

Zadaci za vežbe

Zoran Ovcin¹

2026 - 2027

¹Katedra za matematiku, Fakultet tehničkih nauka, Univerzitet u Novom Sadu

Srećno sa radom

Ovo je zanimljiv i koristan predmet, puno ćete naučiti.

Zoran Ovcin

Novi Sad, 2026. godina

Sadržaj

Uvod	1
1 Vežbe 1 i 2	3
1.1 Osnove C programiranja	3
1.2 Napredni zadaci	8
2 Vežbe 3	11
2.1 Programiranje u C-u	11
2.2 Pretraživanje	16
2.3 Polinomi	18
3 Vežbe 4	21
3.1 Sortiranje	21
4 Vežbe 5	23
4.1 Matrice	23
5 Vežbe 6	27
5.1 Stack i Queue	27
6 Vežbe 7	31

6.1 Implementacija Stack i Queue	31
7 Vežbe 8	39
7.1 Grafovi	39
8 Vežbe 9	45
8.1 BFS i DFS	45
9 Vežbe 10	55
9.1 Minimalno pokrivajuće drvo (MST)	55
10 Vežbe 11	57
10.1 Drva i binarna drva	57
11 Vežbe 12	65
11.1 Problem angažovanja	65
11.2 Problem trgovačkog putnika (TSP)	66

Uvod

Za praćenje gradiva ovog kursa potrebno je osnovno poznavanje programiranja u C-u. Takođe, potreban je pristup nekom programskom okruženju (**IDE**¹) na kojem će se moći vežbati programiranje u C-u i isprobavati zadaci.

Kompajler koji sam koristio je MinGW koji sam instalirao u Program Files direktorijum. Potom sam „c:\Program Files\w64devkit\bin" direktorijum (folder) stavio u putanju (path).

Moja preporuka za IDE je VS Code sa ekstenzijama za C / C++ programiranje.

¹Integrated Development Environment - Integrisano razvojno okruženje

Glava 1

Vežbe 1 i 2

1.1 Osnove C programiranja

Za početak treba napraviti Hello World! program, učitati u VS Code, kompajlirati i izvršiti da se vidi da radi:

hello.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     printf("Hello_World!\n");
7     return 0;
8 }
```

Kako se u VS Code učitava, kompajlira i izvršava program.

Kako se dobijeni mašinski program može preneti na drugi kompjuter i izvršiti iz komandnog prozora.

Potom zdravo program:

zdravo.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     char ime[30];
7     printf("Vaše_ime_je:_");
8     scanf("%s", ime);
9     printf("Zdravo_%.s.\n", ime);
10    return 0;
11 }
```

Potom naučiti osnovne tipove podataka i prenošenje pri pozivu funkcija. Prenosjenje može biti po vrednosti (samo osnovni tipovi) ili po adresi.

Šta daje kao izlaz ovaj program?

swap1.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void swap(int a, int b)
5 {
6     int temp;
7     temp = a; a = b; b = temp;
8 }
9
10 void main()
11 {
12     int a = 2, b = 3;
13     printf("a_=%3d,b_=%3d\n",a,b);
14     swap(a, b );
15     printf("a_=%3d,b_=%3d\n",a,b);
16 }
17
18 // a =  2, b =  3
```

Zašto se vrednosti promenljivih a i b nisu zamenile?

Da li je problem u imenima promenljivih? Da li sledeći program radi zamenu vrednosti promenljivama a i b?

swap2.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void swap(int x, int y)
5 {
6     int temp;
7     temp = x; x = y; y = temp;
8 }
9
10 void main()
11 {
12     int a = 2, b = 3;
13     printf("a=_%3d,_b=_%3d\n",a,b);
14     swap(a, b );
15     printf("a=_%3d,_b=_%3d\n",a,b);
16 }
```

Šta daje kao izlaz ovaj program?

swap3.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int a = 2, b = 3;
5
6 void swap(int x, int y)
7 {
8     int temp;
9     temp = a; a = b; b = temp;
10 }
11
12 void main()
13 {
14     printf("a=_%3d,_b=_%3d\n",a,b);
15     swap(a, b );
16     printf("a=_%3d,_b=_%3d\n",a,b);
17 }
18
19 // a = 3, b = 2
```

Da li je upotreba globalnih promenljivih preporučljiva u programiranju?

Kako se koristi memorija pri upotrebi ovog programa i šta se dobija kao izlaz?

swap.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void swap(int *a, int *b)
5 {
6     int temp;
7     temp = *a; *a = *b; *b = temp;
8 }
9
10 void main()
11 {
12     int a = 2, b = 3;
13     printf("a=_%3d,_b=_%3d\n",a,b);
14     swap(&a, &b);
15     printf("a=_%3d,_b=_%3d\n",a,b);
16 }
17
18 // a =  3, b =  2
```

Napisati i objasniti kako radi program za nalaženje NZD Euklidovim algoritmom.

nzd.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int nzd(int a, int b)
5 {
6     int c;
7     c = a % b;
8     while(c) {
9         a = b;
10        b = c;
11        c = a % b;
12    }
13    return b;
14 }
15
16 int main()
17 {
18     int a, b, c;
19     a = 858;
20     b = 420;
21     c = nzd(a,b);
22     printf("nzd(%d,%d)=_%d\n",a,b,c);
23     return 0;
24 }
```

Napisati program za rešavanje kvadratne jednačine i testirati ga.

kvadrjed.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #define epsilon 1e-14
5
6 // gcc kvadrjed.c -o kvadrjed -lm
7
8 int main()
9 {
10     double a,b,c,d,x1,x2;
11
12     a = 1.0;
13     b = 2.0;
14     c = 2.0;
15     d = b*b-4*a*c;
16
17     if(d>epsilon){
18         x1 = -b/2/a - sqrt(d)/2/a;
19         x2 = -b/2/a + sqrt(d)/2/a;
20         printf("x1=%5.2f,x2=%5.2f\n",x1,x2);
21     }
22     else
23     if(d < -epsilon){
24         printf("x1=%5.2f+_i_*_%5.2f,x2=%5.2f+_i_*_%5.2f\n",
25             -b/2/a,-sqrt(-d)/2/a, -b/2/a,sqrt(-d)/2/a);
26     } else
27         printf("x1=%5.2f,x2=%5.2f\n",-b/2/a,-b/2/a);
28
29     return 0;
30 }
```

Objasniti upotrebu epsilon konstatnte za floating point brojeve.

1.2 Napredni zadaci

Napisati program za nalaženje drugog po veličini elementa u nizu brojeva.

drugipovel.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() // trazi drugi po veličini element i njegov indeks
5  {
6      int A[]={ 12,5,4,1,9,2,3}; // Radi ako su elementi niza različiti!!!
7      // Dodatni zadatak – modifikovati da funkcionise kad ima istih!
8      int Prvi, Drugi, ind1, ind2;
9      int duzinaNiza = sizeof(A) / sizeof(A[0]);
10     if (A[0]<A[1])
11     {
12         Prvi=A[1];
13         Drugi=A[0];
14         ind2=0;
15         ind1=1;
16     }
17     else
18     {
19         Prvi=A[0];
20         Drugi=A[1];
21         ind2=1;
22         ind1=0;
23     }
24
25     // Sad poredimo inicijalne vrednosti Prvog i Drugog
26     // sa ostalim članovima niza
27     for(int i=2;i<duzinaNiza;i++)
28     {
29         if (A[i]>Prvi)
30         {
31             Drugi=Prvi;
32             ind2=ind1;
33             Prvi=A[i];
34             ind1=i;
35         }
36         else if (A[i]<Prvi && (A[i]>Drugi))
37         {
38             Drugi=A[i];
39             ind2=i;
40         }
41     }
42
43     printf("Drugi_element_je_na_%d.poziciji_u_nizu_i_jednak_je_%d\n",
44           ind2+1,Drugi);
45     return 0;
46 }
```

Napisati u pseudo kodu program koji proverava da li je niz sortiran.

