Katedra za računarsku tehniku i informatiku

Algoritmi i strukture podataka

Milo V. Tomašević

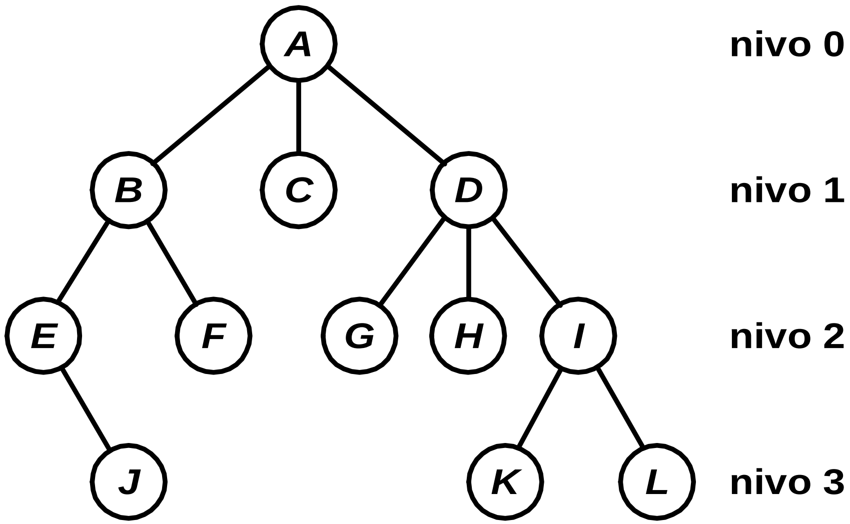
Odsek za softversko inženjerstvo [SI]

Nelinearne strukture podataka

- Jedan element strukture može biti u vezi sa više od dva druga elementa – višedimenzionalnost
- Vrste struktura
  - ✓ stabla
  - ✓ grafovi
- Operacije
- Memorijska reprezentacija
  - ✓ ulančana (češće)
  - ✓ sekvencijalna

Stabla

- Konačan, neprazan skup elemenata - čvorova
  - ✓ postoji poseban čvor - **koren**
  - ✓ ostali čvorovi se mogu razdvojiti u disjunktne podskupove koji su stabla - **podstabla**



Koreno stablo

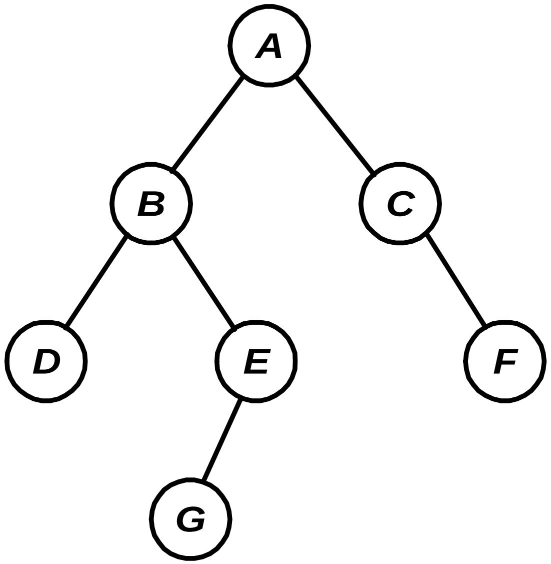
Terminologija

- grane
  - stepen (ulazni, izlazni)
- čvorovi
  - neterminalni i terminalni (listovi)
  - otac, sinovi, braća
  - predak, potomak
- put, dužina puta
- nivo, visina (dubina)
- šuma

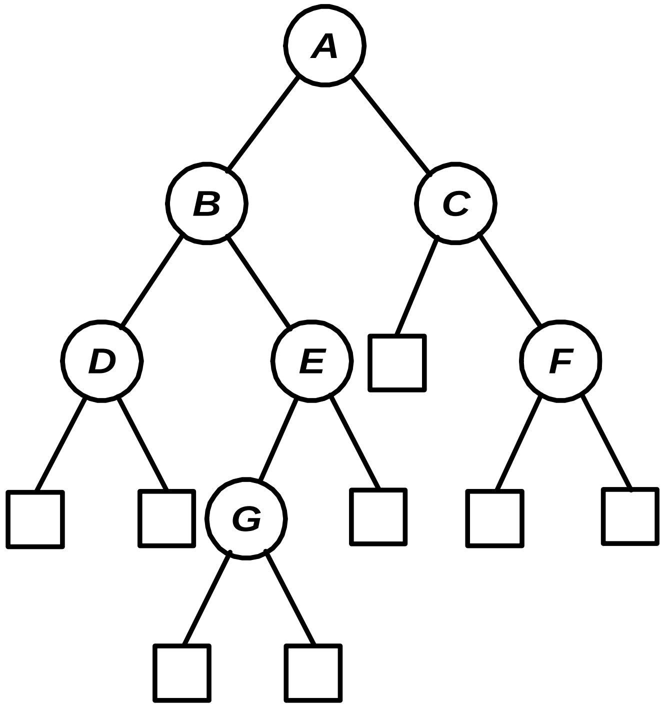
Definicije

- stabla
  - ✓ slična
  - ✓ ekvivalentna
  - ✓ uređena
  - ✓ poziciona
- interna dužina puta - *PI*
- eksterna dužina puta - *PE*
- interni čvorovi $n_{max}=(m^{h+1} - 1)/(m - 1)$
- eksterni čvorovi $e = n(m - 1) + 1$

Interni i eksterni čvorovi



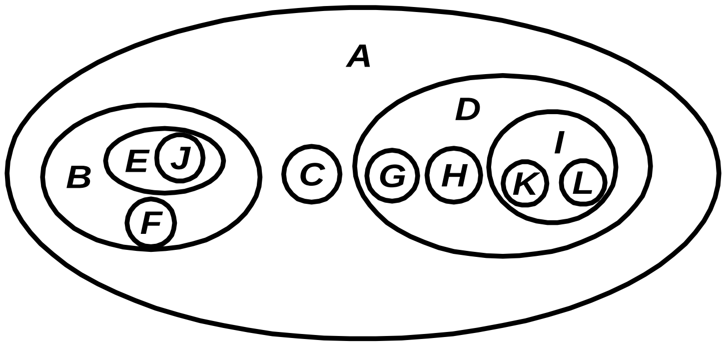
a)



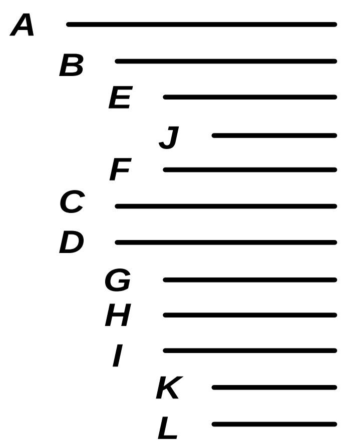
b)

Predstavljanje stabla

a)



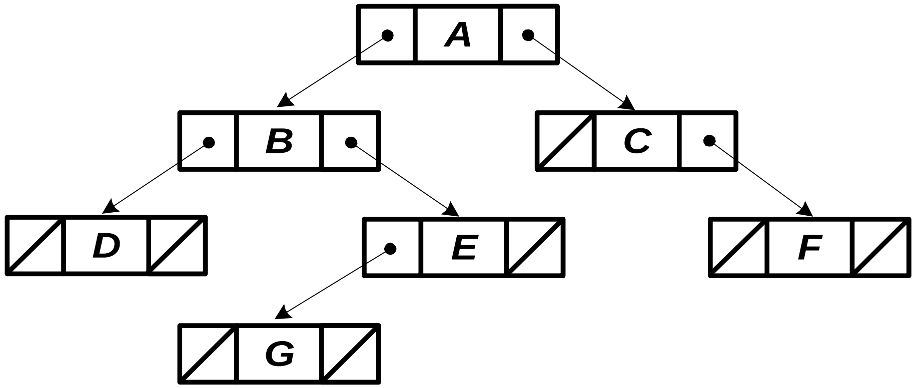
b)



Alternativne grafičke predstave:

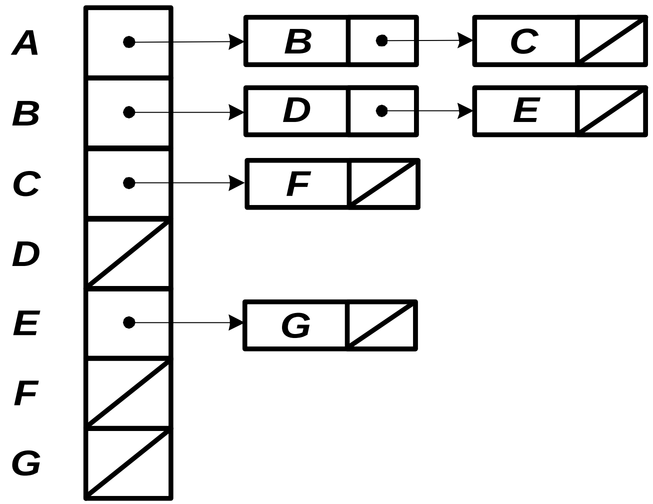
- ugnežđeni skupovi
- identacija

Ulančana reprezentacija



- fleksibilna za proizvoljne topologije
- neefikasno korišćenje prostora
- ponekad i pokazivač na oca

Ulančana reprezentacija



- jedna ulančana lista za sinove svakog čvora
- efikasnije korišćenje prostora

Sekvencijalna reprezentacija

1	*A*	2	3
2	*B*	4	5
3	*C*	0	6
4	*D*	0	0
5	*E*	7	0
6	*F*	0	0
7	*G*	0	0

	A	*B*	*C*	*D*	*E*	*F*	*G*
V	0	1	1	2	2	3	5

Za čvor *k* u vektoru:

otac - $\lfloor (k + m - 2)/m \rfloor = \lceil (k - 1)/m \rceil$

sinovi - $m(k - 1) + 2,\ m(k - 1) + 3, \ldots, mk + 1$

Binarna stabla

- Konačan skup čvorova koji je ili prazan
  ili se sastoji od korena sa dva posebna podstabla
  (levim i desnim) koja su, takođe, binarna stabla

- Različito od uređenog stabla sa m=2!

- $PE = PI + 2n$

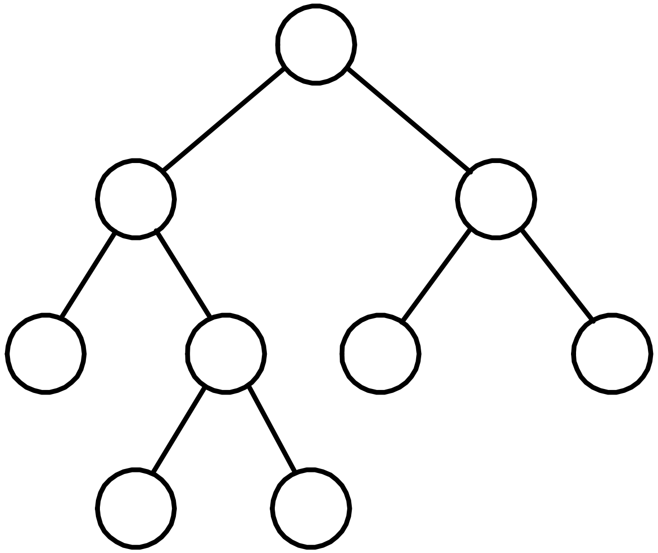
  1: $PI = 0$, $PE = 2$

  n: $PE(n) = PI(n) + 2n$

  $n+1$:
$$
\begin{aligned}
PE(n+1) &= PE(n) - k + 2(k+1)\\
&= PE(n) + k + 2\\
&= PI(n) + 2n + k + 2\\
&= PI(n+1) + 2(n+1)
\end{aligned}
$$

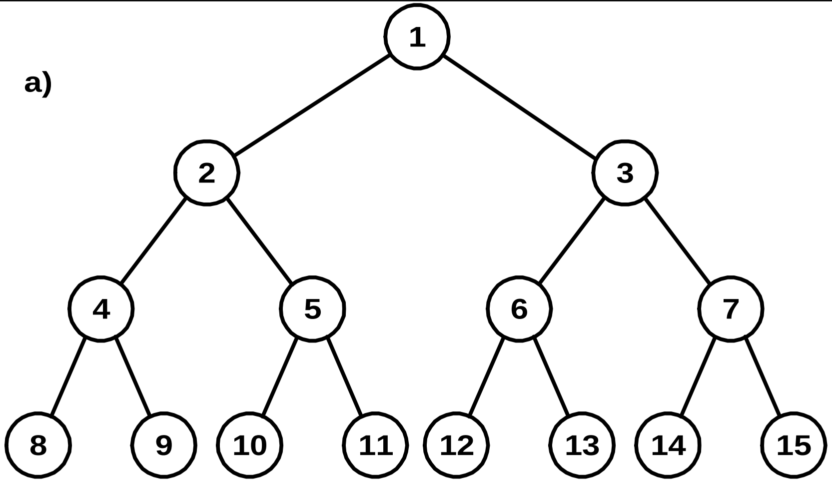
Vrste binarnih stabala

- Puno stablo
  - ✓ svi čvorovi grananja imaju stepen 2
  - ✓ $n0 = n2 + 1$
  - ✓ $n = 2n2 + 1$



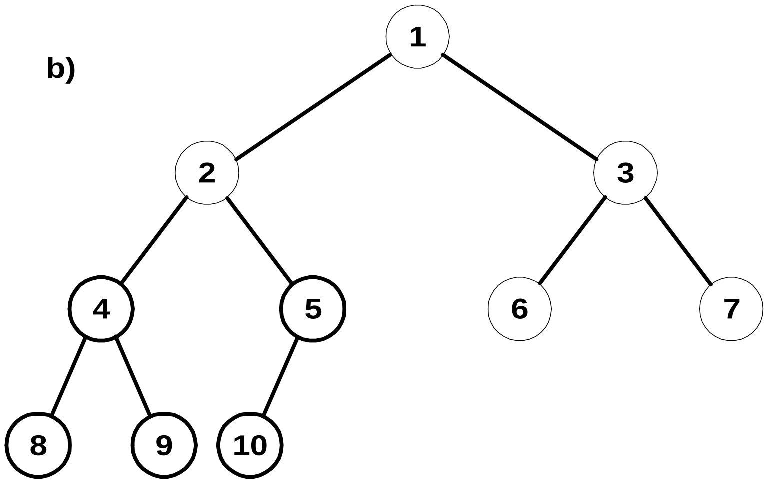
Vrste binarnih stabala

a)



kompletno stablo

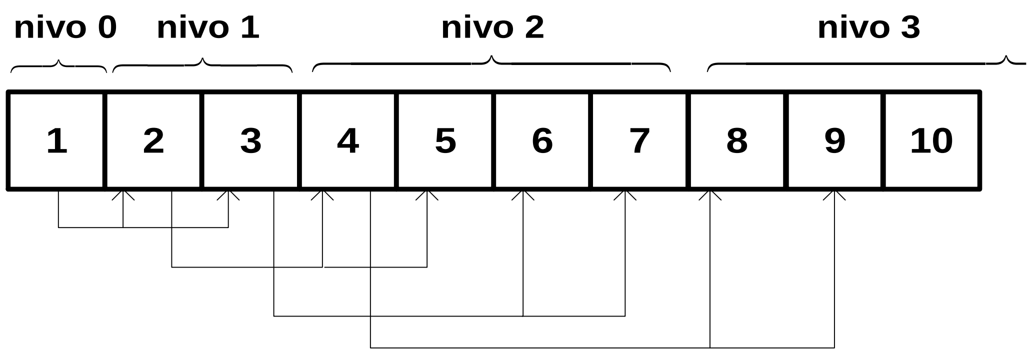
b)



skoro kompletno stablo

Vrste binarnih stabala

- broj čvorova $n_{max} = 2^{h+1} - 1$
- visina $h_{min} = \lceil \log (n + 1) \rceil - 1$
- vektorska reprezentacija, za čvor i
  - ✓ otac $\lfloor i/2 \rfloor$
  - ✓ levi sin $2i$, $2i \le n$
  - ✓ desni sin $2i+1$, $2i + 1 \le n$



