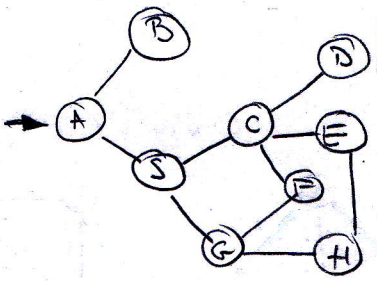
BFS

```
BFS (graph G) {
    init: ∀v num(v) = 0 ;
    brojac = 0;
    init red;
    while (∃ neposjećeni vrhovi v) {
        num(v) = ++brojac;
        red.add(v);
        while (red.IsNotEmpty()) {
            prvi = red.First();
            for (neposjećeni susjedi u od v) {
                num(u) = ++brojac;
                red.add(u)
            }
        }
    }
}
```

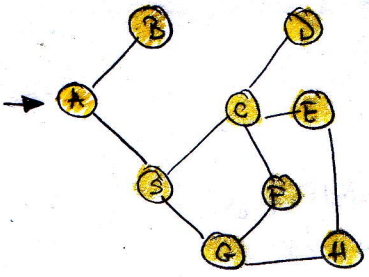


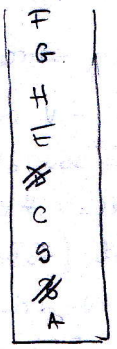
A B S C G D E F H

DFS

```
_DFS (graph G) {
    init: ∀v num(v) = 0
    brojac = 0;
    while (∃ neposjećeni vrhovi v u G) {
        DFS(v);
    }

DFS (v) {
    num(v) = ++brojac;
    for (susjedi u od v) {
        if (num(u) == 0) {
            DFS(u)
        }
    }
}
```

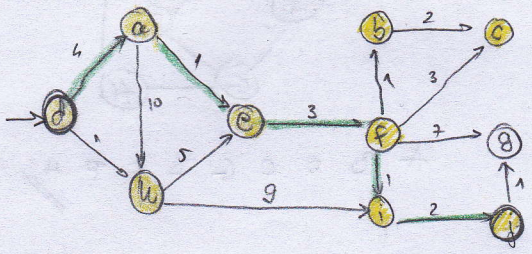




A B S C D E H G F

NAJKRAĆI PUTEVI U GRAFU

DIJKSTRIN A.

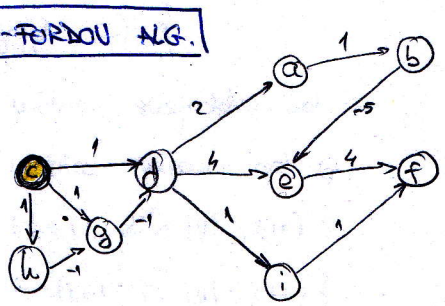


	d	h	a	e	f	b	i	c	j
a ∞	4	(4)$_d$	/	/	/	/	/	/	
b ∞	∞	∞	∞	∞	9^f	/	/	/	
c ∞	∞	∞	∞	∞	11^f	11^f	(11^f)	/	
d (0)	/	/	/	/	/	/	/	/	
e ∞	∞	6^h	(5^a)	/	/	/	/	/	
f ∞	∞	∞	∞	(8^e)	/	/	/	/	
g ∞	∞	∞	∞	∞	15^f	15^f	15^f	15^f	
h ∞	(1^d)	/	/	/	/	/	/	/	
i ∞	∞	10^h	10^h	10^h	9^f	(9^f)	/	/	
j ∞	∞	∞	∞	∞	∞	∞	11^i	(11^i)	

//

```
Dijkstra ( source, dest) {
    init: ∀ v  num(v) = ∞ ;
    num(source) = 0 ;
    ToBeCh = ∀ v ;
    while ( ToBeCh.IsNotEmpty() ) {
        v = vrh s min d ;
        if( v == dest) return;
        ToBeCh.remove(v) ;
        for( susjedi u od v ) {
            pom = d(v) + e(u,v) ;
            if( d(u) > pom) {
                d(u) = pom;
                prethodnik(u) = v
            }
        }
    }
}
```

BELLMAN-FORDOV ALG.



