Raspršeno adresiranje (hashing)

Pokušat ću ukratko objasniti što je hash, čemu služi i kako se realizira. To je namijenjeno onima koji imaju još poneku nejasnoću vezanu uz hash, a za sve ostale tu će biti što općenitije riješeni primjeri, tako da ako ne ništa, mogu to naučiti napamet :(.

Hash je zapravo struktura podataka (baš kao i neki niz ili lista).
Zasniva se na podjeli velike količine prostora u manje blokove koji se pune podatcima po određenom pravilu.
Konkretno, bit hash-a je hash funkcija (u primjerima obično data kao "int adresa(int n)").
Kvaliteta izvedbe hash-a najviše ovisi o njoj, ali nas obično ne traže da napišemo tu funkciju... ili nam je daju kao gotovu ili je svedu na neku trivijalu, npr. ostatak pri dijeljenju sa brojem pretinaca:

```
int adresa(int n) {
        return n % M;   // M je broj pretinaca
}
```

Ono kompliciranо kod hash-a je zapravo napisati dobru hash funkciju... ali, to mi ne moramo znati... ipak su to neki napredniji algoritmi...
Kad znamo koja je adresa pretinca pomoću funkcija za pozicioniranje unutar datoteke možemo trenutno učitati neki pretinac. Sve ostalo ću objašnjavati preko komentiranog koda.

Definirati macro naredbe i konstante koje se koriste za našu hash tabelu:
Recimo da imamo zadatak: Jedan zapis tablice raspršenih adresa sadrži šifru proizvoda (long), naziv proizvoda (50+1 znak) i cijenu (float). Veličina zapisa na disku je 1024. Prazni zapis sadrži šifru jednaku nuli. Očekuje se do 1000 zapisa, a tablica je dimenzionirana za 20% veći kapacitet od očekivanog. Znači moramo napisati:
Kod:

```
#define BLOK 1024                          // Veličina bloka na disku
#define N 1000                             // Ocekivani broj zapisa:
#define C ((int) (BLOK / sizeof (zapis))) // Broj zapisa u pretincu
(bloku)
#define M ((int) (N/C * 1.2))             // Broj pretinaca, kapacitet 30%
                                              veci od minimalnog

typedef struct {
        int sifra;
        char naziv[50+1];
        float cijena;
} zapis;
```

BLOK: Blok je veličina koju učitavamo jednim pozivom fread() funkcije (efikasnije nam je učitati cijeli blok, tj. 1024 ili 512 byte-ova nego učitavati zapis po zapis koji u našem slučaju ima samo 59 byte-ova, jer time smanjujemo broj pristupanja disku koji svejedno čita uvijek sektor po sektor (a on je veličine 512 ili više byte-ova)).
N: Očekivani broj zapisa koji moramo smjestiti u tablicu.
C: Broj zapisa u jednom pretincu (bloku).
M: Broj pretinaca u koje možemo stavljati naše zapise. Minimalan broj potrebnih pretinaca je N/C (zaokruženo na najbliži veći prirodni broj (funkcija ceil()), ali mi povećavamo broj pretinaca (za 20%, 30%, ...), tako da bi dobili što bolju raspršenost. Tj. ako povećamo broj pretinaca, izlazni rezultat naše funkcije adresa neće biti broj iz intervala [0, N/C], već [0, (N/C)*1.2]. Treba nam što veći M da ima što manje preljeva... jer preljevi su veliki minus raspršenog adresiranja...

