Tietorakenteet (syksy 2012)

Harjoitus 3 (21.9.2012)

1. $\mathcal{O}$-filosofiaa

(a) Tehtävänä on toteuttaa algoritmi, jolle annetaan n lukua sisältävä taulukko ja joka laskee taulukon lukujen summan. On helppoa keksiä algoritmi, jonka aikavaativuus on $\mathcal{O}(n)$: tavanomainen for-silmukka, joka käy luvut läpi ja laskee niiden summan muuttujaan. Perustele, miksi $\mathcal{O}(n)$ on myös paras mahdollinen aikavaativuus ongelman ratkaisevalle algoritmille.

(b) Tilavaativuus tarkoittaa, kuinka paljon muistia algoritmi käyttää syötteen lisäksi. Ei ole harvinaista, että algoritmin aikavaativuus on $\mathcal{O}(n)$, mutta tilavaativuus on vain $\mathcal{O}(1)$. Tällöin algoritmi tulee toimeen kiinteällä määrällä apumuuttujia. Onko sen sijaan mahdollista, että algoritmin aikavaativuus on $\mathcal{O}(1)$, mutta tilavaativuus on $\mathcal{O}(n)$?

(c) Tarkastellaan taulukkoa, jossa on kokonaislukuja:

5	2	1	3	1

Koko taulukon sisällön voi kutistaa yhteen kokonaislukuun kertomalla peräkkäisiä alkulukuja, joiden potenssit ovat taulukon luvut: $2^5 \cdot 3^2 \cdot 5^1 \cdot 7^3 \cdot 11^1 = 5433120$. Tästä luvusta saa selville alkuperäiset luvut etsimällä luvun alkutekijät ja lukemalla niiden potenssit.

Luvuille tuntuu siis olevan tuhlaavaista varata viisi kohtaa taulukossa, kun yksikin riittäisi:

5433120	-	-	-	-

Sama menetelmä tepsii mille tahansa taulukolle, jossa on n kokonaislukua. Näyttää siltä, että alkuperäisen taulukon tilavaativuus on $\mathcal{O}(n)$, kun taas uuden taulukon tilavaativuus on vain $\mathcal{O}(1)$. Onko johtopäätös oikea?

2. Analysoi seuraavien rekursiivisten algoritmien aika- ja tilavaativuudet, kun kullekin algoritmille annetaan syötteeksi kokonaisluku n. Ensin kannattaa ehkä kokeilla algoritmin toimintaa muutamalla syötteellä.

(a)
```
private static void rek1(int n) {
    System.out.println(n);
    if (n != 0) {
        rek1(n-1);
    }
}
```

(b)
```
private static void rek2(int n) {
    System.out.println(n);
    if (n != 0) {
        rek2(n-1);
        rek2(n-1);
    }
}
```

3. Yhteen suuntaan linkitetyssä listassa on vain *next*-osoittimet, ei *prev*-osoittimia. Tämä hankaloittaa delete-operaation toteuttamista. Toteutetaan poistot *laiskasti*: Listan solmuihin lisätään kenttä *deleted*, ja alkion poistaminen listasta toteutetaan yksinkertaisesti vaihtamalla sen solmun *deleted* arvoon true. Lisäksi pidetään kirjaa listassa olevien poistettujen solmujen lukumäärästä, ja vasta kun näitä kertyy yli puolet listan kokonaispituudesta, ne todella poistetaan listarakenteesta.

Esitä pseudokoodilla toteutukset operaatioille search, insert ja delete. Arvioi myös menetelmien tehokkuutta verrattuna kahteen suuntaan linkitettyyn listaan.

4. Näytä miten jono toteutetaan linkitettynä rakenteena siten että operaatiot ovat vakioaikaisia. Toteutuksella tarkoitetaan tässä samantyyppistä pseudokoodiesitystä kuin luentojen sivuilla 116-118 on tehty pinolle tai jollakin kielellä "mahdollisimman selkeästi" toteutettua versiota. Oikeiden kielten valmiiden kirjastojen käyttö ei ole sallittua.

Muista havainnollistaa jonon operaatioiden toimintaa kuvien tms. avulla.

5. **(a)** Näytä miten jonoa on mahdollista simuloida kahden pinon avulla: Toteuta operaatiot enqueue(Q, k), dequeue(Q) ja empty(Q). Oleta tässä jono ja pinot rajattoman kokoisiksi. Mikä on simuloidun jonon kunkin operaation aikavaativuus?

(b) Näytä miten pinoa on mahdollista simuloida kahden jonon avulla. Analysoi kyseisen pinon kunkin operaation aikavaativuus kuten kohdassa a.

(c) Voiko yhdellä pinolla simuloida yhtä jonoa? Entä voiko yhdellä jonolla simuloida yhtä pinoa? Tässä tehtävässä aikavaativuuksia ei tarvitse ottaa huomioon.