nedelja informatike 3

Napredne pretrage u grafovima

Nikola Nedeljković

Matematička gimnazija
Nedelja informatike 3

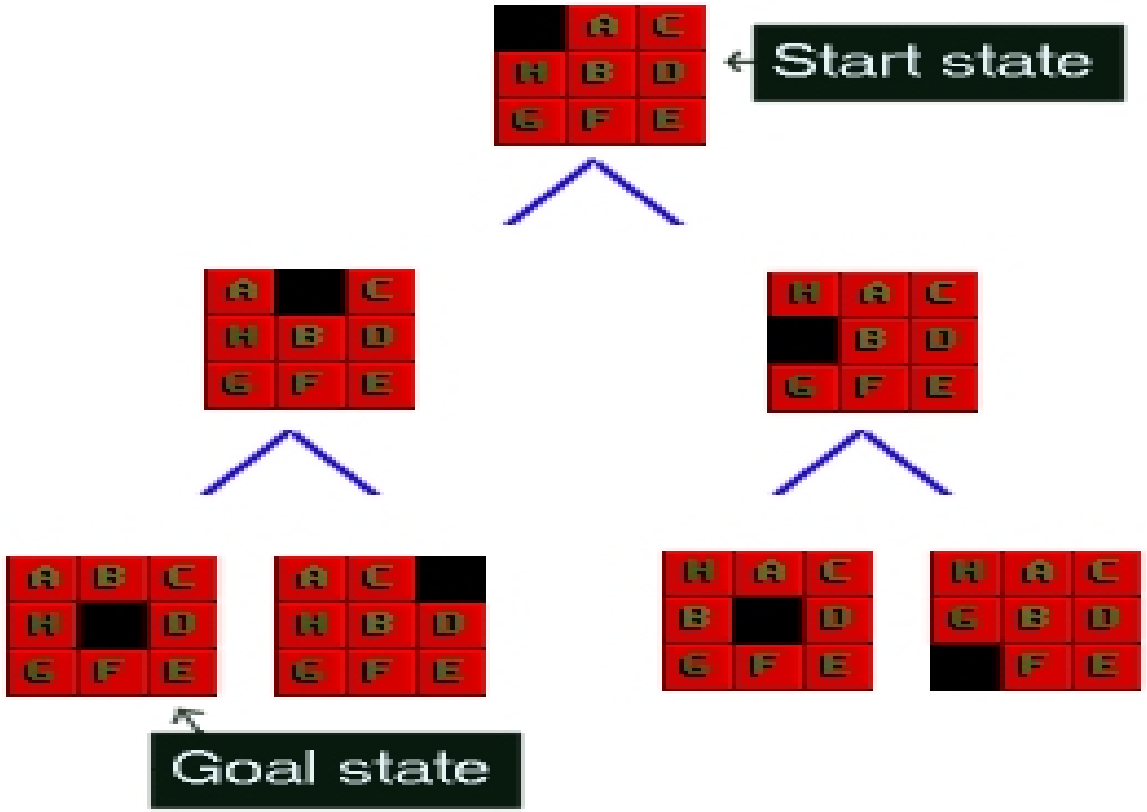
12. decembar 2016.

Prerequisites i cilj

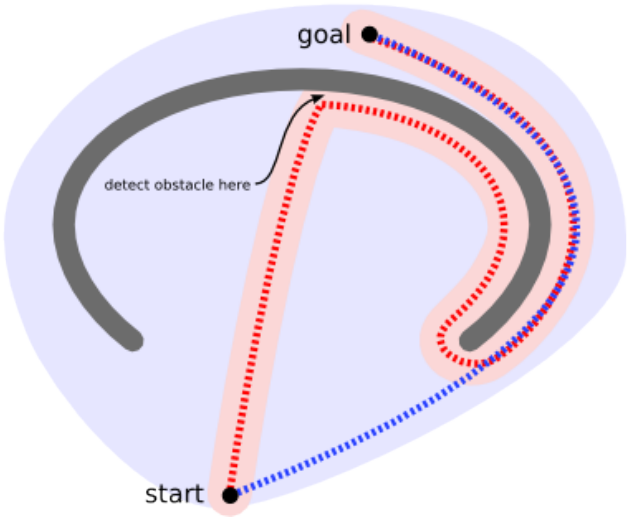
- DFS i BFS
- Koristićemo:
  - Dijkstrin algoritam sa modifikacijama (jedan od razloga zašto je dat u zadatku ACKO na kvalifikacionom testu)
  - C++ STL prioritetni red za implementaciju i poredjenje Best First Search-a, Dijkstre i A* algoritma.
  - AI heuristike - slično kao na Andrejevom predavanju pre dve godine.
- Hoćemo da dodjemo do traženog stanja ili čvora u grafu(goal node) pod određenim uslovima koje standardni algoritmi pretrage možda ne bi mogli da podrže.

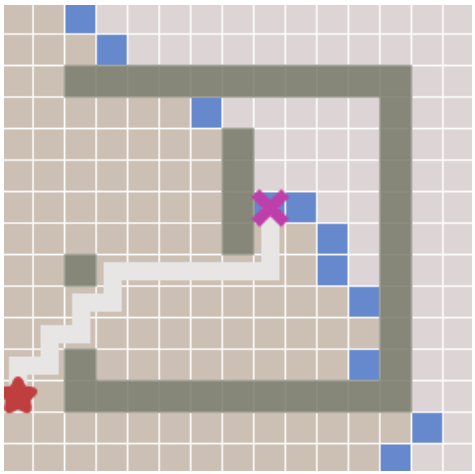
Primeri

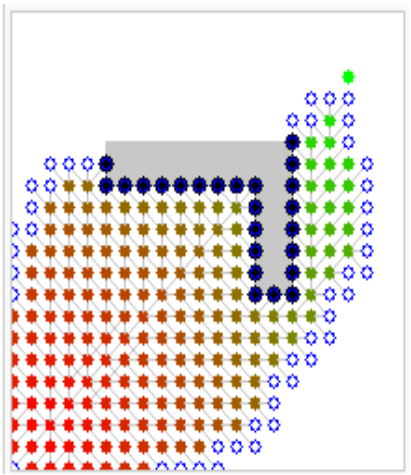
- 2d grid, stanje igre ili jednostavno put izmedju dva grada.



