



# Veštačka inteligencija



Algoritmi za traženje u Python-u (I deo)

# Neinformisani (slepi) algoritmi

---

- ▶ Traženje po širini (breadth-first search)
- ▶ Traženje po dubini (depth-first search)
- ▶ Traženje sa uniformnom cenom
- ▶ Traženje ograničeno po dubini
- ▶ Bidirekciono traženje
- ▶ Nedeterminističko traženje



# Traženje po širini

---

- ▶ Formirati **red** koji inicijalno sadrži samo polazni čvor.
- ▶ Dok se **red** ne isprazni ili se ne dođe do ciljnog čvora, proveriti da li je prvi element ciljni čvor
  - ▶ Ako je prvi element ciljni čvor, ne raditi ništa.
  - ▶ Ako prvi element nije ciljni čvor, ukloniti ga iz **reda** i dodati sve njegove sledbenike (ako ih ima i ako nisu već posećeni) u **red**.
- ▶ Ako je pronađen ciljni čvor, pretraga je uspešno završena, u suprotnom pretraga je neuspešna.



# Traženje po dubini

---

- ▶ Formirati **stek** koji inicijalno sadrži samo polazni čvor.
- ▶ Dok se **stek** ne isprazni ili se ne dođe do ciljnog čvora, proveriti da li je prvi element ciljni čvor
  - ▶ Ako je prvi element ciljni čvor, ne raditi ništa.
  - ▶ Ako prvi element nije ciljni čvor, ukloniti ga iz **steka** i dodati sve njegove sledbenike (ako ih ima i ako nisu već posećeni) na **stek**.
- ▶ Ako je pronađen ciljni čvor, pretraga je uspešno završena, u suprotnom pretraga je neuspešna.



# Traženje po širini i dubini

---

## ▶ Format grafa

- ▶ { nodel : [dest11, dest12, ... ],
- ▶ node2 : [dest21, dest22 , ... ],
- ▶ ... }

## ▶ Funkcije

- ▶ breadth\_first\_search(graph, start, end)
- ▶ depth\_first\_search(graph, start, end)



# Traženje po širini

---

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



# Traženje po širini

---

- ▶ Pomoćne strukture i promenljive koje koristi funkcija `breadth_first_search`:
  - ▶ Red čvorova (oznaka) koje treba posetiti
    - ▶ `queue_nodes`
  - ▶ Niz posećenih čvorova (oznaka)
    - ▶ `visited`
  - ▶ Niz parova čvorova (oznaka) oblika (čvor prethodnik)
    - ▶ `prev_nodes`
  - ▶ Niz čvorova (oznaka) na putu od polaznog do ciljnog čvora
    - ▶ `path`
  - ▶ Logička promenljiva koja ukazuje da je pronađen ciljni čvor
    - ▶ `found_dest`



# Traženje po širini – Algoritam

---

- ▶ Dodati polazni čvor u red
- ▶ Dodati polazni čvor u posećene
- ▶ Dodati da polazni čvor nema prethodnika
- ▶ Sve dok ciljni čvor nije pronađen i red nije prazan
  - ▶ Pročitati čvor iz reda
  - ▶ Obraditi čvor
  - ▶ Za svaki odredišni čvor do kog postoji poteg od obrađenog čvora
    - ▶ Ako odredište nije posećeno
      - Postaviti prethodnika za odredište na obrađeni čvor
      - Ako je odredište jednako ciljnom čvoru postavi vrednost na pronađen i izaći iz petlje
      - Dodati odredište u posećene
      - Dodati odredište u red
- ▶ Ako je ciljni čvor pronađen:
  - ▶ Dodati ciljni čvor u put
  - ▶ Postaviti trenutni čvor na prethodnika ciljnog čvora
  - ▶ Sve dok prethodnik postoji
    - ▶ Dodati ciljni čvor u put
    - ▶ Postaviti trenutni čvor na prethodnika trenutnog čvora





# Funkcija `breadth_first_search` (I)

---

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

```
def breadth_first_search(graph, start, end):
```

```
    if start is end:
```

```
        path = list()
```

```
        path.append(start)
```

```
        return path
```

```
    queue_nodes = queue.Queue(len(graph))
```

```
    visited = set()
```

```
    prev_nodes = dict()
```

```
    prev_nodes[start] = None
```

```
    visited.add(start)
```

```
    queue_nodes.put(start)
```

```
    found_dest = False
```



# Funkcija breadth\_first\_search (II)

---

- ▶ Glavna petlja funkcije `breadth_first_search` koja obrađuje čvorove korišćenjem reda

```
while (not found_dest) and (not queue_nodes.empty()):  
    node = queue_nodes.get()  
    process(node)  
    for dest in graph[node]:  
        if dest not in visited:  
            prev_nodes[dest] = node  
            if dest is end:  
                found_dest = True  
                break  
    visited.add(dest)  
    queue_nodes.put(dest)
```



# Funkcija breadth\_first\_search (III)

---

- Kraj funkcije `breadth_first_search` formira put od početnog do ciljnog čvora

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



# Traženje po širini – Primer grafa i poziva

---

- ▶ Primer grafa

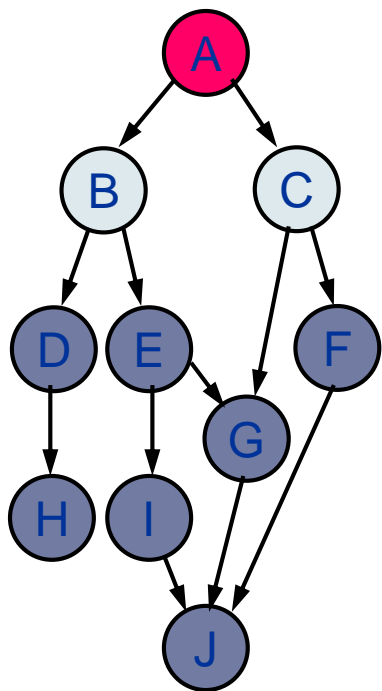
```
graph_simple = {  
    'A' : ['B', 'C'],  
    'B' : ['D', 'E'],  
    'C' : ['F', 'G'],  
    'D' : ['H'],  
    'E' : ['G', 'I'],  
    'F' : ['J'],  
    'G' : ['J'],  
    'H' : [],  
    'I' : ['J'],  
    'J' : []  
}
```

- ▶ Primer poziva funkcije za pretagu po širini

```
path = breadth_first_search(graph, start, end)
```

