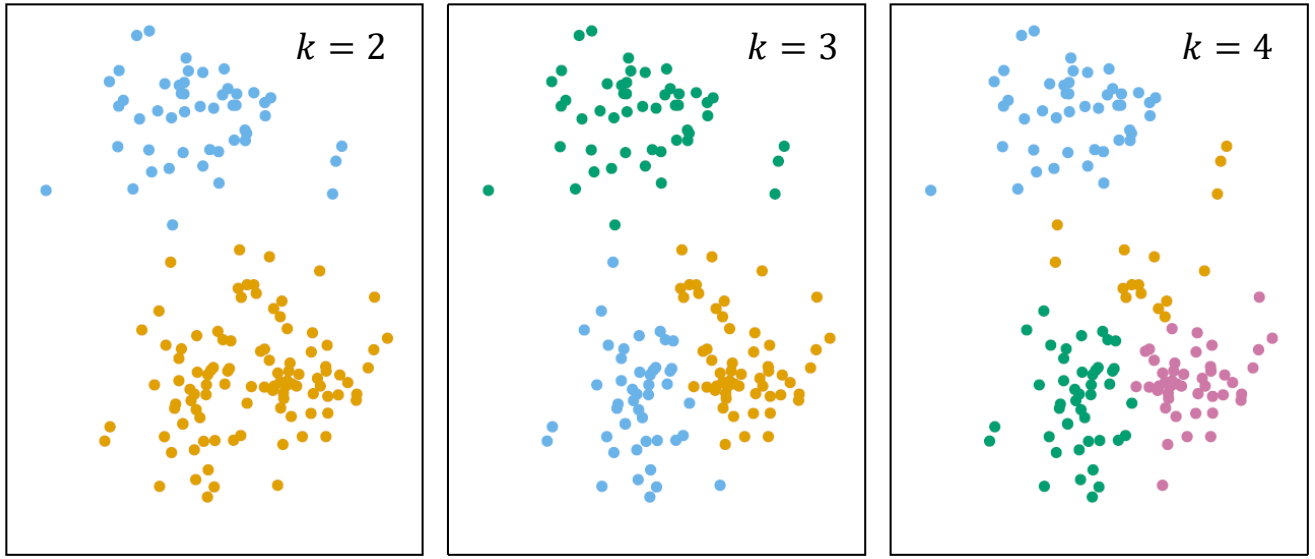
- uvod v strojno učenje
- učenje odločitvenih dreves
- učenje dreves iz šumnih podatkov (rezanje dreves)
- ocenjevanje učenja
- diskretizacija atributov, obravnava manjkajočih vrednosti
- naivni Bayesov klasifikator
- nomogrami
- k najbližjih sosedov
- lokalna utežena regresija
- regresijska drevesa
- linearni modeli
- hierarhično gručenje
- gručenje z metodo k voditeljev

II. preiskovanje

- definicija in cilji področja
- primeri umetnih in realnih problemov

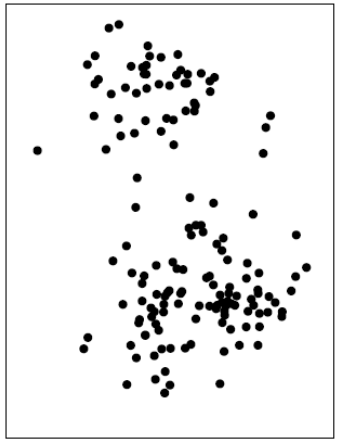
Metoda voditeljev

- angl. *k-means clustering*
- postopek:
  - izberi **začetno število gruč** k in **naključno priredi** vsak učni primer eni od gruč
  - ponavljaj do konvergence (dokler se pripadnost gručam spreminja):
    - za vsako izmed k gruč izračunaj **centroid** (točka, določena s srednjo vrednostjo atributov vseh primerov, ki pripadajo gruči)
    - spremeni pripadnost vsakega primera tisti gruči, katere centroid je najbližji (glede na izbrano mero razdalje)
- primer: različne rešitve za $k = 2, k = 3$ in $k = 4$

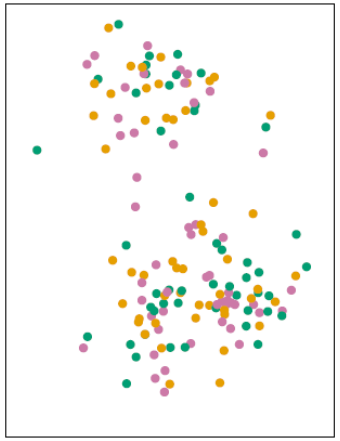


Primer izvajanja

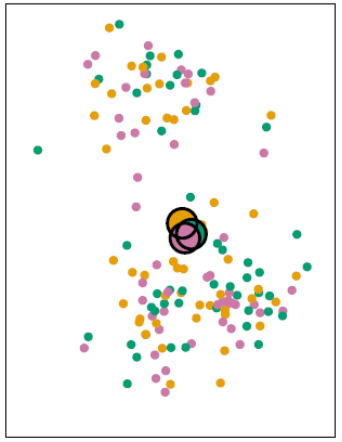
učna množica



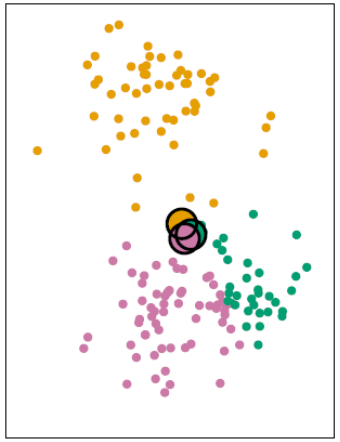
naključna določitev gručam



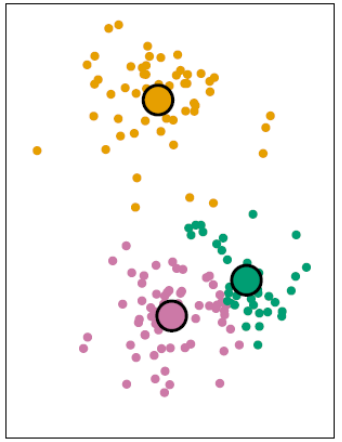
izračun začetnih centroidov gruč



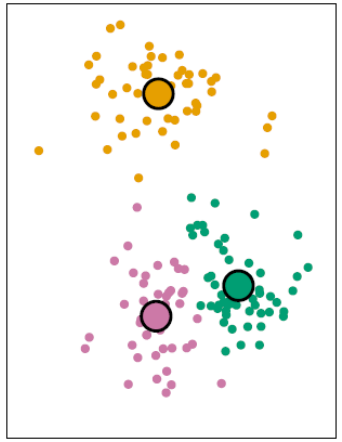
določitev pripadnosti primerov najbližjemu centroidu



ponovni izračun centroidov gruč

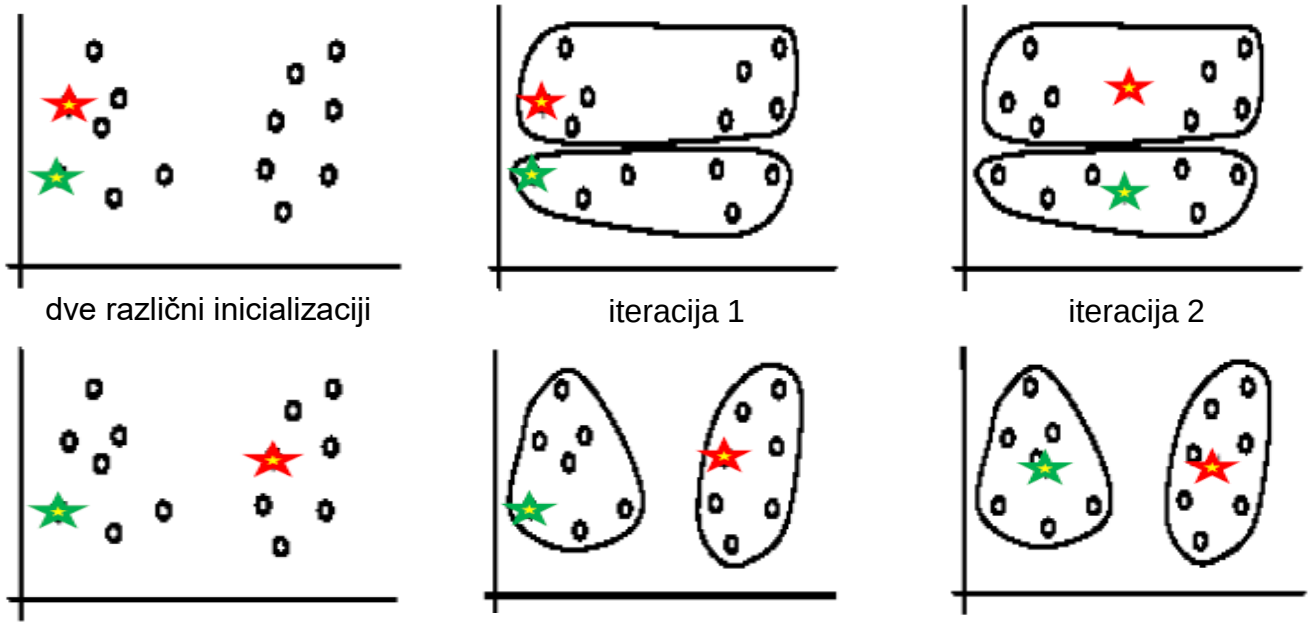


rezultat po 10 iteracijah



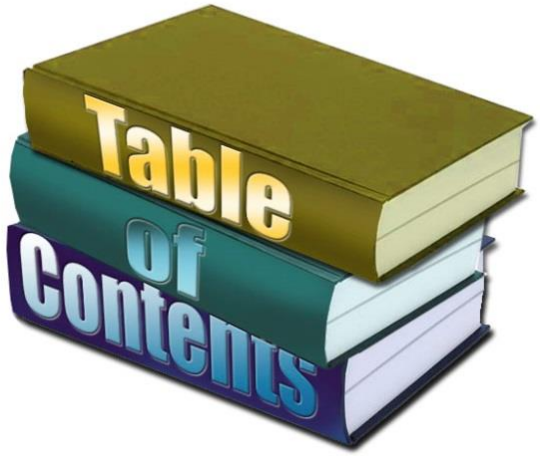
Lastnosti algoritma

- metoda na vsakem koraku znižuje varianco znotraj gruč (s prerazporejanjem pripadnosti primerov najbližji gruči)
- algoritem ne najde globalnega optimuma (glede na cilj minimizacije variance znotraj gruč), rešitev je odvisna od začetne inicializacije
- izračun centroidov je potrebno prilagoditi, če so atributi diskretni
- algoritem je občutljiv na šum (angl. *outliers*)
- težka objektivna evalvacija, vendar uporabno v praksi



