Testausdokumentti

Tietorakenteiden harjoitustyö

Mitä on testattu

JUnitin avulla on testattu algoritmien ja aputietorakenteiden toimimista yleisellä tasolla, eli esim. sitä, että algoritmit laskevat etäisyydet oikein maalipisteeseen sekä matkalla oleviin pisteisiin, osaavat pitää kirjaa poluista ja päivittävät polun pisteiden attribuutteja oikein. JUnitia ei ole käytetty testaamaan algoritmien käyttämää aikaa.

Algoritmien ajankäyttöä on testattu AlgoritmienVertailu-luokassa toteutettujen toimintojen avulla. Mittaukseen on käytetty System.currentTimeMillis()-toimintoa, joka mittaa ajan millisekunteina tietyllä hetkellä.

Millaisilla syötteillä testaus tehtiin

Ajan mittauksessa oli käytössä kolme erikokoista labyrinttia 2-ulotteisina taulukoina: pieni (20x20 ruutua), keskikokoinen (100x100 ruutua) ja iso (400x400 ruutua). Taulukoihin generoitiin jokaisen testiajon alussa seinät niin, että ruudussa on seinä 30% todennäköisyydellä. Muihin ruutuihin generoitiin jokin luku 1-9, joka ilmaisee ruudun vaikeakulkuisuutta, 1 on helppokulkuisin ja 9 vaikeakulkuisin. Testauksessa käytettiin kaikkien algoritmien testaamiseen kullakin kerralla samaa generoitua labyrinttia, jotta saadut tulokset olisivat vertailukelpoisia.

Lisäksi testasin tyhjällä/seinättömällä labyrintilla kaikkia algoritmeja, erityisesti saadakseni varmuuden siitä, että A* toimii Dijkstraa paremmin tällaisessa tapauksessa, jossa sen pitäisi toimia paremmin. Lähtöpiste oli tässä tapauksessa asetettava niin, että etsittävä polku ei ole suoraan kulmasta kulmaan, sillä muuten A* ei osaa lähteä suoraan maalia kohti, kun kaikki maaliin päin kulkevat polut ovat yhtä pitkiä (esim. suoraan “siksakkia” maalia kohti kulkeva polku on saman pituinen kuin polku, joka menee ensin suoraan vas. yläkulmasta vas.alakulmaan ja sieltä suoraan oik. alakulmaan eli maaliin). Asetin lähtöpisteen siis oikeaan yläkulmaan ja maalipisteen oikeaan alakulmaan.

Pienimuotoisesti testasin myös tehdä labyrintin, jonka lyhimällä polulla on vaikeakulkuista maastoa eli “vuori”, ja katsoin millä vaikeakulkuisuuden arvoilla polku kulkee vuoren yli ja millä kiertää sen. Myös tässä tapauksessa testaus sujui järkevämmin, kun lähtöpiste ja maalipiste olivat labyrintin samalla laidalla.

Miten testit voidaan toistaa

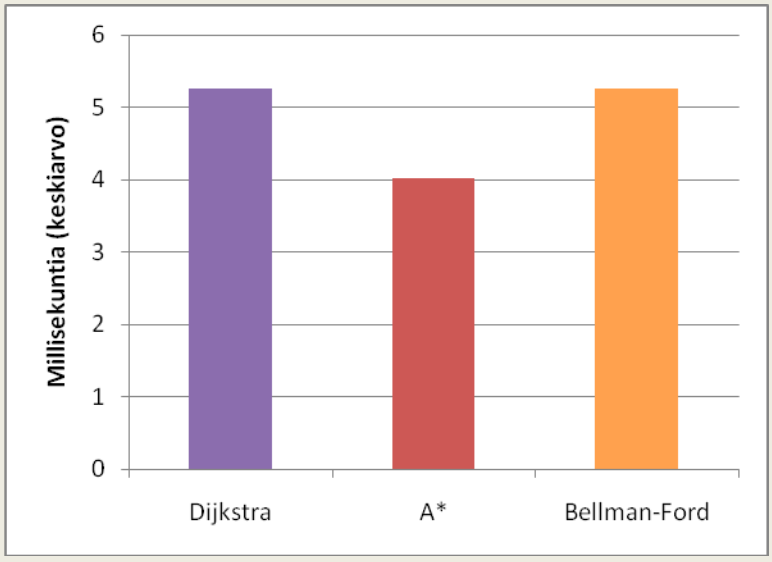
Testit voidaan toistaa ajamalla ohjelma ja antamalla syötteenä minkä kokoisella labyrintillä halutaan testata, sekä haluttu algoritmi. Useamman algoritmin voi ajaa peräkkäin käyttäen samaa labyrinttiä. Tuloksena tulee löytynyt polku (maalista lähtien) ja sen pituus (jos maali saavutettiin) sekä kuinka paljon aikaa kului.

