

17.IAM002.AI.Diskretne i kombinatorne metode za računarsku grafiku

Animacija u inženjerstvu

školska 2025/26

Literatura

- [1] Z. Ovcin, Algoritmi i optimizacioni postupci na diskretnim strukturama, osnove i savremeni pristupi, Fakultet tehničkih nauka, Univerzitet u Novom Sadu, 2026
<https://nblok306.ftn.uns.ac.rs/~zoran/A/DiKMzRG/pocetak.pdf>
<https://nblok306.ftn.uns.ac.rs/~zoran/A/DiKMzRG/folije.pdf>
<https://nblok306.ftn.uns.ac.rs/~zoran/A/DiKMzRG/>
<https://nblok306.ftn.uns.ac.rs/~zoran/>
- [2] Z. Ovcin, J. Đokić, Zbirka zadataka za Diskretne i kombinatorne metode za računarsku grafiku, Fakultet tehničkih nauka, Univerzitet u Novom Sadu, 2026

Bodovi i datumi

	Deo 1	Deo 2	Usm.	Σ
MAX	40	40	20	100
MIN	18	18	0	51
Datumi				

Istorija upotrebe mašina za računanje

- Tablice trigonometrijske i logaritamske
 - Sumerske
 - Starogrčke
 - Napier 1614, Briggs 1617
- Antikythera (Ancient Greek), Abacus (Roman Empire)
- Difference engine 0 (Charles Babbage, 1822)
- Floating point, Human computers (Lorentz 1920's)
- Turing's machine, Computability (Alan Turing 1936)
- Universal machine, Z3 (Konrad Zuse 1943)
- Eniac (1946), Edvac (Von Neumann 1949)
- UNIVAC I mainframe computer (Eckert–Mauchly CC, 1951)
- Apple II (1977), Commodore 64 (1982), ZX Spectrum (1982)

- IBM PC (Intel 8088, 1981), Apple Macintosh 128K (Motorola 68k, 1984)

Operativni sistemi

- PCDOS, MSDOS (Microsoft), System Software 1.0 (MAC OS Apple)
- VAX VMS (DEC), Unix (Bell System, AT&T)
- Linux (Linus Torvalds, GNU, Intel 80386, 1991-1994)
- Windows 3.12, Windows 95, Windows XP, Windows 7, 10, 11 (Microsoft)
- Apple macOS (Apple MAC, 2001), iOS (Apple iPhone, 2007)
- Android (Google, 2008)

Internet

- IPv4 (DARPA, 1980), ARPANET (1981), TCP/IP (1982), IPv6 (IETF 1998)
- WWW (CERN, 1989), ISP (1990s), Google(1998), Facebook (2004)

Algoritmi i kompjuterski programi

Šta je algoritam?

Šta je kompjuterski program?

Interpretiranje vs Izvršavanje

Istorija kompjuterskog programiranja

- Da li je kompjuterski program algoritam?
- Da li se svaki kompjuterski program prevodi na mašinski jezik?
- Da li se Java programi prevode na mašinski jezik?
- Da li postoji programski jezik koji se može i kompajlirati i interpretirati?
- Da li su HTML, CSS, Java, JavaScript, TypeScript programski jezici?
- Da li postoje programi čije izvršavanje daje isti rezultat na svim kompjuterima?

Primer programa za kompajliranje

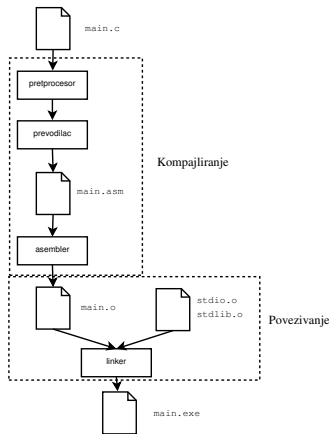
Programiranje

Primer programa za interpreter

```
set.seed(12345);  
n <- 5000;  
s <- numeric(n);  
for(k in 1:n)  
  {s[k] <- ks.test(runif(100), 'punif')$p.value};  
sum(s < .05)/n
```

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
int main()  
{  
  char ime1[]="Petra";  
  char ime2[]="Petar";  
  
  if (strcmp(ime1,ime2)){  
    printf ("%s'_i_'%s'_se_razlikuju.\n", ime1, ime2);  
  }  
  else{  
    printf ("%s'_i_'%s'_se_ne_razlikuju.\n", ime1, ime2);  
  }  
  
  if (strncmp(ime1,ime2,3)){  
    printf ("%s'_i_'%s'_se_razlikuju_do_treceg_slova.\n", ime1, ime2);  
  }  
  else{  
    printf ("%s'_i_'%s'_se_ne_razlikuju_do_treceg_slova.\n", ime1, ime2);  
  }  
  
  return 0;  
}
```

Programski jezici



Proces kompajliranja u C-u

1949: Assembler (Kathleen Booth, University of London)

1957: FORTRAN (John Backus, IBM)

1958: Algol (ETH Zürich, Association for Computing Machinery, IFIP)

1959: COBOL (Dr. Grace Murray Hopper), LISP (John McCarthy, MIT)

1964: BASIC (John G. Kemeny, Thomas E. Kurtz, Dartmouth College)

1970: Pascal (Niklaus Wirth, ETH Zürich, Palo Alto Research Center)

1972: C (Dennis Ritchie, Bell Labs), SQL (IBM)

1978: MATLAB (Cleve Moler, University of New Mexico)

1983: C++ (Bjarne Stroustrup, Bell Labs)

1987: Perl (Larry Wall, System Development Corporation)

1990: Haskell (Functional Programming committee)

1991: Python (Guido Van Rossum, CWI Netherlands), Visual Basic (Microsoft)

1993: R (Ross Ihaka, Robert Gentleman, University of Auckland, New Zealand)

1995: Java (Sun Microsystems), PHP (Rasmus Lerdorf), Ruby (Yukihiro Matsumoto), JavaScript (Netscape, Brendan Eich)

2000: C# (Microsoft)

2003: Scala (École Polytechnique Fédérale de Lausanne), Groovy (Apache)

2009: GO (Google)

2014: Swift (Apple)

Pseudokod

function Ime funkcije

Telo funkcije

...

Komanda za izlazak iz funkcije (obavezna): **return** izraz

...

end function

procedure Ime procedure

...

end procedure

Naredbe za regulisanje toka programa su:

for ... to ... do	while ... do	repeat	if ... then
...
...	else
...
end for	end while	until ...	end if

Naredba **break** izvodi iz kontrolne petlje.

Naredba \leftarrow je direktiva pridruživanja ($:=$).

Osnovni tipovi: numerički, karakter, logički (Bulovski), pokazivač, string, struktura.

Izrazi nad numeričkim i logičkim tipovima koriste matematičke i logičke operacije.

Netačno je nula (0 ili **false**) a tačno (**true**) je sve što je različito od nule, obično jedan (1).

Argumenti se prenose kao u C-u. Niz je adresa na prvi element niza. Tip niza je implicitno poznat.

Podrazumevano je da su promenljive **lokalne**, po potrebi **globalne**.

1: **function** PROGRAM1(A)

2: $n \leftarrow \text{length}(A)$

3: flag \leftarrow true

4: **for** $j \leftarrow 1$ **to** $n - 1$ **do**

5: **if** $A[j] > A[j + 1]$ **then**

6: flag \leftarrow false

7: **end if**

8: **end for**

9: **return** flag

10: **end function**

Šta sa ulaznim nizom A radi algoritam dat levo?

Da li se može ubrzati, a da daje isti rezultat?

Napisati ekvivalentan program bez upotrebe for petlje (koristiti **while** petlju).

```

function PROGRAM1A(A)
   $n \leftarrow \text{length}(A)$ 
  flag  $\leftarrow$  true
  for  $j \leftarrow 1$  to  $n - 1$  do
    if  $A[j] > A[j + 1]$  then
      flag  $\leftarrow$  false
      break
    end if
  end for
  return flag
end function

```

```

function PROGRAM1B(A)
   $n \leftarrow \text{length}(A)$ 
  flag  $\leftarrow$  true
   $j \leftarrow 1$ 
  while flag & ( $j \leq n - 1$ ) do
    if  $A[j] > A[j + 1]$  then
      flag  $\leftarrow$  false
    end if
     $j \leftarrow j + 1$ 
  end while
  return flag
end function

```

```

function PROGRAM1C(A)
   $n \leftarrow \text{length}(A)$ 
  flag  $\leftarrow$  true
   $j \leftarrow 1$ 
  while flag & ( $j \leq n - 1$ ) do
    flag  $\leftarrow A[j] \leq A[j + 1]$ 
     $j \leftarrow j + 1$ 
  end while
  return flag
end function

```

Analiza kompleksnosti

Zadatak je da se na osnovu veličine ulaznih parametara odredi (najbolja) asimptotska oznaka za vreme izvršavanja.

Koriste se specijalne asimptotske oznake koje porede beskonačne veličine.

