Kisakoodarin käsikirja

Antti Laaksonen

2. marraskuuta 2016

Sisältö

Alkusanat **v**

I Perusasiat **1**

1 Johdanto **3**

2 Aikavaativuus **15**

3 Järjestäminen **23**

4 Tietorakenteet **33**

5 Täydellinen haku **43**

6 Ahneet algoritmit **51**

7 Dynaaminen ohjelmointi **57**

8 Tasoitettu analyysi **69**

9 Välikyselyt **75**

10 Bittien käsittely **87**

II Verkkoalgoritmit **95**

11 Verkkojen perusteet **97**

12 Verkon läpikäynti **103**

13 Lyhimmät polut **109**

14 Puiden käsittely **117**

15 Virittävät puut **123**

16 Syklittömät verkot **129**

17 Vahvasti yhtenäisyys **135**

18 Puukyselyt **141**

19 Polut ja kierrokset **147**

20 Virtauslaskenta **153**

III Uusia haasteita 161

21 Lukuteoria **163**

22 Kombinatoriikka **169**

23 Matriisit **175**

24 Todennäköisyys **179**

25 Peliteoria **185**

26 Merkkijonot **191**

27 Neliöjuurialgoritmit **199**

28 Lisää segmenttipuusta **203**

29 Geometria **211**

30 Pyyhkäisyviiva **217**

Osa I

Perusasiat

Osa II

Verkkoalgoritmit

Luku 11

Verkkojen perusteet

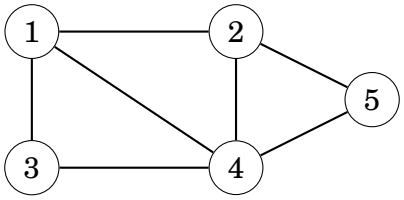
Monen ohjelmointitehtävän voi ratkaista tulkitsemalla tehtävän verkko-ongelmana ja käyttämällä sopivaa verkkoalgoritmia. Tyypillinen esimerkki verkosta on tieverkosto, jonka rakenne muistuttaa luonnostaan verkkoa. Joskus taas verkko kätkeytyy syvemmälle ongelmaan ja sitä voi olla vaikeaa huomata.

Tässä kirjan osassa tutustumme verkkojen käsittelyyn liittyviin tekniikoihin ja kisakoodauksessa keskeisiin verkkoalgoritmeihin. Aloitamme aiheeseen perehtymisen käymällä läpi verkkoihin liittyviä käsitteitä sekä erilaisia tapoja pitää verkkoa muistissa algoritmeissa.

11.1 Käsitteitä

Verkko (*graph*) muodostuu solmuista (*node* tai *vertex*) ja niiden välisistä kaaristä (*edge*). Merkitsemme tässä kirjassa yleensä verkon solmujen määrää muuttujalla n ja verkon kaarten määrää muuttujalla m. Lisäksi numeroimme verkon solmut kokonaisluvuin $1, 2, \ldots, n$.

Esimerkiksi seuraavassa verkossa on 5 solmua ja 7 kaarta:

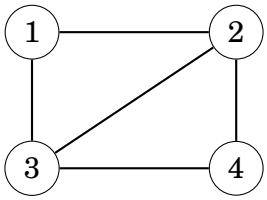


Polku (*path*) on solmusta a solmuun b johtava reitti, joka kulkee verkon kaaria pitkin. Polun pituus (*length*) on kaarten määrä polulla. Esimerkiksi yllä olevassa verkossa mahdollisia polkuja solmusta 1 solmuun 5 ovat:

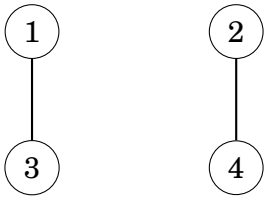
- $1 \to 2 \to 5$ (pituus 2)
- $1 \to 4 \to 5$ (pituus 2)
- $1 \to 2 \to 4 \to 5$ (pituus 3)
- $1 \to 3 \to 4 \to 5$ (pituus 3)
- $1 \to 3 \to 4 \to 2 \to 5$ (pituus 4)