* $d_{new}(u) = d(v) + e(u,v)$

A	B	C	D	E	F	G	H	I
∞	∞	0	∞	∞	∞	∞	∞	∞
3	∞	0	1	5	9	1	1	2
3	∞	0	0	5	3	0	1	2
2	4	0	-1	-1	2	0	1	1
1	3	0	-1	-2	1	0	1	0
1	2	0	-1	-3	1	0	1	0

ab	be	cd	cg	ch	da	de	di	ef	gd	hg	if
x	x	✓	✓	✓	✓	✓	✓	✓			
									✓	✓	✓
✓	✓	/	/	/	✓	/	✓	/	✓	/	/
✓	✓	/	/	/	✓	/	✓	/	/	/	✓
✓	✓	/	/	/	/	/	/	/	/	/	/

```
BellmanFord_Slow(graph G)
    init: ∀v  d(v) = ∞
    d(source) = 0
    while( ∃ : d(u) > d(v) + e(v,u) ){
        d(u) = d(v) + e(v,u);
        predhodnik(u) = v;
    }
}

BellmanFord_Smart(graph G){
    init: ∀v d(v) = ∞
    d(source) = 0
    ToBeCh = source;
    while( ToBeCh.IsNotEmpty() ){
        v = vrh iz ToBeCh
        for(svi susjedi u od v){
            pom = d(v) + e(u,v)
            if(pom < d(u)){
                d(u) = pom
                predhodnik(u) = v
            }
        }
    }
}
```

WFI (21 2014)

D_0

	A	B	C	D	E
A	0	3	9	8	-3
B	8	0	8	7	1
C	8	8	0	8	8
D	2	8	-3	0	8
E	8	8	8	8	0

T_0

	A	B	C	D	E
A	/	A	A	/	A
B	/	/	/	B	B
C	/	/	/	/	/
D	D	/	D	/	/
E	/	/	/	E	/

D_1

	A	B	C	D	E
A	0	3	9	8	-3
B	8	0	8	7	1
C	8	8	0	8	8
D	2	(5)	-3	0	(-1)
E	8	8	8	8	0

	A	B	C	D	E
A	/	A	A	/	A
B	/	/	/	B	B
C	/	/	/	/	/
D	D	A	D	/	A
E	/	/	/	E	/

D_2

	A	B	C	D	E
A	0	3	9	(10)	-3
B	8	0	8	7	1
C	8	8	0	8	8
D	2	5	-3	0	-1
E	8	8	8	8	0

	A	B	C	D	E
A	/	A	A	B	A
B	/	/	/	B	B
C	/	/	/	/	/
D	D	D	A	D	/
E	/	/	/	E	/

$D_3 = T_2$

	A	B	C	D	E
A	0	3	9	10	-3
B	8	0	8	7	1
C	8	8	0	8	8
D	2	5	-3	0	-1
E	8	8	8	8	0

	A	B	C	D	E
A	/	A	A	B	A
B	/	/	/	B	B
C	/	/	/	/	/
D	D	D	A	D	/
E	/	/	/	E	/

D_4

	A	B	C	D	E
A	0	3	(7)	10	-3
B	(9)	0	(4)	7	1
C	8	8	0	8	8
D	2	5	-3	0	-1
E	(10)	(13)	(5)	8	0

	A	B	C	D	E
A	/	A	D	B	A
B	D	/	D	B	B
C	/	/	/	/	/
D	D	A	D	/	A
E	D	A	D	E	/

D_5

	A	B	C	D	E
A	0	3	(2)	(5)	-3
B	9	0	4	7	1
C	8	8	0	8	8
D	2	5	-3	0	-1
E	10	13	5	8	0

	A	B	C	D	E
A	/	A	D	(E)	A
B	D	/	D	B	B
C	/	/	/	/	/
D	D	D	A	D	/
E	D	A	D	E	/

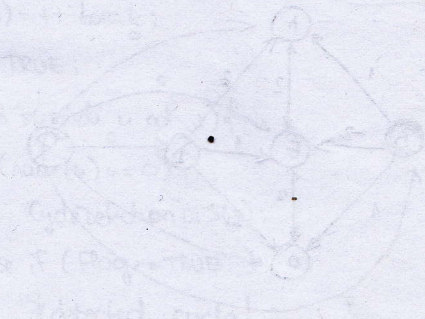
$A \to D$ =5

$A \xrightarrow{-3} E \xrightarrow{8} D$

upr.