Primeri - cont.







Terminologija

- **b** - faktor grananja (branching factor) - prosecan broj suseda čvora u grafu
- **d** - dubina do traženja konačnog čvora (ako postoji)
- **heuristika(h)** - funkcija koja procenjuje koliko smo daleko od cilja (što je bolja heuristika to ce bolje raditi algoritam)
- Npr. za primer *8puzzle* postoji 362,880 različitih stanja. Ako čuvamo u memoriji već posećenja stanja (što možda nije poželjno) imamo u svakom koraku u proseku 2 mogućnosti (2 ivice u grafu) pa je **b**=2, a **d**=2 (za dati primer).

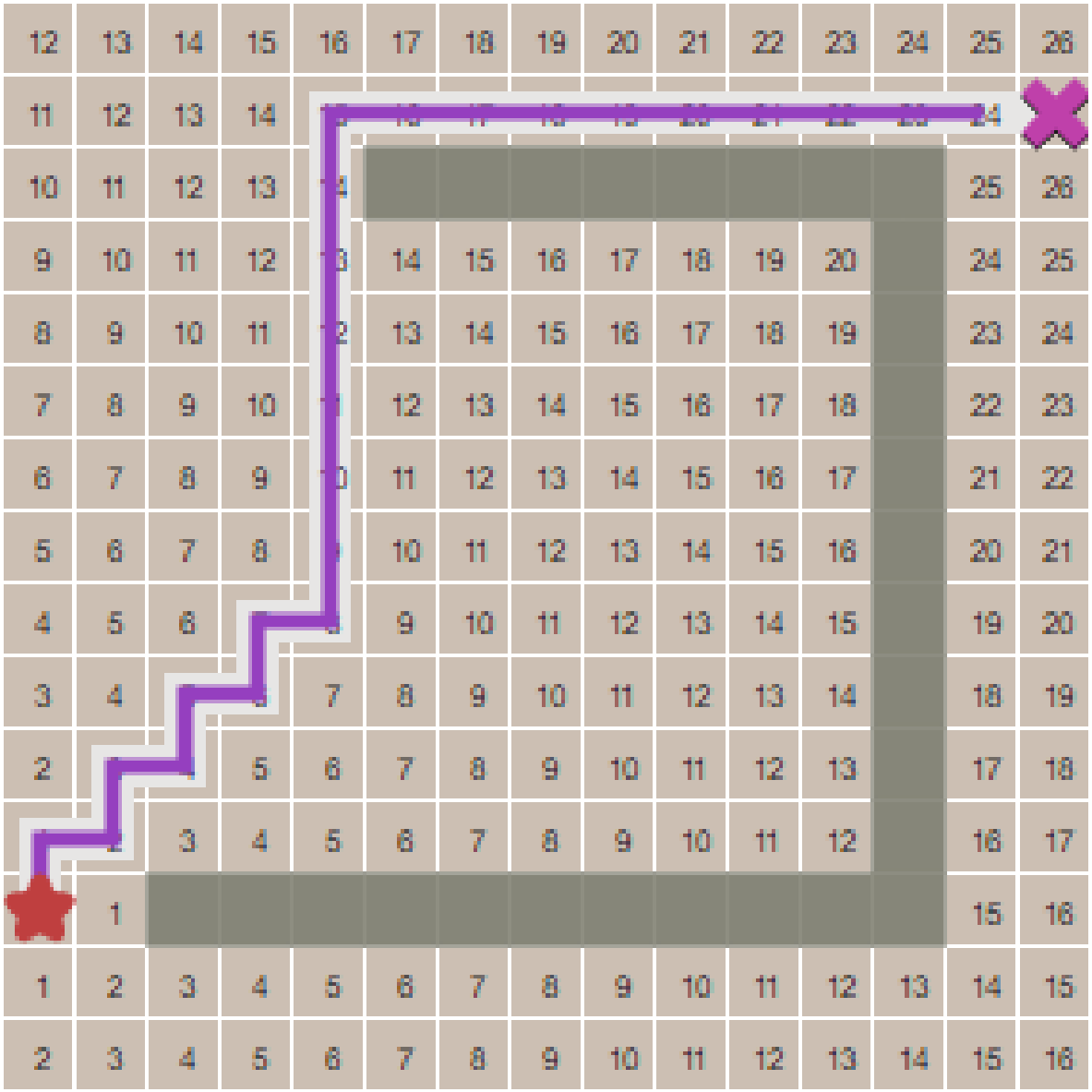
Šta ćemo raditi?

- Izložićemo C++ implementacije Dijkstre (sa prioritetnim redom), Best-first-search-a i A* algoritma, diskusovati i porediti ih, a zatim dati konkretne primere rada A* algoritma sa određenim heuristikama i odraditi razne varijacije A*:
  - Beam search
  - IDA*
  - Jump-Point-Search(JPS)

Dijkstra with Priority Queue

```
using pii = pair<int, int> ;
vector<pii> e[MAXN];
int dijkstraPQ(int start, int goal)
{
    priority_queue <pii, vector<pii>, greater<pii>> pq;
    unordered_map<int, int> d;
    ///vector <int> d(V+1, MAXDIST), closed_set(V+1, 0);
    pq.push(make_pair(0, start));
    while(!pq.empty())
    {
        pii p = pq.top();
        int pu = p.second, pw = p.first;///pair(tezina, cvor) kao d[cvor] = tezina
        pq.pop();
        if (d.count(pu))///vec se nalazi, nema potrebe za gledanje opet
            continue;

        d[pu] = pw;
        if (pu == goal) return pw;///nasli smo goal

        for (auto next : e[pu])
            if (!d.count(next.second) || pw+next.first < d[next.second])
                pq.push(make_pair(pw+next.first, next.second));
    }
    return -1;
}
```