```

function PROGRAM1A(A)
  n ← length(A)
  flag ← true
  for j ← 1 to n-1 do
    if A[j] > A[j+1] then
      flag ← false
      break
    end if
  end for
  return flag
end function

```

```

function PROGRAM1B(A)
  n ← length(A)
  flag ← true
  j ← 1
  while flag & (j ≤ n-1) do
    if A[j] > A[j+1] then
      flag ← false
    end if
    j ← j+1
  end while
  return flag
end function

```

```

function PROGRAM1C(A)
  n ← length(A)
  flag ← true
  j ← 1
  while flag & (j ≤ n-1) do
    flag ← A[j] ≤ A[j+1]
    j ← j+1
  end while
  return flag
end function

```

Napisati deo programa u C-u koji proverava da li je niz integer vrednosti dužine n sortiran.

sortiran.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int A[]={ 12,5,4,1,9,2,3};
5  int n=sizeof(A)/sizeof(A[0]);
6  int jesortiran(int A[], int n)
7  {
8
9      // Ovak kod napisati
10
11
12
13
14
15
16      return flag;
17  }
18
19  int main()
20  {
21      int j;
22      int sortiran=jesortiran(A,n);
23      if(sortiran){
24          printf("Niz_je_sortiran.\n");
25      }
26      else{
27          printf("Niz_nije_sortiran.\n");
28      }
29      return 0;
30  }

```

Napisati program koji proverava da li je data reč u nizu reči smeštenih u niz odvojenih NULL karakterom (\0).

imarec.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int main()
6 {
7     char *niz[] = {"Ana", "voli", "Milovana", "\0"};
8     char rec[] = "Milovana";
9     int i=0;
10    while (*niz[i]){
11        if (!strcmp(niz[i++], rec))
12        {
13            printf("Reč_%s'_se_nalazi_u_datom_nizu_na_rednom_broju_%d.", rec, i);
14            return 0;
15        }
16    }
17    printf("Reč_%s'_se_ne_nalazi_u_datom_nizu.", rec);
18    return 1;
19 }
```

Glava 2

Vežbe 3

2.1 Programiranje u C-u

Napisati u programskom jeziku C program koji prebrojava broj znakova u tekstu enkodiranom UTF8 standardom.

Računar čuva tekst kao niz bajtova. Jedan bajt ima 256 mogućih vrednosti. Unicode sadrži više od milion znakova: latinicu, ćirilicu, slova č, ć, š, ž, đ, kineske znakove, emodžije... Zato je za mnoge znakove potrebno više od jednog bajta.

Svaki Unicode znak ima svoj redni broj (*kodnu tačku*), koji se piše u obliku *U+heksadekadni broj*. Na primer, slovo a je U+0061, a slovo č je U+010D.

UTF-8 je pravilo koje kaže koliko bajtova dobija svaki znak i kako ti bajtovi izgledaju (n = broj bajtova):

Kodne tačke	n	Oblik bajtova	Primeri
U+0000–U+007F	1	0xxxxxxx	a, Z, 7, razmak
U+0080–U+07FF	2	110xxxxx 10xxxxxx	č, đ, ŽK
U+0800–U+FFFF	3	1110xxxx 10xxxxxx 10xxxxxx	€, kineski znakovi
U+10000–U+10FFFF	4	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx	emodžiji

Bitovi označeni sa x sadrže redni broj znaka. Početni bitovi svakog bajta govore kakvu ulogu taj bajt ima:

- **prvi bajt** znaka počinje sa 0, 110, 1110 ili 11110, i to nam govori koliko bajtova ima ceo znak;
- svaki **nastavni bajt** počinje sa 10.

Prvih 128 znakova (ASCII) zapisuje se isto kao ranije, jednim bajtom. Zato je svaki stari ASCII tekst ujedno i ispravan UTF-8 tekst.

Primeri

Znak	Kodna tačka	Bitovi	UTF-8 bajtovi
a	U+0061	1100001	61
č	U+010D	00100 001101	C4 8D
š	U+0161	00101 100001	C5 A1
€	U+20AC	0010 000010 101100	E2 82 AC
emodži :-)	U+1F600	000 011111 011000 000000	F0 9F 98 80

Na primer, za slovo č: $010D_{16} = 100001101_2$. Taj broj ima 9 bitova, pa staje u dva bajta sa 5 + 6 slobodnih bitova:

$$110\text{ }00100\text{ }10\text{ }001101 = 11000100\text{ }10001101 = C4\text{ }8D.$$

Zato je broj bajtova u utf8 fajlu obično veći od broja znakova. Reč „čaša“ ima 4 znaka, ali 6 bajtova: C4 8D 61 C5 A1 61.

Svaki znak ima *tačno jedan* prvi bajt. Dakle, broj znakova jednak je broju bajtova koji **nisu** nastavni, tj. koji nisu oblika 10xxxxxx.

Da bismo proverili prva dva bita bajta c, koristimo operaciju & nad bitovima:

- c & 0xC0 zadržava samo prva dva bita, jer je $C0_{16} = 11000000_2$;
- bajt je nastavni ako je rezultat jednak $80_{16} = 10000000_2$.

brojznakovautf8.c

```

1  #include "brojznakovautf8.h"
2
3  // Broji znakove (ne bajtove) u UTF-8 tekstu iz datoteke f.
4  // Svaki znak počinje tačno jednim bajtom koji NIJE oblika 10xxxxxx,
5  // pa je dovoljno prebrojati bajtove koji nisu nastavni.
6
7  long brojznakova(FILE *f)
8  {
9      long znakova = 0;
10     int c;
11     while((c = fgetc(f)) != EOF){
12         if((c & 0xC0) != 0x80) // Nije nastavni bajt 10xxxxxx
13             znakova++;        // Počinje novi znak
14     }
15     return znakova;
16 }
```

Datoteku otvaramo u binarnom režimu ("rb") jer želimo da čitamo bajtove onakve kakvi jesu. Funkcija fgetc vraća jedan bajt kao broj od 0 do 255, ili EOF na kraju datoteke.

Kompiliranje i izvršavanje

```

1 gcc -Wall -o brojznakova main.c brojznakovautf8.c
```

brojznakovautf8.h

```
1 #ifndef BROJZNAKOVAUTF8_H
2 #define BROJZNAKOVAUTF8_H
3
4 #include <stdio.h>
5
6 long brojznakova(FILE *f);
7
8 #endif
```

main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "brojznakovautf8.h"
4
5 int main(int argc, char *argv[])
6 {
7     if(argc < 2){
8         printf("Upotreba: _%s_datoteka\n", argv[0]);
9         return 1;
10    }
11    FILE *f = fopen(argv[1], "rb");
12    if(f == NULL){
13        printf("Ne_mogu_da_otvorim_datoteku_%s.\n", argv[1]);
14        return 1;
15    }
16    long znakova = brojznakova(f);
17    fclose(f);
18    printf("Znakova:_%ld\n", znakova);
19    return 0;
20 }
```

Za tekst „čša žđ Ѓирилица €“ i jedan emodži program ispisuje 19 znakova.