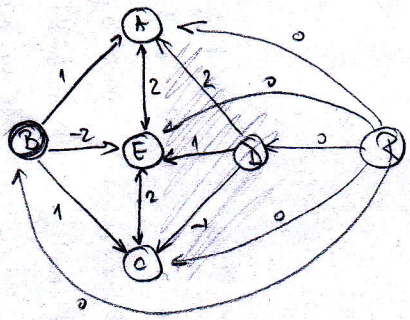
	A	B	C	D	E
A					
B					
C					
D					
E					

```
WFI (graph G) {
    D = matrica susjedstva od G
    P = matrica puteva od G
    for (k=1; k < |V| ; k++) {
        for(i=1; i<|V|; i++) {
            for(j= 1; j<|V|; j++) {
                if (D[i,j] > D[i,k] + D[k,j]) {
                    D[i,j] = D[i,k] + D[k,j];
                    P[i,j] = P[k,j];
                }
            }
        }
    }
}
```



RIJETKI GRAFOVI – TRANSFORMACIJA

(21 2011., ④)



① BF

R	R	D	R	R	
A	B	C	D	E	R
∞	∞	∞	∞	∞	0
0	0	0	0	0	0
0	0	-1	0	-2	0

~~A B C D E~~ E

⇒ d(A) = 0
d(B) = 0
d(C) = -1
d(D) = 0
d(E) = -2

② transformacija : $w'(A,B) = w(A,B) + d(A) - d(B)$

$w'(B,A) = 1 + 0 - 0 = 1$

$w'(B,E) = -2 + 0 - (-2) = 0$

$w'(B,C) = 1 + 0 - (-1) = 2$

$w'(A,E) = 2 + 0 - (-2) = 4$

$w'(E,A) = 2 + (-2) - 0 = 0$

$w'(E,C) = 2 + (-2) - (-1) = 1$

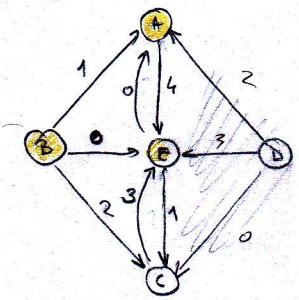
$w'(C,E) = 2 + (-1) - (-2) = 3$

$w'(D,A) = 2 + 0 - 0 = 2$

$w'(D,E) = 1 + 0 + 2 = 3$

$w'(D,C) = -1 + 0 + 1 = 0$

⇒



③ Dijkstra:

	B	E	A	C
A ∞	1	0 E	/	/
B 0	/	/	/	/
C ∞	2	1 E	1 E	/
D ∞	∞	∞	∞	∞
E ∞	0	/	/	/

④ Obrnuta transformacija: $L(A,B) = L'(A,B) + d(B) - d(A)$

$L(B,A) = 1 + 0 - 0 = 1$

$L(B,C) = 2 + (-1) - 0 = 1$

$L(B,D) = \infty + 0 - 0 = \infty$

$L(B,E) = 0 + (-2) - 0 = -2$

DETEKCIJA CIKLUSA

```
CycleDetectionDFS(graph G){
    init: ∀ v : num(v) = 0;
    brak = 0
    while( ∃ neobiđeni vrh v u G){
        CycleDetectionDFS(v);
    }
}
```

```
CycleDetectionDFS (v){
    num(v) = ++ brak;
    Flag = TRUE;
    for (svi susjedi u od v){
        if (num(u) == 0)
            CycleDetectionDFS(u);
        else if (Flag == TRUE)
            // detected cycle!
    }
    Flag = FALSE;
}
```