```

1: function PROGRAM1( $A$ )
2:    $n \leftarrow \text{length}(A)$ 
3:   flag  $\leftarrow$  true
4:   for  $j \leftarrow 1$  to  $n - 1$  do
5:     if  $A[j] > A[j + 1]$  then
6:       flag  $\leftarrow$  false
7:     end if
8:   end for
9:   return flag
10: end function

```

Analizirati vreme izvršavanja programa levo u zavisnosti od vremena izvršavanja linija koda $c_2, c_3, c_4, c_5, c_6, c_9$ i dužine ulaznog niza n .

vreme izvršavanja	c_2	c_3	c_4	c_5	c_6	c_9
broj izvršavanja	1	1	n	$n - 1$	m	1

Best case	$T(n) = c_2 + c_3 + nc_4 + (n - 1)c_5 + 0c_6 + c_9 = \Theta(n)$
Worst case	$T(n) = c_2 + c_3 + nc_4 + (n - 1)c_5 + (n - 1)c_6 + c_9 = \Theta(n)$
Average case	$T(n) = c_2 + c_3 + nc_4 + (n - 1)c_5 + \lfloor \frac{n-1}{2} \rfloor c_6 + c_9 = \Theta(n)$

Asimptotske oznake

$$\Theta(g) = \{f \mid (\exists c_1 > 0)(\exists c_2 > 0)(\exists n_0 \in \mathbb{N})(\forall n)(n \geq n_0) \Rightarrow (0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n))\}$$

$$O(g) = \{f \mid (\exists c > 0)(\exists n_0 \in \mathbb{N})(\forall n)(n \geq n_0) \Rightarrow (0 \leq f(n) \leq c g(n))\}$$

$$\Omega(g) = \{f \mid (\exists c > 0)(\exists n_0 \in \mathbb{N})(\forall n)(n \geq n_0) \Rightarrow (0 \leq c g(n) \leq f(n))\}$$

$$o(g) = \{f \mid (\forall c > 0)(\exists n_0 \in \mathbb{N})(\forall n)(n \geq n_0) \Rightarrow (0 \leq f(n) \leq c g(n))\}$$

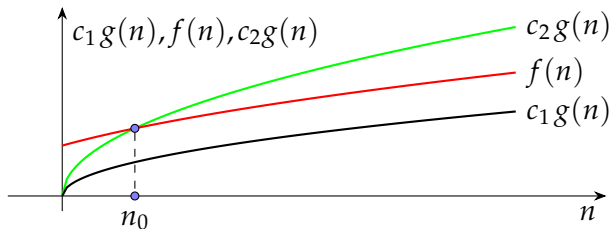
$f \in \Theta(g)$, pišemo $f = \Theta(g)$, funkcija f se ponaša kao $\Theta(g)$ (kao veliko teta od g).

$$f = \Omega(g) \iff f \geq g$$

$$f = O(g) \iff f \leq g$$

$$f = \Theta(g) \iff f = g$$

$$f = o(g) \iff f < g$$



$\frac{2}{3}n^2 - 2n = \Theta(n^2)$, jer počev od $n_0 = 4$, za $c_1 := \frac{1}{6}$ i $c_2 = \frac{2}{3}$, važi $0 < c_1 n^2 \leq \frac{2}{3}n^2 - 2n \leq c_2 n^2$.

Za nizove f i g koji teže ka beskonačnosti ako je $f = o(g)$ pišemo $f \prec g$.

Ako je $0 < \alpha < \beta$, i $1 < a < b$, onda važi:

$$\dots \prec \log \log n \prec \log n \prec n^\alpha \prec n^\beta \prec a^n \prec b^n \prec n! \prec n^n \prec \dots$$

Rekurzija

```
function FACTORIAL(n)  
  if  $n \leq 1$  then  
    return 1  
  else  
    return  $n \cdot \text{FACTORIAL}(n - 1)$   
  end if  
end function
```

	slobodna memorija
kraj rekurzije	FACTORIAL(1) = 1
	FACTORIAL(2) = 2 * 1 = 2
poziv funkcije	FACTORIAL(3) = 3 * 2 = 6
	Sistemska RAM

```
function FACTORIAL(n)  
   $f \leftarrow 1$   
  for  $k \leftarrow 2$  to  $n$  do  
     $f \leftarrow f \cdot k$   
  end for  
  return  $f$   
end function
```

slobodna memorija
2, 3
1 * 2 * 3 = 6
Sistemska RAM

k promenljiva
 f promenljiva

Stirlingova aproksimacija $n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$

Fibonačijevi brojevi

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

```
function FIBONACCI( $n$ )
```

```
  if  $n \leq 1$  then
```

```
    return  $n$ 
```

```
  else
```

```
    return FIBONACCI( $n - 1$ ) + FIBONACCI( $n - 2$ )
```

```
  end if
```

```
end function
```

$$\text{FIBONACCI}(n) = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2} \right)^n$$

Broj poziva funkcije FIBONACCI je manji od broja čvorova punog binarnog drveta sa n nivoa, koji iznosi $2^0 + 2^1 + 2^2 + \dots + 2^{n-1} = 2^n - 1$.

Vreme izvršavanja rekurzivne funkcije (levo) je $T(n) = O(2^n)$.

Vreme izvršavanja iterativne verzije (desno) $T'(n) = \Theta(n)$, što je lošije od računanja pomoću formule koje je $T''(n) = \Theta(1)$.

```
function FIBONACCI( $n$ )
```

```
  if  $n \leq 1$  then
```

```
    return  $n$ 
```

```
  else
```

```
     $f_0 \leftarrow 0$ 
```

```
     $f_1 \leftarrow 1$ 
```

```
    for  $k \leftarrow 2$  to  $n$  do
```

```
       $f \leftarrow f_0 + f_1$ 
```

```
       $f_0 \leftarrow f_1$ 
```

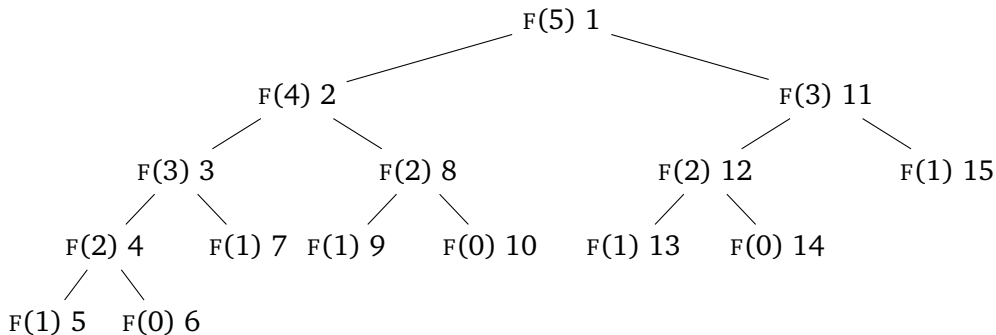
```
       $f_1 \leftarrow f$ 
```

```
    end for
```

```
  end if
```

```
  return  $f$ 
```

```
end function
```



Na kompjuteru je izmereno vreme izvršavanja rekurzivne funkcije FIBONACCI za $n = 40$ i za $n = 42$: $T(40) = 5.5492s$ i $T(42) = 14.5920s$. Ako je vreme izvršavanja približno jednako $T(n) = c \cdot q^n$, izračunati q i c .

$$q = \sqrt{T(42)/T(40)} = 1.6216, \quad c = T(40)/q^{40} = 2.2210 \cdot 10^{-8}$$

Koristeći dobijene vrednosti naći vreme da se izvrši FIBONACCI (50) i FIBONACCI (100).

$$T(50) = 697.6741s = 11\text{min}38s$$

$$T(100) = 2.1916e + 13s = 694.96 \text{ godina}$$

$$T(n) = \Theta \left(\left(\frac{1+\sqrt{5}}{2} \right)^n \right) \approx \Theta(1.618^n).$$

BUBBLE SORT

```
1: procedure PROGRAM2(A)
2:    $n \leftarrow \text{length}(A)$ 
3:   repeat
4:      $flag \leftarrow \text{true}$ 
5:     for  $j \leftarrow 1$  to  $n - 1$  do
6:       if  $A[j] > A[j + 1]$  then
7:          $\text{swap}(A, j, j + 1)$ 
8:          $flag \leftarrow \text{false}$ 
9:       end if
10:    end for
11:  until  $flag$ 
12: end procedure
```

Analizirati PROGRAM2 ($[5, 2, 4, 6, 1, 3]$)

broj prolaza	1	2	3	4	5
menja se:	5 sa 2	5 sa 1	4 sa 1	2 sa 1	
	5 sa 4	5 sa 3	4 sa 3		
	6 sa 1				
	6 sa 3				

```

procedure PROGRAM2A(A)
  n ← length(A)
  repeat
    flag ← true
    for j ← 1 to n − 1 do
      if A[j] > A[j + 1] then
        swap(A, j, j + 1)
        flag ← false
      end if
    end for
    n ← n − 1
  until flag
end procedure

```

```

procedure PROGRAM2B(A)
  n ← length(A)
  repeat
    newn ← 0
    for j ← 1 to n − 1 do
      if A[j] > A[j + 1] then
        swap(A, j, j + 1)
        newn ← j
      end if
    end for
    n ← newn
  until n = 0
end procedure

```

algoritam	br. poređenja	br. zamena
PROGRAM2	25	9
PROGRAM2A	15	9
PROGRAM2B	14	9

INSERTION SORT