II. PREISKOVANJE

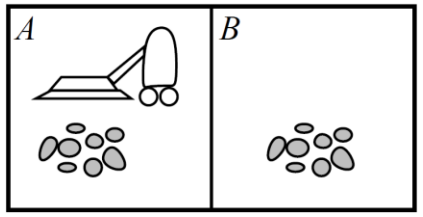
Pregled

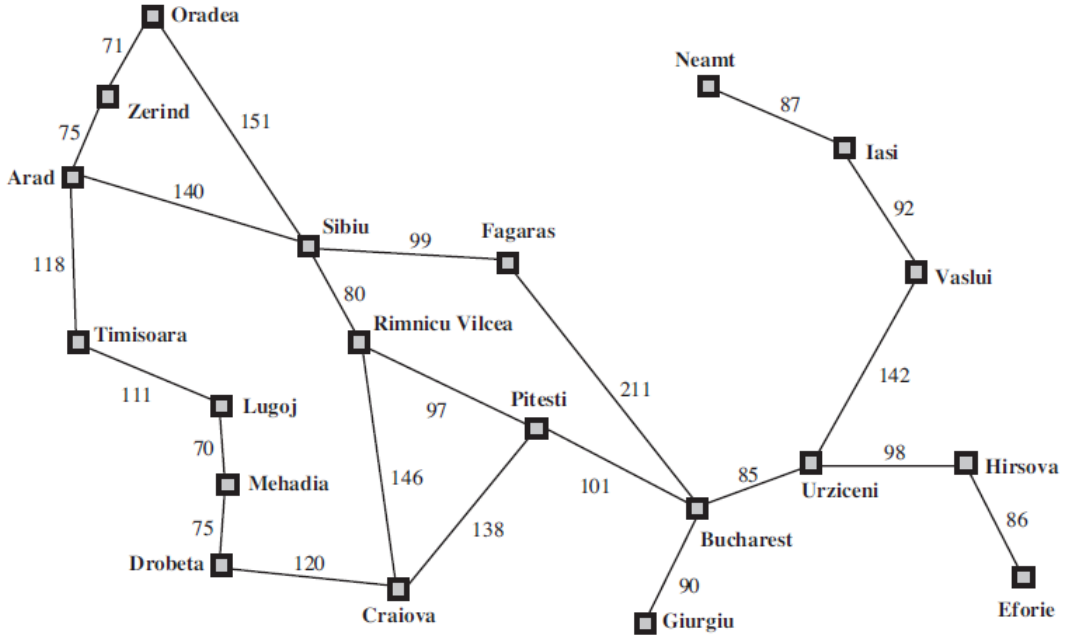


- preiskovanje prostora stanj
  - definicija in cilji področja
  - primeri umetnih in realnih problemov
  - neinformirani preiskovalni algoritmi
    - iskanje v širino
    - iskanje v globino
    - iterativno poglabljanje
    - dvosmerno iskanje
    - cenovno – optimalno iskanje

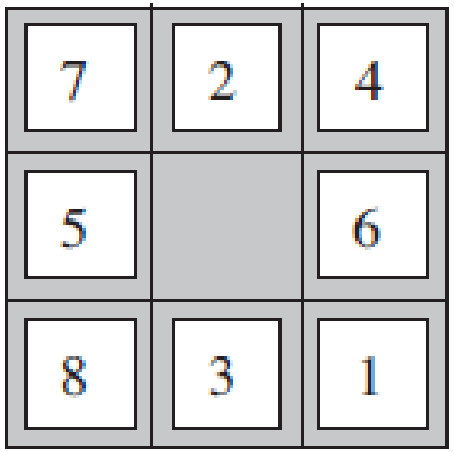
Primeri problemov

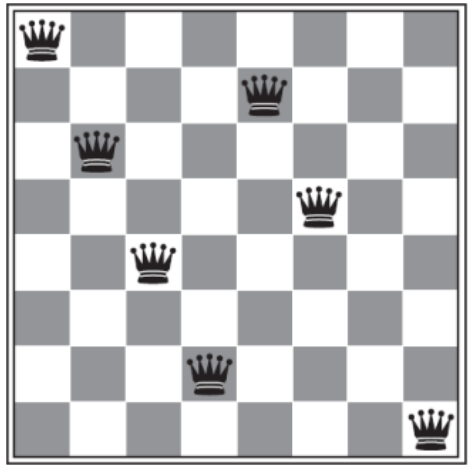
- avtomatski sesalec
- iskanje najkrajše poti
- igra 8 ploščic
- 8 kraljic na šahovnici
- planiranje v svetu kock
- manevriranje robotov
- trgovski potnik





- potrebujemo:
  1. način za abstrakcijo problema
  2. simbolično predstavitev (graf, drevo)
  3. uporabo algoritma na simbolični predstavitvi za iskanje poti do cilja





Definicija problema

Za formalni opis problema potrebujemo:

- **začetno stanje**
- opis vseh možnih **akcij**, ki so na razpolago v posameznih stanjih
- **prehodno funkcijo**, ki definira naslednika stanja
$$rezultat(trenutno\_stanje, akcija) = novo\_stanje$$
- **ciljni** predikat:
$cilj(stanje) = \{true, false\}$
- **cenilna funkcija**, ki vsaki poti določi ceno (numerično vrednost)
$$c(stanje, akcija, stanje') \geq 0$$

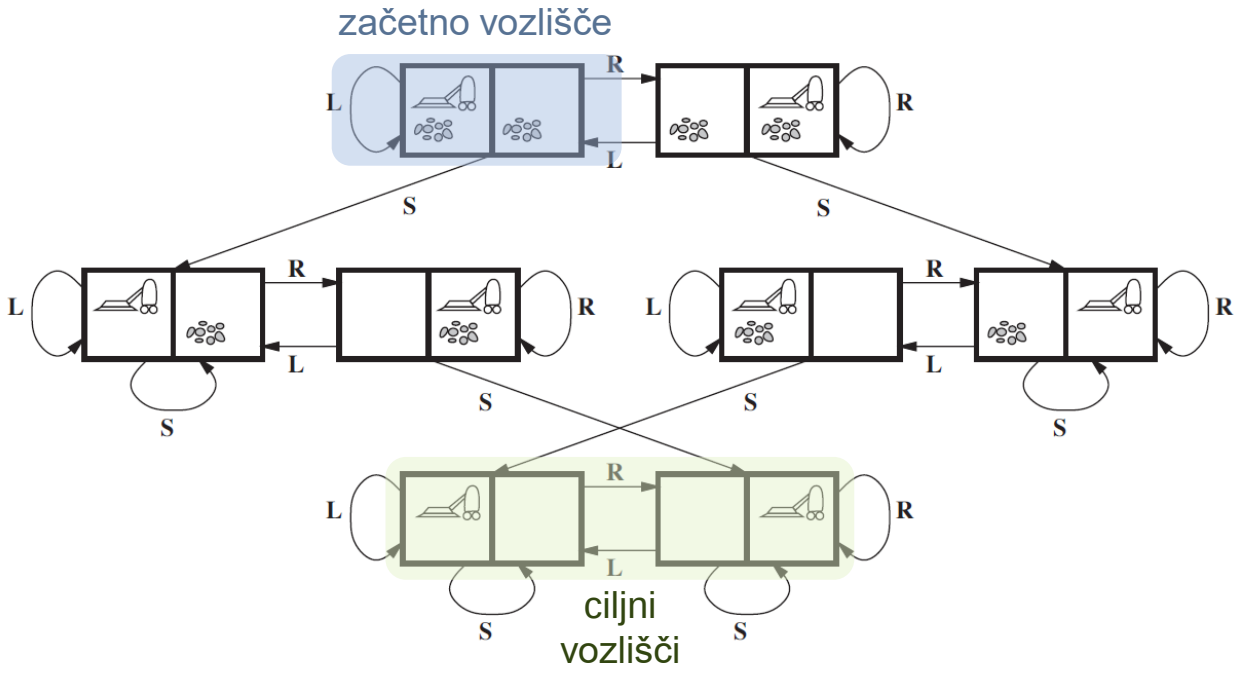
Definicija problema

- opisani način podajanja problema omogoča, da problem predstavimo z usmerjenim grafom, ki ponazarja ***prostor stanj***:
  - **vozlišča**: stanja (problemske situacije)
  - **povezave**: akcije (dovoljene poteze)
  - **pot** v grafu: zaporedje stanj, ki ga povezuje zaporedje akcij
  - eno **začetno** vozlišče
  - eno ali več **ciljnih** vozlišč

