1 Algoritmi i strukture podataka - 2. dvočas

1.1 Liste

Karakteristične operacije za listu su: kreiranje čvora (O(1)), dodavanje na početak (O(1)), dodavanje na kraj (O(n)), dodavanje sortirano (O(n)), pretraživanje (O(n)), ispisivanje liste (O(n)), brisanje liste (O(n)). Ukoliko nije drukčije navedeno u zadacima se podrazumeva da svaki čvor liste sadrži ceo broj.
Osnovne funkcije za rad sa listama u C-u

Svi zadaci se odnose na jednostruko povezane liste.

1. Napisati *nerekurzivnu* funkciju prostorne složenosti $O(1)$ u C-u koja obrće datu listu, tj. preusmerava pokazivače tako da prvi element postaje poslednji, drugi postaje pretposlednji,..., poslednji postaje novi početak liste.
Rešenje:

```
cvor* obrni_listu (cvor *glava){
  cvor* pr = NULL;
  cvor* tr = glava;
  cvor* sl;

  while (tr != NULL){
    sl = tr->sl;

    tr->sl = pr;
    pr = tr;
    tr = sl;
  }

  return pr;
}
```

2. Napisati *nerekurzivnu* funkciju u C-u koja spaja dve sortirane liste u jednu koristeći nov memorijski prostor maksimalne veličine O(1).
Rešenje:

```
cvor* spoji_liste (cvor *lista1, cvor *lista2){
  cvor *t, *k, *p, *glava;

  if (lista1 == NULL)
    return lista2;
  else if (lista2 == NULL)
    return lista1;

  if (lista1->n > lista2->n){
    t = lista1;
    lista1 = lista2;
    lista2 = t;
  }

  glava = lista1;
  while (lista1->sl != NULL && lista2 != NULL){

    // Ukoliko je element liste 2 po velicini izmedju 2 elementa liste 1,
    // smestamo njega i sve naredne koji zadovoljavaju taj uslov u listu 1
    if (lista1->n < lista2->n && lista1->sl->n > lista2->n){

      // Pamtimo sledeci u prvoj listi
      t = lista1->sl;

      lista1->sl = lista2;
```

```
      while (lista2->sl != NULL && lista2->sl->n < t->n)
        lista2 = lista2->sl;

      // Pamtimo sledeci u drugoj listi
      p = lista2->sl;

      // Povezujemo drugu listu sa prvom listom
      lista2->sl = t;
      // Prelazimo na naredne elemente u listama
      lista1 = t;
      lista2 = p;
    }
    // Ukoliko je lista1 = 1,3, a lista2 = 4, prelazimo na naredni u prvoj listi
    // to jest na cvor koji sadrzi 3
    else
      lista1 = lista1->sl;
  }
  // Stigli smo do kraja prve liste, produzujemo je da sadrzi ostatak druge liste
  if (lista1->sl == NULL)
    lista1->sl = lista2;

  return glava;
}
```

3. Napisati *nerekurzivnu* funkciju u C-u koja briše sve čvorove u listi u kojima se pojavljuje broj koji se prenosi kao argument. Kolika je složenost napisane funkcije?

4. Napisati *nerekurzivnu* funkciju koja od listi $L1$ i $L2$ formira listu L koja sadrži alternirajuće raspoređene elemente lista $L1$ i $L2$ (prvi element iz $L1$, prvi iz $L2$, drugi iz $L1$, drugi iz $L2$, itd). Ne formirati nove čvorove već samo postojeće čvorove rasporediti u jednu listu.

1.2 Stek

Sa stekom se radi po LIFO principu (Last In First Out). Osnovne 3 operacije koje se izvode na steku su: PUSH (dodaj element na vrh steka), POP (skini i vrati element sa vrha steka), TOP (vrati element sa vrha steka bez skidanja). Sve ove operacije su složenosti O(1). Ukoliko se pokuša sa dodavanjem elementa na stek koji je već pun dolazi do prekoračenja, a ukoliko se pokuša sa skidanjem elementa sa praznog steka dolazi do potkoračenja. Često se implementiraju i funkcije za proveru da li je stek pun/prazan kao i funkcija koja vraća broj elemenata na steku. Stek predstavljamo preko niza i preko liste.

Napomena: ideja steka i reda je da korisnika ne zanima sama implementacija vec samo interfejs. U pseudokodovima nećemo da se bavimo implementacijom ovih struktura, već ćemo ih koristiti kao da su već implementirane.

