582359 Algoritmit ongelmanratkaisussa (kevät 2013)
Viikon 3 ratkaisuja

1. Tutustu dynaamiseen ohjelmointiin kurssisivun materiaalin avulla. Laske sen jälkeen seuraavat materiaalin esimerkkeihin liittyvät tulokset käsin dynaamisella ohjelmoinnilla:

   (a) Monellako tavalla luvun 10 voi esittää nopan silmälukujen summana? (luku 6.2)

   (b) Mikä on suurin mahdollinen lukujen summa reitissä seuraavassa ruudukossa? (luku 6.4)

6	3	9	4	8
4	8	2	3	3
3	6	4	6	9
1	7	1	2	5
9	3	9	9	1

   Ratkaisu:

   (a) Summat ovat seuraavat:

k	0	1	2	3	4	5	6	7	8	9	10
$N(k)$	1	1	2	4	8	16	32	63	125	248	492

   Siis summan 10 voi muodostaa 492 tavalla nopista.

   (b) Ruudukosta S tulee seuraava:

6	9	18	22	30
10	18	20	25	33
13	24	28	34	43
14	31	32	36	48
23	34	43	52	53

   Siis suurin mahdollinen summa reitillä on 53.

2. Lähtökohtana on positiivinen kokonaisluku n ja tehtävänä on esittää se positiivisten kokonaislukujen summana. Esimerkiksi jos $n = 4$, mahdollisia esitystapoja on 8:

   - $1 + 1 + 1 + 1$
   - $1 + 1 + 2$
   - $1 + 2 + 1$
   - $2 + 1 + 1$
   - $1 + 3$
   - $3 + 1$
   - $2 + 2$
   - 4

   Toteuta tehokas algoritmi, joka laskee luvun n esitystapojen määrän. Montako esitystapaa on, kun $n = 50$?

   Ratkaisu:

   Seuraava rekursiivinen funktio laskee ratkaisun:

$$S(k) = \begin{cases} 1 & \text{jos } k = 0 \\ \sum_{i=0}^{k-1} S(i) & \text{jos } k > 0 \end{cases}$$

   Tiedostossa `Summat.java` on ratkaisun toteutus dynaamisella ohjelmoinnilla. Funktion arvon voi myös laskea suoraan ilman rekursiota: $S(k) = 2^{k-1}$. Tämän ratkaisun voi ymmärtää ottamalla lähtökohdaksi luvun esityksen $1 + 1 + 1 + \cdots + 1$, poistamalla osan +-merkeistä ja laskemalla yhteen vierekkäiset luvut, joiden välissä ei ole +-merkkejä. Yhteensä +-merkkejä on $k - 1$, joten +-merkkien osajoukkoja on 2^{k-1}.

3. Tehtävä on sama kuin edellä, mutta mukaan hyväksytään vain summat, joissa luvut ovat järjestyksessä pienimmästä suurimpaan. Nyt jos $n = 4$, mahdollisia esitystapoja on 5:

   - $1 + 1 + 1 + 1$
   - $1 + 1 + 2$
   - $1 + 3$
   - $2 + 2$
   - 4

   Toteuta tähän tehtävään tehokas algoritmi ja laske vastaus, kun $n = 250$.

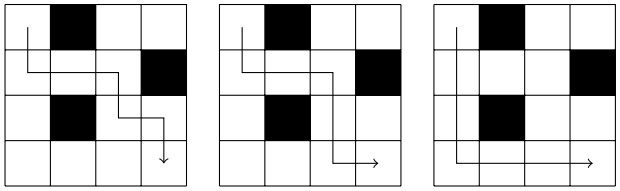
   Ratkaisu:

   Seuraava rekursiivinen funktio laskee ratkaisun:

$$S(k, r) = \begin{cases} 0 & \text{jos } k < 0 \text{ tai } r < 1 \\ 1 & \text{jos } k = 0 \\ S(k - r, r) + S(k, r - 1) & \text{jos } k > 0 \end{cases}$$

   Ideana on, että k on muodostettava summa ja r on poistettava luku. Vaihtoehtoja on kaksi: joko luku poistetaan summasta tai sitten poistettava luku pienentyy. Tämä varmistaa, että summa muodostuu järjestyksessä olevista luvuista. Tiedostossa `Summat2.java` on ratkaisun toteutus.

4. Tehtävänä on etsiä reitti ruudukon vasemmasta ylänurkasta oikeaan alanurkkaan. Jokaisessa vaiheessa reitillä saa liikkua askeleen oikealle tai alaspäin. Lisäksi reitti saa kulkea vain valkoisten ruutujen kautta. Esimerkiksi seuraavassa ruudukossa kelvollisia reittejä on 3:



   Tee tehokas algoritmi, joka laskee reittien yhteismäärän annetussa ruudukossa. Laske algoritmilla reittien määrä kurssisivulla olevassa 50×50 -ruudukossa `sokkelo.txt`.