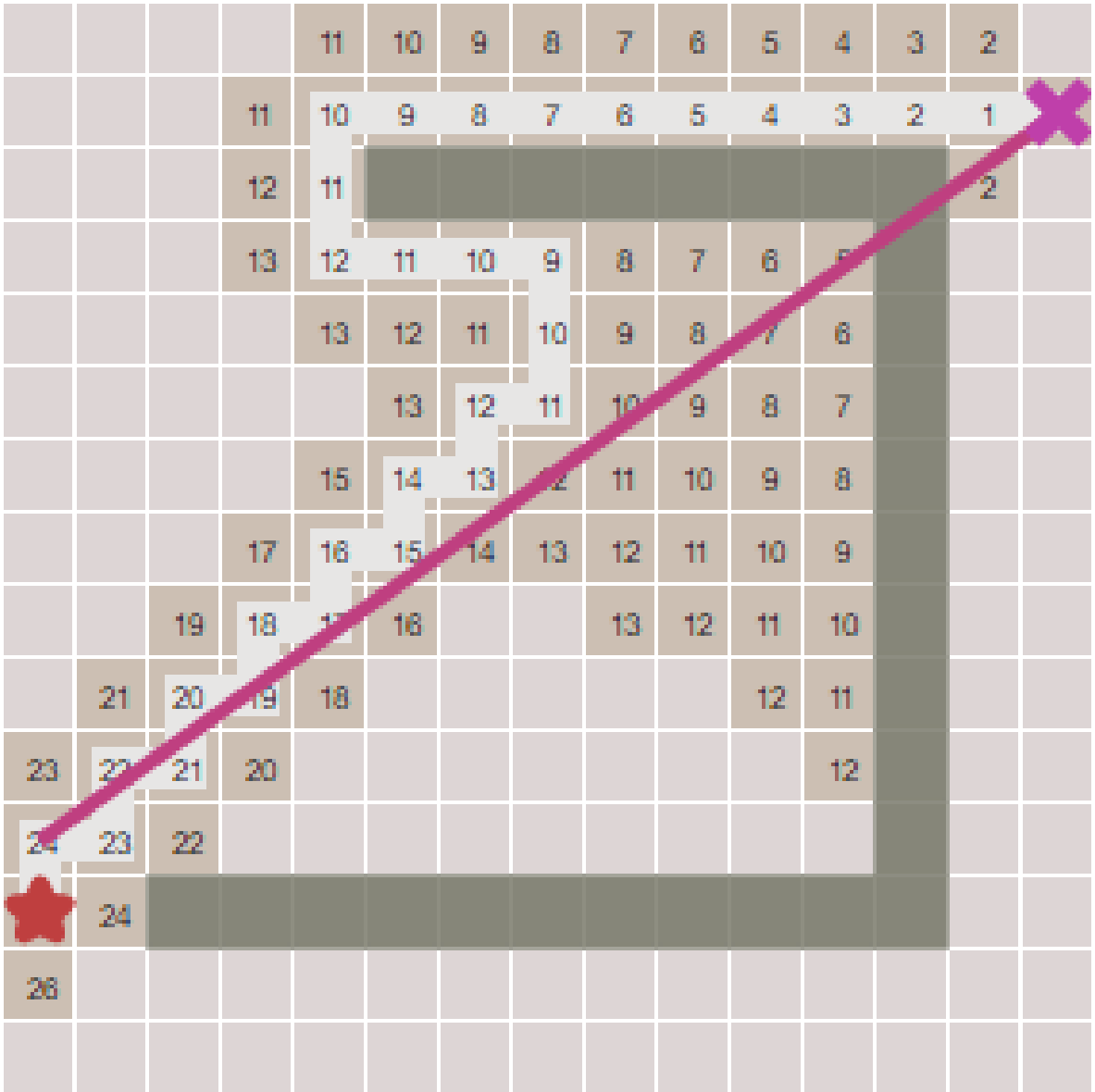
Dijkstra with Priority Queue



Greedy Best First Search

```
int heuristika(int node, int goal) {return (goal - node);}
void greedyBestFirstSearch(int start, int goal)
{
   priority_queue<pii, vector<pii>, greater<pii>> pq;
   unordered_map<int, bool> used; unordered_map<int, int> came_from;
   ///bitno nam je samo da nadjemo put do ciljnog cvora
   ///rekonstruisemo ga pomocu came_from klasicno
   came_from[start] = 0;
   pq.push(make_pair(0, start));
   while(!pq.empty())
   {
        pii p = pq.top();
        int pu = p.second;///pair(heuristika, cvor)
        pq.pop();
        if (used.count(pu))///vec se nalazi, nema potrebe za gledanje opet
            continue;
        used[pu] = 1;
        if (pu == goal) break;///nasli smo goal
        for (auto next : e[pu])
            if (!used.count(next.second))
            {
                int priority = heuristika(next.second, goal);
                pq.push(make_pair(priority, next.second));
                came_from[next.second] = pu;
            }
   }
}
```

