ZADACI ZA VJEŽBU

a) Napišite rekurzivnu funkciju brojpojavljivanja koja za zadano cjelobrojno polje i zadanu vrijednost vraća broj elemenata polja koji imaju tu vrijednost.

b) Napišite funkciju koja stvara novo polje koje sadrži samo elemente ulaznog polja koji se pojavljuju točnoN puta (svaka vrijednost u novom polju smije se pojaviti samo jedno i predak nije bitan):

int \* vratiVisestruke(int \*ulaz, int brelUl, int n, int \*brelIz);

Primjer: za ulazno polje: 6, 2, 3, 4, 2, 2, 4, 6, 4, 3 i N=2 izlazno polje je: 6, 3

Napišite i glavni program u kojem definirajte cjelobrojno polje od 20 elemenata, a zatim ga napunite slučajno odabranim cijelim brojevima u intervalu [10, 50], pozovite funkciju, ispišite dobiveno polje te oslobodite dinamički stvorenu memoriju.

#include <stdio.h>

#define DG 10

#define GG 50

#define N 2

int brojPojavljivanja (int element, int \*polje, int br){

int brojac=0;

if (br>0){

if (polje[0]==element) brojac=1;

return brojac + brojPojavljivanja (element, polje+1, br-1);

}

else

return brojac;

}

int \* vratiVisestruke(int \*ulaz, int brelUl, int n, int \*brelIz){

int \*novi;

int i, brojac = 0;

novi = (int\*) malloc(brelUl \* sizeof(int));

for(i=0; i<brelUl; i++){

//pogledati koliko se puta u ulaznom polju javlja element i ako se javlja N puta a do sada već nije unesen dodaj ga na ulaz

if (brojPojavljivanja(ulaz[i], ulaz, brelUl)== n && brojPojavljivanja(ulaz[i], novi, brojac )== 0)

novi[brojac++]=ulaz[i];

}

novi = (int\*) realloc(novi, brojac \* sizeof(int));

\*brelIz = brojac;

return novi;

}

int main() {

int i, breleme;

int \*polje, \*novi;

srand((unsigned)time(NULL));

polje = (int\*) malloc(20 \* sizeof(int));

for (i=0; i<20; i++){

polje[i]= rand()%(GG-DG+1) + DG;

printf("%d ", polje[i]);

}

printf("\n");

novi = vratiVisestruke(polje, 20, N, &breleme);

for (i=0; i<breleme; i++){

printf("%d ", novi[i]);

}

free(polje);

free(novi);

return 0;

}

***Složenost***

1. Odredite apriornu složenost (u ovisnosti o **m** i **n**) sljedećih programskih odsječaka i detaljno obrazložite svoje odgovore.

1. for(i=0;i<n;i++) {

m=2\*n;

while(m>0){

suma+=m;

if(m%5==0){

m=m/3;

}

m--;

}

}

1. for(i=1; i<n; i++){

for(j=0; j<n; j++){

x=x+j;

i=i\*2;

}

}

1. Vanjska petlja obavlja se n puta. Unutarnja dijeli 2n sa 3, ali ne u svakom koraku nego otprilike svaki peti put. Broj ponavljanja unutarnje petlje prema tome nije veći od 5\*log3(2n). Iz toga slijedi da je ukupna složenost O(n\*log(n)).
2. Nakon prvog izvođenja unutarnje petlje varijabla i biti veća od n što je uvjet prekida vanjske petlje. Zato je složenost programskog odsječka jednaka složenosti unutarnje petlje i iznosi O(n)

2. Odredite apriornu složenost sljedećih programskih odsječaka i detaljno obrazložite svoje odgovore.

1. int Funkcija1(int \*mat, int n, int maxstu){

int i, j, suma = 0;

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

for (j = 0; j < n; j++){

printf("%d\n", mat[i\*maxstu + j]);

if (i + j == 2) suma += mat[i\*maxstu + j];

