Tietorakenteiden harjoitustyö
Testaus

Jukka Koskelin
013865465

Työn aihe: A* -reitinhakualgoritmia käyttävä ohjelma joka muodostaa kaksiulotteisen kartan ja etsii reitin kahden annetun pisteen välillä.

Puutteet

Toteutuksessa ja lopullisessa testauksessa havaittu seuraavat puutteet:

- Ohjelmassa on mukana vielä yksi Javan tietorakenne, LinkedList johon tallennetaan läpikäydyt noodit. Yritin korvata tämän Keolla, mutta jostain syystä Kekoon alkoi muodostumaan Noodeja, joiden edelliseksi asetettu noodi peri ensimmäiseksi mainitun noodin, eli siis käytännössä silmukka. Oletettavasti ongelma on että joillain kahdella noodin arvoilla ylikirjoitettu hashcode muodostuu identtiseksi, jolloin jokin laskentaluokan getViereinen -metodin vertailuista toimii väärin. Kuitenkin PriorityQueuen !kaydytNoodit.contains(temp) tuntuisi toimivan oikein, joten ongelma paikallistunee Keko-luokan sisaltaakoSaman ja/tai Noodi-luokan hashcode- tai equals -metodeihin. Sen perusteella minkä ehdin testaamaan näytti että pienet kartat toimivat ok, mutta suuremmissa tulee duplikaattinoodeja. Palautettu siis käyttöön versio jossa käydyt noodit sijoitetaan LinkedListiin.
- Kartan raja-arvot. Syötteet joissa loppupiste on kartan äärilaidalla (esim. 50x50 -kartalla 50,50) ilmoittavat ettei annettua koordinaattia löydy kartalta. Ihan hölmö virhe joka käytännössä johtuu siitä että että taulukon indeksi alkaa nollasta ja päättyy 49:ään mikäli kooksi annetaan 50. Fiksumpi olisi tehnyt raja-arvoja testaavan yksikkötestin heti aluksi, tyhmempi unohti. Olisi kohtuullisella työllä korjattavissa mutta 02.30 yöllä jos menee sorkkimaan kaikkia silmukoita, saa todennäköisesti aikaan vain toimimattoman ohjelman.
- AVL-puun toteutus. AVL-puu ei vieläkään hyväksy saman matka-arvion sisältäviä noodeja. Meni aika kahden edellä mainitun ongelma ihmettelyssä ja git:in kanssa tappelemisessa.

Yksikkötestaus

Yksikkötestaus suoritetaan Junit 4 -testeillä ja kattavuus hallitaan JaCoCo -työkalulla (TikiOne JaCoCo -plugin).