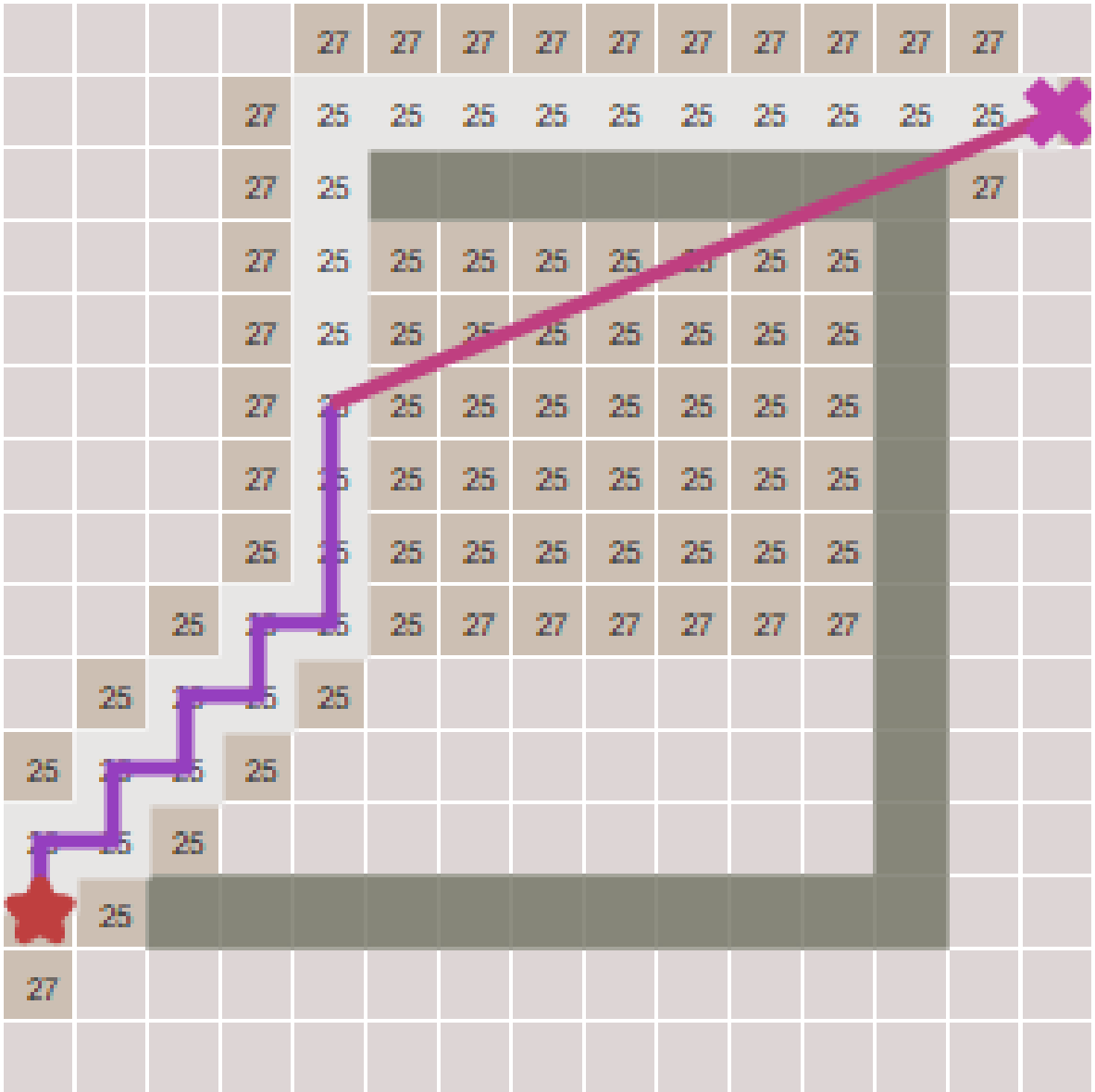
Greedy Best First Search



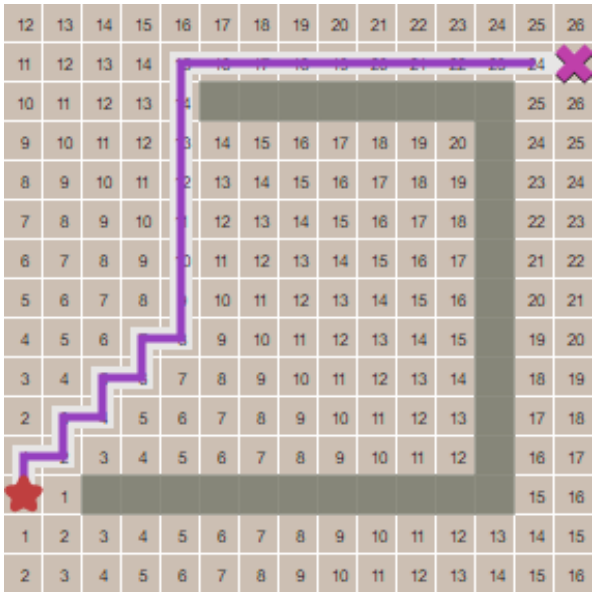
A* algoritam

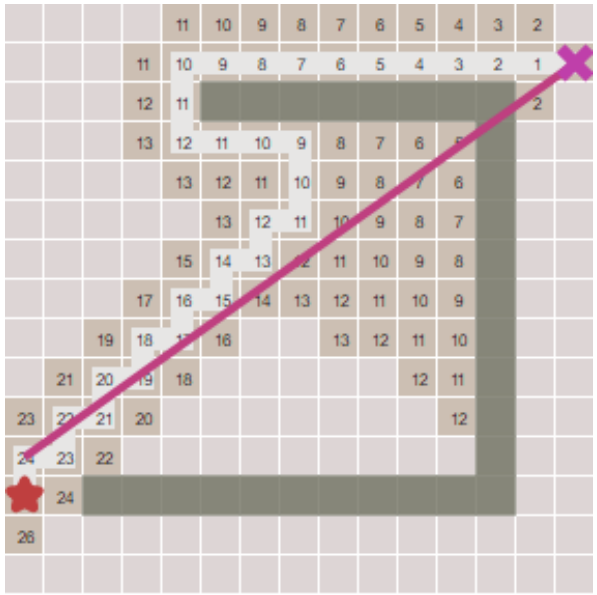
```
int heuristika(int node, int goal) {return (goal - node);}
void aStar(int start, int goal)
{
   priority_queue<pii, vector<pii>, greater<pii>> pq;
   unordered_map<int, int> came_from, d;

   came_from[start] = 0;
   d[start] = 0;
   pq.push(make_pair(0, start));

   while(!pq.empty())
   {
        pii p = pq.top();
        int pu = p.second; pq.pop();///(f(n), node)

        if (pu == goal) break;///nasli smo goal

        for (auto next : e[pu])
            {
                int newd = d[pu] + next.first;
                if (!d.count(next.second) || newd < d[next.second])
                {
                    d[next.second] = newd;
                    int priority = newd + heuristika(next.second, goal);///f(n) = g(n)+h(n)
                    pq.push(make_pair(priority, next.second));
                    came_from[next.second] = pu;
                }
            }
   }
}
```