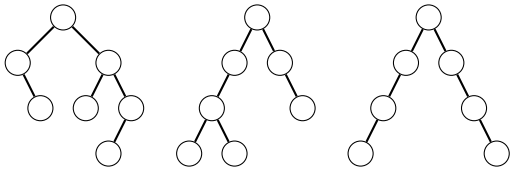
   Ratkaisu:

   Seuraava rekursiivinen funktio laskee ratkaisun:

$$R(y, x) = \begin{cases} 0 & \text{jos ruutu } (y, x) \text{ on musta} \\ 1 & \text{jos } y = 0 \text{ ja } x = 0 \\ R(y, x - 1) & \text{jos } y = 0 \\ R(y - 1, x) & \text{jos } x = 0 \\ R(y, x - 1) + R(y - 1, x) & \text{muuten} \end{cases}$$

   Tiedostossa `Sokkelo.java` on ratkaisun toteutus.

5. Seuraavissa binääripuissa on 7 solmua:



   Yhteensä erilaisia 7-solmuisia binääripuita on olemassa 429 kpl.

   Tee tehokas algoritmi, joka laskee n-solmuisten binääripuiden määrän. Laske algoritmilla, montako 30-solmuista binääripuuta on olemassa.

Ratkaisu:

Seuraava rekursiivinen funktio laskee ratkaisun:

$$B(k) = \begin{cases} 0 & \text{jos } k = 0 \\ \sum_{i=0}^{k-1} B(i)B(k-1-i) & \text{jos } k > 0 \end{cases}$$

Ideana on käydä läpi kaikki vaihtoehdot, montako solmua on vasemmassa ja oikeassa alipuussa. Joka tilanteessa tällaisten puiden yhteismäärä saadaan kertomalla keskenään vasemman ja oikean alipuiden määrät. Tiedostossa `Binaaripuut.java` on ratkaisun toteutus.

6. Tehtävänä on muodostaa annettu rahamäärä mahdollisimman pienellä määrällä kolikoita. Oletuksena on, että minkä tahansa rahamäärän voi muodostaa kolikoilla.

   Tehtävään on olemassa seuraava ahne algoritmi: käy kolikot läpi suurimmasta pienimpään ja ota jokaista mukaan mahdollisimman monta. Algoritmi toimii oikein, jos se muodostaa minkä tahansa rahamäärän pienimmällä määrällä kolikoita.

   (a) Osoita, että ahne algoritmi toimii, kun käytössä ovat eurokolikot (1 ja 2 euron kolikot sekä 1, 2, 5, 10, 20 ja 50 sentin kolikot).

   (b) Etsi mahdollisimman yksinkertainen esimerkki kolikkojoukosta, jolla ahne algoritmi ei toimi.

   (c) Tee algoritmi, joka ilmoittaa annetusta kolikkojoukosta, toimiiko ahne algoritmi.

Ratkaisu:

(a) Kohdassa (c) esitettävä algoritmi osoittaa tämän.

(b) Tarkastellaan kolikkojoukkoa $\{1, 3, 4\}$. Kun rahamäärä on 6, ahne algoritmi muodostaa summan $1 + 1 + 4 = 6$. Kuitenkin paras ratkaisu olisi $3 + 3 = 6$.

(c) Ideana on käydä läpi rahamääriä järjestyksessä ja verrata ahneen algoritmin ja dynaamisen ohjelmoinnin algoritmin toimintaa. Jos ahne algoritmi antaa jollakin rahamäärällä liian suuren tuloksen, se ei ole toimiva. Ongelmana on tietää, miten pitkälle rahamääriä täytyy tutkia.

Yksi hyvä yläraja on $2k$, jossa k on suurin kolikko. Jos ahne algoritmi muodostaa rahamäärät $1 \ldots 2k$ optimaalisesti, niin se muodostaa kaikki rahamäärät optimaalisesti. Tällöin rahamäärissä $k \ldots 2k$ kolikon k valinta tuottaa optimiratkaisun. Oletetaan sitten, että ahne algoritmi ei toimisi, kun rahamäärä on yli $2k$. Toisin sanoen pitäisi olla rahamäärä $r > 2k$, jossa kolikon k valinta estäisi optimiratkaisun muodostuksen. Tämä ei ole kuitenkaan mahdollista, koska rahamäärän r muodostuksessa ennen pitkää jäljellä oleva rahamäärä on välillä $k \ldots 2k$ ja tällöin kolikko k kelpaa.

Tiedostossa `Kolikot.java` on yllä kuvatun algoritmin toteutus.