582359 Algoritmit ongelmanratkaisussa (kevät 2013)
Viikon 5 ratkaisuja

1. Toteuta yksiulotteinen summataulukko ja $\sqrt{n}$-rakenne. Tee testi, jossa taulukossa on 1000000 lukua ja siitä lasketaan 50000 summaa satunnaisilta väleiltä. Paljonko menee aikaa, kun summat lasketaan suoraan taulukosta for-silmukalla, summataulukosta ja $\sqrt{n}$-rakenteesta?

   Ratkaisu:

   Tiedosto `Kyselytesti.java` sisältää testikoodin. TKTL:n työasemalla tulokset olivat:

toteutus	aika
for-silmukka	2116 ms
summataulukko	7 ms
$\sqrt{n}$ -rakenne	224 ms

2. Toteuta kaksiulotteinen summataulukko ja testaa sen toimivuus.

   Hahmottele, miten kolmiulotteisen summataulukon avulla voisi laskea kolmiulotteisen taulukon minkä tahansa kolmiulotteisen alitaulukon lukujen summan ajassa $O(1)$.

   Ratkaisu:

   Kaksiulotteisen summataulukon toteutus testeineen on tiedostossa `Summa2D.java`.

   Kolmiulotteisessa tapauksessa idea on sama, mutta kaavasta tulee mutkikkaampi. Tutkitaan tilannetta, jossa laskettavana on summa kohdasta $T[z1][y1][x1]$ kohtaan $T[z2][y2][x2]$. Lähtökohtana on koko summa loppunurkkaan asti:

   $$S[z2][y2][x2]$$

   Tästä poistetaan seuraavat alitaulukoiden summat:

   $$S[z1-1][y2][x2] + S[z2][y1-1][x2] + S[z2][y2][x1-1]$$

   Tähän lisätään seuraavat alitaulukoiden summat:

   $$S[z1-1][y1-1][x2] + S[z1-1][y2][x1-1] + S[z2][y1-1][x1-1]$$

   Tästä poistetaan vielä seuraava summa:

   $$S[z1-1][y1-1][x1-1]$$

   Yhteensä taulukosta S haetaan joka kyselyllä korkeintaan 8 summaa, joten kyselyt ovat vakioaikaisia. Yleisemmin n-ulotteisessa taulukossa kyselyn saa tehtyä hakemalla 2^n summaa taulukosta.

3. Lähtökohtana on seuraava taulukko:

9	1	4	9	8	5	5	6

   (a) Piirrä segmenttipuu, jonka avulla voi laskea taulukon välien summia.

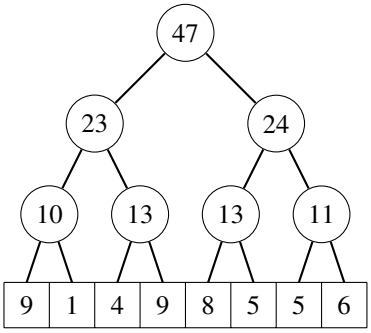
   (b) Näytä, miten segmenttipuusta saadaan laskettua summa luvusta 4 lukuun 8 ja luvusta 1 lukuun 6.

   (c) Näytä segmenttipuun muutokset, kun luvun 4 tilalle muutetaan 5 ja luvun 6 tilalle muutetaan 3.

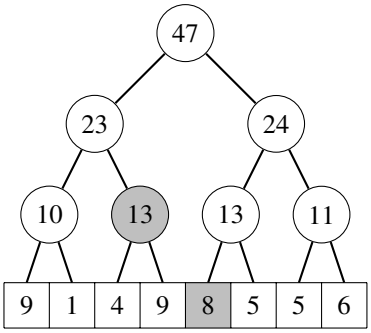
   (d) Toista kohdat (a)–(c) niin, että summien sijasta kyselyt antavat pienimmän luvun välillä.

