JMBAG

IME I PREZIME

# Programiranje 2 – kolokvij, 3. 5. 2024.

**Napomene.** Svako rješenje napišite isključivo na papir sa zadatkom jer jedino njega predajete. Pomoćne račune smijete raditi na drugim papirima koje će vam dati dežurni asistent. U svim zadacima **obavezno** pišite postupak!
Dozvoljeno je korištenje isključivo pribora za pisanje i brisanje, te službenog podsjetnika. Kalkulatori, te razne tablice, papiri i sl. nisu dozvoljeni! **Mobitele** isključite i pospremite daleko od sebe! Ako se ustanovi da **kod sebe** imate mobitel za vrijeme kolokvija, kolokvij se poništava i pokreće se stegovni postupak protiv vas.

**Zadatak 1** (12 bodova) Zeko skače po nizu i u košaricu skuplja mrkve. U svakom polju niza `int vrt[50];` nalazi se pozitivan cijeli broj koji označava broj mrkvi koje zeko može premjestiti u svoju košaricu kada se nađe na tom polju. Zeko kreće skakati sa polja na **indeksu 0** s kojeg kupi mrkve u košaricu te može skočiti za **2** ili **5 polja u desno**.

Nakon što skoči na polje, ukupan broj mrkvi s tog polja zeko doda u svoju košaricu, a vrijednost polja u nizu ostaje **nepromijenjena**. Na njegovu sreću, svaki put kada skoči na polje na kojem je više mrkvi nego je bilo na prethodnom, u košaricu mu se doda **5 gratis mrkvi**. Skakanje prestaje kada je skupio **barem 100 mrkvi** ili kada je skočio na polje izvan niza `vrt` (polje sa indeksom većim ili jednakim 50). Dodatno, nakon prestanka skakanja, ako je suma mrkvi u nizu `vrt` sa polja koje zeko nije posjetio **neparan broj**, dobiva **10 gratis mrkvi**.

Napište **rekurzivnu** funkciju `skokovi`, s argumentima po izboru, koja će vratiti **na koliko načina skakanja** zeko može skupiti barem 100 mrkvi. Dodatno, preko **varijabilnih argumenata**, funkcija `skokovi` treba vratiti **maksimalan broj mrkvi** koje zeko može skupiti te **minimalan broj skokova** s kojima je to uspio. Napišite funkciju `main` u kojoj pozivate funkciju `skokovi`.

**Primjer:** Za potrebe primjera, promotrimo `int vrt[10]={2,1,1,1,1,95,1,1,102,1};`

- Broj načina da skupi barem 100 mrkvi: **2** (za skokove 2,2,2,2 ili za skok 5)
- Maksimalan broj mrkvi koje može skupiti: **122**
- Minimalan broj skokova s kojima to može ostvariti: **4**

**Napomena:** Nije dozvoljeno korištenje dodatnih polja.

```
#include<stdio.h>

int skokovi(int* vrt, int indeks, int prethodno_mrkvi, int kosarica, int broj_skokova, int suma,
             int* max_mrkvi, int* min_skokova)
{
    int i;

    if(indeks<50) //prvo ažuriramo košaricu doskokom na polje unutar niza
    {
        if(indeks && prethodno_mrkvi<vrt[indeks]) kosarica+=5;
        kosarica+=vrt[indeks];
    }

    if(indeks>49 || kosarica>=100)
    {
        for(i=indeks+1;i<50;i++) suma+=vrt[i]; //preostale mrkve u nizu

        if(suma%2) kosarica+=10;

        if(kosarica<100) return 0; //treba provjeriti je li skupio barem 100 mrkvi izlaskom iz niza

        if(kosarica>*max_mrkvi || (kosarica==*max_mrkvi && broj_skokova<*min_skokova))
        {
            *max_mrkvi=kosarica;
            *min_skokova=broj_skokova;
        }
```

```
        return 1;
    }

    int count=0;

    if(indeks+1<50) suma+=vrt[indeks+1];

    count+=skokovi(vrt,indeks+2,vrt[indeks],kosarica,broj_skokova+1,suma,max_mrkvi,min_skokova);

    for(i=indeks+2;i<indeks+5 && i<50;i++)
        suma+=vrt[i];

    count+=skokovi(vrt,indeks+5,vrt[indeks],kosarica,broj_skokova+1,suma,max_mrkvi,min_skokova);

    return count;
}

int main()
{
    int vrt[50];
    int max_mrkvi=0, min_skokova, koliko_nacina;

    koliko_nacina=skokovi(vrt,0,0,0,0,0,&max_mrkvi,&min_skokova);

    printf("Nacina:%d, max mrkvi:%d, min skokova:%d\n",koliko_nacina,max_mrkvi,min_skokova);

    return 0;
}
```

