PROGRAMIRANJE II

P-09: Osnovi grafova

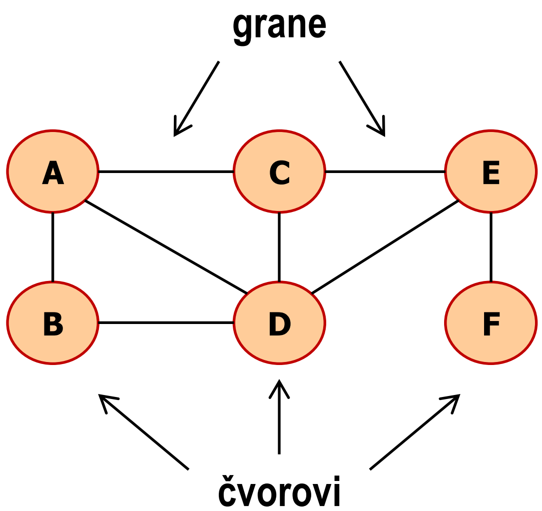
prof. dr **Dražen Brđanin**
2023/24

P-09: Grafovi

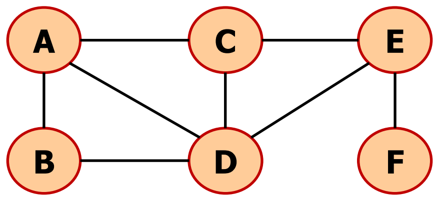
- **Sadržaj predavanja**
  - Osnovni pojmovi o grafovima
  - Reprezentacija grafa
  - Obilazak grafa
  - Obuhvatno stablo
  - Najkraća rastojanja

Osnovni pojmovi o grafovima

- **GRAF**:
  - nelinearna struktura podataka koju čine:
    - **V - skup čvorova** (nodes/vertices), i
    - **E - kolekcija grana** (edges/arcs)
  - **grane**
    - predstavljaju binarne veze (odnose) između čvorova
    - koriste se za modelovanje proizvoljnih nelinearnih relacija
    - grane mogu biti:
      - **neusmjerene (simetrične)** i
      - **usmjerene (asimetrične)**

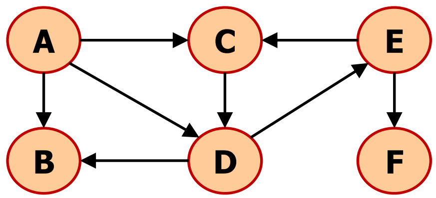


neusmjereni graf



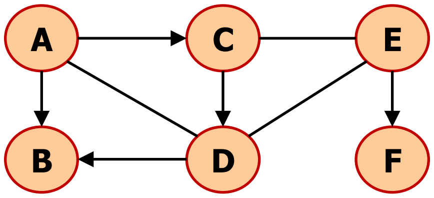
sve veze u grafu su neusmjerene

usmjereni graf



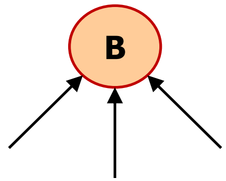
sve veze u grafu su usmjerene

mješoviti graf



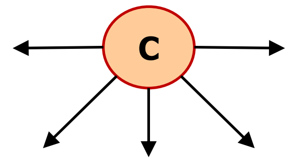
u grafu postoje i usmjerene i neusmjerene veze

Osnovni pojmovi o grafovima



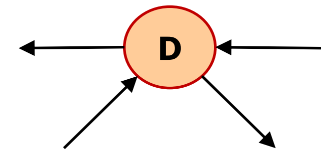
ulazne grane čvora
(grane usmjerenog grafa koje ulaze u dati čvor)

Ulazni stepen čvora (*in-degree*):
broj ulaznih grana u čvoru
$indeg(\mathrm{B})=3$



izlazne grane čvora
(grane usmjerenog grafa koje izlaze iz datog čvora)

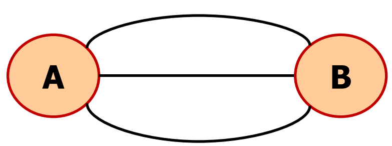
Izlazni stepen čvora (*out-degree*):
broj izlaznih grana iz čvora
$outdeg(\mathrm{C})=5$



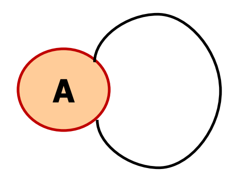
Stepen čvora (degree):
ukupan broj ulaznih i izlaznih grana u datom čvoru
$deg(\mathrm{D})=4$

Po definiciji, **graf čine skup čvorova i kolekcija grana**:

- svaka grana predstavlja vezu između dva čvora
- između neka dva čvora mogu da postoje **paralelne** ili **višestruke** veze/grane (npr. između čvorova u elektr. mreži može da postoji više grana)



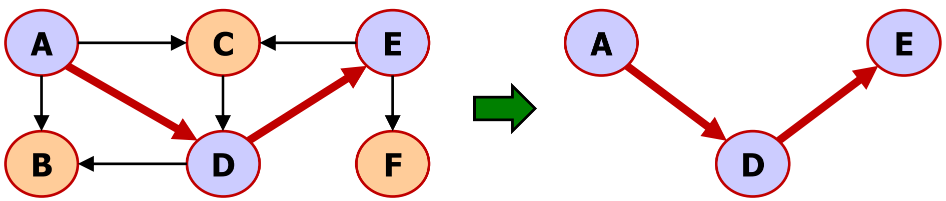
Petlja: grana koja počinje i završava istim čvorom



