Ilkka Lehikoinen
013787637

Aineopintojenharjoitustyö:
Tietorakenteet ja algoritmit

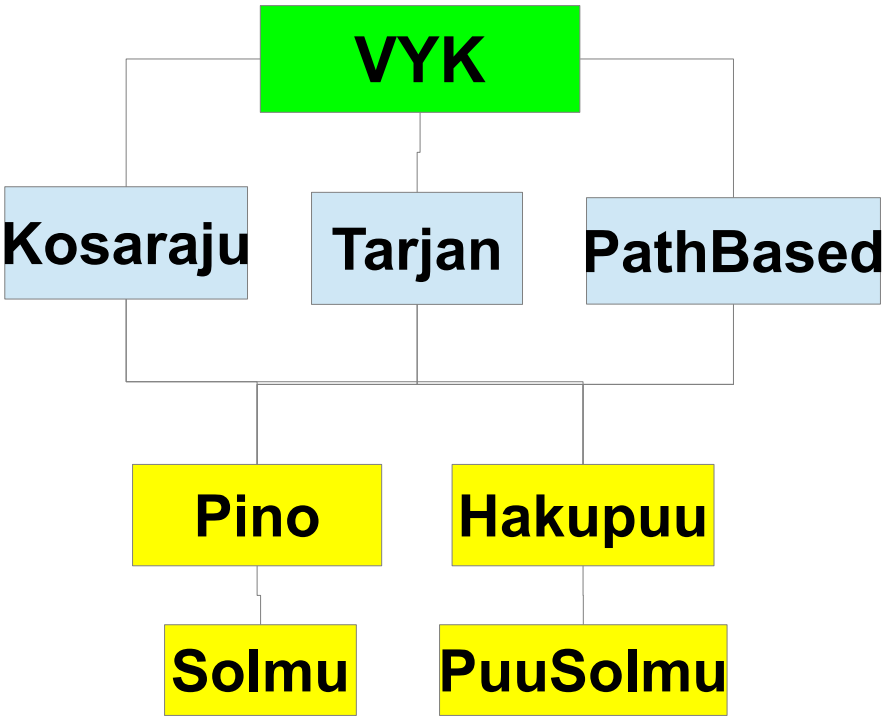
Verkon vahvasti yhtenäiset komponentit

Toteutusdokumentti

Ohjelman yleisrakenne

Ohjelman keskiössä on luokka VYK (vahvasti yhtenäiset komponentit), joka suorittaa varsinaisten algoritmien ajon generoimillaan suunnatuilla verkoilla ja laskee näiden suoritukseen kuluvan ajan. Lisäksi ohjelmassa on kolme erillistä algoritmiä (Kosaraju, Tarjan ja PathBased) verkon vahvasti yhtenäisten komponenttien löytämiseksi. Lisäksi ohjelmistoon kuuluvat apuluokat Solmu ja Pino, joilla on toteutettu algoritmien tarvitsema pino, sekä apuluokat PuuSolmu ja HakuPuu, joilla on toteutettu yksinkertainen rakenne tiedon tallennusta varten.

VYK-luokassa generoidaan suorituskykytestauksessa tarvittavat verkot ja suoritetaan algoritmit samoille verkoille, jotta voidaan tehdä havaintoja niiden suoritusajasta suhteessa toisiinsa. Muut luokat kukin käsittää yhden algoritmin, jolla etsitään verkosta vahvasti yhtenäiset komponentit.



Kuva 1. Ohjelmiston rakenne

Lisäksi ohjelmisto käsittää apuluokat Pino ja Solmu sekä HakuPuu ja PuuSolmu.

VYK

Luokassa generoidaan suuria suunnattuja verkkoja muiden luokkien algoritmien suorituskykytestejä varten.

Lisäksi suorituskykytestaus suoritetaan tämän luokan kautta, laskemalla kunkin algoritmin verkon vahvasti yhtenäisten komponenttien löytämiseen käyttämä aika.

