Strukture podataka i algoritmi 1
(zadatak - max 30 poena)

Jun, 2024

Nakon upisa na studije, studentima se dodeljuju mentori kao vid pomoći tokom studija. Za svakog mentora se zna njegov ID (ceo broj), ime i prezime (niz karaktera) i maksimalan broj studenata kojima može da bude mentor (ceo broj). Studenti prilikom prijave imaju svoj ID (ceo broj), ime i prezime (niz karaktera), broj poena sa upisa (realan broj) i 2 želje za mentora - ID-jevi mentora. Zahtevi se rešavaju po redosledu prijave, tj. Po redosledu po kom su navedeni u fajlu. Ukoliko ima slobodnih mesta kod mentora kog je student naveo kao prvu želju, studentu će biti dodeljen taj mentor, a u suprotnom se razmatra druga želje. Ukoliko ni kod drugog mentora nema slobodnog mesta student ostaje neraspoređen. Student kao drugu ili obe želje može navesti 0 (broj nula), što znači da on nema želje. Ukoliko je naveo za obe želje 0, on je automatski neraspoređen, a ukoliko je samo kao druga želja 0, onda će biti pokušano raspoređivanje po prvoj želji. Po završetku inicijalne raspodele neraspoređeni studenti se redom, jedan po jedan, raspoređuju kod mentora koji u tom trenutku ima najviše slobodnih mesta. Pretpostavka je da slobodnih mesta ima više nego studenata. Nakon što se svi studenti rasporede ispisati koliko kod kog mentora ima slobodnih mesta i njegov procenat zauzeća. Naknadno, student može poredati molbu za promenom mentora. Ukoliko kod željenog mentora ima slobodnih mesta, on će biti prebačen.
(kompletna main funkcija - 3 poena)

Za rešavanje problema napisati sledeće funkcije:

a) Definisati sve potrebne složene tipove podataka neophodne za rešavanje opisanog problema.
(3 poena, obavezno)
b) Napisati funkciju **UcitajMentore** koja iz datoteke *Mentori.txt* učitava podatke o mentorima i formira listu/niz mentora.
(3 poena, obavezno)
c) Napisati funkciju **NadjiMentora** koja za dati ID mentora vraća pokazivač na određenog mentora.
(3 poena, obavezno)
d) Napisati funkciju **UcitajStudente** koja iz datoteke *Studenti.txt* učitava podatke o studentima, formira listu/niz studenata i vrši raspoređivanje po mentorima.
(3 + 4 poena)
Napomena. Rapoređivanje se može izdvojiti u posebnu funkciju**.**
Bonus 4 poena ukoliko je prioritet pri rasoređivanju broj poena pri upisu, a ne redosled prijave
e) Napisati funkciju **RasporedNeraspoređenih** koja raspoređuje neraspoređene studente kod mentora sa najviše slobodnih mesta.
(4 poena)
f) Napisati funkciju **Ispis** koja za svakog mentora ispisuje broj slobodnih mesta i procenat zauzetih mesta.
(2 poena)
g) Napisati funkciju **Promena** koja sa standardnog ulaza učitava ID studenta i ID mentora kod kog student želi da pređe i realizuje promenu, ukoliko je moguće.
(5 poena)

Pri učitavanju podataka podrazumevati da su svi podaci korektno zadati.
Podaci iz datoteke se mogu učitavati uz pretpostvku da se na početku nalazi broj podataka u datoteci ili se podaci mogu učitavati do kraja datoteke.
Dozvoljeno je proširivanje struktura i definisanje novih, kao i definisanje drugih funkcija.
Maksimalan broj poena se dobija samo u slučaju da postoji veza između struktura koje čuvaju mentore i struktura koje čuvaju studente, tj. ako postoje pokazivači “na jednu” ili “na obe strane”.
Optimalno definisanje struktura i funkcija koje se mogu koristiti više puta donosi bonus poene.

Zadatak se može rešavati korišćenjem povezanih lista, nizova ili kombinacijom.
Zadatak rešiti bez korišćenja globalnih promenljivih i bez unapred definisanih dužina korišćenih nizova (izuzetak može da bude pomoćni niz za učutavanje stringova)**.**
Za funkcije koje nisu definisane mora da postoji deklaracija funkcije i kratak opis koji deo problema funkcija treba da reši i mogu se pozivati kao da postoji kompletno definisana funkcija.

Rešenje studenta

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

#define LEN 255

typedef struct student{
    int id;
    char *imePrezime;
    float poeni;
    int idm1;
    int idm2;
    int rasporedjen;

    struct student *sledeci;
}Student;

typedef struct element{
    Student *student;
    struct element *sledeci;
}Element;

typedef struct mentor{
    int  id;
    char *imePrezime;
    int max;
    int trenutno;
    Element *studenti;

    struct mentor *sledeci;
}Mentor;

Mentor *UcitajMentore(char *nazivFajla)
{
    FILE *in = fopen(nazivFajla,"r");
    Mentor *glava = NULL;

    while(!feof(in)){
        Mentor *novi = (Mentor*)malloc(sizeof(Mentor));

        fscanf(in,"%d\n",&novi->id);

        char ulaz[LEN];
        fgets(ulaz,LEN,in);
        ulaz[strlen(ulaz)-1] = '\0';
        novi->imePrezime = (char*)malloc((strlen(ulaz)+1)*sizeof(char));
        strcpy(novi->imePrezime,ulaz);

        fscanf(in,"%d\n",&novi->max);

        novi->trenutno = 0;
        novi->studenti = NULL;

        novi->sledeci = glava;
        glava = novi;
    }
    fclose(in);
    return glava;
}