Prosti graf: graf u kojem nema petlji i višestrukih veza

Osnovni pojmovi o grafovima

Put/Putanja (*path*): alternativna (naizmjenična) sekvenca povezanih čvorova i grana (čvor-grana-čvor-...-čvor) između neka dva čvora u grafu

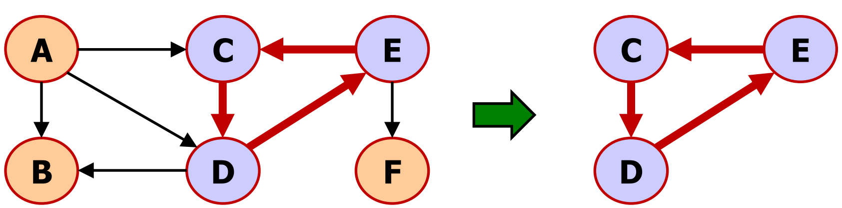


Prosta putanja – svi čvorovi u putanji su različiti (nijedan čvor nije sadržan više puta)

Usmjerena putanja – sve grane u putanji su usmjerene i obilaze se u odgovarajućem smjeru

Npr. putanja **A-(A,D)-D-(D,E)-E** je **prosta** (svaki čvor sadržan samo jednom) i **usmjerena** (obje grane su usmjerene i obilaze se u adekvatnom smjeru)

Ciklus (*cycle*): putanja koja ima isti početni i krajnji čvor



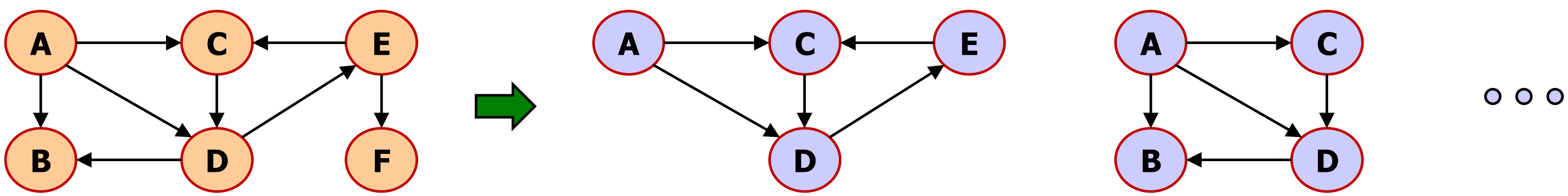
Prosti ciklus – svi čvorovi u ciklusu (osim početnog i krajnjeg) su različiti

Usmjereni ciklus – sve grane u ciklusu su usmjerene i obilaze se u odgovarajućem smjeru

Npr. ciklus **C-(C,D)-D-(D,E)-E-(E,C)-C** je **prost** (svaki čvor osim C je sadržan samo jednom) i **usmjeren** (sve grane su usmjerene i obilaze se u adekvatnom smjeru)

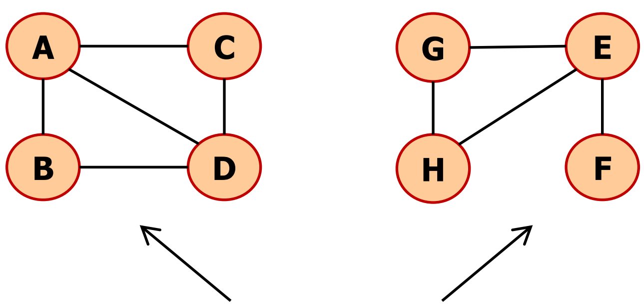
Osnovni pojmovi o grafovima

Podgraf grafa G je graf P čiji su čvorovi i grane podskupovi čvorova i grana grafa G



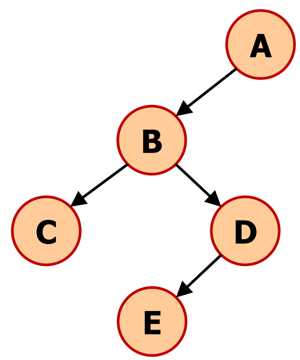
Povezani graf je graf u kojem između svaka dva čvora postoji putanja

Nepovezani graf je graf u kojem postoje čvorovi između kojih ne postoje putanje

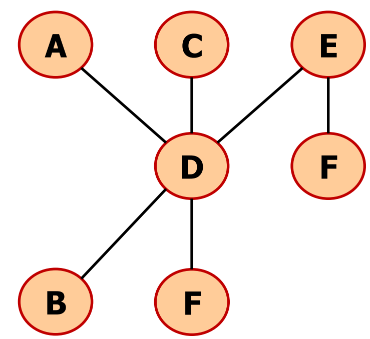


Povezane komponente su najveći povezani podgrafovi u nepovezanom grafu

Stablo je povezani graf bez ciklusa



korjeno stablo
(*rooted tree*)



slobodno stablo
(*free tree*)

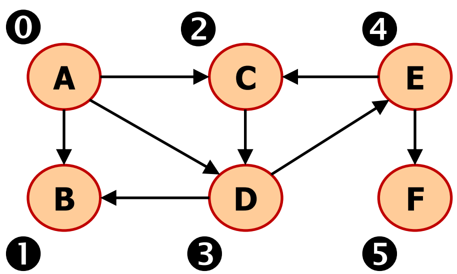
Reprezentacija grafa

Matrična reprezentacija (matrica susjednosti)

