Tietorakenteet ja algoritmit syksy 2012

Laskuharjoitus 1

1. Tietojenkäsittelijä voi ajatella logaritmia usein seuraavasti: a-kantainen logaritmi log_a n kertoo, kuinka monta kertaa luku n pitää jakaa a:lla, ennen kuin päästään lukuun 1. Esimerkiksi log_2 8 on 3, koska 8/2/2/2 on 1. Jos lukuun 1 ei päästä tasajaoilla, logaritmin lopussa on desimaaliosa, joka kuvaa viimeistä vaillinaista jakoa. Esimerkiksi log_2 9 on noin 3,17, koska 9/2/2/2 on 1,125, eli kolmen jaon jälkeen onvielä hieman matkaa lukuun 1, mutta neljäs täysi jako veisi jo selvästi sen alapuolelle. Laske seuraavat logaritmit ilman laskinta:

(i) log_2 16
(ii) log_4 4
(iii) log_3 81
(iv) log_{11} 1
(v) log_{11} 121
(vi) log_{10} 10000

Huom: Tietojenkäsittelijöille 2-kantainen logaritmi on käytetyin ja tärkein. Myöhemmin kurssilla tullaan puhumaan "logaritmisesta"useassa yhteydessä ja näiden kohdalla tarkoitetaan aina 2-kantaista logaritmia ellei toisin mainita.

Mietittävää: Logaritmi on (yllättävän?) "nopea". Luvun 1000 logaritmi on noin 10, luvun 1 000 000 logaritmi on noin 20 ja luvun 1 000 000 000 logaritmi on noin 30.

Hyöty: Binäärihaku pystyy löytämään valmiiksi järjestetystä taulukosta alkion "logaritmisessa-ajassa". Tämä tarkoittaa sitä, että miljoona alkioisesta taulukosta binäärihaku löytää alkion puolittamalla taulukkoa vain 20 kertaa! (Huomattavasti nopeampaa kuin miljoonan alkion tarkastaminen yksitellen).

2. Tarkastellaan ei-negatiivisten kokonaislukujen binääriesityksiä (eli 2-kantaisia esityksiä). Esimerkiksi luvulle 50 saadaan seuraava binääriesitys: 110010. Esitystä 110010 voi ajatella lyhenteenä seuraavalle: $1*2^5+1*2^4+0*2^3+0*2^2+1*2^1+0*2^0 = 32+16+0+0+2+0 = 50$. Muistutus: $2^0 = 1$.

(a) Mikä on suurin ei-negatiivinen kokonaisluku, jonka pystyt esittämään tasan kuudella merkillä binääriesityksenä? Entä pienin?

(b) Mikä on suurin ei-negatiivinen kokonaisluku, jonka pystyt esittämään tasan kymmenellä merkillä binääriesityksenä? Entä pienin?

(c) Kuinka pitkä on luvun 200 binääriesitys.

(d) Kuinka ensimmäisen tehtävän logaritmien laskenta liittyy edellä olevien laskentaan? Osaatko muodostaa sen avulla yleisen kaavan minkätahansa ei-negatiivisen kokonaisluvun binääriesityksen pituudelle?

Huomio: Jokaisen parittoman kokonaisluvun binääriesitys loppuu merkkiin 1 ja parillisen merkkiin 0.

Mietittävää: Javan int-tyypin suurin arvo on 2,147,483,647 ja pienin -2,147,483,648. Yhteensä javan int-tyypillä on erilaisia arvoja siis 4,294,967,295 kappaletta. Kuinkakohan monta bittiä näiden esittämiseen tarvitaan...

3. Lue rekursiosta seuraavasta linkistä

http://www.ohjelmointiputka.net/oppaat/opas.php?tunnus=rekursio

Rekursion hallinta on erittäin tärkeää kurssin kannalta. Jos et osaa tehdä seuraavia tehtäviä omin päin, mene välittömästi TiRa-pajaan ja pyydä ohjaajalta avustusta rekursion oppimisessa!

Seuraavassa pari yksinkertaista rekursiota käyttävää lämmittelytehtävää. Allaoleviin tehtäviin ei normaalisti käytettäisi rekursiota, eli teemme nämä ainoastaan harjoituksen vuoksi.

Huom: jos et ymmärrä mitä tehtävässä kysytään tai saa tehtävää tehdyksi, tule kysymään TiRa-pajasta neuvoa!

a) Tee metodi, joka tulostaa parametrinaan saamansa määrän tähtiä, eli *-merkkejä. Sen lisäksi funktio kutsuu itseään rekursiivisesti jos parametrin arvo on positiivinen.

