Toteutusdokumentti

Ohjelma on hajautettu eri paketteihin luokkien toimenkuvan mukaan: tietorakenteet, algoritmit ja käyttöliittymä ovat erillään. Käytännössä Main-luokka starttaa käyttöliittymän, joka käynnistää halutun algoritmin.

Aika- ja tilavaatimukset

Molemmat algortimit käyttävät hyväkseen pinoa, kekoa ja apumatriisia, johon verkon solmut tallennetaan. Pino vie tilaa korkeintaan löydetyn reitin pituuden verran, apumatriisi sekä keko sokkelon koon verran (leveys*korkeus). Keossa ei tule kuitenkaan olemaan missään vaiheessa kaikkia solmuja samaan aikaan. Tilavaatimus on siis O(reitin pituus + 2*(labyrintin leveys*korkeus) → O(n), missä n on labyrintin koko. Käyttöliittymä käyttää hyväkseen vielä yhtä apumatriisia reitin piirtämiseksi näkyviin, mutta myös sen tilavaatimus on O(n), eikä se muuta kokonaisvaatimusta. Koordinaatti-tietorakenteeseen tallennetaan parhaillaan 8 eri arvoa ja näitä luodaan labyrintin koon verran. Näiden yhteisvaatimus on siis O(8n), jolloin koko ohjelman tilavaatimus pysyy edelleen lineaarisena.

Reitinhaussa käytettyjen keko-operaatioiden heapify, removeMin ja insert suoritusajat riippuvat keon puun korkeudesta, jolloin näiden aikavaatimukset ovat O(log(n)). Pinon push- ja pop -metodit toimivat vakioajassa ja taulukon kasvattamiseen menee aikaa pinon senhetkisen pituuden n verran: O(n).

Reitinhakualgoritmien aputaulukoiden luomiseen ja koordinaattien luomiseen menee aikaa O(|V|). Reitin tallentamiseen pinoon sekä tulostamiseen menee molempiin aikaa reitin pituuden n verran (n kertaa pinon pop- ja push -metodit): O(n) .

Dijkstrassa jokaisella Relax-komennolla tallennetaan kekoon korkeintaan neljä muuta solmua. Solmun löysääminen tehdään kuitenkin jokaiselle verkon solmulle, jolloin kekoonkin lisätään kaikki solmut. Tämän aikavaatimus on siis O(|V|*log|V|). Dijkstran kokonaisaikavaatimus on siis O(n + n*log(n)), missä n on verkon solmujen lukumäärä.

A*-algoritmin aikavaatimus on pahimmassa tapauksessa sama kuin Dijkstran. Parhaimmillaan se voi kuitenkin löytää reitin ajassa O(|V| + 4*n) → O(|V| + n) , missä n on lyhin reitti maali- ja lähtöpisteiden välillä.

Löydetyn k-pituisen reitin tallentaminen ja piirtäminen n-kokoiseen labyrinttiin vie aikaa O(k+n). Siispä koko ohjelman tilavaatimus on O(n) ja aikavaatimus O(n + n*log(n)), missä n on verkon solmujen lukumäärä (=labyrintin koko). Astar toimii usein tätä nopeammin. Sadallatuhannella satunnaisella reitinhaulla Astar on joitain sekunnin kymmenyksiä Dijsktraa nopeampi. Suuremmilla verkoilla nopeusero voi hyvinkin kasvaa.

Puutteet ja parannusehdotukset

Alkuperäinen suunnitelma oli implementoida liikkuva maalipiste ohjelmaan. Ajanpuutteen vuoksi tätä ei kuitenkaan ehditty toteuttaa. Graafinen käyttöliittymä auttaisi visualisoimaan käytetyn labyrintin paremmin.

Verkon kulmikkaan rakenteen takia olisi voinut olla nopeampaa suorittaa reitinhakua vain risteyskohdissa ja esteisiin törmätessä: Nyt aikaa menee hukkaan tilanteissa, joissa on vain yksi mahdollinen etenemissuunta. Sokkelo on myös hieman yksinkertainen sekä hyvin pieni: Algoritmien suorituskykyerot tulisivat paremmin esille suuremmissa, mahdollisesti satunnaisissa sokkeloissa.