Projektni zadatak iz programiranja

Memorijski i vremenski efikasno cuvanje frekvencije velikog broja stringova

-Lazar Aleksic i Matea Jankovic, odeljenje II-7

Postoje dva velika problema kod cuvanja velikog broja stringova zajedno sa njihovom frekvencijom.

Prvi problem je pretrazivanje za iste stringove jer moramo uporedjivati nas trazeni string sa svakim sto konzumira dosta resursa.

Drugi problem je samo cuvanje toliko karaktera, to zauzima veliki memorijski prostor zbog ponavljanja karaktera koji zelimo da kompresujemo.

Najbolji pristup sto se tice kompresije bi bilo da cuvamo stablo takvo da svaki cvor cuva po jedan bit, medjutim ovo stablo bi bilo sporo za pretrazivanje.

Najbrzi pristup za pretragu bi bio da svaki karakter cuva sve moguce sledece karaktere posle njega (odnosno sve ascii karaktere do 33 do 127) medjutim ovo bi bilo memorijski inefikasno.

Mi smo izabrali pristup koji uzima najbolje od oba opcije, osrednje vreme za pretragu i osrednju kompresiju, svaki karakter cuvamo u 3 razlicita cvora, a svaki cvor sadrzi jednu cifru njegovog ascii koda (naprimer, razmak kao '033' bi smo cuvali u tri cvora, 0, 3 i 3).

Za ovakvu strukturu smo koristili "rekurzivnu strukturu"

typedef struct head{

struct node\* zero;

struct node\* one;

int freq;

int afreq;

}Head;

typedef struct node{

struct tail\* zero;

struct tail\* one;

struct tail\* two;

struct tail\* three;

struct tail\* four;

struct tail\* five;

struct tail\* six;

struct tail\* seven;

struct tail\* eight;

struct tail\* nine;

}Node;

typedef struct tail{

struct head\* zero;

struct head\* one;

struct head\* two;

struct head\* three;

struct head\* four;

struct head\* five;

struct head\* six;

struct head\* seven;

struct head\* eight;

struct head\* nine;

}Tail;

Struktura glava (eng. Head) pokazuje na strukturu cvor (eng. Node) koja pokazuje na struktru rep (eng. Tail), a struktura rep ponovo pokazuje na glavu i tako se ponavlja.

Druge dve strukture ce nam biti Fifo (popularniji kao eng. Pipe), i njega cemo korisiti za prosledjivanje ascii vrednosi iz unosa u funkciju za pravljenje stabla. (Bolje je nego da se iznova zove rekurzija sa svaki karakter u reci)

I listu cemo koristit za demonstriranje citanja sa stabla i upisivanje stringa zajedno sa njegovom frekvencijom.

typedef struct pipe{

uint8\_t podatak;

struct pipe\* sledeci;

}Pipe;

typedef struct list{

char str[128];

int afreq;

struct list\* sledeci;

}List;

Za Fifo cemo koristiti 3 funkcije, "make" "insert" i "delete", make sluzi za inicijalno kreiranje Fifa kada je prvobitno postavljen na NULL, a insert za svako sledece dodavanje. Delete sluzi za citanje pa brisanje sa Fifa.

void make(Pipe\*\* pocetak, Pipe\*\* kraj, int x){

Pipe\* novi=(Pipe\*)malloc(sizeof(Pipe\*));

novi->sledeci=NULL;

novi->podatak=x;

(\*pocetak)=novi;

(\*kraj)=novi;

}

void insert(Pipe \*\*kraj, int x){

Pipe\* novi=(Pipe\*)malloc(sizeof(Pipe));

novi->sledeci=NULL;

novi->podatak=x;

(\*kraj)->sledeci=novi;

(\*kraj)=novi;

}

int delete(Pipe \*\*pocetak, Pipe \*\*kraj){

int retval=(\*pocetak)->podatak;

if((\*pocetak)==(\*kraj)){

free((\*pocetak));

(\*pocetak)=NULL;

(\*kraj)=NULL;

return retval;}

Pipe\* novi=(\*pocetak);

(\*pocetak)=(\*pocetak)->sledeci;

free(novi);

return retval;}

Za listu imamo samo neke osnovne funckije, stampaj, ReadHead (vezana i za stablo pa ce biti objasnjena kasnije), SelectionSortIVelicina (radi selection sort i ujedno odredjuje velicinu).

void stampaj(List\* lista){

printf("\n");

while(lista!=NULL){

printf("%s:%i\n", lista->str, lista->afreq);

lista=lista->sledeci;

}

}

void SelectionSortIVelicina(List\* lista, int\* n){

List\* i=lista;

while(i!=NULL){

List\* j=i->sledeci;

List\* min=i;

while(j!=NULL){

if(j->afreq>min->afreq){

min=j;