Yhtenäisyys

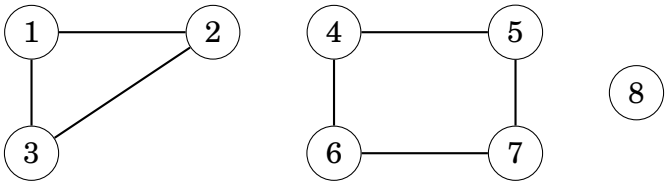
Verkko on yhtenäinen (*connected*), jos siinä on polku mistä tahansa solmusta mihin tahansa solmuun. Esimerkiksi seuraava verkko on yhtenäinen:



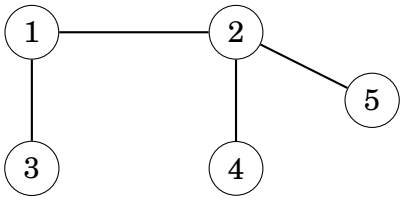
Seuraava verkko taas ei ole yhtenäinen, koska esimerkiksi solmusta 1 ei ole polkua solmuun 2.



Verkon yhtenäiset osat muodostavat sen komponentit (*components*). Esimerkiksi seuraavassa verkossa on kolme komponenttia: {1, 2, 3}, {4, 5, 6, 7} ja {8}.

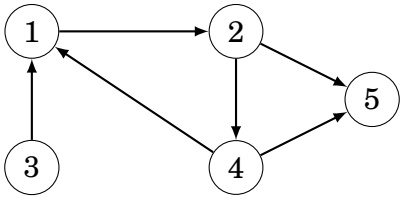


Puu (*tree*) on yhtenäinen verkko, jossa on n solmua ja $n-1$ kaarta. Siinä jokaisen solmun välillä on yksikäsitteinen polku, ja jos minkä tahansa kaaren poistaa, verkko ei ole enää yhtenäinen. Esimerkiksi seuraava verkko on puu:



Kaarten suunnat

Verkko on suunnattu (*directed*), jos verkon kaaria pystyy kulkemaan vain niiden merkittyyn suuntaan. Esimerkiki seuraava verkko on suunnattu:



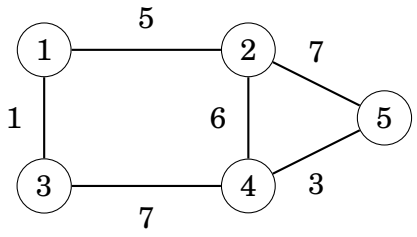
Yllä olevassa verkossa solmusta 3 on polku kaikkiin muihin verkon solmuihin. Esimerkiksi solmusta 3 pääsee solmuun 5 pääsee polkua $3 \to 1 \to 2 \to 5$. Sen sijaan solmusta 5 ei lähde polkua mihinkään muuhun solmuun.

Suunnattu verkko on vahvasti yhtenäinen (*strongly connected*), jos mistä tahansa solmusta on polku mihin tahansa toiseen solmuun.

Sykli (*cycle*) on polku, jonka ensimmäinen ja viimeinen solmu on sama. Esimerkiksi yllä olevassa verkossa on sykli $1 \to 2 \to 4 \to 1$. Jos verkossa ei ole yhtään sykliä, se on syklitön (*acyclic*).

Kaarten painot

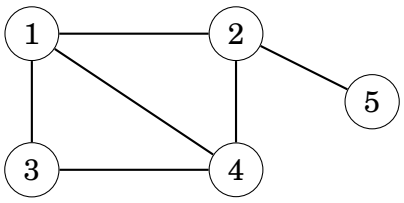
Painotetussa (*weighted*) verkossa jokaisesta kaaresta tiedetään sen paino. Tavallinen tulkinta on, että painot kuvaavat kaarien pituuksia. Seuraavassa on esimerkki painotetusta verkosta:



Painotetussa verkossa polun pituus on sen kaarten painojen summa. Esimerkiksi polun $1 \to 2 \to 5$ pituus on $5 + 7 = 12$ ja polun $1 \to 3 \to 4 \to 5$ pituus on $1 + 7 + 3 = 11$. Jälkimmäinen polku on lyhin polku solmusta 1 solmuun 5.

