Testausdokumentaatio

Testauksessa keskitytään pääosin ohjelman algoritmeihin sekä tietorakenteisiin. Javan valmiit ominaisuudet tiedostonkäsittelyn osalta oletetaan toimiviksi. Samoin getterit ja setterit oletetaan toimiviksi Javan totutulla tavalla, joten näitä ei testata. Kansiosta cobertura löytyy coverage-raportti.
Testausta on myös tehty ajamalla ohjelmaa erilaisilla syötteillä ja reittivaihtoehdoilla ja pyritty varmistamaan ohjelma toimivuus.

Algoritmit Dijkstra ja A*

Algoritmien testaus keskittyy kaikkien kolmen metodin toimivuuden tarkasteluun. testInitialize() metodi alustaa ohjelman vaatimat verkon solmut ja kaaret. Testaan, että luonti onnistuu annettujen solmujen ja kaarien joukosta. testRoute() metodi suorittaa itse reitin haun. Tarkistan, että algoritmi todellakin antaa oikean reitin käyttäjälle. Metodilla testPrint() tarkistan, että polku lähdöstä maaliin tulostuu oikein, jolloin saan varmistettua vielä reitin pituuden sekä edeltäjäsolmujen oikean järjestyksen.

Tietorakenteet keko ja lista

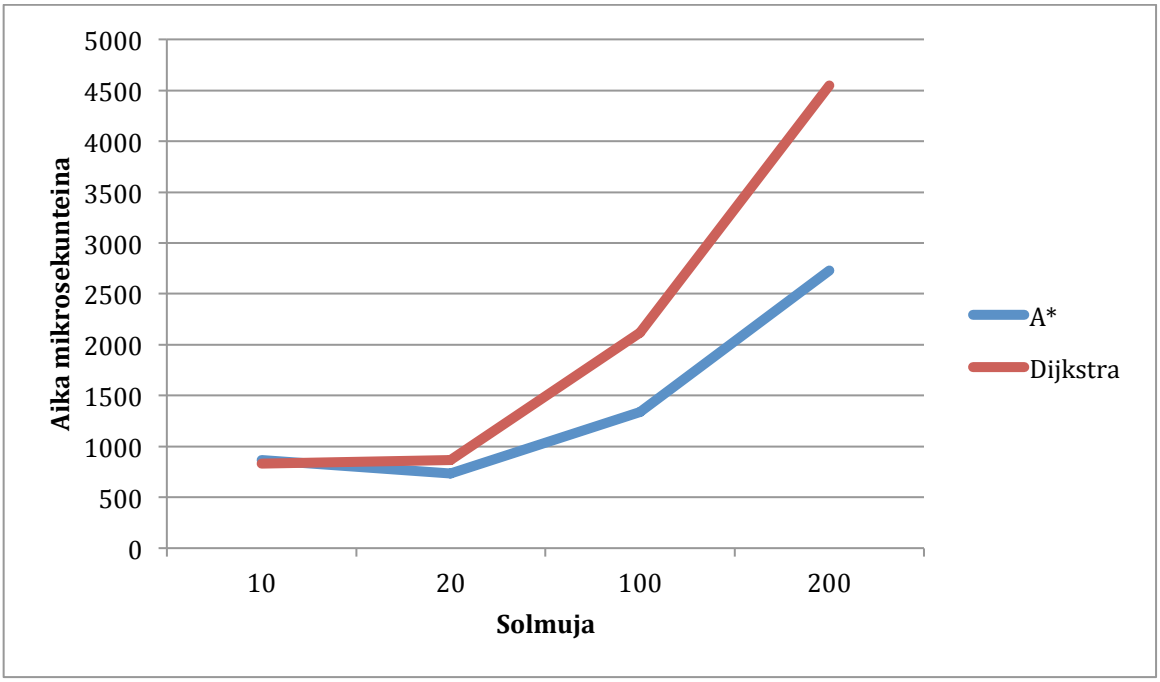
Tietorakenteista testaan kaikki niiden julkiset metodit. Julkiset metodit käyttävät luokan yksityisiä metodeja, joten testien tulokset näyttävät myös ovatko tietorakenteen yksityiset metodit toimineet oikein.

