Algoritmi i strukture podataka 2019./2020.

Zadatci za vježbu – hash

U zadatcima se koristi sučelja `IHashableValue, IHash` te klasa `HashableElement`:

```
template <typename T, typename K> class IHashableValue {
 public:
   virtual K GetKey() const = 0;
};

template <typename T, typename K> class HashElement {
 public:
   IHashableValue<T, K> *value;
   HashElement *next;
   HashElement(IHashableValue<T, K> *value) { this->value = value; }
};

template <typename T, typename K> class IHash {
 protected:
   size_t size;
   //u zadacima koji slijede se koristi samo jedan od iduća dva retka
   IHashableValue<T, K> **hash; //koristi se za hash s otvorenim adresiranjem
   HashElement<T, K> **hash; //koristi se za hash s ulančavanjem

 public:
   virtual void Add(IHashableValue<T, K> *element) const = 0;
   virtual IHashableValue<T, K> *Get(K key) const = 0;
   virtual void Print() const = 0;
};
```

1.zadatak

Podatke o djelatnicima je potrebno organizirati u memorijski rezidentnu tablicu raspršenog adresiranja veličine 1000. Tablica raspršenog adresiranje organizirana je metodom ulančavanja. Potrebno je napisati:

a) Pretprocesorske direktive kojima se definira veličina tablica i klasa za pohranu podataka o osobama. Svaka osima ima ime koje je tipa string. Ime se smatra šifrom.
b) Funkciju za upis elemenata u tablicu raspršenog adresiranja
c) Funkciju za dohvat podataka iz tablice raspršenog adresiranja

Funkcije pod b) i c) trebaju imati prototipove sukladno sučelju IHash (Add i Get).

Funkcija raspršenja je zadana i nije ju potrebno pisati: `int HashStringToInt(string stringArray, size_t hashSize);`

Rješenje: a)

```
#define M 1000
class Person : public IHashableValue<Person, string> {
 public:
   string name;
   Person() {}
   Person(string name) { this->name = name; }
   virtual string GetKey() const { return name; }
};
```

b)

```
virtual void Add(IHashableValue<T, K> *element) const {
   int i = HashStringToInt(element->GetKey(), this->size);
   HashElement<T, K> *el = new HashElement<T, K>(element);
   el->next = hash[i];
   hash[i] = el;
}
```

c)

```
virtual IHashableValue<T, K> *Get(K key) const {
   int i = HashStringToInt(key, this->size);
   HashElement<T, K> *head;
   for (head = hash[i]; head && (head->value->GetKey() != key);
        head = head->next)
      ;
   if (head == nullptr)
      return nullptr;
   return head->value;
}
```

2.zadatak

Podatke o djelatnicima je potrebno organizirati u memorijski rezidentnu tablicu raspršenog adresiranja veličine 5000. Memorija za tablicu raspršenog adresiranja rezervira se u konstruktoru korištenjem operatora new. Kolizija se rješava dvostrukim raspršenim adresiranjem.

Potrebno je napisati:

a) Pretprocesorske direktive kojima se definira veličina tablica i klasu za pohranu podataka o osobama. Svaka osoba ima šifru (cijeli broj) i ime (string).
b) Funkciju za upis u tablicu raspršenog adresiranja koja je organizirana dvostrukim raspršenim adresiranjem te konstruktor hash-tablice.
c) Glavni program u kojem se stvara objekt klase-hash tablice i poziva funkcija za upis u tablicu. Podatke nije potrebno posebno učitavati iz datoteke ili s konzole, već je potrebno pretpostaviti da već postoje i samo pozvati funkciju za upis.

Prilikom realizacije funkcije za upis koristiti predefinirane funkcije raspršenog adresiranja:

```
int HashDoubleHashing1(int key) const { return key % M; }

int HashDoubleHashing2(int key) const { return 1 + key % (M - 1); }
```

Funkcija pod b) treba imati prototipove sukladno sučelju IHash (Add).