Sadržaj matrice definiše se na sljedeći način

$$a[i][j] = \begin{cases} 1, & (i,j) \in E \\ 0, & (i,j) \notin E \end{cases}$$

usmjereni graf

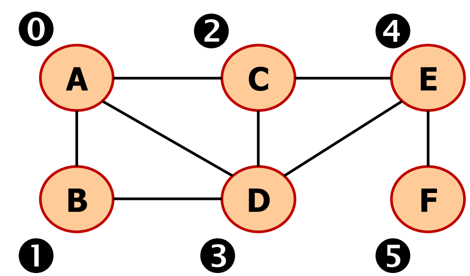


j

i

	0	1	2	3	4	5
0	0	1	1	1	0	0
1	0	0	0	0	0	0
2	0	0	0	1	0	0
3	0	1	0	0	1	0
4	0	0	1	0	0	1
5	0	0	0	0	0	0

neusmjereni graf

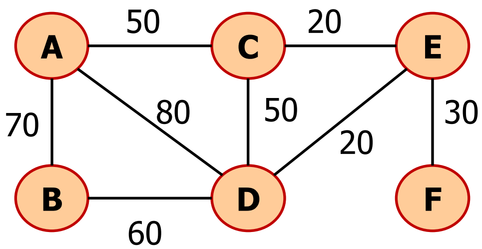


	0	1	2	3	4	5
0	0	1	1	1	0	0
1	1	0	0	1	0	0
2	1	0	0	1	1	0
3	1	1	1	0	1	0
4	0	0	1	1	0	1
5	0	0	0	0	1	0

Reprezentacija grafa

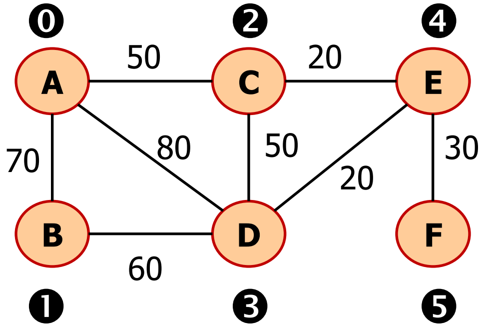
Težinski graf

Grane imaju težinu (npr. cijena, dužina, ...)



Sadržaj matrice susjednosti za težinske grafove

$$a[i][j] = \begin{cases} w(i,j), & (i,j) \in E \\ 0, & (i,j) \notin E \end{cases}$$



	0	**1**	**2**	**3**	**4**	**5**
0	0	70	50	80	0	0
1	70	0	0	60	0	0
2	50	0	0	50	20	0
3	80	60	50	0	20	0
4	0	0	20	20	0	30
5	0	0	0	0	30	0

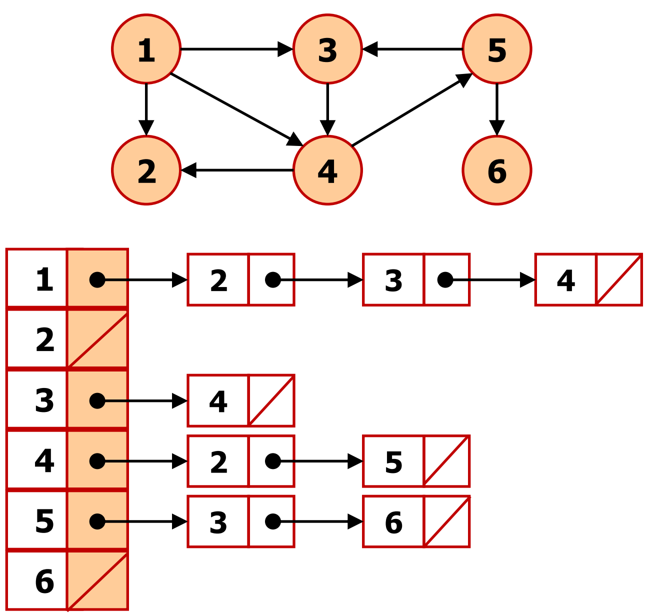
Reprezentacija grafa

Ulančana reprezentacija (lista susjednosti)

Graf se predstavlja pomoću:

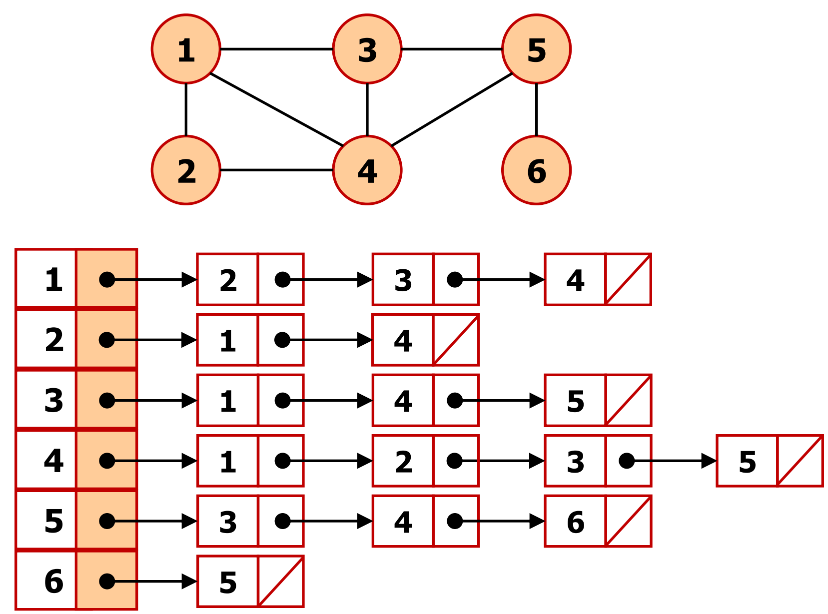
- **vektora zaglavlja za svaki čvor grafa**
- **ulančane liste za svaki čvor grafa**
- **svaki element u listi reprezentuje jednu granu**

usmjereni graf



vektor zaglavlja

neusmjereni graf



vektor zaglavlja

Obilazak grafa

Obilazak (eng. *traversal*) grafa:

- sistematična procedura kojom se svaki čvor u grafu posjeti (obradi) samo jednom
- na neki čvor može da se naiđe više puta, ali se samo prvi put obradi:
- graf nema neki specifičan čvor (kao što korjeno stablo ima korijen) od kojeg bi započeo obilazak
- ako se neki čvor ne posjeti zbog nedostižnosti, nastavlja se sa nekim neposjećenim
- **obilazak može da započne od**:
  - eksplicitno zadatog čvora
  - slučajno izabranog čvor
- **poredak** (rezultat obilaska) **zavisi od**:
  - izbora početnog čvora
  - strategije (algoritma) obilaska
- **osnovni algoritmi** za obilazak grafa zasnovani na susjednosti:
  - **obilazak po širini (Breath-First Search – BFS)**
  - **obilazak po dubini (Depth-First Search – DFS)**

Obilazak grafa

Obilazak grafa po dubini (DFS):

- **procedura veoma slična preorder obilasku** korjenog stabla
- **strategija obilaska**:
  - posjeti početni čvor
  - posjeti jednog njegovog susjeda
  - posjeti neposjećenog susjeda prethodnog čvora
  - ako nema neposjećenog susjeda, vrati se na posljednjeg prethodnika koji ima neposjećenog susjeda
- potreban **pomoćni niz** koji sadrži informaciju o posjećenim čvorovima, na početku:

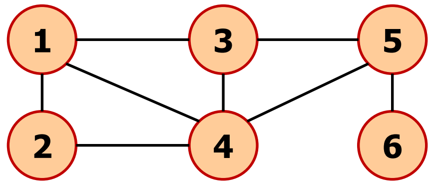
```
for i=1 to n do
  visit[i]=FALSE
```

- **rekurzivni obilazak** od nekog čvora *u*:

```
DFS(u)
  visit[u]=TRUE
  OBRADA(u)
  for { v, (u,v)∈E } do
     if visit[v]=FALSE then DFS(v)
```

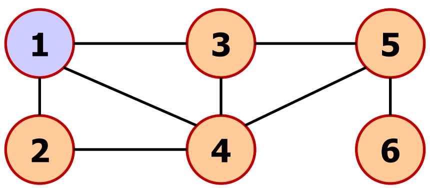
Obilazak grafa

Primjer: Obići dati graf po dubini počevši od čvora "1"



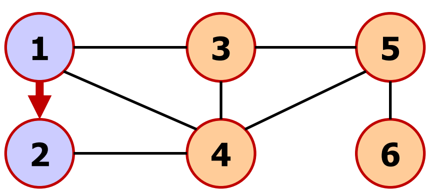
v[1]	v[2]	v[3]	v[4]	v[5]	v[6]
F	F	F	F	F	F

❶



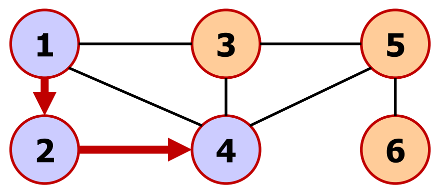
v[1]	v[2]	v[3]	v[4]	v[5]	v[6]
T	F	F	F	F	F

❷



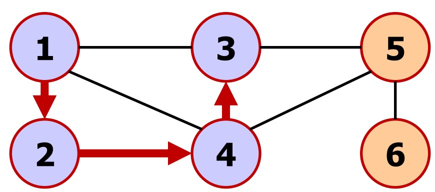
v[1]	v[2]	v[3]	v[4]	v[5]	v[6]
T	T	F	F	F	F

❸



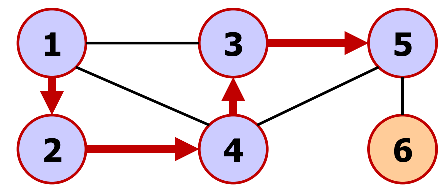
v[1]	v[2]	v[3]	v[4]	v[5]	v[6]
T	T	F	T	F	F

❹



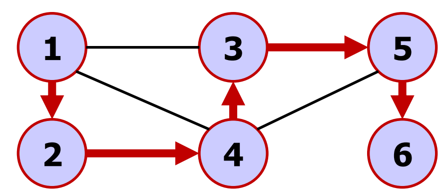
v[1]	v[2]	v[3]	v[4]	v[5]	v[6]
T	T	T	T	F	F

❺



v[1]	v[2]	v[3]	v[4]	v[5]	v[6]
T	T	T	T	T	F

❻



v[1]	v[2]	v[3]	v[4]	v[5]	v[6]
T	T	T	T	T	T

Rezultat obilaska: ❶→❷→❹→❸→❺→❻

Obilazak grafa

```c
/* obilazak grafa po dubini */
#include <stdio.h>
#define MAX 10
typedef struct G { int n; char nodes[MAX]; int ms[MAX][MAX]; } GRAF;
void DFS(GRAF *g)
{
    int visit[MAX]={};
    void dfs_visit(int u)
    {
       int v;
       printf("%c",g->nodes[u]);
       visit[u]=1;
       for (v=0; v<g->n; v++)
          if (g->ms[u][v] && !visit[v]) dfs_visit(v);
    }
    dfs_visit(0);
}
int main()
{
    GRAF g = { 6, {'1','2','3','4','5','6'},
               { {0,1,1,1,0,0}, {1,0,0,1,0,0}, {1,0,0,1,1,0},
                 {1,1,1,0,1,0}, {0,0,1,1,0,1}, {0,0,0,0,1,0} }};
    DFS(&g);
    return 0;
}
```