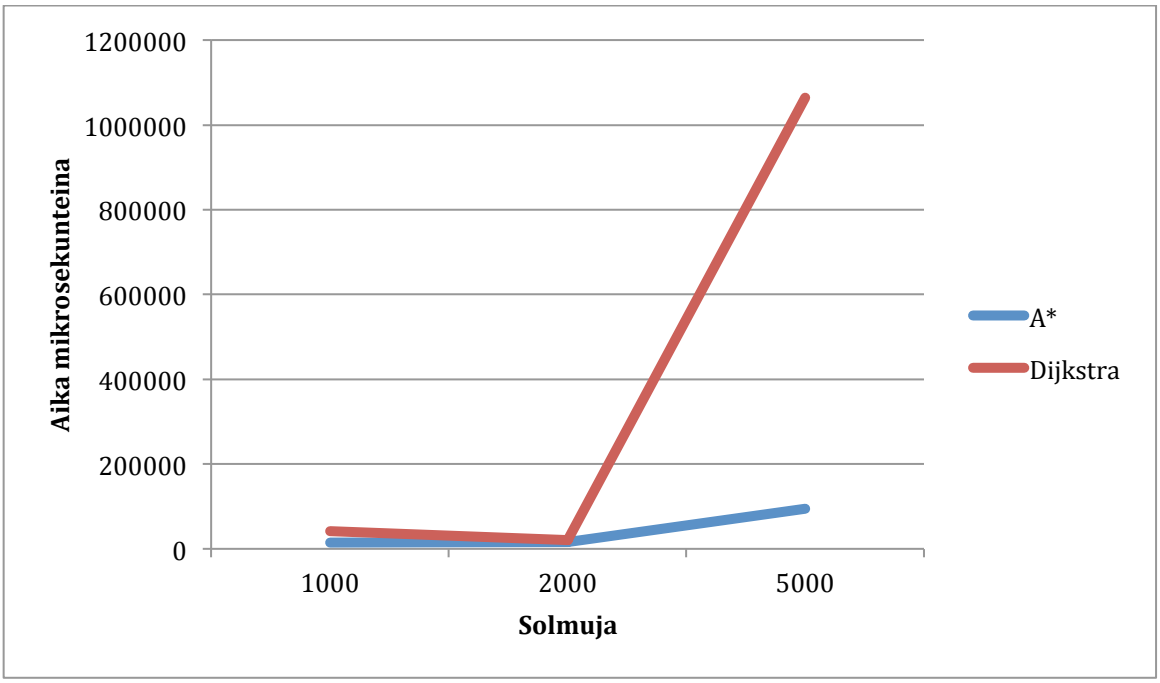
Sen lisäksi kirjoitin luokalle Mapper lyhyet testit, joilla tarkastin erityisesti, että ohjelman lukiessa kartan omaan LinkedList –tietorakenteeseen, kaikki toimii oletetulla tavalla.

Suorituskyky

Ohjelman toteutustavasta johtuen suorituskykytestaus ei ollut kovin suoraviivaista suurempiin karttoihin siirryttäessä. Kartat nimittäin täytyy generoida. Ensin ajattelin tehdä testailua varten oman haaran Githubiin, jossa algoritmeja olisi ajettu 2-ulotteisessa taulukossa mutta tajusin nopeasti, että sekin vaatii jonkin verran työtä ohjelman rakenteen parissa. Päätin siis generoida karttoja joissa on enemmän solmuja ja kaaria. Testauksessa käytetty koodi löytyy githubista erillisestä branchista ‘testing’. Ohjelma on pelkistetty ja käyttäjä voi valita kuinka monta Nodea hän haluaa verkkoon ja kumman algoritmin suoritettavan. Testeissä huomasi, että heuristiikan vaikutus oli melko suuri kun siirrytiin isompiin karttoihin. Ohjelma tosin generoi aina satunnaisen kartan, joten aivan yksi yhteen tuloksia ei voi verrata.

Itse suhtaudun seuraavan sivun kaavioihin hieman varauksella. Luulen, että osittain sattumalta Dijkstra on saanut suurempia ajoaikoja testien satunnaisuuden takia. Toinen mikä selittää noin selkeää eroa, on testeissä käytetty heuristiikan ja kaaripainojen muodostus A*-algoritmille, jonka ansiosta moni kaari on ollut suuruudeltaan erittäin lähellä pisteiden välistä euklidista etäisyyttä.
Seuraava iteraatio testauksessa voisi olla esimerkiksi koodin muokkaus siten, että ajetaan toistuvia hakuja samalla generoidulle satunnaiskartalle. Kuten monesti, tälläkin kertaa ensimmäisen iteraation jälkeen tajusi, kuinka oikeasti oli kannattanut testailla algoritmeja.


5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
Aika mikrosekunteina
10
20
100
200
Solmuja
A*
Dijkstra


1200000
1000000
800000
600000
400000
200000
0
Aika mikrosekunteina
1000
2000
5000
Solmuja
A*
Dijkstra