}

return suma;

}

1. int Funkcija2(int \*mat, int n, int maxstu){

int i, j;

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

for (j = 0; j < n; j++){

printf("%d\n", mat[i\*maxstu + j]);

if (i + j == 2) return mat[i\*maxstu + j];

}

}

1. int Funkcija3(int \*mat, int n, int maxstu){

int i, j;

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

for (j = 0; j < n; j++){

printf("%d\n", mat[i\*maxstu + j]);

if (i == 2) return mat[i\*maxstu + j];

}

}

1. Bez obzira koji je rezultat naredbe (i + j == 2) program će proći kroz cijelu matricu koja ima n2 elemenata, pa je zato složenost O(n2).
2. Uvjet (i + j == 2) prvi put je ispunjen točno kada je i = 0 i j = 2. Do tada smo samo 2 puta prošli naredbu if (i + j == 2) suma += mat[i\*maxstu + j];, a treći put izlazimo iz petlji.

Složenost je 6 brojimo li naredbu printf(...) i if(...), dakle konstantna, odnosno O(1).

1. Uvjet (i == 2) prvi put je ispunjen točno kada je i = 2 i j = 0. Do tada smo samo 2\*n puta prošli naredbu if (i + j == 2), a onda izlazimo iz petlji.

Složenost je 2\*n\*2 brojimo li naredbu printf(...) i if(...), odnosno O(n).

***Hash***

1. Jedan zapis datoteke organizirane po načelu raspršenog adresiranja definiran je strukturom:

typedef struct{

int sifra;

char naziv[50+1];

double cijena;

} zapis;

Zapis je prazan ako je na mjestu šifre vrijednost nula. Parametri za raspršeno adresiranje nalaze se u datoteci parametri.h i oni su:

- BLOK..........veličina bloka na disku

- MAXZAP........broj zapisa

- C.............broj zapisa u jednom pretincu

- M.............broj pretinaca

Ključ zapisa je šifra artikla, a transformacija ključa u adresu obavlja se zadanom funkcijom:   
  
int adresa(int sifra);

Napišite funkciju koja će pronaći i vratiti prvi zapis koji se ne nalazi u onom pretincu u kojem je po svojoj šifri trebao biti (zbog preljeva). Funkcija preko svog imena vraća 1 ili 0 ovisno o tome postoji li takav zapis. Prototip funkcije je:

int pronadji(FILE \*f, zapis \*z);

int pronadji(FILE \*f, zapis \*z) {

int i, j;

for (i = 0; i < M; i++) {

fseek (f, i\*BLOK, SEEK\_SET);

fread (pretinac, sizeof (pretinac), 1, f);

for (j = 0; j < C; j++) {

if (pretinac[j].sifra != 0) {

if (adresa(pretinac[j].sifra) != i){

\*z = pretinac[j];

return 1;

}

}

}

}

return 0;

}

2. Jedan zapis tablice raspršenih adresa sadrži šifru djelatnika (long), prezime i ime (40+1 znak) i spol (1 znak). Veličina bloka na disku je BLOK. Prazni zapis sadrži šifru jednaku nuli. Očekuje se do 1000 djelatnika, a tablica je dimenzionirana za 20% veći kapacitet od očekivanog. Za slučaj preljeva koristio se ciklički susjedni pretinac. Napisati naredbe #define kojima se određuju parametri raspršenog adresiranja. Napisati funkciju koja će prebrojati koliko ima punih pretinaca. Prototip funkcije je:

int broji (char \*datoteka);

Rješenje:

struct s {

long sifra;

char imePrezime[40+1];

char spol;

};

typedef struct s zapis;

#define N 1000

#define C ((int) (BLOK / sizeof (struct zapis)))

#define M ((int) (N / C \*1.2))

int broji(char \*datoteka) {

int i, j, br = 0; /\* br – broj punih zapisa \*/

zapis pretinac[C];

