Strukture podataka
(2OER7O02)

Ispitni zadaci – pregled

Vežbe

Šta ćemo raditi

- Polja
- Lančane liste
- Magacin, red i dvostrani red
- *Hash* tablice

1

Polja

Ispitni zadaci

Polja

Polinom stepena N zadat je poljem svojih koeficijenata. Napisati funkciju **double* GetDerivate(int m, double x)**, koja određuje i vraća m-ti izvod datog polinoma u tački x.

$4x^9 + 8x^7 + 11x^5 + 3x^2 + 26x + 17$

0	1	2	3	4	5	6	7	8	9
17	26	3	0	0	11	0	8	0	4

0	1	2	3	4	5	6	7	8	9
26	6	0	0	55	0	56	0	36	0

Polja

Napisati funkciju ***double* MulMat(int tip, int M, double* mat1, double* mat2)***, koja množi dve trougaone matrice dimenzija MxM, zapamćene kao vektori (i to samo nenulti elementi). Ukoliko je tip=1, matrice su donje trougaone, a ako je tip=2, matrice su gornje trougaone. Voditi računa o optimalnosti rešenja (ne množiti nepotrebne članove). Matrica koja se vraća istog je tipa kao i ulazne matrice.

Trougaona matrica

a_{00}	0	0	...	0
a_{10}	a_{11}	0	...	0
a_{20}	a_{21}	a_{22}	...	0
...	...	...	...	...
a_{m0}	a_{m1}	a_{m2}	...	a_{mm}

i = 0 offset = 0

i = 1 offset = 1

i = 2 offset = 3

i = 3 offset = 6

i = 4 offset = 10

i = k offset = 1+2+…+k = k*(k+1)/2

a(i,j) = vec[i *(i+1) / 2 + j]; i ≥ j

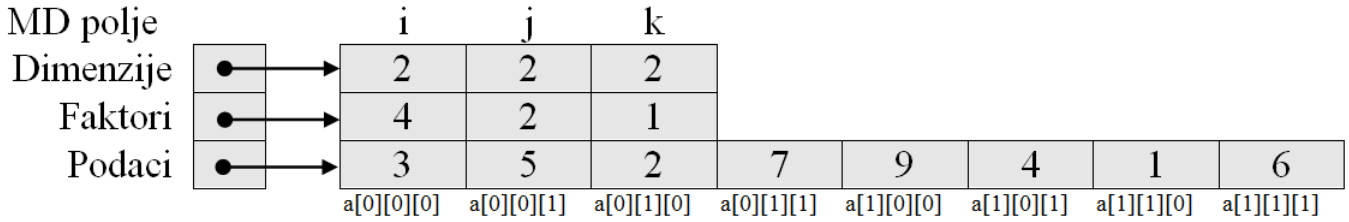
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a_{00}	a_{10}	a_{11}	a_{20}	a_{21}	a_{22}	a_{30}	a_{31}	a_{32}	a_{33}	a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	...

Množenje trougaonih matrica

$$\begin{bmatrix} a_{00} & 0 & 0 & \dots & 0 \\ a_{10} & a_{11} & 0 & \dots & 0 \\ a_{20} & a_{21} & a_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ a_{m0} & a_{m1} & a_{m2} & \dots & a_{mm} \end{bmatrix} \times \begin{bmatrix} b_{00} & 0 & 0 & \dots & 0 \\ b_{10} & b_{11} & 0 & \dots & 0 \\ b_{20} & b_{21} & b_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ b_{m0} & b_{m1} & b_{m2} & \dots & b_{mm} \end{bmatrix} = \begin{bmatrix} c_{00} & 0 & 0 & \dots & 0 \\ c_{10} & c_{11} & 0 & \dots & 0 \\ c_{20} & c_{21} & c_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ c_{m0} & c_{m1} & c_{m2} & \dots & c_{mm} \end{bmatrix}$$

Polja

Napisati konstruktor i funkciju za dodavanje vrednosti u 5-to dimenzionalno polje. Konstruktorom zadati maksimalnu vrednost svakog od indeksa. Smatrati da indeksi počinju od 0.



Polja

Data su dva niza: ***data*** dužine ***m*** i ***index*** dužine ***n***. Niz ***index*** sadrži indekse elemenata niza ***data*** koji su različiti od nule sortirane u *rastućem* redosledu. Napisati funkciju koja postavlja svaki element jednak nuli (sa indeksom ***k***) na osnovu dva najbliža nenulta elementa (sa indeksima ***p*** i ***q***, gde je ***p<q***) po sledećoj formuli **data[k] = ((q-k)*data[p] + (k-p)*data[q]) / (q-p)**. Voditi računa o efikasnosti rešenja.