GRAF je struktura koja reprezentuje graf:

- **n – broj čvorova**
- **nodes – nazivi čvorova**
- **ms – matrica susjednosti**

DFS je funkcija za obilazak grafa (uvijek od čvora 0):

- **visit – pomoćni niz**
- **dfs_visit – obilazi graf**

GRAF g reprezentuje graf iz prethodnog primjera

Rezultat obilaska:

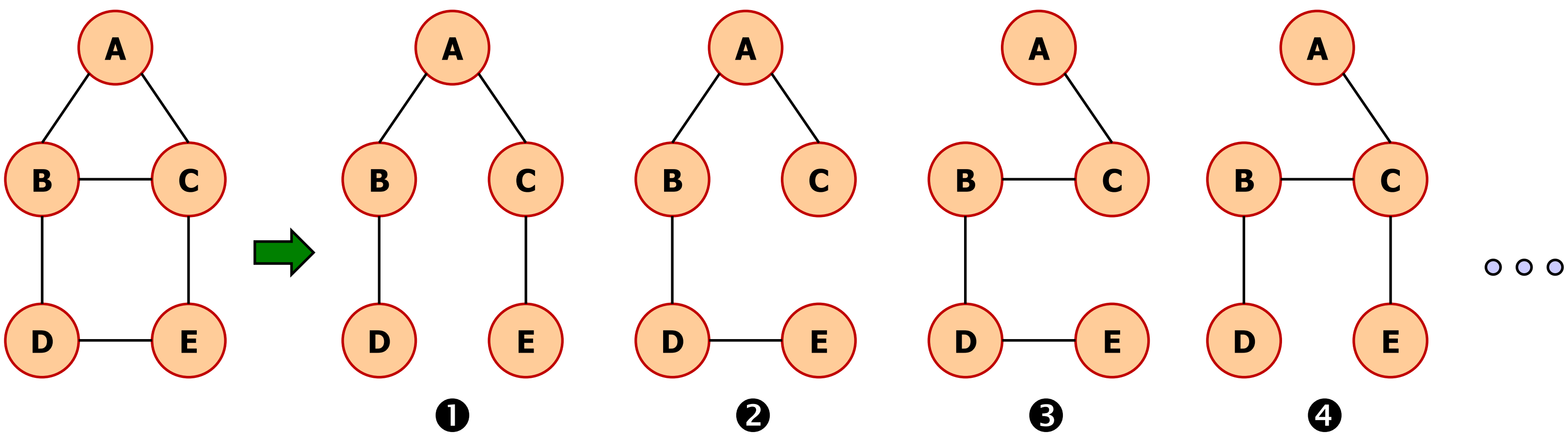
124356

Obuhvatno stablo (stablo razapinjanja)

obuhvatno stablo
(eng. *spanning tree*)

Obuhvatno stablo S(U,E′) povezanog neusmjerenog grafa G(V,E):

- **sadrži sve čvorove grafa (U=V)**
- **sadrži određen broj grana iz grafa tako da su svi čvorovi povezani, ali bez ciklusa (E′⊆E)**



