



Katedra za računarstvo
Elektronski fakultet, Univerzitet u Nišu

Veštačka inteligencija

Algoritmi traženja u Python-u (II deo)

Optimalni algoritmi traženja

- ▶ Algoritam grananja i ograničavanja (Branch and Bound)
- ▶ A* algoritam



Algoritam grananja i ograničavanja

- ▶ Kreiraj listu puteva (P)
- ▶ Dodaj polazni čvor (S) u listu puteva (P)
- ▶ Sve dok se prvi put iz P ne završava ciljnim čvorom (G), ili je lista puteva (P) prazna:
 - ▶ Uzmi prvi element (put) iz liste puteva (P).
 - ▶ Proširi prvi put za jedan korak prema svim susedima kreiranjem X novih puteva.
 - ▶ Odbaci sve puteve koji formiraju petlje.
 - ▶ Dodaj sve preostale nove puteve u listu puteva (P).
 - ▶ Sortiraj sve puteve u listi puteva (P) po tekućoj vrednosti heuristike u rastućem redosledu.
- ▶ Ako je pronađen ciljni čvor (G) traženje je uspešno, u suprotnom ne.

A* algoritam traženja

- ▶ Jedan od najefikasnijih algoritama pretrage.
- ▶ A* izbegava puteve koji su skupi.
- ▶ Kombinuje stvarnu cenu putanje $g(n)$ sa predviđenom cenom $h(n)$.
- ▶ Ukupna cena se računa kao: $f(n) = g(n) + h(n)$.
- ▶ Svi čvorovi u redu se sortiraju po vrednosti $f(n)$.

A* – Algoritam

- ▶ Kreirati listu čvorova za obradu (OS) i listu obrađenih čvorova (CS).
- ▶ Dodati polazni čvor (S) u listu čvorova za obradu (OS).
- ▶ Dok se lista čvorova za obradu ne isprazni ili se ne dođe do ciljnog čvora:
 - ▶ Pronaći čvor S_i sa najmanjom ukupnom cenom puta (f) u listi čvorova za obradu.
 - ▶ Ako je S_i ciljni čvor (G) put je pronađen.
 - ▶ Ako nije, pronaći sve njegove sledbenike. Za svakog sledbenika (P_i):
 - ▶ Ukoliko se P_i ne nalazi u listi čvorova za obradu (OS) i ne nalazi se u listi obrađenih čvorova (CS), dodati sledbenika u listu za obradu, izračunati g i f za taj čvor i postaviti S_i kao njihovog prethodnika.
 - ▶ Ukoliko se P_i nalazi u listi za obradu, proveriti da li je cena puta do tog čvora sada manja od već upisane cene puta za taj čvor. Ukoliko jeste, ažurirati cenu puta do čvora i promeniti njegovog prethodnika na tekući čvor koji se obrađuje (S_i).
 - ▶ Izbaciti S_i iz liste za obradu i upisati u listu obrađenih čvorova.
- ▶ Ako je pronađen ciljni čvor (G) traženje je uspešno, u suprotnom nije.

A* – Funkcija

- ▶ Funkcija **a_star** izdvaja listu čvorova koji čine put od polaznog do ciljnog čvora.
 - ▶ `a_star(graf, start, end)`
- ▶ Ulazni parametri:
 - ▶ Graf
 - ▶ Polazni čvor
 - ▶ Ciljni čvor
- ▶ Format grafa:

```
graf = {  
    node1: (h1, [(dest11, g11), (dest12, g12)]),  
    node2: (h2, [(dest21, g21), (dest22, g22)]),  
    ...  
}
```

A* – Funkcija

- ▶ Pomoćne strukture i promenljive koje koristi funkcija `a_star`:
 - ▶ Set čvorova (oznaka) koje treba posetiti
 - ▶ `open_set`
 - ▶ Set posećenih čvorova (oznaka)
 - ▶ `closed_set`
 - ▶ Parovi čvorova (oznaka) oblika (čvor prethodnik)
 - ▶ `prev_nodes`
 - ▶ Dictionary koji pamti stvarnu cenu puta od polaznog čvora do nekog čvora (čvor vrednost)
 - ▶ `g`
 - ▶ Niz čvorova (oznaka) na putu od polaznog do ciljnog čvora
 - ▶ `path`
 - ▶ Logička promenljiva koja ukazuje da je pronađen ciljni čvor
 - ▶ `found_end`