Minimizacija interne dužine puta

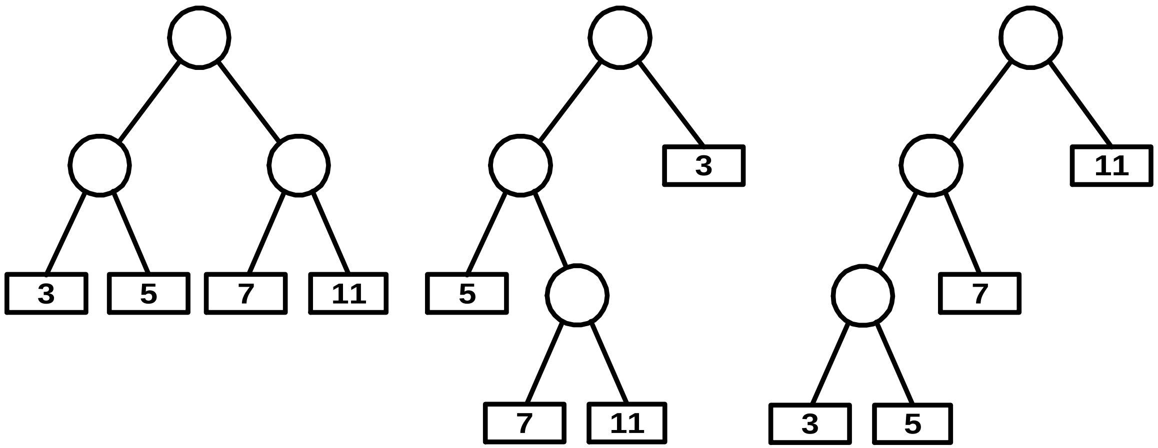
- Najgori slučaj
  - degenerisano stablo - jedan čvor po nivou
  - $PI = n(n - 1)/2 \sim O(n^2)$
- Prosečan slučaj $O(n \sqrt{n})$
- Najbolji slučaj
  - čvorovi što bliže koranu
  - $PI_{min} = 0+1+1+2+2+2+2+\ldots$
    $= \sum \lfloor \log n \rfloor$
    $\sim O(n \log n)$
  - svi listovi na dva susedna nivoa

Težinska eksterna dužina puta

- Eksternim čvorovima pridružene “težine” *w*

- Težinska eksterna dužina puta

$$PWE = \sum w_i l_i$$



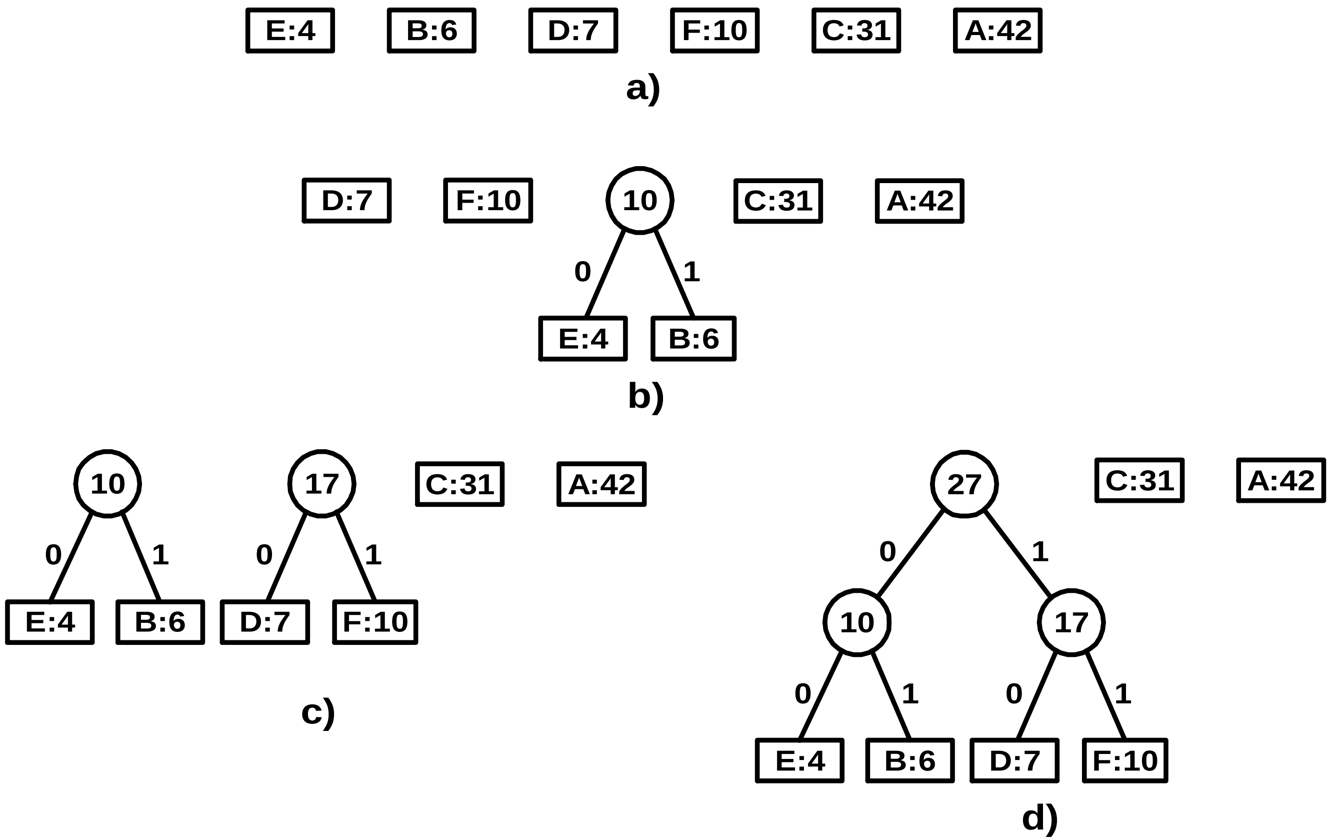
Huffman-ov algoritam

```
HUFFMAN(W, e)
for i = 1 to e do
    z = GETNODE
    w(z) = W[i]
    PQ-INSERT(H, z)
end_for
for i = 1 to e - 1 do
    z = GETNODE
    x = PQ-MIN-DELETE(H)
    y = PQ-MIN-DELETE(H)
    w(z) = w(x)+w(y)
    left(z) = x
    right(z) = y
    PQ-INSERT(H, z)
end_for
return z
```

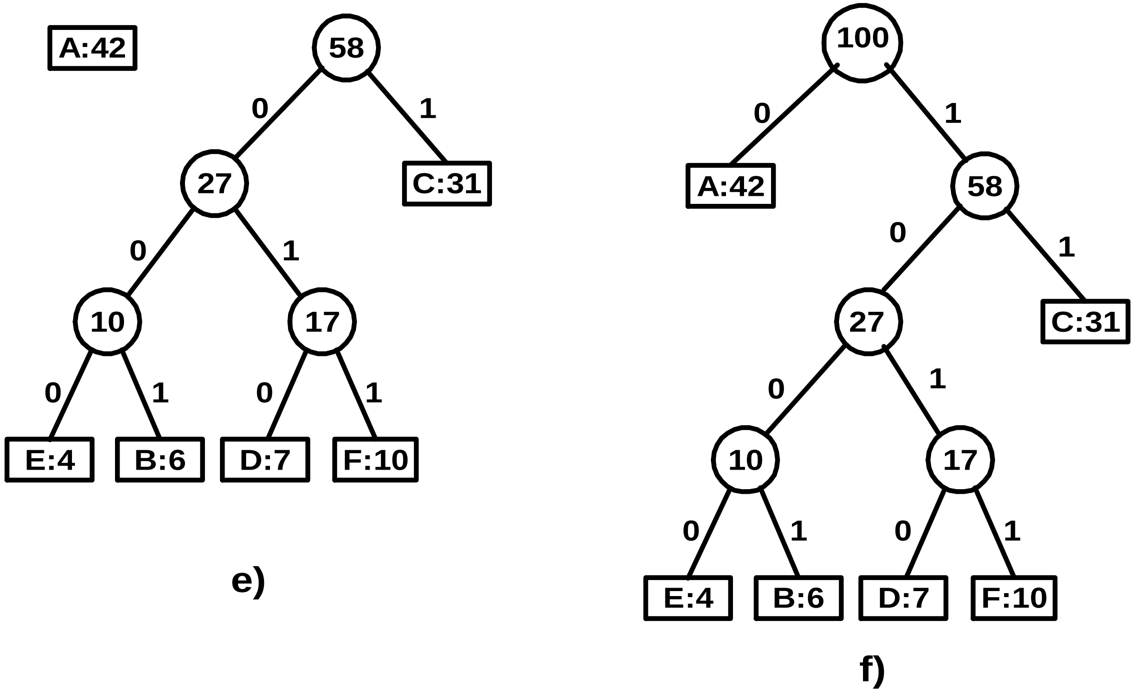
~ $O(e \log e)$

Huffman-ov algoritam

Primena – Huffman-ovi kodovi



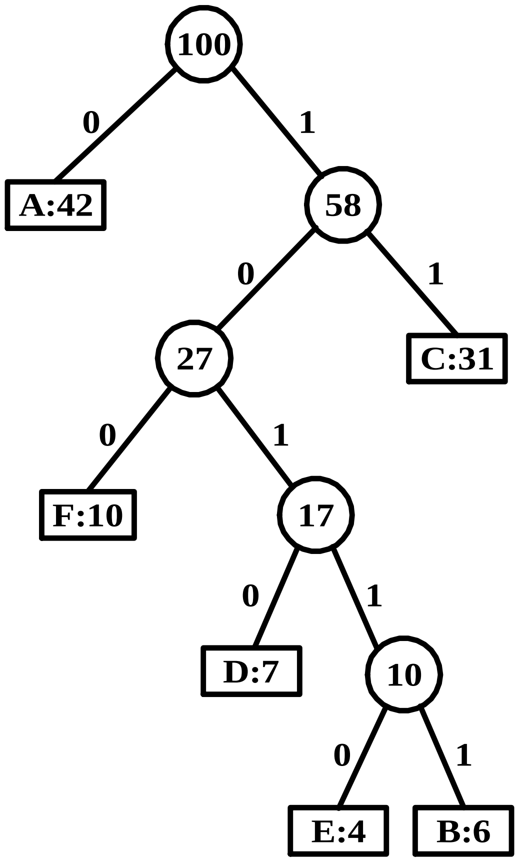
Huffman-ov algoritam



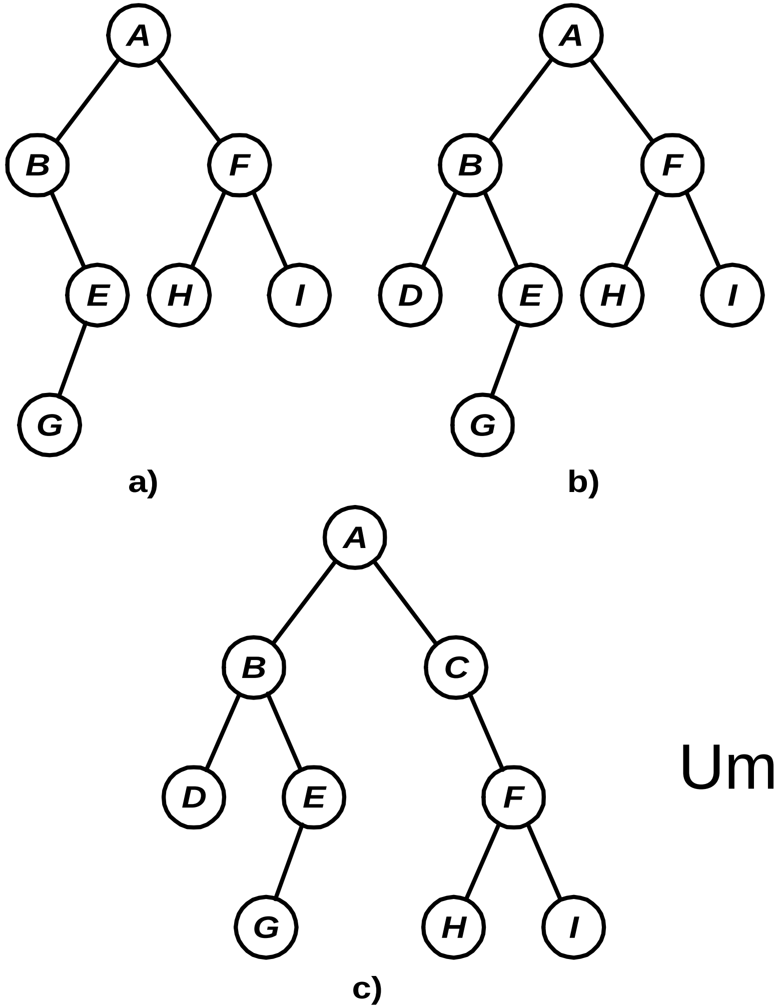
Simboli	A	B	C	D	E	F
Verovatnoće	42	6	31	7	4	10
Kodovi	0	1001	11	1010	1000	1011

Huffman-ov algoritam

Alternativno rešenje sa stablom veće visine



Operacije sa binarnim stablom



Umetanje čvora D

Umetanje čvora C

Obilazak binarnog stabla

Rekurzivne realizacije obilazaka

<u>PREORDER(*root*)</u>
if (*root* ≠ nil) **then**
    P(*root*)
    PREORDER(*left*(*root*))
    PREORDER(*right*(*root*))
end_if

<u>POSTORDER(*root*)</u>
if (*root* ≠ nil) **then**
    POSTORDER(*left*(*root*))
    POSTORDER(*right*(*root*))
    P(*root*)
end_if