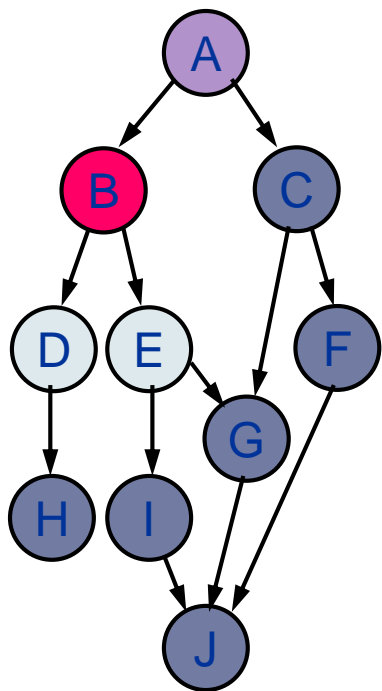


# Traženje po širini (ilustracija)



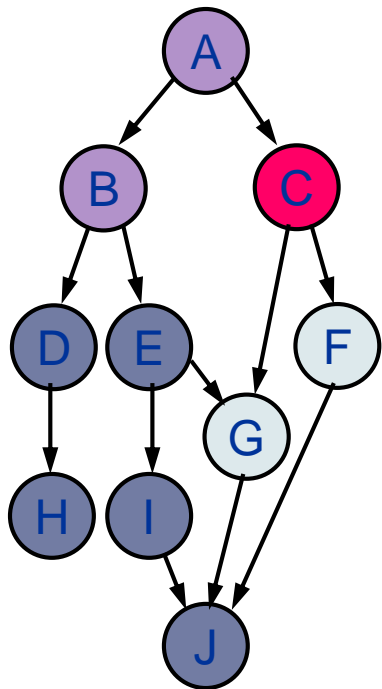
d. r.	za obradu	potomci	obrađeni	prethodnik
0	A	-	-	(A -)
1	B C	B C	A	(B A) (C A)

# Traženje po širini (ilustracija)



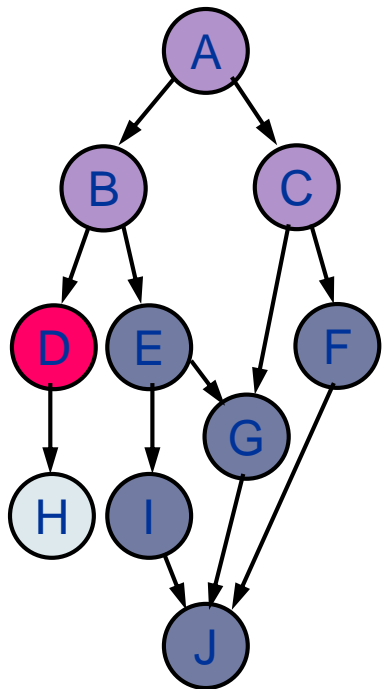
d. r.	za obradu	potomci	obrađeni	prethodnik
0	A	-	-	(A -)
1	B C	B C	A	(B A) (C A)
2	C D E	D E	A B	(D B) (E B)

# Traženje po širini (ilustracija)



d. r.	za obradu	potomci	obrađeni	prethodnik
0	A	-	-	(A -)
1	B C	B C	A	(B A) (C A)
2	C D E	D E	A B	(D B) (E B)
3	D E F G	F G	A B C	(F C) (G C)

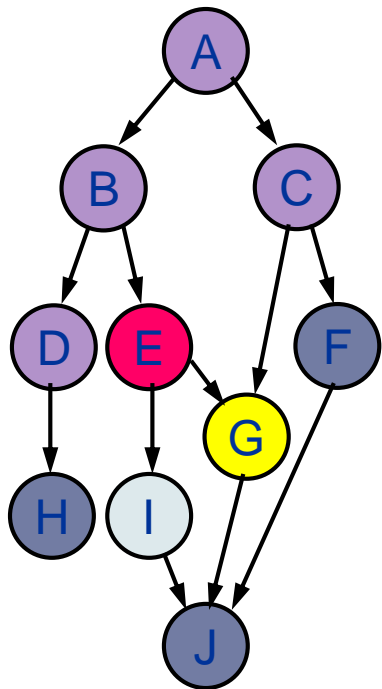
# Traženje po širini (ilustracija)



d. r.	za obradu	potomci	obrađeni	prethodnik
0	A	-	-	(A -)
1	B C	B C	A	(B A) (C A)
2	C D E	D E	A B	(D B) (E B)
3	D E F G	F G	A B C	(F C) (G C)
4	E F G H	H	A B C D	(H D)

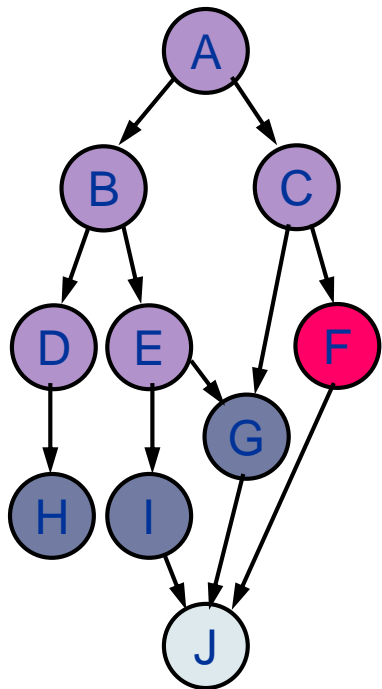


# Traženje po širini (ilustracija)



d. r.	za obradu	potomci	obrađeni	prethodnik
0	A	-	-	(A -)
1	B C	B C	A	(B A) (C A)
2	C D E	D E	A B	(D B) (E B)
3	D E F G	F G	A B C	(F C) (G C)
4	E F G H	H	A B C D	(H D)
5	F G H I	I	A B C D E	(I E)

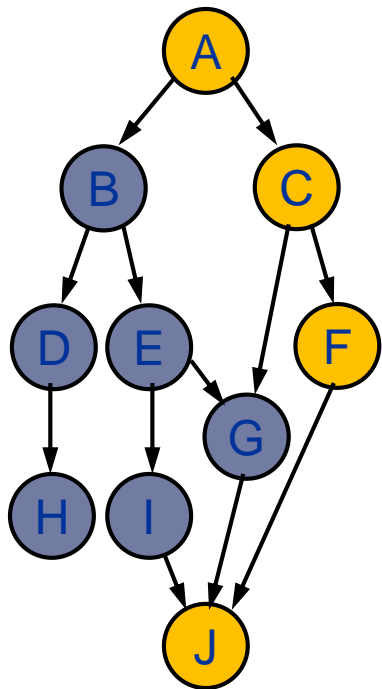
# Traženje po širini (ilustracija)



d. r.	za obradu	potomci	obrađeni	prethodnik
0	A	-	-	(A -)
1	B C	B C	A	(B A) (C A)
2	C D E	D E	A B	(D B) (E B)
3	D E F G	F G	A B C	(F C) (G C)
4	E F G H	H	A B C D	(H D)
5	F G H I	I	A B C D E	(I E)
6	G H I J	J	A B C D E F	(J F)

# Traženje po širini (ilustracija)

---



## ► Prethodnici

- (A -) (BA) (CA) (D B) (E B) (F C)  
(G C) (H D) (I E) (J F)

## ► Put

**(A C F J)**



# Traženje po dubini

---

- ▶ Formirati **stek** koji inicijalno sadrži samo polazni čvor.
- ▶ Dok se **stek** ne isprazni ili se ne dođe do ciljnog čvora, proveriti da li je prvi element ciljni čvor
  - ▶ Ako je prvi element ciljni čvor, ne raditi ništa.
  - ▶ Ako prvi element nije ciljni čvor, ukloniti ga iz **steka** i dodati sve njegove sledbenike (ako ih ima i ako nisu već posećeni) na **stek**.
- ▶ Ako je pronađen ciljni čvor, pretraga je uspešno završena, u suprotnom pretraga je neuspešna.