Napomena

Program pretpostavlja da je datoteka ispravan UTF-8 tekst. Ako nije, program neće prestati da radi, ali broj znakova može biti pogrešan.

brojznakovautf8provera.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Broji znakove (ne bajtove) u UTF-8 tekstu iz datoteke f i pri tome
5  // proverava da li je tekst ispravan UTF-8 (RFC 3629).
6  // Vraća broj znakova ako je tekst ispravan, a -n ako bajt sa rednim
7  // brojem n (bajtovi se broje od 1) krši UTF-8 pravilo.
8  //
9  // Vodeći bajt određuje koliko nastavnih bajtova 10xxxxx sledi:
10 // 00-7F 0xxxxxx 0 nastavnih
11 // C2-DF 110xxxx 1 nastavni (C0 i C1 bi dali predugačak zapis)
12 // E0-EF 1110xxxx 2 nastavna
13 // F0-F4 11110xxx 3 nastavna (F5-FF bi dali kod veći od 10FFFF)
14 // Za neke vodeće bajtove drugi bajt mora biti u užem opsegu:
15 // E0 -> A0-BF (inače predugačak zapis)
16 // ED -> 80-9F (inače surogat D800-DFFF)
17 // F0 -> 90-BF (inače predugačak zapis)
18 // F4 -> 80-8F (inače kod veći od 10FFFF)
19 // Ako se datoteka završi usred znaka, neispravan je vodeći bajt tog znaka.
20 long brojznakova(FILE *f)
21 {
22     long znakova = 0;
23     long poz = 0; // Redni broj poslednjeg pročitano bajta
24     long pocetak; // Redni broj vodećeg bajta tekućeg znaka
25     int c, n, i, min, max;
26     while((c = fgetc(f)) != EOF){
27         poz++;
28         if(c <= 0x7F) n = 0;
29         else if(c >= 0xC2 && c <= 0xDF) n = 1;
30         else if(c >= 0xE0 && c <= 0xEF) n = 2;
31         else if(c >= 0xF0 && c <= 0xF4) n = 3;
32         else return -poz; // Nastavni bajt bez vodećeg, C0, C1 ili F5-FF
33         pocetak = poz;
34         min = 0x80; max = 0xBF; // Uobičajen opseg nastavnog bajta
35         if(c == 0xE0) min = 0xA0;
36         else if(c == 0xED) max = 0x9F;
37         else if(c == 0xF0) min = 0x90;
38         else if(c == 0xF4) max = 0x8F;
39         for(i = 0; i < n; i++){
40             if((c = fgetc(f)) == EOF)
41                 return -pocetak; // Znak nije završen
42             poz++;
43             if(c < min || c > max)
44                 return -poz; // Nije ispravan nastavni bajt
45             min = 0x80; max = 0xBF; // Ostali nastavni bajtovi
46         }
47         znakova++;
48     }
49     return znakova;
50 }

```

2.2 Pretraživanje

Napisati u pseudo kodu program koji proverava da li nizu brojeva postoji zadati broj i program koji proverava da li u sortiranom nizu brojeva postoji zadati broj.

```
function IMA BROJ(A, broj)  
  n ← length(A)  
  flag ← false  
  for j ← 1 to n do  
    if A[j] = broj then  
      flag ← true  
      break  
    end if  
  end for  
  return flag  
end function
```

```
function IMA BROJ SORT(A, broj)  
  n ← length(A)  
  flag ← false  
  for j ← 1 to n do  
    if A[j] = broj then  
      flag ← true  
      break  
    end if  
    if A[j] > broj then  
      break  
    end if  
  end for  
  return flag  
end function
```

Napisati u programskom jeziku C program koji proverava da li u sortiranom nizu brojeva postoji zadati broj.

imabrojsort.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int A[] = {1,2,3,4,5,9,12};
5  int n = sizeof(A) / sizeof(A[0]);
6  int broj = 7;
7
8  int imabrojsort(int A[], int pocetak, int kraj, int broj)
9  {
10     // Ovaj kod napisati
11
12
13
14
15
16
17
18 }
19
20 int main()
21 {
22     int mesto = imabrojsort(A, 0, n-1, broj);
23     if(mesto){
24         printf("Broj %d se nalazi u nizu na mestu %d.\n", broj, mesto);
25     }
26     else{
27         printf("Broj %d nije u datom nizu.\n", broj);
28     }
29     return 0;
30 }
```

2.3 Polinomi

1. Dat je kod za računanje vrednosti polinoma. Polinom $a_n x^n + \dots + a_1 x + a_0$ je smešten u niz $a = [a_0, a_1, \dots, a_n]$.


```

1: function POLINOM( $a, x$ )
2:    $n \leftarrow \text{length}(a) - 1$ 
3:    $p \leftarrow a_0$ 
4:    $t \leftarrow 1$ 
5:   for  $k \leftarrow 1$  to  $n$  do
6:      $t \leftarrow t \cdot x$ 
7:      $p \leftarrow p + a_k \cdot t$ 
8:   end for
9:   return  $p$ 
10: end function

```

Izračunati broj sabiranja $S(n)$ i broj množenja $M(n)$ potrebnih da se izračuna vrednost polinoma stepena n datim aloritmom.

Rešenje: $S(n) = n, M(n) = 2n$.

2. Napisati pseudo kod algoritma HORNER za računanje vrednosti polinoma Hornerovom šemom.

```

function HORNER( $a, x$ )
   $n \leftarrow \text{length}(a)$ 
   $y \leftarrow a_n$ 
  for  $i \leftarrow n - 1$  downto 0 do
     $y \leftarrow x \cdot y + a_i$ 
  end for
  return  $y$ 
end function

```

3. Izračunati broj sabiranja $S_H(n)$ i broj množenja $M_H(n)$ potrebnih da se izračuna vrednost polinoma stepena n Hornerovom šemom:

	a_n	a_{n-1}	$\dots$	a_1	a_0
x	a_n	$x \cdot a_n + a_{n-1}$	$\dots$	$\dots$	$p_n(x)$

Rešenje: $S_H(n) = n, M_H(n) = n$.

4. Dat je polinom $p = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$ vrednostima koeficijenta u nizu A numerisanom od 0 do n . ($a[0] = a_0, a[1] = a_1, \dots, a[n] = a_n$) Kvadrat polinoma p se računa po formuli

$$p^2 = \sum_{i=0}^{2n} b_i x^i, \text{ gde je } b_i = \sum_{j=\max(0, i-n)}^{\min(i, n)} a_j a_{i-j}. \quad (2.1)$$

Na pr. $(a_0 + a_1x + a_2x^2)^2 = a_0a_0 + (a_0a_1 + a_1a_0)x + (a_0a_2 + a_1a_1 + a_2a_0)x^2 + (a_1a_2 + a_2a_1)x^3 + a_2a_2x^4$.

Napisati pseudokod procedure koja računa koeficijente polinoma $p^2 = b_0 + b_1x + b_2x^2 + \dots + b_{2n}x^{2n}$ i smešta ih u niz b ($b[0] = b_0, b[1] = b_1, \dots, b[2n] = b_{2n}$).

procedure KVADRAT(a, b)

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

...

end procedure

5. Koristeći (2.1), koliko sabiranja i koliko množenja se vrši za računanje $(1 + 2x + 3x^2 + 4x^3)^2$?
6. Koristeći (2.1), koliko sabiranja i koliko množenja se vrši za računanje p^2 za pol. p stepena n ?