<u>INORDER(*root*)</u>
if (*root* ≠ nil) **then**
    INORDER(*left*(*root*))
    P(*root*)
    INORDER(*right*(*root*))
end_if

Preorder

```
PREORDER-I(root)
PUSH(S, root)
while (not STACK-EMPTY(S)) do
    next = POP(S)
    while (next ≠ nil) do
        P(next)
        if (right(next) ≠ nil) then
            PUSH(S, right(next))
        end_if
        next = left(next)
    end_while
end_while
```

□ $O(n)$

S	*next*	posećeni čvor	preorder poredak
A			
	A	A	A
C	B	B	AB
CE	D	D	ABD
CE	nil		
C	E	E	ABDE
C	G	G	ABDEG
C	nil		
	C	C	ABDEGC
F	nil		
	F	F	ABDEGCF
I	H	H	ABDEGCFH
I	nil		
	I	I	ABDEGCFHI
	nil		

Postorder

```
POSTORDER-I(root)
next = root
while (next ≠ nil) do
    PUSH(S, next)
    next =left(next)
end_while
while (not STACK-EMPTY(S)) do
    next = POP(S)
    if (next > 0) then
        PUSH(S, -next)
        next = right(next)
        while (next ≠ nil) do
            PUSH(S, next)
            next = left(next)
        end_while
    else
        next = - next
        P(next)
    end_if
end_while
```

~ O(n)

Obilazak po nivoima

```
LEVEL-ORDER(root)
next = root
INSERT(Q, next)
while (not QUEUE-EMPTY(Q)) do
    next = DELETE(Q)
    P(next)
    if (left(next) ≠ nil) then
        INSERT(Q, left(next))
    end_if
    if (right(next) ≠ nil) then
        INSERT(Q, right(next))
    end_if
end_while
```

~ $O(n)$

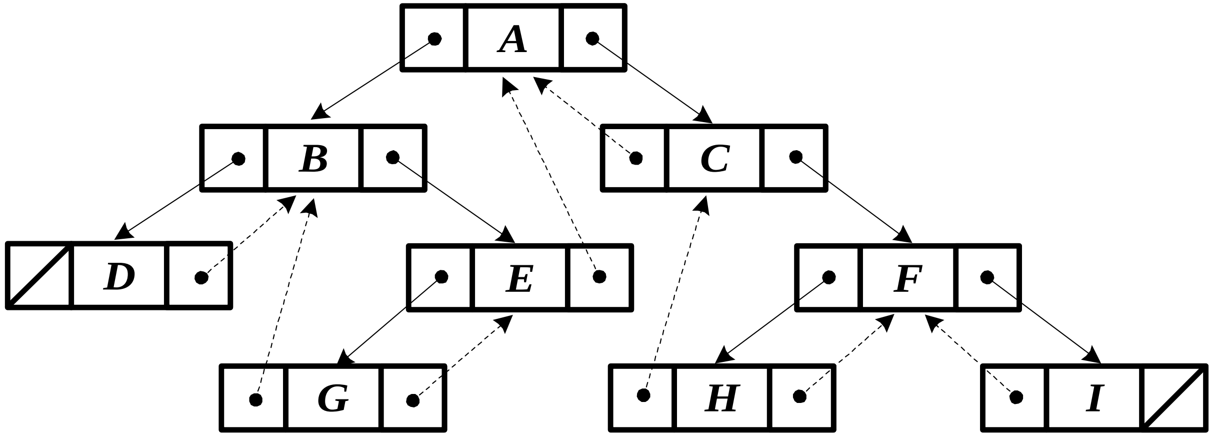
Algoritam slične strukture i za preorder!

Povezana binarna stabla

- Problemi
  - efikasniji obilazak
  - određivanje prethodnika i sledbenika za proizvoljno zadati čvor
  - neiskorišćeni pokazivači (*n*+1 – više od 50%)

- Povezana (*threaded*) binarna stabla
  - strukturni pokazivači
  - veze po izabranom načinu obilaska

Povezana binarna stabla



lf = 1 pokazivač na levo podstablo

lf = 0 veza na prethodnika

rf = 1 pokazivač na desno podstablo

rf = 0 veza na sledbenika

Povezana binarna stabla

Nalaženje sledbenika zadatog čvora po inorderu

```
INSUCC(x)
s = right(x)
if (rf(x) = 1) then
        while (lf(s) = 1) do
                s = left(s)
        end_while
end_if
return s
```

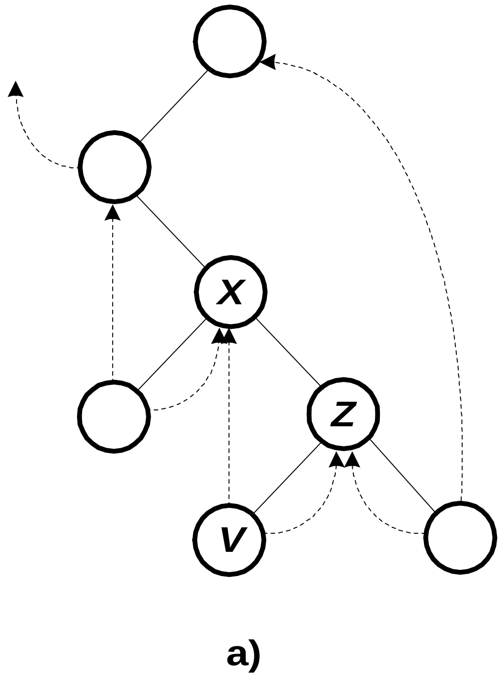
```
INORDER-THR(root)
next = root
while (lf(next) = 1) do
        next = left(next)
end_while
repeat
        P(next)
        next = INSUCC(next)
until next = nil
```

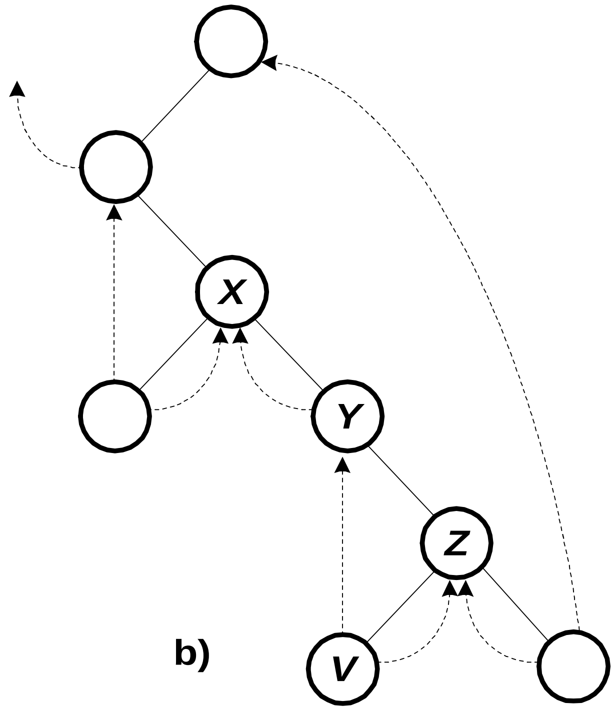
Obilazak povezanog stabla po inorderu

Povezana binarna stabla

Umetanje u povezano stablo

<u>INSERT-RT(*x*, *y*)</u>
right(*y*) = *right*(*x*)
rf(*y*) = *rf*(*x*)
left(*y*) = *x*
lf(*y*) = 0
right(*x*) = *y*
rf(*x*) = 1
if (*rf*(*y*) = 1) **then**
    s = INSUCC(*y*)
    left(*s*) = *y*
end_if



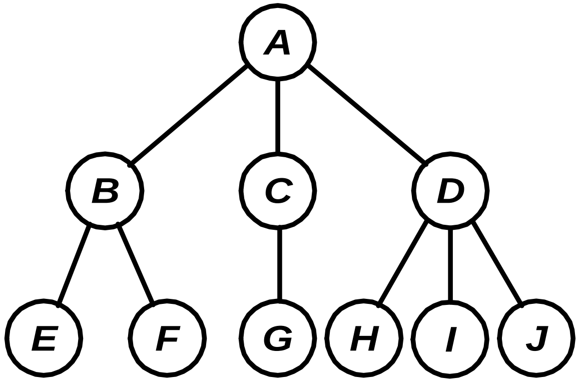


Stabla višeg stepena

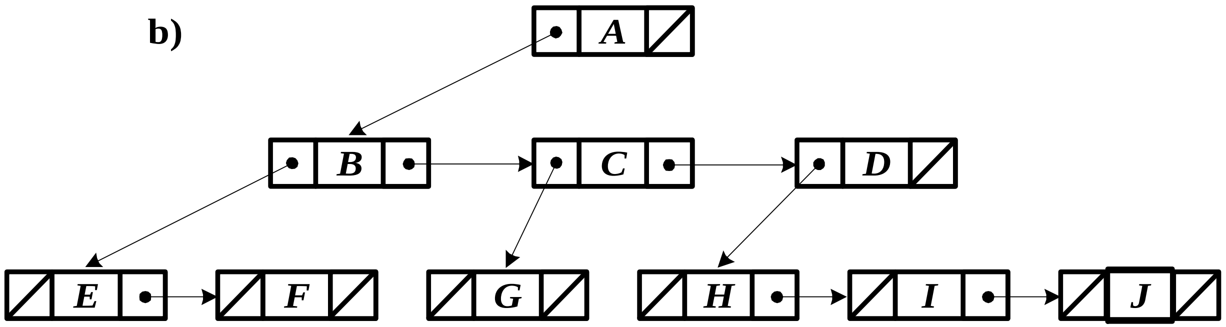
- Problem u stablima stepena $m > 2$
  - neefikasno korišćenje prostora u ulančanoj reprezentaciji
  - neiskorišćeni pokazivači $n(m-1)+1$ iskorišćeni pokazivači $n-1$
- Rešenje
  - odgovarajuće binarno stablo iste semantike
  - binarna relacija “najlevlji sin – desni brat”
  - svi sinovi istog oca u ulančanoj listi

Stabla višeg stepena

a)



b)



Stabla višeg stepena

```
CON-M2BIN(P)
m = INPUT(P)
root = b = GETNODE
info(b) = info(m)
PUSH(S, (level(m), b))
```

Konverzija
m-arnog stabla u
odgovarajuće
binarno stablo

```
while (m = INPUT(P)) do
    b = GETNODE
    info(b) = info(m)
    pred_level = level(TOP(S))
    pred_addr = addr(TOP(S))
    if (level(m) > pred_level) then
        left(pred_addr) = b
    else
        while (pred_level > level(m)) do
            POP(S)
            pred_level = level(TOP(S))
            pred_addr = addr(TOP(S))
        end_while
        right(pred_addr) = b
        POP(S)
    end_if
    PUSH(S, (level(m), b))
end_while
return root
```

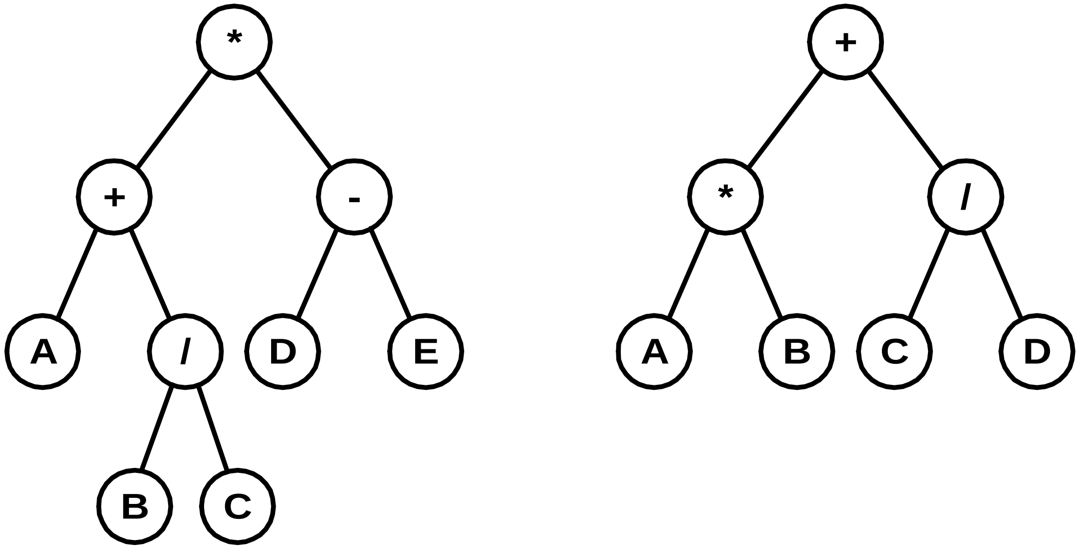
Stabla višeg stepena

Tekući čvor	S	Pokazivač
0,*A*	0,*A*	
1,*B*	0,*A* 1,*B*	*left* (*A*) = *B*
2,*E*	0,*A* 1,*B* 2,*E*	*left* (*B*) = *E*
2,*F*	0,*A* 1,*B* 2,*F*	*right* (*E*) = *F*
1,*C*	0,*A* 1,*C*	*right* (*B*) = *C*
2,*G*	0,*A* 1,*C* 2,*G*	*left* (*C*) = *G*
1,*D*	0,*A* 1,*D*	*right* (*C*) = *D*
2,*H*	0,*A* 1,*D* 2,*H*	*left* (*D*) = *H*
2,*I*	0,*A* 1,*D* 2,*I*	*right* (*H*) = *I*
2,*J*	0,*A* 1,*D* 2,*J*	*right* (*I*) = *J*

Predstavljanje aritmetičkih izraza

- Binarno stablo predstavlja aritmetički izraz
  - ✓ čvorovi grananja – unarni i binarni operatori
  - ✓ listovi - operandi

(A+B/C)(D-E)　　　　A*B+C/D



a)　　　　b)

Predstavljanje aritmetičkih izraza

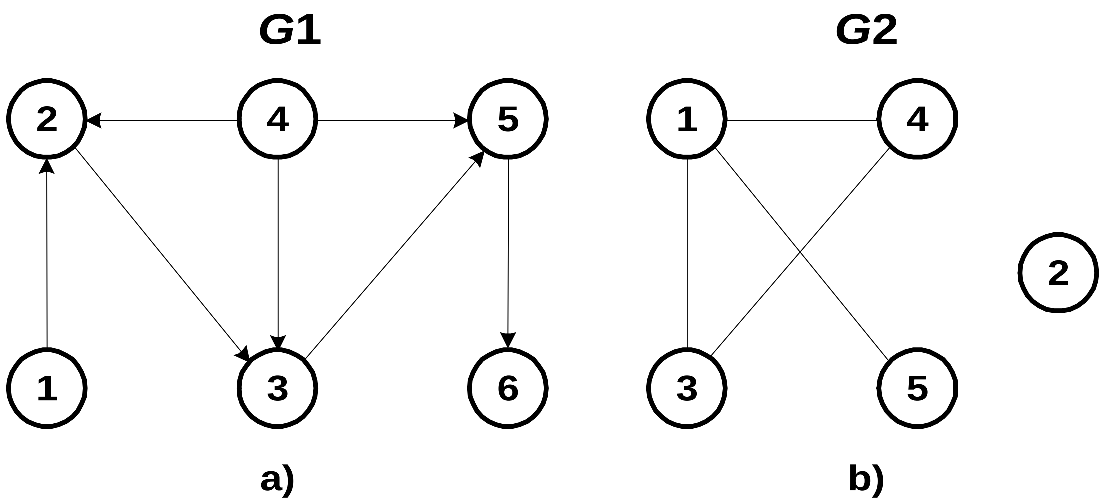
- Obilazak stabla
  - ✓ preorder daje prefiksni izraz
  - ✓ postorder daje postfiksni izraz
  - ✓ inorder daje infiksni izraz (ako nema zagrada!)

