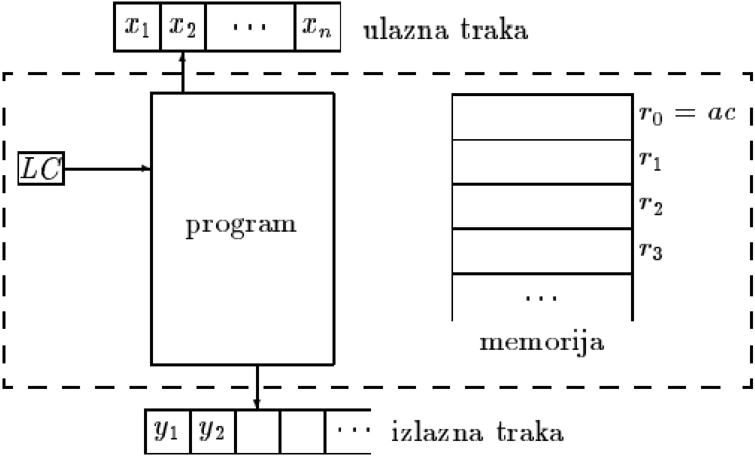
RAM, RASP, TM

na slajdovima nije sve zapisano - treba odgledati snimak predavanja

RAM - Random access machine

Slika mašine RAM:



Spisak naredbi:

kod op. adresa

1. LOAD operand
2. STORE operand
3. ADD operand
4. SUB operand
5. MULT operand
6. DIV operand
7. READ operand
8. WRITE operand
9. JUMP label
10. JGTZ label
11. JZERO label
12. HALT —

- Ispred naredbe može stajati labela pa ":".
- Opšti oblik naredbe je:

  labela: KodNaredbe operand/labela

Adresiranje

- Adresiranje (tj. argument operacije) može da bude
  - Neposredno, koristi se oznaka: $= i$
  - Direktno, koristi se oznaka: i
  - Indirektno, koristi se oznaka: $*i$
- Sadržaj memorije - odnosno registara memorije možemo zatati pomoću preslikavanja $c : \mathbb{N}_0 \to \mathbb{Z}$, gdje je $c(i)$ sadrzaj $i-$tog registra (tj. sadrzaj registra r_i).
- Stanje RAM mašine u određenom trenutku je određuju c i LC.
- Na početku izvršavanja je $c(i) = 0$ za svako $i \geq 0$, LC se odnosi na prvu naredbu programa P, a izlazna traka je potpuno prazna.
- Za operand a sa $v(a)$ označavamo vrijednost operanda a i određujemo ga sa:

$$v(= i) \equiv i, \quad v(i) \equiv c(i), \quad v(*i) = c(c(i)).$$

- Naredbe se izvršavaju jedna za drugom (kako su i napisane) osim kad naiđemo na instrukciju skoka.

Značenje naredbi

	Naredba	Značenje
1.	LOAD a	$c(0) \leftarrow v(a)$
2.	STORE i	$c(i) \leftarrow c(0)$
	STORE $*i$	$c(c(i)) \leftarrow c(0)$
3.	ADD a	$c(0) \leftarrow c(0) + v(a)$
4.	SUB a	$c(0) \leftarrow c(0) - v(a)$
5.	MULT a	$c(0) \leftarrow c(0) \cdot v(a)$
6.	DIV a	$c(0) \leftarrow [c(0)/v(a)]$
7.	READ i	$c(i) \leftarrow$ tekući ulazni simbol
	READ $*i$	$c(c(i)) \leftarrow$ tekući ulazni simbol
8.	WRITE a	štampa se $v(a)$
9.	JUMP b	$LC \leftarrow b$
10.	JGTZ b	ako je $c(0) > 0$ onda $LC \leftarrow b$, a inače $LC \leftarrow LC + 1$
11.	JZERO b	ako je $c(0) = 0$ onda $LC \leftarrow b$, a inače $LC \leftarrow LC + 1$
12.	HALT	izvršavanje se prekida

