NASP - SPARSE TABLICA

① (5.) 4 mil. p.o. 4B pokazivači 8MB
30 razl. daksica 1B
~8 daksica 4B
20 god 2B

KLASIČNA:
4.000.000 × 30 × 20 = 2.400.000.000 × 4B = 9600 MB

RIJETKO POPUNJENA TABLICA:
4.000.000 × 8 × 20 × (4 + 2 + 1 + 3×8) = 2200 MB

Bolje je koristiti dvoju tablicu

b) Bolje je koristiti dvoju tablicu jer s porastom broja godina dobe tablice linearno rastu za sparse tablica ponaše zauzima više mem

② broj posjeta < 255 1B
~7 razl. gradova
25.000 stanovnika
Sparse 200 B
556 gradova

a) KLASIČNA TABLICA

25.000 × 556 × 1B = 13.9 MB

SPARSE TABLICA

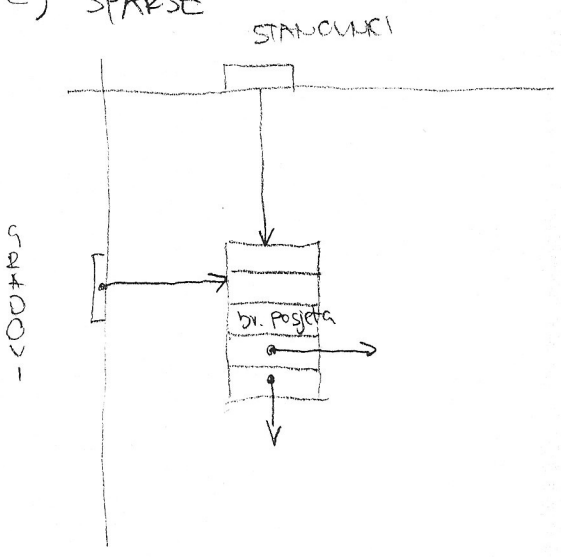
25000 × 7 × 200 B = 35 MB

Bolja je dvoja tablica

b) $M_{ST} = \frac{S \times PG}{SG} = \frac{PG}{G} = \frac{7}{556} = 0.013 = 1.3\%$

$M_{ST} = 100\%$

c) SPARSE



RB STABLO:

I DODAVANJE ČVORA:

1) NOVI ČVOR JE KORIJEN → PREBOJAMO GA U CRNO //
2) RODITELJ I UJAK SU CRVENI → PREBOJAMO P i U U CRNO, G U CRVE
   ↳ G JE NOVI ČVOR
3) RODITELJ CRVEN I UJAK CRN
   3.1) AKO JE N BLIŽI U ROTIRAMO OKO P
4) ROTIRAMO P OKO G UZ ZAMJENU BOJA P i G

II BRISANJE ČVORA

1) SIBLING S JE CRVEN → ROTIRAMO S OKO P
   ↳ BLIŽE DIJETE OD S JE NOVI S → NASTAVLJAMO S 2.
2) AKO JE S CRN I DJECA OD S CRNA (ILI IH NEMA)
   → S POSTAJE CRVEN
   → P POSTAJE ČVOR ZA PROMATRANJE
   2.1) AKO JE P CRVEN → CRN //
   2.2) AKO JE P CRN PROMATRA SE DALJE
   *) AKO JE P KORIJEN → CRNIMA SE ODBACUJE
3) AKO S IMA CRVENO DIJETE
   3.1) BLIŽE CRVENO → ROTIRA SE OKO S + ZAMJENA BOJA
4) ROTIRAMO S OKO P UZ ZAMJENU BOJA
   4.1) DALJE DIJETE OD S POSTAJE CRNO

AVL STABLO:

I DODAVANJE ČVORA:

1. FR(P) = ±2 ; FR(Q) = ∓1 → ROTIRAMO Q OKO P
2. FR(P) = ±2 ; FR(Q) = ∓1 → ROTIRAMO R OKO Q PA R OKO P

II BRISANJE ČVORA:

1. FR(P) = ±2 ; FR(Q) = ±1
   FR(P) = ±2 ; FR(Q) = 0
   } ROTIRAMO Q OKO P
2. FR(P) = ±2 ; FR(Q) = ∓1 → ROTIRAMO R OKO Q PA R OKO P

CRVENO CRNA STABLA

DEFINICIJSKA PRAVILA

1) svaki čvor je crven ili crn
2) korijen je crn
3) svaki list (čvor koji ne sadrži informaciju) je crn
4) dva potomka crvenog čvora su crna
5) svaka staza od nekog čvora do bilo kojeg lista koji je njegov potomak prolazi istim brojem crnih čvorova