Ratkaisu:

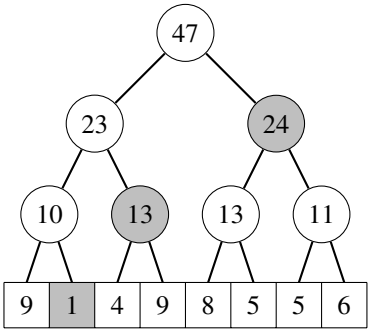
(a) Tässä on taulukkoa vastaava segmenttipuu summien laskemiseen:



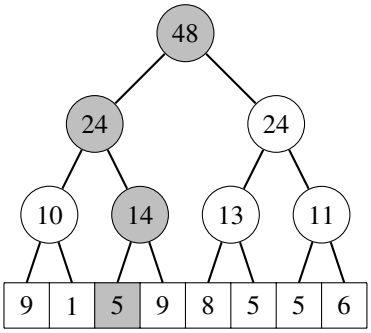
(b) Summa luvusta 4 lukuun 8 on $13 + 8 = 21$:



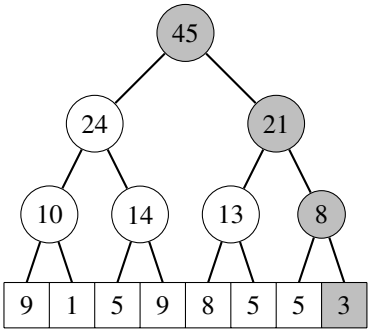
Summa luvusta 1 lukuun 6 on $1 + 13 + 24 = 38$:



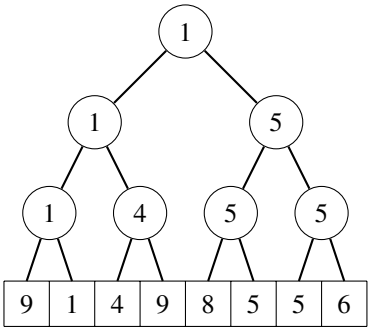
(c) Kun luvusta 4 tulee 5, segmenttipuu muutuu näin:



Kun luvusta 6 tulee 3, segmenttipuu muuttuu näin:



(d) Seuraavan segmenttipuun avulla voi selvittää välin pienimmän luvun:



Kyselyt ja muutokset tapahtuvat vastaavasti kuin kohdissa (a)–(c). Ainoa ero on, että joka vaiheessa kahden luvun summan sijasta valitaan luvuista pienempi.

4. Tutustu binääri-indeksipuuhun. Mitä hyviä ja huonoja puolia siinä on verrattuna segmenttipuuhun?

Ratkaisu:

Hyviä puolia:

- muistinkäyttö on puolet pienempi kuin segmenttipuussa
- toteutus vaatii hyvin vähän koodia
- käytännössä nopeampi kuin segmenttipuu

Huonoja puolia:

- soveltuu lähinnä summien laskemiseen
- perustuu bittikikkailuun, jota voi olla vaikea ymmärtää