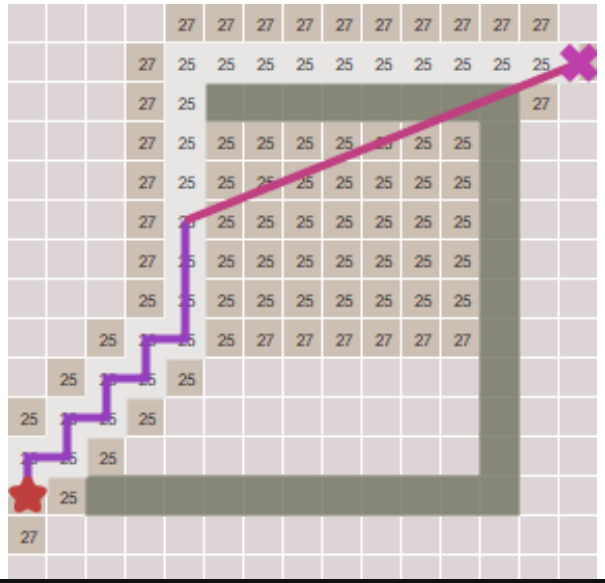
A* algoritam


27 27 27 27 27 27 27 27 27 27
27 25 25 25 25 25 25 25 25 25 25
27 25 27
27 25 25 25 25 25 25 25 25
27 25 25 25 25 25 25 25 25
27 25 25 25 25 25 25 25 25
27 25 25 25 25 25 25 25 25
25 25 25 25 25 25 25 25 25
25 25 25 27 27 27 27 27 27
25 25 25
25 25
25 25
27

Poredjenje





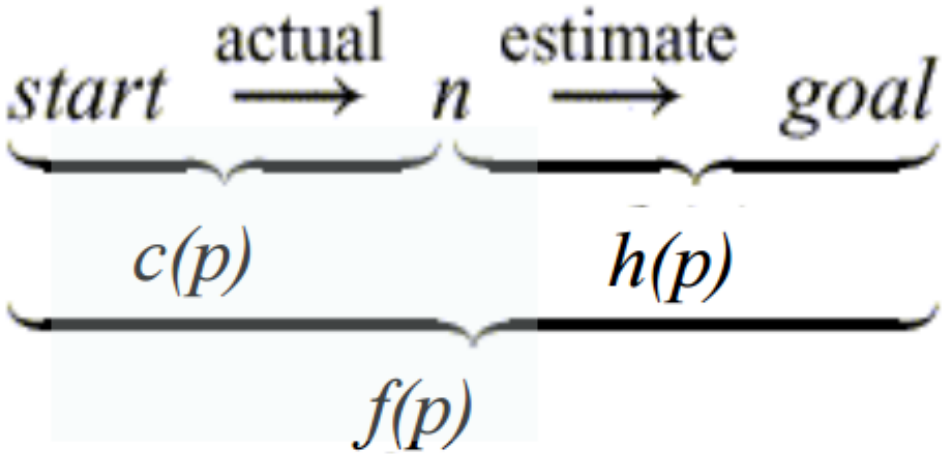


Poredjenje - pitanja

- Koji algoritmi uopšte nalaze bilo kakav put do čvora (ako on postoji)?
- Koji algoritmi nalaze najkraći (najbolji) put do konačnog čvora?
- Ocena vremenske/memorijske složenosti za sva 3 algoritma?

Osnove

- Za nalaženje traženog čvora koristimo i najkraće distance do medju-čvorova (Dijkstra) kao i procenu distance do krajnjeg čvora (BestFirstSearch).

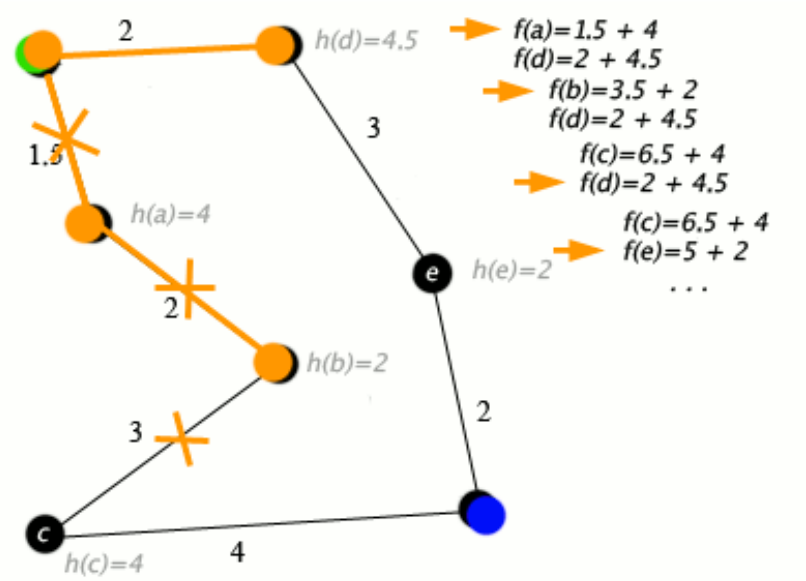


- Neka je $c(p)$ trenutno najmanje tačno rastojanje do čvora p a $h(p)$ procena rastojanja do krajnjeg čvora od čvora p. Tada se $f(p) = c(p) + h(p)$ baca na prioriteni red i prema tome nalazi put.

Osnove-cont.

- Neka je $h^*(p)$ tačno najkraće rastojanje od čvora p do krajnjeg čvora.
- Šta ako je $h(p) > h^*(p)$?
- A* je kompletan (nalazi rešenje ako ono postoji) i optimalan (nalazi optimalan put do cilja) ako:
  - Težine ivica su $> \epsilon > 0$
  - Heuristika $h(n)$ je *admissible* - tj. $(\forall p) h(p) \leq h^*(p)$
- U tom slučaju kažemo da je A* *admissible*.