- reševanje problema – **iskanje poti v grafu s preiskovanjem**

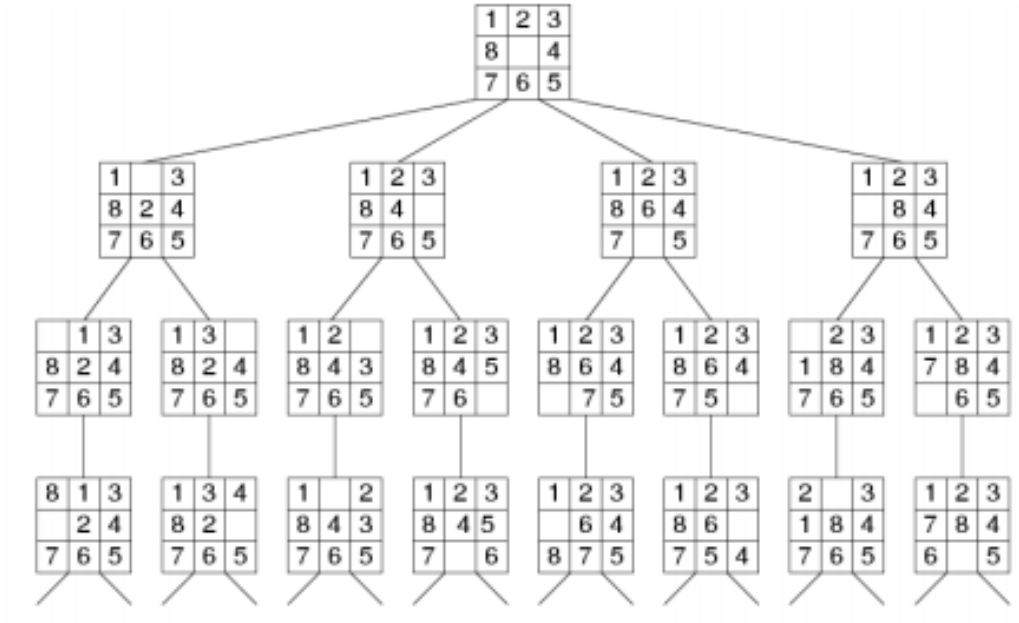
- **rešitev problema** – zaporedje akcij (pot), ki vodi od začetnega do ciljnega vozlišča
  - optimalna rešitev problema izmed vseh možnih ima najnižjo ceno poti

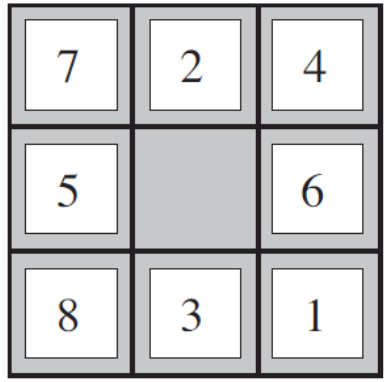
Primer: sesalec



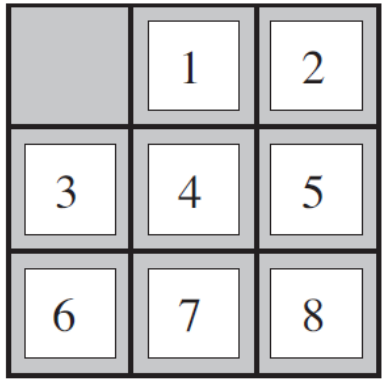
- **stanja** – kombinacija lokacije robota in umazanije v prostorih A in B ($2 \times 2^2 = 8$ stanj)
- **akcije** – levo, desno, sesaj (in počakaj?)
- **cilj** – oba prostora čista
- **cena akcije** – npr. 1 za vsako potezo

Primer: igra 8 ploščic





začetno stanje

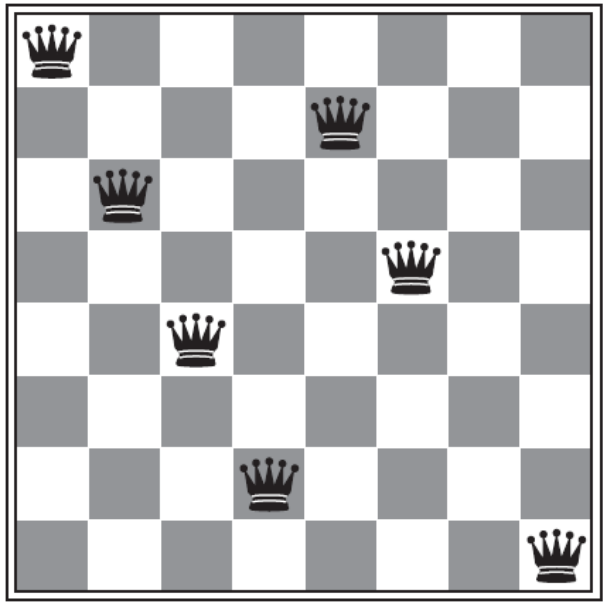


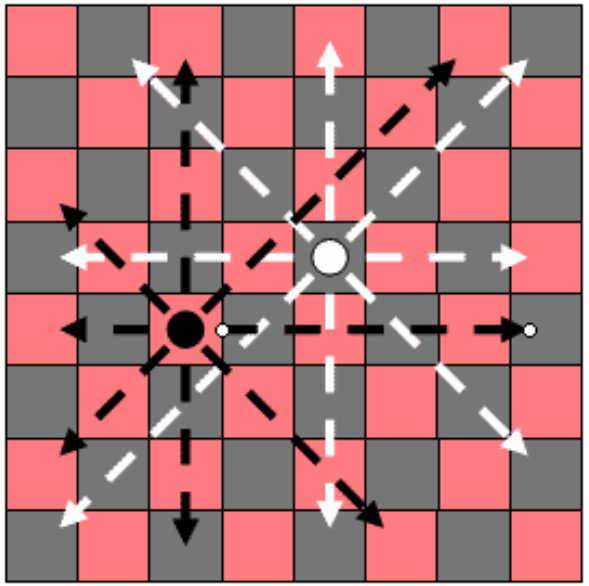
končno stanje

- **stanja** – lokacije ploščic
- **akcije** – premiki "prazne" ploščice levo, desno, gor, dol
- **cilj** – podano ciljno stanje
- **cena akcije** – 1 za vsako potezo

- problem je NP-poln
- igra 3x3 ima 9!/2 =181.440 dosegljivih stanj
- igra 4x4 ima okoli $1{,}3 \times 10^{12}$ dosegljivih stanj
- igra 5x5 ima okoli 10^{25} dosegljivih stanj

Primer: problem 8 kraljic





- **stanja** – postavitve 0-8 kraljic na plošči ($64 \times 63 \times \cdots \times 57 = 1.8 \times 10^{14}$) stanj
- **začetno stanje** – prazna igralna plošča
- **akcije** – postavitve nove kraljice na prazno polje
- **cilj** – 8 kraljic na plošči, nobeni dve se ne napadata

bolj optimalna formulacija problema:

- stanja – postavitve 0-8 kraljic od leve proti desni, v vsak stolpec ena (2057 stanj)
- akcije – postavitve nove kraljice na prazno polje v prvi skrajno levi prosti stolpec

Primer: Knuthov neskončni prostor

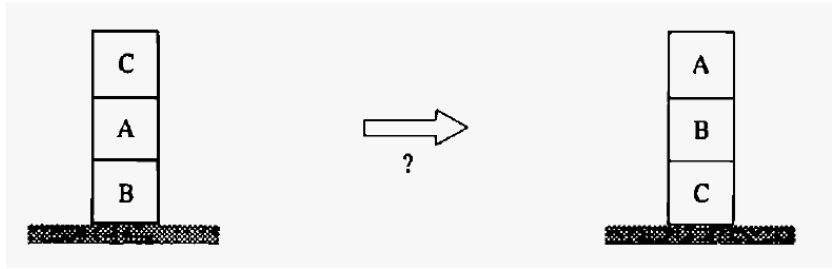
- Knuthova predpostavka: začenši s številom 4 in zaporedjem treh operacij (faktoriela, kvadratni koren, zaokroževanje navzdol), lahko dosežemo poljubno pozitivno celo število. Npr. število 5 lahko zapišemo kot:

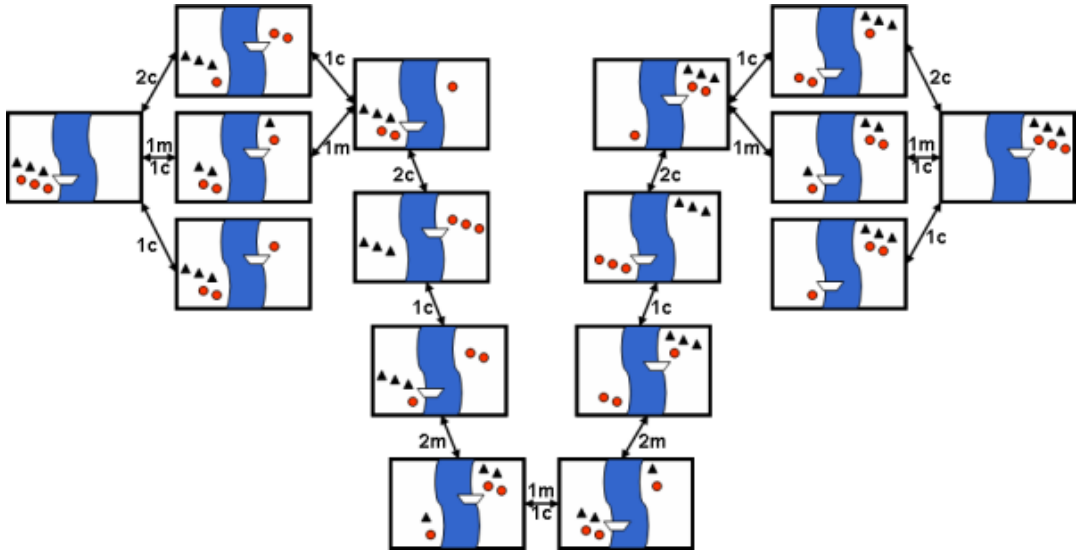
$$5 = \left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}}} \right\rfloor$$

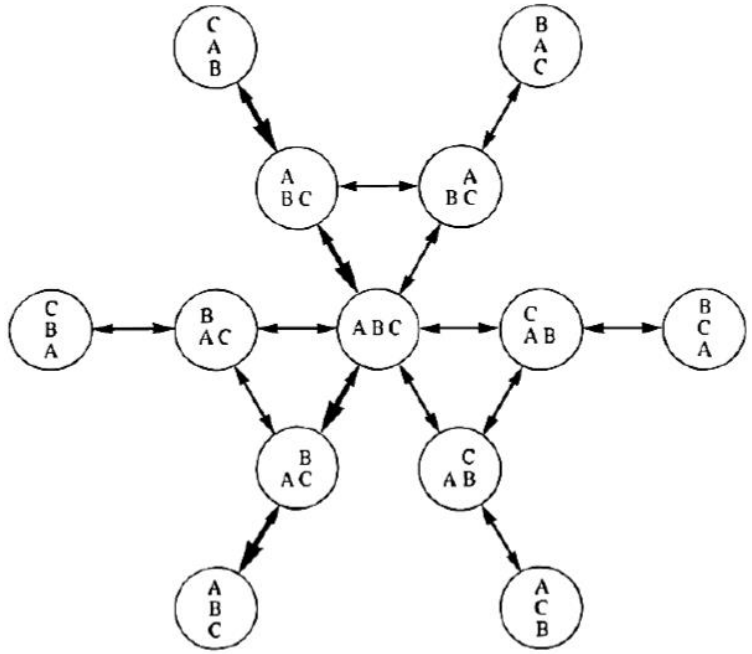
- **začetno stanje** – število 4
- **akcije** – uporaba ene izmed operacij *{faktoriela, kvadratni koren, zaokroževanje navzdol}*
- **stanja** – pozitivna števila
- **ciljno stanje** – želeno pozitivno število