Rješenje:

a)

```
#define M 5000

class Person : public IHashableValue<Person, int> {
 public:
   int id;
   string name;
   Person() {
      this->id = 0;
      this->name = nullptr;
   }
   Person(int id, string name) {
      this->id = id;
      this->name = name;
   }
   virtual int GetKey() const { return id; }
};
```

```
   HashDoubleHashing(size_t size) {
      this->size = size;
      this->hash = new IHashableValue<T, K> *[this->size];
      this->Empty();
   }
```

b)

```
   virtual void Add(IHashableValue<T, K> *element) const {
      int h1 = HashDoubleHashing1(element->GetKey());
      int h2 = HashDoubleHashing2(element->GetKey());
      int index;
      for (int i = 0; i < this->size; i++) {
         index = (h1 + i * h2) % this->size;
         if (hash[index] == nullptr) {
            hash[index] = element;
            break;
         }
      }
   }
```

c)

```
     HashDoubleHashing<Person, int> hash(M);
      for (size_t i = 0; i < MAXINPUTELEM; i++) {
         hash.Add(&A[i]);
      }
```

3.zadatak

Podatke o djelatnicima organiziraju se u memorijski rezidentnu tablicu raspršenog adresiranja veličine 8. Kolizija se rješava kvadratnim ispitivanjem.

Zadane su pretprocesorske direktive i klasa:

```
#define MAXINPUTELEM 6
#define M 8
#define c1                                                                    \
   0.5 // constants to be used by the quadratic probing function, set to 0.5
       // guarantee diverse indices for tables which have size of 2^k
#define c2 0.5

class Person : public IHashableValue<Person, int> {
 public:
   int id;
```

```
   string name;
   Person() {
      this->id = 0;
      this->name = nullptr;
   }
   Person(int id, string name) {
      this->id = id;
      this->name = name;
   }
   virtual int GetKey() const { return id; }
};
```

Potrebno je napisati:

a) Funkciju za pražnjenje tablice raspršenog adresiranja
b) Funkciju za ispis tablice raspršenog adresiranja
c) Funkciju raspršenog adresiranja (Adresa) koja implementira metodu množenja. Knuthova konstanta koja se koristi za šifre veličine 16 bita je A=40503.

Funkcija pod b) treba imati prototip sukladno sučelju IHash (Print).

a)

```
   void Empty() {
      size_t i;
      for (i = 0; i < this->size; i++) {
         hash[i] = nullptr;
      }
   }
```

b)

```
   virtual void Print() const {
      size_t i;
      IHashableValue<T, K> *tmp;
      std::cout << "Hash table contents:\n";
      for (i = 0; i < this->size; i++) {
         std::cout << i << " ";
         tmp = hash[i];
         if (tmp != nullptr) {
            std::cout << tmp->GetKey() << " ";
         }
         std::cout << "\n";
      }
   }
```

c)

```
   int HashMultiplicationMethod(int key) const {
      unsigned int A = 2654435769; // constant A is chosen according to the
                                   // recommended value for 32-bit words (Knuth)
      unsigned int shift = 29; // (word size) - k = 32 - 3 (32 is due to 32-bit
                               // words, 3 comes from the exponent - 2^3)
      return (A * key) >> shift;
   }
```

4.zadatak

Svaki zapis datoteke `podaci.dat` organizirane po načelu raspršenog adresiranja sadrži podatke o jednom studentu i njegovoj ocjeni na kolegiju Algoritmi i strukture podataka: *šifru studenta (znakovni niz duljine 10 znakova), ime studenta (duljine do 20 znakova), prezime studenta (duljine do 30 znakova)* te *ocjenu (`int`)*.

