PODATKOVNE STRUKTURE IN ALGORITMI

1. DOMAČA NALOGA

28.10.2018

1. naloga:

(i) Naslednje funkcije razvrstite v nepadajočem vrstem redu glede na asimptotično rast (asimptotično kompleksnost veliki $\mathcal{O}$):

$$f_1(n) = 2^{2^{1000000}}, f_2(n) = 2^{1000000n}, f_3(n) = \binom{n}{2}, f_4(n) = n\sqrt{n}.$$

Vse odgovore je potrebno utemeljiti!

Primer: Funkcija $f(n) = n$ raste asimptotično počasneje kot funkcija $g(n) = n^2$; z drugimi besedami, $f(n) \in \mathcal{O}(g(n))$, toda $g(n) \notin \mathcal{O}(f(n))$.

(ii) Z uporabo definicije velikega $\mathcal{O}$ pokažite $n^{1+0,001} \notin \mathcal{O}(n)$.

Funkcije si po nepadajačem vrstnem redu sledijo tako: f_1, f_4, f_3, f_2.

Vrstni red je takšen zato, ker f_1 ni odvisna od n, kar pomeni, da je konstantna funkcija in tako sklepamo da narašča s konstanto hitrostjo, zato lahko sklepamo, da ima najhitrejšo asimptotično rast. Sledi ji f_4, ki je zelo hitro naraščajoča funkcija, ampak ni na prvem mestu zato, ker ima počasno asimptotično rast pri visokih vrednostih. Zatem je f_3 , ki je linearna funkcija in počasneje narašča kot f_4, kljub temu, da se f_4 upočasni pri visokih vrednostih. Zadnja je pa f_2, ki pa je eksponentna funkcija, kar pomeni da eksponentno narašča, kaj pa je zelo hitro.

$n^{1+0,001}$ > a*n, to velja v primeru, da je a = 1 in n = 1, če je a večji kot 1 pa naša enačba več ni veljavna, saj se nam vrednost poveča za toliko kolikor je a in je zato desna stran enačbe vedno večja od leve, če hočemo da ta enačba velja more a vesno vsebovati vrednost 1.

2. naloga: Za vsakega izmed spodnjih algoritmov poiščite njegovo časovno zahtevnost.

(i) Algoritem *Algo1(A)*

Vhodni podatek: celoštevilsko polje A velikosti $n \geq 1$

Izhodni podatek: vsota vseh elementov polja A

$s \leftarrow A[0]$
for $i \leftarrow 1$ **to** $n-1$ **do**
  $s \leftarrow s + A[i]$
return s

Časovna zahtevnost algoritma Algo1(A) je n, saj imamo celoštevilsko polje in for loop, kar pomeni, da vstavljamo v naš algoritem polje z n števili in zato, ker to polje v for loopu seštevamo je naša časovna zahtevnost O(n).

(ii) Algoritem *Algo2(A)*

Vhodni podatek: celoštevilsko polje A velikosti $n \geq 1$

Izhodni podatek: vsota vseh predponskih vsot polja A

$s \leftarrow 0$
for $i \leftarrow 1$ **to** $n-1$ **do**
  $s \leftarrow s + A[0]$
  for $j \leftarrow 1$ **to** i **do**
    $s \leftarrow s + A[j]$
return s

Časovna zahtevnost algoritma Algo2(A) je n^2, saj imamo celoštevilsko polje in dva for loopa, kar pomeni, da vstavljamo v naš algoritem polje z n števili in zato, ker hočemo dobiti vsoto vseh predponskih vsot polja A potrebujemo dva for loopa. Vsak for loop rabi n časa, da dokonča svojo nalogo. Tako lahko sklepamo, da zato da dokončata svojo nalogo potrebujeta n*n časa, kar pa je isto kot n^2. Torej naša časovna zahtevnost je 0(n^2).

(iii) Algoritem *Algo3(A)*

Vhodni podatek: celoštevilski polji A in B velikosti $n \geq 1$

Izhodni podatek: število elementov v B, enakih vsoti vseh predponskih vsot v A

$c \leftarrow 0$
for $i \leftarrow 1$ **to** $n-1$ **do**
  $s \leftarrow 0$
  for $j \leftarrow 1$ **to** $n-1$ **do**
    $s \leftarrow s + A[0]$
    for $k \leftarrow 1$ **to** j **do**
      $s \leftarrow s + A[k]$
  if $B[i] == s$ **then**
    $c \leftarrow c + 1$
return c

Časovna zahtevnost algoritma Algo3(A) je n^3, saj imamo tri for loope. Vsak for loop potrebuje, da dokonča svojo nalogo n časa, kar pomeni, da vsi trije potrabujejo skupaj n*n*n časa da dokončajo svojo nalogo. Torej naša časovna zahtevnost je 0(n^3).

3. naloga: Zančna invarianta. Naj bo A celoštevilsko polje dolžine n. Dan je naslednji algoritem:

foo(A)
```
foo(A)
    n ← A.length
    x ← A[1]
    for i ← 2 to n do
        if A[i] > x then
            x = A[i]
    return x
```

(i) Kaj dela algoritem *foo(A)*?

(ii) Napišite zančno invarianto za algoritem *foo(A)*.

(iii) Dokažite, da algoritem *foo(A)* deluje pravilno z uporabo zančne invariante.

i) Naš algoritem foo(A) deluje tako, da vrne največji element v podanem polju. V našem primeru je to polje A.

ii) Na začetku zančne invariante for loopa bi morala x spremenljivka vsebovati največji element našega podpolja.

iii) Na začetku algoritma foo(A) podamo v x vrednost A[1] (x <-- A[1]), če se naša zančna invarianta ustavi pri i, pomeni, da x vsebuje največji element podpolja A[1:i]. Zančna invarianta A[i] > x lahko odda rezultat A[i], kar bi pomenilo, da je to največji element v polju. Nato se vrne na začetek for loopa, kjer je vrednost A[i] nova vrednost A[1], kar pomeni, da se A[i] v bistvu A[1] naloži v x in tako zančna interacija drži. Lahko pa je tudi obratno, da je največji element A[1:i], ki je vsaj toliko velik kot A[i], to pomeni, da je največje število A[1:i+1] enakoA[1:i]. Algoritem ne spremeni vrednosti in zančna invarianta drži. Na koncu vrne zančna invarianta spremenljivko x, ki ima naš največji element polja, kar pomeni, da smo dokazali zančno invarianto.