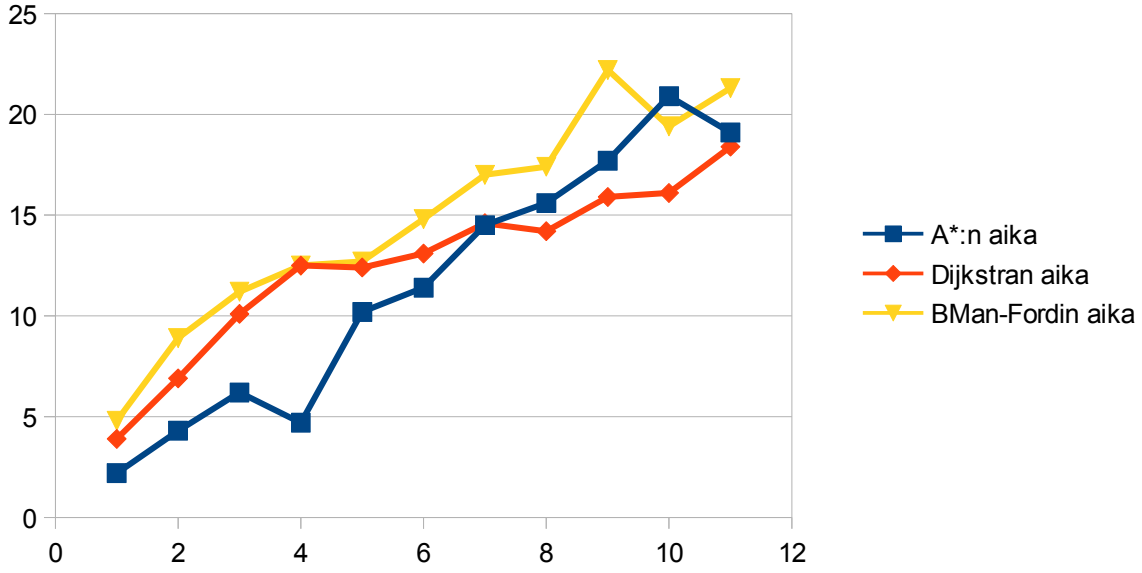
Tietorakenteet ja algoritmit -harjoitustyö

Toteutusdokumentti

Harjoitustyön tarkoituksena oli toteuttaa ja vertailla keskenään kolmea yleisintä polunhakualgoritmia: Dijkstra, Bellman-Ford ja A*.

Seuraavassa kuvassa esitetään algoritmien kuluttamat keskimääräiset ajat millisekunneissa. Algoritmeja testattiin eri kokoisissa yksinkertaisissa verkoissa, joiden sivujen pituudet olivat kooltaan 20-120. Kokeiluja tehtiin jokaiselle koolle yhteensä 10 kertaa. Verkot olivat kaksiulotteisia kokonaislukutaulukoita, joiden alkioiden arvot olivat joko 10 tai 1.



Kuva 1: Käytetyn ajan kokeilutulokset

Dijkstra

Dijkstran toiminta perustuu paras solmu ensin -läpikäyntiin. Ajo aloitetaan aloitussolmusta, jonka kaikki vierussolmut lisätään minimikekoon. Seuraava läpikäytävä solmu päätetään minimikeon avulla. Kekoehtoa ylläpitää solmun paino. Dijkstran aikavaativuuden tulisi olla logaritminen kerrottuna kaarien ja solmujen summalla. ($O((|(E)|+|(V)|)\log|(V)|)$).

Kuvassa 1 on punaisella merkitty Dijkstran algoritmin keskimääräinen ajankulutus. Kuvaaja on melko tasainen ja muistuttaa logaritmifunktion kuvaajaa ollen hitaasti kasvava. Aikavaativuudessa siis onnistuttiin melko hyvin. Toisaalta mitä suuremmaksi verkon koko kasvoi, sitä epätasaisemmin algoritmi toimi.

Tilavaativuuden tulisi olla suoraan verrannollinen verkon kokoon. Toteutuksessa Dijkstran algoritmi varasi muistia aina kaksi kertaa verkon kokoon nähden. Toteutus luo ensin yhden kaksiulotteisen taulukon, jonka alkiot koostuvat olioista, jotka kuvaavat verkon solmuja. Solmuoliot sisältävät tiedot painosta ja polun koordinaateista. Minimikeko sisältää yhtä monta alkiota kuin verkossa on solmuja.

