Luku 21

Neliöjuurirakenteet

Neliöjuurirakenteissa on ideana jakaa n-alkioinen taulukko noin $\sqrt{n}$ väliin niin, että jokaisella välillä on noin $\sqrt{n}$ alkiota. Tämän ansiosta sekä välien läpikäynti että tietyn välin alkioiden läpikäynti onnistuvat ajassa $O(\sqrt{n})$. Neliöjuurirakenne ja segmenttipuu ovat usein vaihtoehtoisia tekniikoita.

21.1 Summakysely

Aloitamme neliöjuurirakenteisiin tutustuminen tutusta ongelmasta, jossa toteutettavana on seuraavat operaatiot:

- muuta kohdassa k olevaa lukua
- laske lukujen summa välillä $[a,b]$

Ideana on jakaa taulukko $\sqrt{n}$-kokoisiin väleihin niin, että jokaiseen väliin tallennetaan lukujen summa välillä. Tilanne näyttää esimerkiksi tältä:

21				17				20				13			
5	8	6	3	2	7	2	6	7	1	7	5	6	2	3	2

Luvun muuttamisen aikavaativuus on $O(1)$, koska riittää muuttaa lukua alkuperäisessä taulukossa sekä vastaavan välin summaa:

21				15				20				13			
5	8	6	3	2	5	2	6	7	1	7	5	6	2	3	2

Välin summan laskeminen taas tapahtuu muodostamalla summa reunoissa olevista yksittäisistä luvuista sekä keskellä olevista $\sqrt{n}$-väleistä:

21				15				20				13			
5	8	6	3	2	5	2	6	7	1	7	5	6	2	3	2

Summan laskemisen aikavaativuus on $O(\sqrt{n})$, koska summa kootaan laskemalla yhteen $O(\sqrt{n})$ summaa $\sqrt{n}$-väleistä, $O(\sqrt{n})$ lukua vasemmasta reunasta sekä $O(\sqrt{n})$ lukua oikeasta reunasta.

Aikavaativuus $O(\sqrt{n})$ on aikavaativuuksien $O(\log n)$ ja $O(n)$ väliin asettuva aikavaativuus. Usein $\sqrt{n}$-algoritmit ovat rakenteeltaan yksinkertaisia, minkä ansiosta algoritmien vakiokertoimet ovat pieniä.

Segmenttipuuna toteutettuna kummankin operaation aikavaativuus olisi $O(\log n)$. Segmenttipuu on siis hitaampi, kun muutetaan taulukon lukua, mutta nopeampi, kun lasketaan välin lukujen summa.

21.2 Arvoväli

Usein $\sqrt{n}$-välillä on jokin tietorakenne yksittäisen arvon sijasta. Toteutetaan nyt neliöjuurirakenne, jonka operaatiot ovat:

- muuta kohdassa k olevaa lukua
- laske, montako lukua on arvovälillä $[l,r]$

Esimerkiksi taulukossa

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

seuraavat 8 lukua ovat arvovälillä $[5,7]$:

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

Jaetaan taas taulukko $\sqrt{n}$ väliin. Nyt jokainen väli sisältää välin luvut järjestyksessä pienimmästä suurimpaan. Esimerkiksi näin:

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

Nyt kunkin $\sqrt{n}$-välin sisällä kaikki tietylle arvovälille kuuluvat luvut ovat peräkkäin. Esimerkiksi välissä $[5,7]$ tilanteena on:

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

Kun taulukon luku muuttuu, vastaava muutos täytyy tehdä $\sqrt{n}$-välille. Tämä onnistuu ajassa $O(\sqrt{n})$, koska välillä on $O(\sqrt{n})$ alkiota ja riittää muuttaa yhtä alkiota ja siirtää se askel kerrallaan oikeaan kohtaan järjestetystä.

Arvovälin kyselyn voi toteuttaa käymällä läpi kaikki $\sqrt{n}$-välit ja laskemalla jokaisella välillä arvoväliin kuuluvien lukujen määrä binäärihaulla. Tämän aikavaati-

vuus on $O(\sqrt{n}\log\sqrt{n})$, koska $\sqrt{n}$-välien määrä on $O(\sqrt{n})$ ja jokainen binäärihaku välillä vie aikaa $O(\log\sqrt{n})$.

Toinen tapa ratkaista tehtävä on käyttää harvaa segmenttipuuta. Neliöjuurirakenne yleistyy kuitenkin myös kyselyyn, jossa laskettavana on arvovälin $[l,r]$ lukujen määrä taulukon välillä $[a,b]$. Tämän toteuttaminen tehokkaasti segmenttipuuta käyttäen olisi hankalampaa.

21.3 Mo'n algoritmi

Mo'n algoritmi on toisenlainen tapa hyödyntää neliöjuurta. Algoritmia voi käyttää tilanteissa, joissa taulukon sisältö ei muutu vaan taulukkoon tehdään vain kyselyitä. Mo'n algoritmi valitsee kyselyiden käsittelyjärjestyksen niin, että algoritmin kokonaisaikavaativuus on perustoteutusta tehokkaampi.

Tutustutaan Mo'n algoritmiin, kun kyselyinä on joukko taulukon välejä ja tehtävänä on selvittää jokaisesta välistä, montako eri alkiota siinä on.