DISJOINT-SET

```
initialize(){
    for(i=1; i<|V|; i++){
        root[i] = next[i] = i;
        length[i] = 1;
    }
}

UnionFind(edge (v,u)){
    rt1 = root[u];
    rt2 = root[v];
    if (rt1 == rt2) return;
    else if (length[rt1] < length[rt2]){
        length[rt2] += length[rt1];
        root[rt1] = rt2;
        for(i = next[i]; i != rt1; i = next[i])
            root[i] = rt2;
    }
    else{
        // obrnuto ↑
    }
    swap(next[rt1], next[rt2]);
}
```

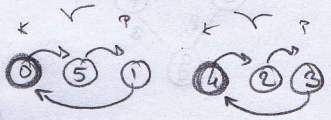
r: 0 1 2 3 4 5
n: 0 1 2 3 4 5
l: 0 1 2 3 4 5

union(0,1), union(4,3)

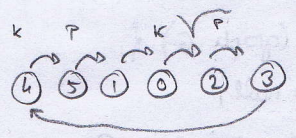
r: 0 0 2 4 4 5
n: 1 0 2 4 3 5
l: 2 1 1 1 2 1

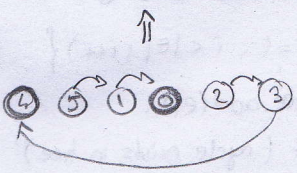
union(2,3), union(0,5)

r: 0 0 4 4 4 0
n: 5 0 3 4 2 1
l: 3 1 1 1 3 1



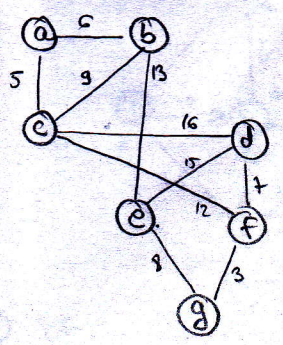
r: 4 4 4 4 4 4
n: 2 0 3 4 5 1
l: 3 1 1 1 6 1





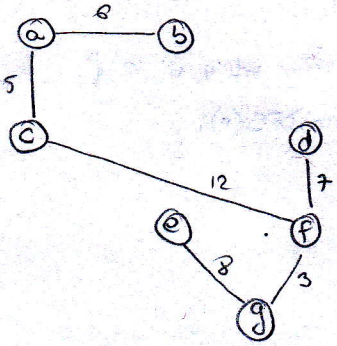
NAJKRAĆA RAZAPINJUĆA STABLA (MST)

KRUSKALOV ALG



5, 6, 8, 12, 16, 15, 12, 8, 3, 7

→ 3, 5, 6, 7, 8, 9, 12, 13, 15, 16



Krustal (graph G)

tree = null;

edges = sorted edges from G

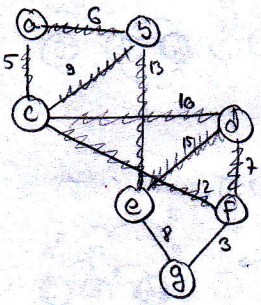
for(i=1 ; i ≤ |E| & |tree| < |V|-1 ; i++) {

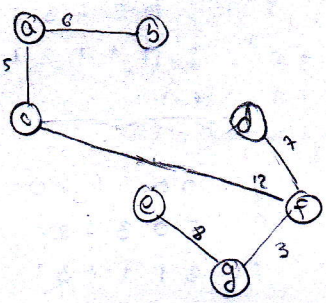
if(ei ne čini ciklus)

tree.add (ei)

$O(E \log_2 V)$

DIJKSTRIN ALG.





Dijkstra (graph G) {

tree = null;

edges = svi e iz G

for(i=1 ; i < |E| ; i++) {

tree.add(ei);

if (cycle exists in tree)