Kosaraju

Kosarajun algoritmi on tuttu jo luento monisteista [1] ja Cormenin kirjasta [2]. Siinä suoritetaan ensin verkon syvyyssuuntainen läpikäynti sattumanvaraisesta solmusta, solmut talletetaan pinoon jälkijärjestyksessä eli kun solmu käsittely on tullut loppuun, lisätään solmu pinoon.
Tämän jälkeen verkolle suoritetaan transpoosi, jossa verkon kaarien suunnat vaihdetaan päinvastaisiksi.
Lopuksi tälle transponoidulle verkolle suoritetaan syvyyssuuntainen läpikäynti alkaen pinon pinnalta olevasta solmusta. Kun läpikäynti ei enää pääse uusiin solmuihin, on löytynyt yksi vahvasti yhtenäinen komponentti. Löydettyihin komponentteihin kuuluvat solmut poistetaan pinosta ja syvyyssuuntainen läpikäynti käynnistetään uudelleen pinon pinnalla olevasta solmusta. Ja tätä jatketaan niin kauan kuin pinossa on jäljellä käsittelemättömiä solmuja.

Kosarajun algoritmin aikavaativuus olisi vieruslistalla toteutettuna $O(V+E)$, mutta vierusmatriisin läpikäynnin vuoksi algoritmin aikavaativuus on $O(V^2)$.

Tarjan

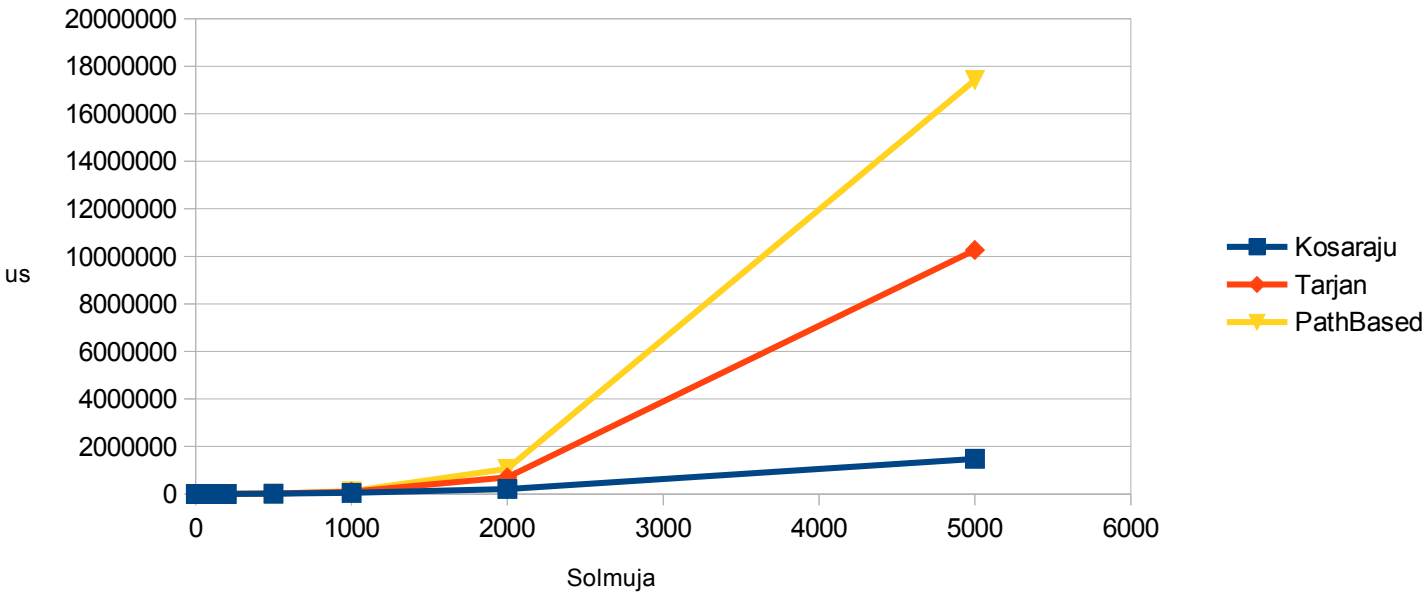
Tarjan-algoritmissa suoritetaan syvyyssuuntaisia läpikäyntejä, niin kauan kuin verkossa on vielä käsittelemättömiä solmuja. Solmut laitetaan pinoon sitä mukaan kun ne tulevat käsittelyyn. Kun läpikäynti palaa niihin alipuusta, solmu otetaan pinosta ja tutkitaan, onko se jonkin vahvasti yhtenäisen komponentin juuri, jolloin sitä ennen pinosta poistetut solmut muodostavat vahvasti yhtenäisen komponentin. Myös Tarjan algoritmin aikavaativuus on vierusmatriisilla toteutettuna $O(V^2)$, mutta olisi mahdollista toteuttaa vieruslistalla aikavaativuudella $O(V+E)$.