Esimerkiksi taulukon välillä

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

on 4 eri alkiota: 1, 2, 6 ja 7.

Suoraviivainen tapa ratkaista tehtävä on käsitellä kyselyt yksi kerrallaan ja pitää yllä sopivaa tietorakennetta, joka laskee eri alkioiden määrän. Kyselyiden välillä alueen vasenta ja oikeaa reunaa siirretään askel kerrallaan ja samalla päivitetään tietorakenteen sisältöä.

Tehtävään sopiva tietorakenne on taulukko tai `map`-rakenne, joka pitää kirjaa kunkin luvun määrästä alueen sisällä. Esimerkiksi yllä olevassa tilanteessa tietorakenteen sisältönä olisi:

alkio	1	2	6	7
määrä	1	2	1	3

Nyt kun esimerkiksi uudeksi väliksi tulee

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

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

niin 2:n määrä vähenee yhdellä ja 5:n ja 6:n määrät kasvavat yhdellä:

alkio	1	2	5	6	7
määrä	1	1	1	2	3

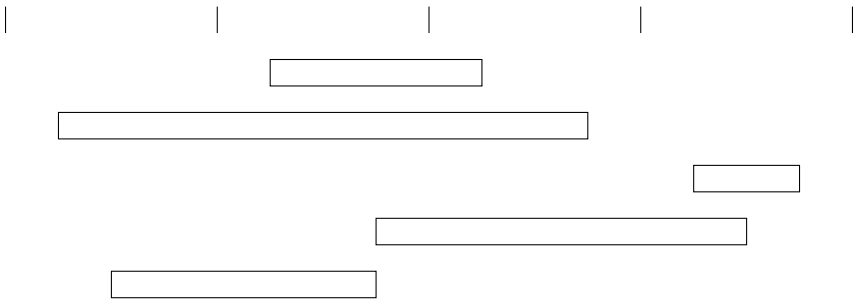
Suoraviivaisessa toteutuksessa n kyselyn käsittely n-alkioisessa taulukossa liikuttaa alueen vasenta ja oikeaa reunaa $O(n^2)$ askelta, koska jokainen vaihdos kyselystä toiseen liikuttaa vasenta ja oikeaa reunaa $O(n)$ askelta.

Algoritmin aikavaativuus syntyy tästä ottamalla huomioon tietorakenteen päivitykseen kuluva aika kullakin askeleella. Tässä tehtävässä päivitykseen voi kulua $O(1)$ tai $O(\log n)$ aikaa, jolloin aikavaativuudeksi tulee $O(n^2)$ tai $O(n^2 \log n)$.

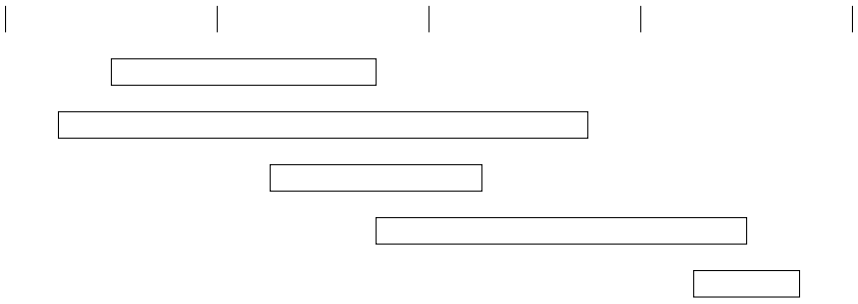
Mo'n algoritmi toimii muuten samalla tavalla, mutta algoritmin alussa kyselyt järjestetään uudestaan. Kyselyt voi tehdä missä tahansa järjestyksessä, koska taulukon sisältö on muuttumaton.

Ideana on järjestää kyselyt kahden kriteerin mukaan. Algoritmi järjestää kyselyt ensisijaisesti sen mukaan, mihin $\sqrt{n}$-väliin kyselyn vasen reuna kuuluu, ja toissijaisesti sen mukaan, mihin kohtaan kyselyn oikea reuna päättyy.

Esimerkiksi jos kyselyt ovat



niin Mo'n algoritmi käsittelee ne järjestyksessä.



Tarkastellaan nyt vasemman ja oikean reunan liikettä, kun vasen reuna on tietyn $\sqrt{n}$-välin sisällä. Kyselyn vaihtuessa vasen reuna liikkuu aina $O(\sqrt{n})$ askelta, koska välin sisällä on $\sqrt{n}$ alkiota. Oikea reuna taas liikkuu aina eteenpäin oikealle, joten se liikkuu kokonaisuutena $O(n)$ askelta $\sqrt{n}$-välin sisällä.

Lisäksi vasen reuna liikkuu algoritmin aikana kokonaisuutena $O(n)$ askelta kohdissa, joissa seuraava kysely on toisen $\sqrt{n}$-alueen sisällä.

Tämän ansiosta sekä vasen reuna että oikea reuna liikkuvat algoritmin aikana $O(n\sqrt{n})$ askelta, eli algoritmin aikavaativuudessa tekijä n muuttuu muotoon $\sqrt{n}$. Jos tietorakenteen päivitys vie aikaa $O(1)$ tai $O(\log n)$, vastaavat aikavaativuudet ovat $O(n\sqrt{n})$ ja $O(n\sqrt{n}\log n)$.