Algoritmi in podatkovne strukture – 2
Prvi kolokvij (2011/12)

Kolokvij morate pisati posamič. Pri reševanju je literatura dovoljena. Pri odgovarjanju bodi natančni in: (i) odgovarjajte *na zastavljena* vprašanja; in (ii) odgovorite na *vsa* zastavljena vprašanja – če boste odgovarjali na vsa vprašanja, lahko dobite dodatne točke.

Čas pisanja izpita je 60 minut.

Veliko uspeha!

NALOGA	TOČK	OD TOČK	NALOGA	TOČK	OD TOČK
1			3		
2			4		

IME IN PRIIMEK: _____

ŠTUDENTSKA ŠTEVILKA: _____

DATUM: _____

PODPIS: _____

1. naloga: *Uvod in osnove.* Imamo polje n celih števil `bla[n]`.

VPRAŠANJA:

1. Napišite rekurzivno funkcijo, ki sprejme kot parameter polje ter izračuna povprečno vrednost elementa.

2. Dokažite pravilnost svoje funkcije.

   NAMIG: Uporabite indukcijo.

3. Ocenite koliko *dostopov* do podatkov izvede vaš program in kolikšna je prosotorska zahtevnost (bodite pri slednjem natančni).

4. Pokažite koliko dostopov do podatkov je najmanj potrebnih za omenjeni izračun.

   NAMIG: Uporabite dokaz s protislovjem.

2. naloga: *Drevesa in disjunktne množice.*

VPRAŠANJA:

1. Imamo 4-iško iskalno drevo in v njem 2011 listov. Najmanj koliko notranjih vozlišč ima takšno drevo? Utemeljite odgovor.

2. Kakšna je najmanjša in kakšna največja globina takšnega drevesa? Utemeljite odgovor.

3. Na predavanjih smo spoznali podatkovno strukturo za upravljanje z disjunktnimi množicami. Zapišite definicijo vozlišča v takšni strukturi.

   NAMIG: Upoštevajte posebne vrednosti za reference pri predstavniku množice.

4. Zapišite algoritem `Find(elt)`, ki vrne ime množice, kateri pripada element `elt` – dejansko vrne predstavnika te množice.

3. naloga: *Številska drevesa.* Imamo naslednje elemente

$$K = 1011, N = 1110, D = 0100, F = 0110, L = 1100, O = 1111, M = 1101 \quad . \tag{1}$$

VPRAŠANJA:

1. Iz omenjenih elementov zgradite številsko drevo, ki je stisnjeno po plasteh (*level-compressed trie*). Stiskajte vedno po dve plasti.

2. Peter Zmeda je napisal nov algoritem za vstavljanje v patricijino številsko drevo:

```
Vstavi(ključ, vozlišče):
  IF NOT vozlišče.list THEN
    levo= (ključ[vozlišče.bit] == 0)
    IF levo THEN vstavi(ključ, vozlišče.levo)
    ELSE         vstavi(ključ, vozlišče.desno)
  ELSE IF vozlišče.ključ <> ključ THEN
    vozlišče.bit=
      prvi bit, kjer se vozlišče.ključ in ključ razlikujeta
    vozlišče.list= FALSE
    levo= (ključ[vozlišče.bit] == 0)
    IF levo THEN
      vozlišče.levo=  NEW(ključ)
      vozlišče.desno= NEW(vozlišče.ključ)
    ELSE
      vozlišče.levo=  NEW(vozlišče.ključ)
      vozlišče.desno= NEW(ključ)
    ENDIF
  ELSE
    // ključ je že v drevesu
  ENDIF
  RETURN
```

   ali algoritem deluje pravilno? Utemeljite odgovor.

3. Pri številskih drevesih smo videli, da, če naredimo vmesni obhod, dobimo izpisane elemente urejene po velikosti. Ali to velja v drevesu, ki ga zgradi Petrov algoritem? Utemeljite odgovor.

   NAMIG: Če dokazujete, da velja, uporabite indukcijo in če dokazujete, da ne velja, najdite primer, ko ne velja. Če se ne znate odločiti, poskusite vstaviti ključe iz en. (1).

4. naloga: *Slovar.*

VPRAŠANJA:

1. Imamo TTF drevo. Vanj vstavite po vrsti elemente iz en. (1). Seveda, tokrat jih vstavljate kot preproste ključe (K, N, ...). Prikažite vse korake vstavljanja.

2. Na predavanjih smo spoznali podatkovno strukturo preskočnih seznamov kot implementacijo slovarja. Recimo, da bi želeli dodati operacijam *Vstavi*, *Briši* in *Najdi* še operacijo *Levi*, ki vrne: *največji element v slovarju, ki je še manjši od iskanega elementa*. Na primer, pri elementih z en. (1) velja

   $$\texttt{Levi(F)} = \texttt{Levi(E)} = \mathrm{D}\,.$$

   Opišite, kako bi izgledalo iskanje levega elementa na preskočnem seznamu. Morda si pomagate s psevdokodo.

   NAMIG: Premislite, kako pridete do iskanega elementa v različnih primerih.

3. Kakšna je časovna zahtevnost vašega algoritma, če je preskočni seznam idealen – elementi so pravilno razporejeni glede na svoje višine.

4. DODATNO: primerjajte časovno zahtevnost iskanja levega elementa v uravnoteženem dvojiškem iskalnem drevesu.