A* – Funkcija

- ▶ Početak funkcije `a_star` priprema pomoćne parametre i strukture za pretragu grafa:

```
def a_star(graf, start, end):  
    found_end = False  
    open_set = set(start)  
    closed_set = set()  
    g = {}  
    prev_nodes = {}  
    g[start] = 0  
    prev_nodes[start] = None
```


A* – Funkcija

- ▶ Glavna petlja funkcije **a_star** koja obrađuje čvorove:

```
while len(open_set) > 0 and (not found_end):
    node = None
    for next_node in open_set:
        if node is None or g[next_node] + graf[next_node][0] < g[node] + graf[node][0]:
            node = next_node

    if node == end:
        found_end = True
        break
```

A* – Funkcija

- ▶ Nastavak glavne petlje, obrada potomaka tekućeg čvora:

```
for (m, cost) in graf[node][1]:
    if m not in open_set and m not in closed_set:
        open_set.add(m)
        prev_nodes[m] = node
        g[m] = g[node] + cost
    else:
        if g[m] > g[node] + cost:
            g[m] = g[node] + cost
            prev_nodes[m] = node
        if m in closed_set:
            closed_set.remove(m)
            open_set.add(m)
open_set.remove(node)
closed_set.add(node)
```

!!!

A* – Funkcija

- ▶ Kraj funkcije `a_star` formira put od početnog do ciljnog čvora:

```
path = []  
if found_end:  
    prev = end  
    while prev_nodes[prev] is not None:  
        path.append(prev)  
        prev = prev_nodes[prev]  
    path.append(start)  
    path.reverse()  
  
return path
```

A* – Primer grafa i poziva funkcije

▶ Primer grafa:

```
graf = {  
    "A": (9, [("B", 4), ("C", 6)]),  
    "B": (6, [("D", 4), ("A", 2)]),  
    "C": (2, [("D", 4), ("E", 1)]),  
    "D": (2, [("E", 2), ("F", 3)]),  
    "E": (3, [("F", 4)]),  
    "F": (0, [("A", 1)])  
}
```

▶ Primer poziva funkcije:

```
path = a_star(graf, "A", "F")  
-> ['A', 'B', 'D', 'F']
```

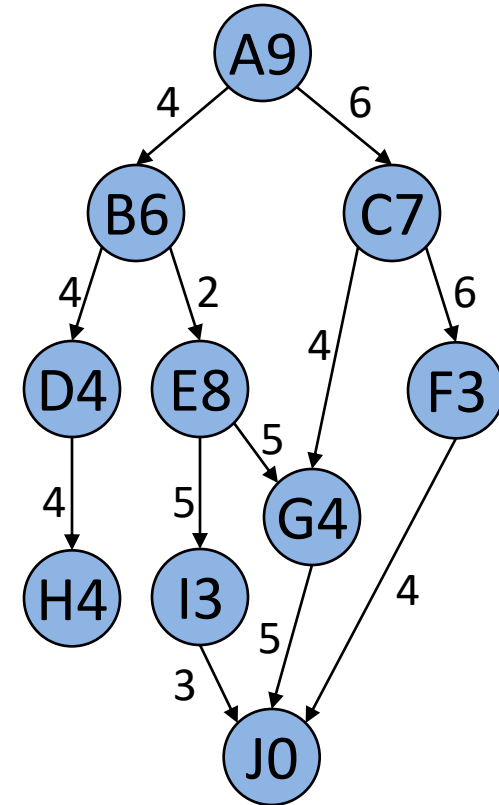
A* algoritam (ilustracija)

▶ Primer grafa:

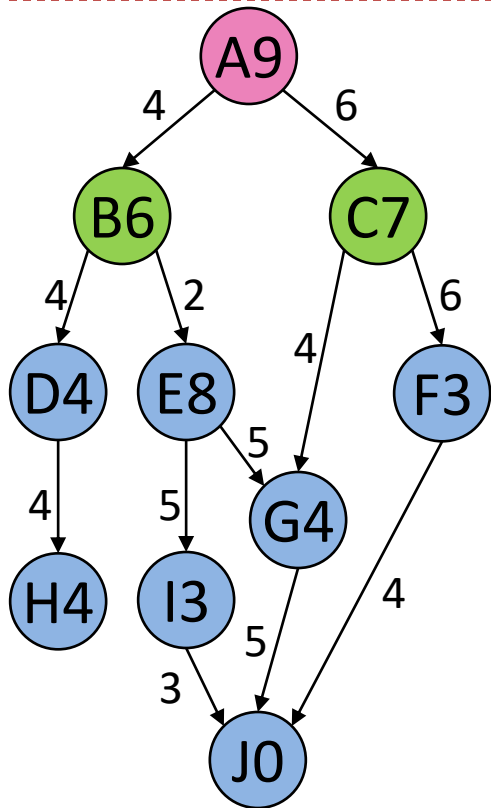
```
graf = {  
  "A": (9, [("B", 4), ("C", 6)]),  
  "B": (6, [("D", 4), ("E", 2)]),  
  "C": (7, [("G", 4), ("F", 6)]),  
  "D": (4, [("H", 4)]),  
  "E": (8, [("G", 5), ("I", 5)]),  
  "F": (3, [("J", 4)]),  
  "G": (4, [("J", 5)]),  
  "H": (4, []),  
  "I": (3, [("J", 3)]),  
  "J": (0, [])  
}
```

▶ Poziv funkcije:

```
path = a_star(graf, "A", "J")
```



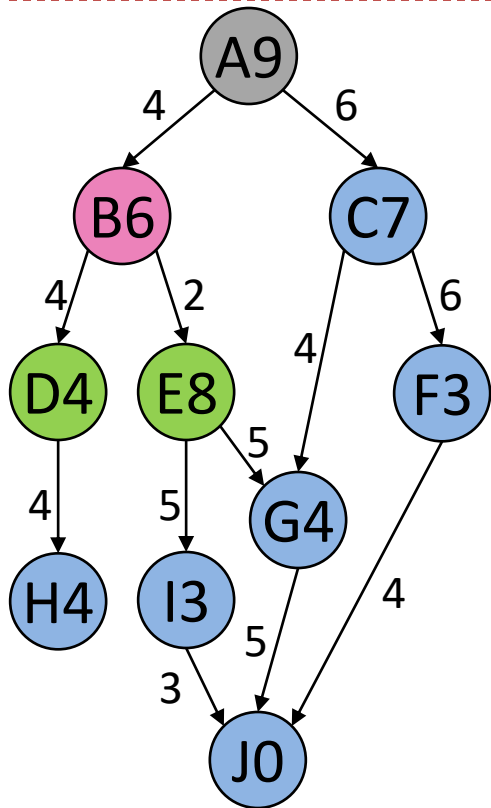
A* algoritam (ilustracija)



i	Za obradu	Potomci	Obrađeni	Prethodnici	g	f
0	A	-	-	A : None	A:0	A:9
1	B C	B C	A	B:A, C:A	B:4, C:6	B:10, C:13



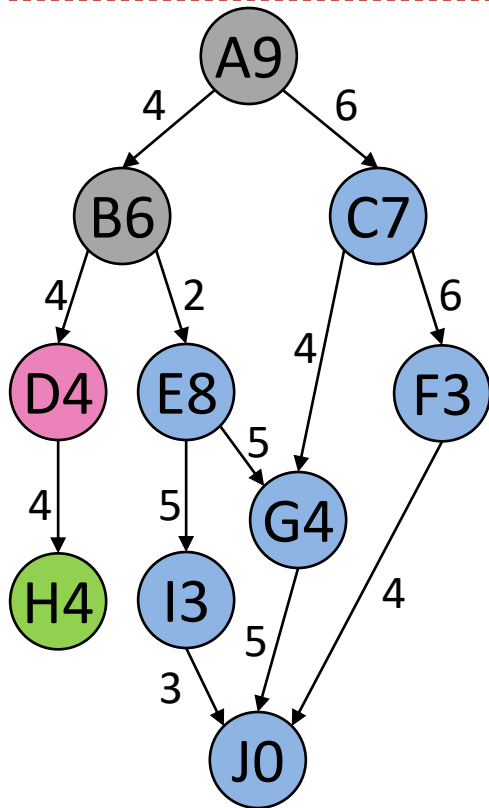
A* algoritam (ilustracija)



i	Za obradu	Potomci	Obrađeni	Prethodnici	g	f
0	A	-	-	A : None	A:0	A:9
1	B C	B C	A	B:A, C:A	B:4, C:6	B:10, C:13
2	D, C, E	D, E	A, B	D:B, E:B	D:8, E:6	D:12, E:14