JMBAG

IME I PREZIME

# Programiranje 2 – kolokvij, 3. 5. 2024.

**Zadatak 2** (10 bodova) Za kvadratnu matricu $X$ reda $n$ kažemo da je **Toeplitzova** ako su joj sve dijagonale konstantne. Npr., za $n = 4$, Toeplitzova matrica je oblika

$$X = \begin{bmatrix} x_0 & x_{-1} & x_{-2} & x_{-3} \\ x_1 & x_0 & x_{-1} & x_{-2} \\ x_2 & x_1 & x_0 & x_{-1} \\ x_3 & x_2 & x_1 & x_0 \end{bmatrix}.$$

Posebni slučaj Toeplitzove matrice je **cirkularna** matrica. Kod cirkularnih matrica se svaki redak, osim prvog, dobiva iz prethodnog retka cikličkim pomicanjem elemenata za jedno mjesto udesno. Primjer cirkularne matrice, za $n = 4$, je

$$Y = \begin{bmatrix} x & y & z & w \\ w & x & y & z \\ z & w & x & y \\ y & z & w & x \end{bmatrix}.$$

a) Napišite funckiju `isToeplitz` koja prima matricu $X \in \mathbb{Z}^{n \times n}$ i n. Funckija vraća 1 ako je matrica Toeplitzova, odnosno 0 ako nije.

b) Napišite funkciju `makeCirculantMatrix` koja prima matricu $X \in \mathbb{Z}^{n \times n}$, indeks $k$ te $n$. Funkcija matricu $X$ mijenja u cirkularnu matricu definiranu $k$-tim retkom te matrice. Npr. za $k = 3$ imamo sljedeću transformaciju

$$\begin{bmatrix} 2 & -3 & 8 & 1 \\ 1 & 2 & 3 & 4 \\ \mathbf{-7} & \mathbf{2} & \mathbf{-1} & \mathbf{0} \\ 2 & 4 & 6 & 8 \end{bmatrix} \Longrightarrow \begin{bmatrix} \mathbf{-7} & \mathbf{2} & \mathbf{-1} & \mathbf{0} \\ 0 & -7 & 2 & -1 \\ -1 & 0 & -7 & 2 \\ 2 & -1 & 0 & -7 \end{bmatrix}$$

**Napomena:** Možete predpostaviti da je $n \leq 20$. **Nije dozvoljeno** korištenje dodatnih polja.

```
#include<stdio.h>

int checkDiagonal(int X[20][20], int n, int i, int j)
{
    int temp = X[i][j];
    while(i<n && j<n)
    {
        if(X[i][j]!=temp)
            return 0;
        i++;
        j++;
    }
    return 1;
}

int isToeplitz(int X[20][20], int n)
{
    int i, j;

    //gornji trokut
    for(i=0; i<n; i++)
        if(checkDiagonal(X,n,0,i)!=1)
            return 0;

    //donji trokut
    for(i=1; i<n; i++)
        if(checkDiagonal(X,n,i,0)!=1)
            return 0;
```

```
    return 1;

}


void makeCirculant(int X[20][20], int k, int n)
{
    int i,j;
    for(i=0; i<n; i++)
        X[0][i] = X[k][i];
    for(i=1; i<n; i++)
    {
        for(j=1; j<n; j++)
            X[i][j] = X[i-1][j-1];
        X[i][0] = X[i-1][n-1];
    }
}
```