DSW - CREATE PERFECT TREE

$h \leq \log_2(n+1)$

$k = 2^h - 1$

$n - k$ left rotation

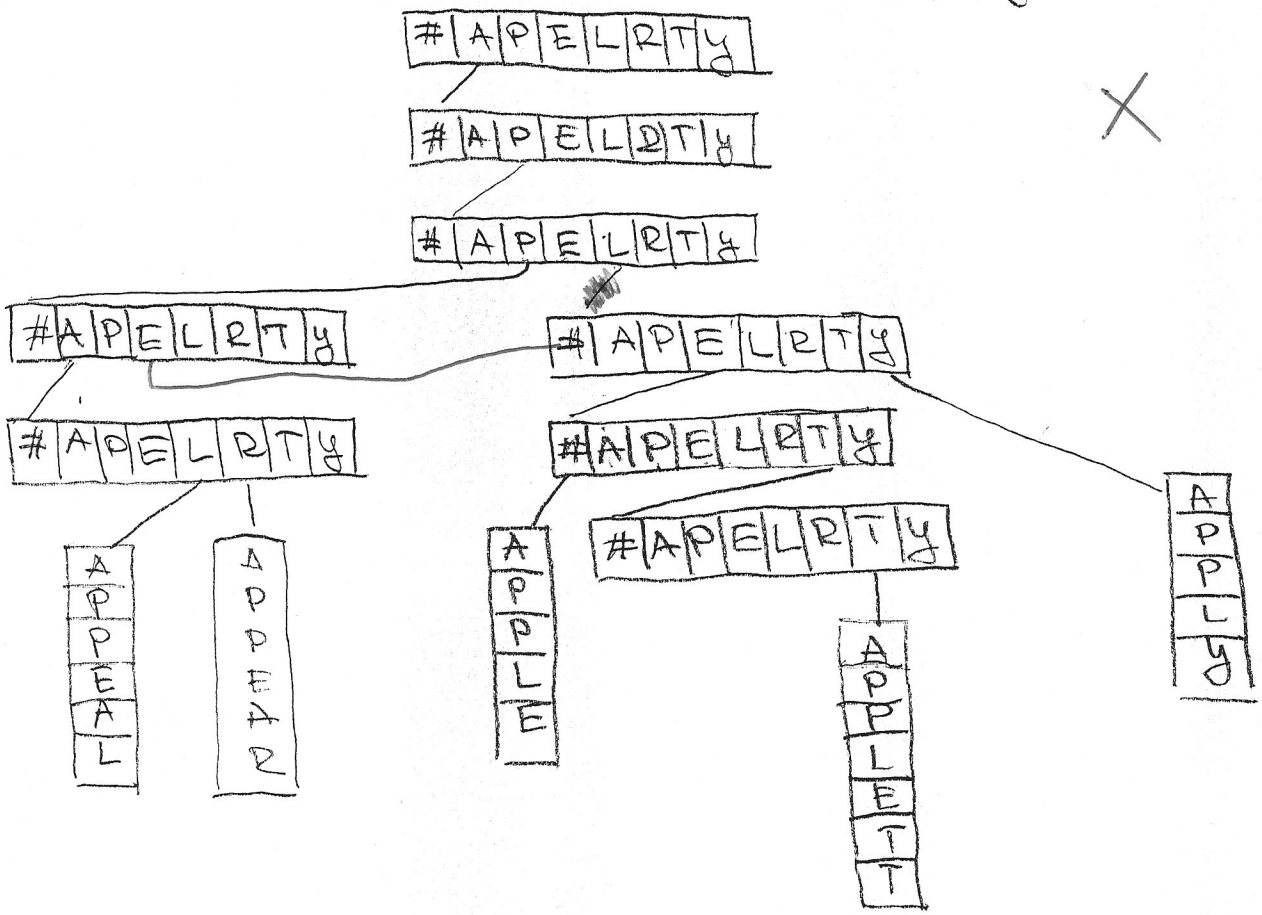
while ($k > 1$)

    $k = k/2$

    make k left rotations

NASP-TRIE

(3) ~~APPEAL~~, ~~APPEAR~~, APPLE, APPLETT, APPLY | A, P, E, L, R, T, Y



GRADIJENTNA METODA OPTIMIZACIJE

$f(x)$; $f: \mathbb{R}^n \to \mathbb{R}$

$x^{(k+1)} = x^{(k)} - \alpha_k \cdot \nabla f(x^{(k)})$; $\alpha_k > 0$

CILJNA F-JA ZA ADALINE: $f(w) = \frac{1}{2} \| x_d^T w - y_d \|^2$

GRADIJENT CILJNE F-JE: $\nabla f(w) = x_d (x_d^T \cdot w - y_d$

$= x_d \cdot \epsilon$

ITERACIJA: $w^{(k+1)} = w^{(k)} - \alpha_k x_d \cdot e^{(k)}$

$e^{(k)} = x_d^T \cdot w^{(k)} - y_d$

$\alpha_k = konst$

GENETSKI ALGORITMI - PSEUDOKOD

1) GENERACIJSKI GENETSKI

```
P = stvori pocetnu populaciju (VEL_POP)
evaluiraj (P)
ponavljaj dok nije kraj
    nova generacija P' = ∅
    ponavljaj dok je veličina (P') < VEL_POP
        odaberi R1, R2 iz P
        {D1, D2} = križaj (R1, R2)
        mutiraj D1, mutiraj D2
        dodaj D1, D2 u P'
    kraj
    P = P'
kraj
```

