Elektrotehnički fakultet Sarajevo
Odsjek za računarstvo i informatiku
Predmet: Algoritmi i strukture podataka

Vježba 1

Cilj vježbe 1 je upoznavanje studenata sa primjenom najčešće korištenih notacija za izražavanje vremenske kompleksnosti kao što su *big*-O, *big*-Ω i *big*-Θ. Studenti će na vježbama analizirati vremensku kompleksnost na odgovarajućim primjerima.

Pripremni zadatak

Napraviti funkciju sa sljedećim prototipom: *void Goldbach (int n, int& p, int& q)* koja testira Goldbachovu hipotezu za broj *n*. Goldbachova hipoteza glasi da se svaki paran broj *n* veći od 2 može napisati kao zbir dva prosta broja *p* i *q*. Ova hipoteza nikada nije dokazana ali nije ni opovrgnuta, odnosno nije pronaĎen broj *n* za koji ne važi, tako da se smatra da je tačna. Dakle, vaša funkcija treba odrediti brojeve *p* i *q* koji su prosti i čiji je zbir *n* i upisati ih u primljene reference. Ukoliko primljeni broj *n* nije paran ili nije veći od 2 funkcija treba baciti izuzetak. Osim toga, treba baciti izuzetak i u slučaju da Goldbachova hipoteza nije tačna. Napravite i kraću main() funkciju koja omogućava unos broja *n*, poziva napisanu funkciju Goldbach i ispisuje na ekranu *p* i *q*, te hvata sve bačene izuzetke.

Diskusija pripremnog zadatka

U pripremnom zadatku se tražilo da napravite funkciju za testiranje *Goldbachove* hipoteze.

a) Koliko dugo se izvršava vaša funkcija?

Da bismo izmjerili vrijeme izvršavanja funkcije možemo koristiti funkciju clock() iz biblioteke ctime (koja se u C-u zove time.h). Funkcija clock() vraća broj "otkucaja sata" od pokretanja programa. Pošto otkucaj sata ne traje jednako dugo na svakom računaru, koristi se konstanta CLOCKS_PER_SEC da se odredi vrijeme izvršavanja u sekundama. Mi ćemo koristiti milisekunde koje će nam dati preciznije vrijeme izvršenja funkcije. Na primjer, nakon što smo dodali #include <ctime> u zaglavlje programa, možemo pisati ovako:

```
clock_t vrijeme1 = clock();
Goldbach(n,p,q);
clock_t vrijeme2 = clock();
int ukvrijeme = (vrijeme2 - vrijeme1) / (CLOCKS_PER_SEC / 1000);
cout << "Vrijeme izvrsenja: " << ukvrijeme << " ms." << endl;
```

Prilikom testiranja, za **n** unosite vrijednosti: 10, 100, 1000, 10000, itd. dok ne dođete do neke dovoljno velike vrijednosti za koju se vaš program izvršava oko 2 sekunde.

b) Pokušajte eksperimentalno odrediti vremensku kompleksnost vašeg rješenja. Umjesto prethodno određnog broja za **n**, unesite broj koji je dva puta veći, a zatim i broj koji je dva puta manji. Svaki put zabilježite vrijeme izvršavanja. Sada pogledajte standardne klase kompleksnosti i uporedite ih sa onim što ste zabilježili. Na primjer, ako se za dva puta veći **n**, program izvršava četiri puta duže, riječ je o kvadratnoj kompleksnosti i slično. Na sličan način bi se mogla procijeniti i memorijska kompleksnost. Nažalost, mjerenje iskorištenja memorije nije jednostavno kao mjerenje vremena izvršavanja.

c) Prodiskutujte različita rješenja zadatka sa tutorom. Zašto su neka rješenja brža od drugih? Može li se za dato rješenje zadatka odrediti matematička zavisnost $f_{mem}(n)$ koja daje ukupnu zauzetu memoriju u bajtima za unesenu vrijednost n? Mogu li se odrediti zavisnosti $f_{op}(n)$ koje određuju broj pojedinih elementarnih operacija (sabiranje, dijeljenje, stavljanje u niz) ovisno o unesenom broju n?

d) Sada probajte prepraviti svoj program tako da se koristi najefikasniji algoritam.

Zadaci za samostalnu vježbu

Zadatak 1. Koja je vremenska kompleksnost algoritma koji pronalazi sumu svih faktorijela od 1 do n (1!+2!+3!....n!)? Na koji način je moguće popraviti vremensku kompleksnost datog koda i kolika će vremenska kompleksnost biti nakon popravke?

```
int s=0;int fact=1;
for(int i = 1; i <= n; i++){
    fact = 1;
    for (int j = 1; j <= i; j++)
    {
       fact = fact * j;
    }
    s+=fact;
}
return s;
```

Zadatak 2. Koja je vremenska kompleksnost sljedećeg algoritma?

```
int s=0;
for( int i = n; i > 0; i-=2) {
    for( int j = 0; j < 2*n; j++ ) {
        for( int k = 0;k < n; k++){
```

```
                    s+=k;
                }
            }
    }
}
```

Zadatak 3. Koja je vremenska kompleksnost sljedećeg algoritma?

```
int s=0;
for( int i = n; i > 1; i /= 2 ) {
    for( int j = 1; j < n; j *= 2 ) {
        for( int k = 0; k < n; k += 2 ) {
            s+=k;
        }
    }
}
```

Zadatak 4. Napisati funkciju

```
bool permutacija(std::string s1, std::string s2)
```

koja prima dva stringa i vraća true ukoliko postoji neka permutacija[1] prvog stringa koja je jednaka nekoj permutaciji drugog stringa. U suprotnom, funkcija treba da vrati `false`.