# Traženje po dubini

---

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



# Traženje po dubini





---

- ▶ Pomoćne strukture i promenljive koje koristi funkcija `depth_first_search`:
  - ▶ Stek čvorova (oznaka) koje treba posetiti
    - ▶ `stack_nodes`
  - ▶ Niz posećenih čvorova (oznaka)
    - ▶ `visited`
  - ▶ Niz parova čvorova (oznaka) oblika (čvor prethodnik)
    - ▶ `prev_nodes`
  - ▶ Niz čvorova (oznaka) na putu od polaznog do ciljnog čvora
    - ▶ `path`
  - ▶ Logička promenljiva koja ukazuje da je pronađen ciljni čvor
    - ▶ `found_dest`



# Traženje po dubini – Algoritam

---

- ▶ Dodati polazni čvor na stek 
- ▶ Dodati polazni čvor u posećene
- ▶ Dodati da polazni čvor nema prethodnika
- ▶ Sve dok ciljni čvor nije pronađen i stek nije prazan 
  - ▶ Pročitati čvor iz steka 
  - ▶ Obraditi čvor
  - ▶ Za svaki odredišni čvor do kog postoji poteg od obrađenog čvora
    - ▶ Ako odredište nije posećeno
      - ☐ Postaviti prethodnika za odredište na obrađeni čvor
      - ☐ Ako je odredište jednako ciljnom čvoru postavi vrednost na pronađen i izaći iz petlje
      - ☐ Dodati odredište u posećene
      - ☐ Dodati odredište u stek 
- ▶ Ako je ciljni čvor pronađen:
  - ▶ Dodati ciljni čvor u put
  - ▶ Postaviti trenutni čvor na prethodnika ciljnog čvora
  - ▶ Sve dok prethodnik postoji
    - ▶ Dodati ciljni čvor u put
    - ▶ Postaviti trenutni čvor na prethodnika trenutnog čvora



# Funkcija `depth_first_search` (I)

---

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

```
def depth_first_search(graph, start, end):  
    if start is end:  
        path = list()  
        path.append(start)  
        return path  
  
    stack_nodes = queue.LifoQueue(len(graph))  
    visited = set()  
    prev_nodes = dict()  
    prev_nodes[start] = None  
    visited.add(start)  
    stack_nodes.put(start)  
    found_dest = False
```





# Funkcija depth\_first\_search (II)

---

- ▶ Glavna petlja funkcije `depth_first_search` koja obrađuje čvorove korišćenjem reda

```
while (not found_dest) and (not stack_nodes.empty()):  
    node = stack_nodes.get()  
    process(node)  
    #for dest in graph[node]:  
    for dest in reversed(graph[node]):  
        if dest not in visited:  
            prev_nodes[dest] = node  
            if dest is end:  
                found_dest = True  
                break  
            visited.add(dest)  
            stack_nodes.put(dest)
```



# Funkcija depth\_first\_search (III)

---

- Kraj funkcije `depth_first_search` formira put od početnog do ciljnog čvora

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



# Traženje po dubini – Primer grafa i poziva

---

- ▶ Primer grafa

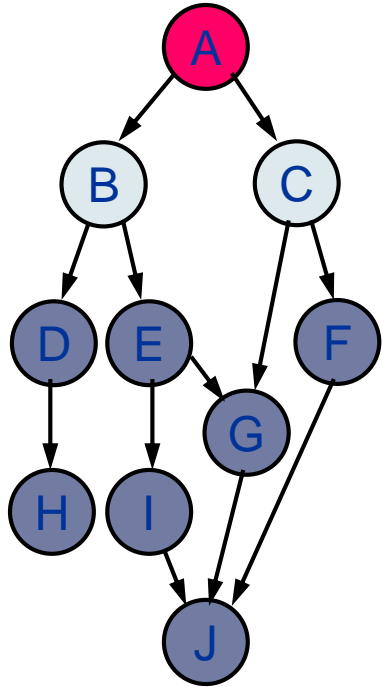
```
graph_simple = {  
    'A' : ['B', 'C'],  
    'B' : ['D', 'E'],  
    'C' : ['F', 'G'],  
    'D' : ['H'],  
    'E' : ['G', 'I'],  
    'F' : ['J'],  
    'G' : ['J'],  
    'H' : [],  
    'I' : ['J'],  
    'J' : []  
}
```

- ▶ Primer poziva funkcije za pretagu po dubini

```
path = depth_first_search(graph, start, end)
```

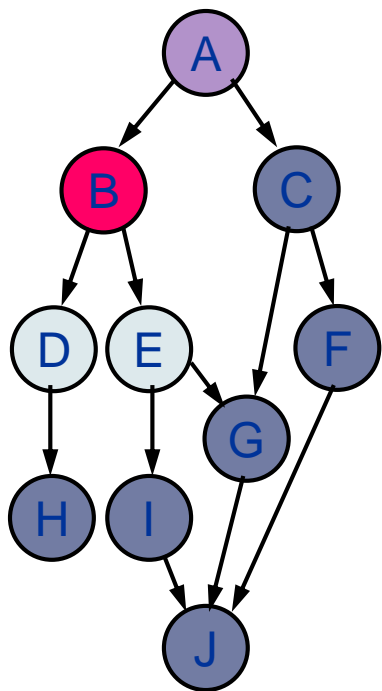


# Traženje po dubini (ilustracija)



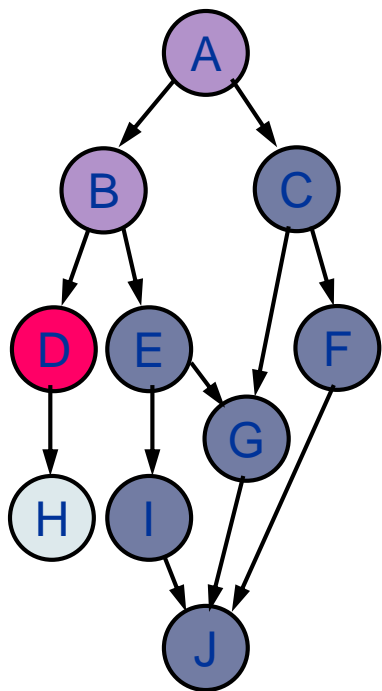
d. r.	za obradu	potomci	obrađeni	prethodnik
0	A	-	-	(A -)
1	B C	B C	A	(B A) (C A)