2) DETALJNI GENERACIJSKI GENETSKI

```
generacijski ga(VEL_POP, pm, pc, k, fja)        -> broj bitova kromosoma
    P = stvori slučajnu populaciju (VEL_POP, k)
    evaluiraj-populaciju (P, fja)
    ponavljaj
        -sortiraj- populaciju (P)
        P' = {P(1), P(2)}                       -> elitizam
        ponavljaj za i iz {1,2 ... VEL-POP/2 - 1}   -> zbog elitizma
            R1 = odaberi_roditelja (P)
            R2 = odaberi_roditelja (P)
            {D1, D2} = križaj_1_tocka_prijeloma (R1, R2, pc)
            mutiraj (D1, pm)
            mutiraj (D2, pm)
            P' += {D1, D2}
        kraj
        P = P'
        evaluiraj populaciju (P, fja)
    dok nije ogtaa
    vrati_najbolju_jedinku
kraj
```

3) ODABIR RODITELJA

- odabrani roditelja iz (P) Genetski

za svaku jedinku P_i iz P

dobrota(P_i) = NV-funkcija(P_i) → ovaj dio je varijabilan

kraj

SD = sumiraj_dobrote svih jedinki

za svaku jedinku P_i iz P

len(P_i) = dobrota(P_i) / SD

kraj

B = generiraj_slucajni_broj (0,1)

akumulirana suma()

za i=1 do VEL_POP

akumulirana suma += len(P_i)

ako je B < akumulirana suma vrati P_i

kraj

vrati zadnju jedinku iz populacije

kraj

4) križaj_1_tocka_prijeloma (R1, R2, Pc, k, +)

B = generiraj_slucajni_broj_iz_intervala (0,1)

ako je B > Pc

vrati {D1, D2}

inače

T = slucajno_odabrani_tocka_prijeloma (1, k-1)

za i iz {1...T}

D1(i) = R1(i)

D2(i) = R2(i)

kraj

za i iz {T+1, k}

D1(i) = R2(i)

D2(i) = R1(i)

kraj

vrati {D1, D2}

kraj kraj

5) mutiraj (D, Pm, k)

za i iz 1 do k

B = generiraj slucajni broj (0,1)

ako je B < pm

okreni_bit(D, i)

kraj

kraj

kraj

NASP - NEURON

p jednadžbi s n nepoznanica

$p = n \Rightarrow$ rješenje je jedinstveno ako ga ima

$p < n \Rightarrow$ beskonačno rješenja, odabiremo ono najmanje norme

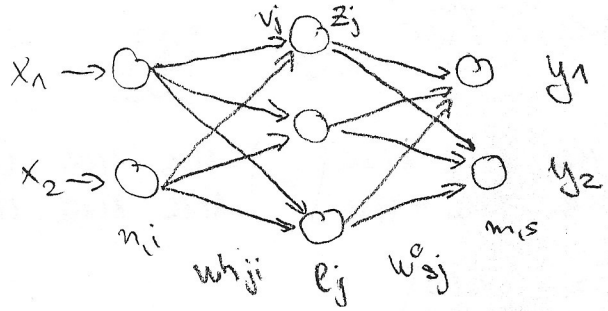
$p > n \Rightarrow$ točno rješenje može, ali ne mora postojati

$p < n \Rightarrow$ Kaczmanov algoritam

$p \geq n \Rightarrow$ Gradijentna metoda

NASP - NEURONSKE MREŽE

6.)



x_1	x_2	y_1	y_2
1	0	0	1

$k=0$: $xd_{1,1} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ $yd_{1,1} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

$W^{h(0)} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$ $W^{o(0)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

$\Theta^{h(0)} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ $\Theta^{o(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

FORWARD PASS

$v = W^{h(0)} \cdot xd_{1,1} - \Theta^h = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$

$z = \frac{1}{1-e^{-v}} = \begin{bmatrix} 1/2 \\ 1/2 \\ 1/2 \end{bmatrix}$

$u = W^{o(0)} \cdot z - \Theta^{o(0)} = \underset{2\times 3}{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}} \cdot \underset{3\times 1}{\begin{bmatrix} 1/2 \\ 1/2 \\ 1/2 \end{bmatrix}} = \underset{2\times 1}{\begin{bmatrix} 0 \\ 0 \end{bmatrix}}$

$y = \frac{1}{1-e^{-u}} = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}$

$EA^o = y - yd_{1,1} = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1/2 \\ -1/2 \end{bmatrix}$

$E1^o = EA^o .* y .* (1-y) = \begin{bmatrix} 1/2 \\ -1/2 \end{bmatrix} \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} = \begin{bmatrix} 1/8 \\ -1/8 \end{bmatrix}$

REVERSE PASS

$\delta^o = E1^o = \underset{2\times 1}{\begin{bmatrix} 1/8 \\ -1/8 \end{bmatrix}}$

$EW^o = \delta^o \cdot z^T = \begin{bmatrix} 1/8 \\ -1/8 \end{bmatrix} \underset{1\times 3}{\begin{bmatrix} 1/2 & 1/2 & 1/2 \end{bmatrix}} = \begin{bmatrix} 1/16 & 1/16 & 1/16 \\ -1/16 & -1/16 & -1/16 \end{bmatrix}$