Data:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
17	0	0	0	0	11	0	0	0	4	0	0	3	0	0	11	0	18	0	24

Index:

0	1	2	3	4	5	6
0	5	9	12	15	17	19

2

Lančane liste

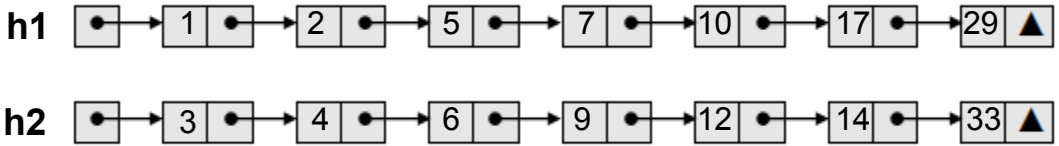
Ispitni zadaci

Lančane liste

Aplikacija vodi evidenciju o broju pobeda aktivnih tenisera u toku karijere. Informacije o teniserima su zapamćene u lančanoj listi uređenoj po broju pobeda. Za svakog tenisera se pamti jedinstveni broj (int), ime i prezime i broj pobeda. Nakon svakog teniskog turnira se ažuriraju bodovi svih učesnika turnira. Napisati funkciju **void UpdatePlayer(char *name, int noWin)** koja ažurira (povećava) broj pobeda zadatog tenisera, čije je ime **name**, za zadatu vrednost **noWin**, ali tako da korišćena struktura ostane uređena.

Lančane liste

Napisati funkciju **Node* Merge(Node* h1, Node* h2)** koja „meša“ dve uređene jednostruko spregnute liste u treću, takođe uređenu, listu. Smatrati da su **h1** i **h2** pokazivači na početke dve liste uređene u rastući redosled. Mešanjem se početne liste uništavaju. Funkcija vraća pokazivač na početak novonastale liste.



Lančane liste

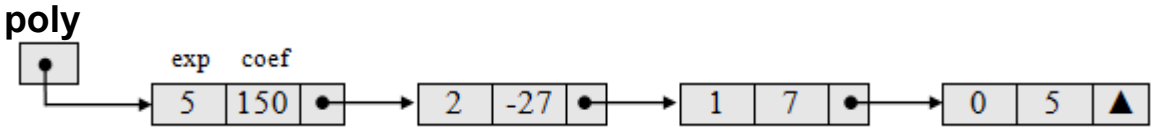
Napisati funkciju za spajanje tri uređene jednostruko ulančane liste u jednu (takođe uređenu) – **Node* Merge(Node* h1, Node* h2, Node* h3)**. Funkcija kao parametre preuzima pokazivače na početke ulaznih lančanih listi, a vraća pokazivač na početak novoformirane liste. Dozvoljen je samo jedan prolazak kroz svaku listu. Ocenjuje se efikasnost rešenja.

Lančane liste

Napisati funkciju za spajanje **n** uređenih jednostruko ulančanih listi u jednu (takođe uređenu) – **Node* Merge(Node* h, int n)**. Funkcija kao parametre preuzima pokazivače na početke ulaznih lančanih listi, a vraća pokazivač na početak novoformirane liste. Dozvoljen je samo jedan prolazak kroz svaku listu, računajući i odredišnu listu. Ocenjuje se efikasnost rešenja.

Lančane liste

Napisati funkciju **Node* MakeDerivate(Node* poly, int ext)**, koja od polinoma koji je prosleđen kao parametar funkcije (**poly** je pokazivač na prvi elemenat liste), formira novi polinom koji predstavlja **ext**-i izvod presleđenog polinoma. Polinom je zapamćen kao jednostruko ulančana lista elemenata koji predstavljaju nenulte članove polinoma. Lista je uređena u opadajući redosled po eksponentima.

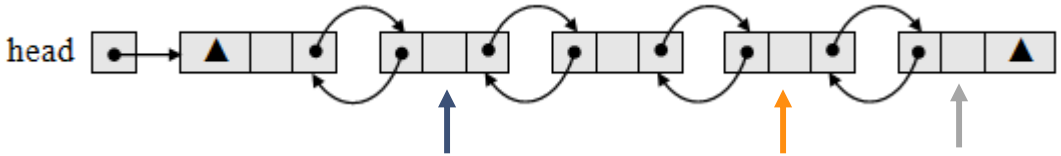


Lančane liste

Napisati funkciju za sortiranje elementa jednostruko povezane lančane liste metodom **Insertion Sort**.