- RAM program P koji učitava n vrijednosti $x_1, \ldots, x_n$ i štampa jednu vrijednost y na izlaznu traku a zatim se zaustavi određuje funkciju $f_P : Z^n \to Z$ sa $f_P(x_1, \ldots, x_n) = y$.
- Za funkciju f kažemo da je RAM izračunljiva ako postoji RAM program P takav da je $f = f_P$. Za takav program P kažemo da računa f.
- Funkcija f jr RAM izračunljiva akko f izračunljiva po Tjuringu.
- Neka je $A = \{1, 2, \ldots, k\}$ azbuka. Neka program P učitava simbole azbuka A sve dok ne učita nulu. Za riječ $w = a_1 \cdots a_n$ kažemo da pripada jeziku programa P, u oznaci L_P, ako kada na ulaznu traku upišemo simbole $a_1, \ldots, a_n, 0$ program na izlaznu traku upiše broj 1 i zaustavi se.
- Kažemo da program P prepoznaje (ili prihvata) jezil L_P.

Tri zadatka

- Zadatak 1. Sastaviti program za RAM koji računa funkciju $f\ :\ \mathbb{Z} \to \mathbb{Z}$ datu sa:

$$f(n) = \begin{cases} n^n & , \quad \text{ako je } n \geq 1 \\ 0 & , \quad \text{inače} \end{cases}$$

- Zadatak 2. Neka je $A = \{1, 2\}$ azbuka i $L = \Big\{ w \in \Omega(A)\ :\ cnt(1, w) = cnt(2, w) \Big\}$ jezik, gdje je $cnt(i, w)$ broj simbola i u rječi w. Sastaviti program za RAM koji prihvata L.

- Zadatak 3. Sastaviti program za RAM koji učitava pozetivne brojeve i smješta ih u registre redom počev od registra r_{11}. Učitavanje se zaustavlja kad se sa ulazne trake pročita broj 0.

$r_0 = ac$	
r_1	n
r_2	$n, \ldots, n^n$
r_3	$n-1, \ldots, 0$

read R1;
if R1 $\leq$ 0 then write 0 else
    begin R2 $\leftarrow$ R1; R3 $\leftarrow$ R1 $-$ 1;
    while R3 $>$ 0 do
        begin R2 $\leftarrow$ R2 $\cdot$ R1;
        R3 $\leftarrow$ R3 $-$ 1 end;
    write R2 end

```
          READ 1
          LOAD 1
          JGTZ pos
          WRITE= 0
          JUMP endif
     pos: LOAD 1 ⤳
          STORE 2
          LOAD 1 ⤳
          SUB= 1
          STORE 3
   while: LOAD 3
          JGTZ continue
          JUMP endwhile
continue: LOAD 2
          MULT 1
          STORE 2
          LOAD 3
          SUB= 1
          STORE 3
          JUMP while
endwhile: WRITE 2
   endif: HALT
```

$r_0 = ac$	
r_1	x
r_2	d
	$\dots$

```
D ← 0;
read X;
while X ≠ 0 do
      begin
      if X ≠ 1 then D ← D − 1
           else D ← D + 1;
      read X
      end;
if D = 0 then write 1
      else write 0
```

```
           LOAD= 0
           STORE 2
           READ 1
    while: LOAD 1
           JZERO endwhile
           LOAD 1 ⇝
           SUB= 1
           JZERO one
           LOAD 2
           SUB= 1
           STORE 2
           JUMP endif
      one: LOAD 2
           ADD= 1
           STORE 2
    endif: READ 1
           JUMP while
 endwhile: LOAD 2
           JZERO output
           WRITE= 0
           HALT
   output: WRITE= 1
           HALT
```

$ac = r_0$	
r_1	$11, 12, \ldots, n+10$
r_2	$x_1, x_2, \ldots, x_n$
	$\ldots$
r_{11}	x_1
	$\ldots$
r_{k+10}	x_k
	$\ldots$
r_{n+10}	x_n
	$\ldots$

```
   R1 ← 11;
1: read R2;
   R0 ← R2;
   if R0 = 0 then stop else
        store* 1;
   R1 ← R1 + 1;
   goto 1
```

```
      LOAD= 11
      STORE 1
read: READ 2
      LOAD 2
      JZERO stop
      STORE* 1
      LOAD 1
      ADD= 1
      STORE 1
      JUMP read
stop: HALT
```