# Traženje po dubini (ilustracija)



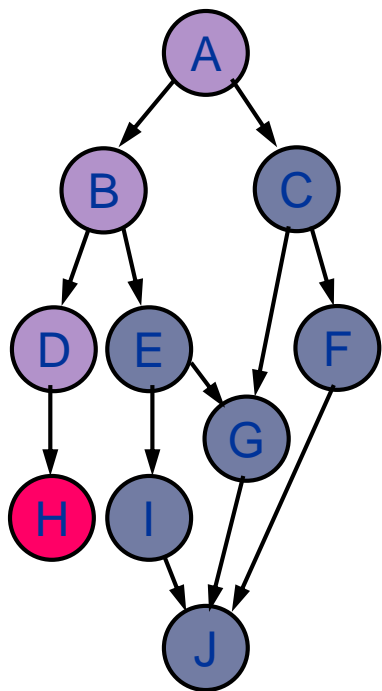
d. r.	za obradu	potomci	obrađeni	prethodnik
0	A	-	-	(A -)
1	B C	B C	A	(B A) (C A)
2	D E C	D E	A B	(D B) (E B)

# Traženje po dubini (ilustracija)



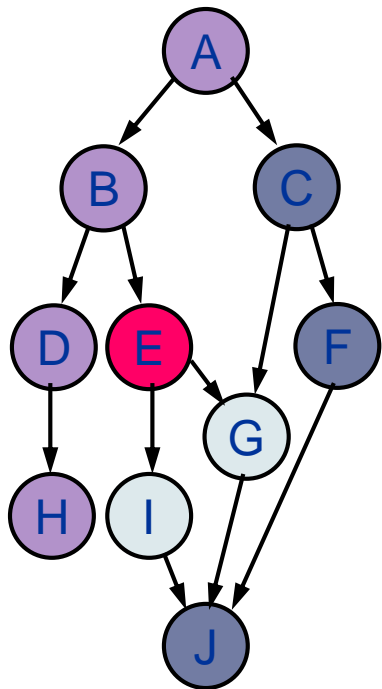
d. r.	za obradu	potomci	obrađeni	prethodnik
0	A	-	-	(A -)
1	B C	B C	A	(B A) (C A)
2	D E C	D E	A B	(D B) (E B)
3	H E C	H	A B D	(H D)

# Traženje po dubini (ilustracija)



d. r.	za obradu	potomci	obrađeni	prethodnik
0	A	-	-	(A -)
1	B C	B C	A	(B A) (C A)
2	D E C	D E	A B	(D B) (E B)
3	H E C	H	A B D	(H D)
4	E C	-	A B D H	-

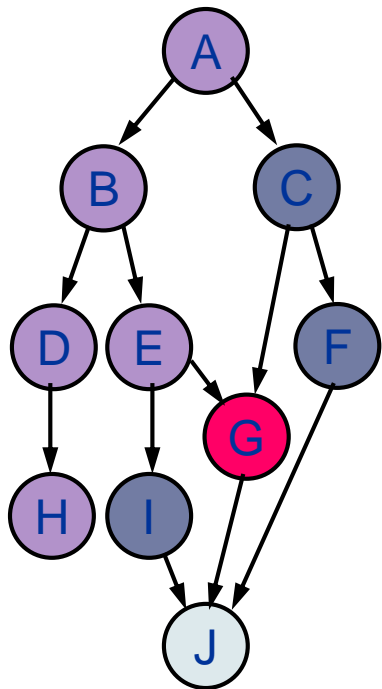
# Traženje po dubini (ilustracija)



d. r.	za obradu	potomci	obrađeni	prethodnik
0	A	-	-	(A -)
1	B C	B C	A	(B A) (C A)
2	D E C	D E	A B	(D B) (E B)
3	H E C	H	A B D	(H D)
4	E C	-	A B D H	-
5	G I C	G I	A B D H E	(G E) (I E)



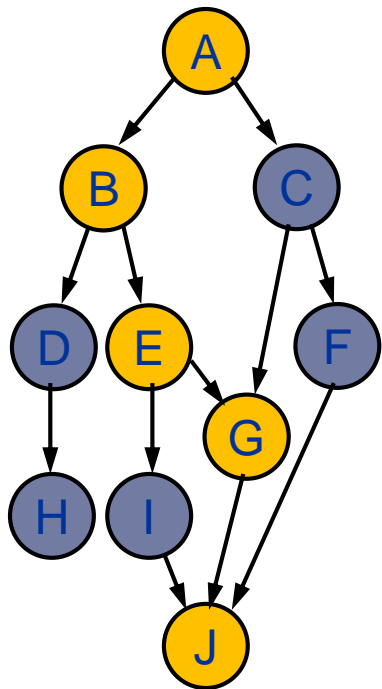
# Traženje po dubini (ilustracija)



d. r.	za obradu	potomci	obrađeni	prethodnik
0	A	-	-	(A -)
1	B C	B C	A	(B A) (C A)
2	D E C	D E	A B	(D B) (E B)
3	H E C	H	A B D	(H D)
4	E C	-	A B D H	-
5	G I C	G I	A B D H E	(G E) (I E)
6	J I C	J	A B D H E G	(J G)

# Traženje po dubini (ilustracija)

---



## ► Prethodnici

► (A -) (BA) (CA) (D B) (E B) (H D)  
(G E) (I E) (J G)

## ► Put

**(A B E G J)**



# Informisani (heuristički) algoritmi

---

- ▶ Metod planinarenja (Hill-Climbing)
- ▶ Prvo najbolji (Best-First)
- ▶ Algoritmi predstavljaju proširenje traženja po dubini korišćenjem informacije o rastojanju svakog čvora do cilja.
- ▶ Heuristika je procenjena cena puta od konkretnog čvora do cilja:
  - ▶  $h(G) = 0$
  - ▶  $h(c) > 0, c \neq G$
- ▶ Cilj heuristike je da se za obradu biraju čvorovi koji više obećavaju da će se stići do cilja (manja im je procenjena cena puta do cilja).



