OSNOVE UMETNE INTELIGENCE
2018/19

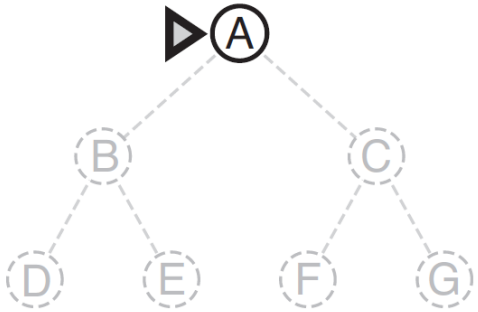
neinformirani preiskovalni algoritmi

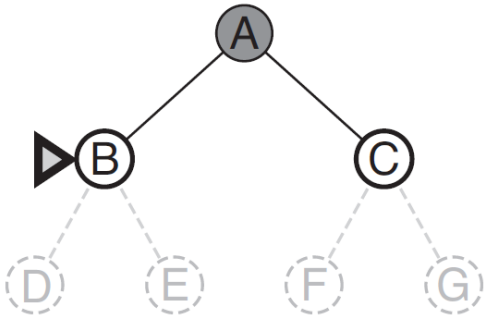
Preiskovanje

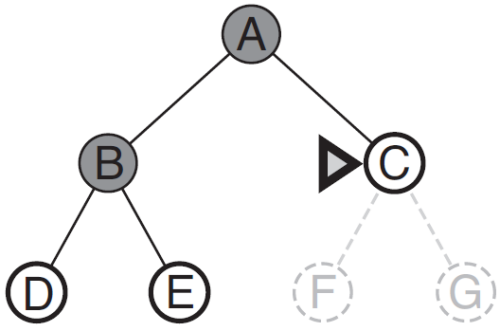
- problem: velika kombinatorična eksplozija možnih stanj
- preiskovalni algoritmi:
  - **neinformirani**: razpolagajo samo z definicijo problema
    - iskanje v širino (*breadth-first search*)
    - iskanje v globino (*depth-first search*)
    - iterativno poglabljanje (*iterative deepening*)
    - cenovno-optimalno 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*)
    - …

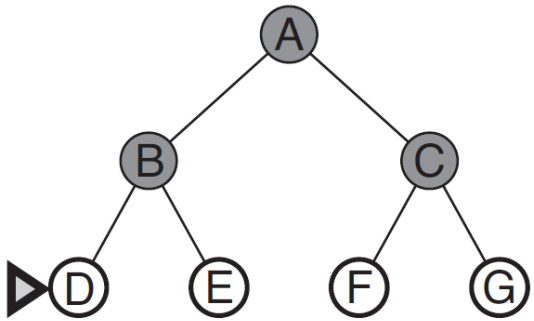
Iskanje v širino

- strategija: vedno razvij najbolj **plitvo še nerazvito** vozlišče
  - implementacija: razvita vozlišča dodamo v vrsto (FIFO) za razvijanje