Primeri drugih umetnih problemov

- misijonarji in ljudožerci
- preurejanje postavitve kock

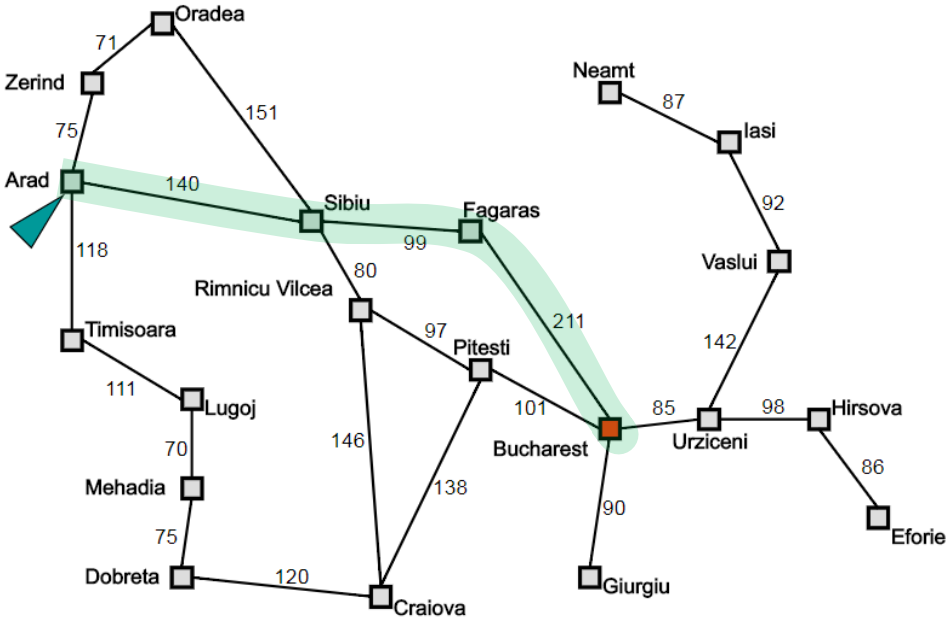






Primeri nekaterih realnih problemov

- iskanje najkrajše poti
- primer obhoda ("obišči vsa mesta vsaj enkrat, začenši v mestu X")
- problem potujočega trgovca - TSP ("najdi najkrajšo pot za obhod vseh mest natanko enkrat, začenši v mestu X")
- postavitev komponent na vezju (minimizacija površine, dolžine povezav itd.)
- navigacija robotov v prostoru
- sestavljanje izdelkov na proizvodni liniji

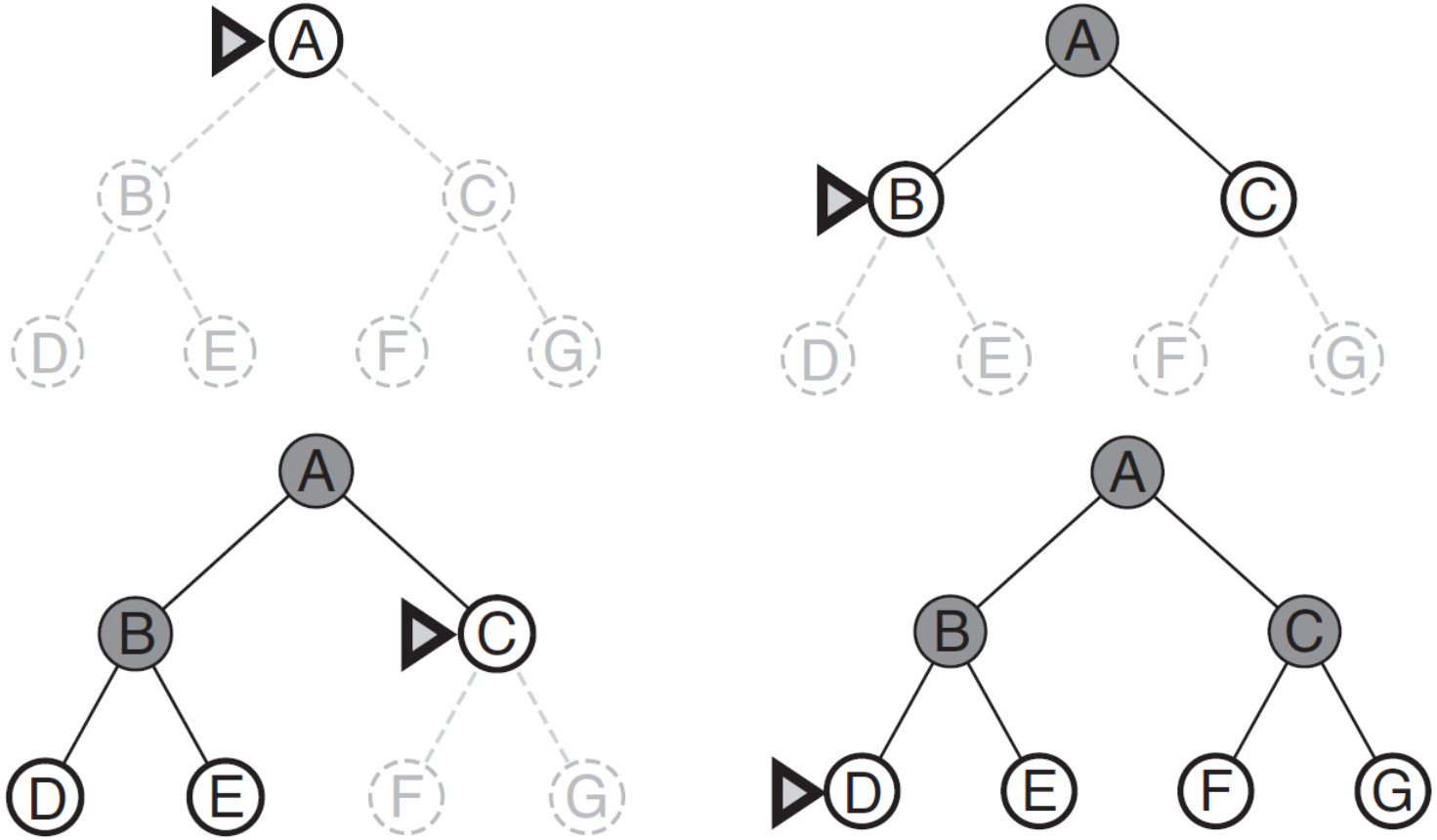


Preiskovanje

- problem: velika kombinatorična eksplozija možnih stanj
- preiskovalni algoritmi:
  - **neinformirani**: razpolagajo samo z definicijo problema
    - iskanje v širino (*breadth-first search*)
    - iskanje v globino (*depth-first search*)
    - iterativno poglabljanje (*iterative deepening*)
    - cenovno-optimalno iskanje (*uniform-cost search*)
  - **informirani**: razpolagajo tudi z dodatno informacijo (domensko znanje, hevristične ocene), kako bolj učinkovito najti rešitev
    - algoritem A*
    - algoritem IDA*
    - prioritetno preiskovanje (*best-first search*)
    - algoritem RBFS (*recursive best-first search*)
    - plezanje na hrib (*hill climbing*)
    - iskanje v snopu (*beam search*)
    - …

Iskanje v širino

- angl. *Breadth-First Search (BFS)*
- strategija: vedno razvij najbolj **plitvo še nerazvito** vozlišče
  - implementacija: razvita vozlišča dodamo v vrsto (FIFO) za razvijanje