JMBAG

IME I PREZIME

# Programiranje 2 – kolokvij, 3. 5. 2024.

**Zadatak 3** (5+10 bodova) U nekoj ulici postoje zgrade u nizu poredane s lijeva na desno različitih visina (visina je reprezentirana prirodnim brojem). Stanari svake zgrade odlučili su u nadograditi na svoju zgradu najmanji broj katova tako da nijedna zgrada desno od njih ne bi bila viša od te. Nadogradnja se vrši istovremeno za sve zgrade u ulici odjednom. Npr: { 4,5,2,3,1,4,1,2 } $\xrightarrow{\text{nadogradnja}}$ { 5,5,4,4,4,4,2,2 } (visine zgrada u ulici prvo su bile kao u nizu lijevo; desno je stanje istih zgrada nakon nadogradnje). Ulicu u računalu spremamo kao niz `int`-ova u kojima je zadnja pozicija jednaka nuli, koja ne označava zgradu nego kraj ulice. Različite ulice (nizove zgrada) spremamo u novi niz, i to zovemo gradom.

a) Napišite funkciju `void nadogradnja (int **x, int n)` koja prima grad `x` te u svakoj od njegovih `n` ulica vrši nadogradnju. Pripazite na složenost!

b) Napišite funkciju `void sortirajGrad (int **x, int n)` koja sortira podatke ulica unutar nekog grada kojemu su sve ulice nadograđene (to ne treba provjeravati) uzlazno po visini najviše zgrade te ulice (smijete pretpostaviti da će te sve visine biti jedinstvene).

Npr. za `x[0]={2,3,0}, x[1]={5,1,0}, x[2]={4,0}` (što predstavalja dvije ulice s dvije zgrade i jednu s jednom zgradom), stanje nakon nadogradnje bi bilo `x[0]={3,3,0}, x[1]={5,1,0}, x[2]={4,0}`, a onda nakon sortiranja `x[0]={3,3,0}, x[1]={4,0}, x[2]={5,1,0}`.
Zadatak nosi 15 bodova, međutim ukoliko se umjesto algoritma sortiranja složenosti $\mathcal{O}(n \log n)$ koristi algoritam složenosti $\mathcal{O}(n^2)$, zadatak nosi maksimalno 10 bodova. Korištenje quickSort algoritma iz biblioteke `stdlib.h` nosi dodatnih 5 bodova, odnosno tako riješen zadatak u sumi nosi 20 bodova.
**Napomena**: Uz standardnu biblioteku `stdio.h`, u ovome zadatku dozvoljeno je korištenje još jedino biblioteke `stdlib.h`.

```
#include<stdio.h>
#include<stdlib.h>

int max (int x, int y) {
    return (x>y) ? x : y;
}

void nadogradnja (int **x, int n) {
    int i,j,len_i;
    for (i=0; i<n; i++) {
        len_i=0;
        while(x[i][len_i++]>0);
        len_i--;
        for (j=len_i-2; j>=0; j--)
            x[i][j]=max(x[i][j],x[i][j+1]);
    }
}

// pomoćna funkcija za vlastite implementacije sortova
void swap(int **x, int **y){
    int *tmp = *x;
    *x = *y;
    *y = tmp;
}

// vlastita implementacija klasičnog sorta
void classicSort(int **niz, int n) {
    int i,j;
    for(i = 0; i < n; ++i){
        for(j = i+1; j < n; ++j){
            if(niz[i][0] > niz[j][0]){
               swap(&niz[i], &niz[j]);
            }
        }
```

```
    }
}

// vlastita implementacija quicksorta
void quickSort(int **niz, int low, int high){
    int i, j;
    if(low < high){
        i = low + 1; j = high;
         while(i <= j){
             while(i <= high && (niz[i][0] < niz[low][0])) ++ i;
             while(niz[j][0] > niz[low][0]) -- j;
             if(i < j) {
                 swap(&niz[i], &niz[j]);
                 }
         }

         if(low < j) swap(&niz[low], &niz[j]);

         quickSort(niz, low, j-1);
         quickSort(niz, j+1, high);
    }
}

// pomoćna funkcija za gotovu funkciju qsort
int comparator(const void *a, const void *b){
    int* nizi = *(int**)a;
    int* nizj = *(int**)b;

    return (nizi[0] - nizj[0]);
}

// pozivi odgovarajućih sort funkcija
void sortirajGrad(int **niz, int n) {
    classicSort(niz,n);                         // Opcija 1 za max 10 bodova
    quickSort(niz,0,n-1);                       // Opcija 2 za max 15 bodova
    qsort(niz, n, sizeof(int*), comparator);    // Opcija 3 za max 20 bodova
}
```