Naapurit ja asteet

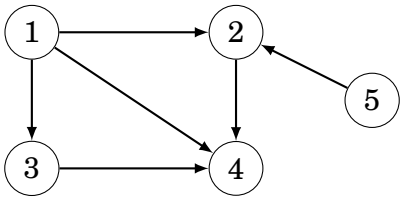
Kaksi solmua ovat naapureita (*neighbor*), jos ne ovat vierekkäin eli niiden välillä on kaari. Solmun aste (*degree*) on sen naapurien määrä. Esimerkiksi seuraavassa verkossa solmun 2 naapurit ovat 1, 4 ja 5, joten sen aste on 3.



Verkon solmujen asteiden summa on $2m$, missä m on kaarten määrä. Tämä johtuu siitä, että jokainen kaari lisää kahden solmun astetta yhdellä. Niinpä solmujen asteiden summa on aina parillinen.

Verkko on säännöllinen (*regular*), jos jokaisen solmun aste on vakio d. Verkko on täydellinen (*complete*), jos jokaisen solmun aste on $n - 1$ eli verkossa on kaikki mahdolliset kaaret solmujen välillä.

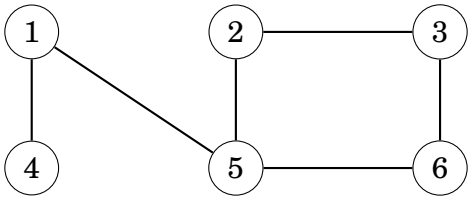
Suunnatussa verkossa lähtöaste (*outdegree*) on solmusta lähtevien kaarten määrä ja tuloaste (*indegree*) on solmuun tulevien kaarten määrä. Esimerkiksi seuraavassa verkossa solmun 2 lähtöaste on 1 ja tuloaste on 2.



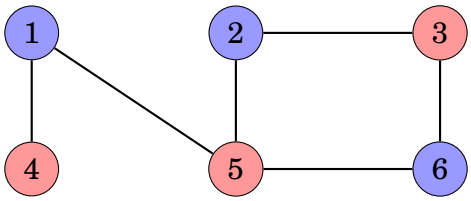
Väritykset

Verkon värityksessä (*coloring*) jokaiselle solmulle valitaan tietty väri niin, että millään kahdella vierekkäisellä solmulla ei ole samaa väriä. Verkko on kaksijakoinen (*bipartite*), jos kaksi väriä riittää sen värittämiseen.

Esimerkiksi verkko



on kaksijakoinen, koska sen voi värittää näin:

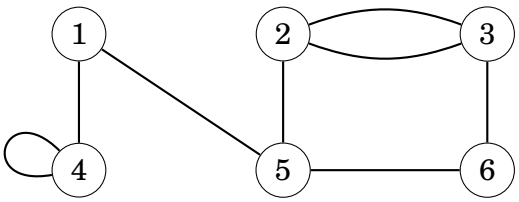


Verkko on kaksijakoinen tarkalleen silloin, kun siinä ei ole sykliä, johon kuuluu pariton määrä solmuja.

Yksinkertaisuus

Verkko on yksinkertainen (*simple*), jos mistään solmusta ei ole kaarta itseensä eikä minkään kahden solmun välillä ole monta kaarta samaan suuntaan. Usein oletuksena on, että verkko on yksinkertainen.

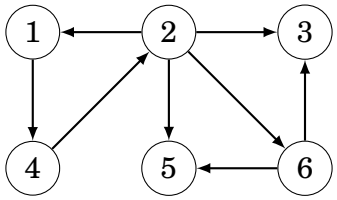
Esimerkiksi verkko



ei ole yksinkertainen, koska solmusta 4 on kaari itseensä ja solmujen 2 ja 3 välillä on kaksi kaarta.