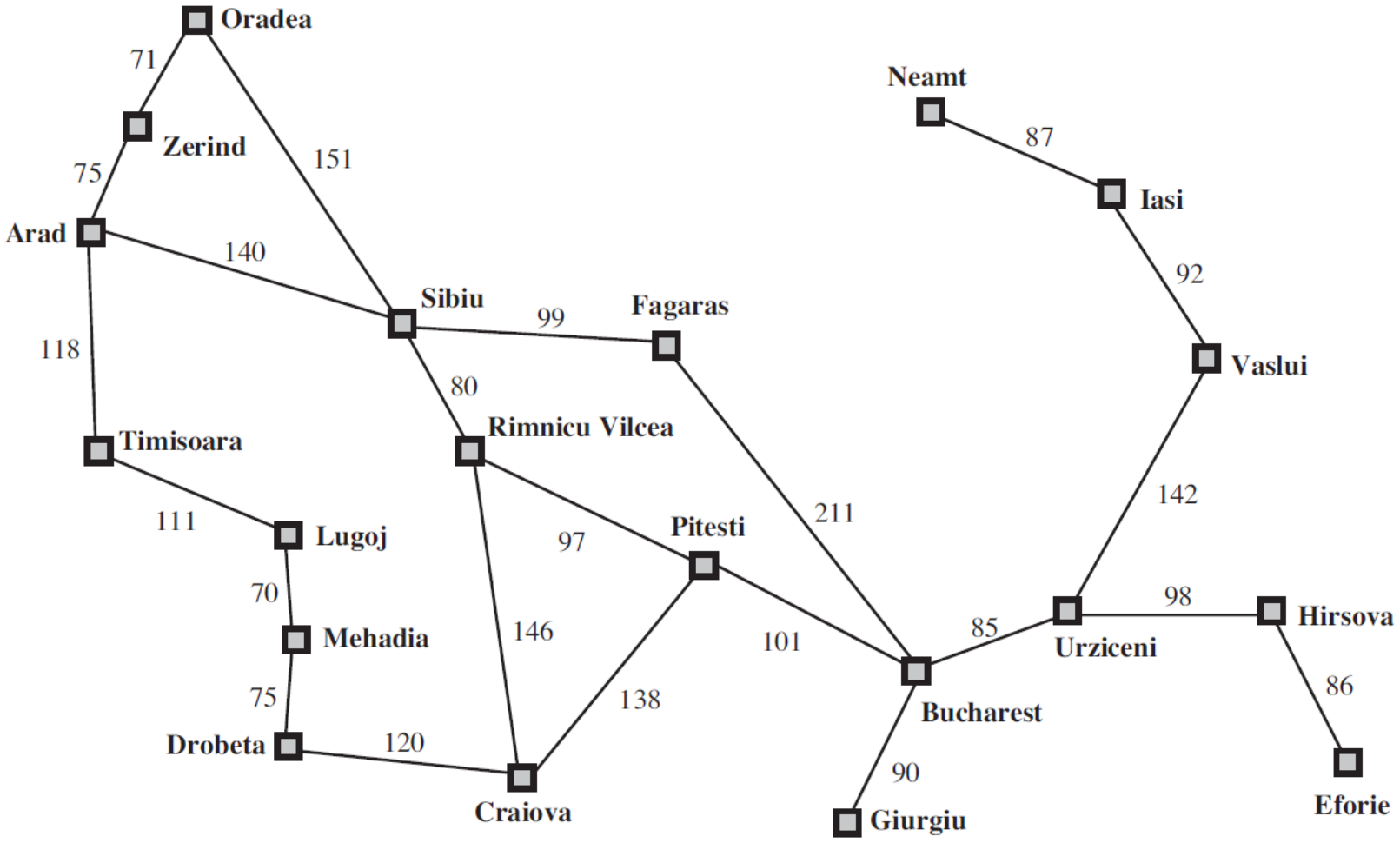
Iskanje v širino

primer iz: Russell, Norvig, Artificial Intelligence: A Modern Approach, 3. izd., Pearson, 2010.

Rosamund Angel Watts, http://slideplayer.com/slide/10602451/

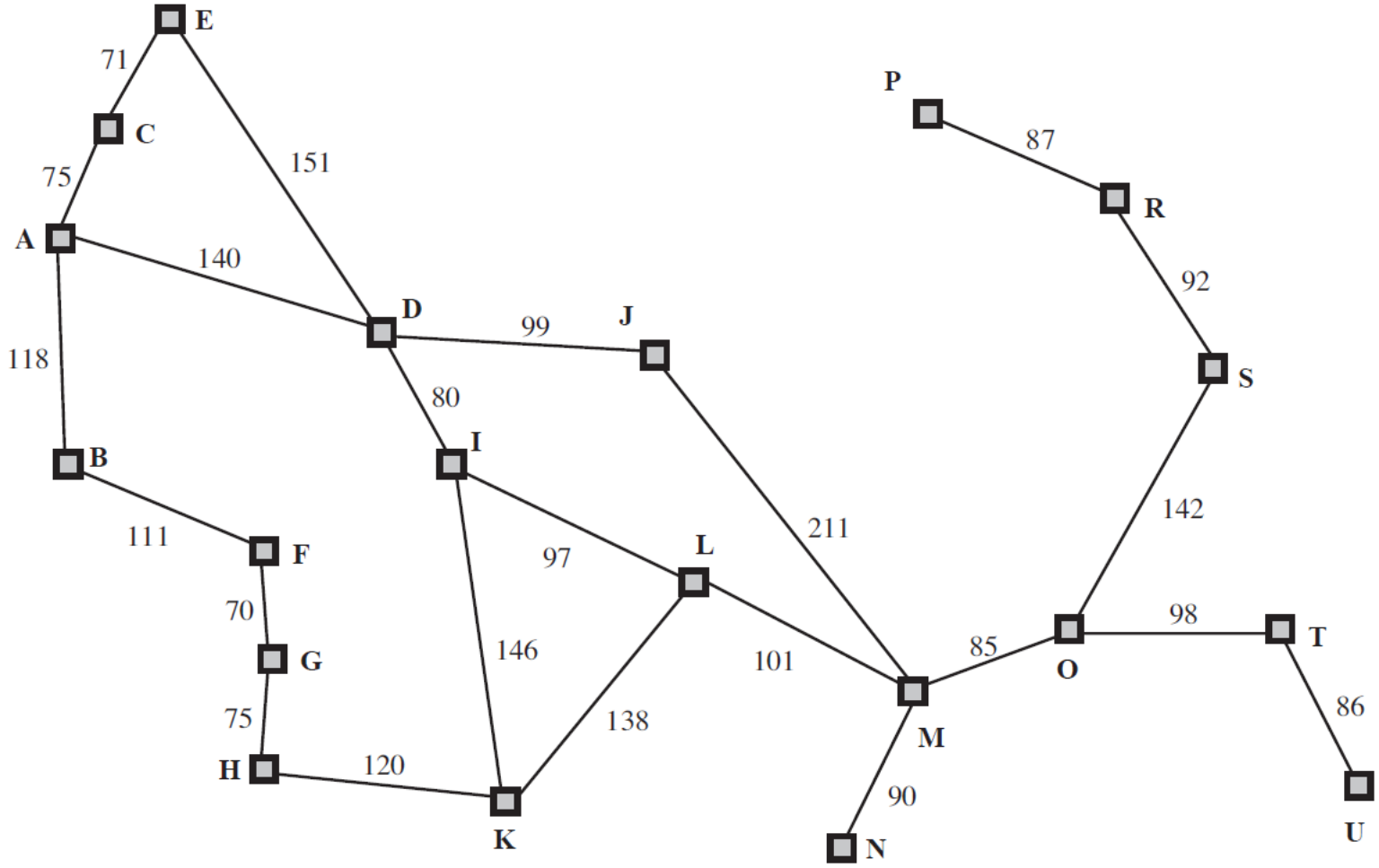
Iskanje v širino

primer iz: Russell, Norvig, Artificial Intelligence: A Modern Approach, 3. izd., Pearson, 2010.



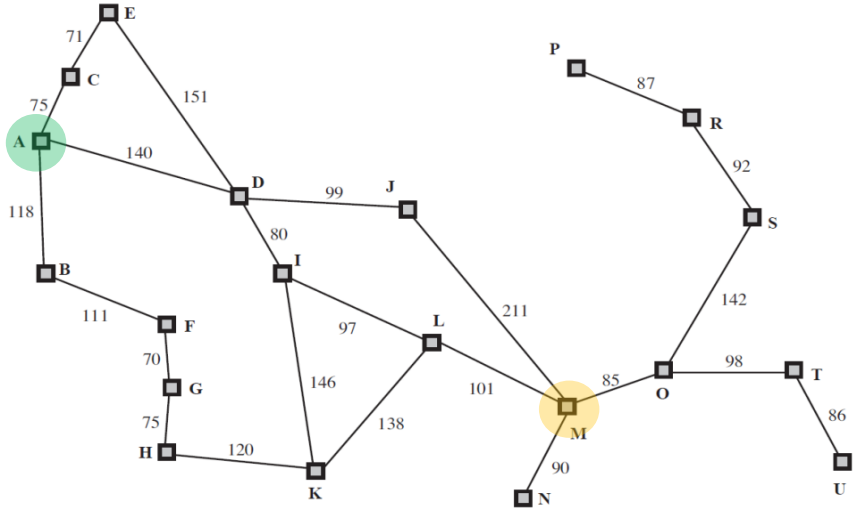
Iskanje v širino

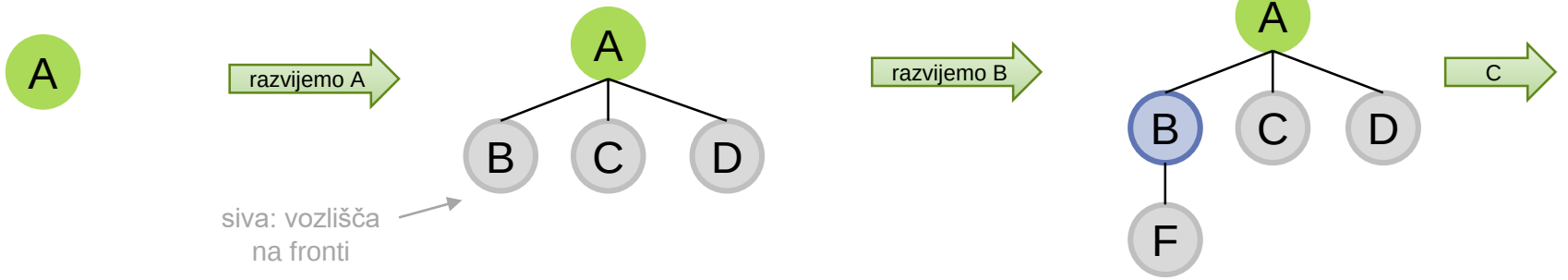
primer iz: Russell, Norvig, Artificial Intelligence: A Modern Approach, 3. izd., Pearson, 2010.