JMBAG

IME I PREZIME

# Programiranje 2 – kolokvij, 3. 5. 2024.

**Zadatak 4** (4+9=13 bodova) Napišite funkciju `char* palindrom(char* s, int* br)` koja prima string te vraća podstring najveće duljine koji je palindrom čija je duljina barem 3. Ako takvih ima više, funkcija vraća onaj podstring koji je najveći po leksikografskom uređaju. Ako polazni string ne sadrži podstring koji je palindrom funkcija vraća prazan string. Za string kažemo da je palindrom ako je invertirani string jednak početnom do na mala i velika slova (npr. string `"abBa"` je palindrom, dok `"abca"` nije). Dodatno, preko varijabilnog argumenta funkcija vraća broj podstringova duljine veće ili jednake 3 koji su palindromi.
**Napomena.** Možete definirati dodatne (pomoćne) funkcije, ali **dodatni nizovi nisu dopušteni**. Smijete koristiti funkcije iz `ctype.h`, `string.h` i `stdlib.h`. Možete pretpostaviti da funkcija `palindrom` prima string duljine barem 3.

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>

int check_palindrom(char* s, int i, int j){
    int pal=1;
    while(i<j){
        if(tolower(s[i])!=tolower(s[j])){
            pal=0;
            break;
        }
        i++;
        j--;
    }
    return pal;
}

char* palindrom(char* s, int* br){
    *br=0;
    int i, j, k;
    int max=0;
    int n=strlen(s);
    int smax_i;
    for(i=0 ; i<=n-3 ; i++){
        for(j=i+2 ; j<n ; j++){
            if(check_palindrom(s, i, j)){
                (*br)++;
                if(j-i+1>max){
                    max=j-i+1;
                    smax_i=i;
                }
                else if(j-i+1==max){
                    int par=0;
                    for(k=0 ; k<max ; k++){
                        if(s[i+k]>s[smax_i+k]){
                            par=1;
                            break;
                        }
                    }
                    if(par==1){
                        smax_i=i;
                    }
                }
            }
        }
    }
```

```
    char* smax=(char*)malloc((max+1)*sizeof(char));
    if(max>0){
        for(k=0 ; k<max ; k++){
            smax[k]=s[smax_i+k];
        }
    }
    smax[max]='\0';
    return smax;
}
```

0

JMBAG

IME I PREZIME

# Programiranje 2 – kolokvij, 3. 5. 2024.

**Napomene.** Svako rješenje napišite isključivo na papir sa zadatkom jer jedino njega predajete. Pomoćne račune smijete raditi na drugim papirima koje će vam dati dežurni asistent. U svim zadacima **obavezno** pišite postupak!
Dozvoljeno je korištenje isključivo pribora za pisanje i brisanje, te službenog podsjetnika. Kalkulatori, te razne tablice, papiri i sl. nisu dozvoljeni! **Mobitele** isključite i pospremite daleko od sebe! Ako se ustanovi da **kod sebe** imate mobitel za vrijeme kolokvija, kolokvij se poništava i pokreće se stegovni postupak protiv vas.

**Zadatak 1** (12 bodova) Vjeverica skače po nizu i u košaricu skuplja žirove. U svakom polju niza `int park[50];` nalazi se pozitivan cijeli broj koji označava broj žirova koje vjeverica može premjestiti u svoju košaricu kada se nađe na tom polju. Vjeverica kreće skakati sa polja na **indeksu 0** s kojeg kupi žirove u košaricu te može skočiti za **3** ili **6 polja u desno**.

Nakon što skoči na polje, ukupan broj žirova s tog polja vjeverica doda u svoju košaricu, a vrijednost polja u nizu ostaje **nepromijenjena**. Na njezinu sreću, svaki put kada skoči na polje na kojem je više žirova nego je bilo na prethodnom, u košaricu joj se doda **10 gratis žirova**. Skakanje prestaje kada je skupila **barem 100 žirova** ili kada je skočila na polje izvan niza `park` (polje sa indeksom većim ili jednakim 50). Dodatno, nakon prestanka skakanja, ako je suma žirova u nizu `park` sa polja koje vjeverica nije posjetila **neparan broj**, dobiva **15 gratis žirova**.

Napište **rekurzivnu** funkciju `skokovi`, s argumentima po izboru, koja će vratiti **na koliko načina skakanja** vjeverica može skupiti barem 100 žirova. Dodatno, preko **varijabilnih argumenata**, funkcija `skokovi` treba vratiti **maksimalan broj žirova** koje vjeverica može skupiti te **minimalan broj skokova** s kojima je to uspjela. Napišite funkciju `main` u kojoj pozivate funkciju `skokovi`.

**Primjer:** Za potrebe primjera, promotrimo `int park[10]={3,2,2,90,1,2,85,2,1,2};`

- Broj načina da skupi barem 100 žirova: **2** (za skok 3 ili za skokove 6,3)
- Maksimalan broj žirova koje može skupiti: **118**
- Minimalan broj skokova s kojima to može ostvariti: **1**

**Napomena:** Nije dozvoljeno korištenje dodatnih polja.