}

j=j->sledeci;

}

char str[32];

strcpy(str, min->str);

strcpy(min->str, i->str);

strcpy(i->str, str);

int pom=min->afreq;

min->afreq=i->afreq;

i->afreq=pom;

i=i->sledeci;

\*n=\*n + 1;

}

}

Osim toga koristimo KMP (Knuth-Morris-Pratt) algoritam za nalazenje podstringa u vecem stringu, medjutim sa modifikovanim pretrazivanje na takav nacin da iskljucivo trazi reci (tako sto proverava da li imamo razmake ispred i iza njih, i da li se nalaze na pocetku ili kraju veceg stringa) i nakon toga ih zamenjuje sa nekim drugim recima. (pokazacemo u glavnom programu)

Osim toga za zamenu reci koristimo memmove zato sto ova funkcija odprilike koristi O(1) u odnosu na pomeranje svakog slova na levo za po jednom dok se ne obrise prvobitna rec, pa onda dodavanje po jednog slova na desno dok ne upisemo novu rec sto bi odprilike zahtevalo O(2n).

Ali algoritam za preprocesiranje stringa kojega trazimo nije promenjen.

void kmp(char\* str, int m, int\* niz){

int j=0;

int i=1;

niz[0]=0;

while(i<m){

if(str[i]==str[j]){

niz[i]=j+1;

i++;

j++;

}else{

if(j!=0)j=niz[j-1];

else{

niz[i]=0;

i++;

}

}

}

}

Nakon ovoga imamo funkcije za pravljenje (alociranje) citavog stabla i funkcije za citanje njega u listu.

void HeadAlloc(Head\* root, Pipe\*\* pocetak, Pipe\*\* kraj);

Head\* TailAlloc(Tail\* root, int x){

switch(x%10){

case 0:

if(root->zero==NULL)root->zero=(Head\*)calloc(1,sizeof(Head));

return root->zero;

case 1:

if(root->one==NULL)root->one=(Head\*)calloc(1,sizeof(Head));

return root->one;

case 2:

if(root->two==NULL)root->two=(Head\*)calloc(1,sizeof(Head));

return root->two;

case 3:

if(root->three==NULL)root->three=(Head\*)calloc(1,sizeof(Head));

return root->three;

case 4:

if(root->four==NULL)root->four=(Head\*)calloc(1,sizeof(Head));

return root->four;

case 5:

if(root->five==NULL)root->five=(Head\*)calloc(1,sizeof(Head));

return root->five;

case 6:

if(root->six==NULL)root->six=(Head\*)calloc(1,sizeof(Head));

return root->six;

case 7:

if(root->seven==NULL)root->seven=(Head\*)calloc(1,sizeof(Head));

return root->seven;

case 8:

if(root->eight==NULL)root->eight=(Head\*)calloc(1,sizeof(Head));

return root->eight;

case 9:

if(root->nine==NULL)root->nine=(Head\*)calloc(1,sizeof(Head));

return root->nine;

}

return NULL;

}

Head\* NodeAlloc(Node\* root, int x){

switch((x%100)/10){

case 0:

if(root->zero==NULL)root->zero=(Tail\*)calloc(1,sizeof(Tail));

return TailAlloc(root->zero, x);

case 1:

if(root->one==NULL)root->one=(Tail\*)calloc(1,sizeof(Tail));

return TailAlloc(root->one, x);

case 2:

if(root->two==NULL)root->two=(Tail\*)calloc(1,sizeof(Tail));

return TailAlloc(root->two, x);

case 3:

if(root->three==NULL)root->three=(Tail\*)calloc(1,sizeof(Tail));

return TailAlloc(root->three, x);

case 4:

if(root->four==NULL)root->four=(Tail\*)calloc(1,sizeof(Tail));

return TailAlloc(root->four, x);

case 5:

if(root->five==NULL)root->five=(Tail\*)calloc(1,sizeof(Tail));

return TailAlloc(root->five, x);

case 6:

if(root->six==NULL)root->six=(Tail\*)calloc(1,sizeof(Tail));

return TailAlloc(root->six, x);

case 7:

if(root->seven==NULL)root->seven=(Tail\*)calloc(1,sizeof(Tail));

return TailAlloc(root->seven, x);

case 8:

if(root->eight==NULL)root->eight=(Tail\*)calloc(1,sizeof(Tail));

return TailAlloc(root->eight, x);

case 9:

if(root->nine==NULL)root->nine=(Tail\*)calloc(1,sizeof(Tail));

return TailAlloc(root->nine, x);

}

return NULL;

}

void HeadAlloc(Head\* root, Pipe\*\* pocetak, Pipe\*\* kraj){

do{

int podatak=delete(pocetak, kraj);

if(podatak/100){

if(root->one==NULL)root->one=(Node\*)calloc(1,sizeof(Node));

root=NodeAlloc(root->one, podatak);