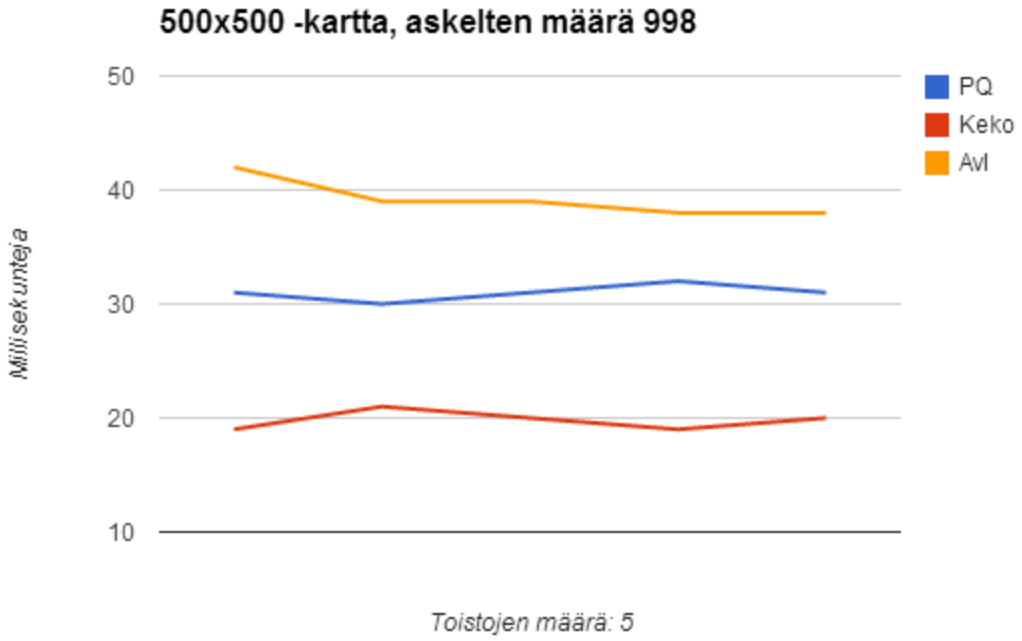
Manuaaliset testitapaukset

Ohjelman keolla tehty versio testattu manuaalisesti seuraavin testitapauksin:

- manuaalinen kartta 50x50, polku 0,0->48,48.
- manuaalinen kartta 50x50, polku -1,0->48,48, nostaa virheen
- manuaalinen kartta 50,50, polku 0,0->50,50, ei toimi odotetusti.
- kartta testi2.png, polku 0,0->39,39, polkua ei löytynyt
- kartta testi2.png, polku 0,0 -> 49,49, polku löytyy, mutta jostain syystä pino-toteutuksella polun piirtäminen kuvaan katkeaa noin puolin väliin polkua. Ei tätä kirjoittaessa järjellistä syytä, mutta aiempi LinkedList-toteutus toimi, joten vian täytyy olla Pinossa.
- kartta testi3.png, polku 1,1 -> 304,217, kuten yllä.

Suorituskykytestaus

Suorituskykytestaus suoritettiin yksikkötesteillä, joissa jokaisessa ajettiin kolme kertaa polun etsintä samanlaista karttaa vasten, samoilla alku- ja loppupisteillä. Testejä oli kaiken kaikkiaan viisi erilaista, joista alle valittiin kaksi - yksi tyhjällä kartalla, ja yksi kuvasta muodostetulla isolla kartalla jossa oli esteitä. Alla tulokset kahdesta eri testikerrasta, molemissa viisi toistokertaa, ajettuna kotitietokoneella jossa käynnissä jonkin verran samanaikaisia ohjelmia.

