ASP 1. mass instrukcije - 2013/14

Rad s datotekama

preskoceno na massovnima, ovaj tutorial vam je dovoljan
http://materijali.fer2.net/File.5927.aspx

Dinamicko alociranje memorije

void* malloc (size_t size);

- Rezervira polje velicne size (u bajtovima)
- Vraca null pointer ako ne moze alocirati memoriju

```
int a = malloc(sizeof(int));          //nije dobro, tip podataka mora biti pokazivac
int *a = malloc(sizeof(int*));        //nije dobro, rezervira memoriju velicine int* (~4 bajta??)
int *a = malloc(sizeof(int));         //nije dobro, malloc vraca void*, treba cast
int *a = (int*)malloc(sizeof(int));   //ispravno!
```

```
void f()
{
   int *a = (int*)malloc(sizeof(int));
   return;
} //nije dobro, prije izlaza iz funkcije treba osloboditi memoriju
// u suprotnom nastaje memory leak
```

```
void f()
{
   int *a = (int*)malloc(sizeof(int));
   free(a);
   return;
} //sad je dobro
```

void* realloc(void* ptr, size_t size);

- povecava/smanjuje memoriju rezerviranu memoriju

```
int *p = (int*)realloc(p, sizeof(int));  //ispravno, vrijede iste stvari kao i za malloc
```

- u slucaju da se kao prvi argument stavi NULL, ponasa se kao malloc

```
int *p = (int*)realloc(NULL, sizeof(int));  ==  int *p = (int*)malloc(sizeof(int));
```

void* calloc (size_t num, size_t size);

- rezervira i inicijalizira memoriju (na 0)
- prvi argument: koliko elemenata se inicijalizira
- drugi argument: velicina pojedinog elementa (u bajtovima)

Slozenosti

a priori
- odreduje se prije izvodenja programa, analizom koda
- 3 vrste
  - **O**( ) – O notacija – najgori slucaj
    - zanemaruju se konstante
  - **Ω**( ) – Omega notacija – najbolji slucaj
  - **Θ**( ) – Theta notacija – prosjecan slucaj
    - ne zanemarujemo konstante

a posteriori
- dobivena mjerenjem na računalu

```
for (i = 0; i < n; ++i)
   if (a[i] == x)
      return i;
```
O(n) – u slucaju da x nije u polju ili se nalazi na zadnjem mjestu
Ω(1) – ako je x odmah prvi clan //a[0]==x
Θ(n/2)

```
for (i = 0; i < n; ++i)
   for (j = 0; j < n; ++j)
      x++;
```
O(n^2)
Ω(n^2)
Θ(n^2)

```
for (i = 0; i < n^2; ++i)
   for (j = 0; j < n^2; ++j)
      x++;
```
O(n^4)
Ω(n^4)
Θ(n^4)

```
for (i = 0; i < n; ++i)
   printf("%d ", i);
```
O(n)
Ω(n)
Θ(n)

```
for (i = 0; i < n; ++i)
   for (j = 0; j < n; ++j)
   {
      printf("%d ", i);
      x++;
   }

for (i = 0; i < n; ++i)
   for (j = 0; j < n; ++j)
      printf("%d ", i);
```

u oba primjera slozenost je **O**(n^2) – zanemarujemo konstante

```
for (i = 0; i < n; ++i)
   return x;

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

oba primjera imaju konstantnu slozenost **O**(1) – izvodenje ne ovisi o n

```
for (i = 0; i < n; ++i)
   for (j = 0; j < n; ++j)
      x++;
```

O(n^2)
Ω(n^2)
Θ(n^2)

```
for (i = 0; i < n; ++i)
   for (j = i; j < n; ++j)
      x++;
```

O(n^2)
Ω(n)
Θ(n^2 / 2)

```
for (i = 0; i < n; ++i)
   for (j = i; j < n; ++j)
      if (a[i][j] == x) return 0;
```

O(n^2)
Ω(1) – naci cemo ga odmah na pocetku a[0][0] == x
Θ(n^2 / 4) – zbog petlji je n^2 / 2 i zbog uvjeta izlaska mnozi se s 1 / 2

```
for (i = 0; i < n; ++i);
   for (j = 0; j < n; ++j)
      printf("%d ", j);
```
O(n) – zasto? tocka zarez je nakon prve petlje => prva petlja ne radi nista

```
for (i = 0, n *= n; i < n; ++i)
   printf("%d ", i);
```
O(n^2)

zasto?
for (i = 0, n *= n; i < n; ++i)
je isto kao da smo napisali

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

ili

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

```
for (i = 0; i < 2n; ++i)
   printf("%d ", i);
```
O(n) – zanemarujemo konstante

```
for (i = 0; i < n; ++i)
   n += 1;
```
O(beskonacno) – petlja se nikad nece prekinut