$EA^h = W^{oT} \cdot E1^o = \underset{3\times 2}{\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}} \underset{2\times 1}{\begin{bmatrix} 1/8 \\ -1/8 \end{bmatrix}} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$

$E1^h = EA^h .* z .* (1-z) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$

$\delta^h = E1^h = \underset{3\times 1}{\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}}$

$EW^h = \delta^h \cdot xd_{1,1}^T = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$

$E\Theta^o = -E1^o = \begin{bmatrix} -1/8 \\ 1/8 \end{bmatrix}$ $E\Theta^h = -E1^h = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$

OSVJEŽAVANJE PARAMETARA

$$W^{h(1)} = W^{h(0)} - \alpha E W^h = \begin{bmatrix} c & c \\ c & c \\ c & c \end{bmatrix} \qquad \Theta^{h(1)} = \Theta^{h(0)} - \alpha \Theta^h = \begin{bmatrix} c \\ c \\ c \end{bmatrix}$$

$$W^{o(1)} = W^{c(0)} - \alpha E W^o = \begin{bmatrix} c & c & c \\ c & c & c \end{bmatrix} - \begin{bmatrix} 1/16 & 1/16 & 1/16 \\ -1/16 & -1/16 & -1/16 \end{bmatrix} = \begin{bmatrix} -1/16 & -1/16 & -1/16 \\ 1/16 & 1/16 & 1/16 \end{bmatrix}$$

$$\Theta^{o(1)} = \Theta^{c(0)} - \alpha E \Theta^o = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \begin{bmatrix} -1/8 \\ 1/8 \end{bmatrix} = \begin{bmatrix} 1/8 \\ -1/8 \end{bmatrix} //$$

Pseudo-kod (C# sintaksa) rekurzivnog rješenja Knapsack problema.

```
KnapsackRec (c,k)
if (c > 0 && k > 0)        //Ako to je element za razmatranje ...
{
   kNo = KnapsackRec(c,k-1);
   if (c >= cost[k])  //Ako je preostali maksimum veliki dovoljan...
      kYes = KnapsackRec(c-cost[k],k-1) + value[k];
   else
       kYes = kNo;
   if (kYes > kNo)
   { w[c,k] = kYes;    //Upis ostvarive vrijednosti u tablicu W(c,k).
     decisions[c,k] = true;    //k-ta ulazi u najbolji izbor
     return kYes; }
   else
   { w[c,k] = kNo;     //Upis ostvarive vrijednosti u tablicu W(c,k).
     decisions[c,k] = false; //k-ta ne ulazi u najbolji izbor
     return kNo;   }
}
else
   return 0;
```

Dodatak: skraćenje postupka

- pogodno za ljude, relativno nespretno za programiranje

	1	2	3	4
v	8	6	12	5
c	5	2	7	3

sort po c →

	2	4	1	3
v	6	5	8	12
c	2	3	5	7

	stvar 2	stvar 4	stvar 1	stvar 3
2	6 →	6 →	6 →	6
3	... 6	6 →	6 →	6
5	... 6	11 →	11 →	11
7	... 6	... 11	14 →	14
8	... 6	... 11	14	14
9	... 6	... 11	14	18
10	... 6	... 11	19 →	19
12	... 6	... 11	... 19	23

$v_0(\cdot) = 0$

$v_k(c) = \max\{v_{k-1}(c),\ v_{k-1}[c - \text{cost}(k)] + \text{value}(k)\}$

SIMPLEX

x_1 – number of component of type A
x_2 – B
x_3 – C

maximize: $P = 7x_1 + 8x_2 + 10x_3$

$x_1, x_2, x_3 \geq 0$

fabrication: $2x_1 + 3x_2 + 2x_3 \leq 1000$

assembly: $x_1 + x_2 + 2x_3 \leq 800$

$-7x_1 - 8x_2 - 10x_3 + P = 0$

$2x_1 + 3x_2 + 2x_3 + S_1 = 1000$

$x_1 + x_2 + 2x_3 + S_2 = 800$

	x_1	x_2	x_3	S_1	S_2	P		
S_1	2	3	2 (PIVOT)	1	0	0	1000	$\frac{1000}{2} = 500$
S_2	1	1	2	0	1	0	800	$\frac{800}{2} = 400$
P	-7	-8	-10	0	0	1	0	

$400 < 500$

PIVOT COLUMN

$\frac{1}{2} R_2 \to R_2$

	x_2	x_2	x_3	S_1	S_2	P	
S_1	2	3	2	1	0	0	1000
$\to S_2$	1/2	1/2	1	0	1/2	0	400
P	-7	-8	-10	0	0	1	0

$-2R_2 + R_1 \to R_1$

$10R_2 + R_3 \to R_3$

	x_1	x_2	x_3	S_1	S_2	P		
S_1	1	2 (PIVOT)	0	1	-1	0	200	$\frac{200}{2} = 100$
x_3	1/2	1/2	1	0	1/2	0	400	$\frac{400}{1/2} = 800$
	-2	-3	0	0	5	1	4000	

