Algoritmi i strukture podataka

ak. god. 2013./14.

Zadatci za 1. laboratorijsku vježbu

1. zadatak

Napišite funkciju koja u petlji učitava nizove znakova i sprema ih, nadovezujući jedan na drugi, u novi niz. Memoriju za novi niz potrebno je alocirati dinamički, a učitavanje nizova mora se prekinuti kada se učita niz ".". Funkcija mora vratiti pokazivač na novi niz. Nizovi koji se učitavaju neće biti dulji od 20 znakova, a alocirana memorija za novi niz ne smije biti veća od ukupne memorije koju zauzimaju svi učitani nizovi (osim niza ''.'').

Napišite glavni program koji poziva funkciju, na zaslon ispisuje niz koji funkcija vraća i oslobađanja dinamički alociranu memoriju.
Primjer: za učitane nizove: `"jedan" "dva" "tri6a 89+" "."`
novi niz mora sadržavati `"jedandvatri6a 89+"`

2. zadatak (nakon konzultacija)

Napišite **rekurzivnu** funkciju `slicnostNizova` koja preko parametara prima dva niza znakova (i ostale nužne parametre) i računa koliko su oni slični. Sličnost se računa kao omjer broja jednakih znakova i ukupne duljine oba niza.

Primjer: za nizove `"aa3"` i `"ba34"` funkcija treba vratiti `0.571429` jer je ukupna duljina oba niza 3 + 4 = 7 i vrijedi:
na poziciji 0 su znakovi a i b koji su različiti
na poziciji 1 su znakovi a i a koji su isti
na poziciji 2 su znakovi 3 i 3 koji su isti

slicnost=(0 + 2 + 2) / 7, a to je `0.571429`

Razmislite može li se funkcija ispravno implementirati ako je prototip funkcije

```
float slicnostNizova (char * niz1, char * niz2);
```

ili je funkciji potrebno predati još jedan parametar pa da prototip npr. bude

```
float slicnostNizova (char * niz1, char * niz2, int *arg1);
```

ili je funkciji potrebno predati još jedan parametar pa da prototip npr. bude

```
float slicnostNizova (char * niz1, char * niz2, int arg11, int arg22);
```

ili kakvim je još prototipovima moguće riješiti zadatak.

Napišite glavni program koji učitava dva niza (svaki max. 20 znakova), poziva funkciju i ispisuje vrijednost sličnosti na zaslon.

3. zadatak

Formatirana datoteka "domene.txt" (nalazi se u repozitoriju predmeta, u mapi "Laboratorijske vježbe") sadrži 100000 zapisa s podacima o nazivima domena (max. 30 znakova) i pripadnim IP adresama (max. 15 znakova). Podatke iz te datoteke pohranite u hash-tablicu izvedenu kao binarnu datoteku, a ključ neka bude naziv domene. Preljeve realizirajte ciklički, upisom u prvi sljedeći slobodni pretinac. Pretinac neka bude veličine 4096 okteta. Hash-tablicu predimenzionirajte za 30%.
Na predavanjima vam je pokazana transformacija znakovnog niza u adresu pretinca koja sumira ASCII vrijednosti znakova u nizu i nakon toga računa ostatak s brojem pretinaca. U stvarnoj primjeni se ona nije pokazala kao dobra jer ne daje dobro raspršenje.

Za transformaciju nizova znakova u broj često se koriste funkcije oblika:

```
unsigned long Hash(char * niz)
{
   unsigned long hash = POCETNA_VRIJEDNOST;
   char c;
   while (c = *niz++)
      hash = MNOZITELJ * hash + c;
   return hash;
}
```

Adresa pretinca za zadani ključ se izračuna kao ostatak cjelobrojnog dijeljenje vrijednosti koju je vratila funkcija `Hash` i broja pretinaca.

Implementacija funkcije koja za početnu vrijednost ima `5381`, a za množitelj vrijednost `33` pokazala je odlične rezultate i često se pronalazi pod imenom *djb* jer je za nju zaslužan Daniel J. Bernstein.

I programski jezik Java koristi gore navedenu funkciju za računanje hash vrijednosti niza znakova. Međutim, implementacija u Javi za početnu vrijednost ima 0, a za množitelj vrijednost 31.
Upućujemo zainteresirane da više o ovome potraže na internetu.

