582359 Algoritmit ongelmanratkaisussa (kevät 2013)
Viikon 1 ratkaisuja

1. Laske seuraavat bittioperaatiot ensin käsin. Varmista sitten ohjelmalla, että tulokset ovat oikein.

   (a) `123 & 321` (and-operaatio)

   (b) `123 | 321` (or-operaatio)

   (c) `123 ^ 321` (xor-operaatio)

   (d) `23 << 3` (bittisiirto vasemmalle)

   (e) `157 >> 4` (bittisiirto oikealle)

   (f) `~123` (not-operaatio)

   Ratkaisu:

   (a)
   ```
      123  001111011
   &  321  101000001
   ---------------------
       65  001000001
   ```

   (b)
   ```
      123  001111011
   |  321  101000001
   ---------------------
      379  101111011
   ```

   (c)
   ```
      123  001111011
   ^  321  101000001
   ---------------------
      314  100111010
   ```

   (d)
   ```
        23      10111
   << 3
   ---------------------
      184  10111000
   ```

   (e)
   ```
      157  10011101
   >>   4
   ---------------------
        9      1001
   ```

   (f)
   ```
   ~  123  001111011
   ---------------------
     -124  110000100
   ```

   Tässä on tulkintana, että alkubitti 1 tarkoittaa negatiivista lukua kahden komplementtina.

2. Perustele seuraavissa kohdissa, miksi vasemmalla ja oikealla olevat koodit toimivat samalla tavalla. Mitä hyviä ja huonoja puolia bittioperaatioiden käytössä on kussakin tilanteessa?

   (a) Parillisuuden tarkastus:

   ```
   if (x % 2 == 0) {
   ```
   ```
   if (x & 1 == 0) {
   ```

   (b) Muuttujan nollaus:

   ```
   x = 0;
   ```
   ```
   x ^= x;
   ```

   (c) Kerto- ja jakolaskut:

   ```
   x = (int)Math.pow(2, 20);
   a = 256 * b;
   c = d / 2;
   e = 24 * f;
   ```
   ```
   x = 1 << 20;
   a = b << 8;
   c = d >> 1;
   e = (f << 4) + (f << 3);
   ```

   (d) Muuttujien a ja b arvojen vaihtaminen:

   ```
   apu = a;
   a = b;
   b = apu;
   ```
   ```
   a = a ^ b;
   b = a ^ b;
   a = a ^ b;
   ```

Ratkaisu:

(a) Merkintä `x & 1` erottaa luvun viimeisen bitin, joka on 0 tai 1 luvun parillisuuden mukaan.

Hyvänä puolena bittioperaatio on prosessorille helpompi suorittaa kuin jakolasku. Huonona puolena tekniikkaa voi käyttää vain, jos jakaja on muotoa 2^k.

(b) `x ^= x` tarkoittaa `x = x ^ x`, ja `x ^ x` on aina 0, koska kaikki bitit ovat samoja.

Hyvänä puolena prosessorista riippuen xor-operaatio voi olla helpompi suorittaa kuin arvon 0 sijoitus. Huonona puolena xor-operaatiota voi käyttää vain nollauksessa.

(c) Bittien siirtäminen askeleen vasemmalle kertoo luvun 2:lla ja askeleen oikealle jakaa luvun 2:lla.

Hyvänä puolena bittien siirtäminen on prosessorille helpompaa kuin yleinen kerto- tai jakolasku. Huonona puolena vain kertominen ja jakaminen 2^k:lla on mahdollista.

(d) Tarkastellaan xor-menetelmän toimintaa, kun `a = 123` ja `b = 321`:

```
    a = 123  001111011
^   b = 321  101000001
-----------------------
    a = 314  100111010

    a = 314  100111010
^   b = 321  101000001
-----------------------
    b = 123  001111011

    a = 314  100111010
^   b = 123  001111011
-----------------------
    a = 321  101000001
```

Ideana on, että vaiheen 1 jälkeen `a`:ssa olevasta xor-arvosta saa sekä `a`:n että `b`:n alkuperäisen arvon xor-operaatiolla. Vastaavan idean voisi toteuttaa myös yhteenlaskulla:

```
a = a + b;
b = a - b;
a = a - b;
```

3. Lukujen $1 \ldots n$ osajoukko voidaan esittää yhtenä kokonaislukuna *bittivektorina*. Jos luku k kuuluu osajoukkoon, niin bittivektorin k. bitti lopusta alkaen on 1, ja muuten bitti on 0. Esimerkiksi osajoukkoa $\{1, 3, 4, 8\}$ vastaa bittivektori 10001101 eli kokonaisluku 141.