Glava 3

Vežbe 4

3.1 Sortiranje

1. Naći primer ulaznog niza dužine n koji će za sortiranje algoritmom SELECTION SORT vršiti $n - 1$ swap.

Rešenje: $n, 1, 2, \dots, n - 1$.

2. Napisati funkciju partition za Hoare verziju quick sorta. Treba da podeli zadati deo niza na manje (od p do $q - 1$) i veće (od q do r) i treba da vrati redni broj q (**). Objasniti na primeru kako radi. Napisati prateće procedure koje se rekurzivno pozivaju.

```
procedure SORTH( $A, p, r$ )  
    ▷ Rekurzivna  
    if  $p < r$  then  
         $q \leftarrow \text{PARTITIONH}(A, p, r)$   
        SORTH( $A, p, q - 1$ )  
        SORTH( $A, q, r$ )  
    end if  
end procedure  
procedure QUICK SORT( $A$ )  
    ▷ Za korisnika  
     $n \leftarrow \text{length}(A)$   
    SORTH( $A, 1, n$ )  
end procedure
```

```
function PARTITIONH( $A, p, r$ )  
    ▷ U skladu sa (**)  
     $x \leftarrow A \left[ \left\lfloor \frac{p+r}{2} \right\rfloor \right]$   
    while  $p \leq r$  do  
        while  $A[p] < x$  do  
             $p \leftarrow p + 1$   
        end while  
        while  $A[r] > x$  do  
             $r \leftarrow r - 1$   
        end while  
        if  $p \leq r$  then  
            swap( $A, p, r$ )  
             $p \leftarrow p + 1$   
             $r \leftarrow r - 1$   
        end if  
    end while  
    return  $p$   
end function
```

Glava 4

Vežbe 5

4.1 Matrice

Napisati proceduru transpose koja transponuje matrice $n \times n$ u mestu

transpose.c

```
1 void transpose(double *A, int n)
2 { // transpose( A nXn ) u mestu
3     int i,j;
4     double x;
5
6     for(i=0;i<n;i++){
7         for(j=i+1;j<n;j++){
8             x = A[i*n+j];
9             A[i*n+j] = A[j*n+i];
10            A[j*n+i] = x;
11        }
12    }
13 }
```

Napisati proceduru koja množi matrice 3X3. Napisati program koji koristeći

tu proceduru množi matrice $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ $\begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$

matrixmult.c

```

1  #include <stdio.h>
2  #define maxn 3
3
4  void matrixmult(double a[maxn][maxn],
5  double b[maxn][maxn], double c[maxn][maxn])
6  {
7  int i,j,k;
8  for(i=0;i<maxn;i++){
9      for(j=0;j<maxn;j++){
10         c[i][j]=0.0;
11         for (k=0;k<maxn;k++)
12             c[i][j]=c[i][j]+a[i][k]*b[k][j];
13     }
14 }
15 }
16
17 int main()
18 {
19     double a[maxn][maxn]; double b[maxn][maxn]; double c[maxn][maxn];
20
21     a[0][0]=1; a[0][1]=2; a[0][2]=3;
22     a[1][0]=4; a[1][1]=5; a[1][2]=6;
23     a[2][0]=7; a[2][1]=8; a[2][2]=9;
24     b[0][0]=9; b[0][1]=8; b[0][2]=7;
25     b[1][0]=6; b[1][1]=5; b[1][2]=4;
26     b[2][0]=3; b[2][1]=2; b[2][2]=1;
27
28     matrixmult(a,b,c);
29
30     printf("Proizvod_matrica_je:\n");
31     for(int i=0;i<maxn;i++){
32         for(int j=0;j<maxn;j++)
33             printf("%5.2f",c[i][j]);
34         printf("\n");
35     }
36     return 0;
37 }

```

Da li se isti program može koristiti za množenje matrica 2×2 , ili 2×3 sa 3×2 ili 3×2 sa 2×3 ?

Proizvod matrica je:

30.00	24.00	18.00
84.00	69.00	54.00
138.00	114.00	90.00

Objasniti prototipove procedura i funkcija za matrice.

Zašto račun sa folija za $X = B^{-1}(C - A)$ vraća -0.00 u matrici?

+-				-+
	1.00	2.00	3.00	
	0.00	1.00	2.00	
	-1.00	-1.00	-0.00	
+-				-+

Da li može transponovanje matrice da se uradi „u mestu” ako se pozove procedura transpose sa istim ulazom i izlazom?

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include "matrice.h"
5
6  int main()
7  {
8      int n = 3;
9
10     double A[]={ 1,2,-3,4,-7,-16,-7,-18,-16};
11     transpose(A,A,n,n);
12     printmatrix(A,n,n);
13 }
```

+-				-+
	1.00	2.00	-3.00	
	2.00	-7.00	-16.00	
	-3.00	-16.00	-16.00	
+-				-+

Glava 5

Vežbe 6

5.1 Stack i Queue

Napisati algoritam za proveru ispravnosti postavljenih zagrada u fajlu. Koristiti stek kao strukturu podataka za čuvanje otvorenih zagrada.

```
function ZAGRADE(ime_ulaza)
    ▷ Proverava ispravnost postavljenih zagrada. Vraća retcode :
    ▷ = 0 - zagrada ispravno postavljene
    ▷ = 1 - otvorena ch1 do kraja nije zatvorena
    ▷ = 2 - zatvorena ch1 prethodno nije otvorena
    ▷ = 3,4,5 - zatvorena ch neuparena sa otvorenom ch1 = (, [, {
    ulaz = open(ime_ulaza); MAKENULL(S); retcode ← 0
    while ¬eof(ulaz) do
        ch ← fgetc(ulaz)
        if (ch = ' ')|( ch = ' ')|( ch = '{') then
            PUSH(S, ch)
        else if (ch = ')')|( ch = ']' )|( ch = '}') then
            if ISEMPY(S) then
                retcode ← 2; ch1 ← ch
                break
            else
                ch1 ← POP(S)
```

```

    if (ch=' ')&(ch1 ≠ ' ') then
        retcode ← 3
        break
    else if (ch=' ')&(ch1 ≠ ' ') then
        retcode ← 4
        break
    else if (ch='}')&(ch1 ≠ '{') then
        retcode ← 5
        break
    end if
end if
end if
end while
if ¬retcode then
    if ISEMPY(S) then
        retcode ← 0; ch1 ← \0
    else
        retcode ← 1; ch1 ← TOP(S); CLEAR(S)
    end if
end if
fclose(ulaz)
return retcode, ch1
end function

```

Dati implementaciju programa za proveru postavljenih zagrada u programskom jeziku C, koristeći ADT Stack.

zgrade.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "stack.h"
4
5  int main(void)
6  {
7      FILE *ulaz;
8      stack S = NULL;
9      int retcode = 0;
10     listdata ch, ch1;
11
12     makenull(&S);
13     ulaz = fopen("ulaz.txt", "r");
14     if (ulaz == NULL){
15         retcode = 6;
16     }
17     while( !retcode && !feof(ulaz) ){
18         ch = fgetc(ulaz);
19
20         if((ch=='(')|| (ch=='[') || (ch=='{')){
21             if(push(&S,ch))
22                 retcode = 7;
23         }
24         else if((ch==')') || (ch==']') || (ch == '}')){
25             if(isempty(S)){
26                 retcode = 2;
27             }
28             else{
29                 ch1 = pop(&S);
30                 if((ch==')')&&(ch1!='(')){
31                     retcode = 3;
32                 }
33                 else if((ch==']')&&(ch1!='[')){
34                     retcode = 4;
35                 }
36                 else if((ch=='}')&&(ch1!='{')){
37                     retcode = 5;
38                 }
39             }
40         }
41     }
42     fclose(ulaz);
43     if(!retcode){
44         if(isempty(S)){
45             retcode = 0;
46         }
47         else{
48             retcode = 1;