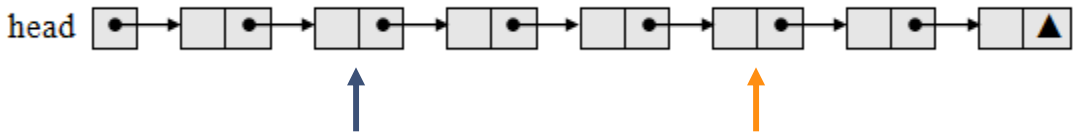
Lančane liste

Napisati funkciju za sortiranje dvostruko spregnute lančane liste metodom ***Selection Sort***. Sortiranje vršiti zamenom linkova, a ne info polja.



Lančane liste

Napisati funkciju **void Transpose2(int key)**, koja pronalazi čvor u jednostruko ulančanoj listi koji sadrži vrednost **key** (ukoliko postoji) i prebacuje ga dve pozicije ispred promenom linkova. Ako je nemoguće prebaciti element 2 mesta unapred, postaviti ga na početak liste.



Lančane liste

Jednostruko spregnuta lista sadrži celobrojne vrednosti uređene u rastući redosled. Napisati funkciju **void Update(int value, int add)** koja povećava zadatu vrednost (**value**) za zadati pozitivan broj (**add**) tako da lančana lista ostane uređena. Smatrati da su vrednosti u lančanoj listi jedinstvene.

Lančane liste

Napisati funkciju **void LList::groupDuplicates()**, koja u dinamičkoj lančanoj listi grupiše (nadovezuje) sve čvorove čiji su info delovi jednaki. Smatrati da su info delovi celobrojnog tipa. Nije dozvoljeno korišćenje pomoćnih funkcija za rad sa lančanom listom.

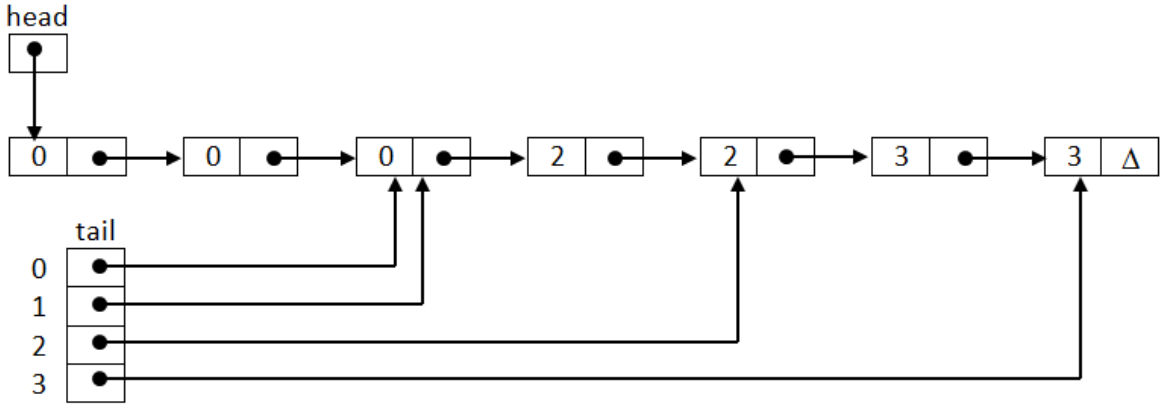
3

Magacin, red i dvostrani red

Ispitni zadaci

Magacin, red i dvostrani red

Projektovati klasu za rad sa redom koji ima 4 nivoa prioriteta (0 – najviši, 3 – najniži), i implemetirati funkcije za dodavanje u red – **void Add(int prioritet, Data* podaci)** i čitanje iz reda – **Data* Get()**. Funkcija za čitanje uklanja podatak iz reda, pri čemu se uklanja najranije dodat podatak najvišeg prioriteta.