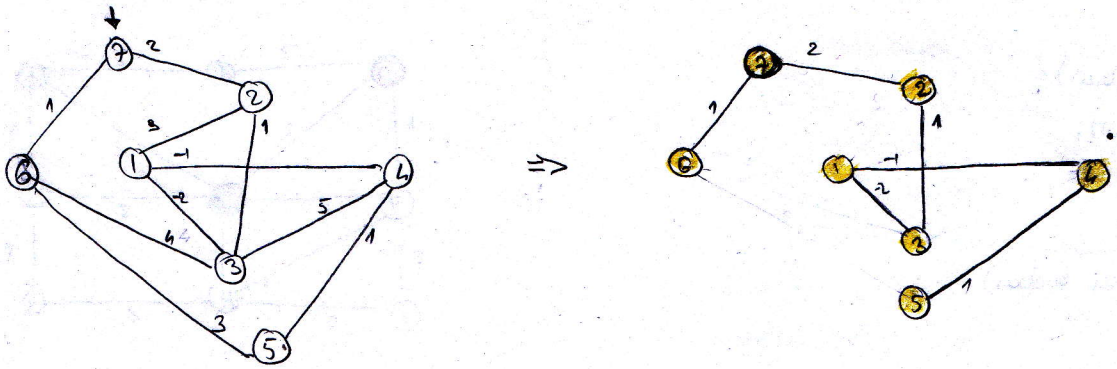
remove max e from cycle;

}

}

PRIHOV ALG

(21. 2013., ③)



```
Prim(graph G) {
    newG = random poč. vrh;
    tree = null;
    while ( |newG| < |V| ) {
        v = vrh iz G najbliži vrhom od vrhova iz newG
        tree.add( e(v,w) );
        newG.add (v);
    }
}
```

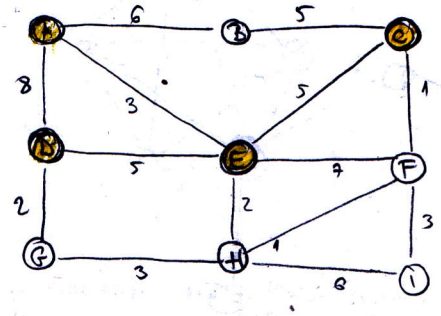
```
EulerCycleSlow(graph G){
  while(1){
    while (∃ brid e(v,u)){
      cikus += e(u,v);
      remove e iz G
      v=u
    }
    if (∃ z s prilježećim bridom)
      v=z
    else
      break;
  }
}
```

FLEURYEV ALG.

```
Fleury (graph G){
  v= random vrh v iz G
  path=v
  ToBeCh= svi e iz G
  while (∃ e(v,w) in ToBeCh)
    if ( e je jedini vrh iz v)
      choose e;
    else
      choose anything but bridge;
    path+=e;
    v=u
    remove e from ToBeCh;
  }
  if (ToBeCh.IsEmpty())
    success;
  else
    failure;
```

CHINESE POSTMAN (CPP) = obilazi sve ulice i vrati se na početak

(ZI 2013/2014. ⑥)