Evo sad, funkcija po funkcija (koristimo gore zadane parametre):
*Napisati funkciju prototipa: int dodaj(FILE *f, zapis novi);*
Koja će dodati novi zapis u hash tablicu, služeći se hash funkcijom: int adresa(int n);
Ključ je šifra zapisa. Za preljeve se koristi metoda cikličkog prelaska u susjedni pretinac.
Funkcija vraća 1 ako je zapis unesen u tablicu/datoteku i smješten u svoj pretinac, -1 ako je upisan kao preljev ili 0 ako je tablica puna.
Kod:

```
#define BLOK 1024
#define N 1000
#define C ((int) (BLOK / sizeof (zapis)))
#define M ((int) (N/C * 1.2))

typedef struct {
        int sifra;
        char naziv[50+1];
        float cijena;
} zapis;

int dodaj(FILE *f, zapis novi) {
        /* polje u koje unosimo cijeli blok/pretinac zapisa */
        zapis pretinac[C];
        int adr, i, j;
        /* adresa pretinca koji odgovara novom zapisu */
        i = adr = adresa(novi.sifra);
        do {
                /* pozicioniramo se na početak pretinca */
                fseek (f, i*BLOK, SEEK_SET);
                /* učitavamo cijeli pretinac u polje */
                fread (pretinac, sizeof (pretinac), 1, f);
                /* pretražujemo cijeli pretinac koji smo spremili u
                 memoriju */
                for (j = 0; j < C; j++) {
                        if (pretinac[j].sifra == 0) {
                             /* našli smo slobodno mjesto u pretincu */
                             pretinac[j] = novi;
                             /* Pozicioniramo se nazad na početak
                             pretinca jer nam se čitanjem pokazivač
                             pomaknuo na kraj pretinca */
                             fseek (f, i*BLOK, SEEK_SET);
                             /* Upisujemo kompletan pretinac u
                             datoteku */
                             fwrite (pretinac, sizeof (pretinac), 1, f);
                             /* ako je i različit od adr (koji sadrži
                             početnu adresu pretinca, znači da je novi
                             zapis upisan kao preljev*/
                             if (i == adr) return 1;
                             else return -1;
                        }
                }
                /* Nismo našli slobodno mjesto u i-tom pretincu... idemo
                   dalje... ciklički!
                   Kad (i + 1) bude jednako M (a pretinac s indeksom M ne
                   postoji), i će postati jednak 0 */
                i = (i + 1) % M;
        } while (i != adr); /* ako smo se opet našli u početnom pretincu,
                               znači da u datoteci više nema mjesta */
        return 0;
}
```

Sad će te vidjeti da su nam za rješavanje svega ostalog potrebne minimalne preinake prethodne funkcije (uzimam da su mi postavke zadatka i svi parametri isti... jer izmjena parametara je trivijala...).
Napisati funkciju koja će prebrojati koliko ima punih pretinaca. Prototip funkcije je:
*int broji (FILE *f);*
Kod:

```
#define BLOK 1024
#define N 1000
#define C ((int) (BLOK / sizeof (zapis)))
#define M ((int) (N/C * 1.2))

typedef struct {
        int sifra;
        char naziv[50+1];
        float cijena;
} zapis;

int broji (FILE *f) {
        zapis pretinac[C];
        int adr, i, j;
        int broj_punih = 0, pun;

        /* Krećem od nultog pretinca, te pretražujem sve do (M-1)-og
           pretinca */
        adr = 0;
        i = adr;
        do {
                fseek (f, i*BLOK, SEEK_SET);
                fread (pretinac, sizeof (pretinac), 1, f);
                pun = 1;        /* pomoćna varijabla */
                for (j = 0; j < C; j++) {
                        if (pretinac[j].sifra == 0) {
                                /* pretinac nije pun... */
                                pun = 0;
                                break;  /* iskoči iz petlje... više nam
                                           nije potrebno provjeravati
                                           ovaj pretinac... */
                        }
                }
                if (pun) broj_punih++;  /* ako je pretinac pun, uvećaj
                                           broj punih */
                i = (i + 1) % M;
        } while (i != adr);
        return broj_punih;
}
```