```
1: procedure INSERTION SORT( $A$ )
2:   for  $i \leftarrow 2$  to length( $A$ ) do                                 $\triangleright n$ 
3:      $key \leftarrow A[i]$                                               $\triangleright n - 1$ 
4:      $j \leftarrow i - 1$                                               $\triangleright n - 1$ 
5:     while  $j > 0 \ \& \ A[j] > key$  do                                $\triangleright \sum_{i=2}^n t_i$ 
6:        $A[j + 1] \leftarrow A[j]$                                         $\triangleright \sum_{i=2}^n (t_i - 1)$ 
7:        $j \leftarrow j - 1$                                             $\triangleright \sum_{i=2}^n (t_i - 1)$ 
8:     end while
9:      $A[j + 1] \leftarrow key$                                           $\triangleright n - 1$ 
10:  end for
11: end procedure
```

$$T(n) = c_2n + (c_3 + c_4)(n - 1) + c_5 \sum_2^n t_i + (c_6 + c_7) \sum_2^n (t_i - 1) + c_9(n - 1),$$

Best case $t_i = 1$, $T_B(n) = \Theta(n)$

$$T_B(n) = c_2n + (c_3 + c_4)(n - 1) + c_5(n - 1) + c_9(n - 1)$$

Worst case $t_i = i$, $T_W(n) = \Theta(n^2)$

$$T_W(n) = c_2n + (c_3 + c_4)(n - 1) + c_5(n(n + 1)/2 - 1) + (c_6 + c_7)n(n - 1)/2 + c_9(n - 1)$$

SELECTION SORT

```
1: procedure SELECTION SORT( $A$ )
2:    $n \leftarrow \text{length}(A)$ 
3:   for  $i \leftarrow 1$  to  $n$  do
4:      $i_{\min} \leftarrow i$ 
5:     for  $j \leftarrow i + 1$  to  $n$  do
6:       if  $A[j] < A[i_{\min}]$  then
7:          $i_{\min} \leftarrow j$ 
8:       end if
9:     end for
10:    if  $i \neq i_{\min}$  then
11:       $\text{swap}(A, i, i_{\min})$ 
12:    end if
13:  end for
14: end procedure
```

$\triangleright 1$
 $\triangleright n + 1$
 $\triangleright n$
 $\triangleright \sum_{i=1}^n t_i$
 $\triangleright \sum_{i=1}^n (t_i - 1)$
 $\triangleright \sum_{i=1}^n (s_i - 1)$
 $\triangleright n$
 $\triangleright r$

Linija algoritma $i \mapsto c_i$ vreme
Neka je $T(n)$ vreme izvršavanja
SELECTION SORT algoritma za niz
dužine n .

$$\begin{aligned} t_i &= n - i + 1, s_i \leq t_i, r \leq n \Rightarrow \\ T(n) &= c_2 + c_3(n + 1) + \\ &\quad + c_4 n + c_5 \left(\frac{n^2}{2} + \frac{n}{2} \right) + \\ &\quad + c_6 \left(\frac{n^2}{2} - \frac{n}{2} \right) + \\ &\quad + c_7 \sum_{i=1}^n (s_i - 1) + \\ &\quad + c_{10} n + c_{11} r \end{aligned}$$

Sledi da je $T(n) = \Theta(n^2)$.

$$T(n) = c_2 + c_3(n + 1) + c_4 n + c_5 \sum_{i=1}^n t_i + c_6 \sum_{i=1}^n (t_i - 1) + c_7 \sum_{i=1}^n (s_i - 1) + c_{10} n + c_{11} r$$

MERGE SORT

procedure SORT(A, p, r)

▷ Procedura koja se rekurzivno poziva

if $p < r$ **then**

$q \leftarrow \lfloor (p + r) / 2 \rfloor$

SORT(A, p, q)

SORT($A, q + 1, r$)

MERGE(A, p, q, r)

end if

end procedure

procedure MERGE SORT(A)

▷ Glavna procedura koju poziva korisnik

$n \leftarrow \text{length}(A)$

SORT($A, 1, n$)

end procedure

Na primer, $p = 1, q = 4, r = 8$ za

MERGE ([1,3,4,5,2,6,7,8], 1, 4, 8)

procedure MERGE(A, p, q, r)

for $k \leftarrow p$ **to** q **do**

$L[k - p + 1] \leftarrow A[k]$

end for

$L[q - p + 2] \leftarrow \infty$

for $k \leftarrow q + 1$ **to** r **do**

$R[k - q] \leftarrow A[k]$

end for

$R[r - q + 1] \leftarrow \infty$

$i \leftarrow 1; j \leftarrow 1$

for $k \leftarrow p$ **to** r **do**

if $L[i] \leq R[j]$ **then**

$A[k] \leftarrow L[i]; i \leftarrow i + 1$

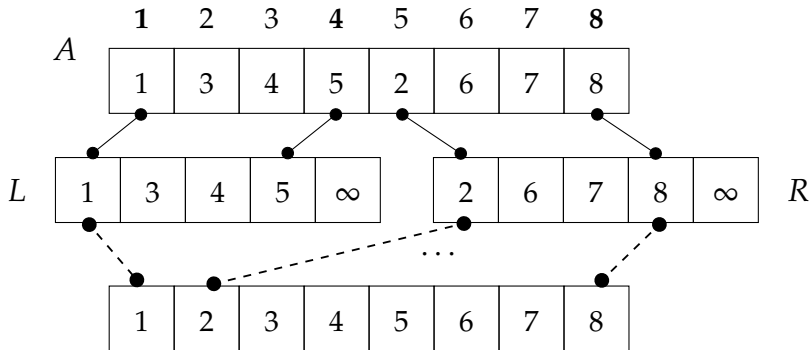
else

$A[k] \leftarrow R[j]; j \leftarrow j + 1$

end if

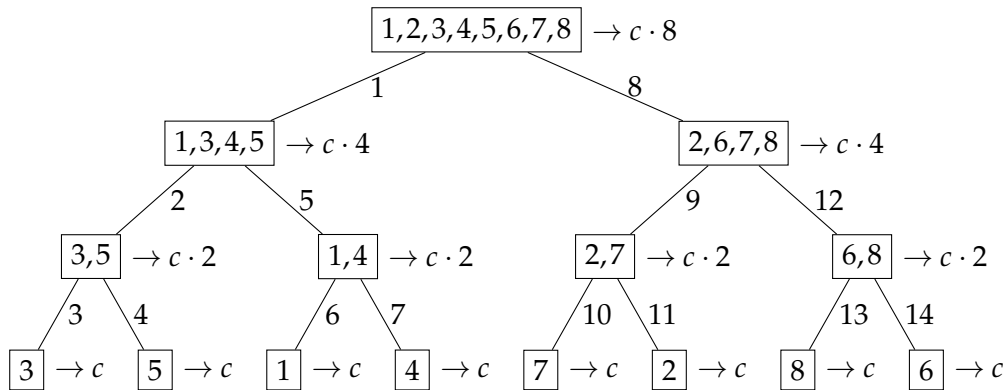
end for

end procedure



Spajanje sortiranih podnizova od 1 do 4 i od 5 do 8: $\text{MERGE}([1,3,4,5,2,6,7,8],1,4,8)$

Vreme izvršavanja procedure $\text{MERGE}(A,p,q,r)$, $T_M(n) = \Theta(n)$, gde je $n = r - p + 1$.



MERGE SORT ([3,5,1,4,7,2,8,6]) za niz sa $n = 2^3 = 8$ elemenata

Neka je $n = 2^k$, $k \in \mathbb{N}$ broj elemenata niza A .

Vreme izvršavanja procedure SORT (A, p, r) je $T_S(m) = \Theta(m) \approx c \cdot m$, gde je $m = r - p + 1$

Na svakom nivou rekurzije imamo ukupan zbir vremena izvršavanja SORT procedure $8 \cdot c$, odnosno, u opštem slučaju $2^k \cdot c = c \cdot n$. Pošto imamo $k + 1 = \log_2 n + 1$ nivoa, vreme izvršavanja MERGE SORT (A) za niz A dužine $n = 2^k$ je $T(n) = (\log_2 n + 1) \cdot c \cdot n = c \cdot n \cdot \log_2 n + c \cdot n = \Theta(n \log n)$.

Koliko puta će biti pozvana procedura MERGE za sortiranje niza dužine $n = 2^k$?

Rešenje: $n - 1$ puta.

Napisati hronološki pozive MERGE koji se izvrše pri pozivanju
MERGE SORT ([5,2,4,6,1,3]).

Rešenje:

MERGE([5,2,4,6,1,3],1,1,2)

MERGE([2,5,4,6,1,3],1,2,3)

MERGE([2,4,5,6,1,3],4,4,5)

MERGE([2,4,5,1,6,3],4,5,6)

MERGE([2,4,5,1,3,6],1,3,6)

QUICK SORT

PARTITION grupiše elemente odabranog podniza (od p do r) premeštanjem i vraća redni broj q elementa koji je na svom mestu po redosledu, tako da ispred njega budu manji ili jednaki od njega, iza njega veći od njega (\star).

procedure SORT(A, p, r)

▷ Rekurzivna

if $p < r$ **then**

$q \leftarrow$ PARTITION(A, p, r)

 SORT($A, p, q - 1$)

 SORT($A, q + 1, r$)

end if

end procedure

procedure QUICK SORT(A)

▷ Za korisnika

$n \leftarrow$ length (A)

 SORT($A, 1, n$)

end procedure

function PARTITION(A, p, r)

▷ U skladu sa (\star)

$x \leftarrow A[r]$

$i \leftarrow p - 1$

for $j \leftarrow p$ **to** $r - 1$ **do**

if $A[j] \leq x$ **then**

$i \leftarrow i + 1$

 exchange(A, i, j)

end if

end for

 exchange($A, i + 1, r$)

return $i + 1$

end function

Poređenje algoritama za sortiranje

U sledećoj tabeli je data asimptotska vrednost vremena izvršavanja za ulaz veličine n za sledeće algoritme: BUBBLE SORT (B), INSERTION SORT (I), SELECTION SORT (S), MERGE SORT (M), QUICK SORT (Q).

Oznake su: Best case: B, Average case: A i Worst case: W. Takođe je dat podatak o redu veličine dodatnog memorijskog prostora potrebnog za sortiranje (M) i podatak o stabilnosti posmatranog algoritma (S).

	B	A	W	M	S
B	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	DA
I	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	DA
S	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	NE
M	$\Theta(n \ln n)$	$\Theta(n \ln n)$	$\Theta(n \ln n)$	$\Theta(n)$	DA
Q	$\Theta(n \ln n)$	$\Theta(n \ln n)$	$\Theta(n^2)$	$\Theta(\ln n)$	NE