$100 < 800$

PIVOT COLUMN

$1/2 R_1 \to R_1$

	x_1	x_2	x_3	S_1	S_2	P	
$\to S_1$	1/2	1	0	1/2	-1/2	0	100
x_3	1/2	1/2	1	0	1/2	0	400
	-2	-3	0	0	5	1	4000

$-1/2\,R_1 + R_2 \rightarrow R_2$

$3\,R_1 + R_3 \rightarrow R_3$

	x_1	x_2	x_3	s_1	s_2	p		
x_2	(1/2) PIVOT	1	0	1/2	-1/2	0	100	$\frac{100}{1/2} = 200$
x_3	1/4	0	1	-1/4	3/4	0	350	$\frac{350}{1/4} = 1400$
	-1/2	0	0	3/2	7/2	1	4300	

PIVOT COLUMN

$200 < 1400$

$2R_1 \rightarrow R_1$

	x_1	x_2	x_3	s_1	s_2	p	
x_2	1	2	0	1	-1	0	200
x_3	1/4	0	1	-1/4	3/4	0	350
	-1/2	0	0	3/2	7/2	1	4300

$-\frac{1}{4}\,R_1 + R_2 \rightarrow R_2$

$\frac{1}{2}\,R_1 + R_3 \rightarrow R_3$

	x_1	x_2	x_3	s_1	s_2	p	
x_1	1	2	0	1	-1	0	200
x_3	0	-1/2	1	-1/2	1	0	300
	0	1	0	2	3	1	4400

$x_1 = 200$

$x_3 = 300$

$\Big\}$ max $p = 4400$

$x_2 = 0$

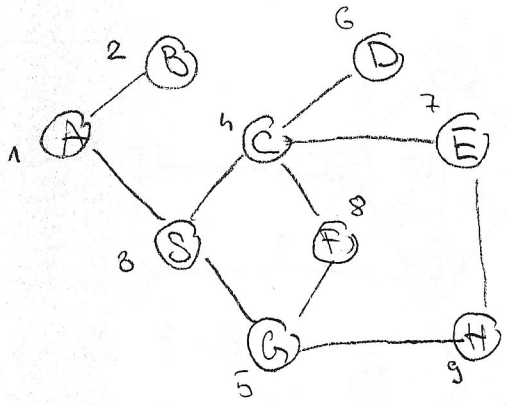
$s_1 = 0$

$s_2 = 0$

$\Big\}$ 0 hours of slack time //

BFS

```
BFS(Graph G)
    init ∀ v, num(v)=0;
    korak=0;
    init red;
    while(∀ neobiđeni vrh v){
        num(v) = ++korak;
        red.add(v);
        while(red.IsNotEmpty){
            prvi= red.First();
            for(neobiđeni susjedi u od v){
                num(u) = ++ korak;
                red.add(u);
            }
        }
    }
}
```

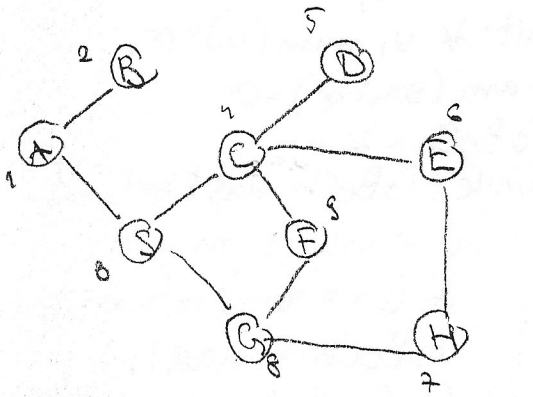


A-B-S-C-G-D-E-F-H

DFS

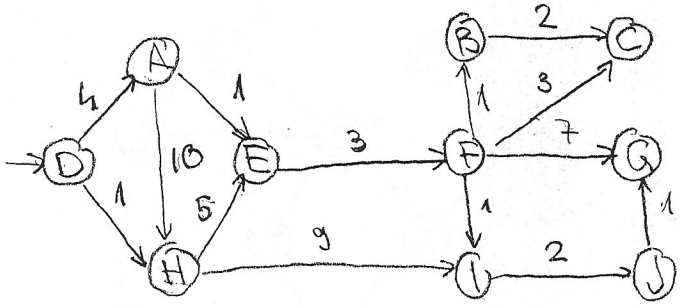
```
-DFS(Graph G){
    init: ∀v, num(v)=0;
    korak=0;
    while(∃ neobiđeni vrh v u G){
        DFS(v);
    }
}

DFS (vrh v){
    num(v)= ++korak
    for(susjed u od v){
        if(num(u)==0){
            DFS(u)
        }
    }
}
```



