# A\* ja Dikstran algoritmin vertailu

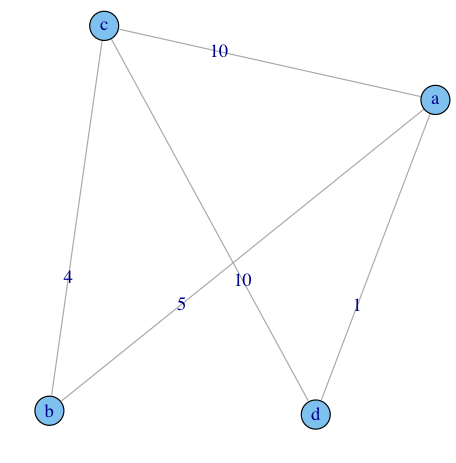
Tehtävänä on toteuttaa kaksi lyhimmän reitin hakemiseen tarkoitettua algoritmia: A\* ja Dikstra sekä vertailla näitä toteutuksia empiirisesti. Toteutusta varten toteutetaan oma aputietorakenne verkosta.

## Algoritmien analyysi

A\* on heuristinen lyhyimmän polun hakemisen algoritmi, joka on kehitetty Dikstran algoritmistä. Hart et al. (1968) esittävät algoritmin toiminnan ja kiinnittävät huomiota heuristiseen algoritmiin ja sen valinnan ongelmiin. Heuristisen algoritmin toiminnan tulisi sopia ongelma-alueeseen, mikä herättääkin mielenkiintoisia kysymyksiä, kuten mikä on sosiaalisiin verkostoihin soveltuva heuristinen algoritmi.

Dikstran algoritmi keskittyy myös lyhyimmän polun hakemiseen. Ideana on käydä läpi kaikki solmut syvyyssuuntaisessa haussa ja päivittää kullekkin päivittää tämän mukaan etäisyyksiä muihin alkioihin (Dijkstra, 1959). Koska toteutuksessa tallennetaan pinoon kaikki solmut, on toteutuksen muistinkäyttö suuruusluokkaa *O( solmu )*. Alkuperäisen algoritmin pahimman tapauksen käsittely suoriutui ajassa *O( kaari2 )*, mutta sen kehitelty versio, missä käytetään kekoa apuna suoriutuu ajassa *O(solmu + kaari log( kaari ) )* (Cormen et al, 2001; Fredman & Tarjan, 1984).

## Keskeiset toteutusluokat

**DataReader**

Toteuttaa tiedoston lukemisen tiedostoformaatista painotetuksi verkoksi ja palauttaa kaikki verkon solmut. Esimerkki tiedostomuodosta ja sitä vastaavasta verkosta

a:b:5

a:c:10

a:d:1

c:b:4

d:c:10

**Graph**

Vastaa yhtä luettua verkkoa kokonaisuudessaan. Sisältää kaikki verkon solmut sekä tarvittavia apumetodeita esimerkiksi verkon esittämiseen matriisimuodossa sekä syvyyssuuntaiseen hakuun, sekä luo lyhyimmät polut graafissa.

**Node**

Node vastaa yhtä verkon solmua. Sillä on nimi sekä lista vierussolmuistaan.

## Tukiluokat

**MyList**

Graafin kaikki solmut ja yksittäisen solmun vierussolmut tulee tallentaa äärettömän pituiseen listaan. Toteuttaa rajapinnan java.util.List, sisäisenä toteutuksena yksisuuntainen aloitussolmullinen lista, en ainakaan usko, että tulee erityisen tarpeelliseksi päästä kulkemaan molempiin suuntiin.

**MyHeap**

Dikstan algoritmia voidaan nopeuttaa kekototeutuksella. Sitä varten toteutetaan kekoluokka binääripuutoteutuksella. Toteuttaa rajapinnan java.util.Queue.

**MyMap**

Toteutetaan hyödyllinen aputietorakenne MyMap, joka vastaa hajautustaulua. Tämän avulla saavutetaan etuja ainakin tiedoston lukemisessa, kun luodut nodet voidaan tallentaa nimeään vastaavasti. Toteuttaa rajapinnan java.util.Map.

## Toteutussuunnitelma

Ensimmäisessä vaiheessa keskitytään DataReaderin, Graphin ja Noden perustoiminnallisuuksien toimintakuntoisuuteen, käyttäen hyödyksi javan todellisia, implementoitua tiedostorakenteita. Tähän työvaiheeseen varataan yksi viikko.

Tämän jälkeen toteutetaan käytetyt tietorakenteet itsenäisesti (“tukiluokat”), ja siirretään varsinainen toteutus hyödyntämään näitä luokkia. Tähän työvaiheeseen varataan kaksi viikkoa.

Viimeisessä vaiheessa toteutetaan kaksi erilaista lyhyimmän polun hakemista ja vertaillaan näitä empiirisesti. Tähän työvaiheeseen varataan kaksi viikkoa.

Toteutusessa tullaan soveltamaan test driven development-paradigmaa, eli yksikkötestit kirjoitetaan ennen varsinaisen toteutuksen aloittamista.

## Viitteet

Hart, P.E.; Nilsson, N.J.; Raphael, B. (1968): "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *Systems Science and Cybernetics*, IEEE Transactions on , vol.4, no.2, pp.100-107

Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs". *Numerische Mathematik* 1: 269–271

Fredman, M. L.; Tarjan, R. E. (1984). "Fibonacci heaps and their uses in improved network optimization algorithms". *25th Annual Symposium on Foundations of Computer Science (IEEE)*: 338–346.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C. (2001). *Introduction to Algorithms (Second ed.).* MIT Press and McGraw-Hill