Toteutusdokumentti

Tietorakenteiden harjoitustyö

Ohjelman yleisrakenne

Ohjelmassa on toteutettu algoritmit Dijkstra, Bellman-Ford ja A*, sekä niiden tarvitsemat aputietorakenteet minimikeko ja linkitetty lista. Ohjelman ajamiseen käytetään Mainin lisäksi AlgoritmienVertailu-luokkaa, jossa on määritelty toiminnot, jotka vastaanottavat syötteitä käyttäjältä. Syötteillä määritellään minkä algoritmin toimintaa halutaan testata, ja minkä kokoisella labyrintillä.

Saavutetut aika- ja tilavaativuudet (V solmujen määrä ja E kaarien joukko)

Dijkstra:

- Alustuksessa käydään läpi kaikki solmut joten aikaa menee O(|V|).
- Keko-operaatioiden vaativuus on O(log n) jos keossa on n alkiota. Kekoon lisätään alkioita |V| kertaa, siis aikaa menee O(|V| log |V|). Samoin keosta poistetaan alkioita |V| kertaa.
- Koska jokainen solmu lisätään kekoon vain kerran, viereiset solmut käydään läpi vain kerran. Siis jokaista kaarta tutkitaan relax-metodissa kerran, jolloin vaativuudeltaan O(log |V|) olevaa decKey-operaatiota kutsutaan enintään |E| kertaa.
- Aikaa kuluu siis läpikäyntiin yhteensä O((|E| + |V|) log |V|).
- Tilavaativuus on O(|V|), koska alussa kaikki solmut ovat keossa ja sen jälkeen keko alkaa pienentyä.

A*:

- Toimii Dijkstraa vastaavasti, eli aikavaativuus on sama, sillä (lisänä Dijkstraan) käytetty distLoppuun-arvo voidaan laskea vakioajassa käyttämällä taulukon kokoa.
- Myös tilavaativuus on sama kuin Dijkstrassa.

Bellman-Ford:

- Alustuksessa käydään läpi kaikki solmut joten aikaa menee O(|V|).
- haeKaaret-metodissa käydään uudelleen solmut läpi joten aikaa menee toiset O(|V|).
- Aikavaativuus riippuu tutkittujen kaarien määrästä/niille tehtyjen relax-operaatioiden määrästä. Algoritmi käy kaaret läpi |V|-1 kertaa ja kutsuu relaxia. Lopuksi se käy kaaret läpi vielä uudelleen etsien negatiivisia syklejä.
- Aikavaativuus on siis O(|V||E|). Vaikka haeKaaret vie aikaa yhden ylimääräisen O(|V|), niin aikavaativuus ei muutu.

- Tilavaativuus on O(|E|), joka on kaaret tallettavat listan koko.

Suorituskyky- ja O-analyysivertailu

Bellman-Ford häviää selvästi muille algoritmeille tehokkuudessa. Aikavaativuus näyttäisi kuitenkin vertautuvan muiden kanssa oikean suuntaisesti, eli suhteessa muihin Bellman-Ford toimii mielestäni melko odotetusti; aikavaativuuksilla O(|V||E|) ja O((|E| + |V|) log |V|) on kuitenkin todellisuudessa melko suuri ero.

Bellman-Fordin käyttämä aika luultavasti pienenisi huomattavasti, jos käyttäisi suunnattuja verkkoja, jolloin riittäisi käydä kaaret läpi vain kerran per kaari, kun suuntaamattomassa verkossa ne on käytävä läpi aina molempiin suuntiin.

A* ja Dijkstra tuntuvat toimivan melko lailla samassa ajassa, riippuen jonkin verran labyrintistä ja sen muodosta sekä esteiden ja lähtö- ja maalipisteiden sijoittelusta. Dijkstra toimii jonkin verran paremmin tilanteissa, joissa lyhimpiä polkuja löytyy useita. A* taas toimii paremmin silloin, jos on selkeämmin löydettävissä vain yksi lyhin polku, eli esim. jos lähtö- ja maalipiste sijaitsevat labyrintin samalla laidalla.

Puutteet ja parannusehdotukset

- Bellman-Fordin aikavaativuutta haittaa se, että kaarien etsinnässä verkko on käytävä läp yhden “ylimääräisen” kerran. Kaaria ei kuitenkaan tässä tapauksessa saanut luotua, ennen kuin kaikki pisteet oli luotu. Jokin ratkaisu tähän saattaisi olla olemassa.
- Dijkstran ja A*:in minimikeot olisi voinut yhdistää käyttämällä jonknlaista kekoalkio-luokkaa, jolla on attribuutteina objekti ja arvo. Yritin toteuttaa tätä, mutta törmäsin ongelmaan siinä kohtaa, kun olisi pitänyt päivittää pisteiden etäisyyksiä ja kutsua keon decKey-operaatiota; eli miten olisi järkevästi saanut päivitettyä sekä itse pisteen että kekoalkion arvoa samalla kertaa.
- Työhön olisi saanut laajuutta tekemällä sen toimimaan niin, että labyrintissa voisi kulkea myös kulmasta kulmaan, sekä esimerkiksi mahdollistamalla painotetut verkot. Näihin ei kuitenkaan jäänyt lopulta aikaa loppuvaiheessa kohdattujen muiden ongelmien vuoksi.