A-B-S-C-D-E-H-G-F

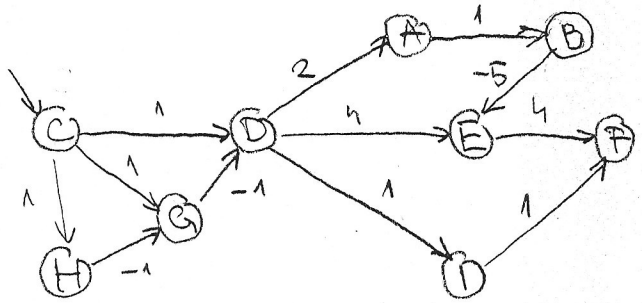
DIJKSTRA - NAJKRAĆI PUT



VRH	/	D	–	A	E	F	B	I	C	J	G
A	∞	4	(4)								
B	∞	∞	∞	∞	∞	(9)					
C	∞	∞	∞	∞	∞	11	11	(11)			
D	O										
E	∞	∞	6	(5)							
F	∞	∞	∞	∞	(8)						
G	∞	∞	∞	∞	∞	15	15	15	15	(12)	
H	∞	(1)									
I	∞	∞	10	10	10	9	(9)				
J	∞	∞	∞	∞	∞	∞	∞	11	(11)		

```
Dijkstra(source, dest)
    init ∀ v, num(v)=∞;
    num(source)=0;
    ToBeCh = ∀ v;
    while(ToBeCh.HasNext()){
        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 ALGORITAM



* dnew(u) = d(v) + e(u,v)

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

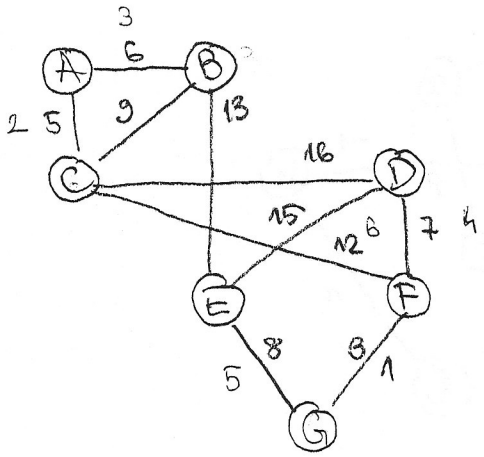
BRIDOVI: ab, be, cd, cg, ch, da, de, di, ef, gd, hg, if

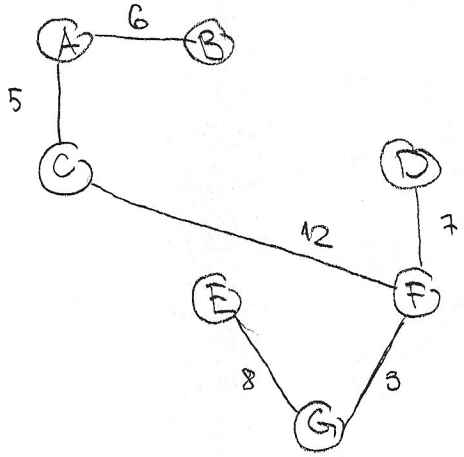
```
BellmanFord_Slow(Graph G){
    init ∀v, d(v)=∞;
    d(source)=0
    while(∃ d(u)>d(v)+e(v,u))
        d(u)=d(v)+e(v,u);
        prethodnik(u)=v;
    }
}
```

```
BellmanFord_Smart(Graph G){
    init ∀v, d(v)=∞;
    d(source)=0;
    ToBeCh=source;
    While(ToBeCh.HasNext){
        v=ToBeCh.Next();
        for(susjedi u od v){
            pom=d(u)+e(v,u)
            if(pom<d(u)){
                d(u)=pom;
                prethodnik(u)=v;
            }
        }
    }
}
```

GRAFOVI – MST

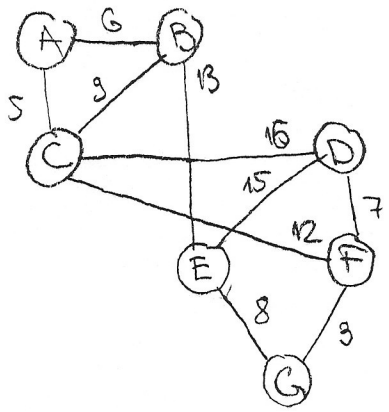
KRUSKALOV ALGORITAM

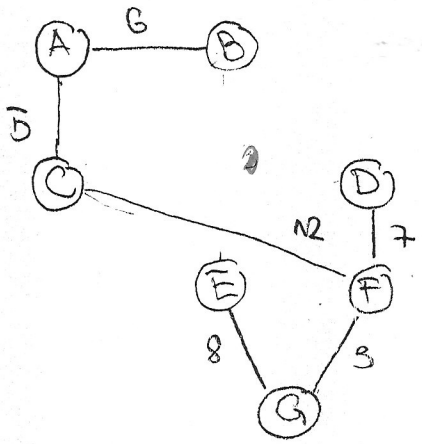