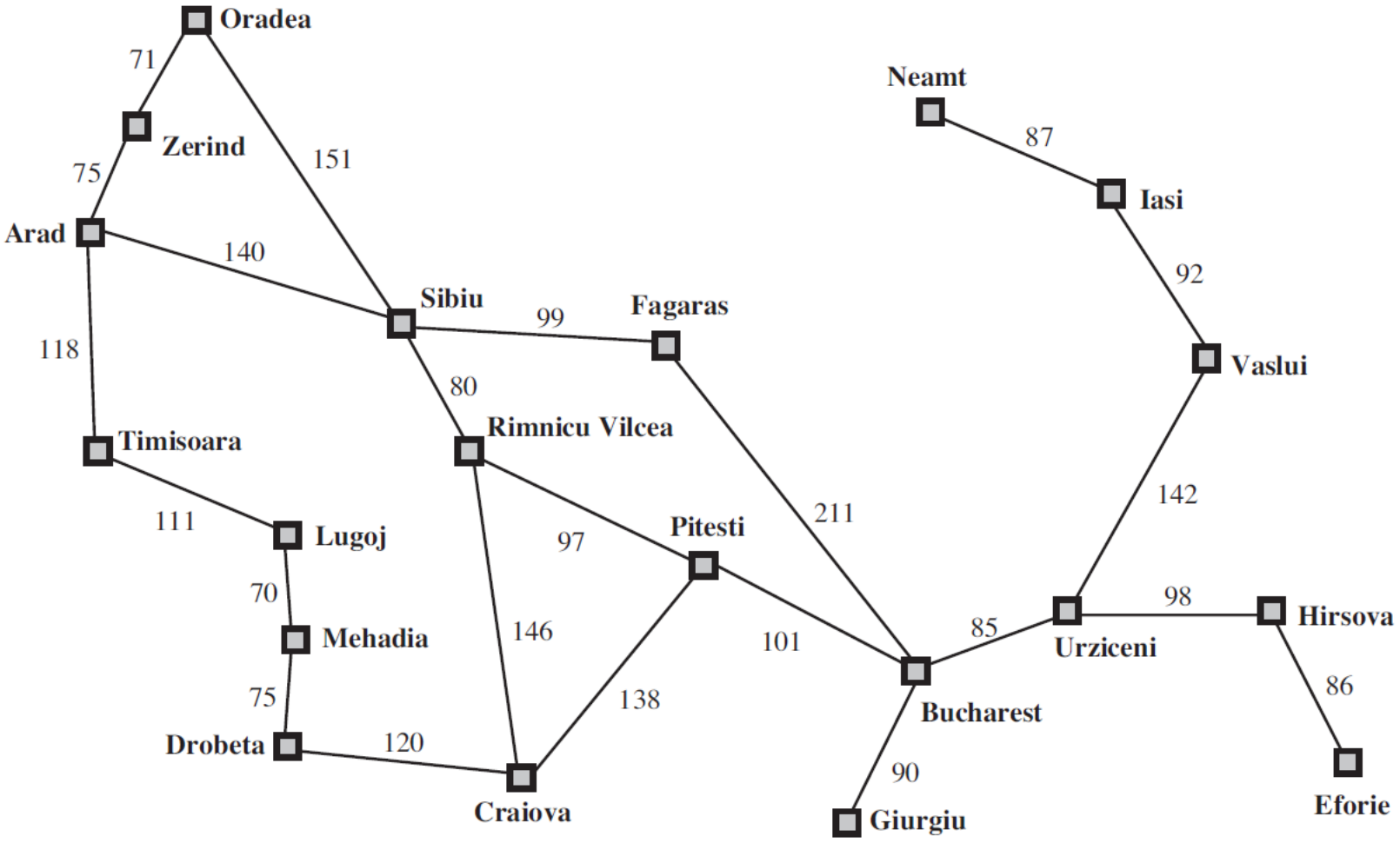
Iskanje v širino

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

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

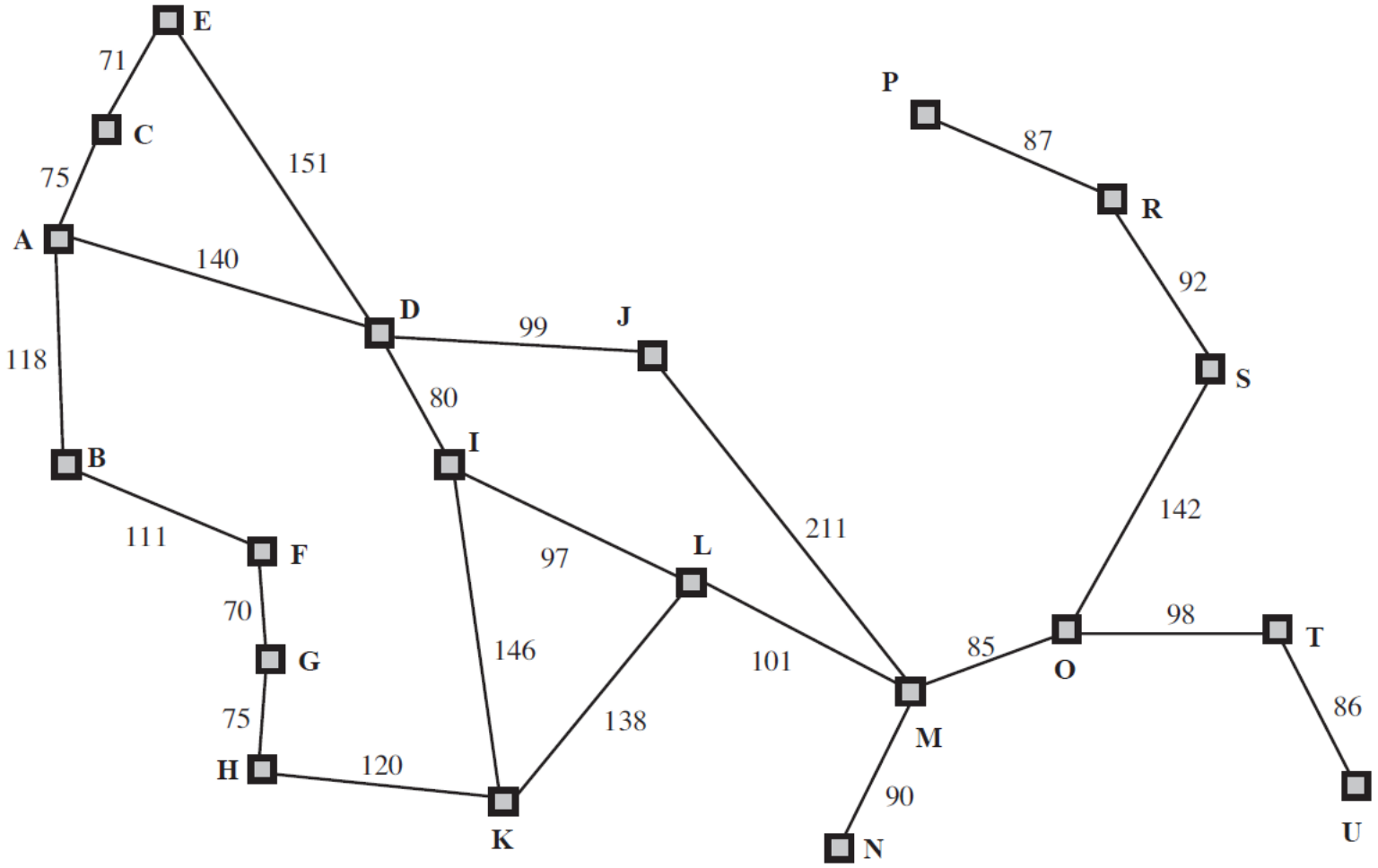
Iskanje v širino

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



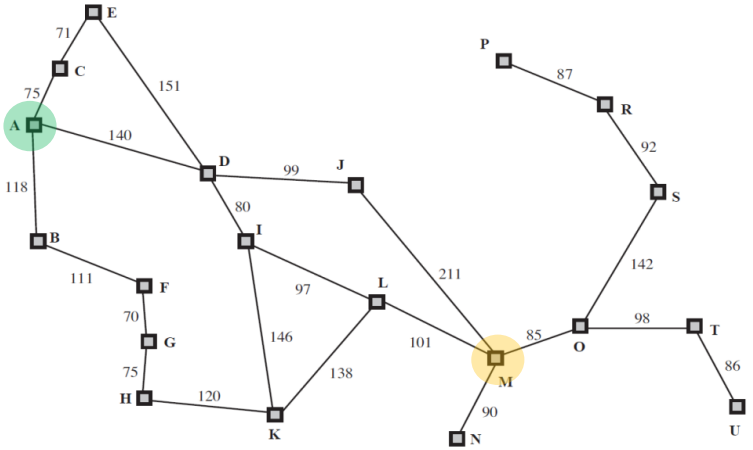