```
#include<stdio.h>

int skokovi(int* park, int indeks, int prethodno_zirova, int kosarica, int broj_skokova, int suma,
             int* max_zirova, int* min_skokova)
{
    int i;

    if(indeks<50) //prvo ažuriramo košaricu doskokom na polje unutar niza
    {
        if(indeks && prethodno_zirova<park[indeks]) kosarica+=10;
        kosarica+=park[indeks];
    }

    if(indeks>49 || kosarica>=100)
    {
        for(i=indeks+1;i<50;i++) suma+=park[i]; //ako smo zavrsili sa skakanjem zbog kosarica>=100

        if(suma%2) kosarica+=15;

        if(kosarica<100) return 0; //treba provjeriti je li skupila barem 100 zirova izlaskom iz niza

        if(kosarica>*max_zirova || (kosarica==*max_zirova && broj_skokova<*min_skokova))
        {
            *max_zirova=kosarica;
            *min_skokova=broj_skokova;
        }
        return 1;
```

```
    }

    int count=0;

    for(i=indeks+1;i<indeks+3 && i<50;i++)
        suma+=park[i];

    count+=skokovi(park,indeks+3,park[indeks],kosarica,broj_skokova+1,suma,max_zirova,min_skokova);

    for(i=indeks+3;i<indeks+6 && i<50;i++)
        suma+=park[i];

    count+=skokovi(park,indeks+6,park[indeks],kosarica,broj_skokova+1,suma,max_zirova,min_skokova);

    //sve kombinacije skokova

    return count;
}

int main()
{
    int park[50];
    int max_zirova=0, min_skokova, koliko_nacina;

    koliko_nacina=skokovi(park,0,0,0,0,0,&max_zirova,&min_skokova);

    printf("Nacina:%d, max zirova:%d, min skokova:%d\n",koliko_nacina,max_zirova,min_skokova);

    return 0;
}
```

JMBAG

IME I PREZIME

# Programiranje 2 – kolokvij, 3. 5. 2024.

**Zadatak 2** (10 bodova) Za kvadratnu matricu $A$ reda $n$ kažemo da je **Toeplitzova** ako su joj sve dijagonale konstantne. Npr., za $n = 4$, Toeplitzova matrica je oblika

$$A = \begin{bmatrix} a_0 & a_{-1} & a_{-2} & a_{-3} \\ a_1 & a_0 & a_{-1} & a_{-2} \\ a_2 & a_1 & a_0 & a_{-1} \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix}.$$

Posebni slučaj Toeplitzove matrice je **cirkularna** matrica. Kod cirkularnih matrica se svaki redak, osim prvog, dobiva iz prethodnog retka cikličkim pomicanjem elemenata za jedno mjesto udesno. Primjer cirkularne matrice, za $n = 4$, je

$$B = \begin{bmatrix} a & b & c & d \\ d & a & b & c \\ c & d & a & b \\ b & c & d & a \end{bmatrix}.$$

a) Napišite funckiju `isToeplitz` koja prima matricu $A \in \mathbb{Z}^{n\times n}$ i n. Funckija vraća 2 ako je matrica Toeplitzova, odnosno 0 ako nije.

b) Napišite funkciju `makeCirculantMatrix` koja prima matricu $A \in \mathbb{Z}^{n\times n}$, indeks $k$ te $n$. Funkcija matricu $A$ mijenja u cirkularnu matricu definiranu $k$-tim stupcem te matrice. Npr. za $k = 2$ imamo sljedeću transformaciju

$$\begin{bmatrix} 2 & \mathbf{-1} & 8 & 1 \\ -1 & \mathbf{2} & -3 & 4 \\ -7 & \mathbf{-3} & -1 & 0 \\ 2 & \mathbf{4} & 6 & 8 \end{bmatrix} \Longrightarrow \begin{bmatrix} \mathbf{-1} & \mathbf{2} & \mathbf{-3} & \mathbf{4} \\ 4 & -1 & 2 & -3 \\ -3 & 4 & -1 & 2 \\ 2 & -3 & 4 & -1 \end{bmatrix}$$

**Napomena:** Možete predpostaviti da je $n \leq 20$. **Nije dozvoljeno** korištenje dodatnih polja.