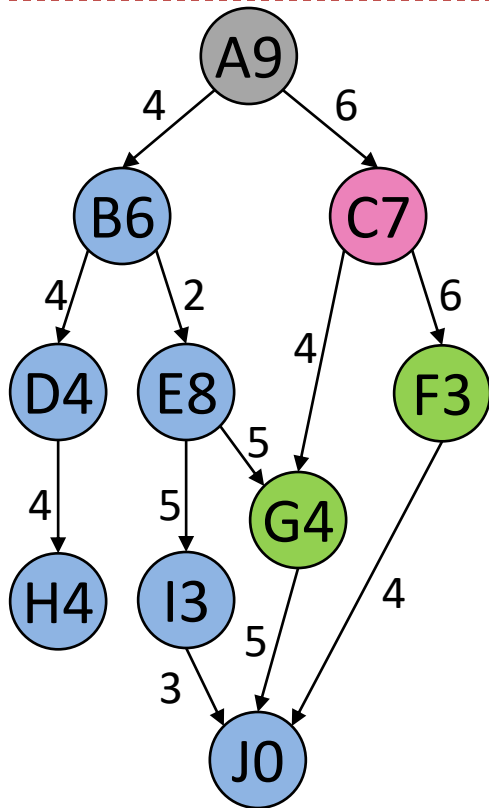
A* algoritam (ilustracija)



i	Za obradu	Potomci	Obrađeni	Prethodnici	g	f
0	A	-	-	A : None	A:0	A:9
1	B C	B C	A	B:A, C:A	B:4, C:6	B:10, C:13
2	D, C, E	D, E	A, B	D:B, E:B	D:8, E:6	D:12, E:14
3	C, E, H	H	A, B, D	H:D	H:12	H:16



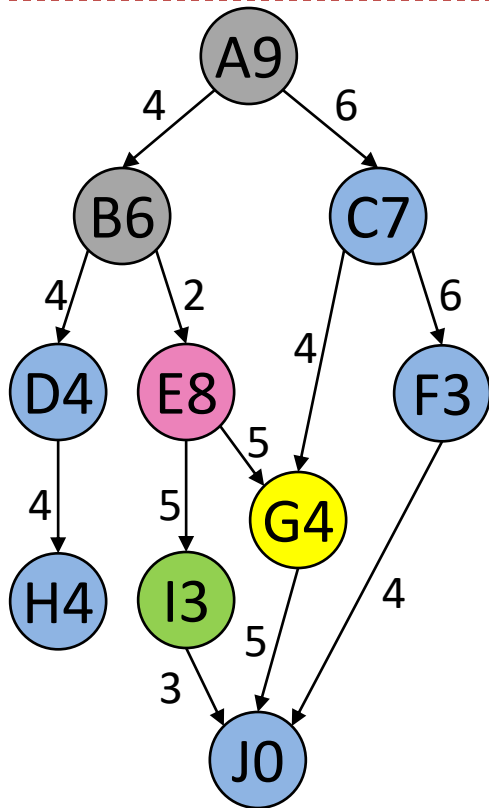
A* algoritam (ilustracija)



i	Za obradu	Potomci	Obrađeni	Prethodnici	g	f
0	A	-	-	A : None	A:0	A:9
1	B C	B C	A	B:A, C:A	B:4, C:6	B:10, C:13
2	D, C, E	D, E	A, B	D:B, E:B	D:8, E:6	D:12, E:14
3	C, E, H	H	A, B, D	H:D	H:12	H:16
4	E, G, F, H	G, F	A, B, D, C	G:C, F:C	G:10, F:12	G:14, F:15



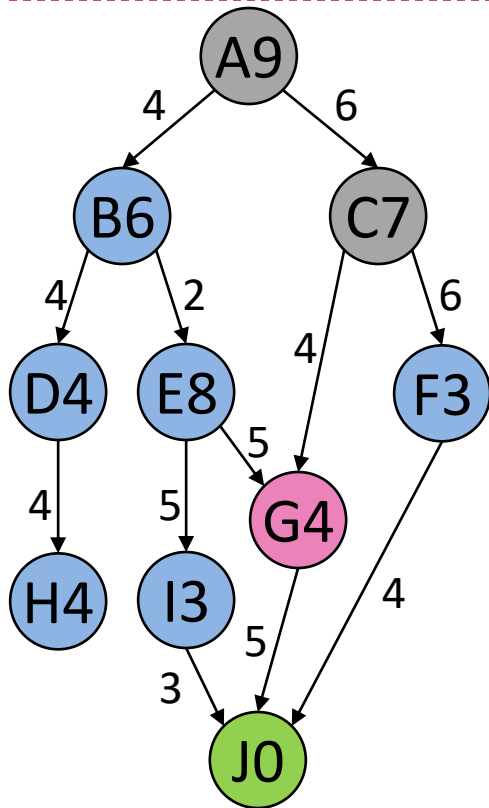
A* algoritam (ilustracija)