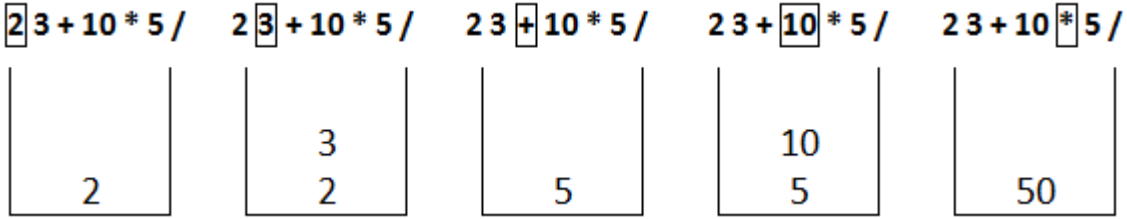


Magacin, red i dvostrani red

Napisati funkciju **void AddPriority(float pri, Data* pData)**, koja dodaje podatak sa prioritetom **pri** na koji ukazuje pokazivač **pData** u odgovarajući red sa prioritetom. Prioritet može imati bilo koju realnu vrednost. Novododati podatak se stavlja iza svih podataka istog ili većeg prioriteta, koji su prethodno dodati u red.

Magacin, red i dvostrani red

Napisati funkciju koja određuje vrednost izraza zadatog u postfiks notaciji (npr. „2 3 + 10 * 5 /“). Deklarisati pomoćnu strukturu koja se koristi za određivanje izraza i napisati funkcije dodavanja i brisanja u pomoćnu struktura. Izraz dat kao niz karaktera tako da su svi brojevi i operatori razdvojeni tačno jednim blanko znakom.

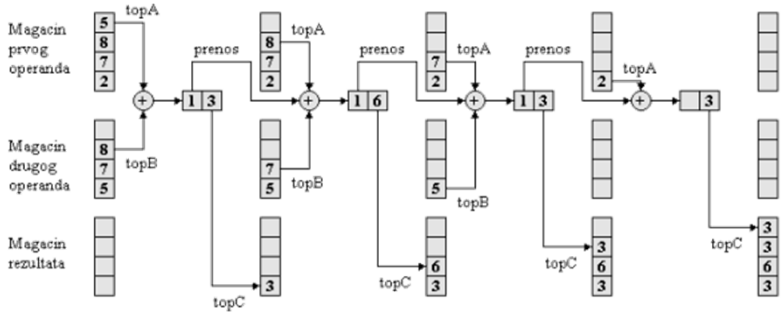


Magacin, red i dvostrani red

Napisati funkciju **bool CheckExpression(char* exp)** koja proverava da li su zagrade u navedenom izrazu pravilno zatvorene. U izrazu se mogu pojaviti male, srednje i velike zagrade. Npr. izraz {[5+q] * [(r-e) + (w+5)]} je ispravan. Smatrati da postoji implementirana pomoćna struktura koja omogućuje rešavanje zadatka.

Magacin, red i dvostrani red

Napisati funkciju **char* AddLong(char* op1, char* op2)**, koja vrši sabiranje dva jako dugačka cela broja zadata u obliku niza karaktera i vraća rezultat u istom obliku. Smatrati da postoji implemetirana odgovarajuća osnovna struktura potrebna za realizaciju ovog zadatka.



Magacin, red i dvostrani red

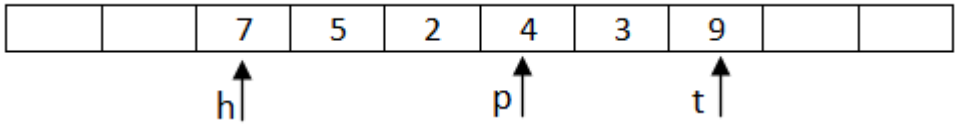
Napisati funkciju **double calcPrefix(char* inStr)**, koja izračunava i vraća vrednost aritmetičkog izraza, zadatog u prefiks notaciji ulaznim nizom karaktera **inStr**. Smatrati da su svi operandi jednocifreni celi brojevi, a aritmetičke operacije: +, -, * i /. Takođe, podrazumevati da u **inStr** nema blanko znakova i da je izraz pravilno zadat, kao i da je pomoćna struktura potrebna za rešavanje problema već implementirana. (Primer: funkcija treba da vrati 216 za izraz '**+45-934').

Magacin, red i dvostrani red

Služba za obradu zahteva korisnika je podelila obradu na dve faze: **FirstStep** i **SecondStep**. Zahtevi se obrađuju u redosledu u kojem pristižu i predstavljeni su samo celobrojnim identifikatorom. Napisati funkcije:

- **void AddRequest(int idReq)**, koja dodaje zahtev za obradu sa identifikatorom **idReq** na kraj reda (pokazivač **t**).
- **void FirstStep()**, koja vrši prvu fazu obrade, zahteva na koju ukazuje pokazivač **p** i pomera dati pokazivač. Ako su svi zahtevi u redu prošli prvu fazu obrade, ova funkcija ne radi ništa.
- **void SecondStep()**, koja vrši drugu fazu obrade, zahteva sa početka reda (pokazivač **h**) i uklanja ga iz reda. Ako zahtev na početku reda nije prošao prvu fazu, ova funkcija ne radi ništa.