```
#include<stdio.h>

int checkDiagonal(int A[20][20], int n, int i, int j)
{
    int temp = A[i][j];
    while(i<n && j<n)
    {
        if(A[i][j]!=temp)
            return 0;
        i++;
        j++;
    }
    return 1;
}

int isToeplitz(int A[20][20], int n)
{
    int i, j;

    //gornji trokut
    for(i=0; i<n; i++)
        if(checkDiagonal(A,n,0,i)!=1)
            return 0;

    //donji trokut
    for(i=1; i<n; i++)
        if(checkDiagonal(A,n,i,0)!=1)
            return 0;
```

```
    return 2;

}


void makeCirculant(int A[20][20], int k, int n)
{
    int i,j;
    for(i=0; i<n; i++)
        A[0][i] = A[i][k];
    for(i=1; i<n; i++)
    {
        for(j=1; j<n; j++)
            A[i][j] = A[i-1][j-1];
        A[i][0] = A[i-1][n-1];
    }
}
```

JMBAG

IME I PREZIME

# Programiranje 2 – kolokvij, 3. 5. 2024.

**Zadatak 3** (5+10 bodova) U nekom drvoredu postoje stabla u nizu poredana s lijeva na desno različitih visina (visina je reprezentirana prirodnim brojem). Pod utjecajem jakog vjetra, svako stablo se lomi te njegova visina pada (ili ostaje ista) i postaje najviši broj takav da nijedno stablo desno od njega nije niže od njega. Lom stabala se vrši istovremeno za sva stabla u drvoredu odjednom. Npr: { `2,1,4,3,5,2,5,4` } $\xrightarrow{\text{lom stabala}}$ { `1,1,2,2,2,2,4,4` } (visine stabala u drvoredu prvo su bile kao u nizu lijevo; desno je stanje istih stabala nakon loma stabala). Drvored u računalu spremamo kao niz `int`-ova u kojima je zadnja pozicija jednaka nuli, koja ne označava stablo nego kraj drvoreda. Različite drvorede (nizove stabala) spremamo u novi niz, i to zovemo šumom.

a) Napišite funkciju `void lomStabala (int **x, int n)` koja prima šumu `x` te u svakom od njezinih `n` drvoreda vrši lom stabala. Pripazite na složenost!

b) Napišite funkciju `void sortirajSumu (int **x, int n)` koja sortira podatke drvoreda unutar neke šume u kojoj su svi drvoredi doživjeli lom stabala (to ne treba provjeravati) silazno po visini najnižeg stabla tog drvoreda (smijete pretpostaviti da će te sve visine biti jedinstvene).

Npr. za `x[0]={4,3,0}, x[1]={1,5,0}, x[2]={2,0}` (što predstavalja dva drvoreda s dva stabla i jedan s jednim stablom), stanje nakon loma stabala bi bilo `x[0]={3,3,0}, x[1]={1,5,0}, x[2]={2,0}`, a onda nakon sortiranja `x[0]={3,3,0}, x[1]={2,0}, x[2]={1,5,0}`.
Zadatak nosi 15 bodova, međutim ukoliko se umjesto algoritma sortiranja složenosti $\mathcal{O}(n \log n)$ koristi algoritam složenosti $\mathcal{O}(n^2)$, zadatak nosi maksimalno 10 bodova. Korištenje quickSort algoritma iz biblioteke `stdlib.h` nosi dodatnih 5 bodova, odnosno tako riješen zadatak u sumi nosi 20 bodova.
**Napomena**: Uz standardnu biblioteku `stdio.h`, u ovome zadatku dozvoljeno je korištenje još jedino biblioteke `stdlib.h`.