```
Kruskal(Graph G){
        tree=null
        edges= sorted edges from G;
        for(i=1; i≤|E| && |tree|<|V|-1; i++){
            if(edges[i] ne čini ciklus)
                tree.add(edges[i]);
        }
}
```

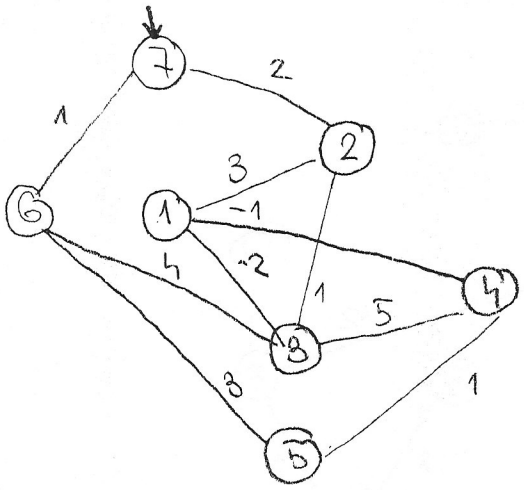
DIJKSTRIN ALGORITAM

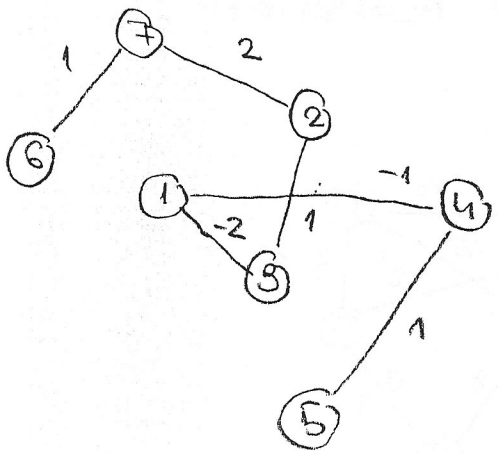





```
Dijkstra(Graph G){
    tree = null;
    edges = svi e iz G
    for(i=1; i≤|E|; i++){
        tree.add(ei);
        if(cycle.exists)
            remove max el from cycle();
    }
}
```

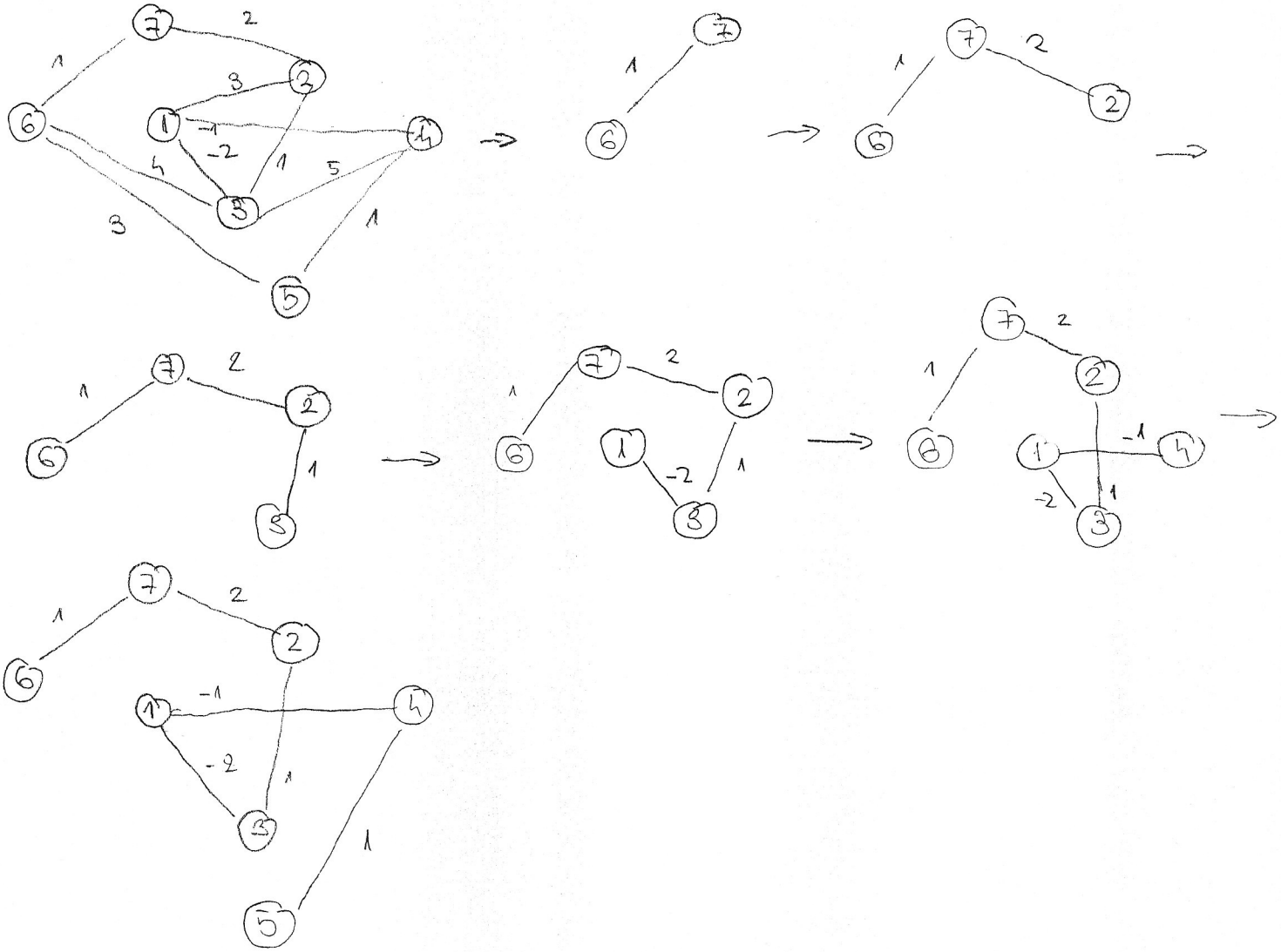
PRIMOV ALGORITAM





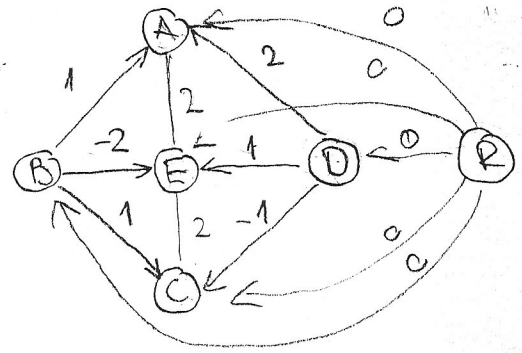