Mentor *NadjiMentora(int id,Mentor *glava)
{
    while(glava && glava->id!=id)
```

```
         glava = glava->sledeci;
    return glava;
}

//funkcija koja dodaje studente sortirano
void DodajSortirano(Student **glava,Student *novi)
{
    if(*glava == NULL){
        *glava = novi;
        return;
    }

    Student *pom = *glava;
    Student *pret = NULL;

    if(pom->poeni < novi->poeni){
        novi->sledeci = *glava;
        *glava = novi;
        return;
    }

    while(pom && pom->poeni > novi->poeni){
        pret = pom;
        pom = pom->sledeci;
    }

    novi->sledeci = pom;
    pret->sledeci = novi;
}

Student *UcitajStudente(char *nazivFajla)
{
    Student *glava = NULL;
    FILE *in = fopen(nazivFajla,"r");

    while(!feof(in)){
        Student *novi = (Student*)malloc(sizeof(Student));
        novi->sledeci = NULL;

        fscanf(in,"%d\n",&novi->id);

        char ulaz[LEN];
        fgets(ulaz,LEN,in);
        ulaz[strlen(ulaz)-1] = '\0';
        novi->imePrezime = (char*)malloc((strlen(ulaz)+1)*sizeof(char));
        strcpy(novi->imePrezime,ulaz);

        fscanf(in,"%f%d%d\n",&novi->poeni,&novi->idm1,&novi->idm2);
        novi->rasporedjen = 0;
        DodajSortirano(&glava,novi);
    }

    fclose(in);
    return glava;
}

//funkcija koja dodaje  novog studenta u listu
void DodajElement(Element **glava,Element *novi)
{
    if(*glava == NULL){
        *glava = novi;
        return;
    }

    Element *pom = *glava;
```

```
    while(pom->sledeci)
         pom = pom->sledeci;
    pom->sledeci = novi;
}

void Rasporedi(Student *glavaS,Mentor *glavaM)
{
    Student *pom = glavaS;

    while(pom){
        Element *novi = (Element*)malloc(sizeof(Element));
        novi->student = pom;
        novi->sledeci = NULL;

        Mentor *mentor = NadjiMentora(pom->idm1,glavaM);
        if(mentor == NULL || mentor->trenutno >= mentor->max){
            mentor = NadjiMentora(pom->idm2,glavaM);
            if(mentor == NULL){
                pom->rasporedjen = 0;
            }
            else{
                if(mentor->trenutno < mentor->max){
                    DodajElement(&mentor->studenti,novi);
                    pom->rasporedjen = 1;
                    mentor->trenutno++;
                }
            }
        }
        else{
                DodajElement(&mentor->studenti,novi);
                pom->rasporedjen = 1;
                mentor->trenutno++;
        }

        pom = pom->sledeci;
    }
}

//funkcija koja pronazali mentora sa najvise slobodnih mesta
Mentor *MaxMesta(Mentor *glava)
{
    Mentor *maxM = glava;
    glava = glava->sledeci;

    while(glava){
        if((maxM->max - maxM->trenutno) < (glava->max - glava->trenutno))
            maxM = glava;

        glava = glava->sledeci;
    }

    return maxM;
}

void RasporedNerasporedjenih(Student *glavaS,Mentor *glavaM)
{
    Student *pom = glavaS;
    Mentor *mentor = NULL;

    while(pom){
        if(!pom->rasporedjen){
            mentor = MaxMesta(glavaM);
            Element *novi = (Element*)malloc(sizeof(Element));
            novi->student = pom;
            novi->sledeci = NULL;
```

```
            DodajElement(&mentor->studenti,novi);
            pom->rasporedjen = 1;
            mentor->trenutno++;
        }
        pom = pom->sledeci;
    }
}

void Ispis(Mentor *glava)
{
    while(glava){
        float procenat = (100.0 * glava->trenutno)/glava->max;
        printf("%d %d %f\n",glava->id,glava->max - glava->trenutno,procenat);

        glava = glava->sledeci;
    }
}

void Obrisi(Student *student,Mentor **mentor)
{
    Element *pret = NULL;
    Element *pom = (*mentor)->studenti;

    while(pom && pom->student->id != student->id){
        pret = pom;
        pom = pom->sledeci;
    }

    if(pret == NULL){
        Element *temp = pom;
        (*mentor)->studenti = pom->sledeci;
        free(temp);
    }
    else{
        pret->sledeci = pom->sledeci;
        free(pom);
    }
}

void Promena(Student *glavaS,Mentor *glavaM)
{
    int idS,idM;
    scanf("%d%d",&idS,&idM);

    Mentor *mentor = NadjiMentora(idM,glavaM);
    Student *student = glavaS;
    while(student && student->id != idS)
          student = student->sledeci;

    if(mentor && mentor->max > mentor->trenutno){
        Obrisi(student,&mentor);
        Element *novi = (Element*)malloc(sizeof(Element));
        novi->student = student;
        novi->sledeci = NULL;

        DodajElement(&mentor->studenti,novi);
        mentor->trenutno++;
    }
}

int main()
{
    Mentor *mentori = UcitajMentore("Mentori.txt");
    Student *studenti = UcitajStudente("Studenti.txt");
    Rasporedi(studenti,mentori);
```

```
    RasporedNerasporedjenih(studenti,mentori);
    Ispis(mentori);
    Promena(studenti,mentori);

    return 0;

}
```