i	Za obradu	Potomci	Obrađeni	Prethodnici	g	f
0	A	-	-	A : None	A:0	A:9
1	B C	B C	A	B:A, C:A	B:4, C:6	B:10, C:13
2	D, C, E	D, E	A, B	D:B, E:B	D:8, E:6	D:12, E:14
3	C, E, H	H	A, B, D	H:D	H:12	H:16
4	E, G, F, H	G, F	A, B, D, C	G:C, F:C	G:10, F:12	G:14, F:15
5	G, I, F, H	I	A, B, D, C, E	I:E	I:11	I:14



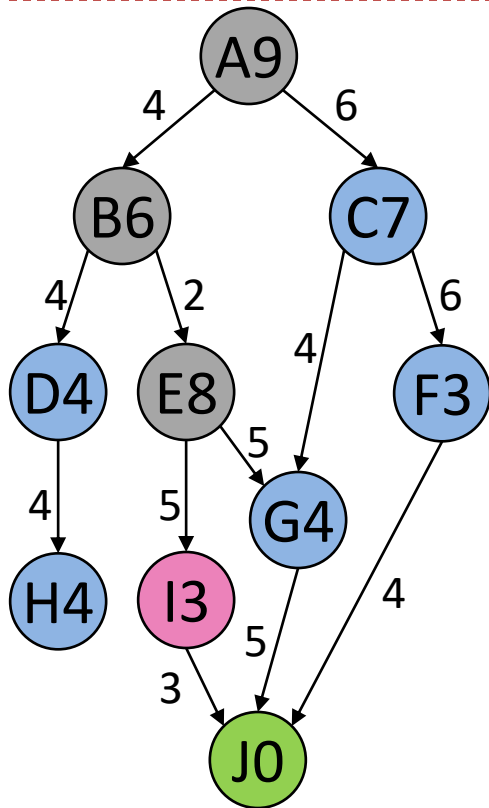
A* algoritam (ilustracija)



i	Za obradu	Potomci	Obrađeni	Prethodnici	g	f
0	A	-	-	A : None	A:0	A:9
1	B C	B C	A	B:A, C:A	B:4, C:6	B:10, C:13
2	D, C, E	D, E	A, B	D:B, E:B	D:8, E:6	D:12, E:14
3	C, E, H	H	A, B, D	H:D	H:12	H:16
4	E, G, F, H	G, F	A, B, D, C	G:C, F:C	G:10, F:12	G:14, F:15
5	G, I, F, H	I	A, B, D, C, E	I:E	I:11	I:14
6	I, J, F, H	J	A, B, D, C, E, G	J:G	J:15	J:15



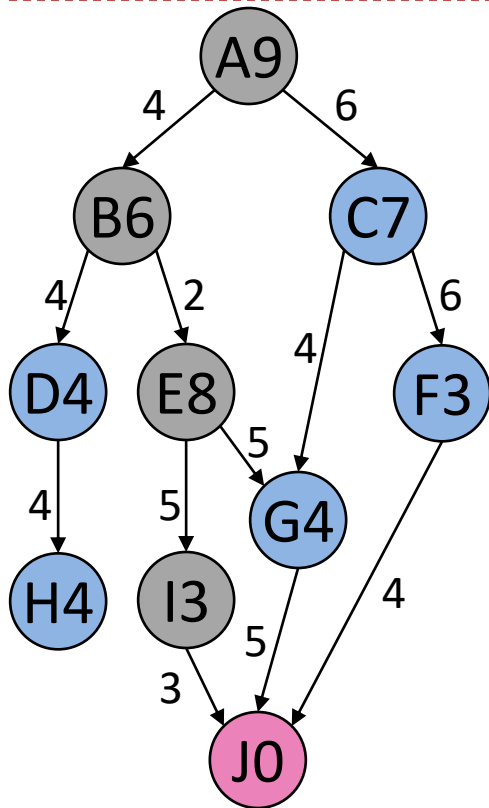
A* algoritam (ilustracija)



i	Za obradu	Potomci	Obrađeni	Prethodnici	g	f
0	A	-	-	A : None	A:0	A:9
1	B C	B C	A	B:A, C:A	B:4, C:6	B:10, C:13
2	D, C, E	D, E	A, B	D:B, E:B	D:8, E:6	D:12, E:14
3	C, E, H	H	A, B, D	H:D	H:12	H:16
4	E, G, F, H	G, F	A, B, D, C	G:C, F:C	G:10, F:12	G:14, F:15
5	G, I, F, H	I	A, B, D, C, E	I:E	I:11	I:14
6	I, J, F, H	J	A, B, D, C, E, G	J:G	J:15	J:15
7	J, F, H	J	A, B, D, C, E, G, I	J:I	J:14	J:14