```

```
49     ch1 = top(S);
50     }
51 }
52 clear(&S);
53
54 switch(retcode){
55     case 0:
56         printf("\nZagrade_su_dobro_postavljene!\n");
57         break;
58     case 1:
59         printf("\nOtvorena_je_zagrada_%c_koja_do_kraja_fajla_nije_zatvorena!\n",ch1);
60         break;
61     case 2:
62         printf("\nZatvorena_je_zagrada_%c_koja_pre_toga_nije_otvorena!\n",ch);
63         break;
64     case 6:
65         printf("\nNe_moze_da_se_otvori_fajl_ulaz.txt!\n");
66         break;
67     case 7:
68         printf("\nNema_dovoljno_memorije!\n");
69         break;
70     default:
71         printf("\nZagrade_nisu_dobro_postavljene!\n");
72         printf("otvorena_zagrada_%c_je_uparena_sa_zatvorenom_zagradom_%c!\n",ch1,ch);
73         break;
74 }
75
76 return retcode;
77 }
```

Glava 6

Vežbe 7

6.1 Implementacija Stack i Queue

Dati implementaciju ADT Stack u programskom jeziku C pomoću niza.

Zbog kompatibilnosti interfejsa sa rešenjem ADT Stek preko povezanih lista zadržali smo isti heder fajl, tako da u istim situacijama prenosimo pokazivač na poslednji ubačen čvor kao parametar S , odnosno adresu tog pokazivača kao parametar ($*S$).

Tako je stack pokazivač na strukturu koja se iz razloga kompatibilnosti zove node u kojoj se pamte elementi steka i dužina steka. Dužina je u polju count i ona je faktički za jedan manja od stvarne dužine steka. U polju node pamtimo niz data maksimalne dužine MAXS u koji smeštamo elemente steka.

Inicijalizacija steka (makenull) se vrši alokacijom memorije za node i potom alokacijom memorije za data sadržaj steka. Na ovaj način je teoretski moguće povećati kapacitet steka realokacijom memorije.

stack.h

```

1  typedef char listdata;
2  typedef struct _node node;
3  typedef node *stack;
4
5  void makenull(stack *);
6  int isempty(stack);
7  int push(stack *, listdata);
8  listdata pop(stack *);
9  listdata top(stack);
10 void clear(stack *);
11 int ismember(stack, listdata *);
12 void printstack(stack);

```

stack.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include "stack.h"
5  #define MAXS 1000
6
7  struct _node
8  {
9      listdata *data;
10     int count;
11 };
12
13 int isequal(listdata * lhs, listdata * rhs)
14 {
15     return !memcmp(lhs, rhs, sizeof(listdata));
16 }
17
18 void makenull(stack *S)
19 {
20     (*S)=malloc(sizeof(node));
21     (*S)->data=malloc(sizeof(listdata)*MAXS);
22     (*S)->count = -1;
23 }
24
25 int isempty(stack S)
26 {
27     return (int) (S->count== -1);
28 }
29
30 int push(stack *S, listdata d)
31 {
32     if((*S)->count+1>=MAXS)
33         return 1;
34     (*S)->data[(*S)->count+1] = d;
35     (*S)->count = (*S)->count + 1;

```

```
36     return 0;
37 }
38
39 listdata pop(stack *S)
40 {
41     listdata d = (*S)->data[(*S)->count];
42     (*S)->count = (*S)->count - 1;
43     return d;
44 }
45
46 listdata top(stack S)
47 {
48     listdata d = S->data[S->count];
49     return d;
50 }
51
52 void clear(stack *S)
53 {
54     free((*S)->data);
55     free(*S);
56 }
57
58 int ismember(stack S, listdata *ch)
59 {
60     int i;
61     for(i=S->count;i>=0;i--)
62     {
63         if(isequal(&(S->data[i]),ch))
64             return 1;
65     }
66     return 0;
67 }
68
69 void printstack(stack S)
70 {
71     int i;
72     for(i=S->count;i>=0;i--)
73     {
74         printf("%c\n",S->data[i]);
75     }
76     printf("\n");
77 }
```

Dati implementaciju ADT Queue u programskom jeziku C pomoću niza.

queue.h

```

1  typedef char listdata;
2  typedef struct _node node;
3  typedef struct _queue *queue;
4
5  void makenullQ(queue *);
6  int isemptyQ(queue);
7  int enqueue(queue, listdata);
8  listdata dequeue(queue);
9  listdata front(queue);
10 void clearQ(queue);
11 int ismemberQ(queue, listdata *);
12 void printqueue(queue);

```

queue.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include "queue.h"
5  #define MAXQ 1000
6
7  struct _queue
8  {
9      listdata *data;
10     int front;
11     int count;
12 };
13
14 typedef struct _queue queues;
15
16 int isequal(listdata * lhs, listdata * rhs)
17 {
18     return !memcmp(lhs, rhs, sizeof(listdata));
19 }
20
21 void makenullQ(queue *Q)
22 {
23     (*Q) = malloc(sizeof(queues));
24     (*Q)->data = malloc(sizeof(listdata)*MAXQ);
25     (*Q)->front = 0;
26     (*Q)->count = 0;
27 }
28
29 int isemptyQ(queue Q)
30 {
31     return !(Q->count);
32 }
33

```

```
34 int enqueue(queue Q, listdata d)
35 {
36     if(Q->count+1==MAXQ){
37         return 1;
38     }
39     Q->data[(Q->front+Q->count)%MAXQ] = d;
40     Q->count++;
41     return 0;
42 }
43
44 listdata dequeue(queue Q)
45 {
46     listdata d = Q->data[Q->front];
47     Q->front = (Q->front+1) % MAXQ;
48     Q->count--;
49     return d;
50 }
51
52 listdata front(queue Q)
53 {
54     listdata d = Q->data[Q->front];
55     return d;
56 }
57
58 void clearQ(queue Q)
59 {
60     free(Q->data);
61     free(Q);
62 }
63
64 int ismemberQ(queue Q, listdata *chp)
65 {
66     int i;
67     for(i=0;i<Q->count;i++)
68         if(isequal(
69             &(Q->data[(Q->front+i)%MAXQ]),
70             chp))
71             return 1;
72     return 0;
73 }
74
75 void printqueue(queue Q)
76 {
77     int i;
78     for(i=0;i<Q->count;i++)
79         printf("%c\n",
80             Q->data[(Q->front+i)%MAXQ]);
81 }
```

Napisati u programskom jeziku C program koji, koristeći ADT Queue, iz fajla ulaz.txt u fajl izlaz.txt prepisuje prvo pojavljivanje karaktera redom kojim se pojavljuju (ignoriše duplikate).