```

int main()
{
    int i,j,n;
    int maxn=100000; int A[maxn];  int B[maxn];
    srand(time(NULL));
    for (i=0;i<maxn;i++){
        A[i] = rand();
        B[i] = A[i];
    }
    n = 1;
    printf("\n_____n_|_____B_____I_____S_____M_____Q\n");
    printf("-----+-----+-----+-----+-----+-----\n");
    for(j=1;j<6;j++){
        n = n*10;
        // Bubble sort
        clock_t start = clock(); bubble_sort(A,n); clock_t end = clock();
        float B_sec = (float)(end - start) / CLOCKS_PER_SEC;
        memcpy(A,B,n*sizeof(int));
        // Insertion sort
        // Selection sort
        // Merge sort
        // Quick sort
        // Scores
        printf("_%7d_|_%8.3f_|_%8.3f_|_%8.3f_|_%8.3f_|_%8.3f_\n",n,B_sec,I_sec,S_sec,M_sec,Q_sec);
    }
    return 0;
}

```

Algoritme za sortiranje smo testirali na nizu random brojeva obima $10^1, 10^2, 10^3, 10^4, 10^5$. Mereno je vreme potrebno da se niz sortira algoritmom kodiranim u C-u na kompjuteru.

n	B	I	S	M	Q
10	7.30E-05	6.08E-05	6.10E-05	0.00034	0.0003
100	0.00021	6.10E-05	0.0001	0.00235	0.0008
1000	0.01873	0.00417	0.00625	0.0233	0.0067
10000	1.92922	0.35455	0.60198	0.23819	0.0669
100000	192.796	35.4994	60.2424	2.38437	0.6754

Poređenje stringova

Nizovi karaktera su stringovi. C kompajleri na kraj niza karaktera pod navodnicima stavljaju NUL karakter (NUL = `\0`)

```
#include <stdio.h>
#include <stdlib.h>
#define maxn 30

// Ovaj program koristiti za testiranje .
// Izlaz: ab = -97, ac = 17, bc = 17.

int zcmp(char*, char*, size_t);

void main (){
    char a[] = "Petr";
    char b[] = "Petra";
    char c[] = "Petar";
    printf("ab_ = %d_\n", zcmp(a,b,maxn));
    printf("ac_ = %d_\n", zcmp(a,c,maxn));
    printf("bc_ = %d_\n", zcmp(b,c,maxn));
}
```

Redosled je "Petar", "Petr", "Petra" (c, a, b).

Broj koji će funkcija `zcmp` vratiti je razlika rednih brojeva u ASCII kodu prvih karaktera koji se razlikuju na adresama `s1` i `s2`. Nula se vraća ako su `n` karaktera jednaki.

Ova funkcija je implementirana u standardnoj biblioteci `string.h` kao `strncmp`.

```
int zcmp(char *s1, char *s2, size_t n){
    unsigned int i = 0;
    int y = 0;
    while(i < n &&
        !(y=s1[i]-s2[i]) && s1[i]){i++;};
    return y;
}
```

Napisati u programskom jeziku C funkciju `zstrcmp` koja upoređuje NULL terminated stringove `str1` i `str2`.

Funkcija treba da vrati:

- ako prvi string prethodi drugom,
- + ako drugi string prethodi prvom,
- 0 ako su stringovi jednaki.

```
int zstrcmp(char *str1, char *str2)
{
    int i=0;
    int r;
    if (str1 == str2) return 0;
    while (!(r=str1[i]-str2[i])
           && str1[i] && str2[i])
        i++;
    return r;
}
```

Lokalizacija

ASCII \subseteq UTF-8

Format informacije za lokalizaciju je:

language[_territory][.codeset][@modifier].

Na primer: `sr_RS.UTF-8@latin` je oznaka za srpski jezik, u Republici Srbiji, kodiran u UTF-8 u latiničnoj varijanti.

```
#include <stdio.h>
#include <locale.h>           // Biblioteka za lokalizaciju
#include <string.h>

void main (){
    char a[] = "njujork";
    char b[] = "nujork";
    char new_locale[] = "sr_RS";
    setlocale (LC_ALL, new_locale); // Srpski jezik

    printf (" strcoll (njujork,nujork) = %d\n", strcoll(a,b));
    printf (" strcmp(njujork,nujork) = %d\n", strcmp(a,b));
}
```

Prethodni C program komandom `setlocale` podešava svih šest informacija po pravilima srpskog jezika. Ako se to ne uradi, podrazumevana lokalizacija je C standard.

Izlaz ovog programa će biti:

```
strcoll(njujork,nujork) = 1  
strcmp(njujork,nujork) = -11
```

To znači da po ASCII standardu slovo j prethodi slovu u, pa stoga i "njujork" prethodi "nujork" (`strcmp = -11`). Po srpskoj latinici slovo n prethodi slovu (digrafu) nj pa "nujork" prethodi "njujork" (`strcoll = 1`).

LC_COLLATE	Definiše redosled poređenja stringova
LC_CTYPE	Definiše klasifikaciju karaktera, velika - mala slova
LC_MESSAGES	Definiše kako se kaže DA i NE
LC_MONETARY	Definiše pravila za pisanje cena i oznaku valute
LC_NUMERIC	Definiše pravila za pisanje brojeva
LC_TIME	Pravila i simboli za datum i vreme

Tabela 1: Šest informacija lokalizacije

Matrice - računanje determinante dovođenjem na gornje trougaonu

```
function DET( $A, n$ )  
   $znak \leftarrow 1$   
  for  $i \leftarrow 1$  to  $n - 1$  do  
     $pm \leftarrow \text{PIVOT}(A, n, i)$   
    if  $\neg pm$  then  
      return 0.0  
    end if  
     $znak \leftarrow znak * pm$   
    for  $k \leftarrow i + 1$  to  $n$  do  
       $\alpha \leftarrow A[k, i] / A[i, i]$   
       $A[k, i] \leftarrow 0.0$   
      for  $j \leftarrow i + 1$  to  $n$  do  
         $A[k, j] \leftarrow A[k, j] - \alpha A[i, j]$   
      end for  
    end for  
  end for  
  if  $A[n, n] = 0$  then  
    return 0.0  
  end if  
   $d \leftarrow znak * A[1, 1]$   
  for  $i \leftarrow 2$  to  $n$  do  
     $d \leftarrow d * A[i, i]$   
  end for  
  return  $d$   
end function
```