PathBased

Kuten Tarjankin niin myös PathBased-algoritmi perustuu syvyyssuuntaiseen läpikäyntiin. Toisin kuin Tarjan-algoritmissa PathBased-algoritmi käyttää kahta pinoa. Ensimmäinen pino ylläpitää tietoa nykyisessä komponentissa olevista solmuista ja toinen pino pitää yllä listaa nykyisellä läpikäyntipolulla olevista solmuista. Solmut ovat siinä järjestyksessä, jossa syvyyssuuntainen läpikäynti ne saavuttaa (Esijärjestys).

Saavutetut aika- ja tilavaativuudet

Kun katsoo pintapuolisesti näiden algoritmien rakennetta niin äkkiseltään voisi luulla että Kosaraju olisi hitain, koska se käsittää kaksi syvyyssuuntaista läpikäyntiä ja verkon transpoosin, kun taas toiset algoritmit käsittävät vain yhden syvyyssuuntaisen läpikäynnin. Kuvasta 1. kuitenkin näkyy, että Kosarajun algoritmi osoittautui kaikkein tehokkaimmaksi.

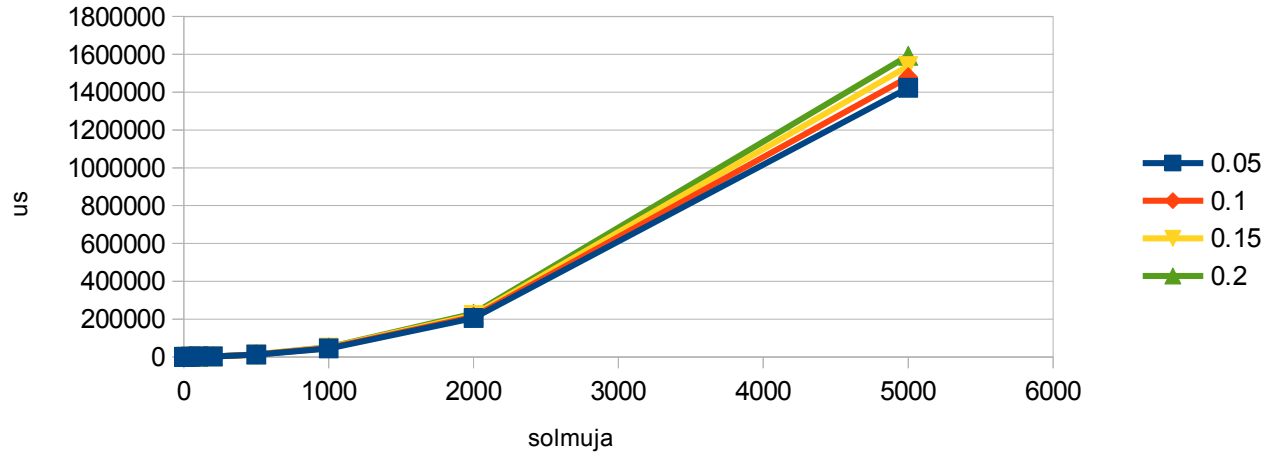


Kuva 1. Esimerkki algoritmien nopeuserosta kaarien tiheydellä 0,1 (Arvot keskiarvoja, n=10)

Kuvasta sitä ei näy selkeästi, mutta Kosarajun algoritmin aikavaativuus on lähellä määriteltyä $O(n^2)$, kun taas sekä Tarjan, että PathBased algoritmit jäävät tästä ja ovat lähes kuutiollisia $O(n^3)$. Koska Tarjan ja PathBased ovat algoritmina hyvin toistensa kaltaisia, vain sillä erotuksella että PathBased käyttää kahta pinoa, niin lähtisin hakemaan ongelmaan pinosta ja siihen liityvistä toiminnoista.