5. Jednodimenzioni niz može iskoristiti za smeštanje dva steka tako da prvi stek raste nagore sa levog kraja niza, a drugi stek raste nadole sa desnog kraja niza. Napisati funkcije za kreiranje steka, proveru da li je stek prazan, proveru da li je stek pun kao i funkcije PUSH, POP i TOP. Koja je vremenska složenost svake od operacija u O notaciji?
Rešenje

Pri implementaciji steka preko niza, ako treba proširiti stek onda je najefikasnije udvostručiti njegovu veličinu (a ne povećavati svaki put veličinu za 1). Kada treba smanjiti veličinu niza koji je iskorišćen za stek? Efikasna rešenja smanjuju veličinu steka na polovinu kada popunjenost dostigne četvrtinu niza. U oba slučaja operacije PUSH i POP imaju najgori slučaj $O(n)$ ali se pokazuje da je prosečna složenost ovih operacija $O(1)$.

Obe implementacije (preko listi i preko niza) imaju svoje prednosti i mane. Kod listi je potrebno uložiti dodatan trud za implementaciju a zauzima se i više prostora i troši više vremena u proseku (jer je potrebno čuvati i menjati i pokazivače) ali je za svaku operaciju garantovano konstantno vreme. Kod nizova se koristi manje memorije ali je najgori slučaj za 2 operacije $O(n)$. Naglo usporenje pri proširivanju niza može biti vrlo nepogodno za pojedine aplikacije. Ako ovo nije bitno i gleda se ukupno vreme, onda korišćenje nizova može predstavljati bolji izbor.

6. Napisati pseudokod algoritma koji proverava da li su zagrade u datoteci ispravno uparene. Na primer, ispravno su uparene zagrade ()[()] a nisu ispravno uparene {(}). Smatrati da je maksimalna veličina steka ograničena na 100 elemenata.

Rešenje:

Algoritam Proveri_uparenost (ime_dat)
Ulaz ime datoteke - ime_dat
Izlaz 1 ako su zagrade ispravno uparene, 0 ako nisu

```
{
  s = kreiraj_stek();

  // Funkcija ucitaj_sledecu_zagradu preskace ostale karaktere i vraca sledecu zagradu
  while ( (z = ucitaj_sledecu_zagradu (ime_dat)) != 0)
    if (z == '(' || z == '{' || z == '['){
      if (s.velicina_steka() == 100)
        error ("prekoracenje steka");
      s.dodaj_na_stek (z);
    }
    else {
      if (s.stek_prazan())
        return 0;
      pom = s.skini_sa_steka ();
      if ( (pom == '(' && z == ')') ||
           (pom == '{' && z == '}') ||
           (pom == '[' && z == ']'))
        continue;
      else return 0;
    }

  if (s.stek_prazan())
    return 1;
  else
    return 0;
}
```

7. Napisati pseudokod algoritma koji proverava da li su HTML etikete u datoteci ispravno uparene. Smatrati da su otvorene etikete oblika *<etiketa>* a zatvorene oblika *</etiketa>*.

8. *Modifikacija Dijkstrinog algoritma sa 2 steka:* Algoritam čita datoteku u kojoj se nalazi aritmetički izraz u infiksnoj notaciji. Dozvoljeni su operatori + i *, a svaki izraz sa dva operanda je okružen zagradama. Kreiraju se dva steka, jedan za operande a drugi za operatore. Kada se naiđe bilo na operand ili operator, stavljaju se na odgovarajući stek. Otvorene zagrade se preskaču a kada se naiđe na zatvorenu zagradu, sa stekova se skidaju 2 operanda i operator, izvršava se operacija i rezultat smešta na stek operanada. Napisati pseudokod ovog algoritma. Uključiti i što detaljniju proveru ispravnosti izraza nad kojim se radi (prekinuti algoritam sa porukom o greški u slučaju neispravnosti).

Rešenje:

Algoritam Dijkstrin_algoritam (ime_dat)
Ulaz ime datoteke - ime_dat
Izlaz Rezultat aritmetičkog izraza, poruka o greški u slučaju neispravnosti

```
{
  operandi = kreiraj_stek();
  operatori = kreiraj_stek();
  while ( (z = ucitaj_sledeci_string(ime_dat)) != 0)
    if (poredi_stringove (z, "+") == true || poredi_stringove (z, "*") == true)
      operatori.dodaj_na_stek(z);
    else if (z.jeste_broj() == 1)
      operandi.dodaj_na_stek(z);
```

```
    else if (poredi_stringove (z, "(") == true)
      continue;
    else if (poredi_stringove (z, ")") == true){
      if (operandi.velicina() < 2 || operatori.velicina() < 1)
        greska ("Nepravilan aritmeticki izraz");
      op1 = operandi.skini_sa_steka();
      op2 = operandi.skini_sa_steka();
      op = operatori.skini_sa_steka();
      rez = izvrsi_operaciju (op1, op, op2);
      operandi.dodaj_na_stek(rez);
    }
    else
      greska ("Nepoznat string na ulazu");

  if (operandi.velicina() != 1 || operatori.velicina() != 0)
    greska ("Nepravilan aritmeticki izraz");

  return operandi.skini_sa_steka();
}
```

1.3 Red

Sa redom se radi po FIFO principu (First In First Out). Dve osnovne operacije su dodavanje elementa na kraj reda i brisanje elementa sa početka reda (obe su složenosti O(1)). Red se kao i stek najčešće implementira preko nizova i listi.