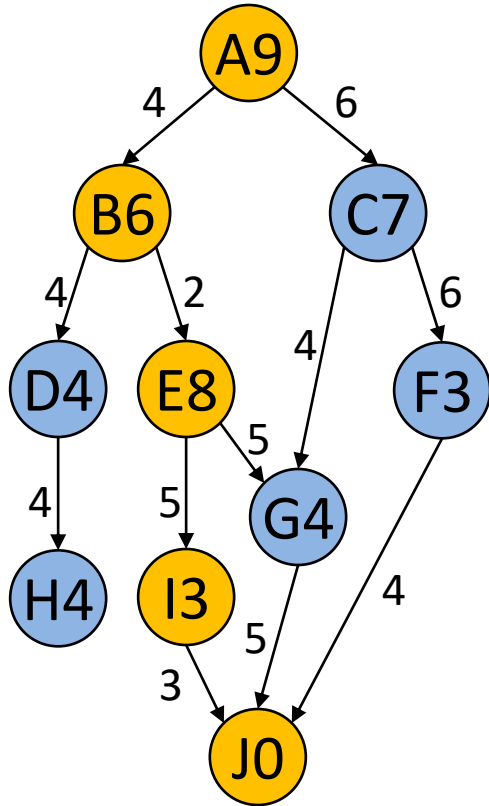
A* algoritam (ilustracija)



i	Za obradu	Potomci	Obrađeni	Prethodnici	g	f
0	A	-	-	A : None	A:0	A:9
1	B C	B C	A	B:A, C:A	B:4, C:6	B:10, C:13
2	D, C, E	D, E	A, B	D:B, E:B	D:8, E:6	D:12, E:14
3	C, E, H	H	A, B, D	H:D	H:12	H:16
4	E, G, F, H	G, F	A, B, D, C	G:C, F:C	G:10, F:12	G:14, F:15
5	G, I, F, H	I	A, B, D, C, E	I:E	I:11	I:14
6	I, J, F, H	J	A, B, D, C, E, G	J:G	J:15	J:15
7	J, F, H	J	A, B, D, C, E, G, I	J:I	J:14	J:14
8	Obradjuje se J - došli smo do cilja					



A* algoritam (ilustracija)



► Prethodnici:

{'A': None, 'B': 'A', 'C': 'A', 'D': 'B',
'E': 'B', 'H': 'D', 'G': 'C', 'F': 'C', 'I': 'E',
'J': 'I'}

► Put:

['A', 'B', 'E', 'I', 'J']



Primer rešavanja problema – A* algoritam

Problem osvajanja figure konjem

Problem osvajanja figure konjem

- ▶ Na šahovskoj tabli se nalazi proizvoljno raspoređenih 8 figura, koje ne menjaju svoja mesta.
- ▶ Rešiti problem prelaska konja sa pozicije (0, 3) na drugi kraj table.
- ▶ Rešenje treba da bude optimalno u odnosu na rastojanje koje će preći na tabli.
- ▶ Konj ne sme da stane na polje koje je već zauzeto nekom drugom figurom.



