# Kisakoodarin käsikirja

Antti Laaksonen

4. joulukuuta 2016

# Sisältö

# Osa I

# Perusasiat

# Osa II

# Verkkoalgoritmit

# Osa III

# Lisäaiheita

# Luku 30

# Pyyhkäisyviiva

***Pyyhkäisyviiva*** on tason halki kulkeva viiva, jonka avulla voi ratkaista useita geometrisia tehtäviä. Ideana on esittää tehtävä joukkona tapahtumia, jotka vastaavat tason pisteitä. Kun pyyhkäisyviiva törmää pisteeseen, tapahtuma käsitellään ja tehtävän ratkaisu edistyy.

Seuraava tehtävä tarjoaa yksinkertaisen esimerkin tekniikasta:

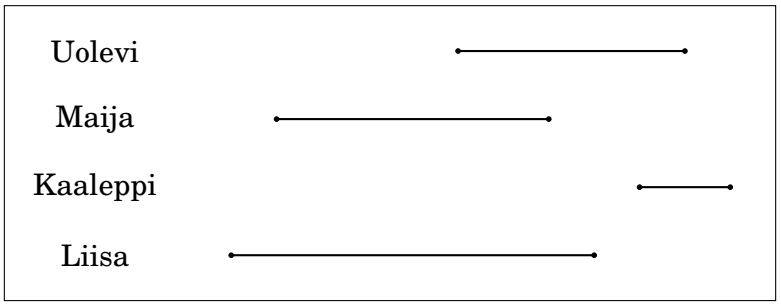
> *Tehtävä:* Yrityksessä on töissä $n$ henkilöä. Tiedät jokaisesta henkilöstä, milloin hän tuli töihin ja lähti töistä tiettynä päivänä. Mikä on suurin määrä henkilöitä, jotka olivat samaan aikaan töissä?

Tehtävän voi ratkaista mallintamalla tilanteen niin, että jokaista henkilöä vastaa kaksi tapahtumaa: tuloaika töihin ja lähtöaika töistä. Pyyhkäisyviiva käy läpi tapahtumat aikajärjestyksessä ja pitää kirjaa, montako henkilöä on töissä milloinkin.

Esimerkiksi tilannetta

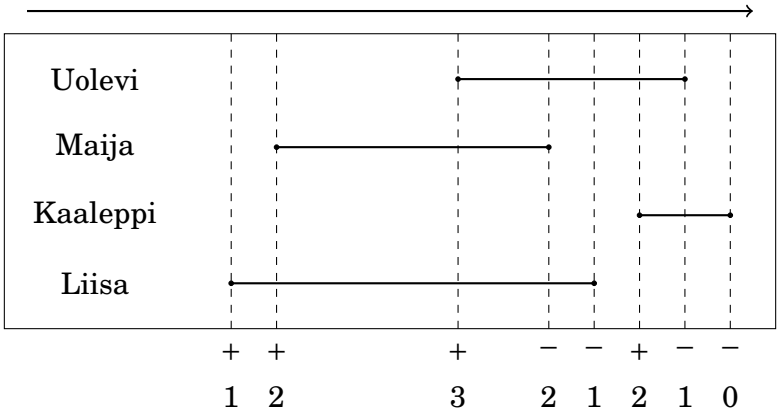
| henkilö | tuloaika | lähtöaika |
|---|---|---|
| Uolevi | 10 | 15 |
| Maija | 6 | 12 |
| Kaaleppi | 14 | 16 |
| Liisa | 5 | 13 |

vastaavat seuraavat tapahtumat:



Pyyhkäisyviiva käy läpi tapahtumat vasemmalta oikealle ja pitää yllä laskuria. Aina kun henkilö tulee töihin, laskurin arvo kasvaa yhdellä, ja kun henkilö lähtee töistä, laskurin arvo vähenee yhdellä. Tehtävän ratkaisu on suurin laskuri arvo pyyhkäisyviivan kulun aikana.

Pyyhkäisyviiva kulkee seuraavasti tason halki:




Kuvan alareunan merkinnät + ja − tarkoittavat, että laskurin arvo kasvaa ja vähenee yhdellä. Niiden alapuolella on laskurin uusi arvo. Laskurin suurin arvo 3 on voimassa Uolevi tulohetken ja Maijan lähtöhetken välillä.