}else{

if(root->zero==NULL)root->zero=(Node\*)calloc(1,sizeof(Node));

root=NodeAlloc(root->zero, podatak);

}

root->freq++;

}while((\*pocetak)!=NULL);

root->afreq++;

}

Kod alociranja nemamo nista kompleksno, na stacku se najvise cuvaju 3 cvora, kada dodjemo do repa na prvi poziv vratimo novu glavu i nastavimo dalje sa kreiranjem, put koje stablo uzima je odluceno od strane cifara ascii karaktera u jednoj reci.

void ReadHead(Head\* root, List\* lista, char passdown[], int i, int ascii);

void ReadTail(Tail\* root, List\* lista, char passdown[], int i, int ascii){

if(root->zero!=NULL)ReadHead(root->zero, lista, passdown, i+1, ascii\*10);

if(root->one!=NULL)ReadHead(root->one, lista, passdown, i+1, ascii\*10+1);

if(root->two!=NULL)ReadHead(root->two, lista, passdown, i+1, ascii\*10+2);

if(root->three!=NULL)ReadHead(root->three, lista, passdown, i+1, ascii\*10+3);

if(root->four!=NULL)ReadHead(root->four, lista, passdown, i+1, ascii\*10+4);

if(root->five!=NULL)ReadHead(root->five, lista, passdown, i+1, ascii\*10+5);

if(root->six!=NULL)ReadHead(root->six, lista, passdown, i+1, ascii\*10+6);

if(root->seven!=NULL)ReadHead(root->seven, lista, passdown, i+1, ascii\*10+7);

if(root->eight!=NULL)ReadHead(root->eight, lista, passdown, i+1, ascii\*10+8);

if(root->nine!=NULL)ReadHead(root->nine, lista, passdown, i+1, ascii\*10+9);

}

void ReadNode(Node\* root, List\* lista, char passdown[], int i, int ascii){

if(root->zero!=NULL)ReadTail(root->zero, lista, passdown, i, ascii\*10);

if(root->one!=NULL)ReadTail(root->one, lista, passdown, i, ascii\*10+1);

if(root->two!=NULL)ReadTail(root->two, lista, passdown, i, ascii\*10+2);

if(root->three!=NULL)ReadTail(root->three, lista, passdown, i, ascii\*10+3);

if(root->four!=NULL)ReadTail(root->four, lista, passdown, i, ascii\*10+4);

if(root->five!=NULL)ReadTail(root->five, lista, passdown, i, ascii\*10+5);

if(root->six!=NULL)ReadTail(root->six, lista, passdown, i, ascii\*10+6);

if(root->seven!=NULL)ReadTail(root->seven, lista, passdown, i, ascii\*10+7);

if(root->eight!=NULL)ReadTail(root->eight, lista, passdown, i, ascii\*10+8);

if(root->nine!=NULL)ReadTail(root->nine, lista, passdown, i, ascii\*10+9);

}

void ReadHead(Head\* root, List\* lista, char passdown[], int i, int ascii){

passdown[i]=(char)ascii;

if(root->zero!=NULL)ReadNode(root->zero, lista, passdown, i, 0);

if(root->one!=NULL)ReadNode(root->one, lista, passdown, i, 1);

if(root->afreq>0){

passdown[i+1]='\0';

List\* novi=(List\*)malloc(sizeof(List));

strcpy(novi->str, passdown);

novi->afreq=root->afreq;

novi->sledeci=lista->sledeci;

lista->sledeci=novi;

}

}

Kod citanja vec imamo malo kompleksniju funkciju jer ona za svaki cvor rekurzivno cuva njegovo trenutno stanje celog stringa zajedno sa stanjem pozicije sledeci karaktera, zajedno sa stanjem ascii koda za sledeci karakter. Kada pronadje cvor cija je prava frekvencija veca od nule, onda taj cvor zajedno sa trenutnim stringom upise u listu. Takodje ovaj proces zahteva dosta resursa i za vece baze jeste zahtevan.