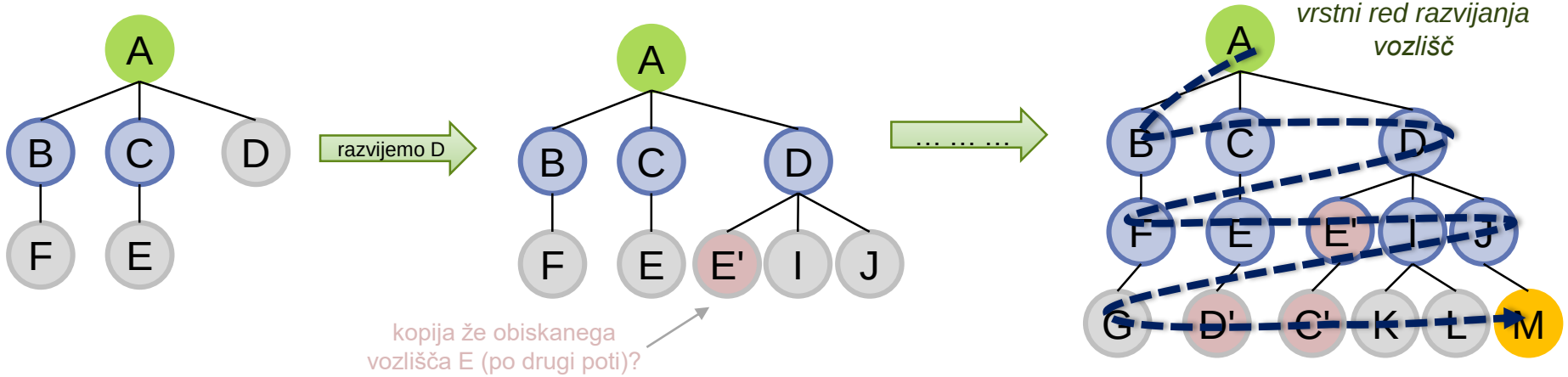


Iskanje v širino

- strategija: vedno razvij najbolj plitvo še nerazvito vozlišče
- odločimo se hraniti kopije že obiskanih vozlišč (zakaj?)
- vozlišča, ki sklenejo cikel, takoj zavržemo (zakaj, je nujno?)

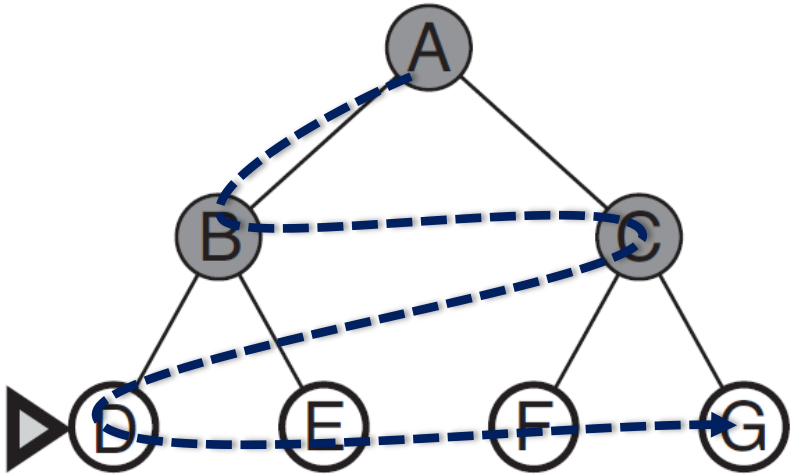






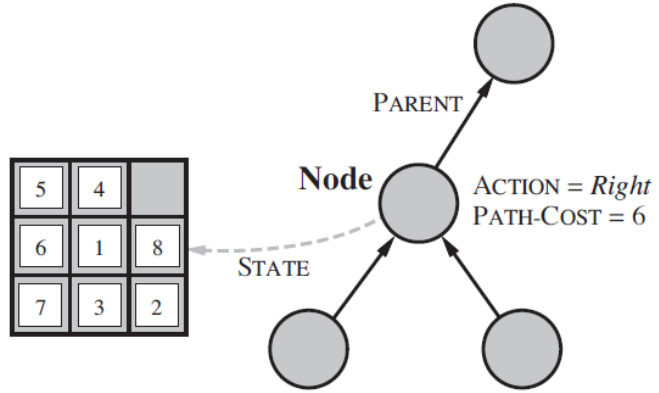
Iskanje v širino

- pojmi:
  - **razvijanje** vozlišča – **generiranje** naslednikov
  - **fronta** – listi drevesa, ki so kandidati za razvijanje (**FIFO vrsta**)
- generira celoten nivo preden se premakne na naslednji nivo
- v spominu **mora pomniti vse alternativne poti**, ki so kandidati, da bodo podaljšane do ciljnega vozlišča
- zagotavlja, da najdemo **najkrajšo rešitev** (A-D-J-M)
  - pozor, v grafu so možni cikli (več poti vodi do ciljnega vozlišča M)
- možne **implementacije** in **pomisleki**:
  - **detekcija ciljnega vozlišča**, dve možnosti:
    - ciljno vozlišče zaznamo šele, ko ga želimo razviti
    - ciljno vozlišče zaznamo, že ko ga generiramo (bolj optimalno)
  - **hranjenje kopij** že videnih vozlišč, da ali ne?
    - ne – pri BFS ni smiselno, ker imajo povezave enako ceno
    - da – kadar nas druga pot lahko pripelje do bolj optimalne rešitve (podane so različne cene povezav, potrebujemo algoritem, ki upošteva tudi cene na povezavah in ne samo vrstni red obiskovanja vozlišč; kasneje…)



Programska implementacija

- vozlišča v drevesu hranijo:
  - STANJE: opis stanja v problemskem prostoru
  - PREDHODNIK: kazalec na predhodnika
  - AKCIJA: akcija, ki je iz predhodnika generirala vozlišče
  - CENA: cena poti od začetnega do trenutnega vozlišča

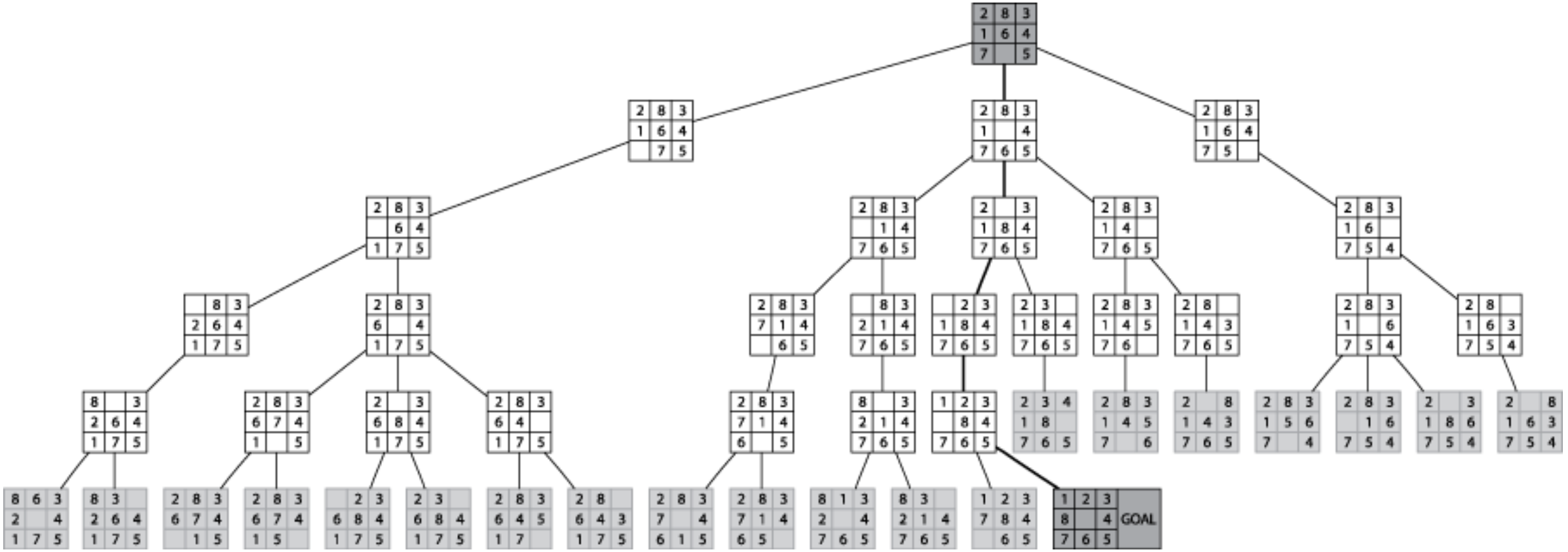



```
function iskanje(problem) {
    vozlišče <- problem.začetno_stanje, cena=0
    if (cilj(vozlišče.stanje)) return rešitev=vozlišče
    fronta <- vozlišče
    obiskana <- ∅
    while(true) {
        if (fronta== ∅) return rešitev= ∅
        vozlišče <- izberi(fronta)
        fronta <- fronta - vozlišče
        obiskana <- obiskana + vozlišče.stanje
        foreach naslednik in nasledniki(vozlišče) {
            if ((naslednik.stanje ∉ obiskana) && (naslednik ∉ fronta))
                    if (cilj(naslednik.stanje) return rešitev=naslednik
                    fronta <- fronta + naslednik
    }
}
```

način dela s fronto določa strategijo preiskovalnega algoritma

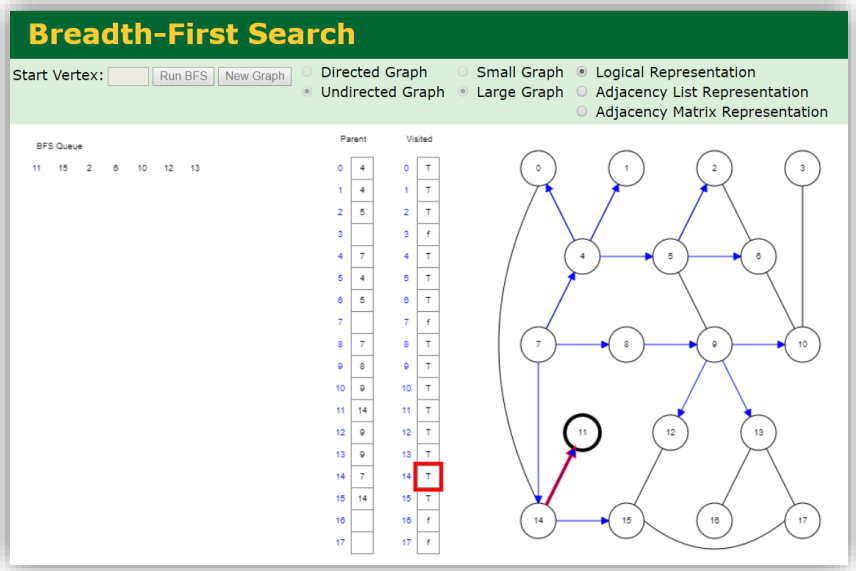
hranjenje obiskanih stanj, do katerih lahko pridemo po različnih poteh, omogoča preprečitev ciklanja preiskovanja