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "queue.h"
4
5  int main(void)
6  {
7      FILE *ulaz;
8      FILE *izlaz;
9      queue Q;
10     listdata ch;
11
12     ulaz = fopen("ulaz.txt", "r");
13     izlaz = fopen("izlaz.txt", "w");
14
15     makenullQ(&Q);
16
17     while((ch=fgetc(ulaz)) != EOF)
18         if(!ismemberQ(Q,&ch))
19             if(enqueue(Q,ch)){
20                 printf("\n_Nema_dovoljno_memorije!\n");
21                 clearQ(Q);
22                 fclose(izlaz);
23                 fclose(ulaz);
24                 return 1;
25             }
26
27     while(!isemptyQ(Q)){
28         fprintf(izlaz, "%c", dequeue(Q));
29     }
30
31     clearQ(Q);
32
33     fclose(izlaz);
34     fclose(ulaz);
35
36     return 0;
37 }
```

Skicirati kompletan graf K_5 , kompletan graf K_6 , kompletan bipartitni graf $K_{2,2}$.

Skicirati usmereni graf „je roditelj odšvog porodičnog stabla polazeći od pradede (ili prababe) kojoj znaš ime.

Glava 7

Vežbe 8

7.1 Grafovi

Dati reprezentaciju tabelom listi susedstva i matricom susedstva kompletan graf K_5 , kompletan graf K_6 , kompletan bipartitni graf $K_{2,2}$.

Popuniti tabelu sa DA/NE:

	K_5	K_6	$K_{2,3}$
Ojlerov			
Hamiltonov			
Planaran			

U programskom jeziku C napisati proceduru koja transponuje graf koristeći proceduru `enqueue_list`.

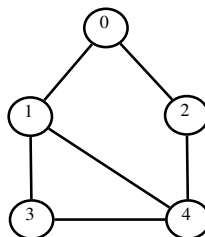
```
1 void transpose_graph(grana G[], int n, grana GT[])
2 {
3     int i,j;
4     grana gr;
5     grana *rear[max_cvorova];
6
7     for(i=0;i<max_cvorova;i++){
```

```
8      GT[i] = NULL;
9      rear[i] = &(GT[i]);
10     }
11
12     for(i=0;i<n;i++){
13         gr = G[i];
14         while(gr){
15             j = gr->data;
16             enqueue_list(&(rear[j]),i);
17             gr = gr->next;
18         }
19     }
20 }
```

Napisati u programskom jeziku C proceduru `adjlist2adjmatrix` koja zapis grafa datog u listi susedstva (Adjacency list) pretvara u zapis matrice susedstva (Adjacency matrix).

Napisati proceduru `adjmatrix2adjlist` koja sa istim ulaznim parametrima radi obrnuto.

Napisati glavni program koji će testirati rešenje na usmerenom grafu grafu desno.



Cvor | Adj(Cvor) :

```

-----+-----
0 | 1, 2,
1 | 0, 3, 4,
2 | 0, 4,
3 | 1, 4,
4 | 1, 2, 3,
-----+-----

```

```

      | 0 1 2 3 4
-----+-----
0 | 0 1 1 0 0
1 | 1 0 0 1 1
2 | 1 0 0 0 1
3 | 0 1 0 0 1
4 | 0 1 1 1 0

```

Cvor | Adj(Cvor) :

```

-----+-----
0 | 1, 2,
1 | 0, 3, 4,
2 | 0, 4,
3 | 1, 4,
4 | 1, 2, 3,
-----+-----

```

```
1  int main(void) {
2      grana G[max_cvorova];
3      grana *rear[max_cvorova];
4      int n;
5      nextnode u;
6      unsigned char M[max_cvorova*max_cvorova];
7      for(u=0;u<max_cvorova;u++){
8          G[u] = NULL;
9          rear[u] = &(amp;G[u]);
10     }
11     enqueue_list(&rear[0],1);
12     enqueue_list(&rear[0],2);
13     enqueue_list(&rear[1],0);
14     enqueue_list(&rear[1],3);
15     enqueue_list(&rear[1],4);
16     enqueue_list(&rear[2],0);
17     enqueue_list(&rear[2],4);
18     enqueue_list(&rear[3],1);
19     enqueue_list(&rear[3],4);
20     enqueue_list(&rear[4],1);
21     enqueue_list(&rear[4],2);
22     enqueue_list(&rear[4],3);
23     n = 5; // broj cvorova
24
25     print_graph(G,n);
26     adjlist2adjmatrix(G,n,M);
27     print_graph_matrix(M,n);
28     clear_graph(G);
29     adjmatrix2adjlist(G,n,M);
30     print_graph(G,n);
31     return 0;
32 }
```

```
1 void adjlist2adjmatrix(grana G[], int n,  
2     unsigned char M[])  
3 {  
4     int i;  
5     grana gr;  
6     for(i=0;i<n*n;i++) M[i] = 0;  
7     for(i=0;i<n;i++){  
8         gr = G[i];  
9         while(gr){  
10            M[i*n+gr->data] = 1;  
11            gr = gr->next;  
12        }  
13    }  
14 }
```

```
1 void adjmatrix2adjlist(grana G[], int n,  
2     unsigned char M[])  
3 {  
4     int i,j;  
5     grana *grp;  
6  
7     for(i=0;i<n;i++){  
8         {  
9             grp = &(G[i]);  
10            for(j=0;j<n;j++){  
11                if(M[i*n+j])  
12                    enqueue_list(&grp,j,0);  
13            }  
14        }
```


Glava 8

Vežbe 9

8.1 BFS i DFS

Dati implementaciju u programskom jeziku C procedure BFS.

```
1  int bfs(grana G[],int n,int s,int d[],int p[])
2  {
3      grana gr;
4      queue Q;
5      int i,color[max_cvorova];
6      cvor u,v;
7      for(i=0;i<n;i++){
8          d[i] = INT_MAX;    //infinity
9          p[i] = -1;        //no predecessor
10         color[i] = 0;     //WHITE
11     }
12     makenullQ(&Q);
13     d[s] = 0;
14     p[s] = -1;
15     color[s] = 1;        //GRAY
16     enqueue(Q,s);
17     while(!isemptyQ(Q)){
```

```
18     u = dequeue(Q);
19     gr = G[u];
20     while(gr){
21         v = gr->data;
22         if(!color[v]){
23             color[v] = 1; //GRAY
24             d[v] = d[u] + 1;
25             p[v] = u;
26             enqueue(Q,v);
27         }
28         gr = gr->next;
29     }
30     color[u] = 2; //BLACK
31 }
32 clearQ(Q);
33 return 0;
34 }
```

Napisati u programskom jeziku C procedure `path` i `path1` koje posle primenjenog BFS algoritma, raspoložuci izlazom `p` nalazi najkraći put od čvora `odavde` do čvora `dovde` i obrnuto.

```
1 void path(int p[], int dovde, int odavde)
2 {
3     if(p[odavde]==-1){
4         printf("\nNo_path!\n");
5         return;
6     }
7     printf("\n%2d->_", odavde);
8     while(!(p[odavde]==dovde)){
9         if(p[odavde]==-1){
10             printf("\nNo_path!\n");
11             return;
12         }
13         printf("%2d->_", p[odavde]);
14         odavde = p[odavde];
15     }
16     printf("%2d", dovde);
17 }
18
19 void path1(int p[], int dovde, int odavde)
20 {
21     if(p[odavde]==-1){
22         printf("\nNo_path!\n");
23         return;
24     }
25     if(p[odavde]==dovde){
26         printf("\n%2d->_%2d", dovde, odavde);
27         return;
28     }
29     else{
30         path1(p, dovde, p[odavde]);
31     }
32     printf("_->_%2d", odavde);
33 }
```

Napisati program za testiranje BFS, path, path1 i diametar.