# Metod planinarenja

---

- ▶ Formirati **stek** koji inicijalno sadrži samo polazni čvor.
- ▶ Dok se **stek** ne isprazni ili se ne dođe do ciljnog čvora, proveriti da li je prvi element ciljni čvor
  - ▶ Ako je prvi element ciljni čvor, put je pronađen.
  - ▶ Ako prvi element nije ciljni čvor,
    - ▶ Ukloniti ga iz **steka**
    - ▶ **Urediti njegove sledbenike** (ako ih ima i ako nisu već posećeni) po rastućim vrednostima heurističke funkcije (rastojanje od ciljnog čvora)
    - ▶ Uređene sledbenike dodati na **stek** tako da prvi element bude sledbenik sa najmanjom vrednošću heurističke funkcije.
- ▶ Ako je pronađen ciljni čvor, pretraga je uspešno završena, u suprotnom pretraga je neuspešna.



# Prvo najbolji

---

- ▶ Formirati **stek** koji inicijalno sadrži samo polazni čvor.
- ▶ Dok se **stek** ne isprazni ili se ne dođe do ciljnog čvora, proveriti da li je prvi element ciljni čvor
  - ▶ Ako je prvi element ciljni čvor, put je pronađen.
  - ▶ Ako prvi element nije ciljni čvor,
    - ▶ Ukloniti ga sa **steka**
    - ▶ Sledbenike dodati u **stek**
    - ▶ **Celokupni stek sortirati** po rastućim vrednostima heurističkih funkcija čvorova (rastojanje od ciljnog čvora)
- ▶ Ako je pronađen ciljni čvor, pretraga je uspešno završena, u suprotnom pretraga je neuspešna.



# Traženje planinarenjem

---

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



# Traženje planinarenjem

---

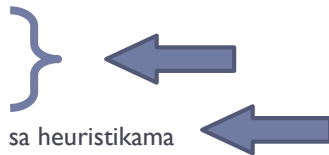
- ▶ Pomoćne strukture i promenljive koje koristi funkcija `hill_climbing_search`:
  - ▶ Stek čvorova (oznaka) koje treba posetiti
    - ▶ `stack_nodes`
  - ▶ Niz posećenih čvorova (oznaka)
    - ▶ `visited`
  - ▶ Niz parova čvorova (oznaka) oblika (čvor prethodnik)
    - ▶ `prev_nodes`
  - ▶ Niz čvorova (oznaka) na putu od polaznog do ciljnog čvora
    - ▶ `path`
  - ▶ Logička promenljiva koja ukazuje da je pronađen ciljni čvor
    - ▶ `found_dest`
  - ▶ Niz parova čvorova (oznaka) oblika (čvor heuristika)
    - ▶ `destinations`



# Traženje planinarenjem – Algoritam

---

- ▶ Dodati polazni čvor na stek
- ▶ Dodati polazni čvor u posećene
- ▶ Dodati da polazni čvor nema prethodnika
- ▶ Sve dok ciljni čvor nije pronađen i stek nije prazan
  - ▶ Pročitati čvor iz steka
  - ▶ Obraditi čvor
  - ▶ Za svaki odredišni čvor do koji postoji poteg od čvora
    - ▶ Dodati par (odredište heuristika) u listu odredišta sa heuristikama
  - ▶ Urediti listu odredišta sa heuristikama u opadajući redosled
  - ▶ Za sve parove odredište heuristika iz uređene liste odredišta sa heuristikama
    - ▶ Ako je odredište nije posećeno
      - ☐ Postaviti prethodnika za odredište na čvor
      - ☐ Ako je odredište jednako ciljnom čvoru postavi vrednost na pronađen i izaći iz petlje
      - ☐ Dodati odredište u posećene
      - ☐ Dodati odredište u stek
- ▶ Ako je ciljni čvor pronađen:
  - ▶ Dodati ciljni čvor u put
  - ▶ Postaviti trenutni čvor na prethodnika ciljnog čvora
  - ▶ Sve dok prethodnik postoji
    - ▶ Dodati ciljni čvor u put
    - ▶ Postaviti trenutni čvor na prethodnika trenutnog čvora





# Funkcija `hill_climbing_search` (I)

---

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

```
def hill_climbing_search(graph, start, end):  
    if start is end:  
        path = list()  
        path.append(start)  
        return path  
  
    stack_nodes = queue.LifoQueue(len(graph))  
    visited = set()  
    prev_nodes = dict()  
    prev_nodes[start] = None  
    visited.add(start)  
    stack_nodes.put(start)  
    found_dest = False
```



# Funkcija hill\_climbing\_search (II)

---

- ▶ Glavna petlja funkcije `hill_climbing_search` koja obrađuje čvorove korišćenjem reda

```
while (not found_dest) and (not stack_nodes.empty()):
    node = stack_nodes.get()
    process(node)
    destinations = list()
    for dest in graph[node][1]:
        element = (graph[dest][0], dest)
        destinations.append(element)
    for dest_heur in sorted(destinations, reverse=True):
        if dest_heur[1] not in visited:
            prev_nodes[dest_heur[1]] = node
            if dest_heur[1] is end:
                found_dest = True
                break
        visited.add(dest_heur[1])
        stack_nodes.put(dest_heur[1])
```



# Funkcija hill\_climbing\_search (III)

---

- Kraj funkcije `hill_climbing_search` formira put od početnog do ciljnog čvora

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



# Traženje planinarenjem – Primer grafa i poziva

---

## ▶ Primer grafa

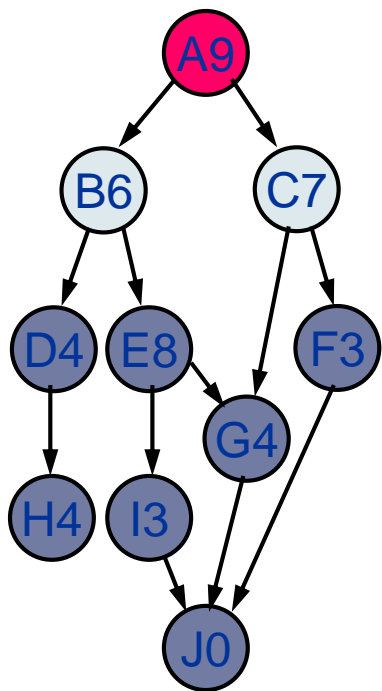
```
graph_simple = {  
    'A' : (9, ['B', 'C']),  
    'B' : (6, ['D', 'E']),  
    'C' : (7, ['F', 'G']),  
    'D' : (4, ['H']),  
    'E' : (8, ['G', 'I']),  
    'F' : (3, ['J']),  
    'G' : (4, ['J']),  
    'H' : (4, []),  
    'I' : (3, ['J']),  
    'J' : (0, [])  
}
```

