Tietorakenteiden harjoitustyö – Testausdokumentti

Ohjelman testaus on jakaantunut kolmeen osaan: yksikkötesteihin, järjestelmätestaukseen sekä suorituskykytestaukseen.

Yksikkötestaus

Ohjelman toteutuksen yhteydessä on kirjoitettu yksikkötestejä, joilla pyritään varmistumaan siitä, että yksittäisten metodien toiminta vastaa tarkoitettua. Yksikkötestit on toteutettu JUnit-testeinä, joten ne on helppo ajaa aina uudelleen. Yksikkötestit onkin ajettu aina kun ohjelmakoodia on muutettu ja testeissä ilmenneet virheet on korjattu aina kun sellaisia on tullut eteen. Tavoitteena on ollut että Github-repositoriossa oleva ohjelmaversio rakentuu ja läpäisee testit – koska Githubiin on pushattu joka ohjelmointipäivä ainakin kerran, testit on pidetty käytännössä jatkuvasti toimivina.

Yksikkötestien rivi- ja haaraumakattavuutta on seurattu Coberturalla. Tällä hetkellä rivikattavuus on n. [84] % ja haaraumakattavuus [85] %.

Järjestelmätestaus

Ohjelmaa kirjoitettaessa on suoritettu myös järjestelmätestausta. Tässä ei ole käytetty mitään testikehikkoa, vaan järjestelmätestaus on koostunut ohjelman komentojen suorittamisesta manuaalisesti. Tämä on ollut mahdollista, sillä ohjelma tarjoaa vain hyvin rajallisen määrän toiminnallisuuksia. Laajemmalla toiminnallisuusjoukolla ja/tai monimutkaisemmalla käyttöliittymällä järjestelmätestit tulisi automatisoida. Tarvittaessa ohjelman suorituksen etenemistä on lisäksi seurattu ajamalla ohjelma debug-moodissa.

Suorituskykytestaus

Ohjelma on koko ajan vastannut nopeasti käyttäjän syötteisiin. Ohjelman suorituskyvyn kvantitatiiviseksi testaamiseksi on ajettu polunetsintäpyyntöjä erikokoisilla aineistoilla. Aineistot ovat erikokoisia joukkoja Suomen kuntia sijaintikoordinaatteineen (kts taulukko alla).

Tiedosto	Solmuja (=kuntia)	Kaaria (=kuntanaapuruuksia)
suomi83.data	83	207
suomi166.data	166	485
suomi249.data	249	713
suomi332.data	332	978
suomi417.data	417	1226

Ohjelma tukee polunetsintäpyyntöjen ajamista skriptitiedostosta, jolloin saman pyyntöjoukon suorittaminen uudelleen on huomattavan helppoa. Lisäksi skriptitiedostoa ajettaessa annetaan algoritmien toistokertojen määrä: esimerkiksi kymmenellä toistokerralla ohjelma ajaa saman polunetsintäpyynnön

kymmenen kertaa molemmilla algoritmeilla ja ilmoittaa tuloksissa algoritmin suoritusaikoina toistokertojen keskiarvoajat.

Testeissä käytetyt aineistot sekä testiajon skriptitiedosto (test2.scr) on tallennettu projektin datakansioon. Testi on toistettavissa yksinkertaisesti ajamalla ohjelmassa skriptitiedosto halutulla toistokertamäärällä (esimerkiksi komento: `* data/test2.scr 100` ajaa skriptitiedostossa listatut polunetsinnät 100 kertaa).

Testiskriptin mukaiset polunetsintätehtävät ovat alla olevan taulukon mukaiset. A*-algoritmin vahvuus verrattuna Dijkstra-algoritmiin on, että se painottaa polunetsinnässä solmuja, joista se arvioi olevan lyhyempi matka maalisolmuun. Jotta algoritmien väliset erot saataisiin paremmin esiin, lähtösolmuiksi on tarkoituksellisesti valittu kuntia, jotka sijaitsevat suunnilleen aineiston kuntajoukon keskellä ja maalisolmuna taas on laidalla sijaitseva kunta.

Tiedosto	Lähtösolmu	Maalisolmu	Lyhimmän polun pituus (askelia)
suomi83.data	Janakkala	Helsinki	5
suomi166.data	Hämeenlinna	Uusikaupunki	9
suomi249.data	Lahti	Uusikaupunki	13
suomi332.data	Jyväskylä	Kotka	9
suomi417.data	Jyväskylä	Kotka	9

Alla olevassa kuviossa on esitetty ajoajat A*- ja Dijkstra-algoritmeille edellä listatuilla aineistoilla ja etsittävillä poluilla. Testi on ajettu 1000 toistokerralla.