Tietorakenteiden harjoitustyö

Toteutusdokumentti

Tehnyt: Taneli Virkkala

Opiskelijanumero: 014013009

Ohjaaja: Kristiina Paloheimo

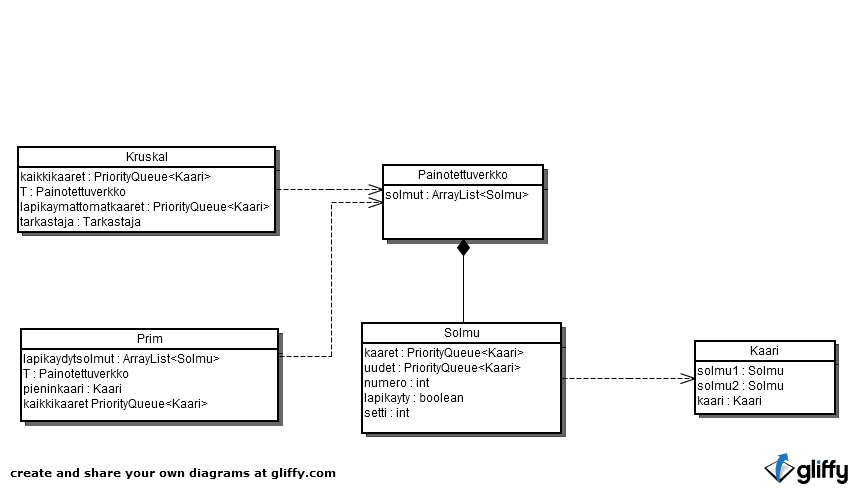
Aihe: Verkon virittävät puut

Kurssi: Tietorakenteiden harjoitustyö: loppukesä 2012

Yliopisto: Helsingin yliopisto, Tietojenkäsittelytieteen laitos

Päivämäärä: 02.09.2012

1. **Ohjelman yleisrakenne**



Painotettuverkko

Koostuu solmuista (ArrayList<Solmu>). Eli siis sisältää solmuolioita. Metodeihin kuuluu mm. solmujen lisäystä sekä tietyn solmun hakua. Verkko on siis tässä tapauksessa vain lista solmuista. Se on myös suuntaamaton. Tosin minun kaarenlisäysmetodini (Painotettuverkko.lisaaKaari(…)) lisää sekä lähtö- että päätesolmulle kaaren. Suunnattuverkko olisi siis mahdollinen.

Solmu

Solmulla on numero, joka kertoo solmun ”nimen”. Eli siis numerolla viitataan tähän kyseiseen solmuun. Jokaisella solmulla on minimikeko omista kaaristaan (PriorityQueue<Kaari>). Kaaret ovat aina suuruusjärjestyksessä. Solmu tuntee naapurinsa vain kaarien kautta.

Kaari

Kaarella on lähtösolmu (Solmu1) sekä päätesolmu (Solmu2). Sen lisäksi sillä on paino (paino). Kaari on Comparable luokka, joten kaaria voi vertailla toisiinsa painon avulla.

Prim

Algoritmille on annettu parametreina aloitussolmu r sekä Painotettuverkko G. G:stä siis etsitään virittävä puu T. Aluksi r:n pieninkaari poimitaan, jonka jälkeen itse kaari lisätään T:hen. Sekä r että sen pienimmän kaaren päätesolmu merkitään käydyiksi. Sen jälkeen siirretään kaikki r:n ja päätesolmun kaaret isoon Minimikekoon (PriorityQueue<Kaari> kaikkikaaret). Nyt aina poimitaan kaikistakaarista pienin kaari. Jos pienimmän kaaren päätesolmua (Solmu2) ei ole merkitty käydyksi, se merkitään ja Solmu2:sen kaaret lisätään kaikkiinkaariin. Jos taas Solmu2 on käyty, poimitaan keosta uusi pienin kaari. Algoritmi loppuu kun virittävä puu T on valmis.

Kruskal

Algoritmille annetaan vain verkko, josta virittävä puu luodaan. Aluksi otetaan kaikkien solmujen kaaret minimikekoon kaikkikaaret (PriorityQueue<Kaari>). Sen jälkeen aletaan aina pienin kaari poistaen merkitä kaaren lähtösolmua (Solmu1) ja päätesolmua (Solmu2) läpikäydyiksi. Lisäksi tehdään lisäys poistetulle kaarelle kekoon kaymattomatkaaret(PriorityQueue<Kaari>) Kun kaikki solmut on läpikäyty, yhdistetään metsät (setit) etsimällä erilaisuudet ja lisäämällä muutama kaari. Kaari napataan tässä tapauksessa käymättömistä kaarista. Algoritmi loppuu kun käymättömät kaaret on tyhjä.