```
function PIVOT( $A, n, m$ )  
   $i_1 \leftarrow m; j_1 \leftarrow m; pm \leftarrow 1$   
  for  $i \leftarrow m$  to  $n$  do  
    for  $j \leftarrow m$  to  $n$  do  
      if  $|A[i, j]| > |A[i_1, j_1]|$  then  
         $i_1 \leftarrow i; j_1 \leftarrow j$   
      end if  
    end for  
  end for  
  if  $A[i_1, j_1] = 0$  then  
    return 0  
  end if  
  if  $i_1 \neq m$  then  
     $pm \leftarrow pm * (-1)$   
    for  $j \leftarrow m$  to  $n$  do  
       $\text{swap}(A[i, j], A[i_1, j])$   
    end for  
  end if  
  if  $j_1 \neq m$  then  
     $pm \leftarrow pm * (-1)$   
    for  $i \leftarrow 1$  to  $n$  do  
       $\text{swap}(A[i, j], A[i, j_1])$   
    end for  
  end if  
  return  $pm$   
end function
```

Broj operacija za računanje determinante reda n

i	$SAB(i)$	$MNO(i)$
1	$(n-1)(n-1)$	$(n-1)(n-1) + n - 1$
2	$(n-2)(n-2)$	$(n-2)(n-2) + n - 2$
\vdots	\ddots	\dots
$n-2$	$2 \cdot 2$	$2 \cdot 2 + 2$
$n-1$	1	$1 + 1$
linija 22	0	$n - 1$
Σ	$\Sigma SAB(i)$	$\Sigma MNO(i)$

$$\sum_{k=1}^n k^2 = \frac{1}{6}n(n+1)(2n+1), \quad \sum_{k=1}^n k = \frac{1}{2}n(n+1)$$

$$\Sigma SAB(i) = \frac{1}{6}(n-1)n(2n+1) = \Theta(n^3)$$

$$\Sigma MNO(i) = \Sigma SAB(i) + \frac{1}{2}(n-1)n + n - 1 = \Theta(n^3)$$

Računanje determinante po definiciji

$$\begin{vmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{vmatrix} = \sum_{\substack{\text{po svim perm.} \\ (i_1, i_2, \dots, i_n)}} (-1)^{\sigma(i_1, i_2, \dots, i_n)} a_{1, i_1} a_{2, i_2} \cdots a_{n, i_n}$$

gde je $\sigma(i_1, i_2, \dots, i_n)$ broj inverzija permutacije (i_1, i_2, \dots, i_n) .

$$\sum SAB(i) = n! - 1 \quad \sum MNO(i) = n!(n - 1)$$

Ako jedno sabiranje traje $3.13E-9s$, množenje $3.75E-9s$. Koliko vremena treba da se saberu i pomnože elementi matrice 100×100 preko definicije?

$$\sum SAB(i) \times 3.13 \times 10^{-9} + \sum MNO(i) \times 3.75 \times 10^{-9}.$$

	vreme
Gausove eliminacije	0.0023s
Po definiciji	$3.49 \times 10^{151}s =$ 1.11×10^{144} godina

Kreiranje C biblioteke za apstraktni tip podataka matrice

matrice.h

```
void addmat(double *, double *, double *, int, int); // sabiranje (I,I,O,I,I)
void multmat(double *, double *, double *, int, int, int); // mnozenje (I,I,O,I,I,I)
void multscal(double, double *, double *, int, int); // mnozenje skalarom (I,I,O,I,I)
void transpose(double*, double *, int, int); // transponovanje (I,O,I,I)
int inverse(double*, double *, int); // inverzna (I/O,O,I)
double det(double*, int); // determinanta (I/O,I)
int printmatrix(double*, int, int); // stampanje (I,I,I)
```

Napisati program u C-u koji rešava matričnu jednačinu $A + BX = C$, gde su

$$A = \begin{bmatrix} 1 & 2 & -3 \\ 4 & -7 & -16 \\ -7 & -18 & -16 \end{bmatrix}, B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 7 & 6 \\ 7 & 8 & -16 \end{bmatrix}, C = \begin{bmatrix} -1 & 3 & 4 \\ 2 & 2 & 10 \\ 16 & 20 & 21 \end{bmatrix}.$$

$$X = B^{-1}(C - A)$$

$X = B^{-1}(C - A)$ u C-u računamo pomoću sledećeg programa

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "matrice.h"
#define epsilon 1e-12

int main()
{
    int nr,n;

    n = 3;
    double A[]={1,2,-3,4,-7,-16,-7,-18,-16};
    double B[]={1,2,3,4,7,6,7,8,-16};
    double C[]={-1,3,4,2,2,10,16,20,21};
    double *minusA=malloc(n*n*sizeof(double));
    double *CminusA=malloc(n*n*sizeof(double));
```

```
double *Binv=malloc(n*n*sizeof(double));
double *X=malloc(n*n*sizeof(double));
```

```
if((nr=inverse(B,Binv,n)){
    printf("\nNe postoji B^(-1),rang(B) je %d.",n-nr);
}
else{
    multscal(-1,A,minusA,n,n);
    addmat(C,minusA,CminusA,n,n);
    multmat(Binv,CminusA,X,n,n,n);
    printmatrix(X,n,n);
}
return 0;
}
```

```
++          +-
|   1.00    2.00    3.00  |
|   0.00    1.00    2.00  |
|  -1.00   -1.00   -0.00  |
++          +-

```

Process returned 0 (0x0) execution time : 0.016

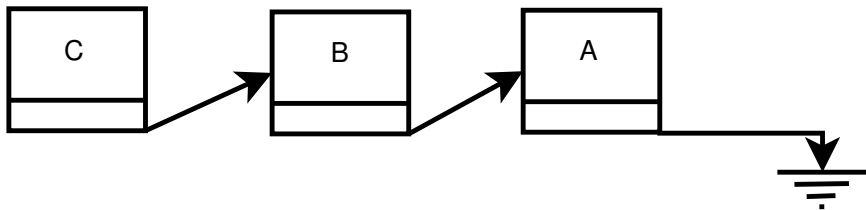
Press any key to continue.

Povezane liste

```
typedef char listdata ;  
typedef struct _node node;  
  
struct _node {  
    listdata data;  
    node *next;  
};
```

Stack

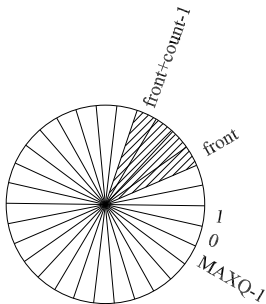
```
typedef char listdata ;  
typedef struct _node node;  
typedef node *stack;  
  
void makenull(stack *);  
int isempty(stack);  
int push(stack *, listdata );  
listdata pop(stack *);  
listdata top(stack);  
void clear(stack *);  
int ismember(stack, listdata *);  
void printstack(stack);
```



Queue

Pomoću niza:

```
struct _queue {  
    listdata data[MAXQ];  
    int front;  
    int count;  
};
```



Pomoću povezanih listi:

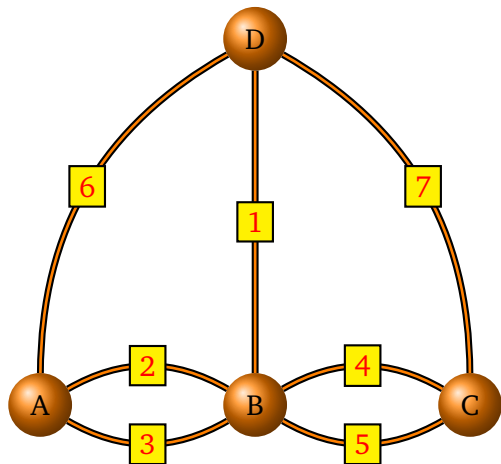
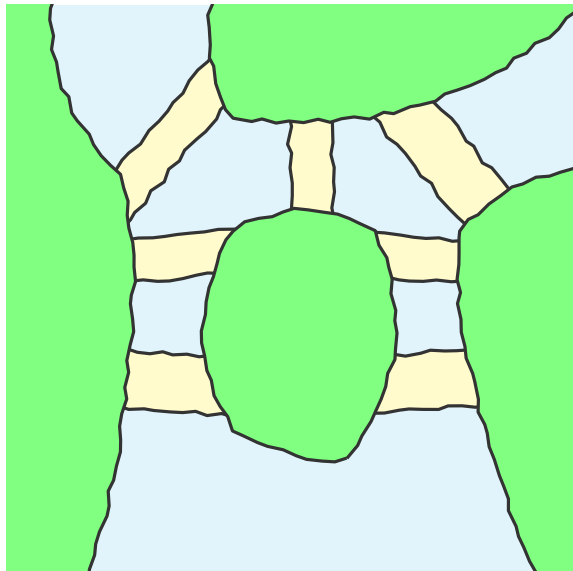
```
struct _node  
{  
    listdata data;  
    node *next;  
};  
struct _queue  
{  
    node *front;  
    node **rear;  
};
```

```
typedef char listdata ;  
typedef struct _node node;  
typedef struct _queue *queue;  
  
void makenullQ(queue *);  
int isemptyQ(queue);
```

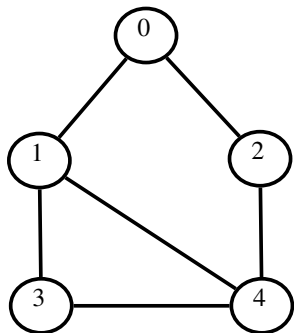
```
int enqueue(queue, listdata);  
listdata dequeue(queue);  
listdata front(queue);  
void clearQ(queue);  
int ismemberQ(queue, listdata *);  
void printqueue(queue);
```