Šifra nula (`"0"`) označava prazan zapis. Veličina bloka na disku je 4096 B. Očekuje se najviše 10000 zapisa, a kapacitet tablice treba biti 15% veći. Prilikom upisa primjenjuje se metoda cikličkog preljeva. Ključ zapisa je **`sifra`**, a pretvorba ključa u adresu se obavlja već pripremljenom funkcijom prototipa:

```
int adresa (char *sifra);
```

Napišite program koji će odrediti prosječnu ocjenu svih studenata čiji su zapisi završili u preljevu. Pretpostavite da u datoteci nije bilo brisanja (i iskoristite tu pretpostavku u radu s datotekom). Na zaslon je potrebno ispisati podatke za sve studente za koje zapisi nisu završili u preljevu, a kojima je ocjena veća ili jednaka prosječnoj ocjeni studenata čiji su zapisi završili u preljevu.

Primjer: Neka je prosječna ocjena `3,64`. Student `Ivan Novak` ima ocjenu `4` i šifru `0036123456`, a njegov zapis ne nalazi se u preljevu. Funkcija na zaslon mora ispitati njegove podatke u sljedećem formatu (prva linija označava znakovna mjesta pri ispisu):

```
1234567890123456789012345678901234567890123456789012345678901234567890
0036123456 Novak                          Ivan                4
```

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

#define N 10000
#define BLOK 4096
#define C BLOK/sizeof (zapis)
#define M (int)(N * 1.15/C)

typedef struct {
  char sifra[10 + 1];
  char ime[20 + 1];
  char prezime[30 + 1];
  int ocjena;
} zapis;

int adresa(int sifra);

int main() {

  FILE *f = NULL;
  zapis pretinac[C];
  int i, j, sumaOcjena = 0, brStudenata = 0;
  float prosjecnaOcjena = 0;

  f = fopen("podaci.dat", "rb");

  /*Dohvat ocjena iz zapisa koji su zavrsili u preljevu*/
  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(strcmp(pretinac[j].sifra,"0") == 0) {
        /*Prazan zapis - prelazimo na sljedeci pretinac*/
        break;
      }
      if(adresa(pretinac[j].sifra) != i) {
        /*Preljev*/
        brStudenata++;
        sumaOcjena += pretinac[j].ocjena;
      }
    }
  }

  /*Racunanje prosjecne ocjene*/
  prosjecnaOcjena = brStudenata == 0 ? 0 : (float)sumaOcjena / brStudenata;

  /*Na zaslon ispisujemo podatke za studente cija je ocjena >= prosjecnoj ocjeni*/
  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(strcmp(pretinac[j].sifra,"0") == 0) {
        /*Prazan zapis - prelazimo na sljedeci pretinac*/
        break;
      }
      if(adresa(pretinac[j].sifra) == i && pretinac[j].ocjena >= prosjecnaOcjena) {
        /*Zapis koji nije preljev*/
        printf("%-11s%-30s%-20s%-9d\n",
          pretinac[j].sifra,
          pretinac[j].prezime,
          pretinac[j].ime,
          pretinac[j].ocjena
        );
      }
    }
  }

  fclose(f);
  return 0;
}
```

5.zadatak

Jedan zapis datoteke organizirane po načelu raspršenog adresiranja sadrži matični broj studenta (int), ime i prezime (50+1 znak), godinu studija (int) te trenutni prosjek ocjena (float). Prazni se zapis prepoznaje po matičnom broju jednakom nula (0). Veličina bloka na disku je 2048 B. Očekuje se najviše 100000 zapisa, a tablica je dimenzionirana za 35% veći kapacitet. Prilikom upisa primjenjuje se metoda cikličkog preljeva. Ključ zapisa je matični broj, a pretvorba ključa u adresu se obavlja već pripremljenom funkcijom **`int adresa(int matbr);`**
Napisati funkciju koja će odrediti u koji je pretinac izvorno (prilikom upisa u datoteku) bilo upućeno najviše zapisa i potom vratiti broj zapisa upućenih u taj pretinac, a koji su završili u preljevu. Ako ima više takvih pretinaca vratiti rezultat za bilo koji od njih.
Funkcija treba imati prototip:

`int fun(FILE *f);`