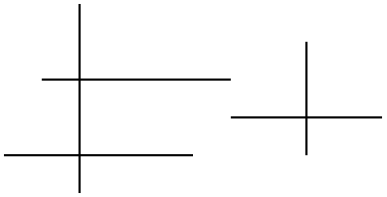
Ratkaisun aikavaativuus on $O(n \log n)$, koska tapahtumien järjestäminen vie aikaa $O(n \log n)$ ja pyyhkäisyviivan läpikäynti vie aikaa $O(n)$.

Tässä luvussa tutustumme kolmeen klassiseen tehtävään, jotka ratkeavat tehokkaasti pyyhkäisyviivan avulla. Ensimmäinen tehtävämme on laskea tasossa olevien janojen leikkauspisteiden määrä.

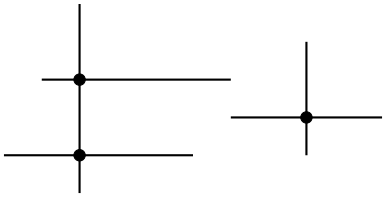
## 30.1 Janojen leikkauspisteet

*Tehtävä:* Annettuna on $n$ janaa, joista jokainen on vaaka- tai pystysuuntainen. Monessako pisteessä kaksi janaa leikkaa toisiaan?

Esimerkiksi tilanteessa



leikkauspisteitä on kolme:

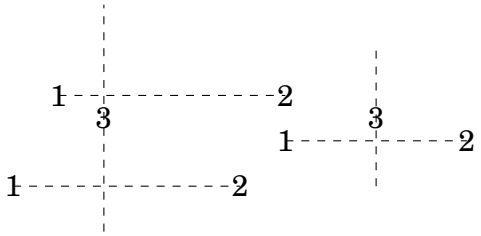


Tehtävä on helppoa ratkaista ajassa $O(n^2)$, koska riittää käydä läpi kaikki mahdolliset janaparit ja tarkistaa, moniko leikkaa toisiaan. Seuraavaksi ratkaisemme tehtävän ajassa $O(n \log n)$ pyyhkäisyviivan avulla.

Ideana on luoda janoista kolmenlaisia tapahtumia:

(1) vaakajana alkaa
(2) vaakajana päättyy
(3) pystyjana

Äskeistä esimerkkiä vastaava pistejoukko on seuraava:




Algoritmi käy läpi pisteet vasemmalta oikealle ja pitää yllä tietorakennetta y-koordinaateista, joissa on tällä hetkellä aktiivinen vaakajana. Tapahtuman 1 kohdalla vaakajanan y-koordinaatti lisätään joukkoon ja tapahtuman 2 kohdalla vaakajanan y-koordinaatti poistetaan joukosta.

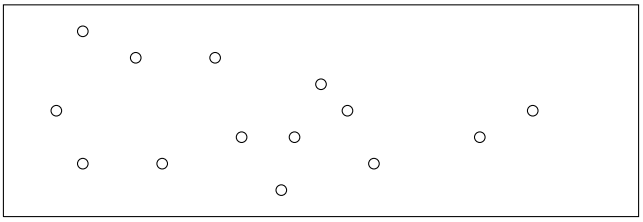
Algoritmi laskee janojen leikkauspisteet tapahtumien 3 kohdalla. Kun pystyjana kulkee y-koordinaattien $y_1 \ldots y_2$ välillä, algoritmi laskee tietorakenteesta, monessako vaakajanassa on y-koordinaatti välillä $y_1 \ldots y_2$ ja kasvattaa leikkauspisteiden määrää tällä arvolla.

Sopiva tietorakenne vaakajanojen y-koordinaattien tallentamiseen on binääri-indeksipuu tai segmenttipuu, johon on tarvittaessa yhdistetty indeksien pakkaus. Tällöin jokaisen pisteen käsittely vie aikaa $O(\log n)$, joten algoritmin kokonaisaikavaativuus on $O(n \log n)$.

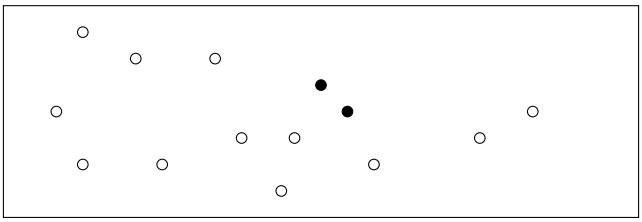
## 30.2 Lähin pistepari

*Tehtävä:* Annettuna on $n$ pistettä kaksiulotteisessa tasossa ja tehtäväsi on etsiä kaksi pistettä, jotka ovat mahdollisimman lähellä toisiaan.