Napisati funkciju koja će vratiti broj zapisa koji su u datoteku upisani kao preljev.
Kod:

```
#define BLOK 1024
#define N 1000
#define C ((int) (BLOK / sizeof (zapis)))
#define M ((int) (N/C * 1.2))

typedef struct {
        int sifra;
        char naziv[50+1];
        float cijena;
} zapis;

int br_preljeva (FILE *f) {
        zapis pretinac[C];
        int adr, i, j;
        int broj_preljeva = 0;

        /* Krećem od nultog pretinca, te pretražujem sve do (M-1)-og
           pretinca */
        adr = 0;
        i = adr;
        do {
                fseek (f, i*BLOK, SEEK_SET);
                fread (pretinac, sizeof (pretinac), 1, f);
                for (j = 0; j < C; j++) {
                        /* ako je šifra različita od 0, znači da zapis
                           postoji */
                        if (pretinac[j].sifra != 0) {
                           if (i != adresa(zapis.sifra)) broj_preljeva++;
                        }
                }
                i = (i + 1) % M;
        } while (i != adr);
        return broj_preljeva;
}
```

Napisati funkciju koja će zapisu u datoteci sa zadanom šifrom povećati cijenu za 10%. Ukoliko ne postoji zapis sa zadanom šifrom funkcija vraća 0, a inače 1.

Kod:

```
#define BLOK 1024
#define N 1000
#define C ((int) (BLOK / sizeof (zapis)))
#define M ((int) (N/C * 1.2))

typedef struct {
        int sifra;
        char naziv[50+1];
        float cijena;
} zapis;

int povecaj_cijenu(FILE *f, int sifra) {
        zapis pretinac[C];
        int adr, i, j;

        adr = adresa(sifra);
        i = adr;
        do {
                fseek (f, i*BLOK, SEEK_SET);
                fread (pretinac, sizeof (pretinac), 1, f);
                for (j = 0; j < C; j++) {
                        if (pretinac[j].sifra == sifra) {
                           pretinac[j].cijena = pretinac[j].cijena * 1.1;
                           fseek (f, i*BLOK, SEEK_SET);
                           fwrite (pretinac, sizeof (pretinac), 1, f);
                           return 1;
                        }
                }
                i = (i + 1) % M;
        } while (i != adr); /* ako smo se opet našli u početnom
                        pretincu, znači da u datoteci nema tog zapisa */
        return 0;
}
```

Napisati funkciju koja će dohvatiti podatke o proizvodu sa zadanom šifrom, i vratit ih preko imena funkcije. Ako proizvod s takvom šifrom ne postoji u datoteci, onda funkcija vraća prazan zapis (šifra zapisa je jednaka nuli).
Funkcija treba imati prototip:
*zapis dohvati(FILE *f, int sifra);*
Kod:

```
#define BLOK 1024
#define N 1000
#define C ((int) (BLOK / sizeof (zapis)))
#define M ((int) (N/C * 1.2))

typedef struct {
        int sifra;
        char naziv[50+1];
        float cijena;
} zapis;

zapis dohvati(FILE *f, int sifra);[/i] {
        zapis pretinac[C];
        int adr, i, j;

        adr = adresa(sifra);
        i = adr;
        do {
                fseek (f, i*BLOK, SEEK_SET);
                fread (pretinac, sizeof (pretinac), 1, f);
                for (j = 0; j < C; j++) {
                        if (pretinac[j].sifra == sifra) {
                                return pretinac[j];
                        }
                }
                i = (i + 1) % M;
        } while (i != adr);  /* ako smo se opet našli u početnom
                                pretincu, znači da u datoteci nema tog
                                zapisa */
        pretinac[0].sifra = 0;
        return pretinac[0];     // vraćamo "prazan" zapis
}
```