Grafovi

Problem 7 mostova Königsberga (Kalinjingrada), Leonhard Euler 1736



Strukture podataka za grafove



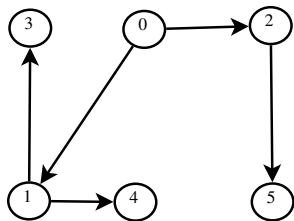
Graf

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

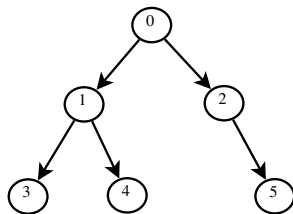
Matrica susedstva

u	Adj(u)
0	1, 2
1	0, 3, 4
2	0, 4
3	1, 4
4	1, 2, 3

Tabela listi susedstva



Usmereni graf



Isti graf kao korensko drvo

u	Adj(u)
0	1, 2
1	3, 4
2	5
3	
4	
5	

Tabela listi susedstva

Grafovi kao apstraktni tip podataka

U programskom jeziku C koristimo Tabele listi susedstva kao osnovni tip podataka nad kojim ćemo izvršavati algoritme

```
typedef struct _node gnode;
typedef gnode *grana;
typedef int nextnode;

struct _node
{
    nextnode data;
    gnode *next;
};
```

Treba nam procedura koja:

- dodaja granu na kraj liste susedstva: *enqueue_list*,
- oduzima prvu granu iz liste susedstva: *dequeue_list*,
- oslobađa dinamički alociranu memoriju jedne liste susedstva: *clear_list*,
- oslobađa dinamički alociranu memoriju grafa: *clear_graf*,
- štampa tabelu liste susedstva: *print_graf*,
- štampa matricu susedstva: *print_graf_matrix*.

```

void enqueue_list(grana **grana_tail_p,
                 nextnode d) {
    grana grana_new = malloc(sizeof(grana));

    grana_new -> data = d;
    grana_new -> next = NULL;
    **grana_tail_p = grana_new;
    *grana_tail_p = &(grana_new->next);
}

nextnode dequeue_list(grana *grana_head)
{
    grana grana_temp = *grana_head;
    nextnode d=-1;

    if(grana_head)
    {
        d = grana_temp -> data;
        *grana_head = grana_temp -> next;
    }
}

```

```

        free(grana_temp);
    }
    return d;
}

void clear_list (grana *adjp)
{
    while(*adjp)
        dequeue_list(adjp);
}

void clear_graph(grana graf[])
{
    int i;
    for(i=0;i<max_cvorova;i++){
        clear_list (&(graf[i]));
    }
}

```

BFS

- BFS polazi od izvora: čvor s i prolazi kroz sve čvorove koji su povezani sa s .
- BFS nalazi d , (najkraću) udaljenost od s za svaki čvor, $d = \infty$ ako nije povezan.
- BFS nalazi π , prethodnika u najkraćem putu, dajući "breadth first tree".
- BFS koristi atribut boja ($color$) $\in \{ \text{WHITE, GRAY, BLACK} \}$ za svaki čvor.
- BFS redom otkriva sve čvorove koji su od s udaljeni za k , a potom za $k + 1$.

function BFS(G, s)

▷ U nizu π vraća prethodnika za čvor

▷ U nizu d vraća udaljenost od čvora s

for each $u \in V[G] \setminus \{s\}$ **do**

$d[u] \leftarrow \infty$

$\pi[u] \leftarrow \text{NULL}$

$color[u] \leftarrow \text{WHITE}$

end for

MAKENULLQ(Q)

$d[s] \leftarrow 0$

$\pi[s] \leftarrow \text{NULL}$

$color[s] \leftarrow \text{GRAY}$

ENQUEUE(Q, s)

while $\neg \text{ISEMPTYQ}(Q)$ **do**

$u \leftarrow \text{DEQUEUE}(Q)$

for each $v \in \text{Adj}(u)$ **do**

if $color[v] = \text{WHITE}$ **then**

$color[v] \leftarrow \text{GRAY}$

$d[v] \leftarrow d[u] + 1$

$\pi[v] \leftarrow u$

ENQUEUE(Q, v)

end if

end for

$color[u] \leftarrow \text{BLACK}$

end while

return d, π

end function

Procedura u programskom jeziku C koja za graf smešten u Adjacency list vraća stepen svih čvorova i funkcija koja vraća diametar grafa. Stepen je broj suseda, diametar je najveće najkraće rastojanje između dva čvora u grafu.

```
1 void stepen(grana G[], int n, int s[])
2 {
3     int i;
4
5     grana gr;
6
7     for(i=0;i<n;i++){
8         gr = G[i];
9         s[i] = 0;
10        while(gr){
11            (s[i])++;
12            gr = gr->next;
13        }
14    }
15
16 }
```

```
int diametar(grana G[], int n)
{
    int diam = 0;
    int d[max_cvorova], p[max_cvorova];
    int i,j;

    for(i=0;i<n;i++){
        bfs(G,n,i,d,p);
        for(j=0;j<n;j++){
            if(d[j]>diam){
                diam = d[j];
            }
        }
    }
    return diam;
}
```

Asimptotski red složenosti procedure stepen

Standardno sa c_k obeležavamo vreme potrebno da se izvrši linija k . Neka graf ima n čvorova i m grana. Za neusmereni graf, kod koga se grana (i, j) smešta i u listu susedstva čvora v_i i listu susedstva čvora v_j , vreme izvršavanja procedure stepen je $T(n, m) =$

$$= c_3 + c_5 + (n + 1) \cdot c_7 + n \cdot c_8 + n \cdot c_9 + \sum_{i=0}^{n-1} (|Adj(v_i)| + 1) c_{10} + \sum_{i=0}^{n-1} |Adj(v_i)| (c_{11} + c_{12}),$$

gde je $|Adj(v_i)|$ broj suseda čvora v_i . Kako je $\sum_{i=0}^{n-1} |Adj(v_i)| = 2m$, sledi

$$T(n, m) = n(c_7 + c_8 + c_9 + c_{10}) + 2m(c_{10} + c_{11} + c_{12}) + c_3 + c_5 + c_7 = \Theta(n + m).$$

Standardno se obeležava broj čvorova grafa V , a broj grana E .

Složenost algoritma stepen je $\Theta(V + E)$.

DFS

Algoritam za pretraživanje usmerenog grafa "u dubinu" (depth¹ first search = DFS).

- DFS prolazi kroz sve čvorove u koje nije posetio.
- DFS rekurzivno nastavlja kroz sve grane čiji su susedi v neistraženi.
- kad DFS istraži sve čvorove koji su susedi od v , backtrack² postupkom se vraća u čvor iz kojeg je stigao u v .
- kad DFS istraži sve grane iz polaznog čvora, nastavlja sa neistraženim čvorovima.
- kad DFS dođe od čvora u do čvora v , upisuje da je predecessor³ od v čvor u .
- DFS koristi atribut boja ($color$) $\in \{ \text{WHITE, GRAY, BLACK} \}$ za svaki čvor. U početku su svi WHITE. Kad se otkrije, čvor postaje GRAY, kad završi sa njim, postaje BLACK.

¹*depth* = dubina, EN

²*backtrack* = vratiti se istim putem, EN

³*predecessor* = prethodnik, EN

- DFS za svaki čvor u zapisuje *timestamps*⁴ $d[u]$ i $f[u]$ ($d[u] < f[u]$) momenta kad je otkrio u (*discovery*) i kad je završio sa u (*finish*).
- Vremenske oznake *timestamps* su iz skupa $\{1, 2, \dots, 2 \cdot |V|\}$.
- Čvor u je WHITE od momenta 1 do $d[u]$, GRAY od $d[u]$ do $f[u]$ i BLACK posle $f[u]$.

Tipovi grana:

T - *tree edge*, grana drveta iz DFS šume, pronalazi novi čvor drveta, (\rightarrow WHITE)

F - *forward edge*, (u, v) je grana unapred ako pronalazi čvor koji već pripada drvetu, t.j. ako je v potomak od u . (\rightarrow BLACK)

B - *back edge*, (u, v) je grana unazad ako je u je potomak od v . (\rightarrow GRAY)

C - *cross edge*, poprečne grane, su sve ostale grane. (\rightarrow BLACK!!)