Primer: igra 8 ploščic

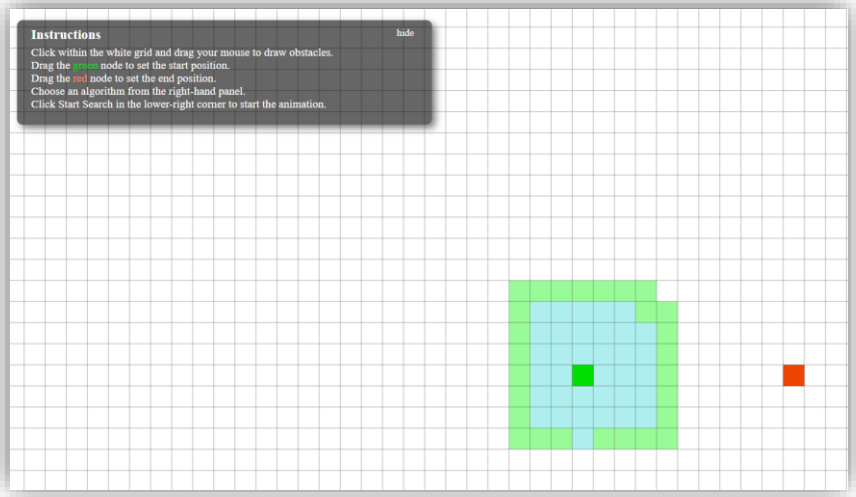


Demo

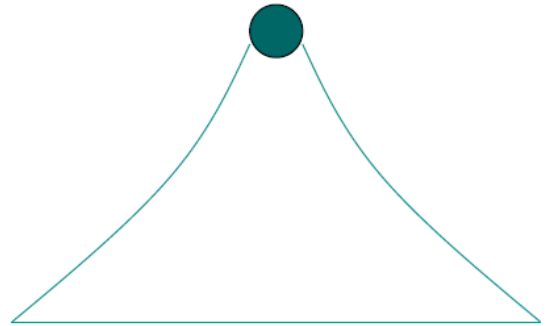
- Data Structure Visualizations
  https://www.cs.usfca.edu/~galles/visualization/BFS.html



- PathFinding
  https://qiao.github.io/PathFinding.js/visual/



Učinkovitost iskanja v širino



- za potrebe analize predpostavimo, da je prostor stanj drevo višine $\boldsymbol{d}$ (*depth*) in stopnje vejanja $\boldsymbol{b}$ (*branching factor*)
  - na nivoju d imamo torej b^d vozlišč

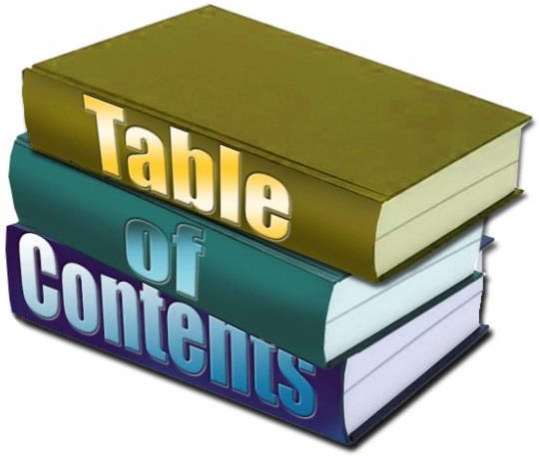
- **POPOLNOST** (angl. *completeness*):
  Ali algoritem zagotovo najde rešitev, če le-ta obstaja?
  - DA! (če je le b končen)
- **OPTIMALNOST** (angl. *optimality*):
  Ali iskanje najde optimalno (najboljšo možno rešitev)?
  - BFS da (če je optimalna najkrajša pot), vendar v splošnem (cena povezav ni 1) to ni nujno.
- **ČASOVNA ZAHTEVNOST**:
  - generirano število vozlišč je $b + b^2 + b^3 + \cdots + b^d = O(b^d)$
  - torej: eksponentna časovna zahtevnost glede na d
- **PROSTORSKA ZAHTEVNOST**:
  - hraniti mora $O(b^{d-1})$ razvitih vozlišč in $O(b^d)$ v fronti – skupaj $O(b^d)$

Zahtevnost iskanja v širino

- primer za faktor vejanja $b = 10$, hitrost generiranja 10^6 vozlišč/sekundo, potreben prostor 1000 bajtov/vozlišče

Globina	Vozlišč	Čas	Spomin
2	110	0,11 ms	107 kB
4	11.110	11 ms	10,6 MB
6	10^6	1,1 s	1 GB
8	10^8	2 minuti	103 GB
10	10^{10}	3 ure	10 TB
12	10^{12}	13 dni	1 PB
14	10^{14}	3,5 let	99 PB
16	10^{16}	350 let	10 EB

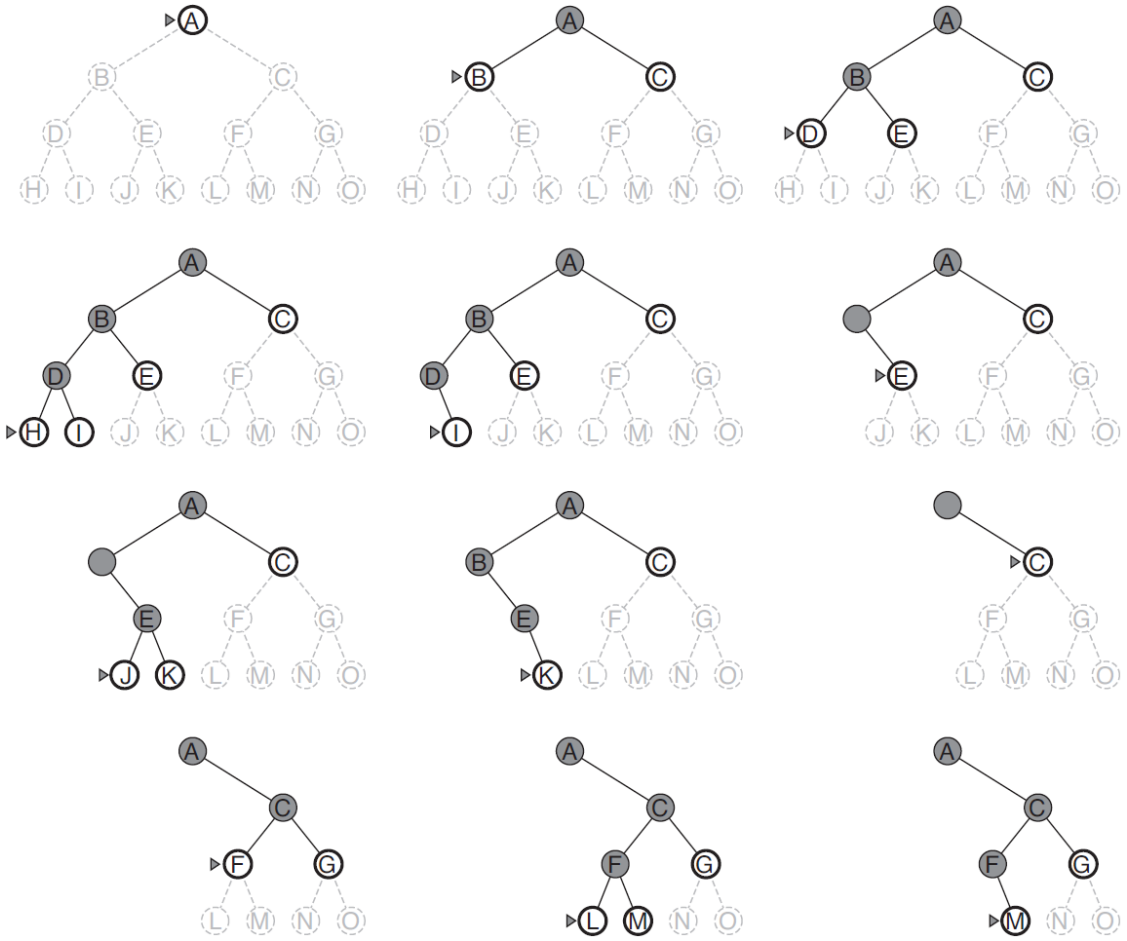
Pregled



- preiskovanje prostora stanj
  - definicija in cilji področja
  - primeri umetnih in realnih problemov
  - neinformirani preiskovalni algoritmi
    - iskanje v širino
    - iskanje v globino
    - iterativno poglabljanje
    - dvosmerno iskanje
    - cenovno – optimalno iskanje

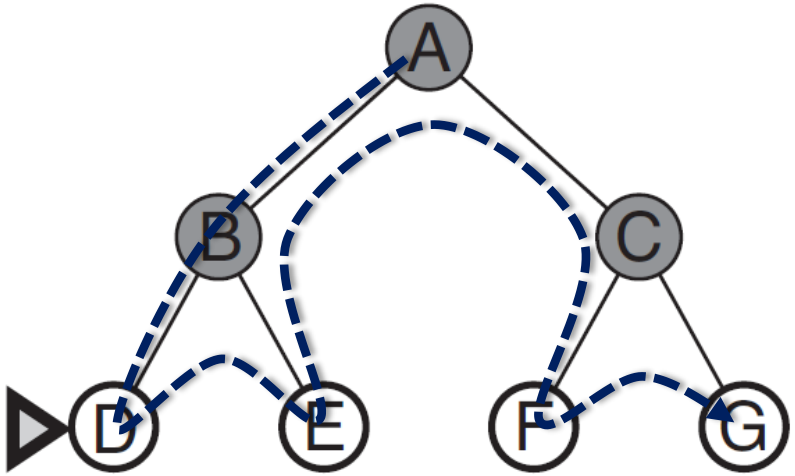
Iskanje v globino

- angl. *Depth-First Search (DFS)*
- strategija: najprej razvij **najglobje še nerazvito** vozlišče
  - implementacija: naslednike dodaj v sklad (LIFO) za razvijanje