Iskanje v širino

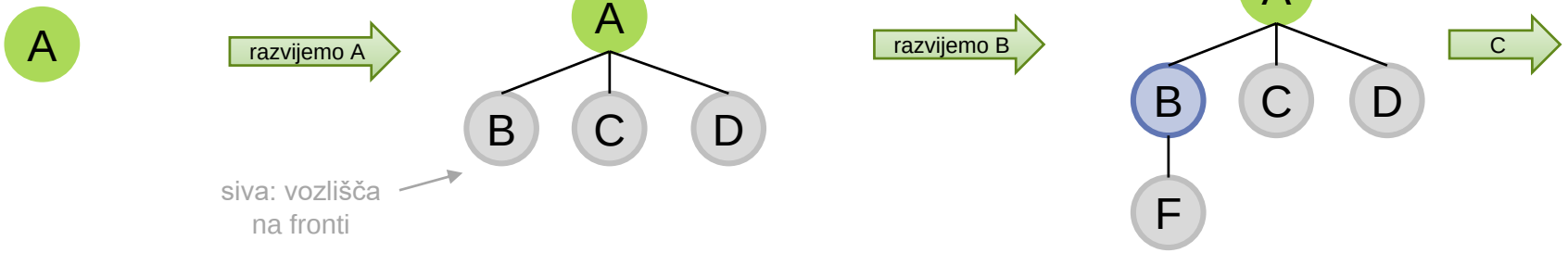
primer iz: Russell, Norvig, Artificial Intelligence: A Modern Approach, 3. izd., Pearson, 2010.

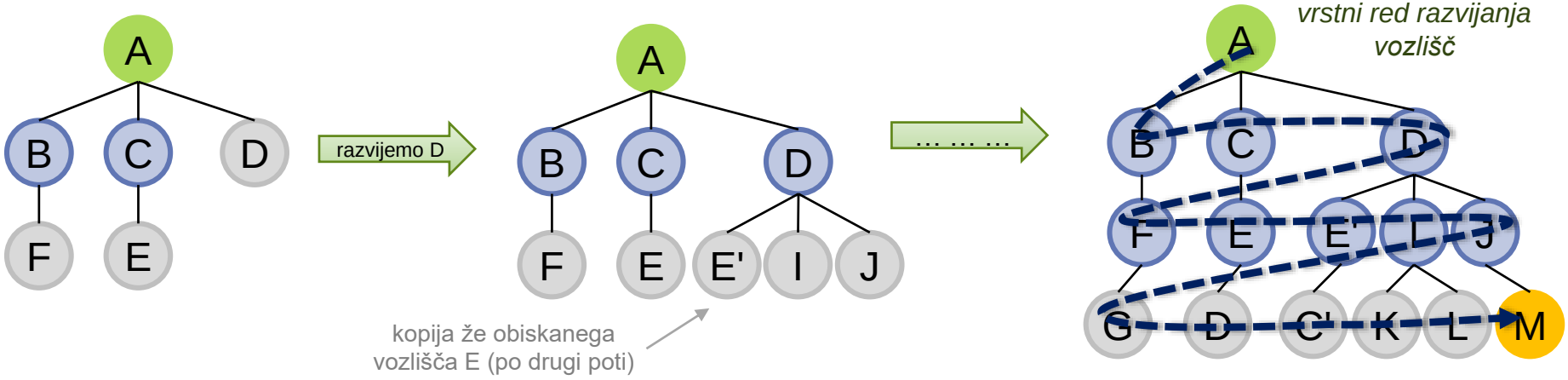


Iskanje v širino

- strategija: vedno razvij najbolj plitvo še nerazvito vozlišče
- odločimo se hraniti kopije že obiskanih vozlišč