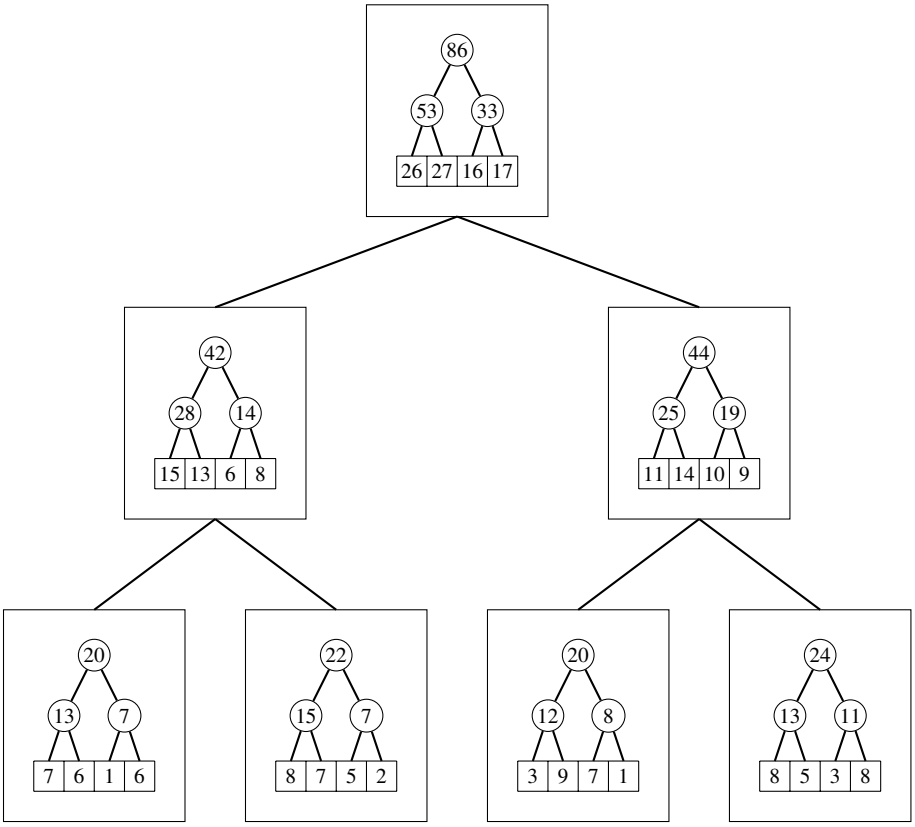
5. Kaksiulotteinen segmenttipuu mahdollistaa kyselyt kaksiulotteisen taulukon minkä tahansa suorakulmion alueelta. Selitä, miten kaksiulotteisen segmenttipuun voi toteuttaa. Mitkä ovat aluekyselyn ja päivityksen aikavaativuudet? Piirrä seuraavaa taulukkoa vastaava segmenttipuu summien laskemiseen:

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

Ratkaisu:

Ideana on muodostaa segmenttipuu, jonka jokainen alkio on segmenttipuu. Suuresta puusta valitaan käsiteltävät taulukon rivit. Vastaavasti pienistä puista valitaan käsiteltävät taulukon sarakkeet. Tällä menetelmällä sekä kyselyn että muutoksen aikavaativuus on $O(\log^2 n)$.

Esimerkin tapauksessa puusta tulee seuraava:



6. Lähtökohtana on lukujen $1 \ldots n$ permutaatio ja tehtävänä on laskea, montako nousevaa alijonoa permutaatiossa on yhteensä. Esimerkiksi permutaatiossa $(1, 4, 2, 3)$ nousevia alijonoja on yhteensä 9: (1), $(1, 2)$, $(1, 2, 3)$, $(1, 3)$, $(1, 4)$, (2), $(2, 3)$, (3) ja (4). Tee tehokas algoritmi ongelmaan ja laske sen avulla vastaus tiedostossa `permutaatio.txt` olevaan tapaukseen, jossa $n = 1000000$.

   Huom.: Vastaus on suuri luku eikä mahdu 64-bittiseen muuttujaan. Jos toteutat ratkaisun Javalla, niin `BigInteger` on hyvä valinta.

   Ratkaisu:

   Ideana on käydä läpi permutaation luvut pienimmästä suurimpaan. Jokaisessa vaiheessa käsiteltävään lukuun päättyvien alijonojen määrä saadaan laskemalla yhteen kaikki alijonot, jotka päättyvät aiemmin käsiteltyyn lukuun ja nykyisen luvun vasemmalle puolelle permutaatiossa. Lisäksi mukaan lisätään alijono, jossa on vain käsiteltävä luku. Tehokas toteutus onnistuu segmenttipuun tai binääri-indeksipuun avulla. Tiedostossa `Permutaatio.java` oleva ratkaisu perustuu segmenttipuuhun.