⁴*timestamps* = vremenske oznake, EN

- U pseudokodu koji sledi promenljiva *time* je globalna promenljiva. Radi jednostavnosti, za DFS-VISIT globalne promenljive su i G (graf), d, f , kao i π .
- Rezultat primene algoritma zavisi od redosleda kojim su numerisani čvorovi i redosleda kojim su čvorovi uneti u *Adj* liste.

```

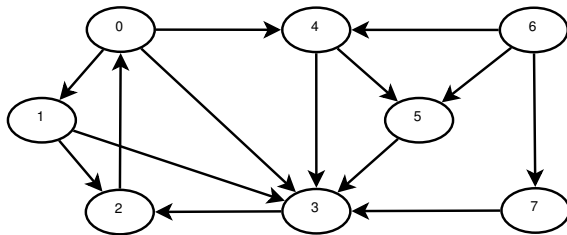
function DFS( $G$ )
  for each  $u \in V[G]$  do
     $color[u] \leftarrow$  WHITE
     $\pi[u] \leftarrow$  NULL
  end for
   $time \leftarrow 0$ 
  for each  $u \in V[G]$  do
    if  $color[u] =$  WHITE then
      DFS-VISIT( $u$ )
    end if
  end for
  return  $d, f, \pi$ 
end function

```

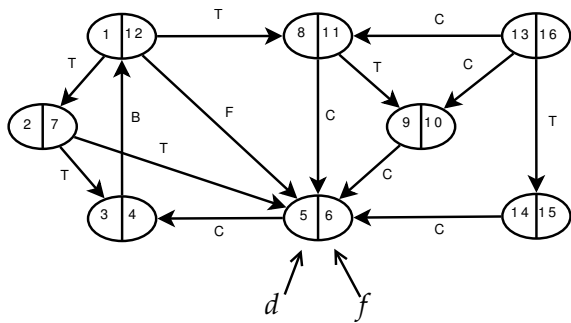
```

procedure DFS-VISIT( $u$ )
   $color[u] \leftarrow$  GRAY
   $time \leftarrow time + 1$ 
   $d[u] \leftarrow time$ 
  for each  $v \in Adj(u)$  do
    if  $color[v] =$  WHITE then
       $\pi[v] \leftarrow u$ 
      DFS-VISIT( $v$ )
    end if
  end for
   $color[u] \leftarrow$  BLACK
   $time \leftarrow time + 1$ 
   $f[u] \leftarrow time$ 
end procedure

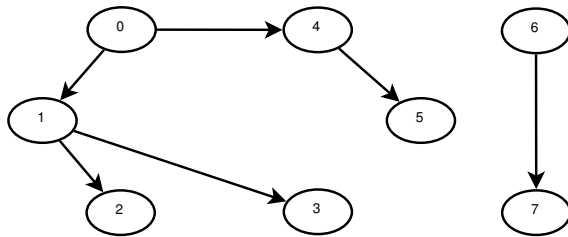
```



Zadati graf



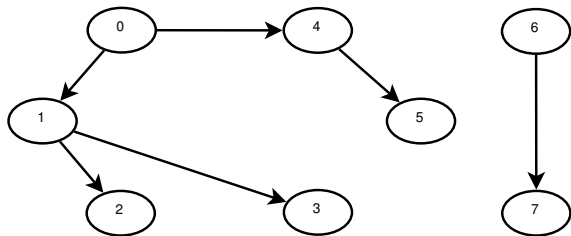
Primena DFS



Šuma DFS

Tabela zagrada

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	()				
1		()									
2			()												
3					()										
4								()					
5									()						
6													()
7														()	



Šuma DFS

Složenost BFS i DFS

Označavamo broj čvorova grafa V i broj grana grafa E .

BFS

Priprema praznih listi koje se vrše za sve čvorove traje $\Theta(V)$, priprema za petlju $\Theta(1)$.

Pozivanje operacije ENQUEUE i DEQUEUE traje $\Theta(1)$. Pošto svaki povezan čvor prođe kroz queue, ukupno se za te operacije potroši $O(V)$ vremena.

Elementi listi susedstva se u algoritmu obrade najviše jednom, kad se čvor skida sa queue. Za njihovu obradu treba $O(E)$ vremena, jer je njihov broj $\Theta(E)$. Sledi $T_{BFS} = O(V + E)$.

DFS

Sama procedura DFS bez DFS-VISIT se izvršava za svaki čvor jednom, stoga traje $\Theta(V)$.

Procedura DFS-VISIT se za svaki čvor v poziva tačno jednom. Unutar nje je petlja koja se izvršava $|Adj(v)|$ puta. Kako je $\sum_{v \in V} |Adj(v)| = \Theta(E)$, sledi $T_{DFS} = \Theta(V + E)$.

Minimalno pokrivaјуće drvo

Na grafu $G = (V, E)$ je data težinska funkcije $w : E \rightarrow \mathbb{R}$. MST = *minimum spanning tree* je pokrivaјуće drvo za koje je zbir težina grana minimalan.

Kruskalov algoritam (koristi strukturu podataka grafa grana)

```
function KRUSKAL( $G, w$ )
```

```
   $A \leftarrow \emptyset$ 
```

```
  for each  $v \in V[G]$  do
```

```
    MAKE-SET( $v$ )
```

▷ za svaki element skup koji ga sadrži

```
  end for
```

```
  sort( $E, w$ )
```

▷ sortiraj grane iz E neopadajuće po w

```
  for each  $(u, v) \in E[G]$  do
```

▷ redom, neopadajuće po w

```
    if FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ ) then
```

```
       $A \leftarrow A \cup \{(u, v)\}$ 
```

▷ dodaj granu

```
      UNION( $u, v$ )
```

▷ spoji skupove

```
    end if
```

```
  end for
```

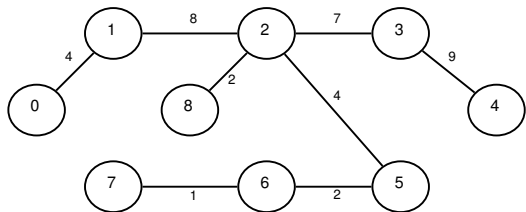
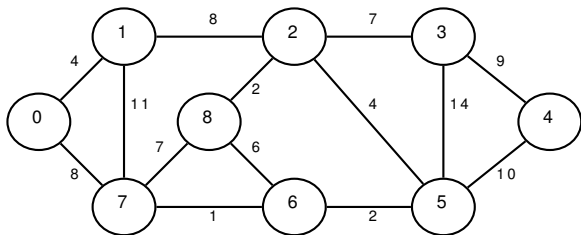
```
  return  $A$ 
```

```
end function
```

Procedura MAKE-SET(v) pravi skup koji sadrži samo element v .

Procedura FIND-SET(v) nalazi skup u kojem je sadržan element v . Taj skup sadrži sve čvorove koji su do tad otkriveni i povezani su sa v dotad formiranim delom pokrivajućeg drveta.

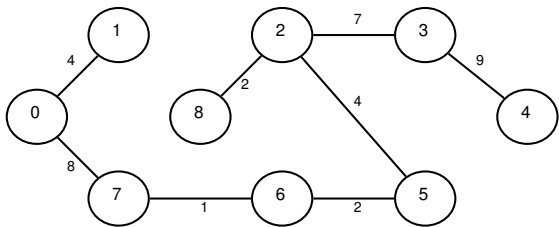
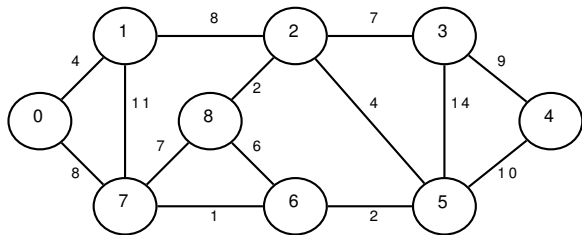
Procedura UNION(u, v) spaja skupove čvorova povezanih sa u i v , jer kad se grana (u, v) doda u pokrivajuće drvo, čvorovi iz tih skupova postaju povezani.



	1	2	3	4	5	6	7	8	
u	6	5	2	2	0	2	1	3	Σ
v	7	6	8	5	1	3	2	4	
w	1	2	2	4	4	7	8	9	

Primov algoritam

```
function PRIM( $G, w, r$ )  
  for each  $u \in V[G]$  do  
     $key[u] \leftarrow \infty; \pi[u] \leftarrow \text{NULL}$   
  end for  
   $key[r] \leftarrow 0$   
   $Q \leftarrow \text{PQ\_BUILD}(V[G], key)$  ▷ lista svih čvorova postaje priority queue  
  while  $\neg \text{PQ\_ISEMPTY}(Q)$  do  
     $u \leftarrow \text{PQ\_EXTRACT\_MIN}(Q, key)$  ▷ isto kao DEQUEUE sa najmanjim  $key$   
    for each  $v \in \text{Adj}(u)$  do  
      if  $\text{PQ\_ISMEMBER}(v, Q) \& (w(u, v) < key[v])$  then  
         $\pi[v] \leftarrow u$  ▷ ako budemo odabrali  $v$ , prethodnik je  $u$ ,  
         $key[v] \leftarrow w(u, v)$  ▷ onda će grana  $w(u, v)$  ući u MST  
      end if  
    end for  
  end while  
   $A \leftarrow \emptyset$   
  for each  $u \in V[G] \setminus \{r\}$  do  
     $A \leftarrow A \cup (\pi[u], u)$  ▷ u listi prethodnika implicitno imamo MST  
  end for  
  return  $A$   
end function
```

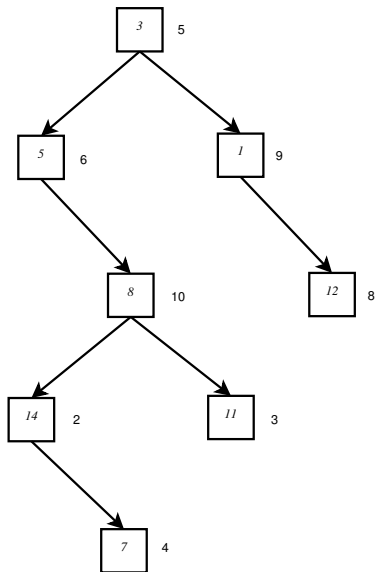


	1	2	3	4	5	6	7	8	
<i>u</i>	0	0	7	6	5	2	2	3	
<i>v</i>	1	7	6	5	2	8	3	4	Σ
<i>w</i>	4	8	1	2	4	2	7	9	37

Binarna drva (korenska)

Rekonstruisati binarno drvo dato u LC-RC reprezentaciji sa korenom na adresi 5.