Izračunavanje izraza predstavljenog stablom

```
CALC-EXP(r)
case (info(r)) of
    op__add:return(CALC-EXP(left(r)) + CALC-EXP(right(r)))
    op__sub:return(CALC-EXP(left(r)) - CALC-EXP(right(r)))
    . . .
    operand:return(VAL(right(r)))
end_case
```

Grafovi

- Modeliranje proizvoljnih nelinearnih relacija
- Graf *G* je par skupova (*V*, *E*)
  - ✓*V* (**čvorovi**) - konačan neprazan skup
  - ✓*E* (**grane**) - binarne relacije između čvorova
  - ✓grana (*u*, *v*) incidentna na čvorovima



Terminologija

- Usmereni, neusmereni i mešoviti grafovi
- Susednost čvorova
- Ulazni i izlazni stepen čvora
- Petlje i paralelne grane
- Prosti graf, multigraf, hipergraf, podgraf
- Težinski graf
- Put, prost put, dostižnost

Terminologija

- Ciklus, ciklični i aciklični grafovi
- Kompletni, gusti i retki grafovi
- Bipartitni graf
- Povezani graf, povezane komponente
- Slobodno stablo
  - ✓ acikličan, povezan, neusmeren graf

Matrična reprezentacija

- Matrica susednosti A
  - ✓ $a[i, j] = 1$ ako $(i, j) \in E$
  - ✓ $a[i, j] = 0$ ako $(i, j) \notin E$

- $a[i, j] = w(i, j)$ za težinske grafove

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

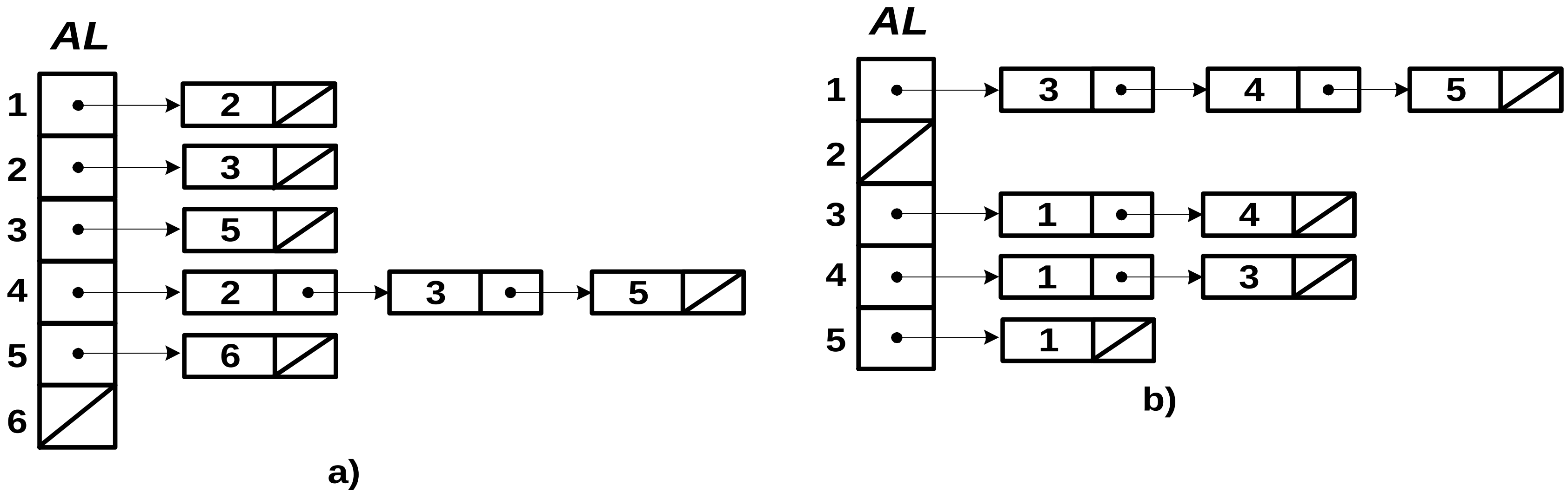
a)

$$A = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

b)

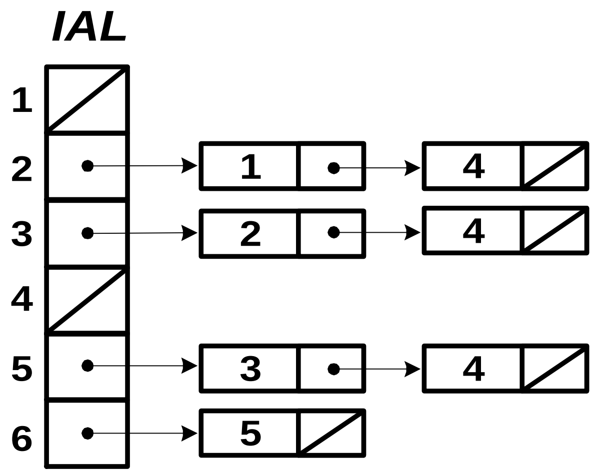
Ulančana reprezentacija

- Liste susednosti
  - ✓ vektor zaglavlja
  - ✓ ulančane liste suseda
  - ✓ element liste odgovara grani



Ulančana reprezentacija

- Inverzne liste susednosti
  - ✓ lista za čvor sadrži sve čvorove kojima je sused
  - ✓ pogodno za izračunavanje ulaznog stepena



- Multiliste
  - ✓ element koji predstavlja granu ulančan u dve liste
  - ✓ identifikacija oba čvora i dva pokazivača

Predstavljanje grafova

- Prostorna složenost

	matrica	liste (netežinski)	liste (težinski)
usmereni	n^2	$n + 2e$	$n + 3e$
neusmereni	$n(n + 1)/2$	$n + 4e$	$n + 6e$

- Matrična reprezentacija pogodnija za
  - ✓ dinamičku promenu broja grana
  - ✓ za direktan pristup grani
- Ulančana reprezentacija pogodnija za
  - ✓ dinamičku promenu broja čvorova
  - ✓ za određivanje suseda

Obilazak grafa

- Svi čvorovi se posete samo jednom u nekom linearnom poretku
- Poredak zavisi od izbora početnog čvora
- Ako se ne posete svi zbog nedostižnosti, nastavlja se sa nekim od neposećenih
- Na čvor se može naići više puta, ali samo prvi put se poseti
- Osnovni algoritmi zasnovani na susednosti:
  - ✓ obilazak po širini (BFS)
  - ✓ obilazak po dubini (DFS)

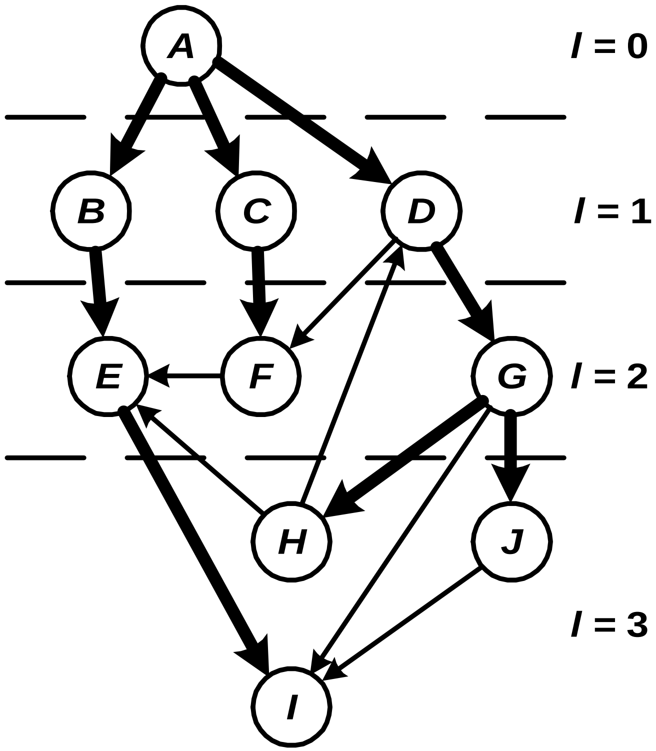
Obilazak grafa po širini

- Strategija algoritma
  - ✓ poseti početni čvor
  - ✓ poseti njegove susede
  - ✓ poseti njihove neposećene susede istim redom, ...
- Posete u “talasima”, po nivoima iste udaljenosti
- Koristi se neprioritetni red za čekanje i vektor posećenosti
- Vremenska složenost
  - ✓ $O(n^2)$ za matričnu reprezentaciju
  - ✓ $O(\max(n, e))$ za ulančanu reprezentaciju

Obilazak grafa po širini

```
BFS(G, v)
for i = 1 to n do
    visit [i] = false
end_for
visit [v] = true
INSERT(Q, v)
while (not QUEUE-EMPTY(Q)) do
    v = DELETE(Q)
    for { u : (v, u) ∈ E} do
        if (not visit [u]) then
            visit [u] = true
            INSERT(Q, u)
        end_if
    end_for
end_while
```

Obilazak grafa po širini



A	*A*
B C D	*ABCD*
C D E	*ABCDE*
D E F	*ABCDEF*
E F G	*ABCDEFG*
F G I	*ABCDEFGI*
G I	*ABCDEFGI*
I H J	*ABCDEFGIHJ*

Obilazak grafa po dubini

- Strategija algoritma
  - poseti početni čvor
  - poseti jednog njegovog suseda
  - poseti jednog neposećenog suseda prethodnog
  - ako ga nema, vraća se do poslednjeg prethodnika koji ima neposećenog suseda
- Slično preorderu kod stabla
- Vremenska složenost
  - $O(n^2)$ za matričnu reprezentaciju
  - $O(\max(n, e))$ za ulančanu reprezentaciju

Obilazak grafa po dubini

- Rekurzivna realizacija

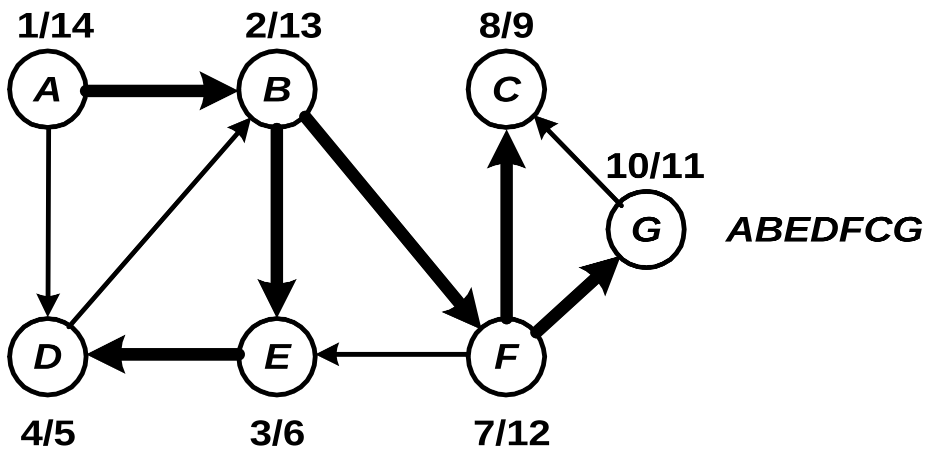
```
DFS-VISIT(v)
visit [v] = true
for { u, (v, u) ∈ E } do
    if (not visit [u]) then
        DFS-VISIT(u)
    end_if
end_for
```

```
DFS(G, v)
for i = 1 to n do
    visit [i] = false
end_for
DFS-VISIT(v)
```

- Može i iterativna realizacija korišćenjem steka

Obilazak grafa po dubini

- Primer



Primene obilaska grafa

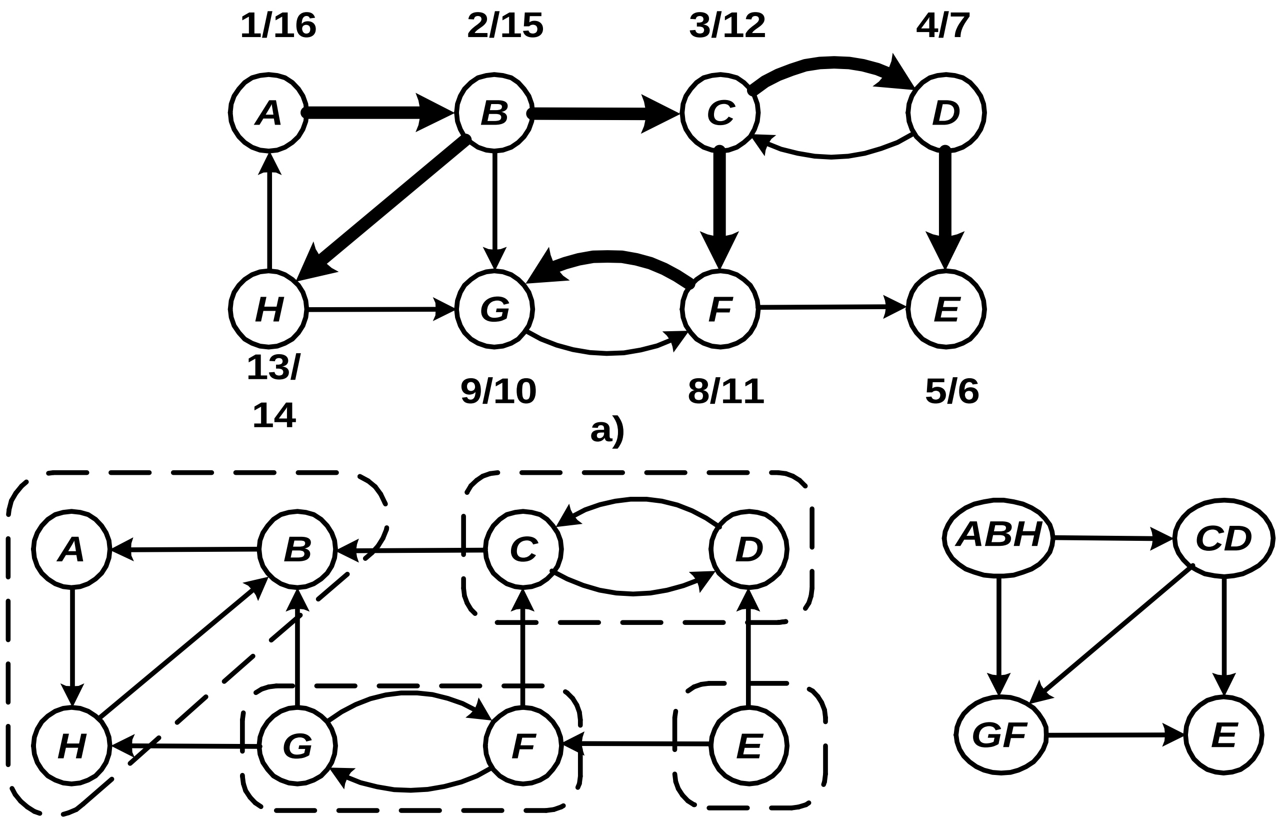
- Određivanje najkraćeg rastojanja između dva čvora (i i j) u netežinskom grafu
  - ✓ rastojanje do j jednako broju nivoa $l[j]$
  - ✓ BFS se startuje od polaznog čvora i ($l[i] = 0$)
  - ✓ pri poseti nekog čvora u preko drugog čvora v postavi se njegov nivo na $l[u] = l[v] + 1$
  - ✓ kad se poseti čvor j rastojanje je nađeno kao minimalni broj grana od čvora i
- Provera cikličnosti
  - ✓ postojanje poprečnih ili povratnih grana