## ▶ Primer poziva funkcije za pretragu planinarenjem

```
path = hill_climbing_search(graph, start, end)
```

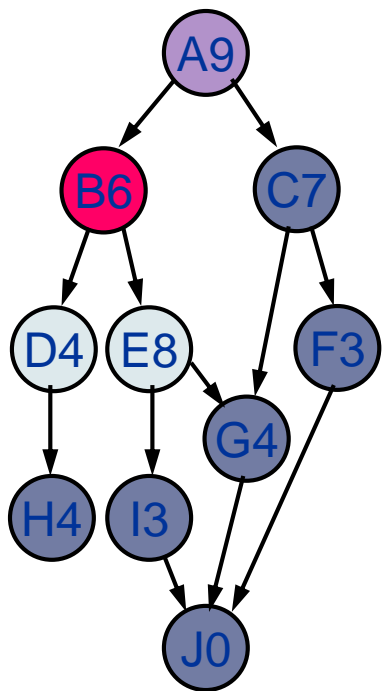


# Algoritam planinarenja (ilustracija)



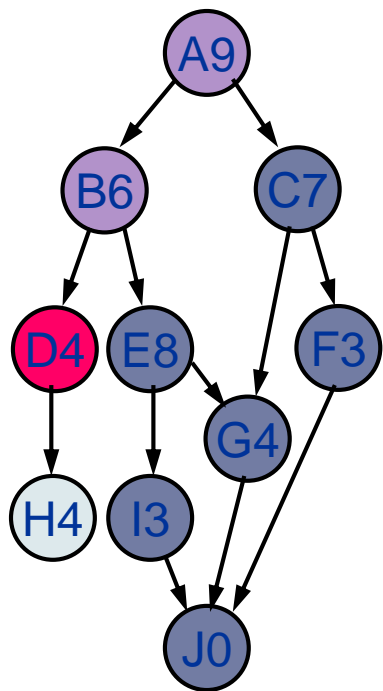
d.r.	za obradu	potomci	obrađeni	prethodnik
0	A	-	-	(A -)
1	B C	B6 C7	A	(B A) (C A)

# Algoritam planinarenja (ilustracija)



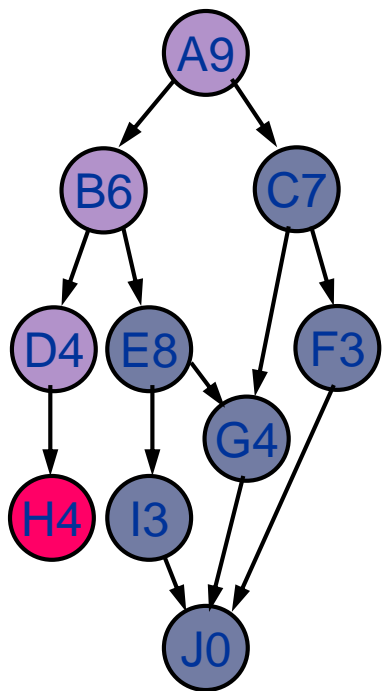
d.r.	za obradu	potomci	obrađeni	prethodnik
0	A	-	-	(A -)
1	B C	B6 C7	A	(B A) (C A)
2	D E C	D4 E8	A B	(D B) (E B)

# Algoritam planinarenja (ilustracija)



d.r.	za obradu	potomci	obrađeni	prethodnik
0	A	-	-	(A -)
1	B C	B6 C7	A	(B A) (C A)
2	D E C	D4 E8	A B	(D B) (E B)
3	H E C	H4	A B D	(H D)

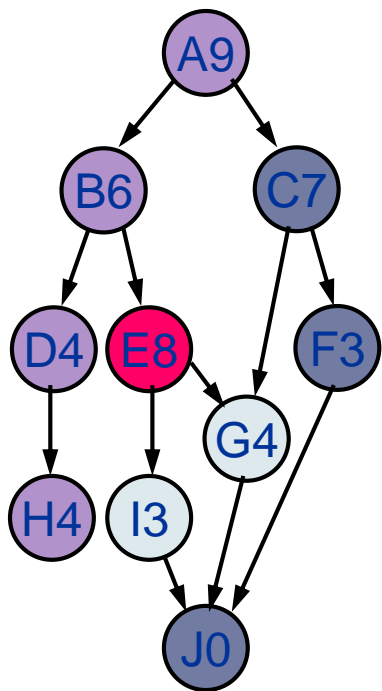
# Algoritam planinarenja (ilustracija)



d.r.	za obradu	potomci	obrađeni	prethodnik
0	A	-	-	(A -)
1	B C	B6 C7	A	(B A) (C A)
2	D E C	D4 E8	A B	(D B) (E B)
3	H E C	H4	A B D	(H D)
4	E C	-	A B D H	-

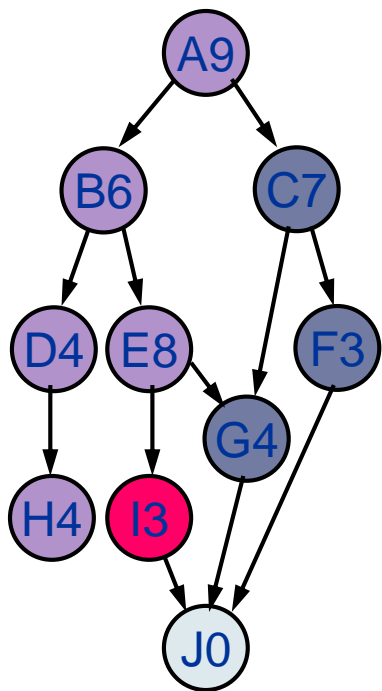


# Algoritam planinarenja (ilustracija)



d.r.	za obradu	potomci	obrađeni	prethodnik
0	A	-	-	(A -)
1	B C	B6 C7	A	(B A) (C A)
2	D E C	D4 E8	A B	(D B) (E B)
3	H E C	H4	A B D	(H D)
4	E C	-	A B D H	-
5	I G C	I3 G4	A B D H E	(G E) (I E)

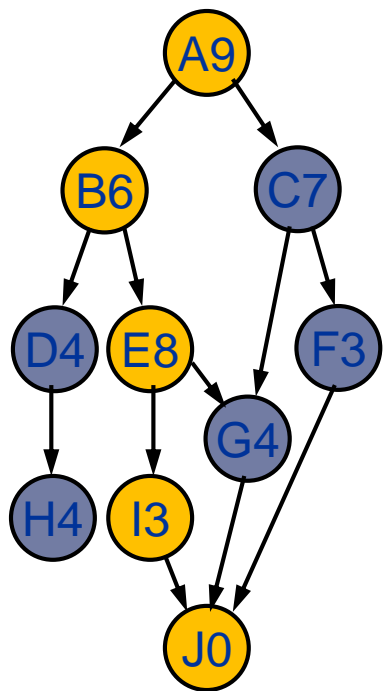
# Algoritam planinarenja (ilustracija)



d.r.	za obradu	Potomci	obrađeni	prethodnik
0	A	-	-	(A -)
1	B C	B6 C7	A	(B A) (C A)
2	D E C	D4 E8	A B	(D B) (E B)
3	H E C	H4	A B D	(H D)
4	E C	-	A B D H	-
5	I G C	I3 G4	A B D H E	(G E) (I E)
6	J G E C	J0	A B D H E I	(J I)