Tulokset graafisessa muodossa

Mittasin algoritmien käyttämää aikaa ajamalla algoritmit useita kertoja ja laskemalla keskiarvot saaduista tuloksista.

Pieni (20 x 20 ruutua)

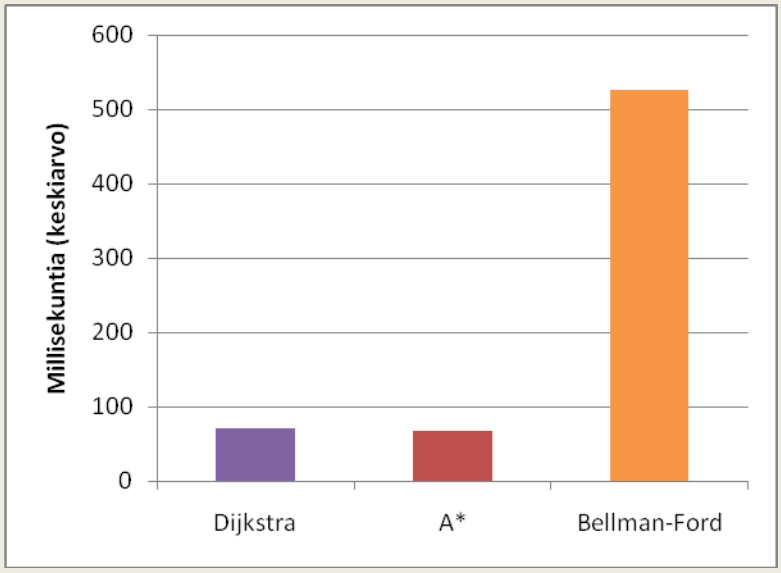
Pienessä labyrintissä ei juurikaan näkynyt eroa käytetyssä ajassa. Kaikki algoritmit suoriutuivat läpikäynnistä nopeasti.



Keskikoko (100 x 100 ruutua)

Keskikokoisessa labyrintissä näkyi jo selkeästi eroa Bellman-Fordin ja muiden välillä.

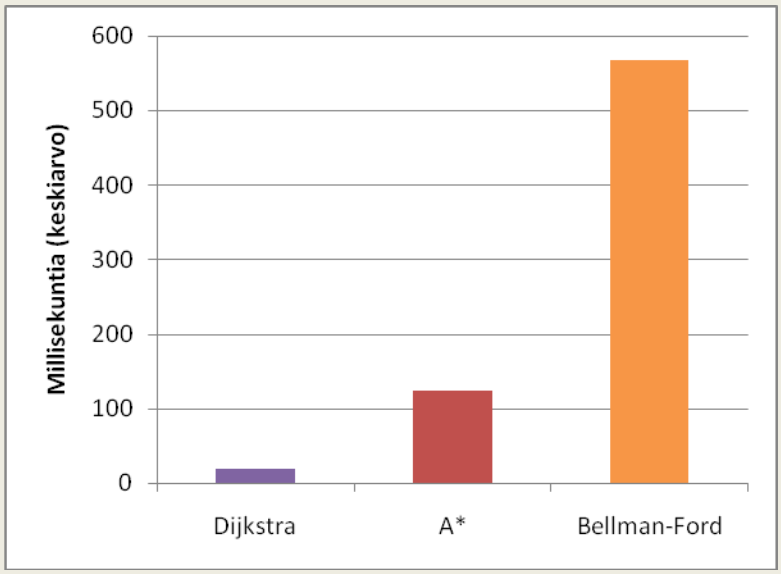
A* ja Dijkstra suoriutuivat keskikokoisesta lähes samassa ajassa.