Primene obilaska grafa

- Određivanje povezanih komponenata u neusmerenom grafu

```
CONN-COMP(G)
n_cc = 0
for i = 1 to n do
    visit[i] = false
end_for
for i = 1 to n do
    if (not (visit[i]) then
        n_cc = n_cc + 1
        DFS-VISIT(i) → CC(n_cc)
    end_if
end_for
```

Primene obilazaka grafa



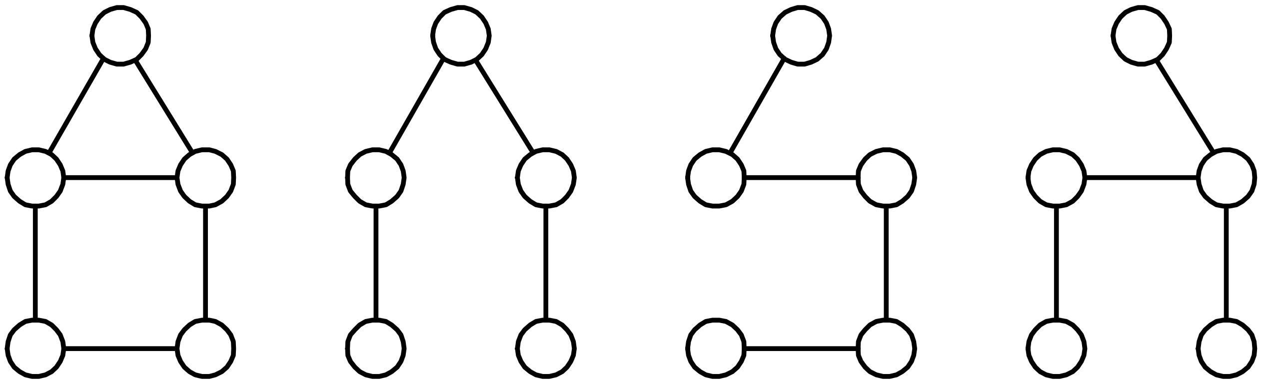
a)

b)

c)

Obuhvatna stabla

- Obuhvatno stablo $ST = (U, E')$ neusmerenog, povezanog grafa $G = (V, E)$
  - ✓ sadrži sve čvorove grafa, $U = V$
  - ✓ sadrži određen broj grana, $E' \subseteq E$, tako da su svi čvorovi povezani, ali da nema ciklusa

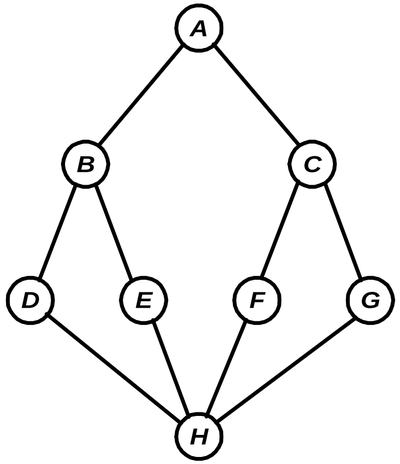


a) b) c) d)

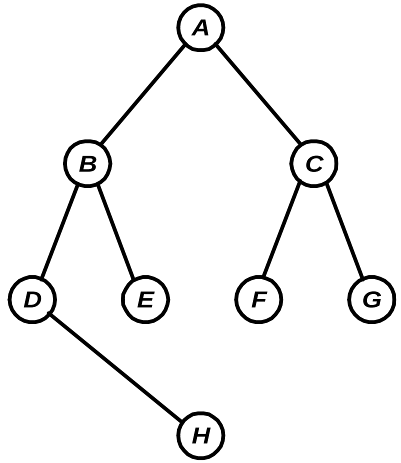
Obuhvatna stabla

- Obuhvatno stablo – slobodno stablo
- Za nepovezan graf – obuhvatna šuma $ST_i = (V_i, E_i)$
- Generiše se pomoću algoritama obilaska BFS ili DFS
  - ✓ na početku *E'* prazan skup
  - ✓ kada se dođe do neposećenog čvora *u*, dolazna grana (*v*, *u*) se uključi u stablo ($E' = E' + \{(v, u)\}$ u **then** delu algoritma
- Broj grana u obuhvatnom stablu - *n* - 1
- Primene

Obuhvatna stabla

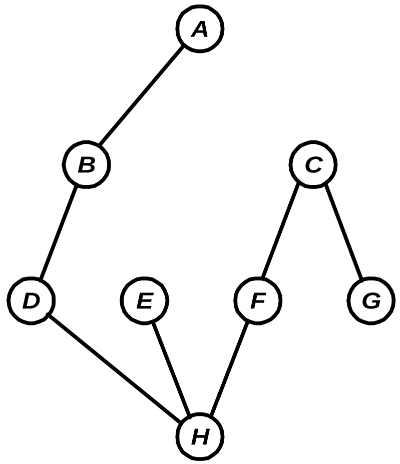


a)



b)

BFS stablo

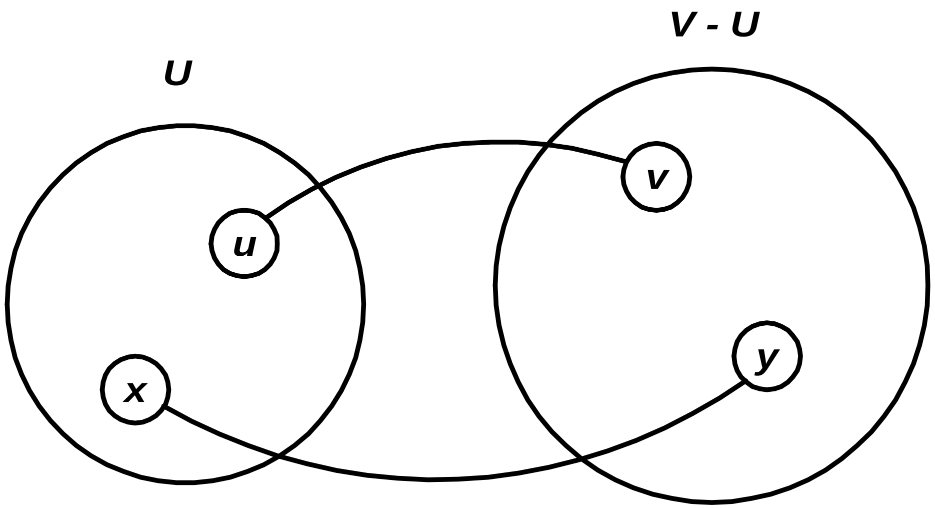


c)

DFS stablo

Minimalna obuhvatna stabla

- Cena obuhvatnog stabla – $\sum w(u, v) \quad (u, v) \in E'$
- MST – obuhvatno stablo čija je cena **minimalna**
- Ako je $U \subseteq V$ i ako je *(u, v)* grana najmanje težine takva da je $u \in U$ i $v \in (V - U)$,
  onda postoji MST koje sadrži *(u, v)*

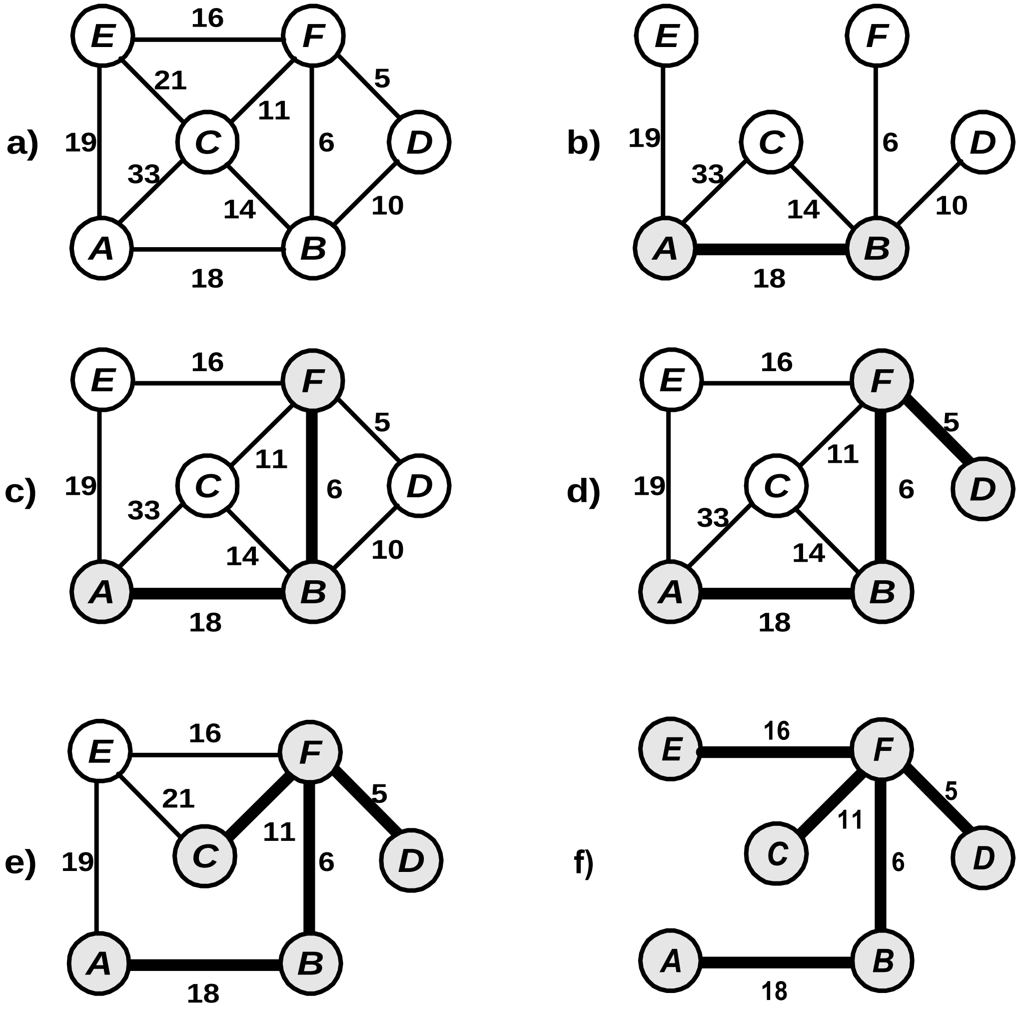


Prim-ov algoritam

- Inkrementalno gradi MST počevši od polaznog čvora dodajući po jednu granu i jedan čvor
- Bira granu najmanje težine od onih
  koje povezuju čvorove koji su već uključeni u MST
  i čvorove koji još nisu uključeni

<u>PRIM(G, s)</u>
$U = \{s\}$
$E' = \emptyset$
while ($U \neq V$) **do**
    find (u, v) $\Rightarrow$ min $\{w(u, v) : (u \in U)$ and $(v \in (V - U))\}$
    $U = U + \{v\}$
    $E' = E' + \{(u, v)\}$
end_while
$MST = (U, E')$

Prim-ov algoritam



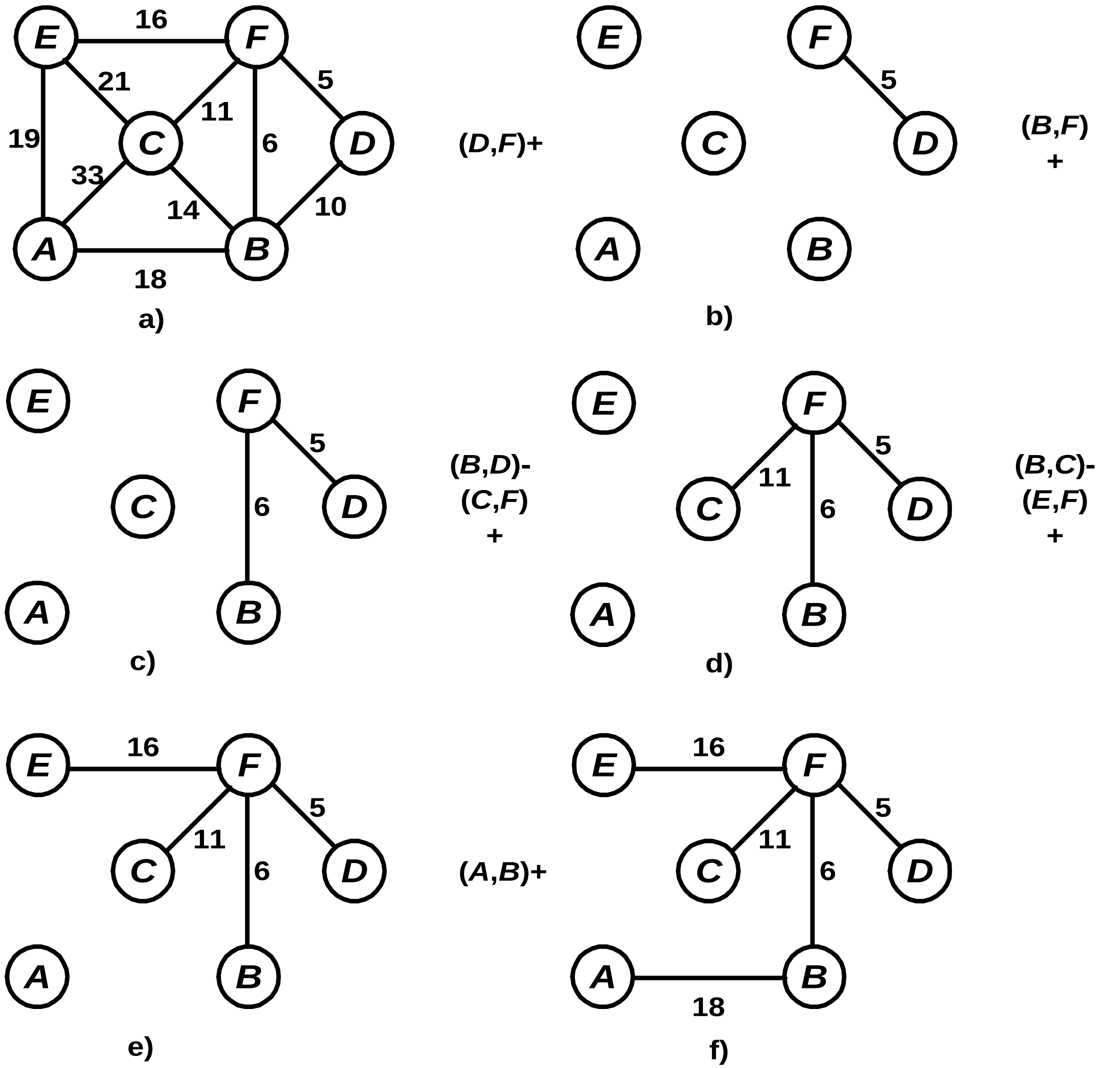
Kruskal-ov algoritam

<u>KRUSKAL(G)</u>
$E' = \emptyset$
for each $(u, v) \in E$ **do**
    PQ-INSERT(PQ, $w(u, v)$)
end_for
$num = 0$
while ($num < n - 1$) **do**
    $w(u,v)$ = PQ-MIN-DELETE(PQ)
    if (($u \in T_i$) and ($v \in T_j$) and ($i \neq j$)) **then**
        $E' = E' + \{(u, v)\}$
        $T_k = T_i + T_j$
        $num = num + 1$
    end_if
end_while
$MST = (V, E')$

