Määrittelydokumentti - Labyrintti

Tietorakenteiden harjoitustyöni aiheena on labyrintti. Ohjelman tarkoituksena on löytää lyhyin reitti labyrintin läpi lähtösolmusta a maalisolmuun b.

Toteutettavat algoritmit ja tietorakenteet

Lähden toteuttamaan lyhyimmän reitin etsimistä labyrintissä A*-algoritmilla. A*-algoritmi on kuin Dijkstran algoritmi, mutta se hyödyntää heuristiikkafunktiota (esimerkiksi euklidista etäisyyttä tai jotakin muuta etäisyysarviota). Algoritmi löytää optimaalisen reitin, jos heuristiikka ei koskaan yliarvioi reitin pituutta lähtösolmusta a maalisolmuun b. Yleensä algoritmi toimii myös melko tehokkaasti, eli ei käy kovin suurta määrää ylimääräisiä solmuja läpi. Tästä syystä valitsin A* -algoritmin omassa harjoitustyössäni lyhyimmän reitin etsimisen lähtökohdaksi. A* -algoritmia on myös käyty hieman läpi Tietorakenteet ja algoritmit sekä Johdatus tekoälyyn -kursseilla, joten ajatus sen taustalla on jokseenkin tuttu entuudestaan. [1][2]

A*-haussa tarvittavan prioriteettijonon toteutuksessa aion hyödyntää itse rakennettua minimikeko-tietorakennetta. Minimikeko on melko kevyt toteuttaa ja sillä saa toteutettua nimenomaan prioriteettijonon tehokkaasti (erityisesti ”heap-insert” ja ”heap-del-min” -operaatioiden avulla). [1]

Ohjelman syötteet ja miten niitä käytetään

Ohjelma (hakualgoritmi) saa syötteekseen labyrintin, joka on esimerkiksi kokonaislukuja sisältävä matriisi. Labyrintti sisältää solmuja, joista yksi on lähtö- ja yksi maalisolmu. Labyrintissä on solmuja, joihin voi päästä, sekä ’estesolmuja’, joihin ei voi kulkea. Solmu-olioissa pidetään kirjaa etäisyydestä lähtösolmuun ja maalisolmuun, sekä

Tavoiteltu aikavaativuus

A*-algoritmin aikavaativuus riippuu heuristiikasta - pahimmillaan vierailtujen solmujen määrä on eksponentiaalinen lyhimmän polun pituuteen nähden, ja parhaimmillaan polynominen [3].

Mahdolliset laajennukset harjoitustyöhön

Jos aikaa jää, pyrin laajentamaan harjoitustyötäni A* -algoritmin ja oman kekototeutuksen lisäksi. Mahdollisia laajennuksia ovat esimerkiksi A* -algoritmin tehokkuuden vertaaminen toiseen etsintäalgoritmiin, esim. *Jump Point Search* -algoritmiin. Tähän laajennusvaihtoehtoon liittyy algoritmien vertailu keskenään sekä etsinnän etenemisen visualisointi.

Toinen mahdollinen laajennus olisi vaihtoehtoinen toteutus prioriteettijonolle, eli esimerkiksi keon korvaaminen AVL-puulla.

Lähteet

[1] Tietorakenteet ja algoritmit -kurssin materiaali: http://www.cs.helsinki.fi/u/floreen/tira2013syksy/tira.pdf

[2] Johdatus tekoälyyn -kurssin materiaali: https://www.cs.helsinki.fi/webfm_send/1245

[3] Wikipedia: http://en.wikipedia.org/wiki/A*_search_algorithm