FILE \*f = fopen(datoteka, "rb");

for (i = 0; i<M; i++) {

fseek(f, i\*BLOK, SEEK\_SET);

fread(pretinac, sizeof(pretinac), 1, f);

for (j = 0; j < C; j++)

if (pretinac[j].sifra == 0) break; /\* prazan zapis \*/

if (j == C) br++; /\* ako je pretinac popunjen \*/

}

return br;

}

3.Jedan zapis datoteke organizirane po načelu raspršenog adresiranja definiran je sljedećom strukturom:

typedef struct{

int sifra;

char naziv[50+1];

int kolicina;

float cijena;

} zapis;

Zapis je prazan ako je na mjestu šifre vrijednost nula. Parametri za raspršeno adresiranje nalaze se u datoteci parametri.h i oni su:

* BLOK : veličina bloka na disku
* MAXZAP : očekivani maksimalni broj zapisa
* C : broj zapisa u jednom pretincu
* M : broj pretinaca

Ključ zapisa je šifra artikla, a transformacija ključa u adresu obavlja se zadanom funkcijom:

int adresa(int sifra);

Napisati funkciju koja će pronaći zapis koji je najviše „udaljen“ od predviđenog pretinca. Ako pojedini zapis nije spremljen kao preljev, njegova je udaljenost 0; inače se udaljenost definira kao broj dodatnih pretinaca koje je potrebno pročitati da bi se zapis pronašao. (Primjerice, ako je M=15, adresa nekog zapisa 13, a zapis se nalazi u pretincu broj 4, udaljenost je 6). Funkcija vraća 0 ako nijedan zapis nije zapisan kao preljev; inače vraća najveću udaljenost. Ako postoji više takvih zapisa, vratiti bilo koji. Funkcija treba imati prototip:

int max\_udaljenost(FILE \*f, zapis \*z);

Rješenje:

int max\_udaljenost(FILE \*f, zapis \*z){

zapis pretinac[C];

int i, j;

int udaljenost, max = 0;

for (i = 0; i < M; i++) {

fseek (f, i\*BLOK, SEEK\_SET);

fread (pretinac, sizeof (pretinac), 1, f);

for (j = 0; j < C; j++) {

if (pretinac[j].sifra != 0) {

/\* Ako zapis nije prazan \*/

if (adresa(pretinac[j].sifra) != i){

udaljenost = i - adresa(pretinac[j].sifra);

if (udaljenost < 0) udaljenost += M;

if (udaljenost > max){

\*z = pretinac[j];

max = udaljenost;

}

}

}

}

}

return max;

}

***Rekurzija***

1. Niz brojeva definiran je rekurzivno na slijedeći način:

f0 = 1,

f1 = 2,

fn = (fn-1 + 1)\* fn-2 , za n > 1.

Napisati rekurzivnu funkciju koja će izračunati n-ti član niza. Odrediti apriornu složenost funkcije. Funkcija mora imati prototip:

long f(int n);

Rješenje:

long f(int n){

if (n < 2){

return (n + 1); // rješavamo oba slucaja

}else{

return (f(n-1) + 1) \* f(n-2);

}

}

2. Zadana je funkcija max\_elem koja vraća indeks najvećeg elementa u cjelobrojnom polju.

Napisati **rekurzivnu** funkciju sort\_rek koja će, koristeći funkciju max\_elem, urediti polje po veličini.

