Toteutusdokumentti

Ohjelmassa käytetään Solmu-olioita, joilla on paljon ominaisuuksia. Ne muodostavat 2-ulotteisen taulukon (luokka Taulukko) ja ilmoittavat oman paikkansa siellä x- ja y-koordinaattiensa avulla. Lisäksi niillä on merkkiarvo, jonka ne esittävät taulukon tulostuksessa. Muista solmujen ominaisuuksista seuraavissa kappaleissa.

Labyrintistä on ideana hakea nopein reitti lähtösolmusta maalisolmuun. Hakualgoritmeina on toteutettu Bellman-Ford sekä keollinen Dijkstra. Molemmat algoritmit ovat omana luokkanaan ja ne perivät luokan AlgoritmienMetodit, joka sisältää kummassakin hakualgoritmissa käytettävät apumetodit sekä yhteiset parametrit. Luokkahierarkia ei tässä toimi, sillä Dijkstra ja Bellman-Ford perivät yläluokaltaan kaikki yhteiset ominaisuutensa. Nyt tilanne on sama kuin jos kissa ja koira eivät perisikään eläintä vaan tassut, suun ja kaikki muut yhteiset ominaisuutensa. AlgoritmienMetodit ei käytä materiaalissa ohjeistettuja taulukoita distance ja path vaan nämä arvot ovat talletettuna solmuihin. Jokaisella solmulla on Solmu-muuttuja path, johon talletetaan se naapuri, jonka kautta kulkee toistaiseksi lyhin tiedetty reitti lähtösolmuun. Distance on solmussa kokonaislukumuuttujana paino, jota sitten löysätessä ja sen jälkeen keossa vertaillaan.

Bellman-Ford käy taulukon läpi V (=solmujen määrä, tässä taulukon leveys kertaa korkeus) kertaa. Jokaisella läpikäynnillä jokaisen solmun vierussolmut käydään läpi ja etäisyys löysätään.

Dijkstra taas tallettaa alustusoperaation jälkeen jokaisen taulukon solmun kekoon ja poistaa sieltä niitä yksitellen metodilla poppaa(). Kun poppaus suoritetaan, tarkistetaan solmun parametri kayty. Mikäli se on true, popataan seuraava alkio, ja mikäli ei, solmun parametri kayty merkitään trueksi ja sen jälkeen sen vieruslista käydään lävitse. Vieruslista on nelipaikkainen Solmu-taulukko ja siihen on taulukon luonnin jälkeen haettu solmun vierussolmut. Mikäli solmu ei ole null (solmu olisi taulukon ulkopuolella tai se on seinä eli paasy=false), relaxoidaan solmun ja vierussolmun välinen etäisyys. Relax muuttaa myös vierussolmun painon() eli distancen. Tämän jälkeen sen paikka keossa haetaan uudelleen. Operaatio jatkuu kunnes keko on tyhjä.

Lyhin reitti saadaan tulostettua, kun muutamme ensin maalisolmun arvoksi R ja etenemme siitä path-muuttujan avulla aina seuraavaan ja siitä taas seuraavaan solmuun, kunnes saavutamme lähtösolmun. Kun kaikkien solmujen arvo muuttuu etenemisen myötä R:ksi, saamme löydetyn reitin R-kirjaimilla näkyviin labyrinttiin. InitialiseSingleSource muuttaa kaikkien solmujen painon todella isoksi, ja lopuksi alkusolmun painon nollaksi. Paino siis vastasi distancetaulukon arvoa kyseisen solmun kohdalla.

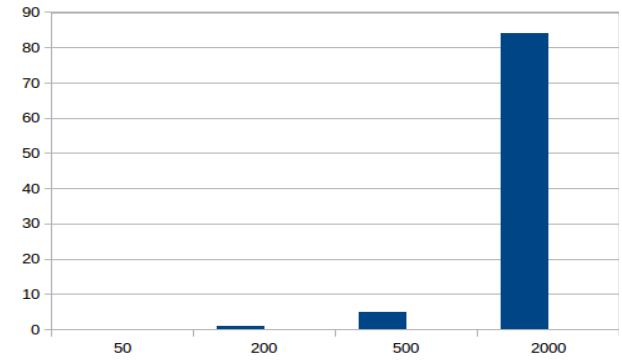
Minimikeossa käytetään Solmu-taulukkoa, jonka koko on niin suuri kuin näillä muistiasetuksilla sovellus salli. Ensimmäistä paikkaa ei käytetä, vaan solmujen talletus alkaa paikasta 1 eteenpäin. Tällöin solmun vanhemman sekä lasten haku taulukosta onnistuu materiaalissa näytetyillä laskutoimituksilla. Keossa prioriteetin määrää paino: mitä pienempi paino, sitä kovempi prioriteetti. Keon toteutuksessa on Solmutaulukon lisäksi käytetty koko-muuttujaa, joka kertoo, montako alkiota keossa on. Se helpottaa solmujen lisäämistä taulukkoon ja niiden poistamista sieltä.