   Tiedostossa `Osajoukko.java` on osajoukon toteutus boolean-taulukkona. Muuta ohjelmaa käyttämään bittivektoria, jolloin osajoukon tallennukseen riittää yksi int-muuttuja.

   Ratkaisu:
   Muutettu toteutus on tiedostossa `Osajoukko2.java`.

4. Syötteenä annetaan positiivinen kokonaisluku n sekä $n-1$ eri kokonaislukua väliltä $1 \ldots n$. Tehtävänä on etsiä puuttuva luku. Esimerkiksi jos $n = 5$ ja luvut ovat 3, 1, 5 ja 4, niin puuttuva luku on 2.

   Esitä tehtävään ajassa $O(n)$ ja tilassa $O(1)$ toimiva ratkaisu, joka hyödyntää xor-operaatiota.

   Ratkaisu:
   Alustava ratkaisu on laskea ensin lukujen $1 \ldots n$ summa ja vähentää siitä kaikki annetut luvut, jolloin tuloksena on puuttuva luku. Esimerkiksi $1+2+3+4+5 = 25$ ja $25-3-1-5-4 = 2$. Tässä on pienenä heikkoutena, että lukujen summa on selvästi annettuja lukuja suurempi.

   Vastaavan ratkaisun voi toteuttaa xor-operaatiolla tehtävän 2(d) tyylisesti. Nyt riittää käyttää joka vaiheessa xor-operaatiota. Esimerkiksi 1 xor 2 xor 3 xor 4 xor 5 = 1 ja 1 xor 3 xor 1 xor 5 xor 4 = 2.

5. Funktio $Y(n)$ vastaa lukujen $1 \ldots n$ ykkösbittien yhteismäärää. Esimerkiksi $Y(5) = 7$, koska lukujen $1 \ldots 5$ bittiesitykset ovat 1, 10, 11, 100 ja 101 ja niissä on yhteensä 7 ykkösbittiä.

   Tee ohjelma, joka laskee tehokkaasti arvon $Y(12345678987654321)$.

Ratkaisu:

Tehokkaan ratkaisun keksimistä auttaa tarkastella lukujen bittiesityksiä kokonaisuutena. Seuraavassa taulukossa ovat lukujen $0 \ldots 20$ bittiesitykset:

```
 0       0
 1       1
 2      10
 3      11
 4     100
 5     101
 6     110
 7     111
 8    1000
 9    1001
10    1010
11    1011
12    1100
13    1101
14    1110
15    1111
16   10000
17   10001
18   10010
19   10011
20   10100
```

Huomataan, että jokaisessa sarakkeessa biteistä muodostuu säännöllinen kuvio. Viimeisessä sarakkeessa kuvio on 0, 1, 0, 1, 0, 1, ..., toiseksi viimeisessä sarakkeessa 0, 0, 1, 1, 0, 0, 1, 1, ..., kolmanneksi viimeisessä sarakkeessa 0, 0, 0, 0, 1, 1, 1, 1, ... jne. Siis sarakkeessa k oikealta lukien bitti vaihtuu 2^{k-1} luvun välein. Tämän säännöllisyyden avulla on mahdollista laskea kaikki tietyssä sarakkeessa olevat bitit samalla kertaa. Tiedostossa `Ykkosbitit.java` on tähän perustuva tehtävän ratkaisu.

6. (a) Käytettävissä ovat luvut $\{1, 2, 4, 8, 16, 32, \ldots\}$ eli muotoa 2^k olevat luvut. Osoita, että mikä tahansa positiivinen kokonaisluku voidaan esittää näiden lukujen osajoukon summana.
   Esimerkiksi $185 = 1 + 8 + 16 + 32 + 128$.

   (b) Osoita sama, kun luvut ovat $\{1, -2, 4, -8, 16, -32, \ldots\}$ eli muotoa $2^k(-1)^k$.
   Esimerkiksi $185 = 1 - 8 + 64 - 128 + 256$.