Keskikoko (100 x 100 ruutua, maalia ei saavutettu)

Tämän testin labyrintissa algoritmit pääsivät vain muutaman askeleen, ennen kuin seinät tulivat vastaan. Käytetty aika on tässä siis käytännössä kulunut pelkkiin alustustoimenpiteisiin.

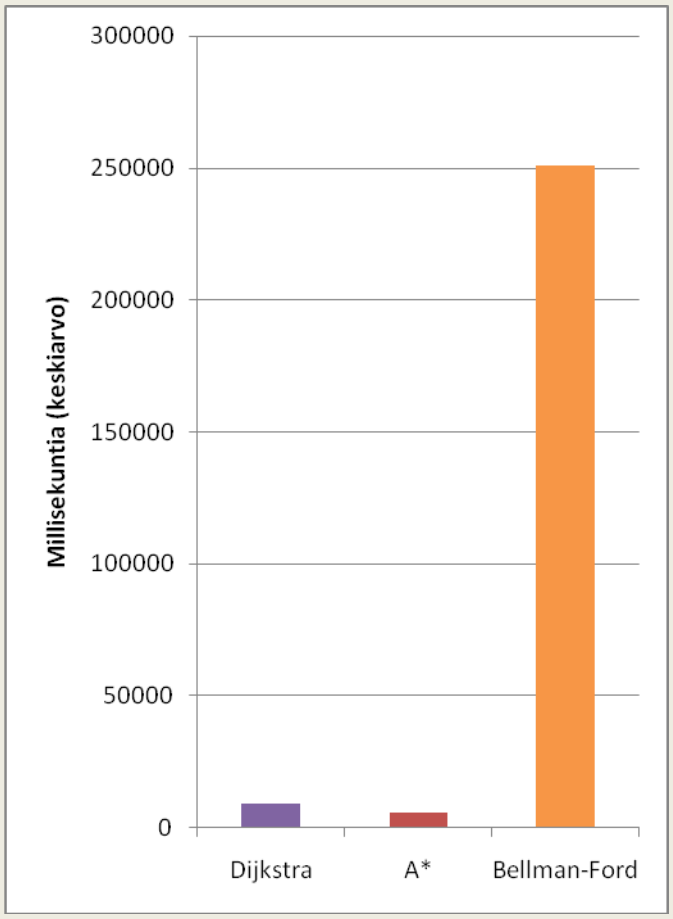
A*:lla menee alustukseen Dijkstraa kauemmin, sillä se laskee pisteille distLoppuunArviot. Bellman-Fordilla aika kuluu kaarien hakemiseen.



Iso (400 x 400 ruutua)