# Algoritam planinarenja (ilustracija)

---



## ► Prethodnici

► (A -) (B A) (C A) (D B) (E B) (H D)  
(G E) (I E) (J I)

## ► Put

**(A B E I J)**

# Prvo najbolji

---

- ▶ Formirati **stek** koji inicijalno sadrži samo polazni čvor.
- ▶ Dok se **stek** ne isprazni ili se ne dođe do ciljnog čvora, proveriti da li je prvi element ciljni čvor
  - ▶ Ako je prvi element ciljni čvor, put je pronađen.
  - ▶ Ako prvi element nije ciljni čvor,
    - ▶ Ukloniti ga sa **steka**
    - ▶ Sledbenike dodati u **stek**
    - ▶ **Celokupni stek sortirati** po rastućim vrednostima heurističkih funkcija čvorova (rastojanje od ciljnog čvora)
- ▶ Ako je pronađen ciljni čvor, pretraga je uspešno završena, u suprotnom pretraga je neuspešna.



# Traženje prvo-najbolji

---

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



# Traženje prvo-najbolji





---

- ▶ Pomoćne strukture i promenljive koje koristi funkcija `best_first_search`:
  - ▶ Red sa prioritetom čvorova (oznaka) sa heuristikama koje treba posetiti
    - ▶ `priority_queue`
  - ▶ Niz posećenih čvorova (oznaka)
    - ▶ `visited`
  - ▶ Niz parova čvorova (oznaka) oblika (čvor prethodnik)
    - ▶ `prev_nodes`
  - ▶ Niz čvorova (oznaka) na putu od polaznog do ciljnog čvora
    - ▶ `path`
  - ▶ Logička promenljiva koja ukazuje da je pronađen ciljni čvor
    - ▶ `found_dest`



# Traženje prvo-najbolji – Algoritam

---

- ▶ Dodati polazni čvor sa heuristikom u red sa prioritetom 
- ▶ Dodati polazni čvor u posećene
- ▶ Dodati da polazni čvor nema prethodnika
- ▶ Sve dok ciljni čvor nije pronađen i red sa prioritetom nije prazan 
  - ▶ Pročitati čvor sa heuristikom iz reda sa prioritetom 
  - ▶ Obraditi čvor
  - ▶ Za svaki odredišni čvor do kog postoji poteg od čvora
    - ▶ Ako je odredište nije posećeno
      - ☐ Postaviti prethodnika za odredište na čvor
      - ☐ Ako je odredište jednako ciljnom čvoru postavi vrednost na pronađen i izaći iz petlje
      - ☐ Dodati odredište u posećene
      - ☐ Dodati odredište sa heuristikom u red sa prioritetom 
- ▶ Ako je ciljni čvor pronađen:
  - ▶ Dodati ciljni čvor u put
  - ▶ Postaviti trenutni čvor na prethodnika ciljnog čvora
  - ▶ Sve dok prethodnik postoji
    - ▶ Dodati ciljni čvor u put
    - ▶ Postaviti trenutni čvor na prethodnika trenutnog čvora



# Funkcija `best_first_search` (I)

---

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

```
def best_first_search(graph, start, end):  
    if start is end:  
        path = list()  
        path.append(start)  
        return path  
  
    priority_queue = queue.PriorityQueue(len(graph))  
    visited = set()  
    prev_nodes = dict()  
    prev_nodes[start] = None  
    visited.add(start)  
    priority_queue.put((graph[start][0], start))  
    found_dest = False
```





# Funkcija best\_first\_search (II)

---

- ▶ Glavna petlja funkcije `best_first_search` koja obrađuje čvorove korišćenjem reda

```
while (not found_dest) and (not priority_queue.empty()):  
    node = priority_queue.get()  
    process(node[1])  
    for dest in graph[node[1]][1]:  
        if dest not in visited:  
            prev_nodes[dest] = node[1]  
            if dest is end:  
                found_dest = True  
                break  
    visited.add(dest)  
    priority_queue.put((graph[dest][0], dest))
```



# Funkcija best\_first\_search (III)

---

- Kraj funkcije `best_first_search` formira put od početnog do ciljnog čvora

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



# Traženje

## prvo-najbolji – Primer grafa i poziva

---

- ▶ Primer grafa

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

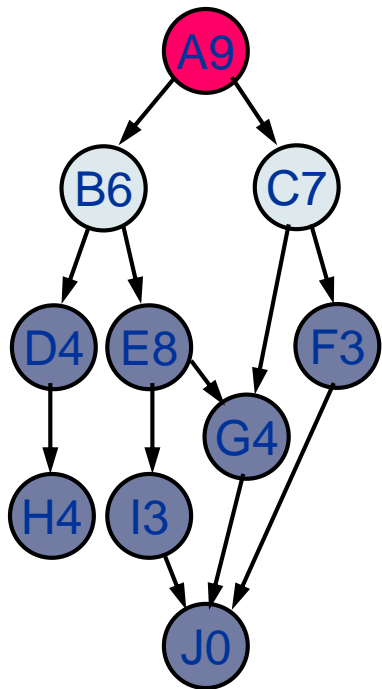
- ▶ Primer poziva funkcije za pretragu prvo-najbolji

```
path = best_first_search(graph, start, end)
```

---

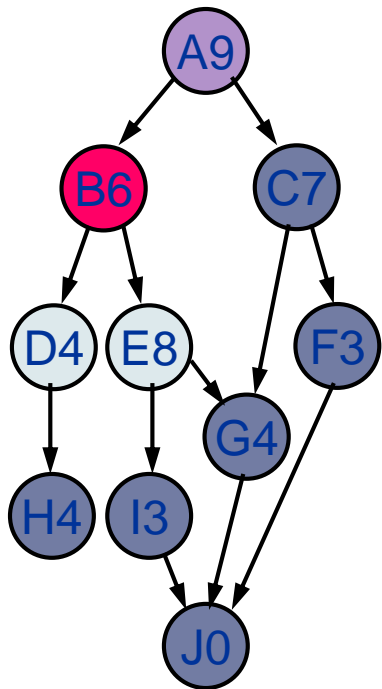


# Algoritam prvo najbolji (ilustracija)



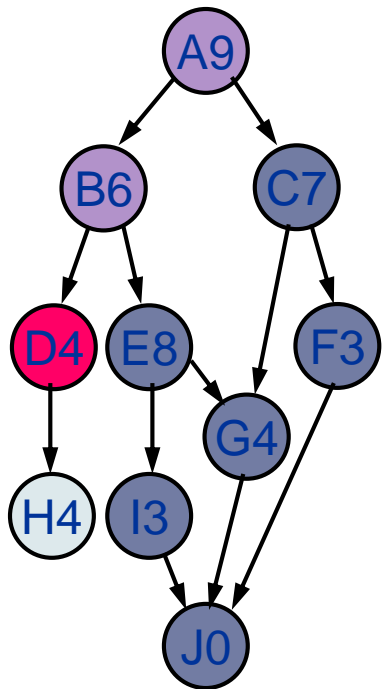
d.r.	za obradu	potomci	obrađeni	prethodnik
0	A9	-	-	(A -)
1	B6 C7	B6 C7	A9	(B A) (C A)