Vaš je zadatak da kopirate u vaše rješenje gore napisanu funkciju `Hash` i isprobate s kojom kombinacijom različitih početnih vrijednost i množitelja dobivate najmanje preljeva prilikom zapisivanja podataka u hash-tablicu. Jedan od načina da to utvrdite bio bi da svaki puta kad promijenite vrijednosti u funkciji `Hash`, ponovo pokrenete program, napravite novu hash-tablicu na disku i pritom pobrojite preljeve. Međutim, to nije najbolje rješenje jer nas zanima samo broj preljeva, a stvarno zapisivanje hash-tablice na disk želimo napraviti tek kad utvrdimo vrijednosti s kojima dobijemo najmanje preljeva.
Da ne bi svaki puta stvarali novu hash-tablicu napišite funkciju `BrojPreljeva` koja će čitati podatke iz ulazne datoteke, za svaki zapis utvrditi adresu pomoću `Hash` funkcije i ostatka cjelobrojnog dijeljenja s brojem pretinaca, te umjesto da zapiše podatak u hash-tablicu, podatak o broju zapisa u pretincu pohranit će u cjelobrojno jednodimenzionalno polje. Polje sadrži onoliko elemenata koliko hash-tablica sadrži pretinaca, na početku su vrijednosti svih elemenata 0, a vrijednosti elemenata polja s indeksom ***i*** na kraju moraju sadržavati broj zapisa koji bi bili zapisani u pretinac ***i***. Umjesto da zapišete zapis u hash-tablicu u pretinac ***i*** povećat ćete vrijednost elementa polja s indeksom ***i*** za 1. Vrijednosti elemenata ne mogu prijeći definirani broj zapisa po pretincima jer to i u stvarnom zapisivanju u hash-tablicu na disku nije moguće, to je preljev koji se upisuje u sljedeći pretinac. Prilikom čitanja ulaznih podataka i popunjavanja polja s brojem zapisa po pretincima odmah u nekoj varijabli sumirajte broj preljeva.

Za najmanje 5 (bilo bi dobro da napravite i za više) različitih početnih vrijednosti i/ili množitelja utvrdite broj preljeva i ručno zapišite podatke o početnoj vrijednosti, množitelju i broju preljeva u neku datoteku (npr. koristeći Notepad), da ne zaboravite podatke koji su se pokazali najboljima.

Na kraju, implementirajte funkciju `Upis` koja stvora hash-tablicu na disku.
Prototipi traženih funkcija su sljedeći:

```
int BrojPreljeva (FILE *fulaz); //vraća broj preljeva

int Upis (FILE *fulaz, FILE *fhashTablica);
//vraća 0 ako nije uspjela zapisati sve zapise, inače vraća 1
```

Napišite glavni program i ako je potrebno dodatne funkcije koje su neophodne za realizaciju zadatka.

4. zadatak (za one koji žele znati više)

U 3. zadatku bilo je potrebno napisati samo jednu funkciju Hash za transformaciju ključa. Ako imate volje napišite još nekoliko funkcija Hash2, Hash3, itd. koje su istog prototipa, ali implementiraju različite algoritme. Više funkcija različitog imena povlači za sobom da svaki puta kad želite isprobati efikasnost neke od hash funkcija treba mijenjati i naredbu koja poziva tu funkcije u funkcijama `BrojPreljeva` i `Upis`.

Kako bi zadržali jedinstvenu implementaciju tih funkcija, a da one ipak mogu raditi s različitim funkcijama za izračunavanje adrese, potrebno je njihove prototipe proširiti s dodatnim parametrom koji je tipa pokazivač na funkciju.

```
int BrojPreljeva (FILE *fulaz, unsigned long (*hash)(char*));
int Upis (FILE *fulaz, FILE * fhashTablica, unsigned long (*hash)(char*));
```

Naravno, unutar funkcija `BrojPreljeva` i `Upis` potrebno je malo promijeniti i naredbu koja poziva hash funkciju, te u glavnom programu poziv funkcija `BrojPreljeva` i `Upis`.

Potražite na Internetu način na koji se koriste pokazivači na funkcije u C-u te napravite potrebne prilagodbe u 3. zadatku kako bi funkcije `BrojPreljeva` i `Upis` mogle pozivati različite hash funkcije.