```
Prim(Graph G){
    newG = random poč. vrh;
    tree = null;
    while(|newG| < |V|){
        v = vrh iz G najbližji robom od vrhova iz newG
        tree.add(e(v,w));
        newG.add(v);
    }
}
```

NASP - GRAFOVI - PRIMOV ALGORITAM



GRAFOVI – DIJKSTRIN ALGORITAM

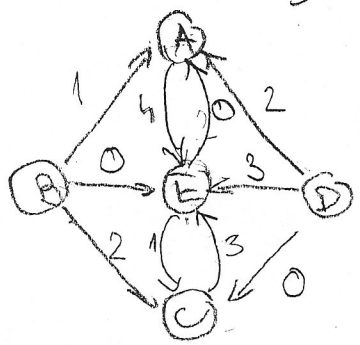
21. 2010/11. (4)



(1) BELLMAN – FORD

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

(2) TRANSFORMACIJA



$w(B,A) = 1+0-0 = 1$
$w(B,E) = -2+0-(-2) = 0$
$w(B,C) = 1+0-(-1) = 2$
$w(A,E) = 2+0-(-2) = 4$
$w(E,A) = 2+(-2)-0 = 0$
$w(C,E) = 2+(-1)-(-2) = 3$
$w(E,C) = 2+(-2)-(-1) = 1$
$w(D,A) = 2+0-0 = 2$
$w(D,E) = 1+0-(-2) = 3$
$w(D,C) = -1+0-(-0) = 0$

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

(3) DIJKSTRA

ITERACIJA / VRH	0	1	2	3	4
	/	B	E	A	C
A	∞	1	0//		
B	0//				
C	∞	2	1	1//	
D	∞	∞	∞	∞	∞//
E	∞	0//			

(4) OBRNUTA TRANSFORMACIJA

$L(B,A) = 1+0-0 = 1$
$L(B,C) = 2+(-1)-0 = 1$
$L(B,D) = \infty+0-0 = \infty$
$L(B,E) = 0+(-2)-0 = -2$

$$L(A,B) = L'(A,B) + d(B) - d(A)$$

GRAFOVI – WFI ALGORITAM

```
WFI(Graph G){
    D = matrica susjedstva od G;
    P = matrica putova 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];
            }
        }
    }
}
```

GRAFOVI – DETEKCIJA CIKLUSA

```
CycleDetectionDFS(Graph G){
    init *v : num(v)=0;
    korak = 0;
    while(∃ nedbiotni vrh v u G){
        CycleDetectionDFS(v);
    }
}
```

```
CycleDetectionDFS(Graph G){
    num(v) = ++korak;
    Flag = true;
    for(svi susjedi u od v){
        if(num(u)==0){
            prethodnik(u) = v;
            CycleDetectionDFS(u);
        }else if(Flag)
            DetektiranCiklus();
    }
    Flag = false;
}
```

EULEROVI GRAFOVI

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

FLEURYJEV ALGORITAM

```
Fleury(Graph G) {
    v= random vrh iz G;
    path=v
    ToBeCh= svi e iz G
    while (∃ e(v,u) u ToBeCh) {
        if(e je jedini vrh iz v)
            choose e;
        else
            choose anything but bridge;
        path += e;
        v=u i
        remove e iz ToBeCh;
    }
    if(ToBeCh.Empty)
        success;
    else
        failture;
}
```

HAMILTONOVI GRAFOVI

BONDY-CHVATAL

```
HamiltenCycle(Graph G){
    init: all edges num(e_i)=0;
    newEdges=0;
    newG=G;
    while(newG has v and deg(v)+deg(u)≥|V|){
        num(e(v,w))=++newEdges;
        newG+=e(v,u);
    }
    if(∃ H.C.){
        while((k=max num in edges in G)>0)
            C=new cycle with ercsscver
    }
}
```