Kruskal-ov algoritam

- Polazi od šume nepovezanih čvorova - podstabala
- Inkrementalno dodaje po jednu granu
- Bira granu koja:
  - ✓ ima najmanju težinu od preostalih, neuključenih
  - ✓ ne zatvara ciklus
- Završava kada se uključi n - 1 grana
- Vremenska složenost - $O(e \log e)$

Kruskal-ov algoritam



Određivanje dostižnosti

- Matrica puta (dostižnosti) P
  - ✓ $p[i, j] = 1$ ako postoji put od i do j
  - ✓ $p[i, j] = 0$ ako ne postoji put od i do j
- $a[i, k]a[k, j] = 1$ ako postoji put od i do j preko k
- $a_{ij}^{(2)} = \sum a[i, k]a[k, j]$ - broj puteva dužine 2 od i do j
- $A^{(2)} = A^2$
- $a_{ij}^{(3)} = \sum a[i, k]^{(2)}a[k, j]$ - broj puteva dužine 3 od i do j
- $A^{(3)} = A^3$, ... , $A^{(m)}$, ...

Određivanje dostižnosti

Algoritam za određivanje matrice puta

- Naći $B^{(n)} = A + A^{(2)} + ... + A^{(n)} = A + A^2 + ... + A^n$
- Članovi matrice puta P se određuju kao:
  - ✓ $p[i, j] = 1$ ako je $b[i, j]^{(n)} > 0$
  - ✓ $p[i, j] = 0$ ako je $b[i, j]^{(n)} = 0$
- Optimizacije:
  - ✓ u prostoru - matrica bitova
  - ✓ u vremenu – **or** i **and** umesto ***** i **+**
- Vremenska složenost - $O(n^4)$

Warshall-ov algoritam

```
WARSHALL(A)
P = A
for k = 1 to n do
    for  i = 1 to n do
        for  j = 1 to n do
            p[i, j] = p[i, j] or (p[i, k] and p[k, j])
        end_for
    end_for
end_for
```

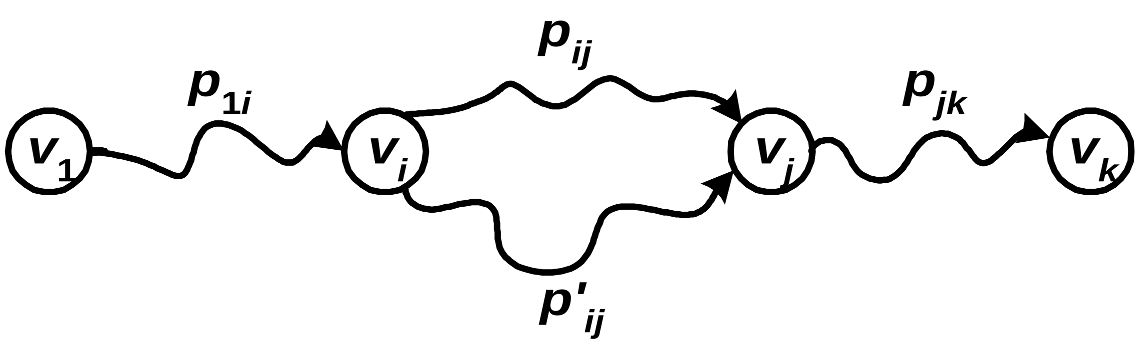
```
if (p[i, k] = 1) then
    for j = 1 to n do
        p[i, j] = p[i, j] or p[k, j]
    end_for
end_if
```

Warshall-ov algoritam

- Složenost:
  - ✓ vremenska - $O(n^3)$
  - ✓ prostorna - $O(n^2)$
- U neusmerenom grafu može lakše:
  - ✓ naći povezane komponente
  - ✓ prostorna - $O(n^2)$
  - ✓ $p[i, j] = p[j, i] = 1$ za sve parove i i j u istoj povezanoj komponenti
  - ✓ $p[i, j] = p[j, i] = 0$ za sve parove i i j u različitim povezanim komponentama
  - ✓ vremenska složenost - $O(n^2)$

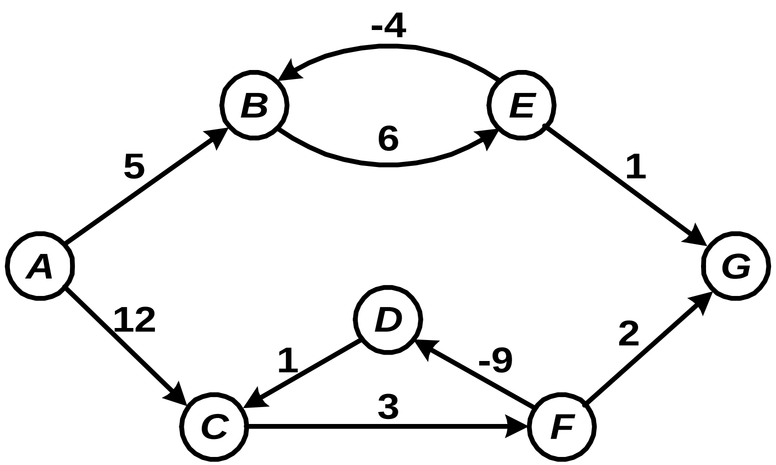
Najkraća rastojanja

- $G = (V, E)$ – usmereni težinski graf
  - ✓ $w(i, j)$ - težina grane od i do j
  - ✓ $w(p_{1k}) = \sum w(i, j)$ – dužina puta od 1 do k
- Najkraće rastojanje između čvorova i i j je:
  - ✓ $d(i, j) = \min\{w(p)\}$ po svim putevima od i do j
  - ✓ ∞ ako j nije dostižan iz i



Najkraća rastojanja

- Grane mogu imati i negativne težine
- Nisu dozvoljeni ciklusi sa negativnom težinom



Floyd-ov algoritam

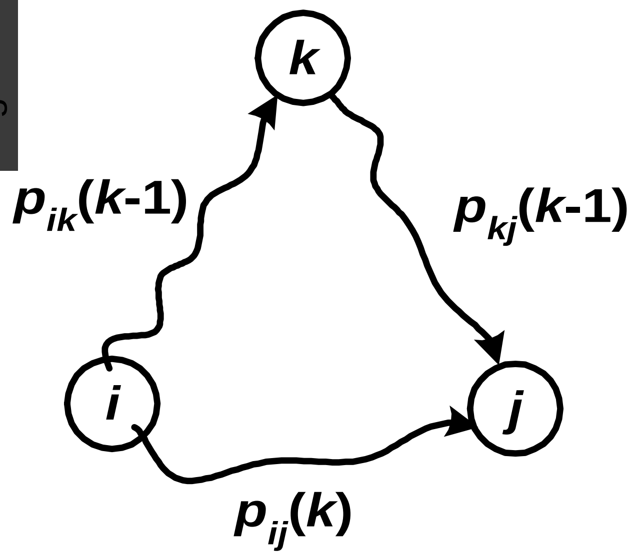
- Najkraća rastojanja i putevi između svih parova čvorova

- Ulaz – matrica težina *W*
  - ✓ $w[i, j] = 0$ ako je $i = j$
  - ✓ $w[i, j] = w(i, j)$ ako je $i \neq j$ i $(i, j) \in E$
  - ✓ $w[i, j] = \infty$ ako je $i \neq j$ i $(i, j) \notin E$

- Izlaz – matrica najkraćih rastojanja *D* i matrica prethodnika *T*

- Inicijalizacija matrice prethodnika *T*
  - ✓ $t[i, j] = 0$ ako je $i = j$ ili $w[i, j] = \infty$
  - ✓ $t[i, j] = i$ ako je $i \neq j$ ili $w[i, j] < \infty$

Floyd-ov algoritam




```
FLOYD(W)
D = W
for k = 1 to n do
    for i = 1 to n do
        for j = 1 to n do
            if (d[i, j] > d[i, k] + d[k, j]) then
                t[i, j] = t[k, j]
                d[i, j] = d[i, k] + d[k, j]
            end_if
        end_for
    end_for
end_for
```

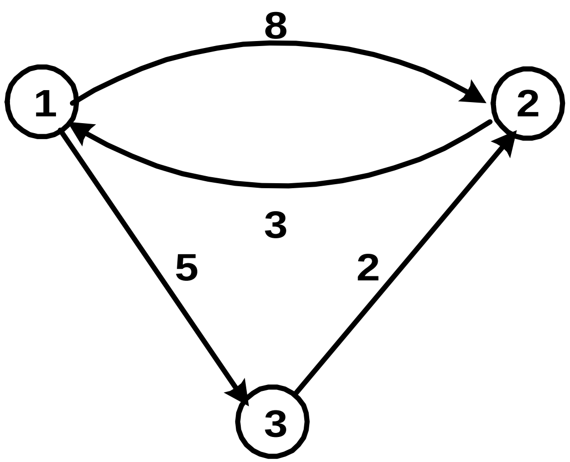
Princip relaksacije ~ $O(n^3)$

Floyd-ov algoritam

- Rekonstrukcija puta

```
PATH(i, j)
if (i = j) then
    PRINT(i)
    return
else
    if (t[i, j] = 0) then
        PRINT(Nema puta između i i j)
    else
        PATH(i, t[i, j])
        PRINT(j)
    end_if
end_if
```

Floyd-ov algoritam



$$D^{(1)} = \begin{bmatrix} 0 & 8 & 5 \\ 3 & 0 & 8 \\ \infty & 2 & 0 \end{bmatrix} \quad T^{(1)} = \begin{bmatrix} 0 & 1 & 1 \\ 2 & 0 & 1 \\ 0 & 3 & 0 \end{bmatrix}$$

$$D^{(2)} = \begin{bmatrix} 0 & 8 & 5 \\ 3 & 0 & 8 \\ 5 & 2 & 0 \end{bmatrix} \quad T^{(2)} = \begin{bmatrix} 0 & 1 & 1 \\ 2 & 0 & 1 \\ 2 & 3 & 0 \end{bmatrix}$$

$$D^{(0)} = \begin{bmatrix} 0 & 8 & 5 \\ 3 & 0 & \infty \\ \infty & 2 & 0 \end{bmatrix} \quad T^{(0)} = \begin{bmatrix} 0 & 1 & 1 \\ 2 & 0 & 0 \\ 0 & 3 & 0 \end{bmatrix} \quad D^{(3)} = \begin{bmatrix} 0 & 7 & 5 \\ 3 & 0 & 8 \\ 5 & 2 & 0 \end{bmatrix} \quad T^{(3)} = \begin{bmatrix} 0 & 3 & 1 \\ 2 & 0 & 1 \\ 2 & 3 & 0 \end{bmatrix}$$

Floyd-ov algoritam

- Primena – određivanje središta grafa
- $ecc(v) = \max \{d[i, v] : i \in V\}$ - ekscentričnost čvora
- $\min \{ecc(v), v \in V\}$ - središte
- Određivanje središta:
  - ✓ naći matricu najkraćih rastojanja *D*
  - ✓ naći maksimume kolona
  - ✓ naći minimume od ovih maksimuma
  - ✓ čvor koji odgovara minimumu - središte

Dijkstra-in algoritam

- Najkraća rastojanja i putevi između jednog čvora (1) i svih ostalih
- Ne dozvoljava negativne težine grana
- Ulaz – matrica težina grana *W*
- Izlaz – vektor najkraćih rastojanja *D* i vektor prethodnika *T*
- Inicijalizacija:
  - ✓ vektora *D* sa prvom vrstom matrice W
  - ✓ vektora *T* ($t[i, j] = 1$ ako je $w[1, i] < \infty$)

Dijkstra-in algoritam

```
DIJKSTRA(W)
S = {1}
for i = 2 to n do
    d [i] = w [1, i]
    if (w [1, i] ≠ ∞) then
        t [i] = 1
    else
        t [i] = 0
    end_if
end_for
```

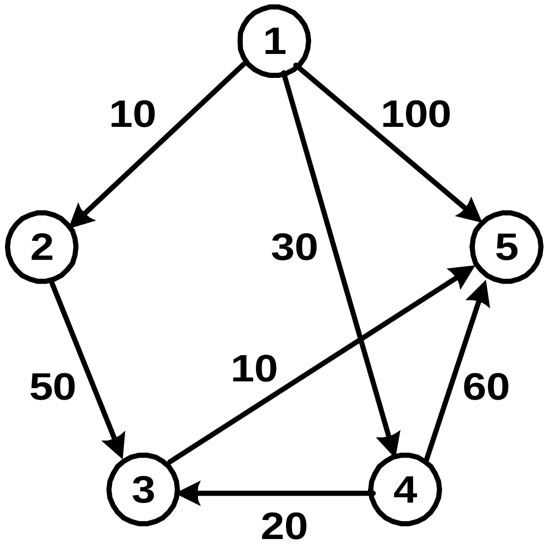
```
for k = 1 to n - 1 do
    find min {d [i] : i ∈ (V – S) }
    S = S + {i}
    for each j ∈ (V – S) do
        if (d [i] + w [i, j] < d [j]) then
            d [j] = d [i] + w [i, j]
            t [j] = i
        end_if
    end_for
end_for
```