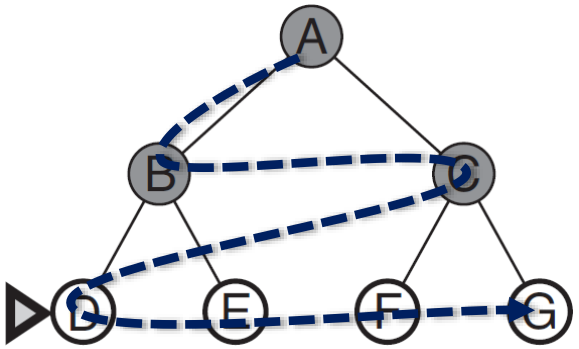






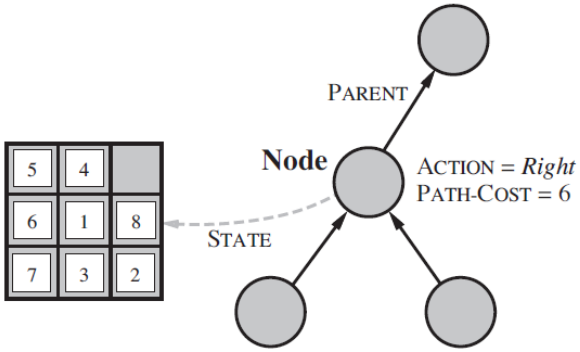
Iskanje v širino

- 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)
- možni implementaciji:
  - ciljno vozlišče zaznamo šele, ko ga želimo razviti
  - ciljno vozlišče zaznamo, že ko ga generiramo (bolj optimalno)