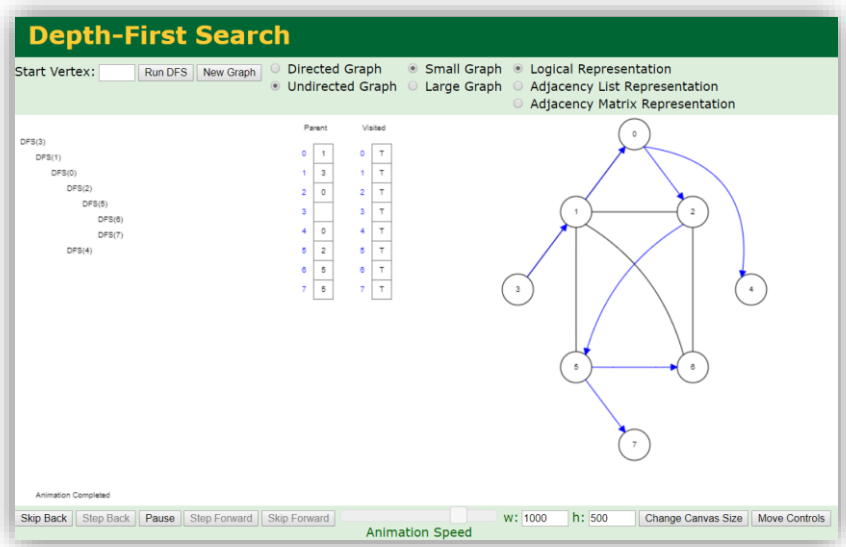
Iskanje v globino

- ne zagotavlja najkrajše rešitve
- nujno potrebno preprečevanje ciklov, možnost neskončno globokega prostora (neskončna globina)
- možna je preprosta implementacija z rekurzijo
- možna optimizacija: raziskanih vej nam ni treba hraniti v spominu (hranimo le pot od začetnega do trenutnega vozlišča)



Demo

- Data Structure Visualizations
  https://www.cs.usfca.edu/~galles/visualization/DFS.html



Učinkovitost iskanja v globino

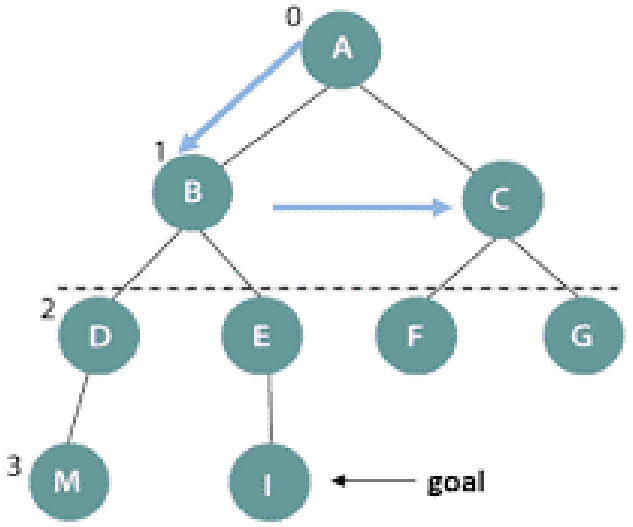
- za potrebe analize predpostavimo, da je prostor stanj drevo:
  - globina (*depth*) optimalne rešitve naj bo ***d***
  - stopnja vejanja (*branching factor*) naj bo ***b***
    na nivoju d imamo torej b^d vozlišč
  - največja globina drevesa naj bo **max**

- **POPOLNOST** (angl. *completeness*):
  - Neuspešen v prostorih z zankami in neskončno globino.
- **OPTIMALNOST** (angl. *optimality*):
  - Ne.
- **ČASOVNA ZAHTEVNOST**:
  - generirano število vozlišč $O(b^{max})$
- **PROSTORSKA ZAHTEVNOST**:
  - hraniti mora samo $O(bm)$ razvitih vozlišč (linearna prostorska zahtevnost!)

Iskanje v globino - izboljšave

- iskanje s sestopanjem (*backtracking search*):
  - namesto vseh naslednikov generiramo samo enega po enega
  - ➔ prostorska zahtevnost $O(m)$

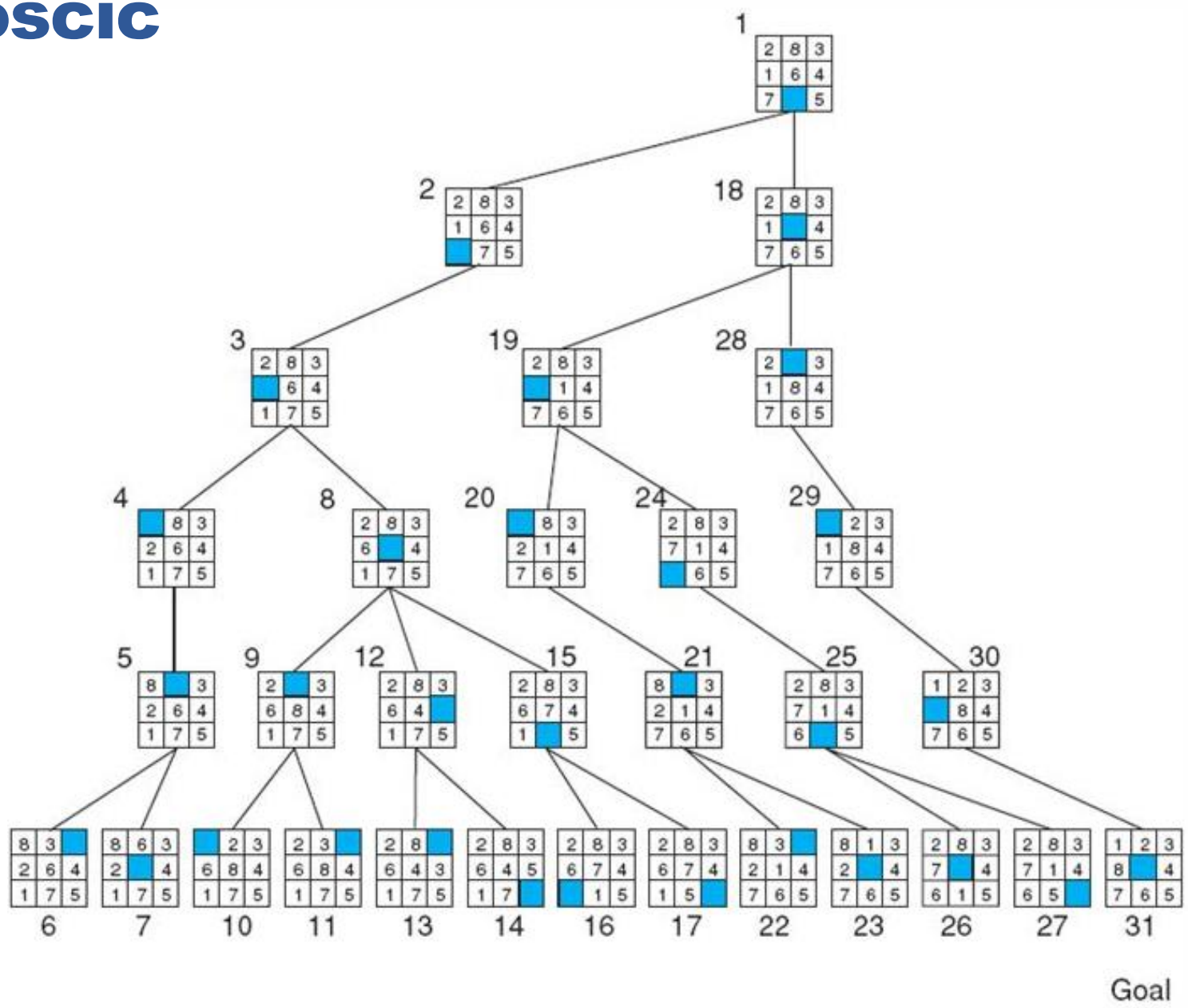
- iskanje z omejitvijo globine (*depth-limited search*):
  - vnaprej definiramo mejo globine l
  - vozlišča na globini l obravnavamo, kot da nimajo naslednikov
  - če izberemo $l < d$, je algoritem nepopoln (ne najde rešitve)
  - če izberemo $l > d$, je algoritem popoln, a neoptimalen
  - časovna zahtevnost je $O(b^l)$, prostorska pa $O(bl)$
  - pri določitvi l pomaga domensko znanje



- iterativno poglabljanje (angl. I*terative Deepening Search (IDS)*)

Primer: igra 8 ploščic

- iskanje z omejitvijo globine (d=6)

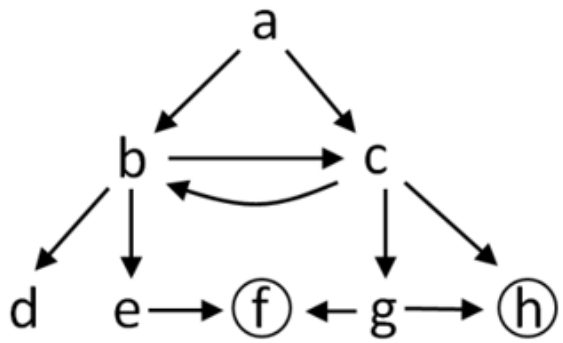


Naloga

- 1. izpit, 30. 1. 2018, 1. naloga

1. NALOGA:

Naj bo a začetno vozlišče preiskovanja, f in h pa sta ciljni vozlišči. Algoritmi preiskovanja naj generirajo naslednike vozlišč po abecednem vrstnem redu. Npr. vrstni red naslednikov vozlišča c je: b, g, h. Vsi preiskovalni algoritmi preverjajo pogoj, ali je dano vozlišče ciljno šele ob času, ko se lotijo razvijanja tega vozlišča. Vsi preiskovalni algoritmi naj tudi razpoznavajo cikle in generirano vozlišče, ki sklene cikel, takoj zavržejo. Vendar pa obravnavajo graf kot drevo. Torej, če pridejo do kakega vozlišča N po različnih poteh, naredijo kopijo N' vozlišča N in obravnavajo N', kot da bi bilo novo vozlišče. Če imata dve vozlišči enako f-oceno, se najprej razvije tisto vozlišče, ki je bilo prej generirano.



a) Katero rešitveno pot vrne **iskanje v globino**? Kakšen je vrstni red generiranih vozlišč?

b) Katero rešitveno pot vrne **iskanje v širino**? Kakšen je vrstni red generiranih vozlišč?

Neinformirani algoritmi