```
1  int main(void) {
2      grana G[max_cvorova];
3      grana *rear[max_cvorova];
4      int d[max_cvorova], p[max_cvorova];
5      int i, n, u, retcode;
6
7      // Učitavanje grafa
8      n = ucitaj_graf(G);
9      print_graph(G,n);
10     bfs(G, n, 0, d, p);
11
12     // Štampanje rastojanja i prethodnika
13     for (i = 0; i < n; i++) {
14         printf("d[%2d]_=_%2d,_p[%2d]_=_%2d\n",
15             i, d[i], i, p[i]);
16     }
17
18     // Štampanje puta od čvora 7 do čvora 0
19     path(p, 0, 7);
20     printf("\n\n");
21
22     // Štampanje puta od čvora 0 do čvora 7
23     path1(p, 0, 7);
24     printf("\n\n");
25
26     // Nalaženje diametra
27     retcode = diametar(G, 8);
28     printf("Diametar_=_%d\n\n", retcode);
29
30     // Čišćenje grafa
31     clear_graph(G);
32     return 0;
33 }
```

Doraditi program `dijametar` tako da zapamti i kako glasi najveće najkraće rastojanje. Koristiti `queue` ili `stack`.

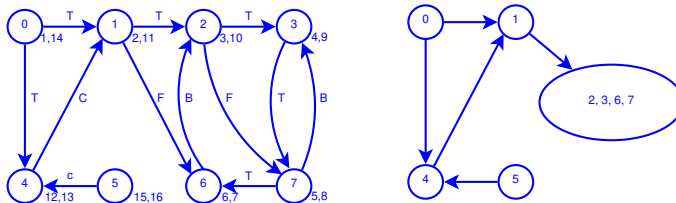
```
1  int dijametar(grana G[], int n)
2  {
3      int diam = 0;
4      int d[max_cvorova], p[max_cvorova];
5      int i,j;
6
7      for(i=0;i<n;i++){
8          bfs(G,n,i,d,p);
9          for(j=0;j<n;j++){
10             if(d[j]>diam){
11                 diam = d[j];
12             }
13         }
14     }
15     return diam;
16 }
```

Nacrtati usmereni graf G koji je dat tabelom listi susedstva:

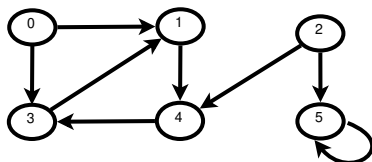
u	$\text{Adj}(u)$
0	1, 4
1	2, 6
2	3, 7
3	7
4	1
5	4
6	2
7	3, 6

Na graf G primeniti DFS algoritam, kod čvorova napisati d i f vrednosti, kod grana napisati tip (TBCF), napraviti tabelu zagrada. Ako je dati graf DAG, dati topološko sortiranje čvorova, ako nije, dati graf komponenti jake povezanosti.

Ovo nije DAG, dole desno je graf komponenti jake povezanosti.



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



Na graf sa slike gore primeniti DFS algoritam. Dati crtež grafa sa napisanim d i f vrednostima pored čvorova i tipom grane (T/B/F/C) na granama.

Držati se leksikografskog redosleda. Napraviti tabelu zagrada. Nacrtati šumu ovog DFS.

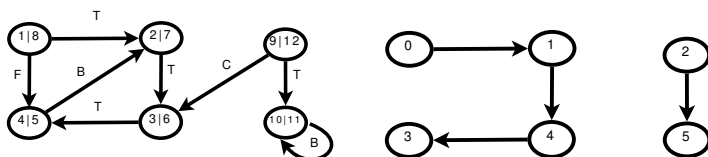
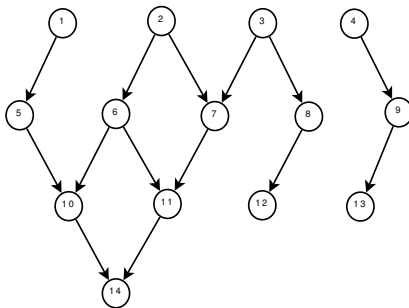


Tabela zagrada:

[illegible]



Primeniti DFS algoritam na graf sa slike. Pored čvorova ispisati d i f vrednosti, na granama napisati tip grane kada se prvi put otkrije: T = tree, F = forward, C = cross, B = back.

Čvorove i grane uzimati leksikografski.

Da li je graf sa slike usmereni aciklični graf (DAG)? Obrazložiti.

Ako se ignorišu usmerenja grana, da li je graf sa slike drvo? Obrazložiti.

Ako je graf sa slike DAG, napisati redosled čvorova koji daje topološko sortiranje dobijeno primenom DFS algoritma iz prethodnog zadatka.

Dat je kompletan neusmereni graf K_5 sa skupom čvorova $\mathcal{N} = \{1, 2, 3, 4, 5\}$.

Napisati skup grana $\mathcal{A}$.

Dati skicu grafa.

Dati tabelu listi susedstva grafa K_5 .

Primeniti algoritam DFS na grafu K_5 uzimajući čvorove leksikografski. Dati za svaki čvor d i f vrednost.

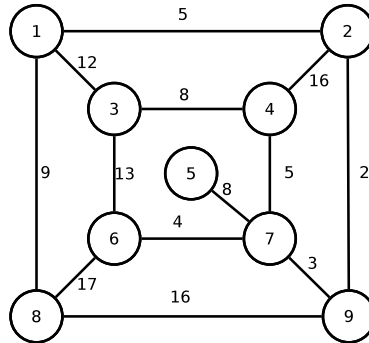
Koji tipovi grana se ne mogu pojaviti u primeni DFS na neusmeren graf?

Primeniti algoritam BFS na grafu K_5 polazeći od čvora 1, uzimajući čvorove leksikografski. Dati za svaki čvor udaljenost d (u broju grana) od čvora 1 i prethodnika p na najkraćem putu prema čvoru 1.

Glava 9

Vežbe 10

9.1 Minimalno pokrivajuće drvo (MST)



U zadacima koristiti graf sa slike, držati se leksikografskog redosleda i ignorisati težine grana.

1. Dati reprezentaciju listama susedstva.
2. Primeniti na isti graf BFS algoritam polazeći od čvora 5, dati tabelu prethodnika i broj koraka od čvora 5.
3. Odrediti dijametar grafa sa slike i obrazložiti.

4. Za graf sa slike levo naći minimalno pokrivajuće drvo Primovom metodom polazeći od čvora 5. Napisati redosled kojim su dodavane grane na minimalno pokrivajuće drvo.
5. Za graf sa slike levo naći minimalno pokrivajuće drvo Kruskalovom metodom. Napisati redosled kojim su dodavane grane na minimalno pokrivajuće drvo.

Glava 10

Vežbe 11

10.1 Drva i binarna drva

Parents:

Father: 1. Prince Philip, Duke of Edinburgh (1921–2021)

Mother: 2. Queen Elizabeth II (1926–2022)

Grandparents:

Paternal:

Deda: 3. Prince Andrew of Greece and Denmark

Baba: 4. Princess Alice of Battenberg

Maternal:

Deda: 5. King George VI

Baba: 6. Queen Elizabeth (The Queen Mother)

Grand-Grandparents

On Prince Philip's side

Pradedas: 7. King George I of Greece

Prababa: 8. Grand Duchess Olga Constantinovna of Russia

Pradedas: 9. Prince Louis of Battenberg (later Mountbatten)

Prababa: 10. Princess Victoria of Hesse and by Rhine

On Queen Elizabeth II's side

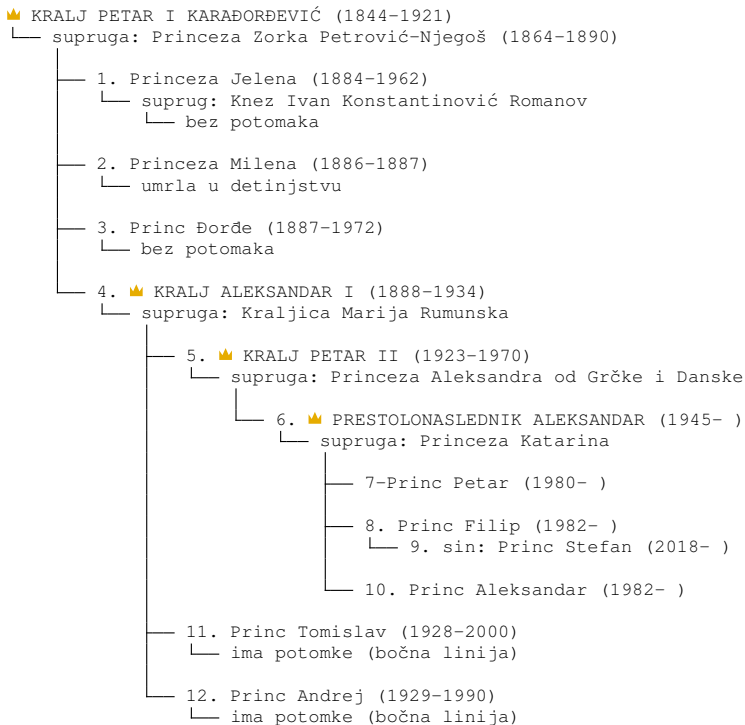
Pradedas: 11. King George V

Prababa: 12. Queen Mary of Teck

Pradedas: 13. Claude Bowes-Lyon, 14th Earl of Strathmore

Prababa: 14. Cecilia Cavendish-Bentinck

2. Napraviti porodično drvo potomaka Kralja Petra Prvog. Koristiti redne brojeve.



Da li je dobijeni graf drvo? Da li je dobijeni graf korensko drvo? Dodati u prethodni graf i supruge. Da li je dobijeni graf drvo? Da li je dobijeni graf korensko drvo?

3. Nacrtati drvo terma izraza $3 \cdot (4 - 5) + 6$ i dati LC-RC reprezentaciju. (6. VIII 2025. godine)

4. Za ulazni niz [5, 1, 6, 4, 8, 10, 3] kreirati binarno drvo pretraživanja (Binary Search Tree) tako da se redom elementi ubacuju:

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

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define max 10
4  int ulaz[]={5,1,6,4,8,10,3};
5  int key[max];
6  int LC[max];
7  int RC[max];
8  int iskorisceno = -1;
9
10 int ubaci(int koren, int broj) {
11     if(koren>-1){
12         if(broj<key[koren])
13             LC[koren]=ubaci(LC[koren],broj);
14         else
15             RC[koren]=ubaci(RC[koren],broj);
16         return koren;
17     }
18     else{
19         iskorisceno++;
20         key[iskorisceno] = broj;
21         LC[iskorisceno] = -1;
22         RC[iskorisceno] = -1;
23         return iskorisceno;
24     }
25 }
26
27 int main() {
28     int i,koren=-1;
29     for(i=0;i<7;i++)
30         koren = ubaci(koren,ulaz[i]);
31     return 0;
32 }
```