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



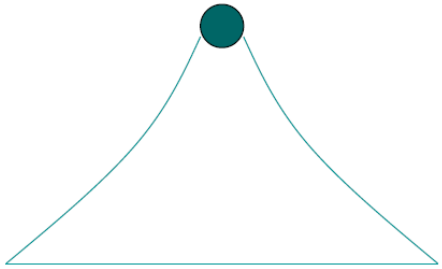

```
function iskanje(problem) {
    vozlišče <- problem.začetno_stanje, cena=0
    if (cilj(vozlišče.stanje)) return rešitev=vozlišče
    fronta <- vozlišče
    obiskana <- ∅
    while(true) {
        if (fronta== ∅) return rešitev= ∅
        vozlišče <- izberi(fronta)
        fronta <- fronta - vozlišče
        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

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)?
  - V splošnem ne nujno (da, če je optimalna najkrajša pot – cena povezav je 1).
- **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 $O(b^d)$

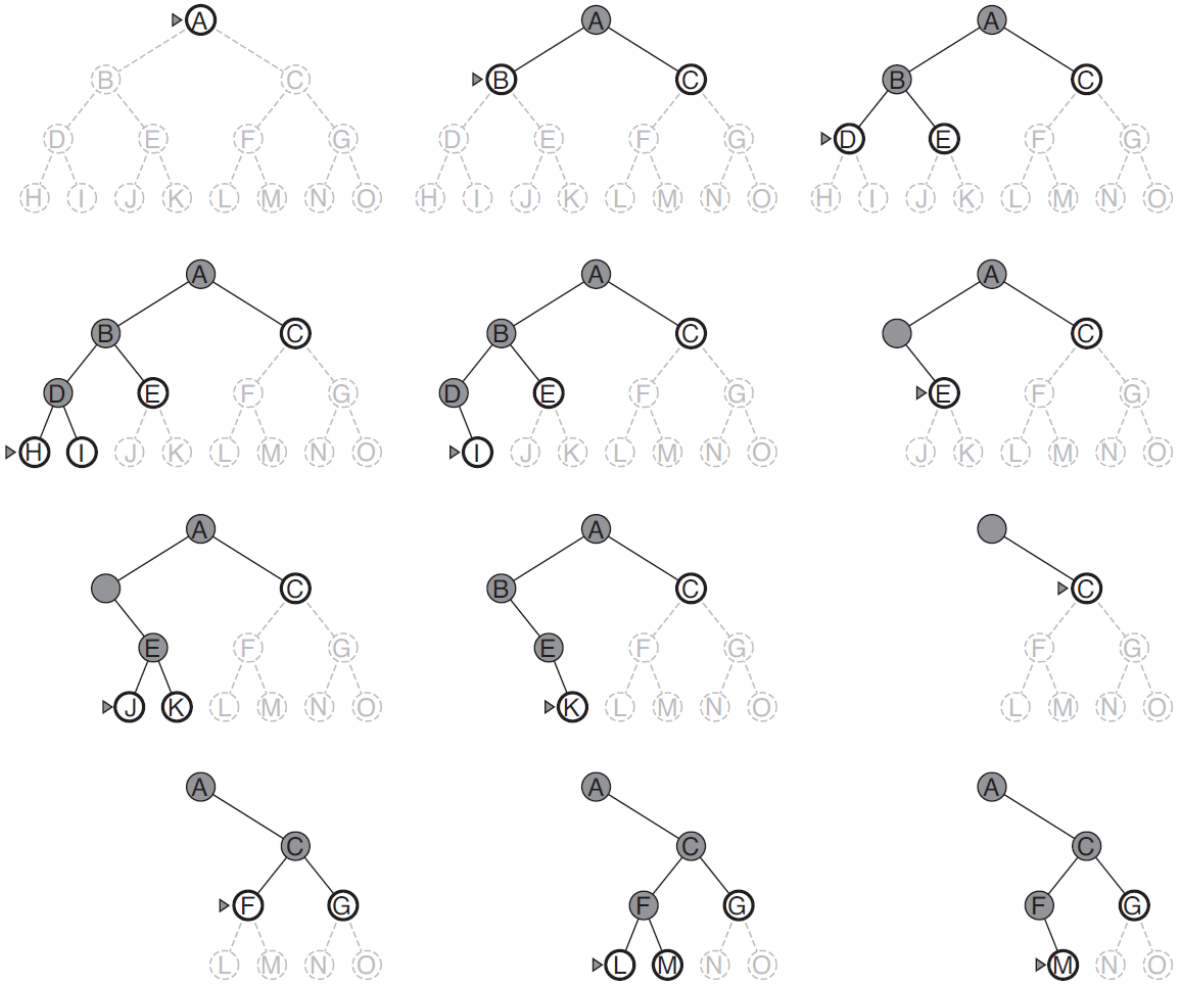
Zahtevnost iskanja v širino

- 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

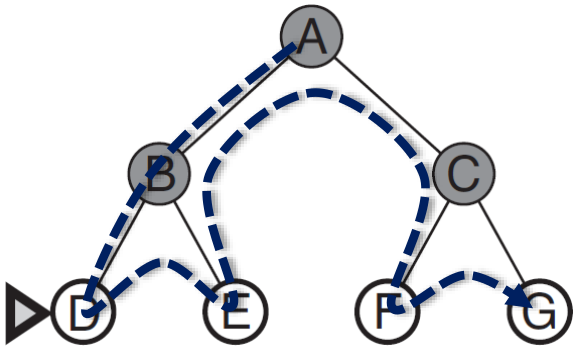
Iskanje v globino

- strategija: najprej razvij **najglobje še nerazvito** vozlišče
  - implementacija: naslednike dodaj v sklad (LIFO) za razvijanje



Iskanje v globino

- ne zagotavlja najkrajše rešitve
- lahko se zacikla (neskončna pot v drevesu)
- možna je preprosta implementacija z rekurzijo
- že raziskanih vej nam ni treba hraniti v spominu (hranimo le pot od začetnega do trenutnega vozlišča)



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*):
  - Ne. Neuspešen je v prostorih z neskončno globino (prostorih z zankami)
- **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

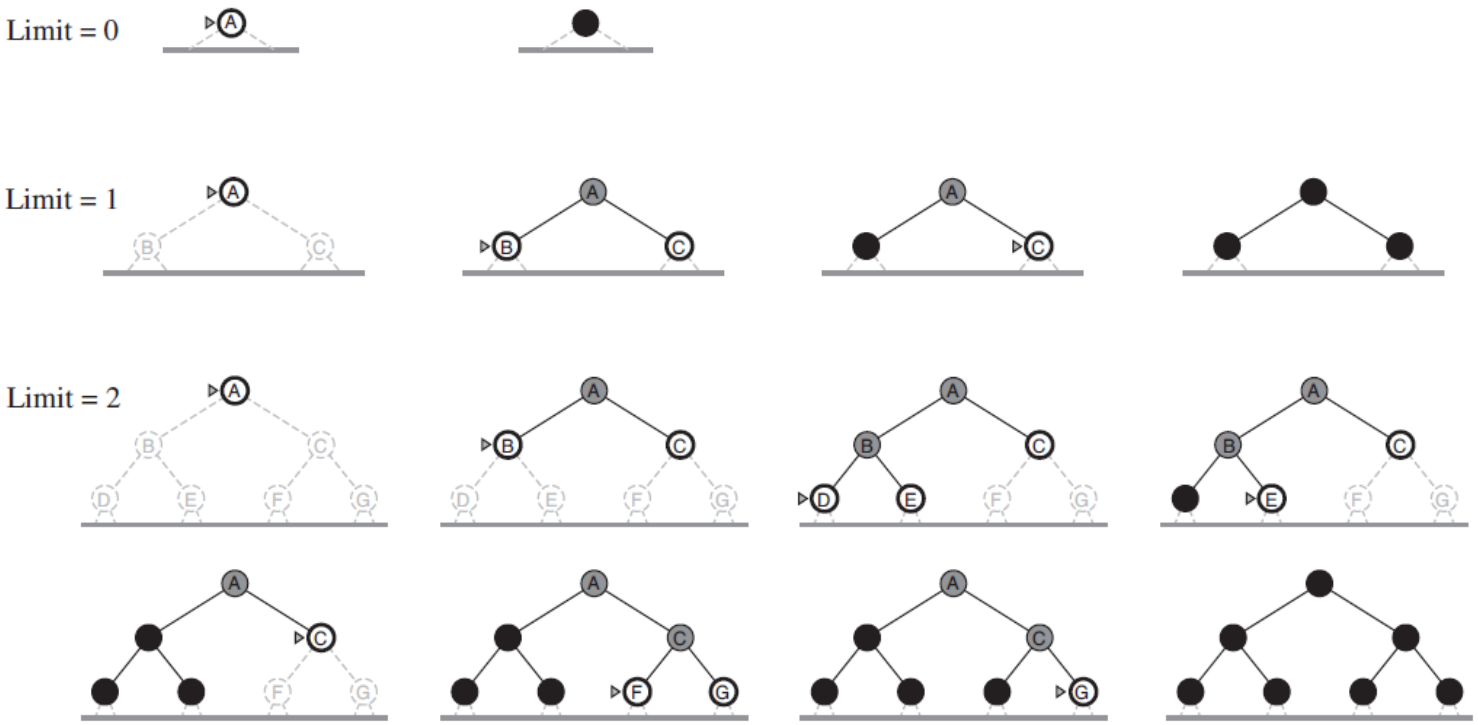
- iskanje s sestopanjem (*backtracking search*):
  - namesto vseh naslednikov generiramo samo enega po enega
  - ➔ prostorska zahtevnost $O(m)$

- 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 (*iterative deepening*)

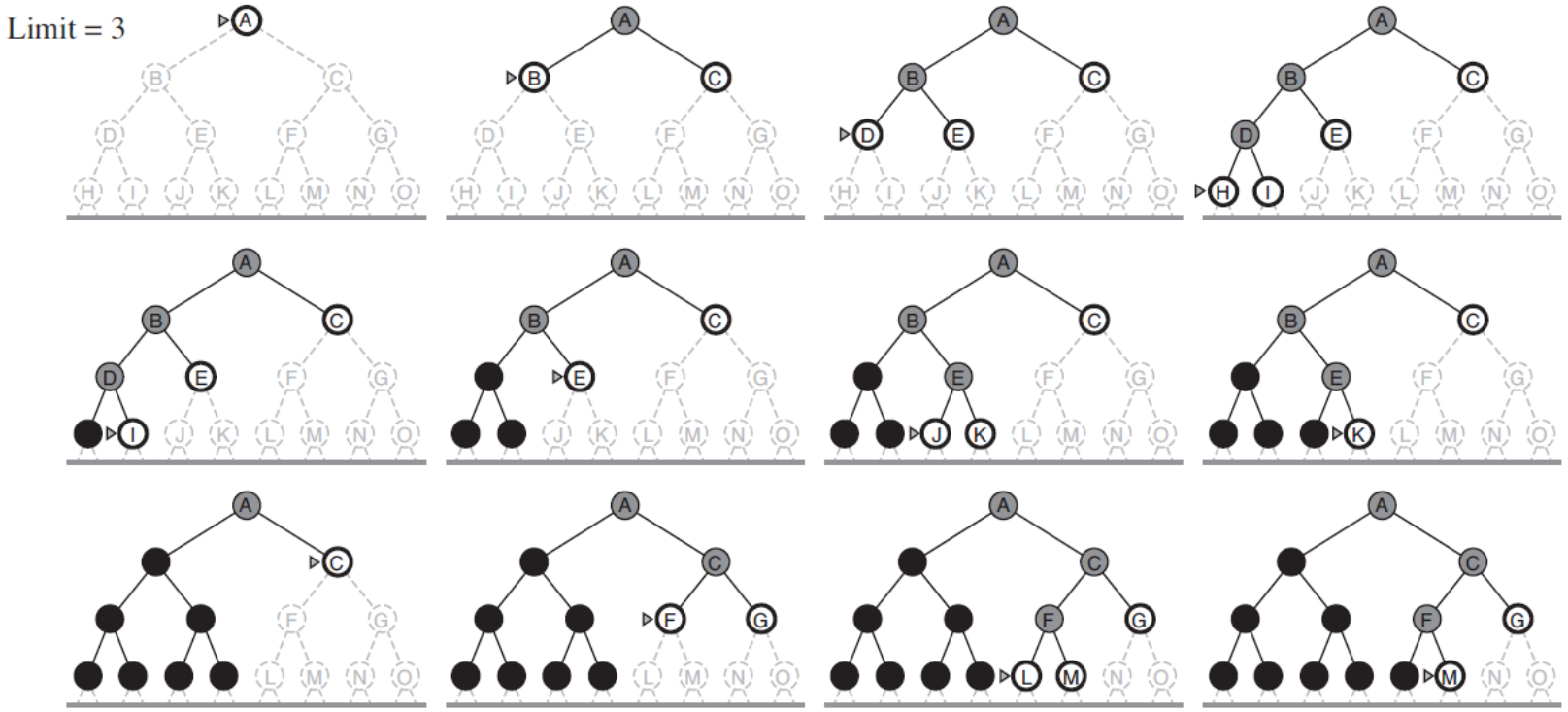
Iterativno poglabljanje

- problem globinsko omejenega iskanja v globino je nastavitev meje l
- rešitev: iterativno poglabljanje (*iterative deepening depth-first search*)
