Reitinhaku verkossa

Toteutus

Harjoitustyön toteutuksessa on käytetty Dijkstran algoritmia ja A*:ä. Kuten määrittelydokumentissa on todettu, Dijkstran algoritmin aikavaativuus on $O((|V|+|E|) \log|V|)$ ja tilavaativuus $O(|V|)$, missä $|V|$ on verkon G solmujen ja $|E|$ kaarien lukumäärä. Nämä vaativuusluokat on johdettu monisteen[1] sivulla 526. Harjoitustyön toteutuksessa päästään samoihin luokkiin, minkä perustelemiseksi tarkastelemme seuraavaksi rinnakkain monisteen pseudokoodia ja harjoitustyön lähdekoodia.

Monisteen sivulla 522 esitetty algoritmi Dijkstra-With-Heap alkaa operaatioilla Initialize-Single-Source ja S = ∅, jotka vievät aikaa $O(|V|)$. Monisteen Initialize-Single-Source:ssa asetetaan distance[v] = 0 ja path[v] = NIL kaikilla $v \in V \setminus \{s\}$. Lähdekoodissa tämä on toteutettu suoraviivaisesti sillä erolla, että lisäksi kutsutaan vakioaikaista operaatiota locationToNode.

Seuraavaksi monisteen pseudokoodissa lisätään kaikki solmut kekoon. Lähdekoodissa tämä tehdään operaatiolla heapInsertAll, jossa käydään läpi kaikki kartan ruudut. Kutakin ruutua vastaa yksi verkon solmuista, joten insert-operaatiota toistetaan yhteensä $|V|$ kertaa. Insert on vaativuudeltaan $O(\log|V|)$: pahimmassa tapauksessa uusi solmu on kuljetettava binääripuun lehdestä juureen asti, ja lähes täydellisen binääripuun korkeus on tunnetusti logaritminen puussa olevien solmujen määrään nähden. Siispä heapInsertAll on vaativuudeltaan $O(|V| \log|V|)$.

Seuraavana vuorossa olevaa while-silmukkaa toistetaan $|V|$ kertaa. Joka kierroksella kutsutaan $O(\log|V|)$ -aikaista operaatiota heap-del-min. Relax- ja heap-dec-key-operaatiot tehdään enintään $|E|$ kertaa. Relax toimii ajassa $O(1)$ ja heap-dec-key ajassa $O(\log|V|)$, joten while-silmukan aikavaativuus on $|V|\ O(\log|V|) + |E|\ O(\log|V|) = O((|V|+|E|) \log|V|)$.

A*:n toiminta on kuvattu monisteen sivuilla 597 – 616. Kuten sivulla 597 todetaan, kyseessä on tavallaan Dijkstran algoritmin laajennos. Tämä näkyy toteutuksessa siten, että molemmat algoritmit käyttävät samaa Searcher-luokkaa. Ero tulee esiin heuristiikkafunktiossa, joka Dijkstran algoritmin tapauksessa palauttaa aina arvon 0. Koska heuristiikkafunktion arvo voidaan laskea vakioajassa, on A*:n aikavaativuus sama kuin Dijkstran algoritmilla.

Vertaillaan seuraavaksi Dijkstran algoritmin ja A*:n suorituskykyä esimerkkitapauksissa. Kuviossa 1 on esitetty tiedostossa map3.txt määritelty kartta (vasemmalla) ja algoritmien löytämä lyhin reitti (oikealla). Aloitussijaintina on (3, 4) ja lopetussijaintina (7, 12).

Molemmat algoritmit päätyivät samaan reittiin, mutta huomionarvoista on etsinnän tehokkuus. Dijkstran algoritmi tutki etsinnän aikana 43 solmua, kun taas A*:lle riitti 12. Ero selittyy sillä, että A* suosii heuristiikkafunktion ansiosta sellaisia hakusuuntia, jotka johtavat mahdollisimman suoraviivaisesti kohti maalia.

---

1 Floréen, P. 2013: Tietorakenteet ja algoritmit. Helsingin yliopiston Tietojenkäsittelytieteen laitos.

```
1 1 0 1 0 1 0 1 0 1 0 0 1      1 1 0 1 0 1 0 1 0 1 0 0 1
0 1 0 1 1 1 1 1 0 1 1 1 1      0 1 0 1 1 1 1 1 0 1 1 1 1
0 1 0 0 1 0 0 1 1 1 0 1 0      0 1 0 0 1 0 0 1 1 1 0 1 0
0 1 1 1 9 1 1 0 0 0 0 1 0      0 1 1 1 * * * 0 0 0 0 1 0
0 1 0 0 0 0 1 1 1 1 0 0 0      0 1 0 0 0 0 * * * * 0 0 0
0 1 1 1 0 0 0 0 0 1 1 1 1      0 1 1 1 0 0 0 0 0 * * * *
0 1 0 0 0 0 0 0 0 0 0 0 1      0 1 0 0 0 0 0 0 0 0 0 0 *
0 1 1 1 0 0 0 0 0 0 0 0 1      0 1 1 1 0 0 0 0 0 0 0 0 *
```

Kuvio 1: Dijkstran algoritmin ja A*:n löytämä reitti tiedoston map3.txt kartalla

Toisessa esimerkkitapauksessa syötteenä oli tiedostossa map4.txt määritelty, 120 x 120 ruudun kokoinen kartta. Aloitussijaintina oli (0, 0) ja lopetussijaintina (29, 29). Dijkstran algoritmi tutki etsinnän aikana 1281 solmua, A* puolestaan 1017 solmua. Kun aloitussijainniksi annettiin (0, 29) ja lopetussijainniksi (29, 0), vastaavat luvut olivat 2485 ja 1568.

Voidaan siis todeta, että esimerkkitapauksissa A* suoriutui lyhimpien polkujen etsimisestä Dijkstran algoritmia tehokkaammin. Erityisesti kuviossa 1 esitetyssä tapauksessa ero oli huomattava, mikä olikin perusteena tämän tyyppisen kartan käyttämiseen vertailussa. Jos tarkoituksena on, kuten tässä harjoitustyössä, toteuttaa molemmat algoritmit, lienee yksinkertaisinta toteuttaa saman tien A*. Tätä on sitten mahdollista käyttää Dijkstran algoritmin toteutukseen ohittamalla heuristiikkafunktio kokonaan tai asettamalla se palauttamaan vakioarvo 0.