```
#include<stdio.h>
#include<stdlib.h>

int min (int x, int y) {
    return (x<y) ? x : y;
}

void lomStabala (int **x, int n) {
    int i,j,len_i;
    for (i=0; i<n; i++) {
        len_i=0;
        while(x[i][len_i++]>0);
        len_i--;
        for (j=len_i-2; j>=0; j--)
            x[i][j]=min(x[i][j],x[i][j+1]);
    }
}

// pomoćna funkcija za vlastite implementacije sortova
void swap(int **x, int **y){
    int *tmp = *x;
    *x = *y;
    *y = tmp;
}

// vlastita implementacija klasičnog sorta
void classicSort(int **niz, int n) {
    int i,j;
    for(i = 0; i < n; ++i){
        for(j = i+1; j < n; ++j){
            if(niz[i][0] > niz[j][0]){
               swap(&niz[i], &niz[j]);
            }
```

```
        }
    }
}

// vlastita implementacija quicksorta
void quickSort(int **niz, int low, int high){
    int i, j;
    if(low < high){
        i = low + 1; j = high;
         while(i <= j){
             while(i <= high && (niz[i][0] < niz[low][0])) ++ i;
             while(niz[j][0] > niz[low][0]) -- j;
             if(i < j) {
                 swap(&niz[i], &niz[j]);
                 }
         }

         if(low < j) swap(&niz[low], &niz[j]);

         quickSort(niz, low, j-1);
         quickSort(niz, j+1, high);
    }
}

// pomoćna funkcija za gotovu funkciju qsort
int comparator(const void *a, const void *b){
    int* nizi = *(int**)a;
    int* nizj = *(int**)b;

    return (nizj[0] - nizi[0]);
}

// pozivi odgovarajućih sort funkcija
void sortirajSumu(int **niz, int n) {
    classicSort(niz,n);                         // Opcija 1 za max 10 bodova
    quickSort(niz,0,n-1);                       // Opcija 2 za max 15 bodova
    qsort(niz, n, sizeof(int*), comparator);    // Opcija 3 za max 20 bodova
}
```

JMBAG

IME I PREZIME

# Programiranje 2 – kolokvij, 3. 5. 2024.

**Zadatak 4** (4+9=13 bodova) Napišite funkciju `char* palindrom(char* s, int* br)` koja prima string te vraća podstring najveće duljine koji je palindrom čija je duljina barem 3. Ako takvih ima više, funkcija vraća onaj podstring koji je najmanji po leksikografskom uređaju. Ako polazni string ne sadrži podstring koji je palindrom funkcija vraća prazan string. Za string kažemo da je palindrom ako je invertirani string jednak početnom do na mala i velika slova (npr. string `"abBa"` je palindrom, dok `"abca"` nije). Dodatno, preko varijabilnog argumenta funkcija vraća broj podstringova duljine veće ili jednake 3 koji su palindromi.
**Napomena.** Možete definirati dodatne (pomoćne) funkcije, ali **dodatni nizovi nisu dopušteni**. Smijete koristiti funkcije iz `ctype.h`, `string.h` i `stdlib.h`. Možete pretpostaviti da funkcija `palindrom` prima string duljine barem 3.

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>

int check_palindrom(char* s, int i, int j){
    int pal=1;
    while(i<j){
        if(tolower(s[i])!=tolower(s[j])){
            pal=0;
            break;
        }
        i++;
        j--;
    }
    return pal;
}

char* palindrom(char* s, int* br){
    *br=0;
    int i, j, k;
    int max=0;
    int n=strlen(s);
    int smax_i;
    for(i=0 ; i<=n-3 ; i++){
        for(j=i+2 ; j<n ; j++){
            if(check_palindrom(s, i, j)){
                (*br)++;
                if(j-i+1>max){
                    max=j-i+1;
                    smax_i=i;
                }
                else if(j-i+1==max){
                    int par=0;
                    for(k=0 ; k<max ; k++){
                        if(s[i+k]<s[smax_i+k]){
                            par=1;
                            break;
                        }
                    }
                    if(par==1){
                        smax_i=i;
                    }
                }
            }
        }
    }
```

```
    char* smax=(char*)malloc((max+1)*sizeof(char));
    if(max>0){
        for(k=0 ; k<max ; k++){
            smax[k]=s[smax_i+k];
        }
    }
    smax[max]='\0';
    return smax;
}
```