Toteutusdokumentti

Ohjelman yleisrakenne

Käyttöliittymä määrittelee ohjelman graafisen ilmeen ja mahdollistaa käyttäjän inputtien antamisen ohjelmalle. Käyttöliittymä antaa myös mahdollisuuden vaihtaa käytettävää algoritmia verkon reitin hakuun.

GridHandler toteuttaa käyttäjän näkemän ruudukon nappien logiikan ja mahdollistaa käyttäjälle sen, että tämä voi muuttaa esteiden sijaintia verkossa.

ButtonHandler määrittelee “find route”-napin logiikan. Tämän avulla ohjelma saa tiedon hakea reittiä verkossa käyttäjän haluamalla hetkellä.

Verkko kuvaa nimensä mukaan verkkoa, missä reittiä haetaan ja se hyödyntää solmujen (Vertex) vieruslistaa Arrayn avulla.

Vertex kuvaa verkon solmuja ja jokaisella solmulla on paino aloitussolmuun s, mitä hyödynnetään reitin haussa.

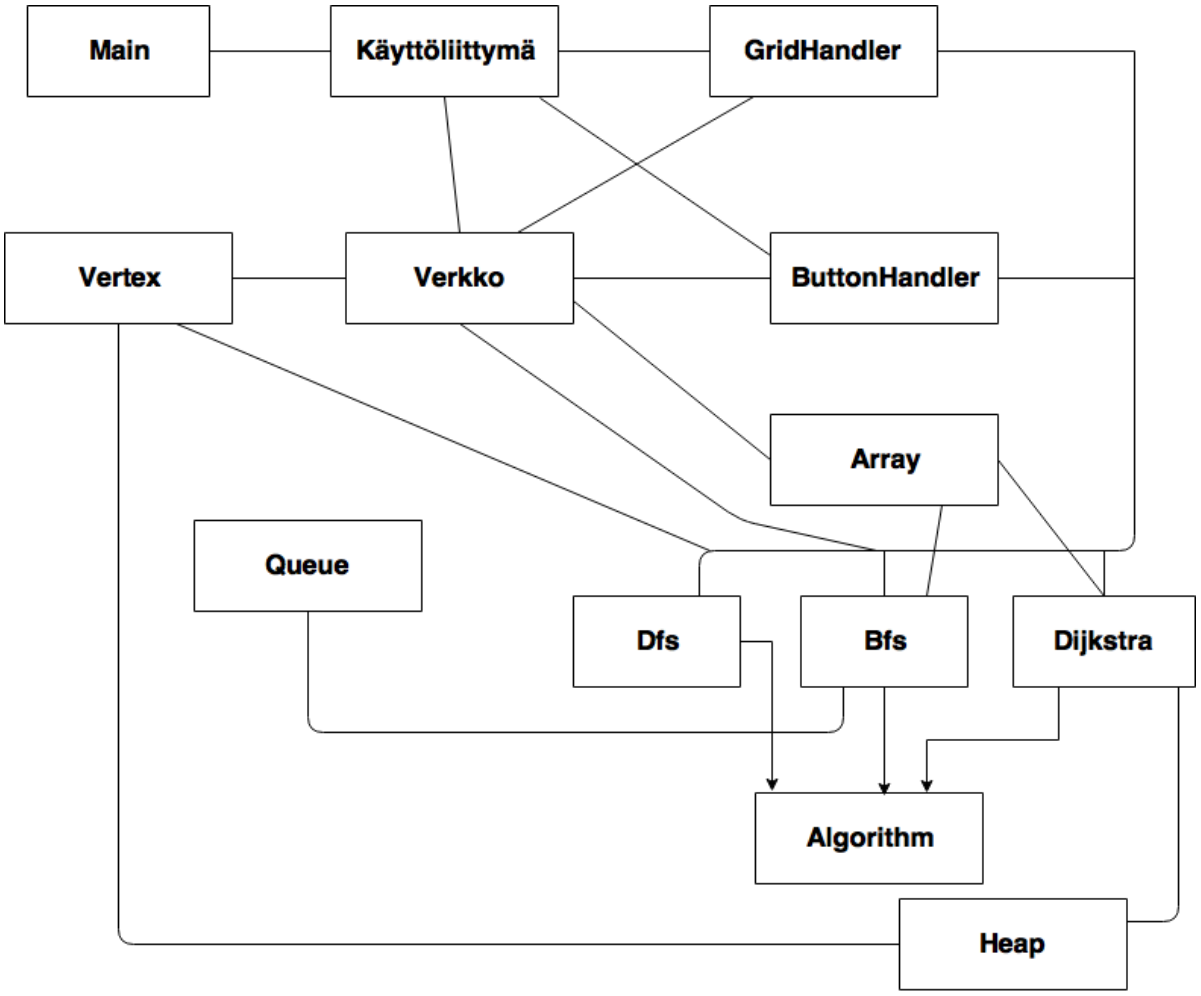
Array toteuttaa geneerisen listan tietorakenteen, johon voidaan tallettaa erilaisia olioita tarpeen mukaan. Tätä hyödynnetään vieruslistoja toteuttaessa verkossa, jota hyödyntävät myös algoritmit.

Algorithm on yleinen rajapinta erilaisille algoritmeille, jolloin reitin haku on helppo toteuttaa samalla metodilla useammalle eri algoritmille.

Dfs/Bfs/Dijkstra ovat omien nimensä mukaisen algoritmin toteutuksia ja hakevat reitin verkossa omalla säännöllään. Nämä perivät Algorithm-luokan ja toteuttavat sen määrittämän findRoute()-metodin.