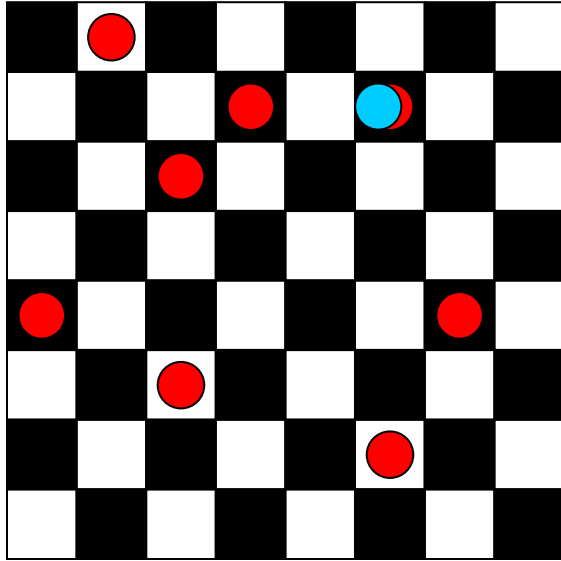
Koraci rešavanje problema

- ▶ Definisati stanja problema
- ▶ Definisati početno stanje
- ▶ Definisati ciljno stanje
- ▶ Definisati prelaze iz jednog stanja u drugo
- ▶ Definisati ograničenja pri prelazu iz jednog stanja u drugo

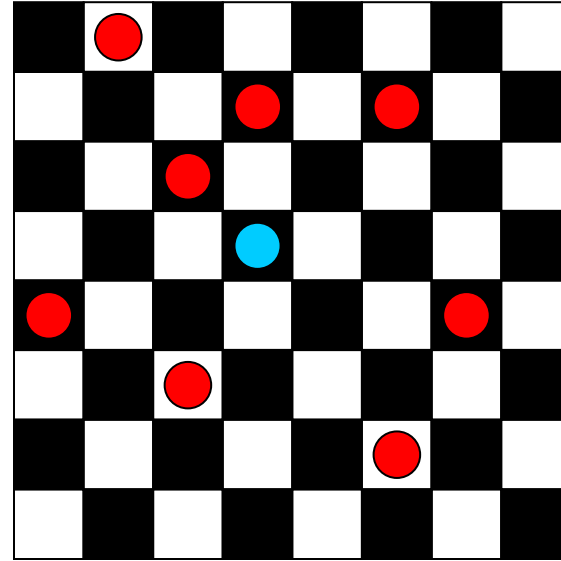


Predstavljanje stanja problema

► Neregularno stanje:

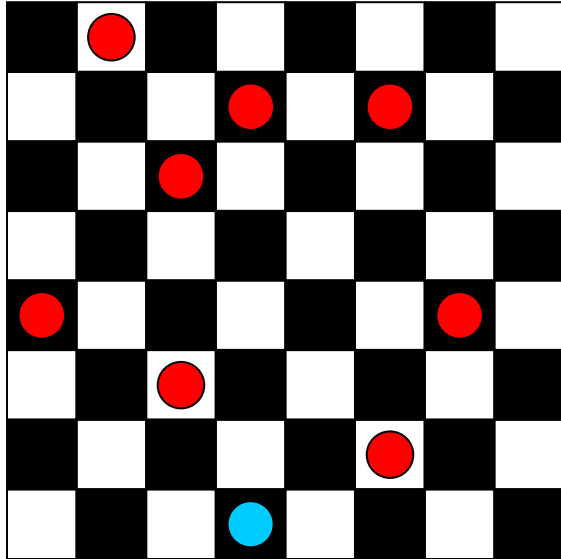


► Regularno stanje:

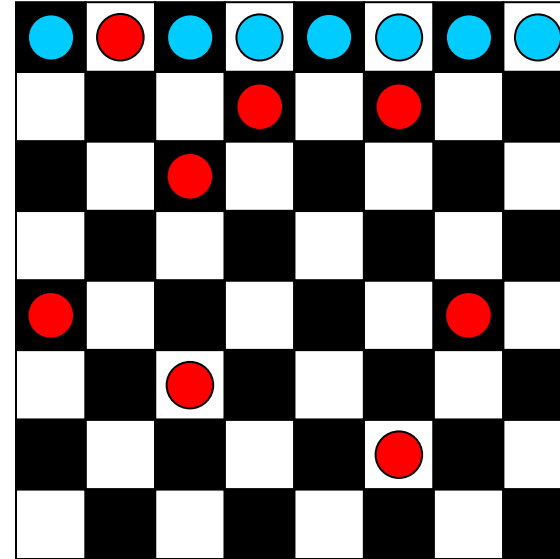


Početno i ciljno stanje

► Početno stanje:

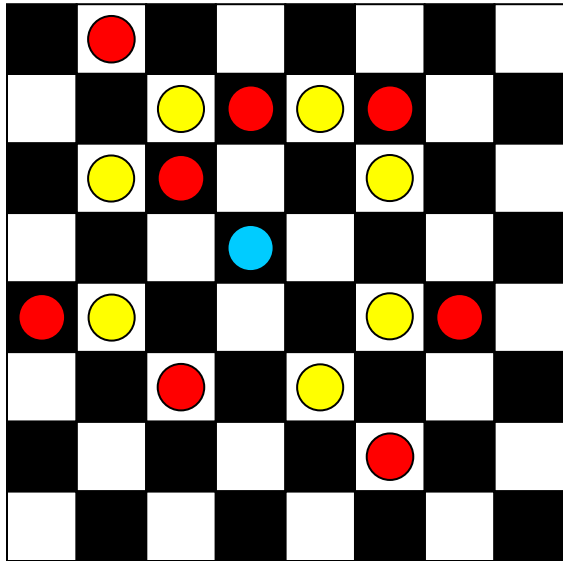


► Ciljna stanja:



Prelaz iz stanja u stanje

► Mogući prelazi:



Implementacija predstavljanje stanja

- ▶ Tabla je statička struktura 8x8, sa osam figura na fiksnim pozicijama
 - ▶ Lista od 8 listi sa po 8 elementa sa 0 tamo gde nema figura i 1 tamo gde je ima.
- ▶ Trenutno stanje definiše pozicija konja.
 - ▶ Tuple od 2 elementa (vrsta kolona)

Izmene A* algoritma

- Početak funkcije `a_star` priprema pomoćne parametre i strukture za pretragu grafa:

```
def a_star(start, tabla):  
    found_end = False  
    open_set = set()  
    closed_set = set()  
    g = {}  
    prev_nodes = {}  
    g[start] = 0  
    prev_nodes[start] = None  
    open_set.add(tuple(start, ))
```



Izmene A* algoritma

- ▶ Glavna petlja funkcije **a_star** koja obrađuje čvorove:

```
while len(open_set) > 0 and (not found_end):
    node = None
    for next_node in open_set:
        if node is None or g[next_node] + h_function(next_node) < g[node] +
h_function(node):
            node = next_node
    if node[0] == 7:
        found_end = True
        break
```



Izmene A* algoritma

- ▶ Nastavak glavne petlje, obrada potomaka tekućeg čvora:

```
for destination in get_destination(node, tabla):
    if destination not in open_set and destination not in closed_set:
        open_set.add(destination)
        prev_nodes[destination] = node
        g[destination] = g[node] + 3
    else:
        if g[destination] > g[node] + 3:
            g[destination] = g[node] + 3
            prev_nodes[destination] = node

        if destination in closed_set:
            closed_set.remove(destination)
            open_set.add(destination)
open_set.remove(node)
closed_set.add(node)
```



Izmene A* algoritma

- Kraj funkcije `a_star` formira put od početnog do cilnog čvora:

```
path = []
if found_end:
    while prev_nodes[node] is not None:
        path.append(node)
        node = prev_nodes[node]
    path.append(start)
    path.reverse()
return path
```



Funkcije specifične za problem

- ▶ Funkcija koja vraća listu potomaka jednoj stanja:

```
def get_destination(node, tabla):  
    all_destinations= [(node[0] - 2, node[1] - 1), (node[0] - 2, node[1] + 1),  
                       (node[0] - 1, node[1] - 2), (node[0] - 1, node[1] + 2),  
                       (node[0] + 1, node[1] - 2), (node[0] + 1, node[1] + 2),  
                       (node[0] + 2, node[1] - 1), (node[0] + 2, node[1] + 1)]  
    return list(x for x in all_destinations if valid_move(x, tabla))
```

- ▶ Funkcija koja proverava da li je stanje validno – da li izlazi iz granica table i da li je polje slobodno:

```
def valid_move(node, tabla):  
    if node[0] < 0 or node[0] > 7 or node[1] < 0 or node[1] > 7 or tabla[node[0]][node[1]] == 1:  
        return False  
    return True
```

Funkcije specifične za problem

```
def h_function(node):  
    return 7 - node[0]
```

```
tabla = [[0, 0, 0, 0, 0, 0, 0, 0],  
          [0, 0, 0, 0, 0, 1, 0, 0],  
          [0, 0, 1, 0, 0, 0, 0, 0],  
          [1, 0, 0, 0, 0, 0, 1, 0],  
          [0, 0, 0, 0, 0, 0, 0, 0],  
          [0, 0, 1, 0, 0, 0, 0, 0],  
          [0, 0, 0, 1, 0, 1, 0, 0],  
          [0, 1, 0, 0, 0, 0, 0, 0]]
```



Rešenje problema