- strategija: začnimo z nizko mejo *limit* in jo povečujmo za 1, dokler ne najdemo rešitve. Na vsakem koraku poženimo iskanje v globino.



Iterativno poglabljanje

- problem globinsko omejenega iskanja v globino je nastavitev meje l
- rešitev: iterativno poglabljanje (*iterative deepening depth-first search*)
- strategija: začnimo z nizko mejo *limit* in jo povečujmo za 1, dokler ne najdemo rešitve. Na vsakem koraku poženimo iskanje v globino.



Učinkovitost iterativnega poglabljanja

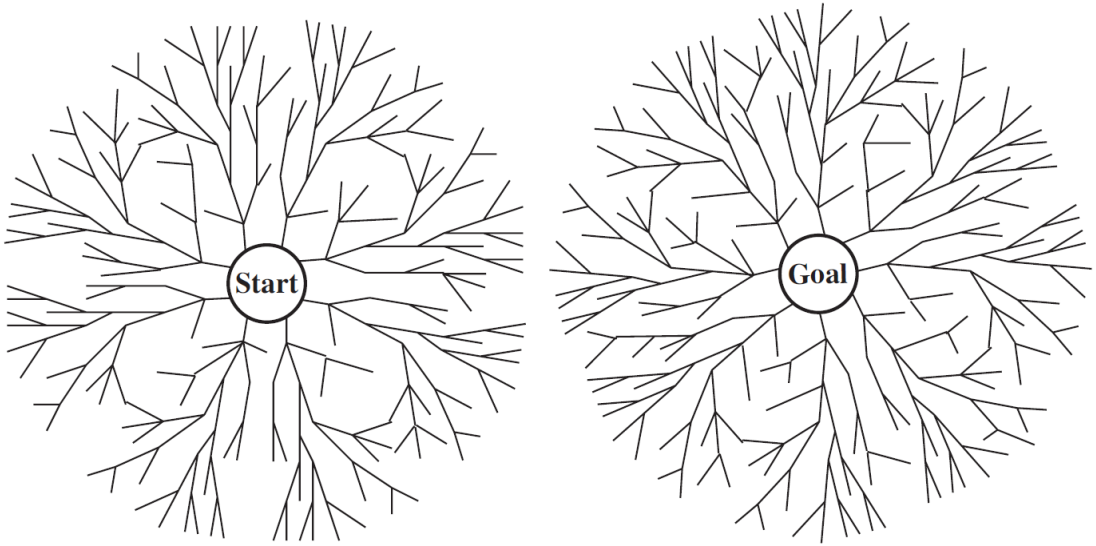
- **popolnost** (angl. *completeness*):
  - Da.
- **optimalnost** (angl. *optimality*):
  - Da (v kolikor iščemo najkrajšo rešitev).
- **časovna zahtevnost**:
  - v iteracijah se ponavlja generiranje istih vozlišč znova
  - generirano število vozlišč je asimptotično enako kot pri iskanju v širino:

$$[\mathrm{b}] + [b + b^2] + \cdots [b + b^2 + \cdots + b^d]$$
$$= db + (d-1)b^2 + \cdots + 2b^{d-1} + b^d = O(b^d)$$

  - kljub temu pa je cena še vedno sprejemljivo višja, ker se največ vozlišč generira na zadnjem nivoju (npr. za $b = 10$, $d = 5$, velja: $N(IDS) = 123.450$, $N(BFS) = 111.110$)
- **prostorska zahtevnost**:
  - hraniti mora samo $O(bd)$ razvitih vozlišč (linearna prostorska zahtevnost!)

- metoda torej kombinira prednosti iskanja v širino (popolnost, optimalnost) in iskanja v globino (linearna prostorska zahtevnost)

Dvosmerno iskanje



- ideja: pognati vzporedni iskanji od začetnega vozlišča proti cilju in vzvratno od cilja proti začetnemu vozlišču z upanjem, da se iskanji "srečata" na polovici poti
- motivacija: zaradi znižanja globine iskanja želimo doseči časovno zahtevnost $b^{d/2} + b^{d/2} = O(b^{d/2})$, kar je manj kot $O(b^d)$

Implementacija dvosmernega iskanja

- za izvedbo vzvratnega iskanja morajo vozlišča imeti kazalec na predhodnika
- ciljno vozlišče mora biti znano
  - pri igri 8 ploščic je npr. znano, pri uganki Sudoku pa ne
- uporabimo lahko poljuben preiskovalni algoritem
  - če uporabimo iskanje v širino, najde algoritem optimalno rešitev