① ODD: vrhovi neparnog stupnja

A, C, D, E

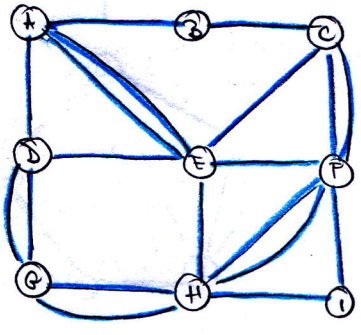
② najkraći putevi između ODD

AC (E) 8	DE 5	13
AD 8	CE 4 (F,H)	12
AE 5	CD 7 (F,H,G)	10

⇒ 10 + 57 = **67**

③ dodati bridove ⟶

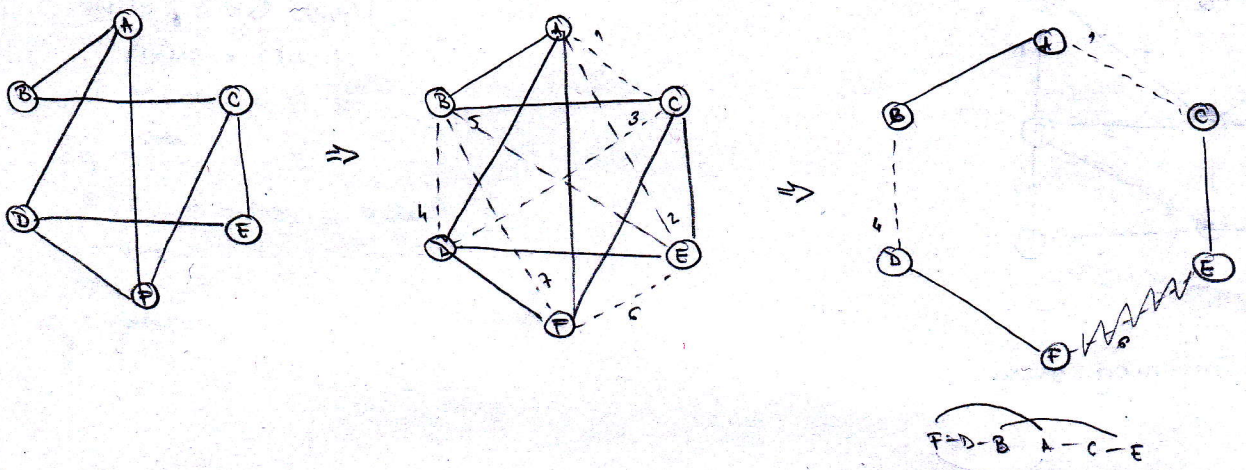
④ Pronađi Eulerov ciklus Fleuryevim alg.

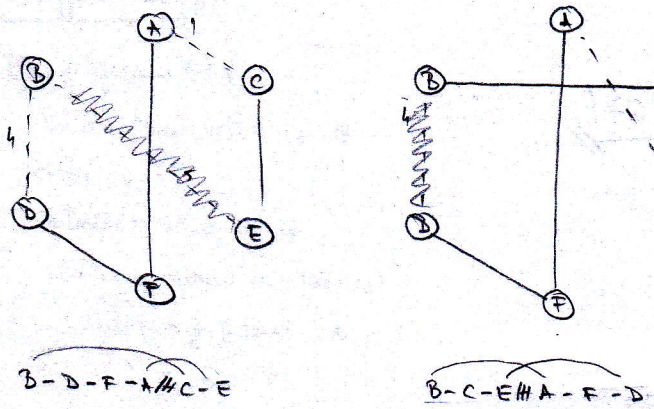


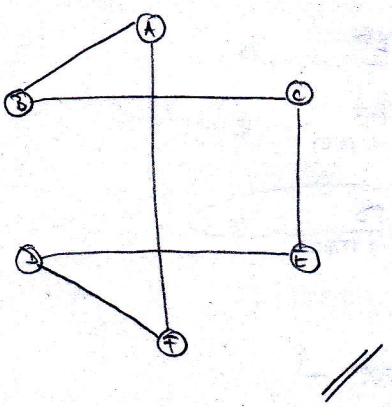
HAMILTONOVI GRAFOVI

V

BONDY-CHVÁTAL

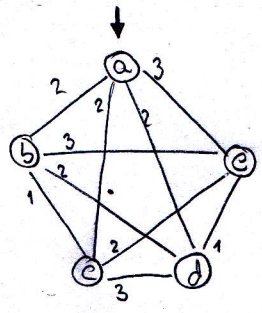




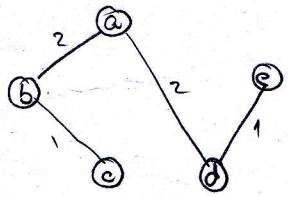


```
HamiltonianCycle (graph G)
    init: all edges num(e_i) = 0
    newEdges = 0;
    newG = G;
    while (newG has v : deg(v) + deg(u) ≥ |V|) {
        num(e(v,u)) = ++newEdges;
        newG += e(v,u);
    }
    if (∃ H.C.) {
        while ((t = max num in edges in c) > 0)
            C = new cycle with crossover
    }
```

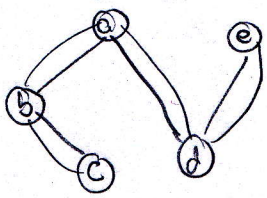
PROBLEM TRGOVAČKOG PUTNIKA (TSP)



1. konstruirati MST (Prim, Dijkstra, Kruskal)



2. Eulerov graf i e.c.



4. e.c. → h.c.