Bellman-Ford

Bellman-Ford on hieman samanlainen kuin Dijkstran algoritmi. Bellman-Ford ei kuitenkaan päätä seuraavaa läpikäytävää solmua keveimmän polun perusteella, vaan käy koko verkon läpi niin monta kertaa, kunnes jännitteitä ei enää ole. Bellman-Fordin tulisi myös tarkistaa jääkö verkkoon negatiivisen painoisia syklejä, joka mahdollistaa algoritmin toiminnan verkossa, jossa kaaret voivat olla negatiivisia. Tässä harjoitustyössä toteutettiin kuitenkin vain verkkoja, joissa kaikki kaaret ovat positiivisia.

Toteutuksessa Bellman-Ford lisää jokaisen solmun vierussolmut linkitettyyn jonoon jos kaaren paino on vaihtunut. Algoritmi käy näin kaikki solmut järjestyksessä läpi. Aikavaativuuden tulisi olla lineaarisesti kasvava ($O(|(V)||(E)|)$). Kokeilun perusteella ajankulutus jäi kuitenkin hieman epäselväksi. Suurimman osan vertailuista Bellman-Ford oli kolmikon hitain.

Tilavaativuuden osalta Bellman-Ford oli kuitenkin kevein. Algoritmi varasi muistia vain verkon solmujen määrän ja jonoon lisättäessä.

A*

A* -algoritmi muistuttaa Dijkstran algoritmia, mutta kaarten painojen lisäksi A* ottaa huomioon arvioidun etäisyyden kohde- ja lähtösolmuun. Aika- ja tilavaativuuksien tulisi olla samat kuin Dijkstrassa. Toteutuksessa A* ei laskenut polkuja kaikkiin mahdollisiin solmuihin vaan kävi verkkoa läpi niin kauan, kunnes saavutti kohdesolmun. Seuraava käsiteltävä solmu määriteltiin minimikeon avulla kuten Dijkstrassa.

Koska toteutuksessa käytettiin verkkona kokonaislukutaulukkoa ei A* aluksi toiminut halutulla tavalla. A* löysi usein painavamman polun kuin muut algoritmit ja saattoi välillä joutua umpikujaan. Ongelma kierrettiin määrittelemällä verkko säännöllisemmäksi, koostuen vain luvuista 1 ja 10. A* ohjelmoitiin pitämään lukua 10 esteenä tai seinänä.

Toteutuksen ajankulutus oli kolmikon epätasaisin. Kuvassa 1 esitettävä käyrä muistuttaa enemmän hintojennousua kaupoissa kuin logaritmifunktiota. A* varasi muistia yhtä paljon kuin Dijkstra, mutta käsitteli paljon vähemmän solmuja kuin muut.

Yhteenveto toteutuksesta

Harjoitustyössä mielestäni parhaiten onnistui Dijkstran toteutus, sillä se oli kolmikosta tasaisimmin toimiva. Heikoin toteutus oli A*, sillä sen aikavaativuus ei ollut lähelläkään saavutettavaa. A* kuitenkin oli suurimman osan kokeiluista kaikkein nopein. Bellman-Ford käsitteli eniten solmuja, eikä oikean Bellman-Fordin tavoin tarkistanut negatiivisia syklejä, joten tämänkään algoritmin toteutus ei ollut parhaimmasta päästä.

Ohjelmassa on myös toimintoja, joita ei voi suoraan jar-tiedostoa ajamalla käyttää, kuten Verkkogeneraattori-luokan metodi, joka luo verkon, jossa on risti keskellä. Satunnaisen verkon luomisen olisi voinut tehdä muutenkin hienostuneemmin, sillä tällä hetkellä verkko luodaan vain ensin täyttämällä kokonaislukutaulukko luvuilla 10, jonka jälkeen kohdesolmun indeksien rivit ja sarakkeet täytetään luvulla 1. Tämän jälkeen aloitussolmusta edetään satunnaiseen vierusalkioon, kunnes saavutetaan alkio, joka sijaitsee joko samalla rivillä tai sarakkeella kuin kohdesolmu.

Kristian Hansson