Suorituskyky- ja O-analyysivertailu

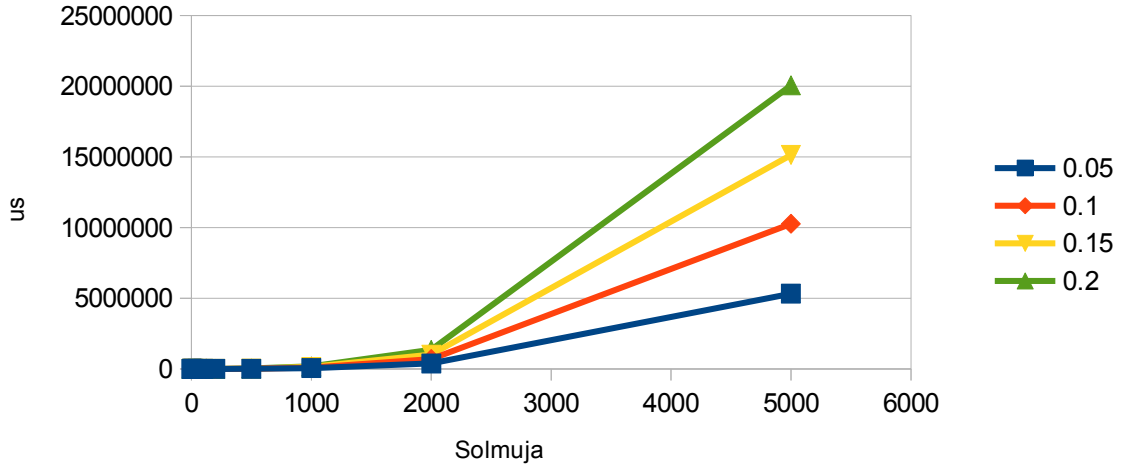
Testattaessa algoritmeja eri kaarien tiheyksillä niin kuvasta 2. huomataan, että Kosaraju on suhteellisen immuuni kaarien määrän kasvulle, kun taas sekä Tarjan, että PathBased hidastuvat lineaarisesti suhteessa kaarien määrään. (Kuvassa 3. Tarjan algoritmi)



Kuva 2. Kosaraju algoritmin suoritusnopeuden riippuvuus kaarien määrästä. (Arvot keskiarvoja, n=5)

Jos Algoritmit olisi toteutettu vieruslistalla, jolloin niiden aikavaativuus olisi ollut O(n+m), tämä

suoritusnopeuden lineaarinen riippuvuus verkon kaarien määrässä olisi ollut odotettavissa.



Kuva 3. Tarjan algoritmin suoritusnopeuden riippuvuus kaarien lukumäärästä. (Arvot keskiarvoja, n=5)

Puutteet ja parannusehdotukset

HakuPuu, johon algoritmit keräävät pinosta vahvasti yhtenäisen komponentin solmut ja samalla tarkistaa ettei mikään solmu esiinny missään komponentissa kahdesti, pitäisi tasapainottaa lyhyempien hakuaikojen varmistamiseksi. Tällä saataisiin hakupuuhun lisäyksen aikavaativuus pienenemään O(n):sta O(log n):ään.

Kun kaikki komponentit on saatu hakupuuhun, niin ne poistetaan kaikki kerralla, eikä hakupuuhun tänä aikana enää lisätä solmuja, ja kun vielä solmujen järjestyksellä ei ole väliä niin solmut voidaan ottaa suoraan juuresta eikä hakupuuta ole enää tarpeen tasapainottaa, näin säästetään tasapainottamisen kuluttama aika hakupuusta poistamisen aikana pois.

Hakupuu sijaan tähän tehtävään olisi voinut myös ajatella kekoa, joten suoritusnopeutta voisi testata myös ratkaisulla, jossa HakuPuu on korvattu keolla.

Algoritmiluokat voisi ohjelmoida myös vieruslistoille, jolloin päästäisiin matriisin läpikäynnistä solmujen ja kaarien läpikäyntiin, joka vähentäisi varsinkin harvassa verkossa matriisin tyhjien kaarien tarkistamista. Samalla algoritmien aikavaativuudet pienisivät O(n+m):ään.

Lisäksi voisi ohjelmoida muunnokset vierusmatriisista vieruslistaan ja päinvastoin.

Koska jo reilun 2500 solmun verkoilla javan ohjelmalle varaama muisti täytyy rekursiopinosta olisi mielenkiintoista kokeilla syvyyssuuntaisen läpikäynnin toteuttaminen rekursion sijaan iteraatiolla. Ensinnäkin sen vaikutus muistiin ja toisaalta suoriitusnopeusvertailu rekursio vastaan iteraatio.

Lähteet

[1] Introduction to Algorithms 3rd ed. T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein. The MIT Press 2009.

[2] Luentomonisteet: http://www.cs.helsinki.fi/u/floreen/tira2013/tira.pdf

[3] Tarjan: http://en.wikipedia.org/wiki/Tarjan%27s_strongly_connected_components_algorithm

[4] PathBased: http://en.wikipedia.org/wiki/Path-based_strong_component_algorithm