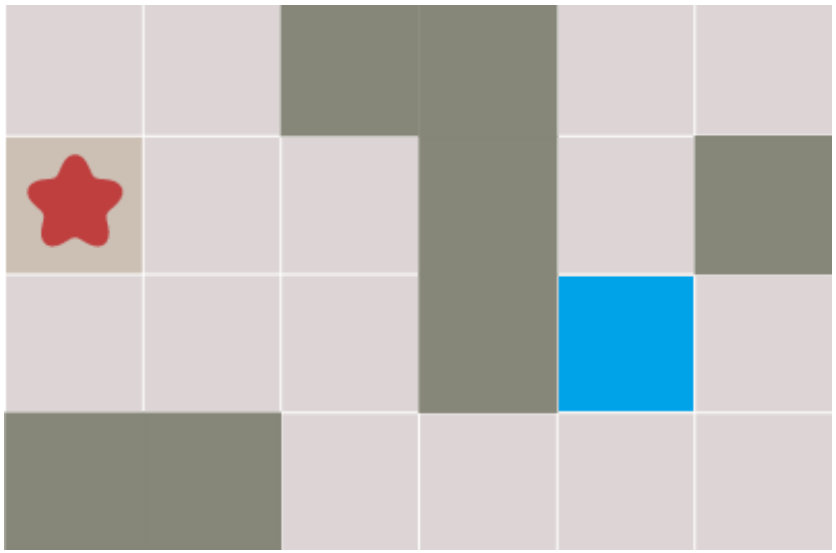
Heuristike - primeri - Vozač



- Pravolinijska distanca!

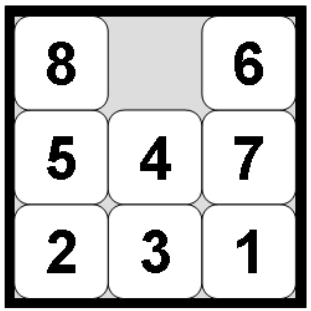
Heuristike - primeri - Robot

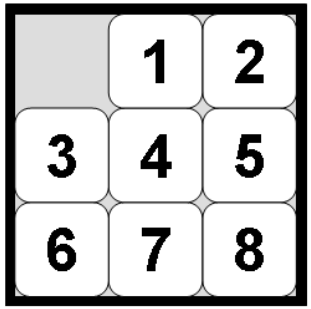
- 2d-grid sa preprekama
- Robot može da se kreće gore/dole/levo/desno sa cenom 1.



- Menhetn distanca - Admissible?
- Šta ako može da se kreće i dijagonalno sa cenom $\sqrt{2}$?

Heuristike - primeri - 8puzzle





- $h_1(n)$ = broj pozicija *tile*-ova koje maše krajnju.
- $h_2(n)$ = suma koraka izmedju trenutne pozicije svakog *tile*-a do njegove krajnje pozicije.
- Koja je bolja i da li su *admissible*?

Rezultati - 8puzzle

d - dubina nalaženja rešenja

- $d = 12$
  - BFS: 3,644,035 puteva
  - $A^*(h_1)$: 227 puteva
  - $A^*(h_2)$: 73 puteva

- $d = 24$
  - BFS: inf puteva
  - $A^*(h_1)$: 39135 puteva
  - $A^*(h_2)$: 1641 puteva

A* admissibility

Zašto je A* kompletan (Završava se, ne ulazi u ciklove)?

- Neka je f_{min} težina optimalnog puta s (nepoznatog ali konačnog ako postoji rešenje)
- Za svaki pod-put p od s važi $f(p) \leq f_{min}$ (Zbog *admissibilnosti* heuristike h).
- Neka je $c_{min} = \epsilon > 0$ minimalna težina svih grana u grafu. Trivijalno svaki put dužine $> f_{min}/c_{min}$ ima težinu veću od f_{min}.
- $\Rightarrow$ Ne može postojati put beskonačne dužine jer se pre njega moraju otkriti svi pod-putevi p, a samim tim i s gde se program završava.

Konzistentna heuristika

- Heuristika h je konzistentna ako

$$\forall(N) \quad h(N) \leq c(N, P) + h(P)$$

, gde je:

  - N bilo koji čvor u grafu,
  - P sused čvora N,
  - $c(N, P)$ težina grane (N, P)
  - $h(G) = 0$, gde je G *goal node*

- Konzistentna heuristika je ujedno i *admissible*!

Konzistentna(monotona) heuristika

- A* algoritam koji koristi konzistentnu heuristiku za pretragu, kada *expand*-uje čvor prvi put, slično kao *Dijkstrin* algoritam zna da je našao najbolji put do tog čvora pa svaki čvor posećuje najviše jednom.
- U ovom slučaju A* algoritam je ekvivalentan *Dijkstrin*-om algoritmu sa izmenjenim težinama ivica:

$$c'(N, P) = c(N, P) + h(P) - h(N)$$

- Konzistentna heuristika se naziva i monotonom.
- Za svaki parcijalan put, $f(N_j) = g(N_j) + h(N_j)$ je ne-opadajuća funckija u smeru ka ciljnom čvoru.

Beam Search

- Šta ako nemamo dovoljno memorije?
- Beam Search ograničava veličinu prioritetnog reda(ili bilo koje strukture koja održava sortirani poredak)
- β - *beam width*
- U pitanju je *trade-off* pa gubimo kompletnost i optimalnost!

IDS

- IDS - Iterative Deepening DFS
- Hoćemo malo memorije, ali očuvanje komplenosti i optimalnosti!
- Slično kao DFS, ali idemo do ograničene dubine, i tu dubinu povećavamo iterativno.
- Memorijska složenost $O(bd)$ kao i kod DFS-a, a vremenska $O(b^d)$.