1.4 Kviksort

Složenost ovog algoritma je u najgorem sluăju $O(n^2)$ (kada se za pivot bira uvek najmanji element što je slučaj i u algoritmu koji je ovde naveden). U praksi se koristi algoritam koji za pivot biva proizvoljan (random) element niza i tada je *prosečna* složenost algoritma $O(n \log n)$. Dodatan memorijski prostor koji se koristi je veličine $O(1)$. Napomena 1: kada se ispituje složenost algoritma sortiranja izračunava se broj poređenja tokom njegovog izvršavanja.

Algoritam Razdvajanje (X, $Levi$, $Desni$)
Ulaz X (niz), $Levi$ (leva granica niza), $Desni$ (desna granica niza)
Izlaz indeks S takav da je $X[i] \leq X[S]$ za sve $i \leq S$ i $X[j] > X[S]$ za sve $j > S$
{
$pivot = X[Levi]$;
$L = Levi$; $D = Desni$;
while $L < D$
{
while $X[L] \leq pivot$ && $L \leq Desni$
$L = L + 1$;
while $X[D] \geq pivot$ && $D \geq Levi$
$D = D - 1$;
if $L < D$
zameni $X[L], X[D]$;
}
$S = D$;
zameni $X[Levi], X[S]$;
return S;
}

Algoritam Q_Sort (X, $Levi$, $Desni$)
Ulaz X, leva i desna granica dela niza koji se sortira
Izlaz sortirani deo niza X
{
if $Levi < Desni$
{

$S = Razdvajanje(X, Levi, Desni)$;
$Q\_sort(X, Levi, S-1)$;
$Q\_sort(X, S+1, Desni)$;
}
}

Algoritam Kviksort (X, n)
Ulaz X (niz od n brojeva)
Izlaz sortirani niz X
{
$Q\_sort(X, 1, n)$;
}

8. Algoritmom Kviksort prikazati postupak sortiranja brojeva 8, 11, 5, 14, 3, 9, 10, 2, 12, 1, 15, 7, 6, 4, 13.
Rešenje: (pivoti su podebljani)

8	11	5	14	3	9	10	2	12	1	15	7	6	4	13	
8	4	5	14	3	9	10	2	12	1	15	7	6	11	13	(zamenjeni 11 i 4)
8	4	5	6	3	9	10	2	12	1	15	7	14	11	13	(zamenjeni 14 i 6)
8	4	5	6	3	7	10	2	12	1	15	9	14	11	13	(zamenjeni 9 i 7)
8	4	5	6	3	7	1	2	12	10	15	9	14	11	13	(zamenjeni 10 i 1; L pokazuje na br. 12, D na br. 2)
2	4	5	6	3	7	1	**8**	12	10	15	9	14	11	13	(zamenjeni 8 i 2)
2	4	5	6	3	7	1									(bira se pivot za prvu polovinu niza - do broja 8)
2	1	5	6	3	7	4									(zamenjeni 4 i 1; L pokazuje na br. 5, D na br. 1)
1	**2**	5	6	3	7	4									(zamenjeni 2 i 1)
		5	6	3	7	4									(bira se pivot za podniz od 5 do 4)
		5	4	3	7	6									(zamenjeni 6 i 4)
		3	4	**5**	7	6									(zamenjeni 5 i 3)
					7	6									(bira se pivot za podniz 7, 6)
					6	**7**									(zamenjeni 7 i 6)
								12	10	15	9	14	11	13	(bira se pivot za podniz 12,...,13)
								12	10	11	9	14	15	13	(zamenjeni 15 i 11)
								9	10	11	**12**	14	15	13	(zamenjeni 12 i 9)
								9	10	11					(bira se pivot za podniz 9, 10, 11, nema promena u ovo
												14	15	13	(bira se pivot za podniz 14, 15, 13)
												14	13	15	(zamenjeni 15 i 13)
												13	**14**	15	(zamenjeni 14 i 13)

Prepisivanjem zadnjeg broja u svakoj koloni dobija se sortirani niz. Formulacija algoritma je preuzeta iz profesorove knjige. U knjizi se nalazi primer sortiranja još jednog niza.