Zadatak 5. Napisati funkciju koja za ulazno n daje sve proste brojeve do n. Napisati funkciju tako da vremenska kompleksnost bude što manja.

Zadatak 6. Napisati funkciju koja prima vektor vektora cijelih brojeva i popravlja ga tako da maksimalni broj elemenata u svakom vektoru bude minimalan a minimalni broj elemenata u svakom vektoru bude maksimalan. Napisati funkciju tako da vremenska kompleksnost bude što manja.

Zadaci za vježbu:

Zadatak 1. Koja je vremenska kompleksnost algoritma koji pronalazi element **x** u nizu sortiranom u opadajućem redoslijedu metodom polovljenja intervala (binarna pretraga):

```
bool pretraga(int niz[], int vel, int x) {
    float pos(float(vel)/2), interval(pos);
    int i;
    do {
        interval = interval / 2;
        i=pos; // Zaokruzivanje
```

```
        if (niz[i]==x) return true;
```

[1]Permutacija stringa se iz početnog stringa dobije zamjenom mjesta proizvoljnog broja karaktera.

```
        if (niz[i]<x) pos=pos-interval;
        else pos=pos+interval;
    } while (interval>=0.5);
    i=pos;
    if (niz[i] == x) return true;
    return false;
}
```

Zadatak 2. Koja je vremenska kompleksnost sljedećeg algoritma:

```
int sum = 0;
for (int n = N; n > 0; n /= 2)
    for (int i = 0; i < n; i++)
        sum++;
```

Zadatak 3. Definisana je struktura Student na sljedeći način:

```
struct Student {
    std::string ime, prezime;
    int brojIndexa;
};
```

Napišite funkciju koja prima vektor studenata i vraća true ako se u vektoru nalaze dva studenta sa istim imenom i prezimenom, a u suprotnom vraća false.

```
bool istoImePrezime(std::vector<Student> studenti);
```

Zadatak 4. Napisati funkciju sa sljedećim prototipom:

```
bool obrnuta(std::vector<std::vector<std::vector<int> > >& vm)
```

Vektor **vm** je vektor matrica cijelih brojeva. Ovaj vektor se sastoji od *n* matrica koje su sve kvadratne matrice dimenzija *mxm* (nije potrebno provjeravati). Članovi matrica su pozitivni cijeli brojevi u opsegu od 0 do maksimalne vrijednosti koja se može upisati u promjenjivu tipa int.

Funkcija treba vratiti logičku vrijednost true ako u vektoru postoje dvije matrice čiji su svi članovi sastavljeni od istih cifara, a u suprotnom treba vratiti false. Na primjer, posmatrajmo matrice A i B:

$$A=\begin{bmatrix} 15 & 3 & 25 \\ 311 & 121 & 2 \\ 5 & 21 & 3112113 \end{bmatrix}, \quad B=\begin{bmatrix} 421 & 51 & 355 \\ 15 & 3 & 25 \\ 1 & 22 & 55355 \end{bmatrix},$$

Elementi matrice A su sastavljeni od cifara 1, 2, 3 i 5, dok se u matrici B javlja još i cifra 4 tako da ove dvije matrice ne zadovoljavaju uslov zadatka. Kada bi element u gornjem lijevom uglu matrice B glasio 21, onda bi i elementi matrice B bili sastavljeni od cifara 1, 2, 3 i 5 pa bi uslov zadatka bio zadovoljen.

Zadatak 5. Napisati funkciju koja prima matricu **N**x**N** cijelih brojeva te u njoj pronalazi *lokalni minimum*. Lokalni minimum matrice se definiše kao element **a[i][j]** koji ispunjava uslove:

a[i][j] <= a[i+1][j]
a[i][j] <= a[i][j+1]
a[i][j] <= a[i-1][j]
a[i][j] <= a[i][j-1]

Članovi na rubu matrice ne smatraju se za lokalni minimum. Funkcija treba upisati koordinate **i** i **j** u primljene reference. Ako matrica ima više lokalnih minimuma, treba vratiti bilo koji od njih. Ako matrica nema niti jedan minimum, **i** i **j** postaviti na vrijednost -1.

Napomena: Ubuduće će se na ovom predmetu podrazumijevati da programsko rješenje zadatka treba imati minimalnu vremensku kompleksnost uz minimalno zauzeće memorije, ako nije u zadatku naglašeno drugačije. Nekada se može desiti da nije moguće postići oboje. U takvim slučajevima vremenska kompleksnost će imati veći prioritet u odnosu na memorijsku.