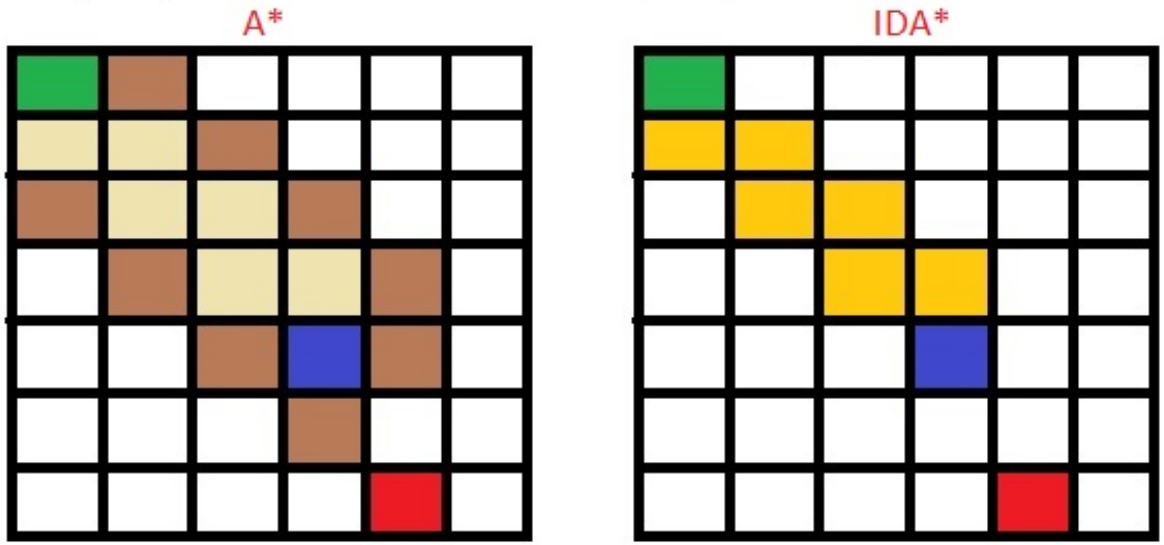
IDA*

- Sličan kao IDS, stim što dubinu posmatra po vrednosti funkcije $f(p) = c(p) + h(p)$
- Pored sličnog imena, i vrednosti funkcije f IDA* i A* nemaju ništa zajedničko, IDA* ne traži put po najmanjoj vrednosti f preko prioritetnog reda već radi DFS.
- Neka je početni *bound* (dubina)=f(s), za *start node* s.
- Ako ne nađemo rešenje pod okvirima trenutnog *bound*-a, povećavamo *bound* kao minimalnu f vrednost koja ga prekoračuje.
- Primer na tabli!

IDA* - pseudokod

```
procedure ida_star(root)
  bound := h(root)
  loop
    t := search(root, 0, bound)
    if t = FOUND then return bound
    if t = ∞ then return NOT_FOUND
    bound := t
  end loop
end procedure

function search(node, g, bound)
  f := g + h(node)
  if f > bound then return f
  if is_goal(node) then return FOUND
  min := ∞
  for succ in successors(node) do
    t := search(succ, g + cost(node, succ), bound)
    if t = FOUND then return FOUND
    if t < min then min := t
  end for
  return min
end function
```

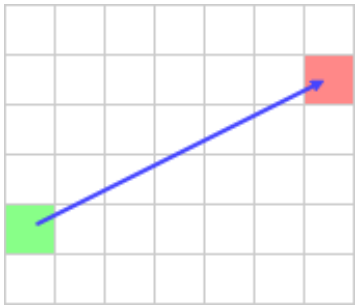
IDA* - memorija

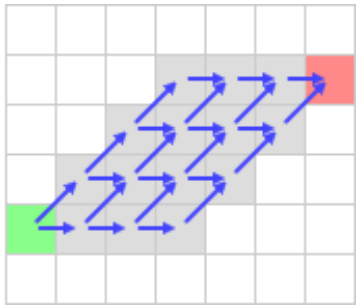


- zeleno - *start node*
- crveno - *goal node*
- plavo - trenutni čvor
- žuto - čvorovi na steku (kao kod DFS-a)
- svetlo sivo - posećeni čvorovi
- braon - čvorovi koji tek treba da se posete

JPS

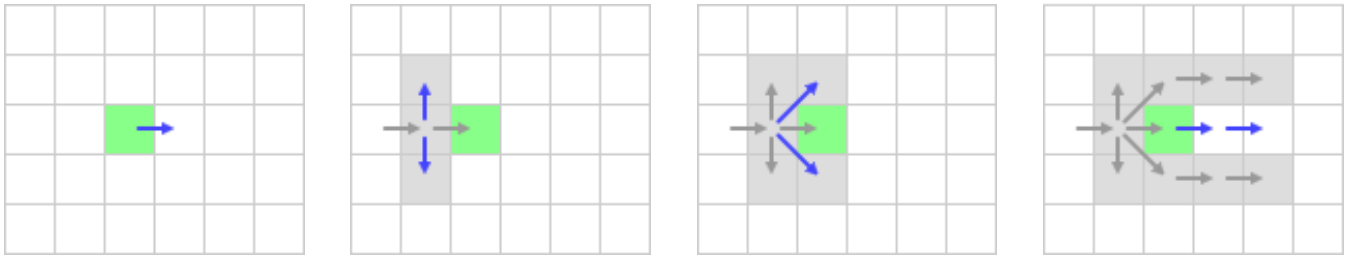
- JPS - Jump Point Search
- 2D grid, cene horizontalno i vertikalno 1, dijagonalno $\sqrt{2}$.
- Koristimo simetriju puteva kako bismo "preskočili"neke čvorove za koje smatramo da ih trenitno nije korisno gledati.



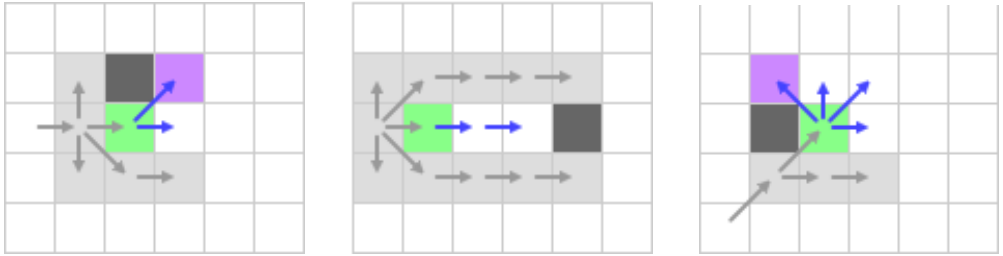


JPS-cont.