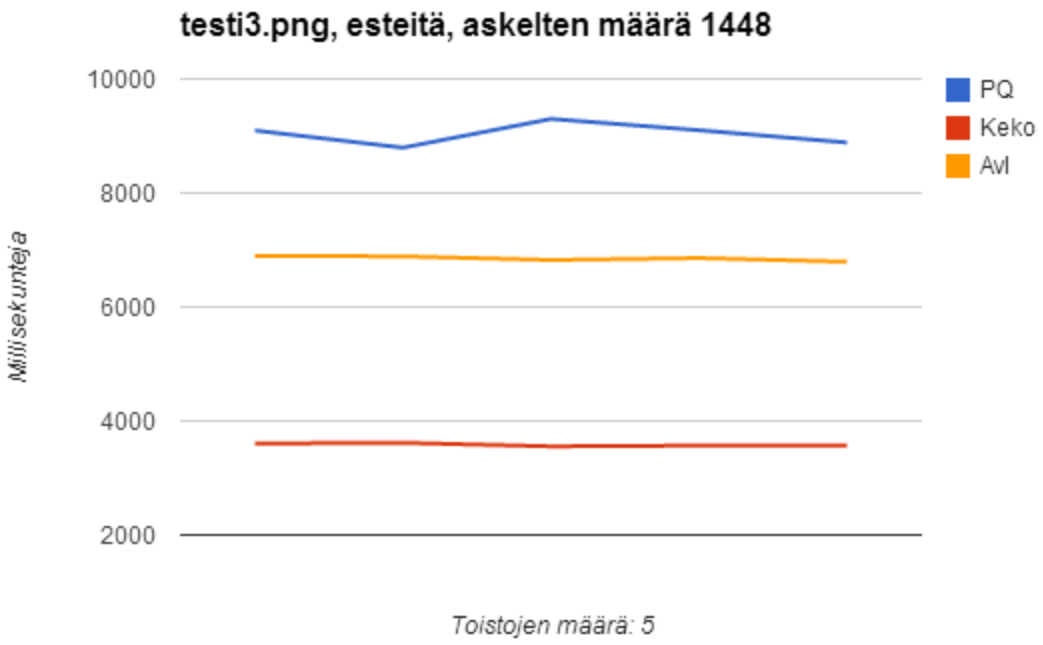


	PQ	Keko	Avl
1	31	19	42
2	30	21	39
3	31	20	39

4	32	19	38
5	31	20	38
KA	31	19.8	39.2

Tyhjän kartan testien tuloksista voidaan nähdä että keko suoriutui kaikista testi-ajoista parhaiten, PriorityQueue toiseksi parhaiten ja hitaimmin toimi Avl-toteutus. Kaikki lienevät kuitenkin samaa nopeusluokkaa.

Tulosten suhteet pysyvät suunnilleen samanlaisina toistokerrasta toiseen.



	PQ	Keko	Avl
1	9096	3605	6898
2	8795	3617	6888
3	9299	3555	6824
4	9101	3569	6858
5	8885	3570	6794
KA	9035.2	3583.2	6852.4

Kun kartta on isompi, mukana on esteitä ja polun määrä suurehko, eroavat tulokset toisistaan enemmän. Keko on edelleen nopein - selkeästi muita nopeampi - mutta tällä kertaa Avl suorituu

PriorityQueue:ta paremmin. Jostain syystä kuvatiedostosta muodostetteva ja esteitä sisältävä kartta toimii tehokkaammin Avl-puulla, mutta täytyy myöntää etten tätä kirjoittaessa ole keksinyt, miksi.

Itse toteutettu keko on selvästi Avl:ää tehokkaampi, joten voitanee olettaa että keko-ehdon toteuttaminen on huomattavasti nopeampaa kuin Avl-puun tasapainottaminen.

Testauspäiväkirja, viikko 1

Ensimmäisellä viikolla on tehty neljä yksikkötestiä joilla testataan reitinhakualgoritmiä sekä niitä julkisia metodeja joita reitinhaun testit eivät kata. Testataan myös kartta -luokan luonti virheellisillä parametreillä.

Yksikkötestien koodikattavuus: Metodit 93%, koko koodi 76%,

Muuta testausta ei ole vielä suoritettu, ohjelma toimii tällä hetkellä vain yksikkötesteillä ajettuna.

Testauspäiväkirja, viikko 2 ja 3

Toisella ja kolmannella viikolla yksikkötestejä järkevöitetty saadun palautteen perusteella, ja keskitytty testaamaan yksittäisiä metodeja (kuitenkin sillä huomiolla että yksikkötestataan ne metodit joita on järkevää testata; usein yhdellä yksikkötestillä on järkevää testata sekä arvojen asetus/nouto -metodit yhdellä kertaa).

Lisäksi eriytetty kommentein ns. integraatioyksikkötestit (kokonaisia ohjelmanosia ja ohjelmalogiikkaa testaavat testit) ja suorituskykytestit. Aiempia integraatiotestejä selkeytetty ja tehty runkototeutusta suorituskykytesteille.

Varsinaista manuaalista testausta ei ole vielä suoritettu, koska ohjelma on edelleen ajettavissa vain yksikkötestien kautta.
Yksikkö- ja integraatiotestien koodikattavuus: Metodit: 97%, koko koodi 87%.

Testauspäiväkirja, viikko 4

Ajanpuutteen vuoksi varsinainen testaus jäi palautusta edeltävälle lauantaille ja sunnuntaille. Ongelmia alkoi kasaantua kun omaa keko-toteutusta kuormitti isommilla (>500) määrillä alkioita. Myös uunituore pino-toteutus oireili kummasti. Aina sitä jälkikäteen oppii/muistaa/huomaa mitä alunperin olisi pitänyt yksikkötestata.