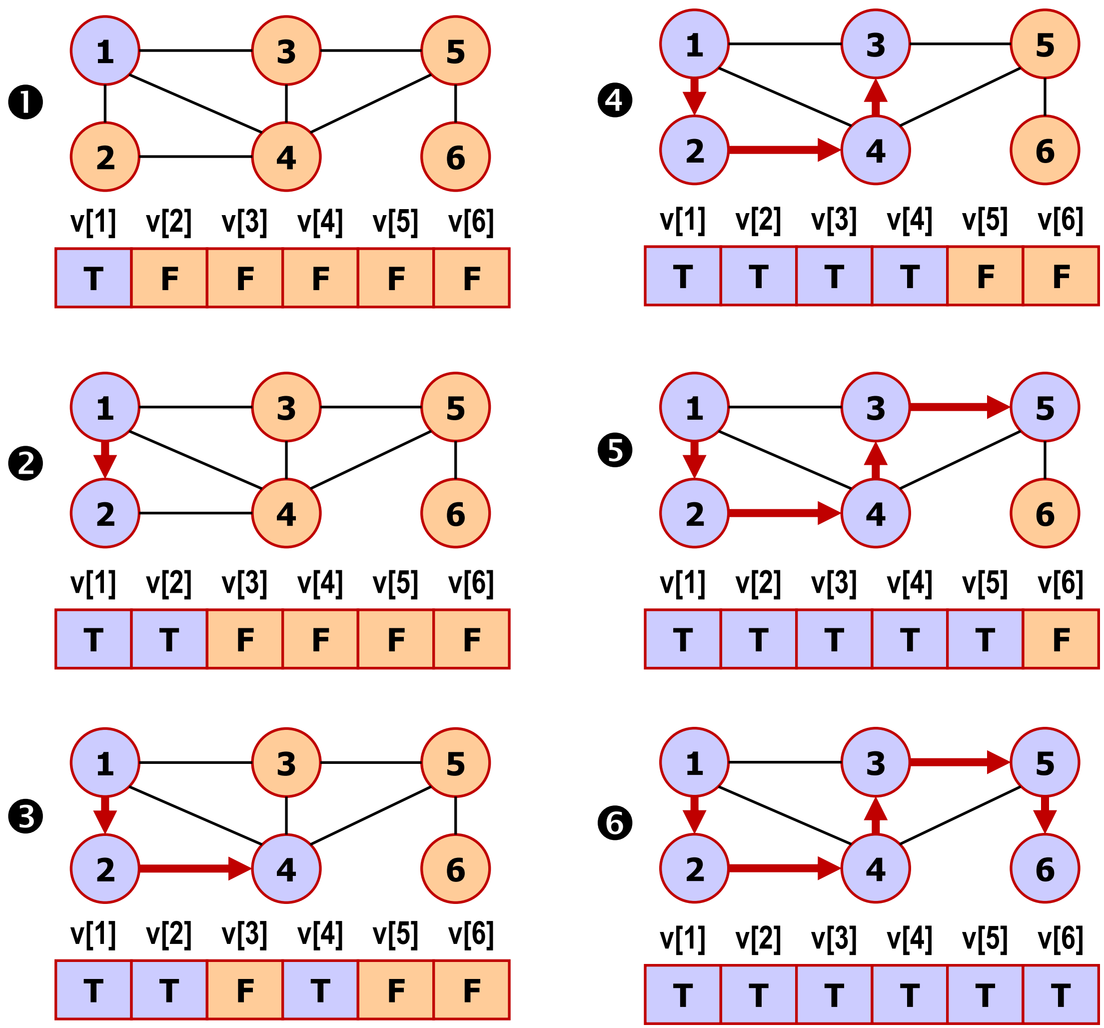
Obuhvatno stablo = slobodno stablo

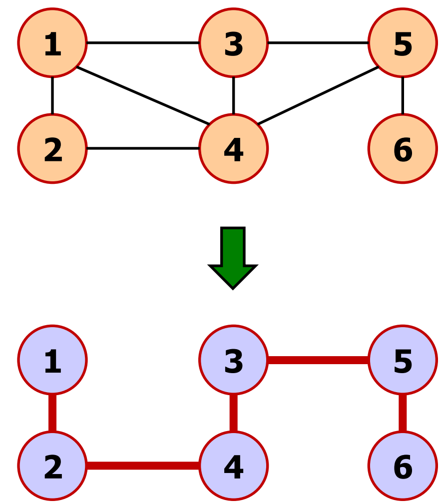
- **broj grana: *n*-1** (*n* – broj čvorova)
- **može da se generiše prilikom DFS ili BFS obilaska**

  (kad se dođe do neposjećenog čvora, u skup E′ se uključi dolazna grana)

Obuhvatno stablo (stablo razapinjanja)

Primjer: Formirati obuhvatno stablo obilaskom grafa po dubini počevši od čvora “1”





	1	2	3	4	5	6
1	0	1	0	0	0	0
2	1	0	0	1	0	0
3	0	0	0	1	1	0
4	0	1	1	0	0	0
5	0	0	1	0	0	1
6	0	0	0	0	1	0

Obuhvatno stablo (stablo razapinjanja)

```
/* formiranje obuhvatnog stabla obilaskom grafa po dubini */
#include <stdio.h>
#define MAX 10
typedef struct G { int n; char nodes[MAX]; int ms[MAX][MAX]; } GRAF;
void ST_DFS(GRAF *g, GRAF *st)
{
    int visit[MAX]={};
    void dfs_visit(int u)
    {
       int v;
       visit[u]=1;
       for (v=0; v<g->n; v++)
          if (g->ms[u][v] && !visit[v]) { st->ms[u][v]=st->ms[v][u]=1;  dfs_visit(v); }
    }
    dfs_visit(0);
}
int main()
{
    int i,j;
    GRAF g = { 6, {'1','2','3','4','5','6'},
             { {0,1,1,1,0,0},{1,0,0,1,0,0},{1,0,0,1,1,0},{1,1,1,0,1,0},{0,0,1,1,0,1},{0,0,0,0,1,0} }};
    GRAF st = { 6, {'1','2','3','4','5','6'} };
    ST_DFS(&g, &st);
    for (i=0; i<st.n; i++) { printf("\n"); for (j=0; j<st.n; j++) printf("%d ", st.ms[i][j]); }
    return 0;
}
```

Rezultat:

```
0 1 0 0 0 0
1 0 0 1 0 0
0 0 0 1 1 0
0 1 1 0 0 0
0 0 1 0 0 1
0 0 0 0 1 0
```

Minimalno obuhvatno stablo

Minimalno obuhvatno stablo S(U,E′) težinskog grafa jeste obuhvatno stablo minimalne cijene/težine

- **cijena stabla:** $$\sum_{(u,v)\in E} w(u,v)$$

(minimal spanning tree - MST)