Käyttöliittymä

Käyttöliittymässä voi luoda itse oman verkon kaarineen ja solmuineen. Verkossa on oltava väh. 2 solmua ja 1 kaarta. Lisäksi sen on oltava yhtenäinen eli joka solmusta pääsee jotain reittiä pitkin toiseen solmuun. Aika mitataan algoritmeille System.nanoTime ()- metodilla. Myös virittävät puut tulostetaan. Verkon voi vaihtoehtoisesti myös napata tiedostosta verkot-kansiosta. Verkot luetaan ascii merkeistä muunnettuna numeroarvoiksi.

1. **Aika- ja tilavaativuudet**

Kruskalin ja Primin algoritmit vievät aikaa O(E log V). E tarkoittaa kaarien lukumäärää ja V solmujen. Ohjelman pitäisi siis tuottaa tulokset ajassa O(2x(E log V)). Tilavaativuus on solmujen lukumäärä O(V). Ohjemassani kuitenkin on myös minimikeko kaarille, joten tilavuus kasvaa. Lisäksi muutama ylimääräinen for/while-lause.

1. **Suorituskyky- ja O-analyysivertailu**

Mielestäni sain ohjelman optimoimaan aika hyvin. Kruskalin algoritmissä tein niin, että vasta kun kaikki solmut on käyty läpi, katsotaan vasta silloin että verkko on yhtenäinen. Näin säästettiin turhalta alkuläpikäymiseltä. Primin algoritmissä tuhotaan alkuperäinen parametrinä annettu verkko, joten suoritus päättyy kun kaikki solmut on käyty läpi. On vaikea sanoa olisiko Primiä pystynyt optimoimaan enempää ilman että olisi käyttänyt liikaa valmiita tietorakenteita.

Primin algoritmi voittaa Kruskalin lähes aina isommilla verkoilla (yli 6 solmua, yli 8 kaarta). Tämä varmaan johtuu myös verkon rakenteesta paljon. Onko samanpainoisia kaaria ja onko alussa raskaita vai kevyitä kaaria. Kruskalin algoritmin suurimpana ongelmana on metsien yhdistäminen eli tarkastaminen milloin verkko on yhtenäinen. Primi on aina viimeisen solmun löydettyään luonut virittävän puun. Omasta mielestänikin Primin algoritmi on yleisesti järkevämpi virittävän puun luomisessa, koska se muistuttaa luonnollisesti ihmisen aivojen toimintaa.

Kruskalin ja Primin algoritmit vievät aikaa O(E log V). E tarkoittaa kaarien lukumäärää ja V solmujen. Ohjelman pitäisi siis tuottaa tulokset ajassa O(2x(E log V)). Tilavaativuus on solmujen lukumäärä O(V).

1. **Työn mahdolliset puutteet ja parannusehdotukset**

Ajanpuutteen vuoksi en päässyt tekemään minimikeon toteutusta mikä olisi ollut todella kriittistä tällaisessa työssä. Myös satunnaisgeneraattori verkoille olisi ollut hauska tehdä. Lisäksi verkon graafisen tulostuksen olisi voinut lisätä. Lisäksi vertailuun olisi voinut laittaa jotain muutakin kuin vain ajan ja virittävän puun tulostuksen.

1. **Lähteet**

Patrick Floreenin Tietorakenteet kurssin materiaali: <http://www.cs.helsinki.fi/u/floreen/tira2012/tira.pdf>

Primin algoritmi: Robert C. Prim (1957)

Kruskalin algoritmi: Joseph Kruskal (1956)

1. **Käyttöohjeet**

Käyttöliittymässä (tiran harkkatyo.java) painamalla (a) voi lisätä solmun verkkoon, jonka jälkeen syötetään kokonaisluku (int). Samaa solmua ei voi lisätä kahdesti.

Painamalla (b) voi lisätä kaaren verkossa olevan kahden solmun välille. Painon saa myös itse päättää. Kahden solmun välillä voi olla vain yksi kaari (minun tietorakenteessani voi kuitenkin on kaksi).

Painamalla (c) voi ajaa algoritmit. Tuloksessa näkyy aika nanosekunteina sekä viritetyt puut, jotka ovat syntyneet. Primin algoritmiin on valittava aloitussolmuksi jokin jo verkossa oleva solmu.

Painamalla (d) voi tulostaa tämänhetkisen luoman käyttäjän testiverkon.

Painamalla (e) käyttäjä voi verkot kansioista ottaa jonkin valmiin verkon.

Painamalla (q) ohjelman suoritus loppuu.