Metodin runko näyttää seuraavalta:

```
private static void tahtia(int lkm) {
   // tulosta lkm tahtea
   // kutsu funktiota rekursiivisesti tulostamaan lkm-1 tahteä
}
```

Käyttöesimerkki:

```
public static void main(String[] args) {
    tahtia(3);
}
```

Ruudulle pitäisi tulostua

```
***
**
*
```

Huom: yksittäinen metodin `tahtia`-komento-osan suoritus siis tulostaa ainoastaan yhden tähtirivin. Eli edellisessä esimerkissä metodi tulee kutsutuksi yhteensä 3 kertaa vaikka main tekeekin ainoastaan yhden kutsun.

b) Pienellä muutoksella ohjelma saadaan tulostamaan tähdet päinvastaisessa järjestyksessä, eli esim. kutsuttaessa `tahtia(3)`, tulostuu:

```
*
**
***
```

Kokeile mikä muutos saa tämän aikaan. Muutos liittyy rekursiivisen metodikutsun paikkaan, ja onnistuu yhtä koodiriviä siirtämällä. Et tarvitse esim. mitään apumuuttujia.

c) Yksittäinen metodi voi kutsua itseään rekursiivisesti useampaan kertaan. Kirjoita metodin runkoon kaksi rekursiivista kutsua. Laittamalla kutsut sopiviin paikkoihin, saadaan pääohjelmassa kutsumalla tahtia(2) aikaan kuvio:

```
*
**
*
```

Ja kutsumalla tahtia(3) kuvio:

```
*
**
*
***
*
**
*
```

Kokeile mikä muutos saa tämän aikaan.

d) Muokkaa edellistä ohjelmaa siten, että mukaan tulee staattinen muuttuja jonka avulla voidaan laskea kuinka monennesta rekursiivisesta kutsusta on kysymys. Tulosta jokaisen tähtirivin perään sen aiheuttaneen rekursiivisen kutsun järjestysnumero.

Seuraavassa hahmotelma:

```
static int kutsut;

public static void main(String[] args) {
    kutsut = 1;
    tahtia(3);
}

private static void tahtia(int lkm) {
    // otetaan talteen monesko kutsu itse ollaan ja kasvatetaan
    // kutsujen yhteenlaskettua lukumäärää
    int kutsunNumero = kutsut++;

    // ...
    // tulosta lkm tahtea ja tulosta kutsunNumero
    // ...
}
```

Edellisen kohdan kutsun tahtia(3) pitäisi näyttää suunilleen seuraavalta (huom. jos kutsut koodissasi rekursiivisesti myös tahtia(0) voi numerointi mennä hieman eri tavalla):

```
* 3
** 2
* 4
*** 1
* 6
** 5
* 7
```

e) Ymmärrätkö varmuudella sataprosenttisesti miten ohjelmasi etenee? Piirrä miten koodin suoritus etenee tai käy ohjelmasi suoritus askeltaen läpi debuggerilla. Piirtämiseen ja debuggerin käyttöön saat ohjausta tirapajassa.

f) Kun alat hallita rekursion toimintaperiaatteen, saat pienellä muokkauksella (lisäämällä kolmannen rekursiokutsun) saat ohjelmasi toimimaan esim. seuraavasi:

```
* 3
* 4
* 5
** 2
```

```
* 7
* 8
* 9
** 6
* 11
* 12
* 13
** 10
*** 1
```

Tee tämä muutos.

4.Luonnollisen luvun n kertoma $n!$ voidaan määritellä seuraavasti:

$$n! = \begin{cases} 1 & \text{jos } n = 0 \\ n \cdot (n-1)! & \text{jos } n > 0 \end{cases}$$

Esimerkiksi $3! = 3 \cdot 2! = 3 \cdot 2 \cdot 1! = 3 \cdot 2 \cdot 1 \cdot 0! = 3 \cdot 2 \cdot 1 \cdot 1 = 6$.

Toteuta Javalla rekursiivinen funktio, joka laskee luvun kertoman yllä olevalla menetelmällä. Seuraavassa on funktion käyttöesimerkki:

```
private static int kertoma(int luku) {
    // tee funktio tähän
}

public static void main(String[] args) {
    // tulostaa 362880
    System.out.println(kertoma(9));
}
```

5. Jos merkkijonon `TIRA` kääntää ympäri, tuloksena on merkkijono `ARIT`.

Toteuta Javalla rekursiivinen funktio, joka kääntää merkkijonon ympäri. Seuraavassa on funktion käyttöesimerkki:

```
private static String kaanna(String merkkijono) {
    // tee funktio tähän
}

public static void main(String[] args) {
    // tulostaa ARIT
    System.out.println(kaanna("TIRA"));
}
```

Vihje: Siirrä ensimmäinen merkki merkkijonon loppuun ja käännä (alkuperäinen) loppuosa rekursiivisesti.

6. Tarkastellaan seuraavaa algoritmia:

A. Valitaan jokin positiivinen kokonaisluku n.

B. Jos n on parillinen, jaetaan se kahdella. Jos taas n on pariton, kerrotaan se kolmella ja lisätään tulokseen yksi.

C. Jos n on yksi, lopetetaan. Muuten palataan kohtaan B.

Esimerkiksi jos valitaan aluksi $n = 6$, algoritmi etenee seuraavasti:

- $6/2 = 3$
- $3 \cdot 3 + 1 = 10$
- $10/2 = 5$
- $5 \cdot 3 + 1 = 16$
- $16/2 = 8$
- $8/2 = 4$
- $4/2 = 2$
- $2/2 = 1$

Tehtävänä on toteuttaa Java-ohjelma, joka simuloi algoritmia annetulla n:n arvolla. Ohjelman tulostuksen tulee olla seuraavanlainen:

```
Anna luku: 6
6 3 10 5 16 8 4 2 1
```

1. Toteuta ohjelma iteratiivisesti (eli ilman rekursiota, joko while- tai for-lauseita hyödyntäen).
2. Toteuta ohjelma rekursiivisesti.

Osaatko sanoa, pysähtyykö ohjelma kaikilla n:n arvoilla?