Dijkstra-in algoritam

- Rekonstrukcija puta od čvora 1 do *i*

```
PATH(i)
if (i = 1) then
      PRINT(1)
      return
else
      if (t [i ] = 0) then
            PRINT(Nema puta do i )
      else
            PATH(t [i ])
            PRINT(i)
      end_if
end_if
```

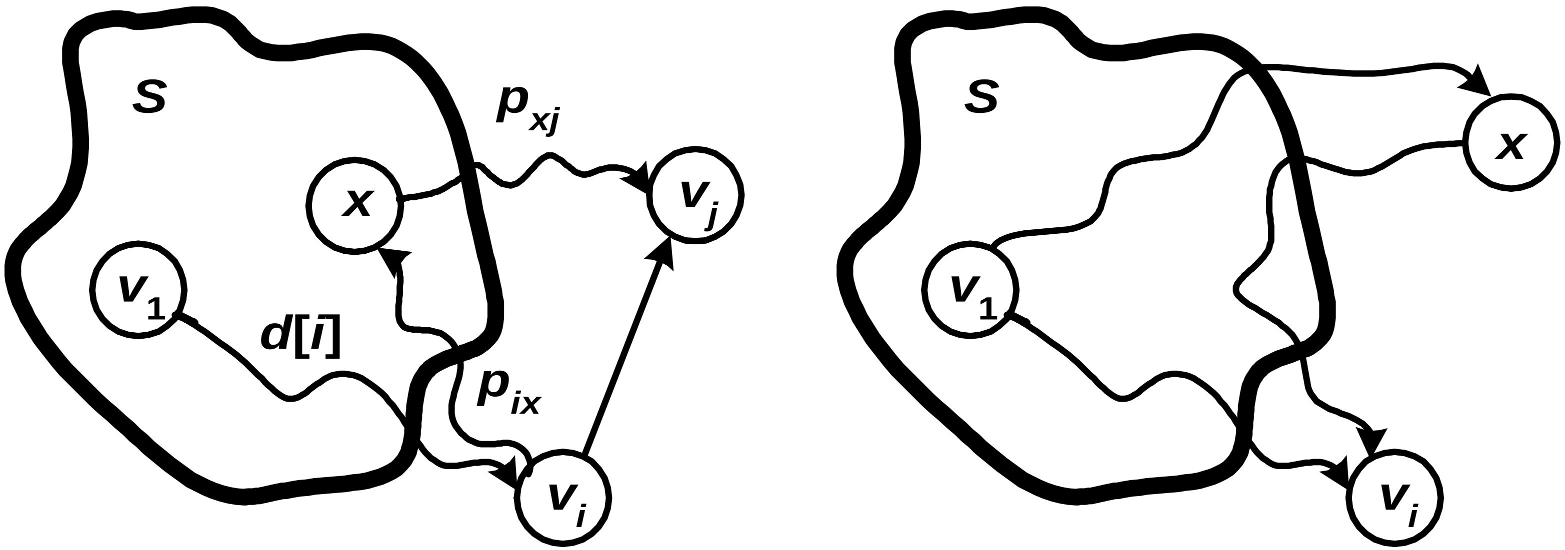
Dijkstra-in algoritam



		d[i]				t[i]			
		2	3	4	5	2	3	4	5
1	-	10	∞	30	100	1	0	1	1
1,2	2	10	60	30	100	1	2	1	1
1,2,4	4	10	50	30	90	1	4	1	4
1,2,4,3	3	10	50	30	60	1	4	1	3
1,2,4,3,5	5	10	50	30	60	1	4	1	3

Dijkstra-in algoritam

- Dokaz korektnosti:
  - ✓ korektnost postupka relaksacije
  - ✓ korektnost određivanja najkraćeg rastojanja



Dijkstra-in algoritam

- Vremenska složenost:
  - za matricu susednosti - $O(n^2)$
  - za liste susednosti - $O(e + n \log n)$
- Algoritam se može iskoristiti i za određivanje:
  - najkraćih rastojanja između svih parova čvorova
  - najkraćeg rastojanja između dva čvora
- Bellman-Ford algoritam – ako su dozvoljene i grane sa negativnom težinom

Protok u grafovima

- Problem optimizacije protoka u distributivnom sistemu
- Protočni graf – težinski, usmereni graf:
  - ✓ protok $f(u, v)$ i kapacitet grane $c(u, v) \geq 0$
  - ✓ izvor S i odredište T neograničenog kapaciteta
- Ograničenje kapaciteta

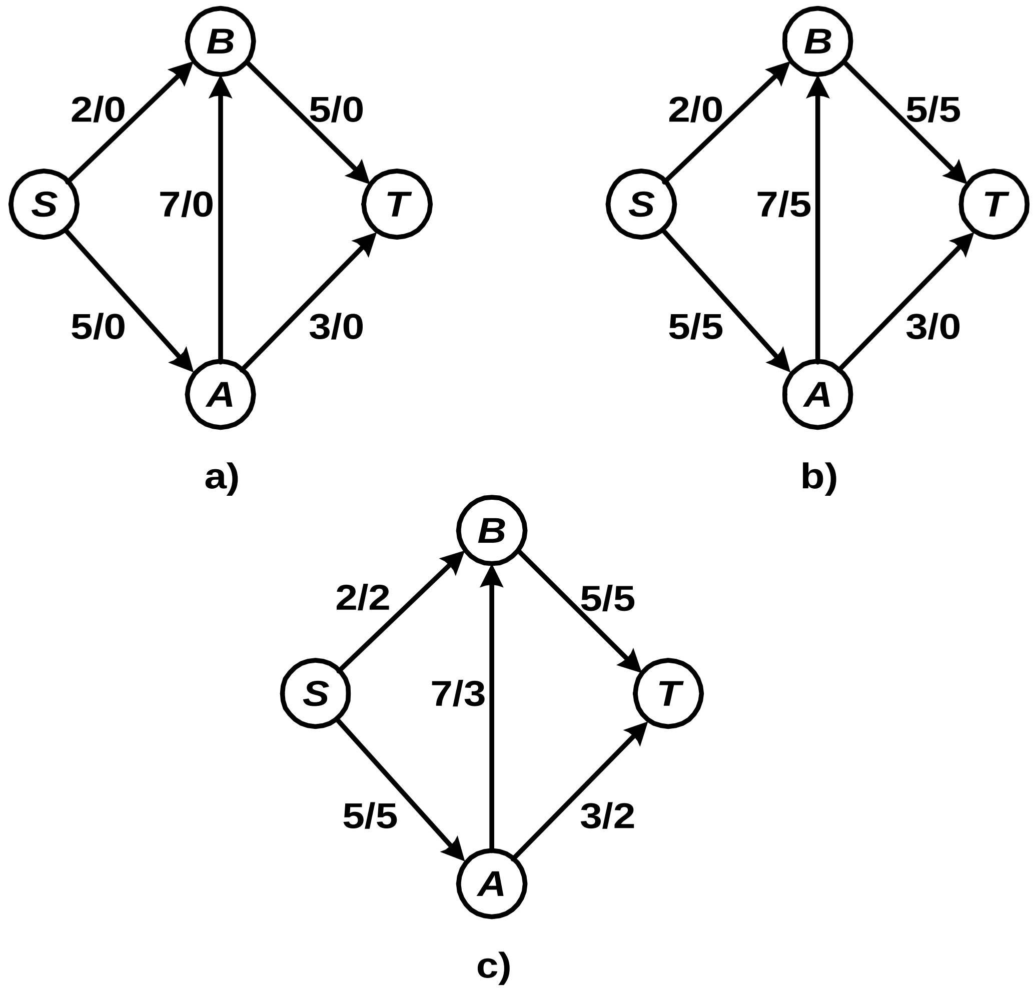
  $f(u, v) \leq c(u, v)$ za sve $(u, v) \in E$
- Simetrija

  $f(u, v) = -f(v, u)$ za sve $(u, v) \in E$
- Održanje protoka:
  - ✓ $\sum f(u, v) = 0$ za sve $u \in V$, osim za S i T
  - ✓ $\sum f(S, v) = \sum f(u, T)$ za S i T

Maksimizacija protoka

- Određivanje najvećeg protoka od *S* do *T* koji protočni graf *G* = (*V*, *E*) može da propusti uz data ograničenja
- Polazi se od nultih početnih protoka grana
- Iterativno se traži put povećanog protoka

Maksimizacija protoka

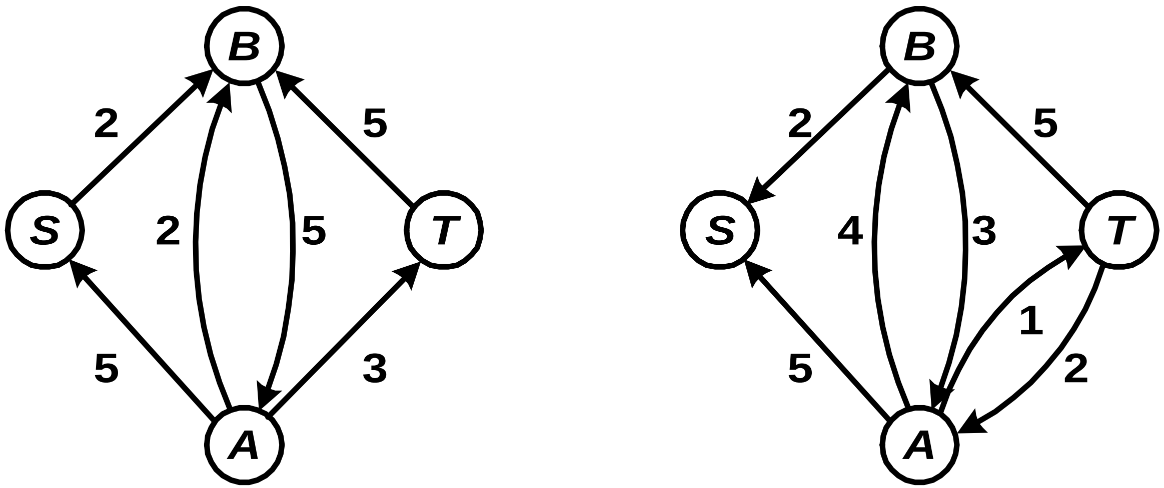


Maksimizacija protoka

- Rezidualni kapacitet grane
  $cf(u, v) = c(u, v) - f(u, v)$
- Rezidualni graf $G_f = (V, E_f)$
  $E_f = \{ (u,v) : u, v \in V , cf(u, v) > 0 \}$
- U početku rezidualni graf isti kao protočni
- Rezidualni graf može da sadrži i
  grane kojih nema u protočnom ($e_f \leq 2e$)
- Traži se put povećanog protoka od S do T u G_f
- Rezidualni kapacitet puta
  $cf(p) = \min \{cf(u, v) : (u,v) \in p \}$

Maksimizacija protoka

- Rezidualni graf

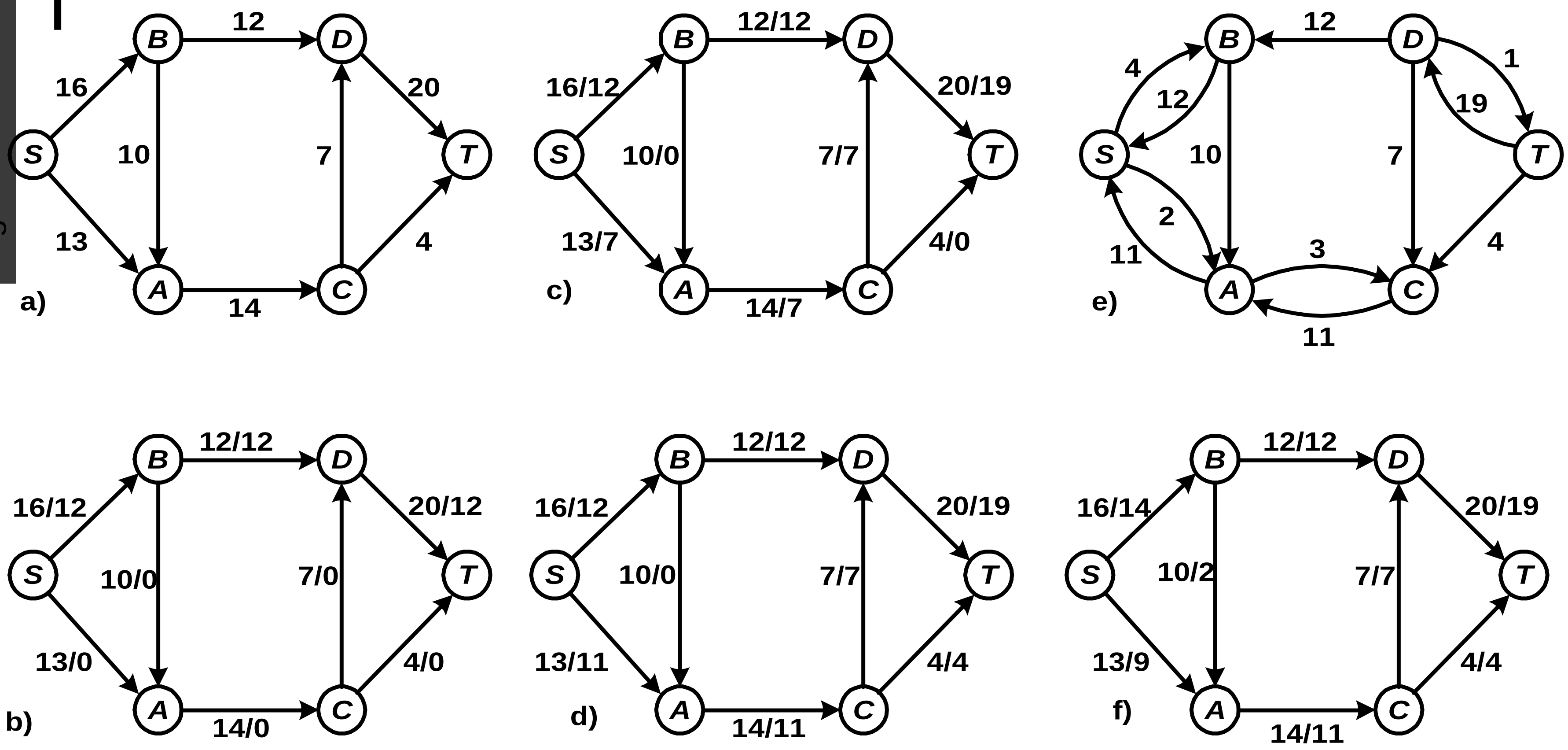


a) b)