- cilj iskanja preverja, ali obstaja med frontama obeh iskanj presečišče

- problemski prostor lahko redefiniramo tako, da en korak iskanja v "dvosmernem" prostoru predstavlja dva koraka (od začetka proti cilju in od cilja proti začetku) v originalnem prostoru
  - če v originalnem prostoru velja

potem definiramo v novem prostoru novi vozlišči (S,E) in (S1, E1)

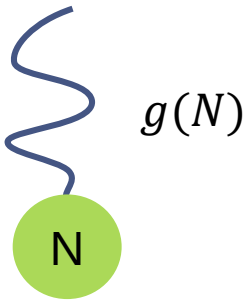
  - v "dvosmernem" prostoru velja (S,E) → (S1, E1), če obstaja v "enosmernem" prostoru povezava med S → S1 in med E1 → E
  - vozlišče (S,E) je v "dvosmernem" prostoru ciljno vozlišče, če velja E=S ali S → E

Primerjava časovnih zahtevnosti

Kriterij	Iskanje v širino	Iskanje v globino	Iskanje z omejitvijo globine	Iterativno poglabljanje	Dvosmerno iskanje
Popolnost	Da (če je b končen)	Ne	Ne	Da (če je b končen)	Da
Optimalnost	Da (če so cene enake)	Ne	Ne	Da (če so cene enake)	Da
Čas. zahtevnost	$O(b^d)$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Prost. zahtevnost	$O(b^d)$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$

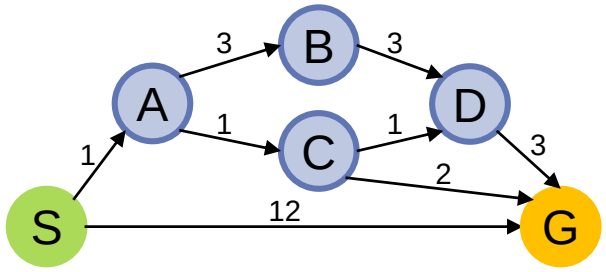
Cenovno-optimalno iskanje

- angl. *uniform-cost search, (best-first search with no heuristic)*
- posplošitev iskanja v širino
  - iskanje v širino je optimalno, če so cene vseh povezav enake 1
- če so cene povezav ≥ 1, je optimalno **razviti vozlišče, ki ima najmanjšo skupno ceno dosedanje poti** – $g(n)$
- fronta je urejena kot prioritetna vrsta
- test, ali je vozlišče ciljno, opravimo šele, ko je vozlišče na vrsti za razvijanje in ne ob generiranju vozlišča
  - zakaj?
  - ciljno vozlišče morda ni optimalno (obstaja boljša rešitev)
  - morda do najdenega optimalnega cilja vodi krajša pot

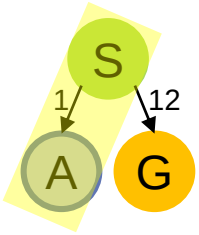


Cenovno-optimalno iskanje

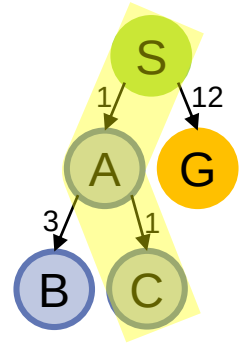
- primer



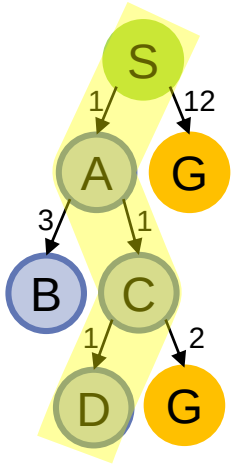
S/0



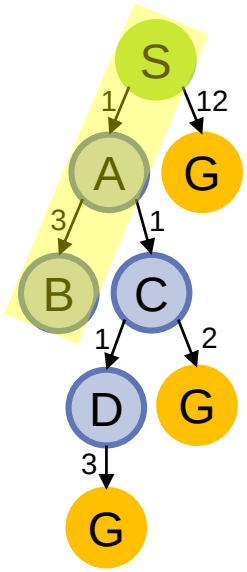
SA/1, SG/12



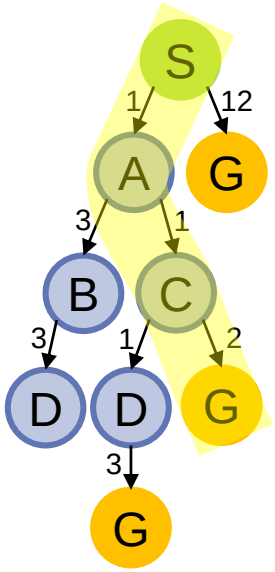