I	K	LC	RC
1	9	7	-
2	14	-	4
3	11	-	-
4	7	-	-
<u>5</u>	3	6	9
6	5	-	10
7	4	6	1
8	12	-	-
9	1	-	8
10	8	2	3



Napisati rekurzivnu proceduru koja ispisuje elemente drveta iz prethodnog zadatka u in-fiksnom redosledu i rekurzivnu proceduru koja dodaje parent polje svim čvorovima drveta.

```
#include <stdio.h>
#include <stdlib.h>
int key[] = {0, 9,14,11, 7, 3, 5, 4,12, 1, 8};
int LC[] = {0, 7,-1,-1,-1, 6,-1, 6,-1,-1, 2};
int RC[] = {0,-1, 4,-1,-1, 9,10, 1,-1, 8, 3};
int parent[]={0,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};
void infix_print(int root) {
    if(root>-1){
        infix_print(LC[root]);
        printf("%3d",key[root]);
        infix_print(RC[root]);
    }
}
void find_parent(int root, int p) {
    if(root>-1){
        parent[root] = p;
        find_parent(LC[root],root);
        find_parent(RC[root],root);
    }
}
```

```
}
}
int main() {
    int i, root = 5;
    printf(" Infix_print:_\n");
    infix_print(root);
    printf("\n");
    printf("Parent:\n");
    for(i=1;i<11;i++)
        printf("%3d",i);
    printf("\n");
    find_parent(5,-1);
    for(i=1;i<11;i++)
        printf("%3d", parent[i]);
    printf("\n");
    return 0;
}
```

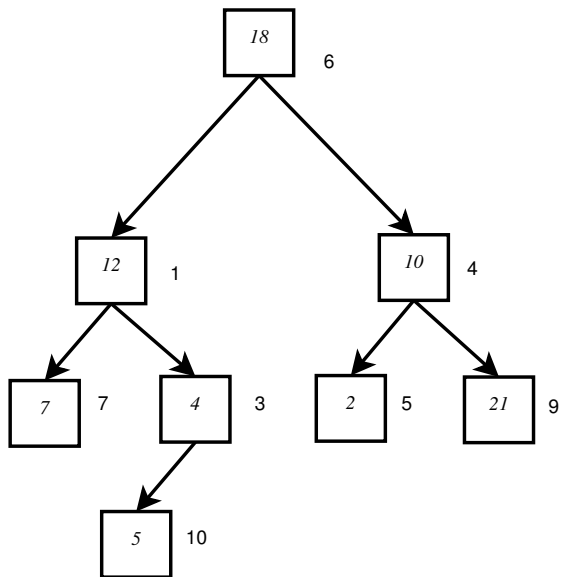
Infix print:

5, 14, 7, 8, 11, 3, 1, 12,

Parent:

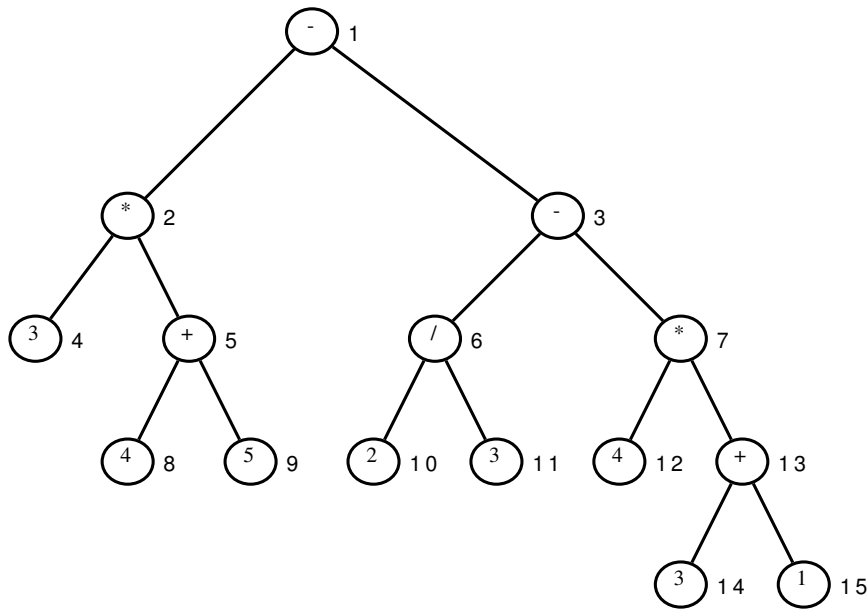
1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
-1, 10, 10, 2, -1, 5, -1, 9, 5, 6,

Dati tabelu LC-RC reprezentacije grafa sa slike.



I	K	LC	RC
1	12	7	3
2			
3	4	10	-
4	10	5	9
5	2	-	-
<u>6</u>	18	1	4
7	7	-	-
8			
9	21	-	-
10	5	-	-

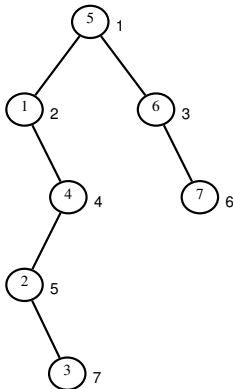
Nacrtati drvo terma izraza $3 \cdot (4 + 5) - (2/3 - 4 \cdot (3 + 1))$ i dati LC-RC reprezentaciju.



I	K	LC	RC
<u>1</u>	-	2	3
2	*	4	5
3	-	6	7
4	3	-	-
5	+	8	9
6	/	10	11
7	*	12	13
8	4	-	-
9	5	-	-
10	2	-	-
11	3	-	-
12	4	-	-
13	+	14	15
14	3	-	-
15	1	-	-

Za ulazni niz [5,1,6,4,2,7,3] kreirati binarno drvo tako da se redom elementi ubacuju:

- ako je manji od korena u levo poddrvo
- inače u desno poddrvo.



```
#include <stdlib.h>
#define max 10
int ulaz[]={5,1,6,4,2,7,3};
int key[max];
int LC[max];
int RC[max];
int iskorisceno = -1;
int ubaci(int koren, int broj) {
    if (koren>-1){
        if (broj<key[koren])
            LC[koren]=ubaci(LC[koren],broj);
        else
            RC[koren]=ubaci(RC[koren],broj);
        return koren;
    }
    else{
        iskorisceno++;
        key[iskorisceno] = broj;
        LC[iskorisceno] = -1;
        RC[iskorisceno] = -1;
        return iskorisceno;
    }
}
int main() {
    int i, koren = ubaci(-1,ulaz[0]);
    for(i=1;i<7;i++) koren = ubaci(koren,ulaz[i]);
    infix_print(koren);
    return 0;}
```

Problem angažovanja

Naći angažovanja radnika $1, \dots, n$ na poslove $1, \dots, n$ (1 radnik - jedan posao), ako su u matrici $C = [c_{i,j}]_{n \times n}$ data vremena $c_{i,j}$ koja su potrebna da radnik i obavi posao j .

Dodelićemo vrednost $x_{i,j} = 1$ ako se radnik i angažuje na posao j , $x_{i,j} = 0$ inače.

$$\zeta = \sum_{i=1}^n \sum_{j=1}^n c_{i,j} x_{i,j} \rightarrow \min$$

$$0 \leq x_{i,j} \leq 1, i, j \in \{1, 2, \dots, n\}, x_{i,j} \in \mathbb{Z},$$

$$\sum_{i=1}^n x_{i,j} = 1, j \in \{1, 2, \dots, n\},$$

$$\sum_{j=1}^n x_{i,j} = 1, i \in \{1, 2, \dots, n\}.$$

Može se rešiti kao *problem linearnog programiranja*. Celobrojni. *Mixed integer linear programming (MILP)*, EN.

Najčešće rešavamo Mađarskom metodom. *Hungarian method*, EN.

Problem trgovačkog putnika

Neka je dat kompletan graf sa n čvorova i nad njegovim granama funkcija težina matricom $C = [c_{i,j}]_{n \times n}$. Težine $c_{i,j}$ predstavljaju dužinu ili cenu putovanja od mesta i do mesta j .

Problem trgovačkog putnika je zadatak nalaženja Hamiltonove konture za koju su težine grana minimalne. *Travelling salesman problem (TSP), EN.*

Dodelićemo vrednost $x_{i,j} = 1$ ako se radnik i angažuje na posao j , $x_{i,j} = 0$ inače.

$$\zeta = \sum_{i=1}^n \sum_{j=1}^n c_{i,j} x_{i,j} \rightarrow \min$$

$$0 \leq x_{i,j} \leq 1, i, j \in \{1, 2, \dots, n\}, x_{i,j} \in \mathbb{Z},$$

$$\sum_{i=1}^n x_{i,j} = 1, j \in \{1, 2, \dots, n\},$$

$$\sum_{j=1}^n x_{i,j} = 1, i \in \{1, 2, \dots, n\}.$$

Zabrana podkontura