```
#define N 100000
#define BLOK 2048
#define C BLOK/sizeof (zapis)
#define M (int)(N*1.35/C)

typedef struct z {
      int matbr;
      char ipr[50+1];
      int godina;
      float prosjek;
} zapis;

int fun(FILE *f){
      zapis pretinac[C];
      int i, j, zeljeni_pretinac;
      int max_pretinac = 0, max_vrijednost = 0;
      int broj_zapisa_za_pretinac[M] = {0};
      int broj_preljeva_za_pretinac[M] = {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) { //provjera je li zapis „pun"
                        zeljeni_pretinac = adresa(pretinac[j].sifra);
                        broj_zapisa_za_pretinac[zeljeni_pretinac]++;
                        if (zeljeni_pretinac != i) {
                              broj_preljeva_za_pretinac[zeljeni_pretinac]++;
                        }
                  }
                  else
                        break; //nakon prvog praznog u pretincu, svi su //ostali
                  prazni
            }
      }
      for (i = 0; i < M; i++) {
            if (broj_zapisa_za_pretinac[i] > max_vrijednost) {
                  max_pretinac = i;
                  max_vrijednost = broj_zapisa_za_pretinac[i];
            }
      }
      return broj_preljeva_za_pretinac[max_pretinac];
}
```

6.zadatak

Zadana je tablica raspršenog adresiranja s 8472 zapisa pohranjena u datoteci "podaci.dat" koja sadrži zapise s podacima o šifri studenta, prosjeku i broju položenih ispita (sami odaberite odgovarajuće tipove podataka!).

Preljevi su realizirani ciklički, upisom u prvi sljedeći slobodni pretinac. Pretinci su usklađeni s veličinom bloka od 512 okteta. Hash-tablica predimenzionirana je za 30%. Neka je ključ šifra studenta. Transformacija ključa u adresu obavlja se zadanom funkcijom:

```
int adresa (int sifra);
```

Napišite pretprocesorske direktive **#define** kojima se određuju parametri raspršenog adresiranja.

Napišite i funkciju **brojPreljeva** koja za zadani redni broj pretinca vraća ukupan broj popunjenih zapisa u zadanom pretincu i broj preljeva upisanih u zadani pretinac.

U glavnom programu koristeći funkciju **brojPreljeva** za svaki pretinac ispišite redni broj pretinca, ukupan broj upisanih zapisa u tom pretincu i broj preljeva u tom pretincu.

```
#define BLOK 512L       // Blok na disku
#define N 8472          // Ocekivani broj zapisa:
#define C ((int) (BLOK / sizeof (struct zapis)))     // Broj zapisa u pretincu
#define M ((int) (N / C * 1.3))    //  Broj  pretinaca,  kapacitet  30%  veci  od
minimalnog:

struct zapis{
  int sifra;
  float prosjek;
  int brojpolozenih;
};

void brojPreljeva (FILE *fh, int rbrPret, int *ukbroj, int * brPreljev) {
  struct zapis pretinac [C];
  int j;
  *ukbroj=0;
  *brPreljev=0;
  fseek (fh, rbrPret *BLOK, SEEK_SET);
  fread (pretinac, sizeof (pretinac), 1, fh);
  for (j = 0; j < C; j++) {
    if (pretinac[j].sifra != 0) {
      if (rbrPret != adresa(pretinac[j].sifra)) (*brPreljev)++;
      (*ukbroj)++;
    }
  }
}
int main() {
  FILE *fh;
  int brUk=0,brpreljeva=0
  if ((fh = fopen ("podaci.dat", "rb")) == NULL) exit (1);

  for (j = 0; j < M; j++) {
    brojPreljeva(fh, j, &brUk, &brpreljeva);
    printf("%d. pretinac   Ukupno: %d Broj preljeva: %d",j,brUk, brpreljeva);
  }
  close(fh);

}
```