Funkcije **AddRequest**, **FirstStep** i **SecondStep** mogu se pozivati proizvoljnim redosledom.



4

Hash tablice

Ispitni zadaci

Hash tablice

Implementirati na programskom jeziku C++ heš funkciju unsigned **int h(ScatterObject o)**, sekundarnu funkciju sa kvadratnim traženjem unsigned **int g(int i)** i funkciju za smeštanje objekta **void insert(ScatterObject o)** u heš tablicu sa otvorenim adresiranjem veličine **m**. Smatrati da su ključevi celobrojni.

Hash tablice

Retko posednuta matrica zapamćena je rasutoj tablici (heš tablica sa otvorenim adresiranjem). Napisati funkcije za: inicijalizaciju – **void init(int m, int n)**, primarnu transformaciju – **int h(int i, int j)**, sekundarnu transformaciju – **int g(...)**, umetanje elementa – **void insert(int i, int j, double val)**, čitanje elementa – **double get(int i, int j)** i množenje dve retkoposednute matrice – **mul()**. Smatrati da je tablica dimenzija m × n, i da ima manje od 20% nenultih elemenata tipa *double*. Samo nenulti elementi se pamte u tablici. **Napomena:** Implemetaciju prilagoditi konkretnom problemu. Generičke funkcije neće biti bodovane.

Hash tablice

Red sa prioritetom implementiran je u obliku *hash* tablice sa unutrašnjim ulančanjem (sa zasebnim prostorom za smeštanje sininima). Prioriteti su definisani celim brojevima u opsegu 0-299. Manja vrednost označava viši prioritet. Maksimalni broj elemenata u redu je 2000. Napisati funkciju za dodavanje i čitanje iz reda.

Hash tablice

Napisati funkciju **Data* Get(int key)**, koja u hash tablici sa unutrašnjim ulančavanjem vraća elemenat sa zadatim ključem i istovremeno premešta ovaj element na prvo mesto u listi sinonima, zadržavajući redosled ostalih sinonima u listi.

Hash tablice

Studentska služba čuva sve podatke o aktivnim studentima (upisani studenti koji nisu diplomirali) u registrima označenim jedinstvenom oznakom (ID). Kako bi se brže pronašao registar za željenog studenta, koristi se rasuta tablica sa otvorenim adresiranjem pri čemu se kao ključ koristi broj indeksa. U tablici se čuva broj indeksa, ime i prezime studenta i broj njegovog registra. Napisati funkciju **int Odredi(int index)** koja određuje broj sinonima studenta sa brojem indeksa **index** u rasutoj tablici.

Hash tablice

Spisak studenata koji su položili ispit iz predmeta Strukture podataka se pamti u heš tablici sa otvorenim adresiranjem koja obezbeđuje najbrže pretraživanje po broju poena. Za svakog studenta sa spiska se pamti broj indeksa, ime i prezime studenta i broj poena. Napisati funkciju **void Update(int ind, int ptsOld, int ptsNew)** koja efikasno pronalazi studenta na osnovu trenutnog broja poena **ptsOld,** sa indeksom **ind**, uvećava broj poena na **ptsNew** i ažurira strukturu tako da ostane validna.

Hash tablice

28.06.2014.

Napisati funkciju **void InsertItems(struct item* items, int no)**, koja polje od **no** elemenata smešta u rasutu tablicu sa unutrašnjim ulančavanjem. Tablica koristi zaseban memorijski prostor za smeštanje sinonima, veličine 40% od ukupne veličine tablice. Potrebno je implementirati i **primarnu** i **sekundarnu** transformaciju. Struktura **item** sadrži sledeća polja: ime – char[20], broj_stanovnika – int, godina_osnivanja – short i prihodi – double.

Hash tablice

Retko posednuta dvodimenzionalna matrica realnih brojeva, dimenzija N×M, zapamćena je u rasutoj tablici. Smatrati da matrica ima manje od 20% nenultih elemenata. Napisati funkcije: **ScatterTable(int N, int M)** – konstruktor klase, **int H(int i, int j)** – primarna transformacija, **int G(int h, int iter)** – sekundarna transformacija (**h** je početna adresa, koju je vratila primarna transformacija, a **iter** je brojač pokušaja) i **void Insert(double data, int i, int j)** – umetanje elementa u tablicu.

Pitanja