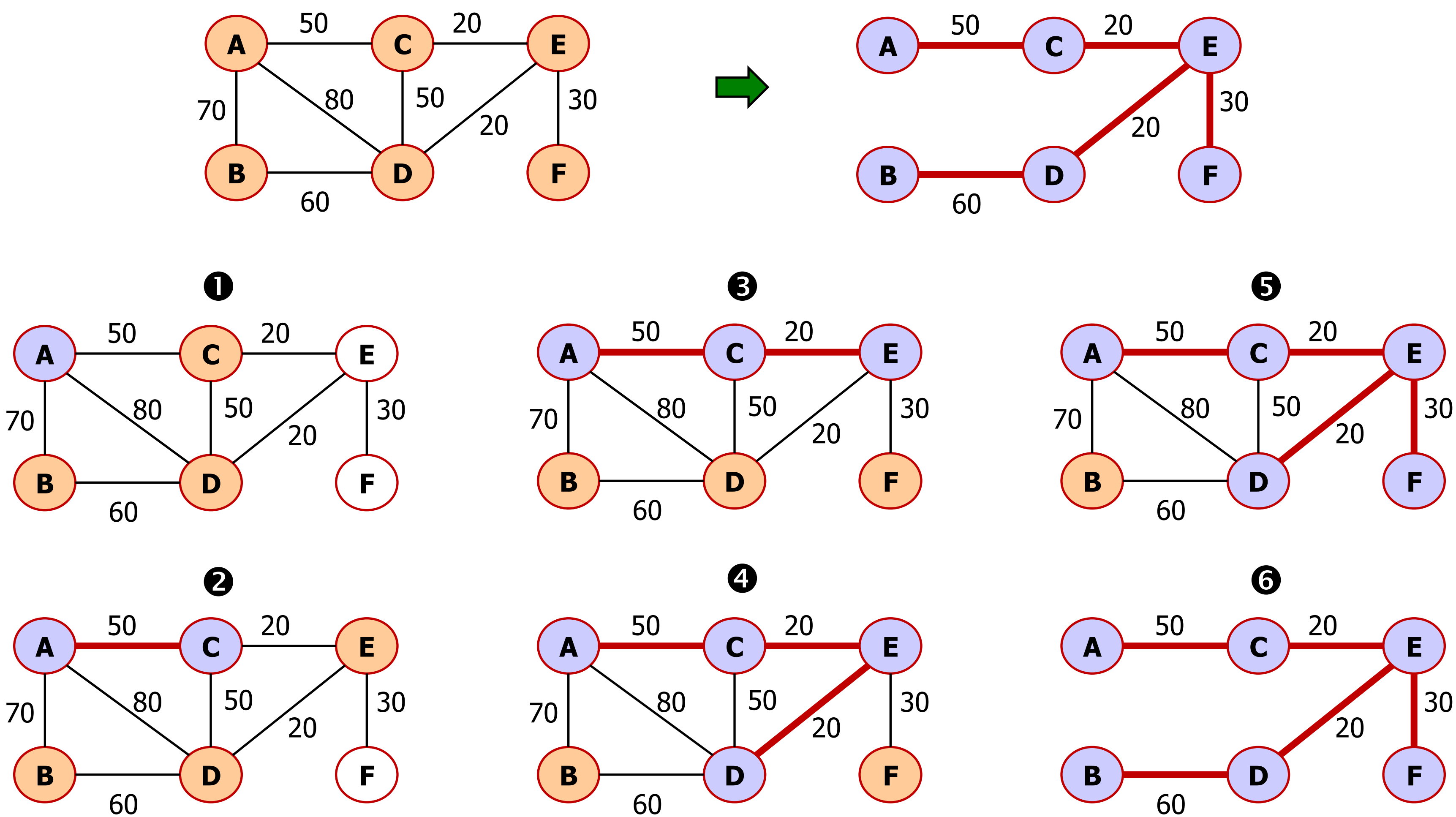
PRIM-ov ALGORITAM za MST

- **inkrementalno gradi MST, počevši od inicijalnog čvora, dodavanjem po jednu granu i jedan čvor**
- **Bira granu najmanje težine među granama koje povezuju čvorove već uključene u MST i čvorove koji još nisu**

```
PRIM(G,s)
   U={s}
   E'=∅
   while (U≠V) do
     find(u,v) ⇒ min {w(u,v) : (u∈U)∧(v∈(V-U))}
     U=U+{v}
     E'=E'+{(u,v)}
  end_while
  MST=(U,E')
```

Minimalno obuhvatno stablo

Primjer: Formirati minimalno obuhvatno stablo primjenom Primovog algoritma



Minimalno obuhvatno stablo

```
/* formiranje MST za neusmjereni graf Primovim algoritmom */
#include <stdio.h>
#include <limits.h>
#define MAX 10
typedef struct G { int n; char nodes[MAX]; int ms[MAX][MAX]; } GRAF;
void mst_prim(GRAF *g, GRAF *mst)
{
  int visit[MAX]={1};
  int allVisited()
  {
    for (int i=0; i<g->n; i++)
      if (visit[i]==0) return 0;
    return 1;
  }
  while (!allVisited())
  {
    int mg=INT_MAX, mu=-1, mv=-1;
    for (int u=0; u<g->n; u++)
      for (int v=0; v<g->n; v++)
        if (visit[u] && !visit[v] && g->ms[u][v])
          if (g->ms[u][v]<mg)
             mg=g->ms[u][v], mu=u, mv=v;
    if (mv>-1)
      visit[mv]=1, mst->ms[mu][mv]=mst->ms[mv][mu]=mg;
  }
}
```

```
int main()
{
  int i,j;
  GRAF g = { 6, {'A','B','C','D','E','F'},
           { {0,70,50,80,0,0},{70,0,0,60,0,0},
             {50,0,0,50,20,0},{80,60,50,0,20,0},
             {0,0,20,20,0,30},{0,0,0,0,30,0} }};
  GRAF mst = { 6, {'A','B','C','D','E','F'} };
  mst_prim(&g, &mst);
  for (i=0; i<mst.n; i++)
  {
     printf("\n");
     for (j=0; j<mst.n; j++)
        printf("%2d ", mst.ms[i][j]);
  }
  return 0;
}
```

Rezultat:

```
 0  0 50  0  0  0
 0  0  0 60  0  0
50  0  0  0 20  0
 0 60  0  0 20  0
 0  0 20 20  0 30
 0  0  0  0 30  0
```