Složenost programa za RAM

- Interesuje nas vremenska i prostorna složenost programa za RAM.
- Treba da preciziramo šta je jedan korak tj. cijena jednog koraka i šta nam je jedna prostorna (memorijska) jedinica.
- U slučaju kriterijuma **uniformne cjene** (najčešće se koristi) izvršavanje jednoe naredbe programa za RAM je jedan korak, a jedan registar predstavlja jednu memorijsku jedinicu.
- Za dati ulaz broj koraka koji koristi RAM program predstavlja njegovu vremensku složenost na tom ulazu, a broj registara koji koristi RAM program predstavlja njegovu prostornu složenost na tom ulazu.
- Kao i ranije složenost za ulaz veličine n u najgorem slučaju predstavlja maksimum složenosti po svim ulazima veličine n.
- Složenost (vremenska/prostorna) za ulaz veličine n u prosječnom (očekivanom) slučaju predstavlja matematičko očekivanje (obično aritmetičku srednu) složenosti po svim ulazima veličine n.

- Komanda "ADD 100" kod uniformne cijene zahtjeva jedan korak. Međutim ako sabirci imaju po nekoliko miliona bita onda očekujemo da će to sabiranje znatno duže trajati od sabiranja recimo $2+3$.
- Za precizniju ocjenu složenosti kad su operandi (argumenti) veliki koristimo kriterijum **logaritamske cijene**.
- Cjelobrojnu logaritamsku funkciju $\ell : \mathbb{Z} \to \mathbb{Z}$ definišemo sa:
  $$\ell(i) = \begin{cases} \lfloor \log_2 |i| \rfloor + 1 & , \quad i \neq 0 \\ 1 & , \quad i = 0 \end{cases} .$$
  Napomena. Za binarni zapis broja i treba $\lfloor \log_2 |i| \rfloor + 1$ bita.
- Ako program koristi registar r_i onda kao negovu prostornu cijenu po logaritamskom kriterijumu uzimamo:
  – $\ell(c(i))$ za trenutni sadržaj $c(i)$,
  – $\ell(\max_k c_k(i))$, gdje je $c_k(i)$ u stvari $c(i)$ nakon izvršenja k-te naredbe.
- Logaritamsku cijenu za operand a, u oznaci $t(a)$ definišemo sa:
  $t(=i) = \ell(i)$,
  $t(i) = \ell(i) + \ell(c(i))$ i
  $t(*i) = \ell(i) + \ell(c(i)) + \ell(c(c(i)))$.

Logaritamska cijena komandi:

Naredba	**Cijena**
1. LOAD a	$t(a)$
2. STORE i	$\ell(\mathrm{ac}) + \ell(i)$
STORE $*i$	$\ell(\mathrm{ac}) + \ell(i) + \ell(c(i))$
3. ADD a	$\ell(\mathrm{ac}) + t(a)$
4. SUB a	$\ell(\mathrm{ac}) + t(a)$
5. MULT a	$\ell(\mathrm{ac}) + t(a)$
6. DIV a	$\ell(\mathrm{ac}) + t(a)$
7. READ i	$\ell(\mathrm{input}) + \ell(i)$
READ $*i$	$\ell(\mathrm{input}) + \ell(i) + \ell(c(i))$
8. WRITE a	$t(a)$
9. JUMP b	1
10. JGTZ b	$\ell(\mathrm{ac})$
11. JZERO b	$\ell(\mathrm{ac})$
12. HALT	1

- Složenost za zadatak 1.

	u. cijena	l. cijena
v. složenost	a_{11}	a_{12}
p. složenost	a_{21}	a_{22}

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} O(n) & O(n^2 \log_2 n) \\ O(1) & O(n \log_2 n) \end{bmatrix} \quad \text{kad} \quad n \to \infty$$

  Računamo $n \cdot n^k$ sa logoritamskom cjenom $\ell(n) + \ell(n^k) = O(k \log n)$. Najviše prostora zahtjeva n^n sa cjenom $\ell(n^n) = O(n \log n)$.
- Neka učitavamo n brojeva $x_1, \ldots, x_n$. Kao veličinu ulaza možemo uzeti broj n ali je za l. cjenu ispravnije uzeti $\ell(x_1) + \cdots + \ell(x_n)$.