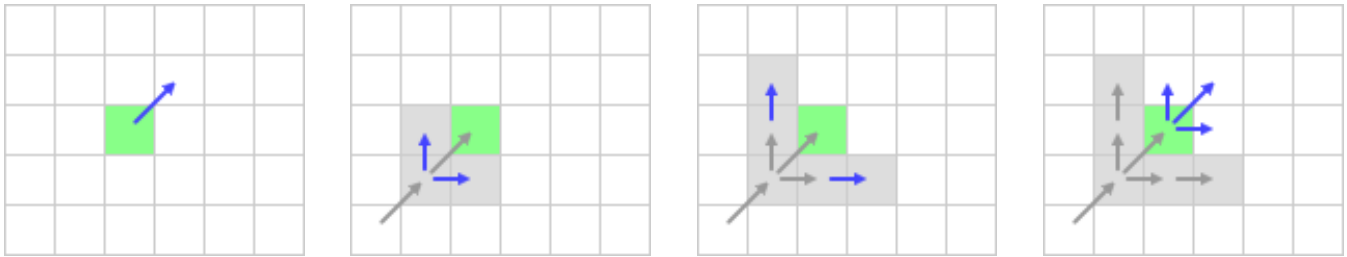
- Vertikalno kretanje - nastavljamo desno bez bacanja čvora na prioritetni red



- Ako imamo prepreke?

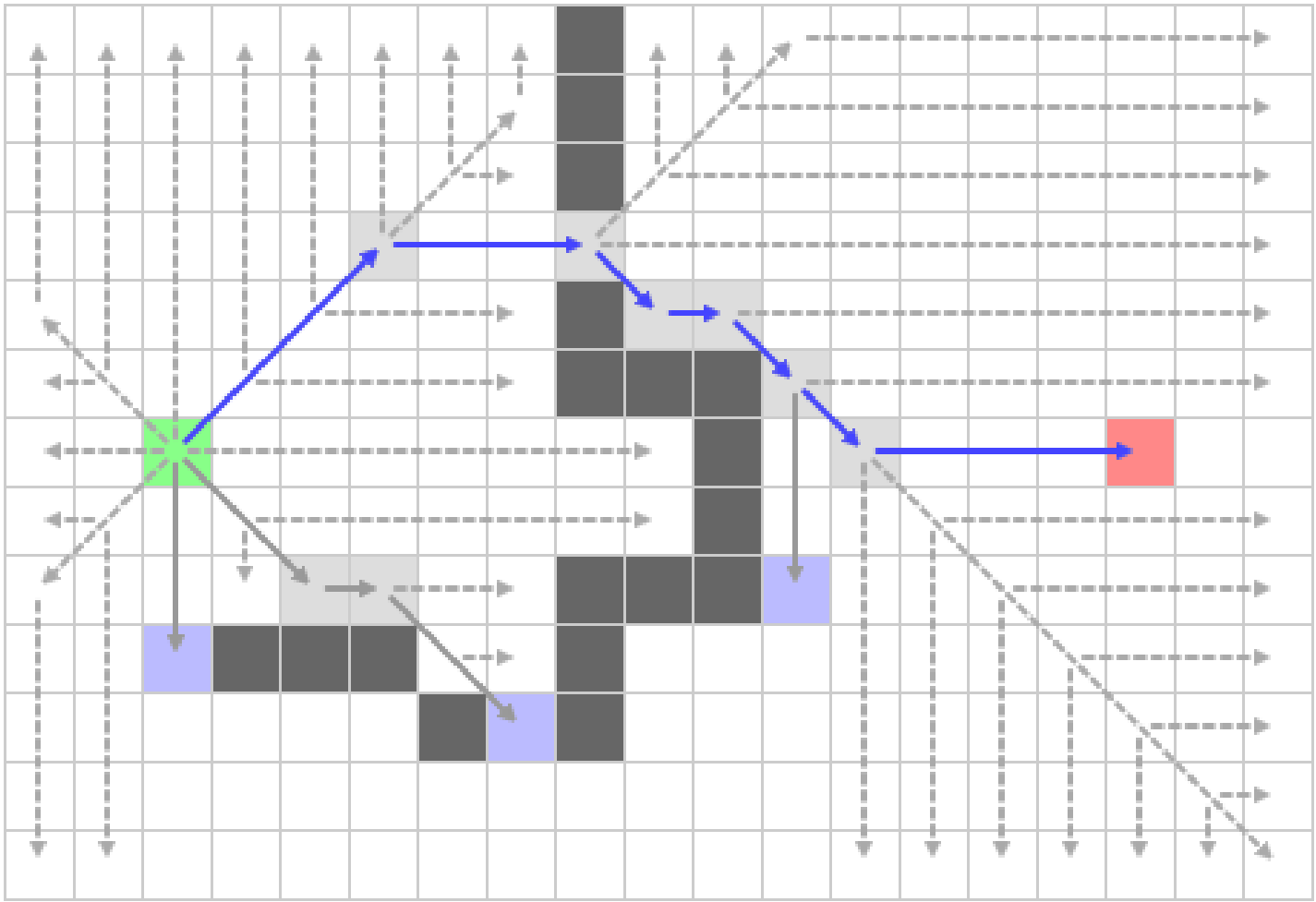


- Dijagonalno kretanje?



JPS-primer

- Memorijski skoro $O(1)$



Reference i dodatno čitanje

- Pathfinding i A*, IDA* -
  - `theory.stanford.edu/~amitp/GameProgramming/`
  - `www.cs.ubc.ca/~mack/CS322/lectures/2-SearchX.pdf` (staviti umesto X brojeve od 5 do 8)
  - `heyes-jones.com/astar.php`
- JPS -
  - `zerowidth.com/2013/05/05/jump-point-search-explained.html`