Napisati funkciju koja će izračunati postotak preljeva. Prototip:
*float postotak_preljeva(FILE *f);*
Kod:

```
#define BLOK 1024
#define N 1000
#define C ((int) (BLOK / sizeof (zapis)))
#define M ((int) (N/C * 1.2))

typedef struct {
        int sifra;
        char naziv[50+1];
        float cijena;
} zapis;

float postotak_preljeva(FILE *f) {
        zapis pretinac[C];
        int adr, i, j;
        int broj_preljeva = 0, broj_zapisa = 0;

        adr = 0;        /* Krećem od nultog pretinca, te pretražujem sve
                           do (M-1)-og pretinca */
        i = adr;
        do {
                fseek (f, i*BLOK, SEEK_SET);
                fread (pretinac, sizeof (pretinac), 1, f);
                for (j = 0; j < C; j++) {
                        /* ako je šifra različita od 0, znači da zapis
                           postoji */
                        if (pretinac[j].sifra != 0) {
                           if (i != adresa(pretinac[j].sifra)) {
                               broj_preljeva++;
                           }
                           broj_zapisa++;
                        }
                }
                i = (i + 1) % M;
        } while (i != adr);
        return (float) broj_preljeva / broj_zapisa;
}
```

Napisati funkciju koja će dodati novi zapis u tablicu. Za preljeve je rezervirano dodatnih 20% pretinaca na kraju tablice.
Funkcija treba vratiti 1 ako je zapis zapisan u svom pretincu, -1 ako je zapisan kao preljev ili 0 ako u preljevnom području više nema mjesta. Prototip je:
*int dodaj2(FILE *f, zapis novi);*
Kod:

```
#define BLOK 1024
#define N 1000
#define C ((int) (BLOK / sizeof (zapis)))
#define M ((int) (N/C * 1.2))

typedef struct {
        int sifra;
        char naziv[50+1];
        float cijena;
} zapis;

int dodaj2(FILE *f, zapis novi) {
        zapis pretinac[C];
        int adr, i, j;

        adr = adresa(novi.sifra);
        i = adr;
        do {
                fseek (f, i*BLOK, SEEK_SET);
                fread (pretinac, sizeof (pretinac), 1, f);
                for (j = 0; j < C; j++) {
                        if (pretinac[j].sifra == 0) {
                              pretinac[j] = novi;
                              fseek (f, i*BLOK, SEEK_SET);
                              fwrite (pretinac, sizeof (pretinac), 1, f);
                              if (i == adr) return 1;
                              else return -1;
                        }
                }
                /* Nismo našli slobodno mjesto u i-tom pretincu... idemo
                   dalje... ALI! Radimo u preljevnom području!
                   Prvi pretinac u preljevnom području se nalazi na
                   adresi N/C! */
                if (i == adr) i = N/C - 1;   /* Jer će sljedeća
                                                operacija uvećati i za 1 */
                i = (i + 1) % M;
        } while (i != 0);      /*  ako smo se opet našli u nepreljevnom
                                   području, znači nema da nema više
                                   mjesta u preljevnom području */
        return 0;
}
```

Ako treba neke komentirati, samo recite... jer... kasno je... i sad nemam nešto volje... ali sutra navečer ću imati vremena.

Programe nisam testirao... tako da 100% ima grešaka... najvjerojatnije ima grešaka i u samom principu... ako netko primijeti grešku, neka javi. Malo je prekasno za ove stvari...

Ako se traži da neki zapis obrišete, onda je dovoljno promijeniti funkciju:
int povecaj_cijenu(FILE *f, int sifra);
tako da joj liniju:
pretinac[j].cijena = pretinac[j].cijena * 1.1;
zamijenite linijom:
pretinac[j].sifra = 0;
Inače... ovo nije pravi način brisanja iz hash tablice (pravo brisanje je daleko kompliciranije)... ali, ako se ne varam, od nas se traži samo ovo.

Pazite! Ako vam napišu da je prototip funkcije, npr.:
int broji (**char *datoteka**);
znači da sami morate otvoriti (i kasnije zatvoriti) datoteku unutar funkcije!

Znam da tutorial nije nešto posebno dobro organiziran... ali pokušao sam sve napraviti u što kraćem vremenu... tako da ga dobijete prije utorka, i da ne radim ništa u pon. (osim fiz... :( )...

Sretno svima na 1. MI!

Ivan