SAC/2, SAB/4, SG/12



SACD/3, SAB/4,
SACG/4, SG/12



SAB/4, SACG/4,
SACDG/6, SG/12



SACG/4, SACDG/6,
SABD/7, SG/12

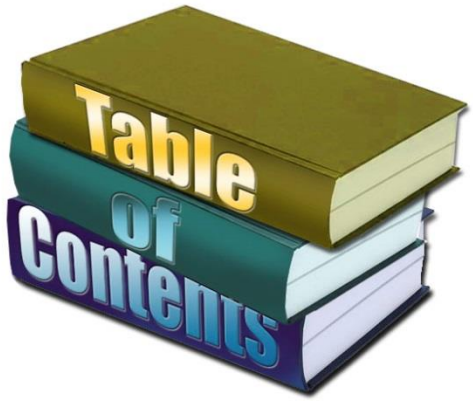
- rešitev: najboljša pot: SACG/4

Učinkovitost iskanja

- za potrebe analize predpostavimo, da je prostor stanj drevo:
  - globina (*depth*) optimalne rešitve naj bo $\boldsymbol{d}$
  - stopnja vejanja (*branching factor*) naj bo $\boldsymbol{b}$, na nivoju $\boldsymbol{d}$ imamo torej b^d vozlišč
  - največja globina drevesa naj bo $\boldsymbol{max}$
  - $\boldsymbol{C}^*$ naj bo cena optimalne rešitve
  - $\boldsymbol{\epsilon}$ naj bo najmanjša cena povezave

- **popolnost** (angl. *completeness*):
  - Da, za cene povezav > 0.
- **optimalnost** (angl. *optimality*):
  - Da.
- **časovna in prostorska zahtevnost**:
  - odvisni sta od cen poti in ne od globine d in vejanja b
  - zahtevnost $O\left(b^{1+\lfloor C^*/\epsilon \rfloor}\right)$, kar je lahko veliko več kot $O\left(b^d\right)$
  - če so vse cene poti enake, se zahtevnost poenostavi v $O\left(b^{1+d}\right)$
    - zakaj $O\left(b^{d+1}\right)$ in ne $O\left(b^d\right)$?

Pregled



- neinformirani preiskovalni algoritmi
  - iskanje v širino
  - iskanje v globino
  - iterativno poglabljanje
  - cenovno-optimalno iskanje

- **informirani preiskovalni algoritmi**
  - hevristično preiskovanje (primer)
  - požrešno preiskovanje
  - A*
  - IDA*
  - kakovost hevrističnih funkcij

Povzetek

- ciljno-usmerjen agent potrebuje identifikacijo (abstrakcijo) **problema** in **cilja**
- problem definiramo z: **začetnim stanjem**, **akcijami**, **prehodno funkcijo**, **ciljnim predikatom**, **ceno** poti
- iskanje rešitve izvajamo s preiskovanjem **prostora stanj**
- preiskovalne algoritme ocenjujemo glede na **popolnost, optimalnost, časovno** in **prostorsko zahtevnost**
- preiskovalne algoritme delimo na **neinformirane** in **informirane**
- implementacija preiskovalnih algoritmov in preprečevanje ciklanja


Informirano
preiskovanje