Prototipi: int max\_elem(long \*polje, int N);

void sort\_rek(long \*polje, int N);

Rješenje:

void sort\_rek(long \*polje, int N) {

int max;

long pom;

/\* polje od jednog elementa \*/

if (N == 1) return;

/\* tražim najveći element \*/

max = max\_rek(polje, N);

/\* stavljam najvećeg na prvo mjesto \*/

pom = polje[max];

polje[max] = polje[0];

polje[0] = pom;

/\* rekurzivno se pozivam za ostatak polja \*/

sort\_rek(polje+1, N-1);

}

3. Napisati rekurzivnu funkciju koja će izračunati istu sumu reda kao i zadana funkcija **f**:

double f(int n) {

double suma = 0;

int i;

for (i = 1; i < n; i++)

suma += 1./(i \* (i + 1) \* (i + 2));

return suma;

}

Rješenje:

double f(int n){

--n;

if (n<1) return 0;

return 1./(n\*(n+1)\*(n+2)) + f(n);

}

ili

double f(int n){

if (n<=1) return 0;

return 1./((n-1)\*(n)\*(n+1)) + f(n-1);

}

***Sortovi***

Zadan je niz brojeva: **1, 5, 7, 4, 3, 6, 8, 2, 9, 0**

Ilustrirajte uzlazno sortiranje zadanog niza brojeva (ispišite niz nakon svake zamjene dvaju elemenata):

a) (*2 boda*) Algoritmom **shell sort** za slijed koraka {4, 2, 1}. Kako izgledaju 4-, 2- i 1-sortirani nizovi?

b) (*2 boda*) Algoritmom **quicksort**. Stožer odaberite metodom aproksimacije medijana temeljem početnog, krajnjeg i središnjeg člana polja.

*Rješenje:*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a) |  |  |  |  |  |  |  |  |  |  | b) |  |  |  |  |  |  |  |  |  |  |
| 1 | 5 | 7 | **4** | 3 | 6 | 8 | **2** | 9 | 0 |  | **1** | 5 | 7 | 4 | **3** | 6 | 8 | 2 | 9 | **0** |  |
| 1 | 5 | 7 | 2 | 3 | **6** | 8 | 4 | 9 | **0** |  | 0 | 5 | 7 | 4 | **1** | 6 | 8 | 2 | **9** | 3 |  |
| 1 | **5** | 7 | 2 | 3 | **0** | 8 | 4 | 9 | 6 |  | 0 | **5** | 7 | 4 | 9 | 6 | 8 | 2 | **1** | 3 |  |
| **1** | 0 | **7** | 2 | 3 | 5 | 8 | 4 | 9 | 6 |  | j | i |  |  |  |  |  |  |  |  |  |
| 1 | 0 | **7** | 2 | **3** | 5 | 8 | 4 | 9 | 6 |  | 0 | 1 | **7** | 4 | 9 | **6** | 8 | 2 | 5 | **3** |  |
| 1 | 0 | 3 | 2 | 7 | **5** | 8 | **4** | 9 | 6 |  |  |  | 3 | 4 | 9 | **6** | 8 | 2 | **5** | 7 |  |
| **1** | **0** | 3 | 2 | 7 | 4 | 8 | 5 | 9 | 6 |  |  |  | 3 | 4 | **9** | 5 | 8 | **2** | 6 | 7 |  |
| 0 | 1 | **3** | **2** | 7 | 4 | 8 | 5 | 9 | 6 |  |  |  |  |  | i |  |  | j |  |  |  |
| 0 | 1 | 2 | 3 | **7** | **4** | 8 | 5 | 9 | 6 |  |  |  | 3 | 4 | 2 | 5 | **8** | 9 | **6** | 7 |  |
| 0 | 1 | 2 | 3 | 4 | 7 | **8** | **5** | 9 | 6 |  |  |  |  |  |  | j | i |  |  |  |  |
| 0 | 1 | 2 | 3 | 4 | **7** | **5** | 8 | 9 | 6 |  |  |  | 3 | **4** | **2** | 5 | **6** | 9 | 8 | 7 |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 7 | 8 | **9** | **6** |  |  |  | 3 | 2 | 4 | 5 |  |  |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 7 | **8** | **6** | 9 |  |  |  |  | j | i |  |  |  |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | **7** | **6** | 8 | 9 |  |  |  |  |  |  |  |  | 7 | 8 | 9 |  |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |  |  |  |  |  |  |  |  |  |  |  |  |