Najkraća rastojanja

CILJ: Odrediti najkraće rastojanje (najkraći put) između dva čvora u grafu

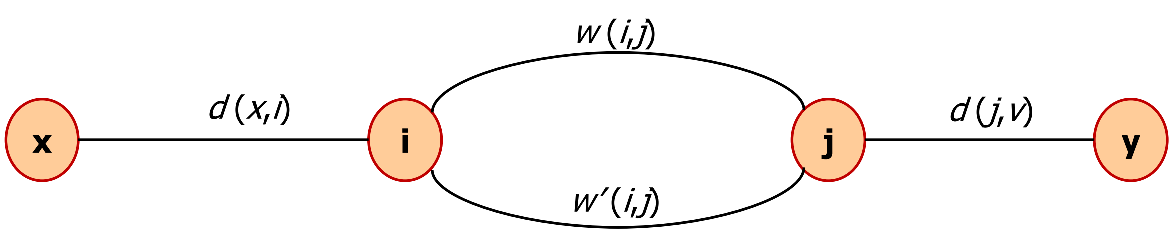
- težina grane reprezentuje rastojanje između dva čvora - $w(i,j)$
- dužina puta od čvora x do čvora y:

$$w(x,y) = \sum_{x \to y} w(i,j)$$

- najkraće rastojanje od čvora x do čvora y:

$$d(x,y) = \min w(x,y) = \min \sum_{x \to y} w(i,j)$$

- ako čvor y nije dostižan iz čvora x (ne postoji putanja), tada je $w(x,y)=\infty$



Najkraća rastojanja

FLOYD-ov algoritam za najkraće rastojanje između čvorova u grafu

- ulaz u algoritam: **matrica težina W**

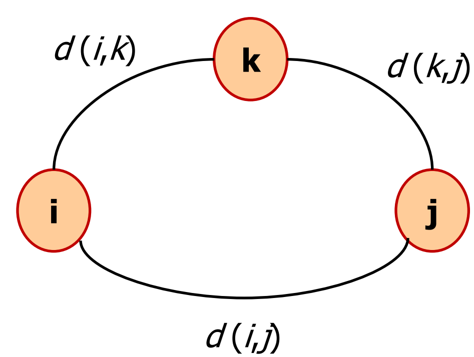
$$w[i][j] = \begin{cases} 0, & i = j \\ w(i,j), & (i,j) \in E \\ \infty, & (i,j) \notin E \end{cases}$$

- izlaz iz algoritma:
  - **matrica najkraćih rastojanja D**
  - **matrica prethodnika T**
- inicijalizacija matrice prethodnika

$$t[i][j] = \begin{cases} 0, & i = j \vee w(i,j) = \infty \\ i, & i \neq j \wedge w(i,j) < \infty \end{cases}$$

Osnovna ideja Floyd-ovog algoritma:

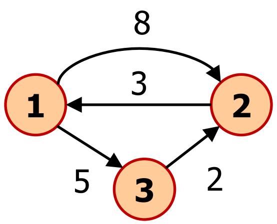
Za svaki par čvorova (*i,j*) treba provjeriti da li je put preko nekog drugog čvora *k* kraći



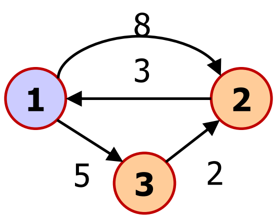

```
d(i,j)>d(i,k)+d(k,j)
    ⇒ d(i,j)=d(i,k)+d(k,j)
      t(i,j)=t(k,j)
```

Najkraća rastojanja

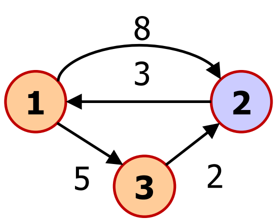
Primjer: Odrediti najkraća rastojanja u grafu primjenom Floyd-ovog algoritma



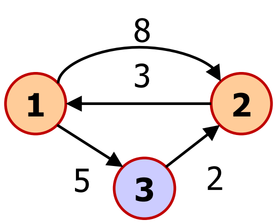
$$D^0 = \begin{bmatrix} 0 & 8 & 5 \\ 3 & 0 & \infty \\ \infty & 2 & 0 \end{bmatrix} \quad T^0 = \begin{bmatrix} 0 & 1 & 1 \\ 2 & 0 & 0 \\ 0 & 3 & 0 \end{bmatrix}$$



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



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



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

Rekonstrukcija putanje:

Npr. (1,2):

Posljednji čvor u putanji: **2**

Čvor prije čvora 2: t[1][2]=**3**

Čvor prije čvora 3: t[1][3]=**1**

Putanja: 1→3→2

Npr. (2,1):

Posljednji čvor u putanji: **1**

Čvor prije čvora 1: t[2][1]=**2**

Putanja: 2→1

Npr. (2,3):

Posljednji čvor u putanji: **3**

Čvor prije čvora 3: t[2][3]=**1**

Čvor prije čvora 1: t[2][1]=**2**

Putanja: 2→1→3

Najkraća rastojanja

```
/* Floydov algoritam za min rastojanja */

void FLOYD()
{
   int i,j,k;
   for (k=1; k<=n; k++)
     for (i=1; i<=n; i++)
       for (j=1; j<=n; j++)
         if (d[i][j]>d[i][k]+d[k][j])
         {
            d[i][j]=d[i][k]+d[k][j];
            t[i][j]=t[k][j];
         }
}
```

```
/* rekonstrukcija putanje */

void putanja(int i, int j)
{
  if (i==j)
     printf("%d",i);
  else
     if (t[i][j]==0)
        printf("Nema putanje\n");
     else
     {
        putanja(i,t[i][j]);
        printf("%d",j);
     }
}
```