Määrittelydokumentti

Työssä etsitään lyhyin reitti lähtöpisteestä maalipisteeseen itse piirretyn labyrintin läpi. Reitin etsimisessä käytetään kolmea eri algoritmia: A*, Bellman-Ford ja Djikstra. Aputietorakenteena käytetään minimikekoa, sillä seuraavaksi käytävä naapuri solmu saadaan vakioajassa. Uuden solmun lisääminen kekoon ja solmun poistaminen keossa tapahtuu arviolta O(log n).

Tavoitteena on myös havainnollistaa algoritmin kulkua labyrintissa.

Ohjelmalle sopiva syöte on esitetty käyttöohjeessa.

Algoritmien kohdalla pyritään optimaaliseen tehokkuuteen eli Djikstra ja A* O(|E| + |V|log|V|) (A* pois luettuna heuristiikka), Bellman-Ford O(|V|*|E|) ja verrokkina on A*, missä on heuristiikkana suora etäisyys maalipisteeseen esteistä piittaamatta (linnuntie) ja jo kuljettu matka.

Toteutusdokumentti

Ohjelman yleisrakenne koki monesti suuria muutoksia, kunnes saavutti nykyisen pisteensä. Suurta refaktorointia aiheutti 10 rivin metodipituus. Punainen viiva on algoritmin löytämä lyhyin reitti ja keltainen alue on algoritmin tarkastamaa aluetta.

n = kaikkien solmujen määrä (voi olla teoriassa siis koko kuvan leveys * korkeus)

e = kaikkien kaarten määrä (solmulla on 8 naapuria)

w = kuvan leveys

h = kuvan korkeus

Minimikeon aikavaativuudet: Kekoon lisääminen O(log n), Keosta poistaminen O(log n). Aikavaativuudet ovat seurausta binääripuumallistaan.

Minimikeon tilavaativuus: Keossa käytettävän taulun suuruus on kuvan pituus x leveys, minkä voi arvioida olevan O(n)

Kuvan lukemisen ja piirtämisen aikavaativuus on O(h*w) = O(n). Tilavaativuus on myöskin O(n).

Algoritmeissa ei ole otettu huomioon kuvan lukemiseen ja piirtämiseen kuluvaa aikaa/tilaa.

Djikstran aikavaativuus: O((n+e)*log(n)). Pahimmillaan voidaan joutua käymään läpi kaikki solmut ja seurauksena poistaa n kertaa solmua keosta (n*log(n)). Voidaan joutua myös relaksoimaan kaikki kaaret, jolloin joudutaan myös lisäämään kaarten määrien verran solmuja kekoon (e * log(n)).

Djikstran tilavaativuus: O(n)

Bellman-Fordin aikavaativuus: O(n*e). Pahimmassa tapauksessa voidaan joutua relaksoimaan eli päivittämään jokaisen solmun kohdalla kaikki naapurit.

Bellman-Fordin tilavaativuus: O(n)

A* aikavaativuus: O((n+e)*log(n)). Sama perustelu kuin Djikstrassa, heuristiikka vaan on eri.

A* tilavaativuus: O(n)

Djikstra on odotetusti nopein algoritmeista ja Astar on hitain "harhaoppisen" heuristiikkansa vuoksi. A*-on yhtä nopea kuin Djikstra, jos labyrintin muoto suosii A*:n heuristiikkaa. Todennäköisintä on, että heuristiikkansa takia se eksyy yrittämään umpikujia helposti. Mutta heuristiikkansa ansiosta, sen tutkittujen solmujen määrä Djikstraan ja Bellman-Fordiin verrattuna on selkeästi pienempi. Bellman-Ford häviää ajassa, koska se käy |V| - 1 kertaa solmut läpi. Djikstra on siis valituista algoritmeista tehokkain.

Astar relaksoi vähiten solmujen välisiä matkoja oikeaan suuntaan osoittavan heuristiikan avulla, kun taas Djikstra sekä Bellman-Ford relaksoivat enemmän. Eron suuruus riippuu käytetystä labyrintista, mutta Djikstra käy läpi vähemmän solmuja kuin Bellman-Ford.Djikstra ja Bellman-Ford löytävät edullisimman painoisen reitin maaliin verrattuna Astariin. Labyrintista riippuen erot vaihtelevat. Keskimäärin ne löytävät kuitenkin lähes samanpainoisen reitin maaliin. Djikstra ja Bellman-Ford löytävät saman painoisen reitin eri reittipituuksilla, mikä johtuu solmuille annetuista painotuseroista. On edullisempaa kulkea suoraan sivuille (paino 1) kuin kulmittain (paino 2).

Reitit löytyvät eri prioriteeteilla riippuen mitä algoritmia käytetään. Astar käy läpi vähiten solmuja, minkä takia se ei löydä yhtä edullista reittiä kuin Bellman-Ford tai Djikstra.

Liitteeksi tiralabraTestausLiite.pdf

Testausraportti

Testaus on hieman hankalaa labyrinttien monimuotoisuuden vuoksi. Tämän takia testikuvioita ajettiin 5 kertaa läpi sekä eri kokoja ja kuvioita. (Kts. testausliite) Labyrinttien koon skaalauksella ei, odotetusti, havaittu olevan eroa algoritmien välisissä suhteissa. Tämän takia alkupään skaalauskokeilujen jälkeen, sitä ei enää harrastettu.Osa metodien testauksista suoritetaa JUnit-testien avulla, mutta kaikkia metodeja varten ko. tapa ei ollut järkevää. Loput metodeista on testattu lukuisten ajojen avulla sekä tulosteita tarkastelemalla. Kuten esim. kuvien lukeminen ja piirtäminen sekä aloitus- ja maalipisteiden olemassaolojen tarkistus. Osa JUnit-testeistä testaa myös useampia metodeja kerrallaan. Testausta suoritettiin siis tulosteiden avulla.

Parannusehdotukset ja puutteet

Testausta voisi aina parantaa sekä JUnittien riippumaattomuutta testikuvista että vakioista. Ne ovat vaan valitettavasti helpoin lähestymistapa. Koen, keon toimintaa voisi optimoida että algoritmien tilavaativuuksia. A*-heuristiikkaa ei voi oikeastaan luokitella puutteeksi vaan ominaisuudeksi (todellinen iA*).

Lähteet

Tietorakenteet luentomateriaali, kevät 2012.

Wikipedia soveltuvin osin.