Maksimizacija protoka

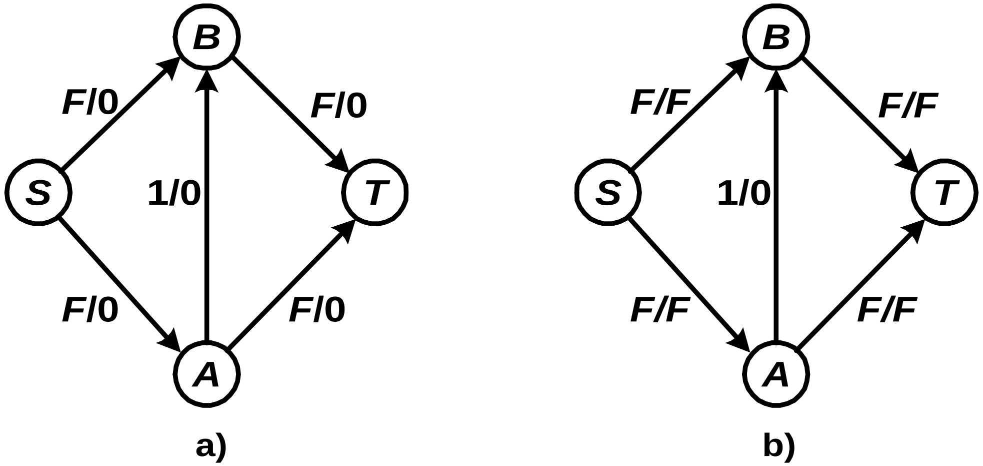
```
FORD-FULKERSON (G)
for each (u,v) ∈ E do
    f(u,v) = 0
    f(v,u) = 0
end_for
while exists p(S, T) in G_f do
    cf(p) = min {cf(u, v) : (u,v) ∈ p}
    for each (u, v) ∈ p do
        f(u, v) = f(u, v) + cf(p)
        f(v, u) = - f(u, v)
    end_for
end_while
```

Maksimizacija protoka



Maksimizacija protoka

- Vremenska složenost zavisi od načina traženja puta povećanog protoka
- Ako se traži preko obilaskom po širini ili dubini, gornja granica $O(ef_{max})$



- Edmonds-Karp-ov algoritam traži put od *S* do *T* sa najmanjim brojem grana po BFS - $O(en^2)$

Maksimizacija protoka

- Generalizacija :
  - više izvora $S_1, ..., S_m$
  - više odredišta $T_1, \ldots, T_n$
- “Superizvor” S i “superodredište” T
- $c(S, S_i) = \infty \; i = 1, \ldots, m$
- $c(T_i, T) = \infty \; i = 1, \ldots, n$
- Primeni se Ford-Fulkerson-ov algoritam

Uparivanje grafa

- Problemi uparivanja dva skupa “jedan na jedan”

- Bipartitni neusmereni graf $G = (V, E)$:
  - ✓ $V = V1 + V2$
  - ✓ $(u, v) \in E \Leftrightarrow (u \in V1 \wedge v \in V2) \vee$
    $(u \in V2 \wedge v \in V1)$

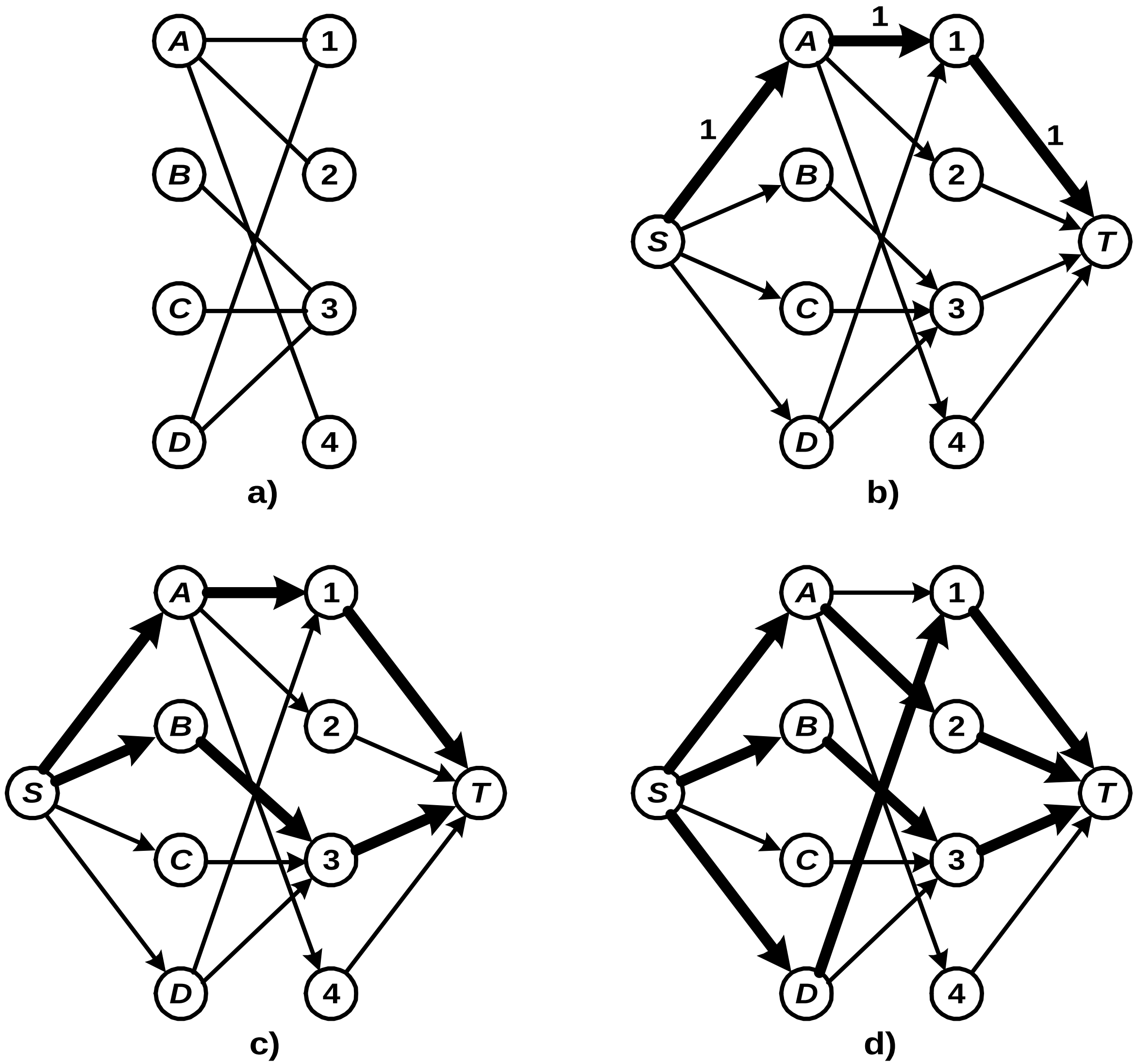
- Upareni skup grana – podskup grana *M*
  tako da je za svaki čvor grafa
  najviše jedna grana iz *M* incidentna na njemu

- Maksimalni upareni skup – upareni skup
  sa najviše grana

Uparivanje grafa

- Određivanje maksimalnog uparenog skupa se svodi na maksimizaciju protoka
- Za bipartitni graf $G = (V, E)$ definiše se protočni graf $G' = (V', E')$
  - ✓ $V' = V + \{S\} + \{T\}$
  - ✓ $E' = \{(S, u) : u \in V1\} + E + \{(v, T) : v \in V2\}$
  - ✓ $c(u,v) = 1$ za sve $(u, v) \in E'$
- Na G' se primeni Ford-Fulkerson-ov algoritam
- Grane $(u, v) \in E$ po kojima teče protok ulaze u maksimalni upareni skup grana
- $f_{max} = \min(n1, n2) \sim O(n\,e)$

Uparivanje grafa



Topološko sortiranje

- Modeliranje složenih projekata usmerenim acikličnim grafovima

- Usmereni aciklični graf $G = (V, E)$:
  - ✓ čvorovi - događaji
  - ✓ grane - aktivnosti

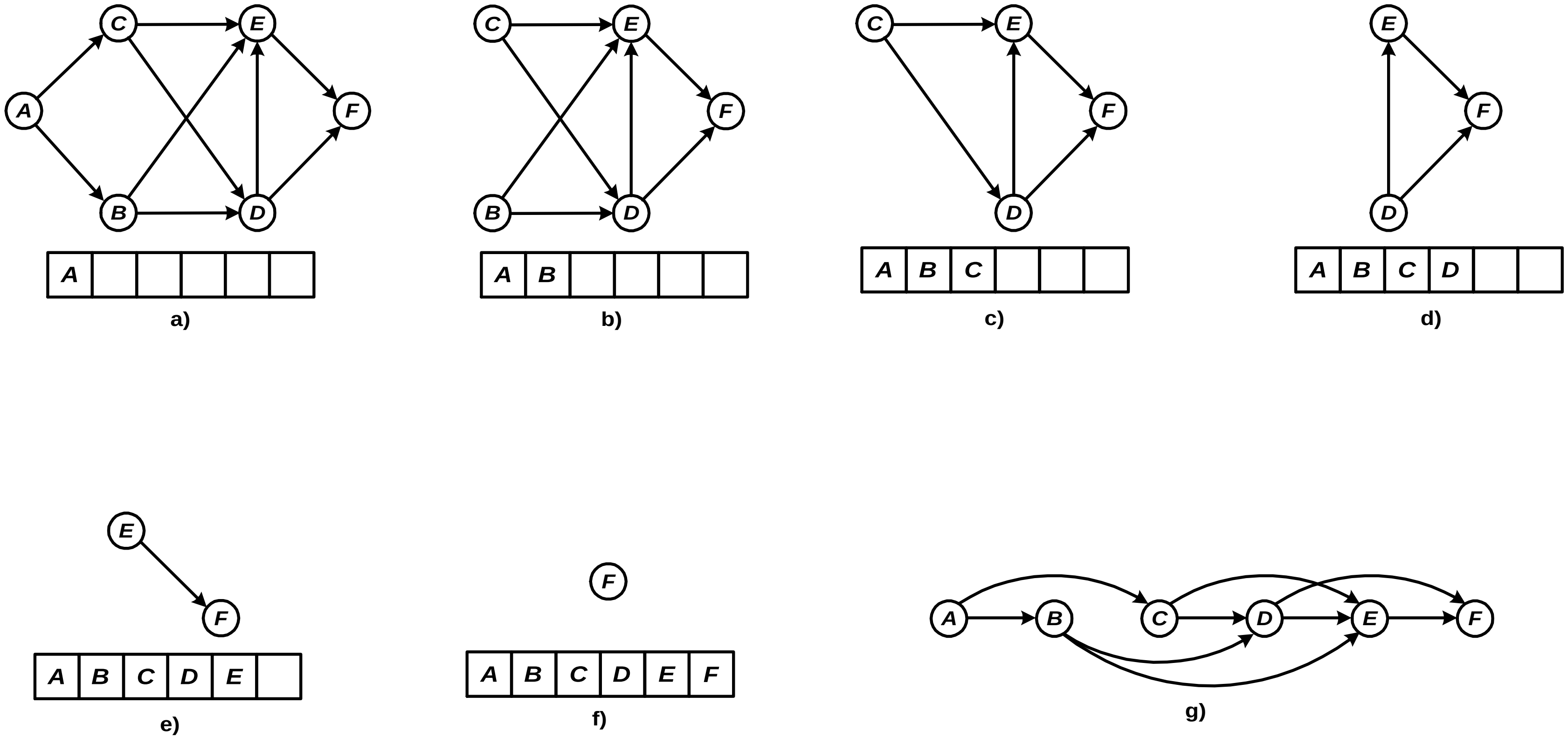
- Topološki poredak – linearni poredak čvorova tako da se, za svaku granu $(u, v) \in E$, čvor u pojavljuje ispred čvora v

Topološko sortiranje

```
TOP-SORT(G)
A = V
B = E
for i = 1 to n do
        find u ∈ A : d_in(u) = 0
        T[i] = u
        A = A – {u}
        for all v, (u, v) ∈ B do
                B = B - {(u, v)}
        end_for
end_for
```

- Vremenska složenost:
  - ✓ za matricu susednosti - $O(n^2)$
  - ✓ za liste susednosti - $O(e + n)$

Topološko sortiranje



Određivanje kritičnog puta

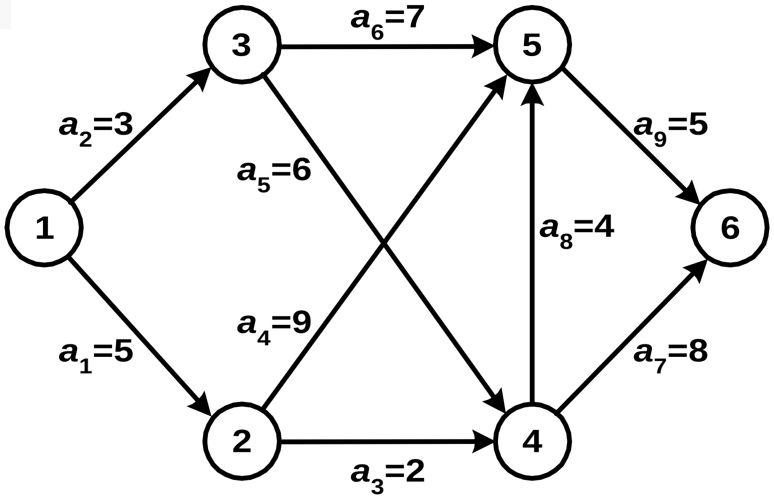
- Težina grane – trajanje aktivnosti
- Čvor – događaj uslovljen završetkom svih aktivnosti simuliranih ulaznim granama,
  pa mogu početi aktivnosti simulirane izlaznim granama
- Kritični put – put najveće dužine
  od početnog do završnog čvora
- Najranije vreme početka izlaznih aktivnosti
  $EST[i] = \max\{EST[j] + w(j, i) : j \in P(i)\}$
- Najkasnije vreme početka izlaznih aktivnosti
  $LST[i] = \min\{LST[j] - w(i, j) : j \in S(i)\}$

Određivanje kritičnog puta

```
CRITICAL-PATH(G)
TOP-SORT(G)
for u = 1 to n do
    i = T[u]
    for each j ∈ P(i) do
        EST[i] = max{EST[j] + w(j, i) }
    end_for
end_for
LST[n] = EST[n]
for u = n downto 1 do
    i = T[u]
    for each j ∈ S(i) do
        LST[i] = min{EST[j] - w(i, j) }
    end_for
end_for
for i = 1 to n do
    L[i] = LST[i] – EST[i]
end_for
```

$\sim O(e + n)$

Određivanje kritičnog puta



i	*EST*	*LST*	*L*
1	0	0	0
2	5	5	0
3	3	4	1
4	9	10	1
5	14	14	0
6	19	19	0

Aktivnost	*l*
*a*1	0
*a*2	1
*a*3	3
*a*4	0
*a*5	1
*a*6	4
*a*7	2
*a*8	1
*a*9	0