Pre komande **return** stanje key, LC i RC vrednosti do broja iskorisceno će biti kao utabeli:

key	5,	1,	6,	4,	8,	10,	3
LC	1,	-1,	-1,	6,	-1,	-1,	-1
RC	2,	3,	4,	-1,	5,	-1,	-1

5. Napisati proceduru pronadji koja nalazi element u BST.

6. Napisati proceduru postfix_print koja BST (Binary Search Tree) ispisuje u postfix redosledu.

```

1  int pronadji(int koren, int broj)
2  {
3      if( koren<0 || key[koren]==broj )
4          return koren;
5      if( key[koren] > broj )
6          return pronadji(LC[koren], broj );
7      return pronadji(RC[koren], broj );
8  }
9
10 void postfix_print(int koren)
11 {
12     if(koren>-1){
13         infix_print(LC[koren]);
14         infix_print(RC[koren]);
15         printf("%3d",key[koren]);
16     }
17 }
18
19 void infix_print(int koren)
20 {
21     if(koren>-1){
22         infix_print(LC[koren]);
23         printf("%3d",key[koren]);
24         infix_print(RC[koren]);
25     }
26 }
```

7. Šta je rezultat primene sledeće verzije main programa?

```
1 int main()
2 {
3     int i,koren=-1;
4     for(i=0;i<7;i++){
5         koren = ubaci(koren,ulaz[i]);
6     }
7
8     postfix_print(koren);
9     printf("\n");
10
11    broj = 7;
12    if (pronadji(0, broj)<0)
13        printf("\nNema_broja_%d_u_nizu.\n",broj);
14    else
15        printf("\nIma_broja_%d_u_nizu.\n",broj);
16    broj = 2;
17    if (pronadji(0, broj)<0)
18        printf("\nNema_broja_%d_u_nizu.\n",broj);
19    else
20        printf("\nIma_broja_%d_u_nizu.\n",broj);
21    broj = 4;
22    if (pronadji(0, broj)<0)
23        printf("\nNema_broja_%d_u_nizu.\n",broj);
24    else
25        printf("\nIma_broja_%d_u_nizu.\n",broj);
26    return 0;
27 }
```

1 3 4 6 8 10 5

Nema broja 7 u nizu.

Nema broja 2 u nizu.

Ima broja 4 u nizu.

8. Koja je složenost procedure pronadji za BST sa n elemenata? Šta je Worst case i šta je Best case?

Odgovor: Za balansirana drva: $O(\log n)$, za nebalansirana drva $O(n)$.

Glava 11

Vežbe 12

11.1 Problem angažovanja

1. U tabeli su data vremena potrebna radnicima (A, B, C, D, E) da obave posao (1, 2, 3, 4, 5), po jedan radnik na jedan posao:

	1	2	3	4	5
A	14	5	11	6	6
B	13	12	11	7	14
C	8	10	13	13	12
D	11	9	13	5	10
E	6	14	10	9	9

Mađarskom metodom odrediti na koji posao treba angažovati kojeg radnika (po jedan radnik na jedan posao) tako da ukupno vreme obavljanja poslova bude minimalno.

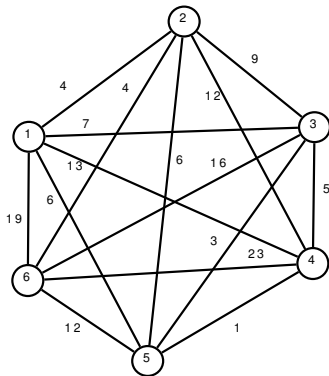
11.2 Problem trgovačkog putnika (TSP)

2. U tabeli su date cene prevoza između 5 gradova.

	1	2	3	4	5
1	-	41	115	37	110
2	28	-	92	51	90
3	103	76	-	95	119
4	66	50	65	-	136
5	100	95	92	115	-

- Polazeći od čvora 1, metodom najjeftinijeg suseda naći približno rešenje problema trgovačkog putnika (TSP).
- Za isti problem naći Mađarskom metodom angažovanje koje je rešenje relaksiranog TSP.
- Komentarirati rešenja (a) i (b).

3. Dat je graf sa cenama putovanja između 6 mesta.



- Definisati problem trgovačkog putnika.
- Polazeći od čvora 1, metodom najbližeg suseda naći približno rešenje problema trgovačkog putnika.
- Za isti problem naći Mađarskom metodom angažovanje koje je rešenje relaksiranog problema trgovačkog putnika.
- Komentarirati dobijena rešenja.