RASP - Random access stored program machine

- Sličan kao RAM stim što postoje dvije glavne razlike:

1. Program se upisuje u memoriju - pa može biti mijenjan u toku izvršavanja;
2. Nema indirektnog adresiranja.

Kodovi naredbi za RASP

Naredba	Kod operacije		
LOAD i	1	DIV i	10
LOAD =i	2	DIV =i	11
STORE i	3	READ i	12
ADD i	4	WRITE i	13
ADD =i	5	WRITE =i	14
SUB i	6	JUMP i	15
SUB =i	7	JGTZ i	16
MULT i	8	JZERO i	17
MULT =i	9	HALT	18

- Svaka komanda se smješta u dva uzastopna registra u prvi registar smještamo kod a u drugi operand ili labelu.
- Stanje je određeno sa memorijskom mapom c i LC.
- Složenost se definiše slično kao za RAM stim što u slučaju logoritamskog kriterijuma dodajemo i $\ell(LC)$.
- Pr. ADD $= i$; $\quad AC \leftarrow AC + i$; $\quad \ell(LC) + \ell(AC) + \ell(i)$

Odnos RAM i RASP

- **Teorema 1.** U slučaju uniformnog ili logoritamskog kriterijuma za svaki program P_1 za RAM vremenske složenosti $T_1(n)$ postoji konstanta $k = k(P_1)$ i postoji njemu ekvivalentan program P_2 za RASP vremenske složenosti $T_2(n)$, po istom kriterijumu, tako da je $T_2(n) \leq k \cdot T_1(n)$.
- **Dokaz**. RASP u odnosu na RAM jedino nema komandu indirektnog adresiranja. Tako da zadržavamo komande koje ne koriste indirektno adresiranje a modeliramo komande koje koriste indirektno adresiranje.

U r_1 privremeno smještamo sadržaj AC. Od registra r_2 počinjemo da smještamo kod za P_2 zaključno sa nekim registrom recimo r_s.

Počev od registra r_{s+1} smještamo podatke tako da je $c_{RASP}(i+s) = c_{RAM}(i),\ i = 1, 2, \ldots$

P_2 dobijamo iz P_1 zamjenom odgovarajućih komandi.

Komandu SKOK i menjamo sa: SKOK labela(i)

Komandu KOD =i menjamo sa: KOD =i,

Komandu KOD i menjamo sa: KOD i+s

Komandu KOD $*i$ menjamo sa (nizom od 6 komandi):

	STORE 1	$c(1) \leftarrow AC$ tj. čuva AC
	LOAD $s+i$	AC dobija indirektnu adresu
	ADD $= s$	korekcija adrese
	STORE $l_{KOD}+1$	upisuje pravu adresu za KOD
	LOAD 1	vraća AC
l_{KOD}:	KOD $-$	izvršava naredbu KOD

Tabela: Simulacija naredbe SUB *i od strane mašine RASP:

registar	sadržaj	smisao
100	3	STORE 1 (ostavi ac)
101	1	
102	1	LOAD r + i (donesi indirektnu adresu)
103	r + i	
104	5	ADD =r (adaptacija adrese)
105	r	
106	3	STORE 111 (ostavi adresu)
107	111	
108	1	LOAD 1 (vrati ac)
109	1	
110	6	SUB b (oduzimanje)
111	b	

U slučaju uniformne cjene vidimo da je $T_2(n) \leq 6 \cdot T_1(n)$.
U slučaju logoritamske cjene $T_2(n) \leq (6 + 11 \cdot \ell(s)) \cdot T_1(n)$.

- **Teorema 2.** U slučaju uniformnog ili logoritamskog kriterijuma za svaki program P_2 za RASP vremenske složenosti $T_2(n)$ postoji konstanta $k = k(P_2)$ i postoji njemu ekvivalentan program P_1 za RAM vremenske složenosti $T_1(n)$, po istom kriterijumu, tako da je $T_1(n) \leq k \cdot T_2(n)$.
- **Dokaz**. Napisaćemo program P_1 za RAM (nezavisno od programa P_2) koji će biti interpreter bilo kog programa za RASP.
  Pretpostavka je da je kod programa P_2 učitan u memoriju za RAM.
  P_1 uzima jednu po jednu komandu od P_2 i izvršava je.
  Memorijska mapa je sledeća:

  r_1 služi za pripremanje adrese,

  r_2 čuva RASP-ov LC,

  r_3 čuva RASP-ov AC

  dok je $c_{RAM}(i+3) = c_{RASP}(i)$.

* P_1 se vrti u petlji koja uzme kod sledeće RASP-ove naredbe, zatim ustanovi koja je to od 18 mogućih naredbi (višestruko IF tj. switch ili case naredba) i predaje kontrolu kodu koji izvršava tu naredbu.