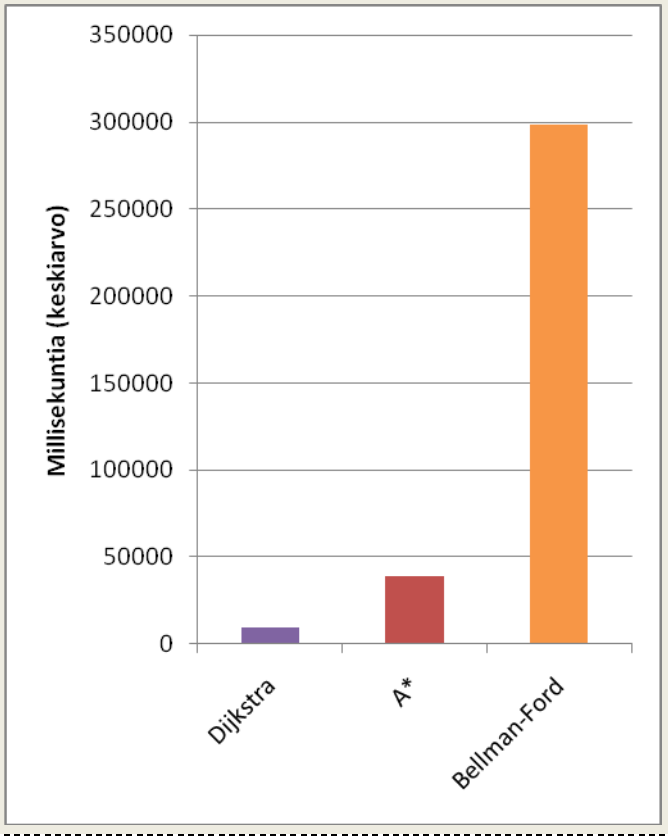
Ero Bellman-Fordin ja muiden välillä näkyy entistä selvemmin.

A* selvisi tästä testilabyrintista hieman Dijkstraa nopeammin.



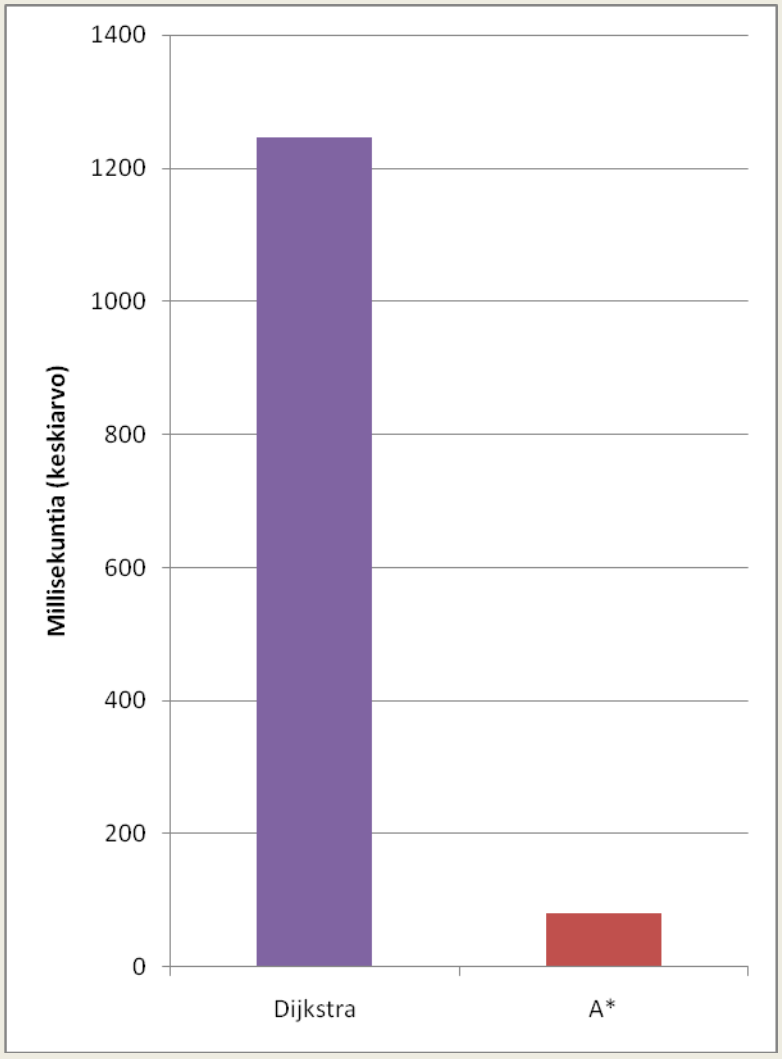
Iso (200 x 800 ruutua)

Hieman eri muotoinen versio isosta labyrintista: tästä Dijkstra selvisi jonkin verran A*:ia paremmin.