11.2 Verkko muistissa

On monia tapoja pitää verkkoa muistissa algoritmissa. Sopiva tietorakenne riippuu siitä, kuinka suuri verkko on ja millä tavoin algoritmi käsittelee sitä. Seuraavaksi käymme läpi kolme tavallista vaihtoehtoa käyttäen esimerkkinä seuraavaa verkkoa:



Vieruslistat

Yleisin tapa pitää verkkoa muistissa on käyttää vieruslistaesitystä. Siinä jokaisella solmulla on vieruslista (*adjacency list*), joka kertoo, mihin solmuihin siitä pääsee suoraan kaarella. Useimmat verkkoalgoritmit pystyy toteuttamaan tehokkaasti käyttäen vieruslistaesitystä.

Vieruslistoja varten voi luoda esimerkiksi taulukon vektoreita

```
vector<int> v[N];
```

minkä jälkeen kaaret voi lisätä näin:

```
v[1].push_back(4);
v[2].push_back(1);
v[2].push_back(3);
v[2].push_back(5);
v[2].push_back(6);
v[4].push_back(2);
v[6].push_back(3);
v[6].push_back(5);
```

Taulukon koko N on valittu niin suureksi, että taulukossa on oma vieruslista jokaiselle verkossa esiintyvälle solmulle.

Vieruslistaesityksen avulla voi käydä tehokkaasti läpi kaikki tietystä solmusta lähtevät kaaret. Se onnistuu seuraavasti solmulle s:

```
for (int i = 0; i < v[s].size(); i++) {
    // käsittele solmu v[s][i]
}
```

Jos verkon kaarilla on painot, vieruslistat voi rakentaa esimerkiksi niin, että jokainen listan solmu on pari, jossa on solmun tunnus ja siihen johtavan kaaren paino. Tällöin verkon määrittelystä tulee

```
vector<pair<int,int>> v[N];
```

ja esimerkiksi kaari solmusta 2 solmuun 5 painolla 4 lisätään näin:

```
v[2].push_back({5,4});
```

Vierusmatriisi

Vierusmatriisi (*adjacency matrix*) kertoo jokaisesta kaaresta, onko se mukana verkossa. Matriisista on tehokasta tarkistaa, onko kahden solmun välillä kaari, mutta toisaalta matriisi vie paljon tilaa, jos verkko on suuri.

Vierusmatriisi on yleensä järkevää tallentaa taulukkona

```
int v[N][N];
```

jossa v[a][b] on 1, jos solmusta a on kaari solmuun b, ja muuten 0. Seuraava koodi lisää esimerkin kaaret verkkoon:

```
v[1][4] = 1;
v[2][1] = 1;
v[2][3] = 1;
v[2][5] = 1;
v[2][6] = 1;
v[4][2] = 1;
v[6][3] = 1;
v[6][5] = 1;
```

Jos verkko on painotettu, luvun 1 sijasta vierusmatriisin voi tallentaa luontevasti kyseisen kaaren painon.

Kaarilista

Kaarilista (*edge list*) sisältää kaikki verkon kaaret. Kaarilista on hyvä tapa tallentaa verkko, jos algoritmissa täytyy käydä läpi kaikki verkon kaaret eikä ole tarvetta etsiä kaarta alkusolmun perusteella.

Kaarilistan voi tallentaa esimerkiksi vektoriin

```
vector<pair<int,int>> v;
```

jossa jokaisessa solmussa on parina kaaren alku- ja loppusolmu. Kaaret lisätään listalle näin:

```
v.push_back(make_pair(1,4));
v.push_back(make_pair(2,1));
v.push_back(make_pair(2,3));
v.push_back(make_pair(2,5));
v.push_back(make_pair(2,6));
v.push_back(make_pair(4,2));
v.push_back(make_pair(6,3));
v.push_back(make_pair(6,5));
```

Toinen tapa toteuttaa kaarilista on tallentaa tiedot kaarten alku- ja loppusolmuista taulukkoihin seuraavaan tapaan:

```
a[1] = 1; b[1] = 4;
a[2] = 2; b[2] = 1;
a[3] = 2; b[3] = 3;
a[4] = 2; b[4] = 5;
a[5] = 2; b[5] = 6;
a[6] = 4; b[6] = 2;
a[7] = 6; b[7] = 3;
a[8] = 6; b[8] = 5;
```

Osa III

Uusia haasteita