Algoritmi i strukture podataka

1. **Induktivno – rekurzivna konstrukcija – pojam, ilustrovanje kroz primere**

Osnovni pristup konstrukcije algoritama je tzv. induktivni tj. rekurzivni pristup. On u svom osnovnom obliku podrazumeva da se resenje problema vece dimenzije pronalazi tako sto umemo da resimo problem istog oblika, ali manje dimenzije i da od resenja tog problema dobijemo resenje problema vece dimenzije.

* Implementacija algoritma moze biti takva da promenljive unutar petlje iterativno azuriraju svoje vrednosti krenuvsi od vrednosti koje predstavljaju resenja elementarnih problema, pa do krajnjih vrednosti koje predstavljaju resenja zadatog probelma. Posto je ovo prilicno slicno principu matematicke indukcije, kazemo da je algoritam definisan induktivno.

int zbir = 0;

for(int i = 0; i < a.size(); i++)

zbir += a[i];

* Implementacija moze biti takva da funkcija koja resava polazni problem sama sebe poziva da bi resila problem istog oblika, ali manje dimenzije i tada kazemo da je algoritam definisan rekurzivno.

int zbir(int a[], int n){

    if(n == 0)

        return 0;

    return zbir(a, n - 1) + a[n-1];

}

1. **Invarijante petlja: pojam, ilustrovanje kroz primere**

Jedan od osnovnih pojmova u analizi i razumevanju iterativnih programa su invarijante petlje. To su logicki uslovi koji vaze neposredno pre petlje, zatim nakon svakog izvrsavanja naredbi u telu petlja i neposredno nakon izvrsavanja cele petlje. Korisne invarijante su one koje garantuju korektnost algoritma koji ta petlja implementira. Invarijante sustinski opisuju znacenje svih promenljivih unutar petlje.

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int minNiza(const vector<int> &a){

    int m = a[0];

    for(int i = 1; i < a.size(); i++)

        m = min(m, a[i]);

    return m;

}

int main(){

    vector<int> a{3, 5, 4, 1, 6, 2, 7};

    cout << minNiza(a) << endl;;

    return 0;

}

1. **Slozenost – vrste slozenosti, asimptotska analiza**

Vazno pitanje za prakticnu primenu napisanih programa je to koliko resursa program zahteva za svoje izvrsavanje. Najvazniji resursi su sigurno vreme potrebno za izvrsavanje programa i zauzeta memorija. Dakle, obicno se razmatraju :

* Vremenska slozenost algoritma;
* Prostorna (memorijska) slozenost algoritma.

Asimptotska analiza se bavi odredjivanjem kako vreme izvrsavanja ili memorijski zahtevi algoritma rastu u odnosu na velicinu ulaznih podataka. Glavni cilj asimptotske analize slozenosti je razumevanje performansi algoritma u najekstremnijim slucajevima, kada ulazni podaci postaju izuzetno veliki. Asimptotska analiza obicno se izrazava pomocu notacija kao sto su O, Ω i Θ koje opisuju gornje, donje i tacno ogranicenje izrazeno u odnosu na funkciju velicine ulaza. Ova analiza omogucava inzenjerima racunarskih sistema da predvide kako ce se njihovi algoritmi ponasati u praksi i omogucava im da donose odluke o izboru algoritama za razlicite zadatke, uzimajuci u obzir vreme izvrsavanja i potrebne resurse.

1. **Matematicke osnove izracunavanja slozenosti – osnovne formule**

Aritmeticki niz:

Geometrijski niz:

Stepene sume:

1. **Klase slozenosti**

Karakteristike osnovnih klasa slozenosti:

* - konstantna slozenost, algoritmi linijsko – razgranate strukture koji se izvrsavaju prakticno momentalno, npr. algoritmi u kojima se implementira neka matematicka formula;
* – logaritamska slozenost, izuzetno efikasno, npr. binarna pretraga;
* – korenska slozenost, “logaritam za one sa jeftinijim ulaznicama” – nemamo najbolja mesta, ali ipak mozemo da gledamo utakmicu, npr. ispitivanje da li je broj prost, faktorizacija broja na proste cinioce;
* – linearna slozenost, optimalno, kada je za resenje potrebno pogledati ceo ulaz, npr. minimum/maksimum serije elemenata;
* – kvazilinearna slozenost, “linearni algoritam za one sa jeftinijim ulaznicama”, efikasni algoritmi zasnovani na dekompoziciji, efikasnom sortiranju, koriscenju struktura podataka sa logaritamskim vremenom pristupa;
* – kvadratna slozenost, obicno ugnezdjene petlje, npr. sortiranje umetanjem;
* – kubna slozenost, obicno visestruko unezdjene petlje, npr. mnozenje matrica;
* – eksponencijalna slozenost, izuzetno neefikasno, npr. ispitivanje svih podskupova;
* – faktorijelna slozenost, izuzetno neefikasno, npr. ispitivanje svih permutacija.

1. **Slozenost nekih cestih oblika petlji**

Slozenost sledece petlje je :

for(int i = 0; i < n; i++)

    //kod slozenosti O(1)