# Algoritam prvo najbolji (ilustracija)



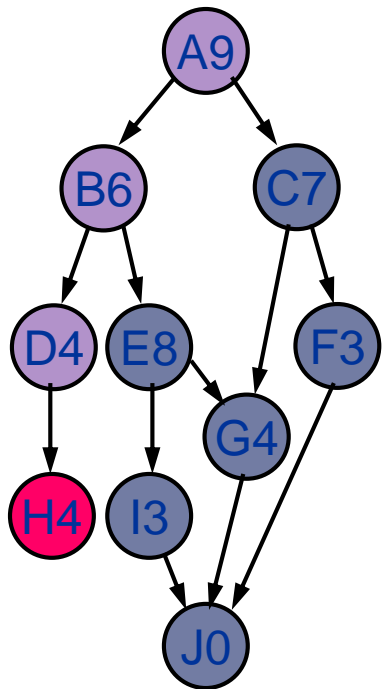
d.r.	za obradu	potomci	obrađeni	prethodnik
0	A9	-	-	(A -)
1	B6 C7	B6 C7	A9	(B A) (C A)
2	D4 C7 E8	D4 E8	A9 B6	(D B) (E B)

# Algoritam prvo najbolji (ilustracija)



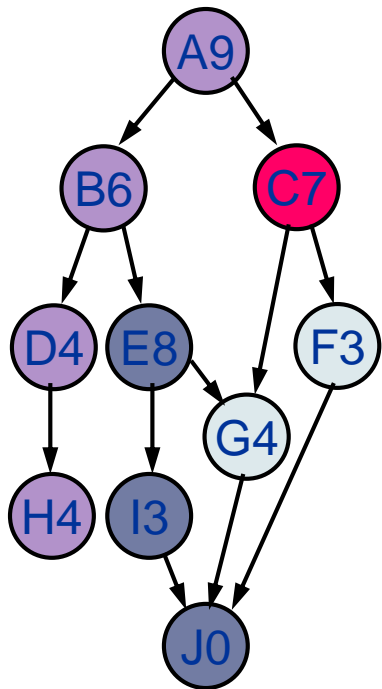
d.r.	za obradu	potomci	obrađeni	prethodnik
0	A9	-	-	(A -)
1	B6 C7	B6 C7	A9	(B A) (C A)
2	D4 C7 E8	D4 E8	A9 B6	(D B) (E B)
3	H4 C7 E8	H4	A9 B6 D4	(H D)

# Algoritam prvo najbolji (ilustracija)



d.r.	za obradu	potomci	obrađeni	prethodnik
0	A9	-	-	(A -)
1	B6 C7	B6 C7	A9	(B A) (C A)
2	D4 C7 E8	D4 E8	A9 B6	(D B) (E B)
3	H4 C7 E8	H4	A9 B6 D4	(H D)
4	C7 E8	-	A9 B6 D4 H4	-

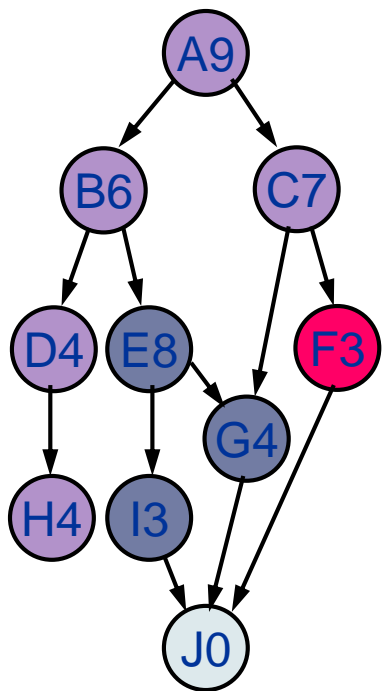
# Algoritam prvo najbolji (ilustracija)



d.r.	za obradu	potomci	obrađeni	prethodnik
0	A9	-	-	(A -)
1	B6 C7	B6 C7	A9	(B A) (C A)
2	D4 C7 E8	D4 E8	A9 B6	(D B) (E B)
3	H4 C7 E8	H4	A9 B6 D4	(H D)
4	C7 E8	-	A9 B6 D4 H4	-
5	F3 G4 E8	F3 G4	A9 B6 D4 H4 C7	(F C) (G C)



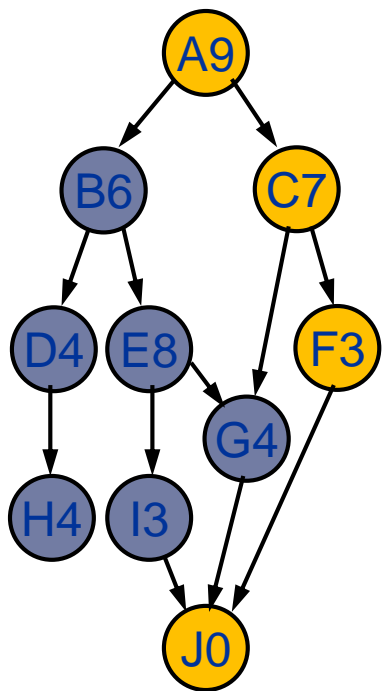
# Algoritam prvo najbolji (ilustracija)



d.r.	za obradu	potomci	obrađeni	prethodnik
0	A9	-	-	(A -)
1	B6 C7	B6 C7	A9	(B A) (C A)
2	D4 C7 E8	D4 E8	A9 B6	(D B) (E B)
3	H4 C7 E8	H4	A9 B6 D4	(H D)
4	C7 E8	-	A9 B6 D4 H4	-
5	F3 G4 E8	F3 G4	A9 B6 D4 H4 C7	(F C) (G C)
6	J0 G4 E8	J0	A9 B6 D4 H4 C7 F3	(J F)

# Algoritam prvo najbolji (ilustracija)

---



## ► Prethodnici

- (A -) (BA) (CA) (D B) (E B) (H D)  
(F C) (G C) (J F)

## ► Put

**(A C F J)**