

Toteutusdokumentti.

Ohjelma rakenne koostuu kolmesta paketista, käyttöliittämästä, polunetsintä algoritmeista ja avustavista tietorakenteista.

Käyttöliittymän päätarkoitus on visualisoida algoritmien toimintaa, ja antaa käyttäjälle mahdollisuus muokata kenttää jossa kyseiset algoritmit toimivat.

Tietorakenteet ovat toteutettu algoritmien toimintaa varten, ja tukevat suoraan niiden toimintaa toteuttamalla tarvittavia työkaluja.

Itse algoritmit ovat ohjelmiston runko ja tekevät suurimman osan työstä, hyödyntäen käyttöliittymää ja avustavia tietorakenteita.

Tietorakenteiden aika ja tilavaativuudet voidaan selvittää pseudokoodin perusteella.

Map olion tilavaativuus on vakio $O(1)$. Koodissa on kaksi Object taulukkoa molemmat koolla 100000 jotta kaikki tarvittava varmasti mahtuu taulukon sisälle.

Aikavaativuus puolestaan on muuten vakio $O(1)$, paitsi kun generoidaan objektin hashcode joka tapahtuu yksinkertaisen for-loopin sisällä. Joten kokonaisuudessaan se on $O(n)$.

Queue olion tilavaativuus saadaan myös vakioksi $O(1)$ koska kaikki taulut ovat ennalta määritetyn kokoisia.

Aikavaativuus Queue oliolla puolestaan on $O(N)$ arvon lisäyksessä, ja $O(2n)$ arvon hakemisessa. Joten voidaan sanoa, että olio toimii kokonaisuudessaan ajassa $O(n)$. Kun arvoa haetaan oliosta, niin Käydään kaikki arvot läpi, jotta saadaan paras arvo selville. Sen jälkeen tulee vielä siirtää kaikkia arvoja haetun arvon jälkeen yksi vasemmalle.

Astar algoritmi koostuu Queue ja Map olioista ja algoritmin omasta koodista. Algoritmi ei oman boolean taulukon lisäksi muita tietorakenteita, ja boolean taulukko on samalla tavalla vakiokokoinen kuin Queue ja Map, joten algoritmin tilavaativuus on myös vakio $O(1)$.

Astar algoritmin aikavaativuus ottamatta huomioon tietorakenteita koostuu while loopista, jonka sisällä on toinen for loop ja useampia vakioaikaisia toimintoja. Aikavaativuudeksi siis koostuu $O(n^m)$ kun otetaan huomioon myös tietorakenteiden kuluttama aika.

JumpPointSearch Algoritmi toimii samoilla tietorakenteilla kuin Astar, joten tilavaativuus on sama $O(1)$.

Aikavaativuus sen sijaan on hieman monimutkaisempi ja se koostuu while loopista jonka sisällä on kaksi peräkkäistä for looppia jotka molemmat sisältävät melko paljon vakioaikaisia operaatioita, joten aikavaativuudeksi saadaan $O(n^m)$

Käyttöliittymän Node olion tilavaativuus on $O(1)$, koska olio ei sisällä muuta kuin yksittäisiä vakioita.

Noden aikavaativuus on myös vakio $O(1)$, koska kaikki getterit ja setterit toimivat suoraan eikä missään kohtaa tarvitse käyttää esimerkiksi for-looppia.

Itse käyttöliittymä eli Grid olio saa myös tilavaativuudeksi vakion taulukon koon suhteen, eli $O(1)$.

Gridin aikavaativuus sen sijaan on $O(n^2)$ konstruktorin perusteella jossa on for-looppisiin upotettu for-looppi taulukon läpikäyntiä varten. Flush metodi toimii myös ajassa $O(n^2)$. ja DrawPath toimii lineaarisessa ajassa $O(n)$ väritettävien nodejen suhteen. Loput metodeista toimivat vakiollisessa ajassa, ja eivät täten hirveästi vaikuta algoritmien aikavaativuuksiin.