Program P_1 :

	STORE 3	$c(3) \leftarrow AC$ tj. čuva AC
	LOAD $= 4$	AC dobija vrijednost 4
	STORE 2	$LC \leftarrow AC$ tj. $LC = 4$
loop:	LOAD $*2$	čitamo KOD sledeće komande
	SUB $= 1$	
	JZERO n_1	ako je KOD=1 skok na n_1 (LOAD i)
	SUB $= 1$	
	JZERO n_2	ako je KOD=2 skok na n_2 (LOAD $= i$)
	SUB $= 1$	
	JZERO n_3	ako je KOD=3 skok na n_3 (STORE i)
	SUB $= 1$	
	JZERO n_4	ako je KOD=4 skok na n_4 (ADD i)
	...	...
	SUB $= 1$	
	JZERO n_{17}	ako je KOD=17 skok na n_{17} (JZERO i)
	HALT	ako je KOD=18 onda se zaustavljamo

Komanda SUB i:

n_6 :	LOAD 2	AC dobija vrijednost LC
	ADD $=1$	$LC \leftarrow LC + 1$
	STORE 2	$LC \leftarrow AC$ tj. LC ukazuje na adresu
	LOAD $*2$	čitamo adresu komande tj. i
	ADD $=3$	izvršimo pomjeraj adrese
	STORE 1	$c(1)$ dobija adresu komande
	LOAD 3	vraćamo sadržaj RASP-ovog AC
	SUB $*1$	izvršimo RASP-ovu operaciju
	STORE 3	$c(3) \leftarrow AC$ tj. čuva RASP-ov AC
	LOAD 2	AC dobija vrijednost LC
	ADD $=1$	$LC \leftarrow LC + 1$
	STORE 2	sada LC ukazuje na sledeću komandu
	JUMP loop	povratak u glavnu petlju

Komanda SUB $= i$:

n_7 :	LOAD 2	AC dobija vrijednost LC
	ADD $= 1$	$LC \leftarrow LC + 1$
	STORE 2	$LC \leftarrow AC$ tj. LC ukazuje na adresu
	LOAD $*2$	čitamo operand komande tj. i
	STORE 1	$c(1)$ dobija operand komande
	LOAD 3	vraćamo sadržaj RASP-ovog AC
	SUB 1	izvršimo RASP-ovu operaciju
	STORE 3	$c(3) \leftarrow AC$ tj. čuva RASP-ov AC
	LOAD 2	AC dobija vrijednost LC
	ADD $= 1$	$LC \leftarrow LC + 1$
	STORE 2	sada LC ukazuje na sledeću komandu
	JUMP loop	povratak u glavnu petlju

Napomena. Prethodne teoreme važe i za prostornu složenost.

Četiri "slaba" modela izračunljivosti

- Razmotramo četri modela za koja možemo reći da nastaju od RAM modela izostavljanjem nekih njegovih mogućnosti.
- Mogućnosti ovih modela su znatno slabije od mogućnosti modela RAM tj. skup izračunljivih funkcija je znatno manji.
- Koraci su jednostavniji pa složenost možemo preciznije da ocjenimo.

- Modeli su:

1. Pravolinijski program
2. Model sa bit operacijama
3. Operacije sa vektorima bita
4. Stabla odlučivanja

Pravolinijski program

- Izostavljamo naredbe skoka kao i naredbe READ, WRITE i HALT. Nema indirektnog adresiranja. Podaci se čuvaju (kao i ulaz i izlas) u konačnom broju registara koji imaju svoja imena.
- LOAD i STORE zamjenujemo sa $\leftarrow$ npr. $c \leftarrow a + b$ zamjenjuje kod

  LOAD a
  ADD b
  STORE c
- Imamo pet komandi:

  $z \leftarrow x + y$,
  $z \leftarrow x - y$,
  $z \leftarrow x \cdot y$,
  $z \leftarrow x/y$ i
  $z \leftarrow k$,
  gdje su x, y, z imena registara (promjenljive) a $k \in \mathbb{Z}$ konstanta.
- Komanda $z \leftarrow x + y$ znači da saberemo sadržaje registara x i y a rezultat smještamo u registar z i sl.