Ratkaisu 1:

(a) Muotoa 2^k olevan luvun bittiesitys on $1\underbrace{0\ldots0}_{k}$, minkä ansiosta minkä tahansa positiivisen kokonaisluvun saa tuotettua laskemalla yhteen tarvittavat bitit tuottavat luvut.
Esimerkiksi luvun 185 bittiesitys on `10111001`, joten sen saa summana bittiesityksistä $1 + 1000 + 10000 + 100000 + 10000000$ eli luvuista $1 + 8 + 16 + 32 + 128$.

(b) Tilanne on samantapainen, mutta negatiiviset luvut hankaloittavat asiaa. Yksi tekniikka on ottaa lähtökohdaksi luvun bittiesitys ja käydä bitit läpi oikealta vasemmalle. Aina ykkösbitin kohdalla vastaava luku 2^k tai -2^k poistetaan luvusta. Tämä jatkuu, kunnes luku on 0.
Esimerkiksi luvun 185 käsittely on seuraava:

luku	bittiesitys	poisto
185	1011100<u>1</u>	1
184	1011<u>1</u>000	-8
192	11<u>1</u>000000	64
128	<u>1</u>0000000	-128
256	<u>1</u>00000000	256
0	0	–

Ratkaisu 2:

Seuraavassa on toinen matemaattisempi todistus tehtävään. Ideana on olettaa, että jokin luku on pienin luku, jolla väite pitää paikkansa, ja näyttää, että tästä seuraa ristiriita.

(a) Tehdään vastaoletus, että luku p on pienin positiivinen kokonaisluku, jota ei voi muodostaa lukujen 2^k summana. Käydään läpi mahdolliset tapaukset, mikä p voi olla:

i. p on 1. Tämä on ristiriita, koska $p = 2^0$ eli sen voi esittää lukujen 2^k summana.

ii. p on parillinen. Nyt luvun $p/2$ voi esittää lukujen 2^k summana, koska p on pienin luku, jota ei voi esittää. Kuitenkin $p/2$:n esityksessä jokaisen luvun voi kertoa 2:lla, jolloin tuloksena on p:n esitys lukujen 2^k summana. Tämä aiheuttaa ristiriidan.

iii. p on pariton. Nyt luvun $(p-1)/2$ voi esittää lukujen 2^k summana. Kuitenkin tästä esityksestä saa luvun p esityksen kertomalla 2:lla ja lisäämällä 1:n, jota ei voi olla ennestään esityksessä. Tämä aiheuttaa ristiriidan.

Yllä olevan perusteella ei ole olemassa pienintä positiivista kokonaislukua p, jota ei voi esittää lukujen 2^k summana, joten kaikki positiiviset kokonaisluvut voi esittää lukujen 2^k summana.

(b) Todistetaan yleisemmin, että kaikki kokonaisluvut voi esittää lukujen 2^k summana. Tehdään vastaoletus, että p on itseisarvoltaan pienin kokonaisluku, jota ei voi muodostaa lukujen $(-2)^k$ summana. Tapaukset ovat:

i. p on -1, 0 tai 1. Tämä on ristiriita koska $-1 = 2^0 - 2^1$, 0 on tyhjä summa ja $1 = 2^0$.

ii. p on parillinen. Nyt luvun $p/(-2)$ voi esittää lukujen $(-2)^k$ summana. Kuitenkin $p/(-2)$:n esityksessä jokaisen luvun voi kertoa -2:lla, jolloin tuloksena on p:n esitys lukujen $(-2)^k$ summana. Tämä aiheuttaa ristiriidan.

iii. p on pariton. Nyt luvun $(p-1)/(-2)$ voi esittää lukujen $(-2)^k$ summana. Kuitenkin tästä esityksestä saa luvun p esityksen kertomalla sen -2:lla ja lisäämällä 1:n, jota ei voi olla ennestään esityksessä. Tämä aiheuttaa ristiriidan.

Yllä olevan perusteella minkä tahansa kokonaisluvun voi esittää lukujen $(-2)^k$ summana. Erityisesti minkä tahansa positiivisen kokonaisluvun voi esittää tällaisena summana.