Sto se tice glavnog programa na pocetku imamo dinamicno i memorijski sigurno unosenje stringa iz datoteke ili standardnog ulaza.

int main(){

FILE\* dat;

char odluka[128];

printf("Unesite ime datoteke ili stdin: ");

fgets(odluka, 127, stdin);

odluka[strcspn(odluka, "\n")]='\0';

if(strcmp(odluka, "stdin")!=0){

dat=fopen(odluka, "r");

}else{

dat=stdin;

}

char\* input=malloc(65);

int maxlen=64;

int len=0;

char ch;

while((ch = fgetc(dat)) != EOF){

if(len >= maxlen){

maxlen += 8;

char\* temp=realloc(input, maxlen+1);

if(temp==NULL){

fprintf(stderr, "Greska pri alokaciji!\n");

free(input);

return 1;

}

input=temp;

}

input[len]=(char)ch;

len++;

}

input[len]='\0';

Otvaramo ili fajl ili kao "stream" teksta uzimamo standardni ulaz, nakon toga proveravamo da imamo dovoljno memorije za cuvanje sledeceg karaktera u string, a u slucaju da nemamo koristimo funkciju realloc koja radi na principu da ce da kopira sve iz stringa "input" u temp ali sa novom maksimalnom velicinom, posle toga free-uje input a mi input postavimo na temp tako da mozemo da nastavimo unos ali na vecem memorijskom bloku. Na kraju unosa nakon sto koristnik unese EOF (End of File) koristeci Ctrl+D (a na DOS operativnim sistemima Ctrl+Z), stavimo NULL na kraj stringa i nastavimo dalje sa programom.

Deo za pravljenje stabla iz glavnog programa je takodje poprilicno jasan, zaobilazimo sve karaktere koji nisu deo reci (razmaci i slicno) i cuvamo njihove ascii vrednosti u Fifu, takodje maxstrlen i substrlen koristimo da bi smo nasli najduzu rec u inputu kako bi znali koliki treba biti niz za karaktere.

Pipe \*pocetak, \*kraj;

List\* lista;

Head\* root;

int maxstrlen=0, substrlen, i;

root=(Head\*)calloc(1, sizeof(Head));

lista=(List\*)calloc(1, sizeof(List));

pocetak=NULL;

kraj=NULL;

len=strlen(input);

while(i<len && input[i]!='\0'){

if(input[i]!=' ' && input[i]!='\n' && input[i]!='\0'){

substrlen=i;

make(&pocetak, &kraj, (int)input[i]);

i++;

while(input[i]!=' ' && input[i]!='\0' && input[i]!='\n'){

insert(&kraj, (int)input[i]);

i++;

}

substrlen=i-substrlen;

if(substrlen>maxstrlen) maxstrlen=substrlen;

HeadAlloc(root, &pocetak, &kraj);

}else{

i++;

}

}

maxstrlen=maxstrlen+1;

U sledecem delu prosto citamo iz generisanog stabla i unosimo u listu koju sortiramo po opadajucem redosledu.

char passdown[maxstrlen];

int nstr=0;

ReadHead(root, lista, passdown, -1, 0);

SelectionSortIVelicina(lista, &nstr);

printf("Unikatne reci: %i\nREC:FREKVENCIJA", nstr);

stampaj(lista);

Ostali deo je samo nastavak kmp algoritma i modifikovana zamena o kojoj smo pricali.

char str[1023];

char replace[1022];

printf("\n\nUnesite rec koju zelite da zamenite: ");

fgets(str, 1022, stdin);

printf("Unesite rec sa kojom zelite da prethodnu zamenite: ");

fgets(replace, 1021, stdin);

str[strcspn(str, "\n")]='\0';

replace[strcspn(replace, "\n")]='\0';

int slen=strlen(str);

int rlen=strlen(replace);

int kmptable[slen];

kmp(str, slen, kmptable);

int j=0;

i=0;

while(i<len){

if(str[j]==input[i]){

i++;

j++;

}else{

if(j!=0) j=kmptable[j-1];

else i++;

}

if(j==slen){

if((input[i-j-1]!=' ' && i-j!=0) || (input[i]!=' ' && i!=len))goto skip;

//printf("\nNadjen na %i\n", i-j);

int start = i - j;

int end = start + rlen;

if (rlen > slen) {

memmove(&input[start + rlen], &input[start + slen], len - start - slen + 1);

} else if (rlen < slen) {

memmove(&input[end], &input[start + slen], len - start - slen + 1);

}

strncpy(&input[start], replace, rlen);

len = strlen(input);

skip:

j=kmptable[j-1];

}

}

puts(input);

I na kraju upis u file, ako je unos stringa bio iz standardnog ulaza onda je korisnik ponudjen i sa time da napravi novi fajl u kome ce sacuvati promenjen string.

if(strcmp(odluka,"stdin")==0){

getchar();

printf("Unesite ime datoteke: ");

fgets(odluka, 127, stdin);

odluka[strcspn(odluka, "\n")] = '\0';

}

if(strcmp(odluka,"stdin")==0){

printf("Ne moze se pisati u stdin.");

if(dat)fclose(dat);

return 0;

}

dat=fopen(odluka,"w");

if (!dat) {

fprintf(stderr, "Greska pri otvaranju %s\n", odluka);

exit(1);

}

fprintf(dat, "%s", input);

fclose(dat);

return 0;

}

-Lazar Aleksic i Matea Jankovic, odeljenje II-7