Heap toteuttaa solmujen pinon tietorakenteen, mitä Dijkstra hyödyntää algoritmissaan.

Queue määrittelee geneerisen jonon tietorakenteen. Tätä hyödynnetään Bfs-algoritmissa.



(Yllä kuvattu ohjelman rakenne ja luokkien riippuvuus toisistaan graafisesti)

Saavutetut aika- ja tilavaativuudet

DFS-algoritmi toimii tavoitellussa aikavaativuudessaan O(|V| + |E|); solmujen color-muuttujien alustus vie aikaa O(|V|), DFS-visit kutsutaan korkeintaan kerran jokaiselle solmulle, eli näitä on enintään |V| kappaletta. DFS-visitin for-lause toistetaan jokaiselle solmun naapurille vieruslistalla, jolloin se toteutetaan |E| kertaa. Siispä aikaa kuluu kokonaisuudessaan O(|V| + |V| + |E|) = O(|V| + |E|). Tilavaativuus on tavoiteltu O(|V|), koska pahimmassa tapauksessa aloitussolmu vie yhtä polkua pitkin kaikkiin muihin solmuihin. Tällöin sisäkkäisiä DFS-visit-kutsuja tehdään |V| kappaletta.

BFS-algoritmi toimii myös tavoitellussa aikavaativuudessaan O(|V| + |E|); alustukseen menee aikaa O(|V|). Jonon enqueue- ja dequeue-operaatiot toteutuvat ajassa O(1) ja nämä tehdään korkeintaan |V| kertaa, jolloin aikaa kuluu O(|V]). Vieruslistojen yhteispituus on O(|E|), mikä on sama kuin niihin käytettävä aika. Eli aikaa menee O(|V| + |V| + |E|) = O(|V| + |E|). Tilavaativuus algoritmilla on O(|V|); Solmulla voi olla maksimissaan 4 naapuria, mutta jonoon voi pahimmillaan kertyä kaikki (tai lähes kaikkien) solmujen naapurit, kun kaikkia solmuja ei ole keretty käydä läpi uusien jo tullessa jonoon.

Dijkstran algoritmi toimii myös tavoitellussa ajassa: Alustukset vievät aikaa O(|V|) ja keko-operaatiot toimivat ajassa O(log n). Kekoon lisääminen toteutetaan |V| kertaa, jolloin aikaa kuluu O(|V|log|V|). Toistolauseen O(log|V|) aikaa vievää operaatio kutsutaan jokaiselle solmulle kerran, siis aikaa menee toistolauseeseen O(|V| log |V|). Relaksaatio vie aikaa O(log |V|) ja heapDecKey toistetaan maksimissaan |E| kertaa. Lopulliseksi aikavaativuudeksi tulee siis O((|E|+|V|)log|V|). Tilavaativuudeksi tulee O(|V|).

Suorituskyky- ja O-analyysivertaiu

Suorituskyvyltään parhaiten toimiva algoritmi on nyt DFS, mitä seuraa BFS ja viimeisenä tulee Dijkstra. DFS:llä reittien hakuun meni keskimäärin 3,6ms, BFS:llä 4,4ms ja Dijkstralla 5,1ms. Tämä johtuu puhtaasti siitä, että verkon rakenne suosii DFS- ja BFS-algoritmeja, kun jokaisen kaaren pituus on vain 1. Jos kaarien pituuksissa olisi eroja (kuten alla mainitussa parannusehdotuksessa), niin Dijkstra suoriutuisi todennäköisesti paremmin kuin nyt.

Verkon alustaminen hoidetaan verkon matriisiesityksen avulla. Yhden ruudun läpikäynti vie aikaa O(1), mutta sen aikana vieruslistaan saatetaan lisätä uusi solmu, joka tarkistaa, ettei samaa solmua lisätä kahdesti ja käy läpi koko vieruslistan. Tämän takia aikaa menee O(|V|) yhteen kierrokseen ja kierroksia on |V| määrä. Siispä verkon alustukseen menee aikaa O($|V|^2$). Tilaa menee matriisiesitykseen O(|V|) ja vieruslistaesitykseen O(|V| +|E|), eli yhteensä O(|V| + |E|).

Työn mahdolliset puutteet ja parannusehdotukset

Työhön olisi voinut toteuttaa ainakin A*-algoritmin. Käyttäjälle voisi antaa mahdollisuuden myös valita mahdollinen “välipysäkki”, minkä kautta reitin tulisi kulkea lopulliseen päämääräänsä. Ohjelmaan voisi myös lisätä muitakin esteitä, kuten esim. ruutu, josta pääsee kulkemaan lävitse, mutta sen läpi kulkeminen olisi kalliimpaa, jolloin sen kiertäminen saattaisi olla parempi vaihtoehto reitille. Dijkstran ja BFS:n osalla voisi olla hyvä sallia reitin haun lopettaminen heti kun maalisolmu on löydetty, jolloin turhaa työtä ei tule, kuten useimmiten käy. Algoritmi ei enää löydä parempaa reittiä verkon rakenteen vuoksi.

Lähteet

(1)Introduction to Algorithms, third edition - Thomas H.Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein s.606 ja tietorakenteiden- ja algoritmien luentokalvot, Kevät 2016 - Jyrki Kivinen (kalvo viikko 9, s.479) (2)Introduction to Algorithms, third edition - Thomas H.Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stei s.597 ja tietorakenteiden- ja algoritmien luentokalvot, Kevät 2016 - Jyrki Kivinen (kalvo viikko 9, s.457) (3)Tietorakenteiden- ja algoritmien luentokalvot, Kevät 2016 - Jyrki Kivinen (Viikko 10, s. 533)