```
for (i = 0; i < i; ++i)
   printf("%d ", i);
```
O(1) – petlja se izvede samo jednom (ili 0 puta? nisam siguran)

Pozivi funkcija i sistemski stog

kod poziva funkcija na stog se sprema okvir funkcije koji sadrzi

1. povratnu adresu na koju se treba vratiti nakon izvršenja pozvane funkcije
2. lokalne varijable funkcije
3. argumente (parametre) funkcije

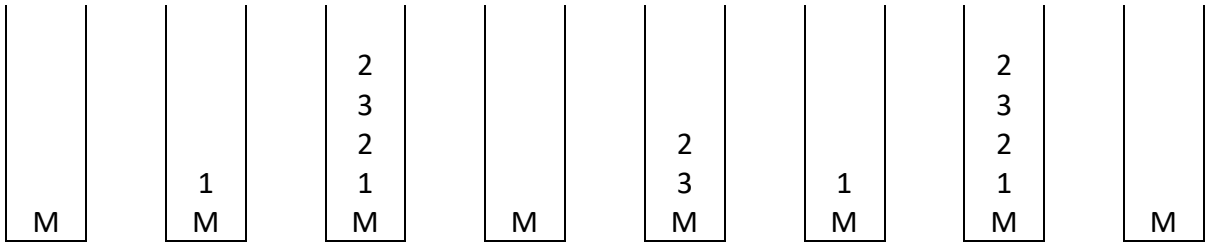
Zadatak: ucitati n brojeva, ispisati proste, ispisati dali je suma prostih brojeva prosta

```
int prost(int x)
{
   int i = 0;
   for (i = 2; i < x; ++i)
      if (x%i == 0) return 0;
   return 1;
}

int main()
{
   int n = 0, i = 0, suma = 0;
   scanf("%d", &n);
   for (i = 0; i < n; ++i)
   {
      scanf("%d", &x);
      if (prost(x))
      {
         printf("%d\n", x);
         suma += x;
      }
   }
   printf("Suma %s prosta\n", prost(suma) ? "je" : "nije");
   return 0;
}
```

prije pokretanja | kod poziva funkcije prost | nakon izlaska iz fun. prost | nakon izlaska iz programa

```
int f1(){ f2(); f3(); }
void f2(){ printf(...);}
int f3(){ f2(); }

int main()
{
   f1();
   f2();
   f3();
   return 0;
}
```

ovo mi se neda posebno crtat otprilike izgleda ovako
m – main
1 – f1()
2 – f2()
3 – f3()



Rekurzije

Rekurzija – funkcija koja poziva samu sebe, ukratko.

```
int suma(int n)
{
   if (n == 0) return 0;  //vazno je napisati uvjet za izlazenje iz rekurzije
   return suma(n - 1) + n;
} //zbroj n prirodnih brojeva
```
O(n)

```
int fibonnaci(int n)
{
   if (n <= 1) return 1;
   return fibonnaci(n - 1) + fibonnaci(n - 2);
}
```
O(2^n)

```
int zbroj(int x)
{
   if (x < 0) x = -x; //u slucaju da je predani broj negativan
   if (x == 0) return 0;
   return x % 10 + zbroj(x / 10);
}

int brojZnameknaka(int x, int z)
{
   if (x < 0) x = -x;
   if (x < 10) return x == z; // 1 ili nula - ovisno o tome da li je x==z
   if (x % 10 == z)
      return 1 + brojZnameknaka(x / 10, z);
   else
      return brojZnameknaka(x / 10, z);
   //umjesto if (x%10==z) i else moze se napisati samo
   //return x%10 == z + brojZnamenaka(x/10, z);
}
```
O(log x)

Moguce je imati rekurziju koja je dobro napisana, ali uzrokuje stack overflow (ako se JAKO puno puta poziva)

```
int traziMaxClanRekurzivno(int a[], int n, int *broj)
{
   if (n == 1) //u slucaju da polje ima samo 1 clan, najveci je upravo taj clan
   {
      *broj = 1;
      return a[0];
   }
   max = traziMaxClanRekurzivno(a + 1, n - 1, broj);
   //PAZI: ne pisati traziMaxClanRekurzivno(a + 1, n - 1, &broj);
   //jer se time salje pointer na pointer u funkciju koja ocekuje pointer
   if (a[0] < max) return max;
   if (a[0] == max) return max; //moze i return a[0]; svejedno je
   if (a[0] > max)
   {
      *broj = 1;
      return a[0];
   }
}//trazi najveci clan i koliko puta se pojavljuje u polju
```

Hash

Nacin zapisa podataka koji omogucava brzo pristupanje

	C							
0	gsogj	Jogpnjsp	g				Neiskoristen prostor	M
1			dsag					
2				asdg				
…					adggga			
	BLOK							

gluposti napisane unutra su zapisani podaci :D

par
N //broj stvari koje zelimo spremiti
BLOK //velicina bloka na hard disku
C = N / BLOK //kapacitet jednog pretinca
M = (N / C) * (1 + ..%) //broj pretinca

strukture pisemo ovako:

```
typedef struct ime
{
   char ...
      ...
} ime;
```

na ovaj nacin netrebamo pisati struct ime zapis; svaki put kad trebamo deklarirati strukturu

Algoritam za pisanje u hash

1. ucitaj podatke u program
2. spremi u tablicu
    1. trazi hash od kljuca
    2. smjesti podatke u tablicu
        1. ucitaj pretinac
        2. nadi prazno mjesto u pretincu
        3. stavi u pretinac
        4. zapisi pretinac u datoteku
    3. ako nije upisan kreni u novi pretinac

Algoritam za trazenje

1. ucitaj podatke iz tablice
    1. trazi hash od kljuca
    2. ucitaj pretinac koji odgovara hashu
    3. potrazi u pretincu
2. ako nije pronaden nastavi u novi pretinac

```
struct student
{
   char prezime[100 + 1];
   char ime[100 + 1];
   char jmbag[10 + 1];
} t, pretinac[C];

//za pisanje u tablicu
while (scanf("%s %s %10s", t.prezime, t.ime, t.jbmag))
{
   poc = i = hash(t.jmbag);
   do
   {
      fseek(ft, i*BLOK, SEEK_SET); //postavi se na pocetak pretinca
      fread(pretinac, sizeof(pretinac), 1, ft); //ucitaj pretinac
      for (j = 0; j < C; ++j)
         if (pretinac[j].jmbag[0] == '\0') //provjeri dal je zapis prazan
         {
            pretinac[j] = t; //spremi ucitani zapis u pretinac
            fseek(ft, i*BLOK, SEEK_SET); //uvijek koristi pozicioniranje od
            //pocetka datoteke, jednostavnije je
            fwrite(pretinac, sizeof(pretinac), 1, ft);
            //fwrite(pretinac, sizeof(student)*C, 1, ft);
            //fwrite(pretinac, sizeof(student), C, ft);
            //bilokoje od te 3 se moze koristiti za upisivanje
            return 1;
         }
      i = (i + 1) % M; //ciklicki predi u sljedeci pretinac
   } while (i != poc);

}
```

```
//za citanje iz tablice, skoro sve isto kao i za citanje
do
{
   fseek(ft, i*BLOK, SEEK_SET);
   fread(pretinac, sizeof(pretinac), 1, ft);
   for (j = 0; j < C; ++j)
      if (pretinac[j].jmbag == trazeni.jmbag)
         return ...;
   i = (i + 1) % M;
}} while (i != poc);
```

Sortiranje

Selection sort

```
void SelectionSort(int A[], int N) {
   int i, j, min;
   for (i = 0; i < N; i++) {
      min = i;
      for (j = i + 1; j < N; j++) {
         if (A[j] < A[min]) min = j;
      }
      Zamijeni(&A[i], &A[min]);
   }
}
```

Primjer sortiranja

3	1	4	1	5	9	2	6
1	3	4	1	5	9	2	6
1	1	4	3	5	9	2	6
1	1	2	3	5	9	4	6
1	1	2	3	5	9	4	6
1	1	2	3	4	9	5	9
1	1	2	3	4	5	6	9
1	1	2	3	4	5	6	9
1	1	2	3	4	5	6	9

Bubble sort

```
void BubbleSort(int A[], int N) {
   int i, j;
   for (i = 0; i < N - 1; i++) {
      for (j = 0; j < N - 1 - i; j++) {
         if (A[j + 1] < A[j])
            Zamijeni(&A[j], &A[j + 1]);
      }
   }
}
```

nisam video prepisat primjer tocno, nebi trebalo biti previse komplicirano

Insertion sort

```
void InsertionSort(int A[], int N) {
   int i, j;
   int pom;
   for (i = 1; i < N; i++) {
      pom = A[i];
      for (j = i; j >= 1 && A[j - 1] > pom; j--)
         A[j] = A[j - 1];
      A[j] = pom;
   }
}
```

primjer sortiranja

6	7	9	1	12	2
1	6	7	9	12	2
1	2	6	7	9	12

Shell sort

```
void ShellSort(int A[], int N) {
   int i, j, korak, pom;
   for (korak = N / 2; korak > 0; korak /= 2) {
      for (i = korak; i < N; i++) {
         pom = A[i];
         for (j = i; j >= korak && A[j - korak] > pom; j -= korak) {
            A[j] = A[j - korak];
         }
         A[j] = pom;
      }
   }
}
```

primjer sortiranja

7	5	3	1	2	**8**
7	5	**3**	1	**2**	8
2	**5**	**3**	**1**	**7**	**8**
2	5	3	1	7	8
2	3	5	1	7	8
1	2	3	5	7	8
1	2	3	5	7	8
1	2	3	5	7	8

Merge sort i Quicksort

- proslo se kroz zadatak iz meduispita prosle godine
- 5. zadatak
- http://www.fer.unizg.hr/_download/repository/ASP-_MI_2013_s_rjesenjima_javno.pdf

Napomena: kodovi za sortove su preuzeti iz prezentacija, primjeri sortiranja najvjerojatnije nisu 100% tocni, oni za hash sigurno nisu 100% tocni, ostalo mislim da je ok