- Primjer programa koji računa $p(x) = a_3x^3 + a_2x^2 + a_1x + a_0$.
- Znamo Hornerovo pravilo $p(x) = \big((a_3x + a_2)x + a_1\big)x + a_0$.
- Program:

  $t \leftarrow a_3 \cdot x$
  $t \leftarrow t + a_2$
  $t \leftarrow t \cdot x$
  $t \leftarrow t + a_1$
  $t \leftarrow t \cdot x$
  $p \leftarrow t + a_0$

- Vremenska složenost $T(n)$= broju naredbi a prostorna $S(n)$= broju promjenljivih.
- Za ovaj model koristimo oznaku $T(n) = O_A(f(n))$ kako bi ukazali da računamo koliko algoritam zahtjeva aritmetičkih operacija.
- $O_A(f(n))$ čitamo $O(f(n))$ aritmetičkih operacija.

Model sa bit operacijama

- Sličan prethodnom modelu samo promjenljive uzimaju vrijednosti iz $\{0,1\}$ a operacije rade sa bitima:

$$z \leftarrow x \wedge y \qquad z \leftarrow x \vee y \qquad z \leftarrow x \oplus y$$
$$z \leftarrow \neg x \qquad z \leftarrow 0 \qquad z \leftarrow 1$$

- Zadatak. Naći zbir dvobitnih brojeva.

Slijedi tekst rješenja:

$$\begin{aligned}
c_0 &\leftarrow a_0 \oplus b_0 \\
u &\leftarrow a_0 \,\&\, b_0 \\
v &\leftarrow u \oplus a_1 \\
c_1 &\leftarrow v \oplus b_1 \\
w &\leftarrow a_1 \vee b_1 \\
x &\leftarrow u \,\&\, w \\
y &\leftarrow a_1 \,\&\, b_1 \\
c_2 &\leftarrow x \vee y
\end{aligned}$$

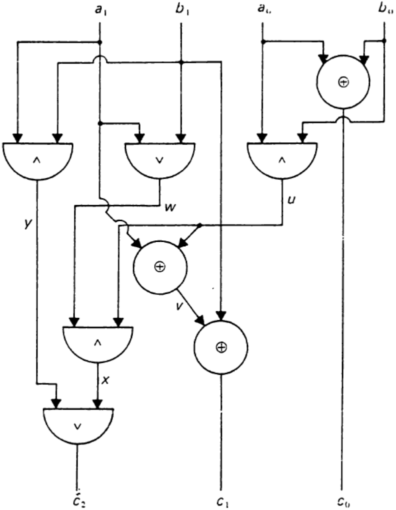
Računa:

$$\boxed{(a_1 a_0)_2 + (b_1 b_0)_2 = (c_2 c_1 c_0)_2}$$

$$\begin{aligned}
c_0 &= a_0 \oplus b_0 \\
c_1 &= (a_0 \,\&\, b_0) \oplus a_1 \oplus b_1 \\
c_2 &= ((a_0 \,\&\, b_0) \,\&\, (a_1 \vee b_1)) \vee (a_1 \,\&\, b_1)
\end{aligned}$$

Slijedi izgled memorije:

a_1	a_0	b_1	b_0	c_2	c_1	c_0	u	v	w	x	y	...



Koristimo oznaku $T(n) = O_B(f(n))$ za $O(f(n))$ bit operacija.

Operacije sa vektorima bita

- U ovom modelu promjenljive su vektori bita potrebne dužine n tj. iz skupa $\{0, 1\}^n$.
- Dužina vektora nije ograničena.
- Ponovo izvodimo logičke operacije samo se one vrše istovremeno na svim bitima vektora.
- Npr. za $x = (x_1, x_2, x_3)$, $y = (y_1, y_2, y_3)$ i $z = (z_1, z_2, z_3)$ operacija $z \leftarrow x \wedge y$ znači da se u jednom koraki izvrši:
  $z_1 \leftarrow x_1 \wedge y_1$, $z_2 \leftarrow x_2 \wedge y_2$ i $z_3 \leftarrow x_3 \wedge y_3$
- Koristimo oznaku $T(n) = O_{BV}(f(n))$ za $O(f(n))$ vektorskih bit operacija.

Stabla odlučivanja

- Kad u algoritmu dominira grananje onda to možemo prikazati pomoću stabla. Vremenska složenost je visina stabla. Oznaka $O_C(f(n))$.
- Zadatak. Urediti tročlani niz a, b, c.