Algoritmit ja tietorakenteet

Tässä harjoitustyössä toteutetaan A* ja Jump Point Search reitinhakualgoritmit. Näiden sivutuotteena syntyy myös Dijkstran algoritmi, joka on käytännössä A* ilman yksittäisille solmuille laskettavaa "matka määränpäähän" -heuristiikkaa.Toteutettavat tietorakenteet ovat verkko, keko, pino ja lista, joita A* ja Dijkstra toteutukset tarvitsevat.

Verkko kuvataan karttamatriisina - perinteistä vieruslista tai vierusmatriisiesitystä ei siis toteuteta. Verkon solmun naapurit ovat kartalla vierekkäin sijaitsevat solmut ja kartalla edetään ilmansuuntien mukaisesti. Ns. väli-ilmansuuntia (ylä- ja alaviistoon mentäessä löytyvät solmut) ei huomioida ainakaan projektin alkuvaiheessa. Verkon solmut kuvataan 'Vertex' olioilla, joihin on helppo tallentaa erilaista dataa algoritmin etenemisestä visualisointia ja debuggausta varten.

Kekona käytetään aluksi Javan PriorityQueue rakennetta, jotta ohjelman runko saadaan nopeasti toimimaan. Samoin toimitaan myös pinon (ArrayDeque) ja listan (ArrayList) kanssa.

Ongelman ratkaisu sekä tietorakenteiden ja algoritmien valinnan perusteluja

Tämä harjoitustyö käsittelee lyhyimmäin reitin löytämistä 2D matriisina esitetyltä kartalta, jossa on saman painoisia ruutuja ja läpäisemättömiä ruutuja (vrt. labyrintti). Kyseessä on tärkeä ongelma useissa tietojenkäsittelytieteen ongelmissa, kuten vaikkapa pelien tekoälyissä. Työn edetessä tutkitaan myös mahdollisuutta lisätä karttaan eri painoisia ruutuja, jolloin ongelma muistuttaa enemmän oikean maailman reititysalgoritmien käyttöä - A* tukee tätä valmiiksi, mutta JPS kanssa toteutus ei välttämättä ole täysin selvää.

Koska ongelman kartta ilmaistaan painottamattomana verkkona, olisi naiivi lähestymistapa suorittaa verkon leveyssuuntainen läpikäynti (BFS). Tämä on järkevämpi ratkaisu kuin samaan lopputulokseen päätyvät Dijkstran ja Bellman-Fordin algoritmit (kunhan kaarien painot merkataan yhtäsuuriksi), koska BFS on hyvin helppo toteuttaa. BFS algoritmia en aio toteuttaa, koska Dijkstra toimii lähes identtisesti sen kanssa painottamattomissa verkoissa: jonon sijaan solmut otetaankin keosta, mutta läpikäynti on kuitenkin aina yksi etäisyys kerrallaan eli ensin aloitussolmu jonka etäisyys on 0, sitten kaikki solmut joiden etäisyys=1, sitten 2 jne.

Valitut A* ja JPS algoritmit ovat huomattavasti tehokkaampia kuin BFS/Dijkstra, joten näiden vertailu naiiviin ratkaisuun ja myös keskenään on mielekästä. Lisäksi, jättämällä BFS pois säilytetään mahdollisuus lisätä eri painoisia kaaria verkkoon. Oletettavasti A* on huomattavasti tehokkaampi kuin BFS/Dijkstra ja JPS taas huomattavasti tehokkaampi kuin A*.

Ohjelman syötteet ja käyttö

Ohjelma käyttää Nathan Sturtevantin tarjoamia tutkimukseen tarkoitettuja open source karttoja jotka ovat .map tiedostoissa projektin /maps kansiossa (ohjelma lataa ne sieltä käynnistyksen yhteydessä).

Ohjelmaa käytetään Netbeans ympäristössä.

Tavoitteena olevat aika- ja tilavaativuudet (m.m. O-analyysi)

Aika ja tilavaativuudet:

Aikavaativuudet:
1. Dijkstra: O((|E| + |V|) log |V|) - kuin BFS, heapify ei tarvita, keko voidaan luoda ajassa O(|V|)
2. A*: O((|E| + |V|) log |V|) - A* aikavaatimus on sama kuin Dijkstralla, vaikka käytännössä nopeampi
3. JPS: ?

Tilavaativuudet:
1. Dijkstra: O(|V|)
2. A*: O(|V|)
3. JPS: ?

Tavoitteena on toteuttaa JPS, A* ja Dijkstra yhtä tehokkaasti kuin alan kirjallisuudessa.

Lähteet

JPS Blogipostaus ja linkki alkuperäiseen artikkeliin
http://harablog.wordpress.com/2011/09/07/jump-point-search/
http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/viewFile/3761/4007

JPS http://zerowidth.com/2013/05/05/jump-point-search-explained.html

A* heuristiikat http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html

PathFinding.js - inspiraation lähde sekä GUI että lähdekoodin osalta
http://qiao.github.io/PathFinding.js/visual/

Kartat http://www.movingai.com/benchmarks/