Sovelluksessa on myös tekstikäyttöliittymä, joka ohjeistaa sovelluksen käytössä. Käyttäjä saa syöttää labyrintin mitat, aloitus ja lopetuspisteen koordinaatit sekä seinätodennäköisyyden. Lisäksi käyttäjä saa päättää, haluaako hän tehdä monta toistoa samoilla syötteillä ja vertailla Dijkstraa ja Bellman-Fordia, vai haluaako hän suorittaa reittihaun toisella algoritmeista. Käyttäjä saa myös halutessaan tulostuksen labyrintistä ennen ja jälkeen haun sekä tiedot haun kestosta. Myös syötteiden oikeellisuus on tarkastettu. Kun käyttöliittymä luo käyttäjän syötteiden mukaisen labyrintin, se ei välttämättä sisällä ratkaisua. Ongelma on ratkaistu niin, että ohjelma luo uuden ja taas uuden labyrintin niin kauan, kunnes reitti lähtöpisteestä maalipisteeseen löytyy. Luontimetodi

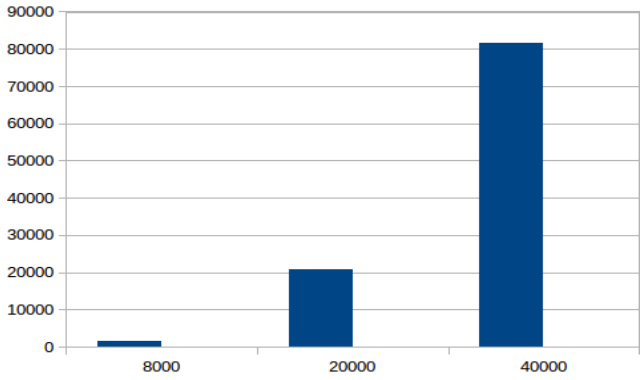
on määritetty Taulukko-luokassa: Luodaan solmu, arvotaan annetulla todennäköisyydellä sen paasy-parametri ja lisätään solmu 2-ulotteiseen solmu-taulukkoon. Luonnin jälkeen käyttöliittymä kutsuu myös jokaiselle solmulle Taulukko-luokan naapurinhakumetodia, jolloin Solmujen vieruslistat saavat sisältönsä.

Manuaalisen testauksen avulla on käynyt selväksi, että Bellman-Ford on monta kertaa hitaampi kuin Dijkstra. Pienillä syötteillä Dijkstra käyttää yhden millisekunnin, kun taas Bellman-Ford kolmisenkymmentä. Annoin myös 200*200 solmun taulukon kummankin algoritmin ratkaistavaksi: Dijkstralla aikaa meni keskimäärin 19 millisekuntia lyhimmän reitin löytämiseen, kun Bellman-Ford käytti aikaa keskimäärin 81638 millisekuntia (20 toistoa).

Vielä alla pylväsdiagrammi, jossa seinätodennäköisyys oli 0.19, toistojen määrä 20 ja taulukon solmujen määrä on pylvästen alla ja korkeus taas kertoo hakujen keskiarvoajan millisekunteina.



Pienillä syötteillä Dijkstralla meni 0 ms, kun taas jo 2000 solmun taulukko vei Bellman-Fordilta keskimäärin 84 ms.



Isommilla syötteillä Dijkstrallakin alkaa kulua useita millisekunteja, mutta selvitään silti alle 20 millisekunnin. Bellman-Ford taas kuluttaa oikeita sekunteja ja minuuttejakin. Vasta Dijkstran saatua 250 000 solmun taulukko aikaa kului keskimäärin 188 millisekuntia.

Vaikuttaisi siltä, että saavutetut aikavaativuudet ovat tavoitteissa, sillä kaikki ylimääräiset operaatiot ovat aikavaativuudeltaan O(1) tai enintään O(n), mikä ei muuta aikavaativuusluokkaa kummankaan algoritmin kohdalla. Tilaakaan ohjelma ei vie kuin O(n), sillä solmu-olio ei sisällä kuin neljä muuta solmua ja muutamia muita muuttujia, jotka eivät tilavaativuusluokkaa hetkauta.

Bellman-Ford olisi voitu toteuttaa tehokkaammin, ettei sen tarvitsisi käydä kaikkia kaaria läpi V kertaa. Ohjelma oli ylipäätäänsäkin vaikea saada toimimaan, joten kaikenlaisia turhia lausekkeita, metodeja ja viritelmiä se on täynnä.