Esimerkiksi tilanteessa



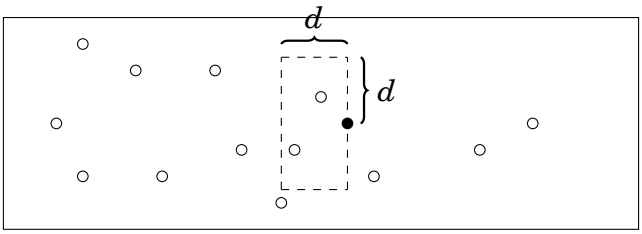
lähin pistepari on seuraava:



Tämäkin tehtävä ratkeaa $O(n \log n)$-ajassa pyyhkäisyviivan avulla. Algoritmi käy pisteet läpi vasemmalta oikealle ja pitää yllä arvoa $d$, joka on pienin kahden pisteen etäisyys. Kunkin pisteen kohdalla algoritmi etsii lähimmän toisen pisteen vasemmalta. Jos etäisyys tähän pisteeseen on alle $d$, tämä on uusi pienin kahden pisteen etäisyys ja algoritmi päivittää $d$:n arvon.

Jos käsiteltävä piste on $(x,y)$ ja jokin vasemmalla oleva piste on alle $d$:n etäisyydellä, sen x-koordinaatin tulee olla välillä $[x-d,x]$ ja y-koordinaatin tulee olla välillä $[y-d,y+d]$. Algoritmin riittää siis tarkistaa ainoastaan pisteet, jotka osuvat tälle välille, mikä tehostaa hakua merkittävästi.

Esimerkiksi seuraavassa kuvassa katkoviiva-alue sisältää pisteet, jotka voivat olla alle $d$:n etäisyydellä tummennetusta pisteestä.



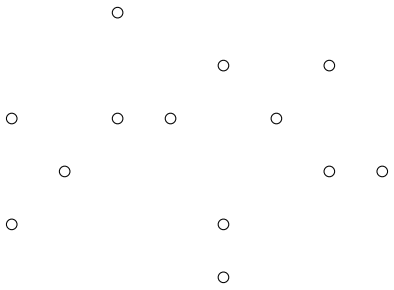

Algoritmin tehokkuus perustuu siihen, että $d$:n rajoittamalla alueella on aina vain $O(1)$ pistettä. Nämä pisteet pystyy käymään läpi $O(\log n)$-aikaisesti pitämällä algoritmin aikana yllä joukkoa pisteistä, joiden x-koordinaatti on välillä $[x-d,x]$ ja jotka on järjestetty y-koordinaatin mukaan.

Algoritmin aikavaativuus on $O(n \log n)$, koska se käy läpi $n$ pistettä ja etsii jokaiselle lähimmän edeltävän pisteen ajassa $O(\log n)$.

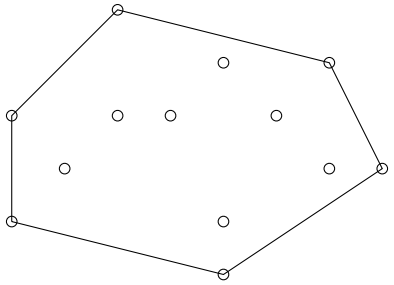
## 30.3 Konveksi peite

***Konveksi peite*** on pienin konveksi monikulmio, joka ympäröi kaikki pistejoukon pisteet. Konveksius tarkoittaa, että minkä tahansa kahden kärkipisteen välinen jana kulkee monikulmion sisällä. Hyvä mielikuva asiasta on, että pistejoukko ympäröidään tiukasti viritetyllä narulla.

Esimerkiksi pistejoukon



konveksi peite on seuraava:

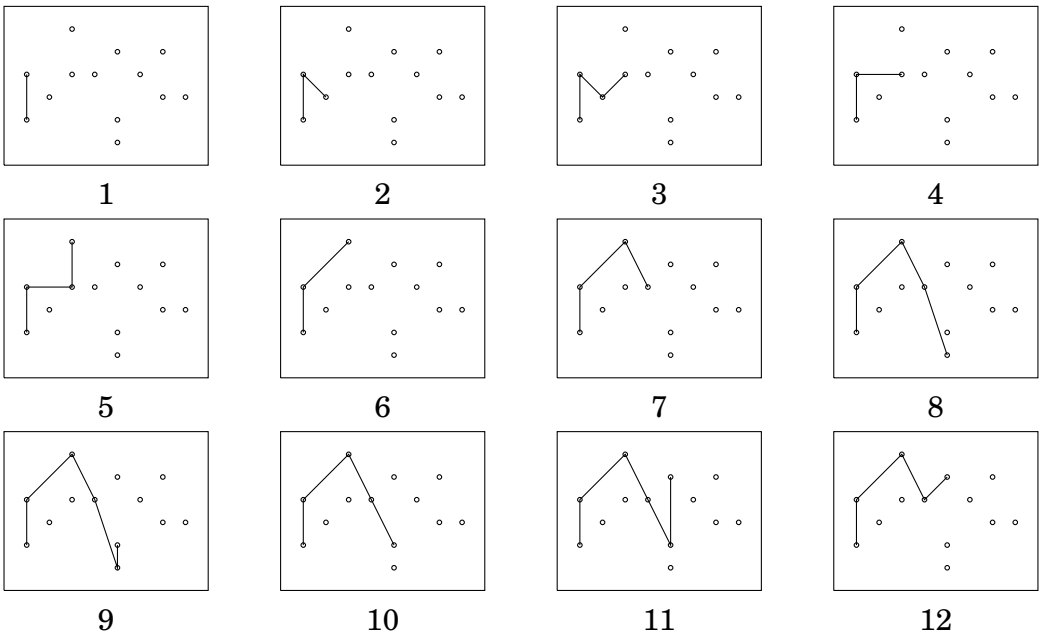


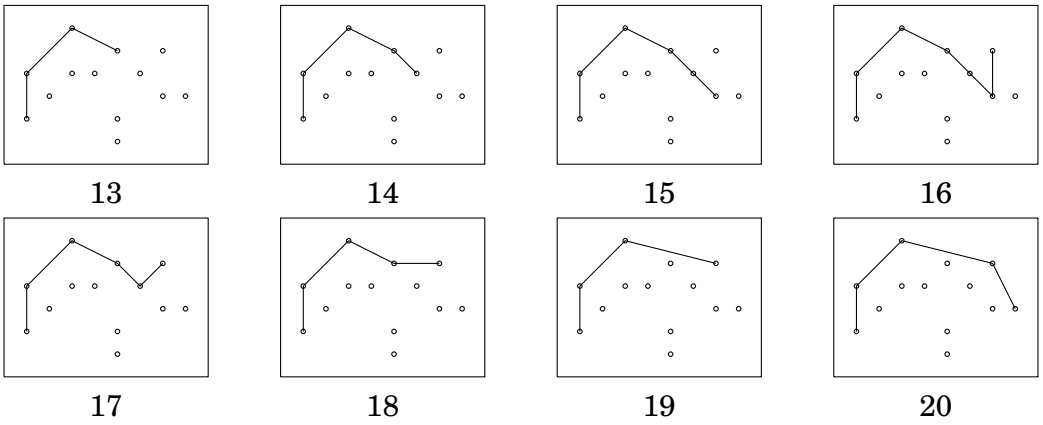
Tehokas ja helposti toteutettava menetelmä muodostaa konveksi peite on ***Andrew'n algoritmi***, jonka aikavaativuus on $O(n \log n)$.

Algoritmi muodostaa konveksin peitteen kahdessa osassa: ensin peitteen yläosan ja sitten peitteen alaosan. Kummankin osan muodostaminen tapahtuu samalla tavalla, ja keskitymme nyt yläosan muodostamiseen.

Algoritmi järjestää ensin pisteet ensisijaisesti x-koordinaatin ja toissijaisesti y-koordinaatin mukaan. Tämän jälkeen se käy pisteet läpi järjestyksessä ja liittää aina uuden pisteen osaksi peitettä. Kuitenkin aina kun kolme viimeistä pistettä peitteessä muodostavat vasemmalle kääntyvän osan, algoritmi poistaa näistä keskimmäisen pisteen.

Seuraava kuvasarja esittää Andrew'n algoritmin toimintaa:






Algoritmi tarkastaa ristitulon avulla, muodostavatko kolme pistettä vasemmalle kääntyvän osan. Algoritmin aikavaativuus on $O(n \log n)$, koska pisteiden järjestäminen vie aikaa $O(n \log n)$ ja sen jälkeen konveksin peitteen muodostaminen vie aikaa $O(n)$.

# Hakemisto