Tyhjä (200 x 200 ruutua, lähtöpiste oikea yläkulma ja maalipiste oikea alakulma)

Jätin tästä testistä Bellman-Fordin pois, sillä olennaisempaa oli tässä verrata Dijkstran ja A*:in suoriutumista. Jo näinkin pienellä labyrintilla ero ajankäytössä oli selkeä. A* kävi läpi vain pienen alueen solmuja labyrintin oikeasta laidasta suunnatessaan suoraan kohti maalia, kun Dikstra kävi läpi kaikki solmut.



Löydetyt polut

Alunperin tehdessäni testausta, joissa labyrintin vaikeakulkuisuus ei vaihdellut (siis kaikkien pisteiden maaston arvo oli 1) testauksessa vaihteli paljon se, löytävätkö kaikki algoritmit labyrintista saman vai eri lyhimmän polun, kun lyhimpiä polkuja on useita. Suurimmaksi osaksi algoritmit löysivät keskenään eri polut – joissakin tapauksissa kävi myös niin, että vain Dijkstra ja A*, tai Dijkstra ja Bellman-Ford löysivät samat polut. Kaikki löydetyt polut olivat kuitenkin lyhimpiä.

Testatessani uudelleen labyrinteillä, joissa maaston vaikeakulkuisuus vaihtelee, algoritmit löysivät saman lyhimmän polun huomattavasti useammin.

Vuori

Tein testausta myös labyrintillä, jossa lyhimmällä polulla on vaikeakulkuista maastoa eli “vuori”. Algoritmien löytämät polut vaihtelivat lievästi, mutta ne kuitenkin ylittivät vuoren aina samasta kohtaa (taulukot 1-3). Taulukoissa 4-7 on kuvattu muutamilla erilaisilla vuoren vaikeakulkuisuuden arvoilla, mistä kohtaa algoritmit ylittivät vuoren (esimerkeissä käytössä Dijkstra).

Dijkstra

1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	3
1	1	1	1	1	1	1	3	4
1	1	1	1	1	1	4	4	5
1	1	1	1	1	1	1	3	4
1	1	1	1	1	1	1	1	3
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1

Taulukko 1

A*

1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	3
1	1	1	1	1	1	1	3	4
1	1	1	1	1	1	4	4	5
1	1	1	1	1	1	1	3	4
1	1	1	1	1	1	1	1	3
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1

Taulukko 2

Bellman-Ford

1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	3
1	1	1	1	1	1	1	3	4
1	1	1	1	1	1	4	4	5
1	1	1	1	1	1	1	3	4
1	1	1	1	1	1	1	1	3
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1

Taulukko 3

1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	3
1	1	1	1	1	1	1	3	4
1	1	1	1	1	1	2	3	4
1	1	1	1	1	1	1	3	4
1	1	1	1	1	1	1	1	3
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1

Taulukko 4

1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	2
1	1	1	1	1	1	1	2	2
1	1	1	1	1	1	2	2	2
1	1	1	1	1	1	2	2	2
1	1	1	1	1	1	1	1	2
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1

Taulukko 5

1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	2	2
1	1	1	1	1	1	2	2	3
1	1	1	1	1	1	2	2	3
1	1	1	1	1	1	1	2	2
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1

Taulukko 6

1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	2
1	1	1	2	2	3	3	3	4
2	2	2	2	3	4	4	4	4
2	2	3	3	3	4	4	5	4
2	2	2	2	3	4	4	4	4
1	1	1	2	2	3	3	3	4
1	1	1	1	1	1	1	1	2
1	1	1	1	1	1	1	1	1

Taulukko 7