TESTAUSDOKUMENTTI

OLLI RÄTY

Tähän asti toteutettu testaus:

Yksikkötestaus:

Haamu:
Metodit ovat pääasiassa yksinkertaisia `get-` ja `set`-metodeja. Sijainnin arvot on rajattu positiivisiin, joten testataan arvoilla sekä positiivisten että negatiivisten arvojen ekvivalenssiluokista.

Kohde:
Luokassa on vain `get-` ja `set`-metodeja (konstruktorin lisäksi). Metodien arvojoukkoja ei ole rajattu, lukuun ottamatta int-muuttujan rajoja. Testattu on yksinkertaisesti satunnaisilla arvoilla ekvivalenssiluokasta.

Pino:
Pinon perustoimintoja testataan erilaisissa tilanteissa. Testataan, että pino palauttaa siinä olevat alkiot oikeassa järjestyksessä. Testataan erikoistilanteet, kuten tyhjästä pinosta poisto. Pinon myös pitää yllä tietoa koostaan, mitä testaan vastaavasti eri tiloissa.

Keko:
Keon perustoimintoja, lisäystä, poistoa, alkion arvon muuttamista yms., testataan erilaisilla arvoilla. Testattu on myös rajatapauksia, kuten tyhjää kekoa, saman arvoisia alkioita, negatiivisia alkioita ja keon 'täyttymistä'.

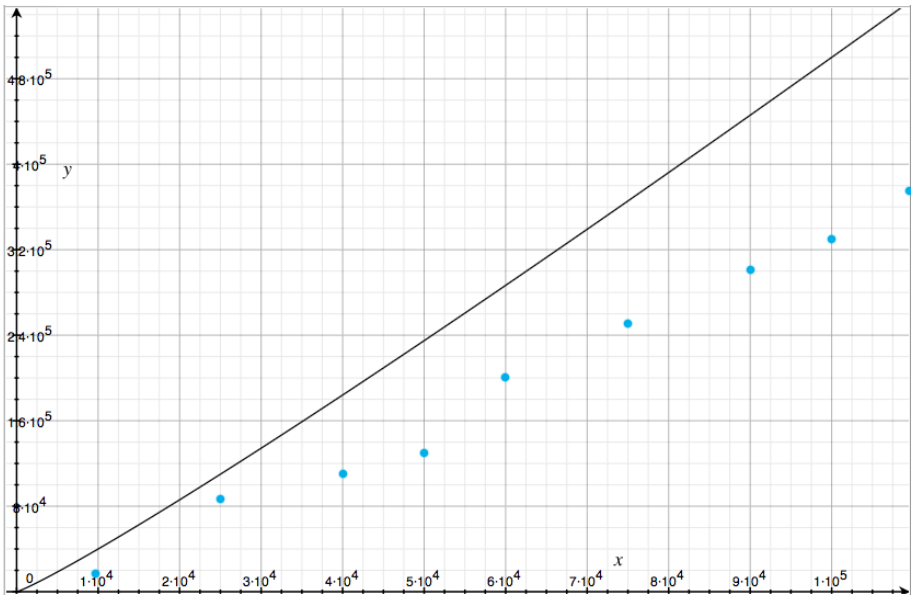
Random:
Luokan `seuraavaAskel(Haamu h, int[][] lab)`-metodi palauttaa seuraavan sijainnin haamulle. Testattu eri tilanteita: haamu voi olla joko risteyksessä, suoralla tai umpikujassa.

A*:
Testataan ensin perustoimivuutta; löytääkö algoritmi lyhimmän reitin. Tätä varten vertaamme yksinkertaisesti löydetyn reitin pituutta odotettuun.

Suorituskykytestaus:

Minimikeko:

Kekoa testattiin luomalla ensin n-kokoinen keko, jonka alkiot olivat satunnaisia lukuja väliltä $0-n$. Sekä `Poll`- että `Add`-toimintojen pitäisi viedä $O(\log(n))$ aikaa. Keolle suoritettiin Poll ja Add n kertaa. Aikaa pitäisi mennä $O(n\log(n))$. Kuvassa tuloksia:

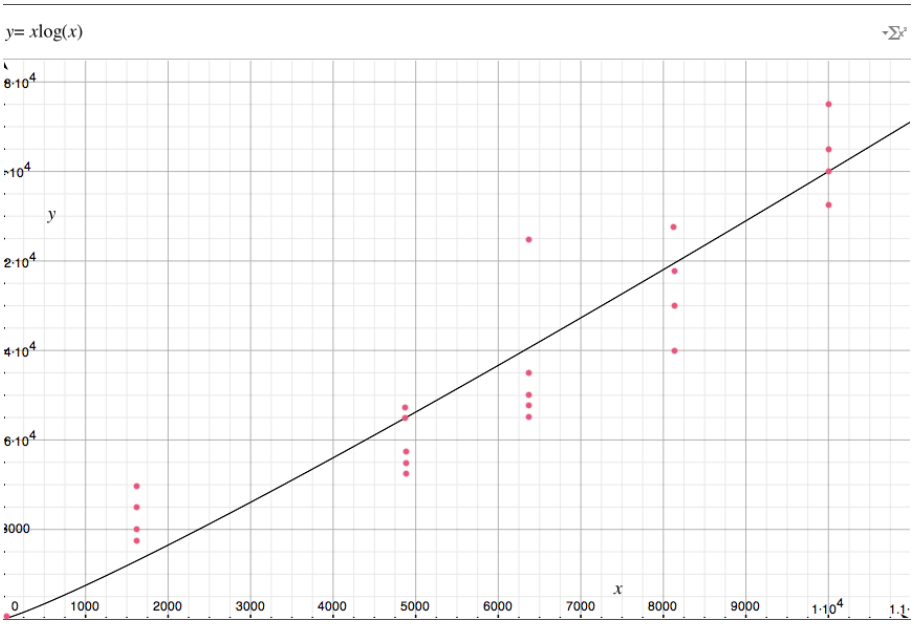


Kuvassa vaaka-akselilla on muuttujan n arvo. Siniset pisteet ovat testin viemiä keskiarvotuloksia muutamilla toistoilla. Kuvaaja $y = x\log(x)$ on kuvassa vertailua varten.

A*:

Algoritmin suorituksen viemää aikaa tastataan erikokoisilla labyrinteilla. Labyrintin ruudut ovat verkon solmuja. Algoritmin pitäisi toimia ajassa $O(|V|\log|V|)$, missä V on solmujen joukko. Tähänastiset tulokset tukevat tätä: 25:n solmun verkko vei alle millisekunnin, 1600:n solmun verkko hieman alle 10ms, ja 10000:n solmun verkko vajaat 40ms.

Ajettiin useita testejä erikokoisilla labyrinteilla. Kuvassa saadut tulokset:



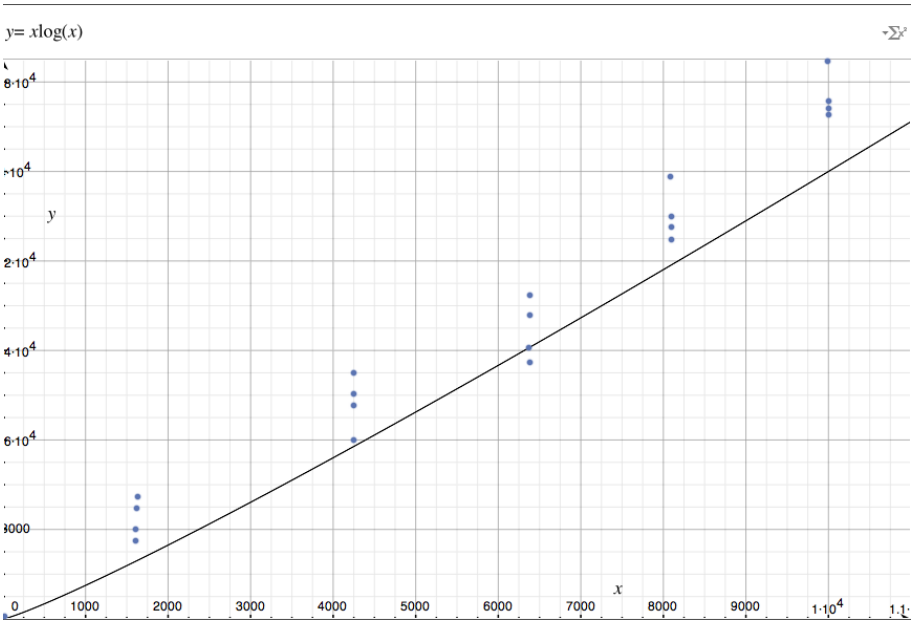
Kuvassa y-akseli kuvaa käytettyä aikaa ja x-akseli verkon solmujen lukumäärää. Pinkillä on merkitty A*-algoritmin käyttämä aika. Kuvaaja $x \log(x)$ on mukana vertailun vuoksi.

Muistin osalta algoritmin pitäisi olla kertaluokkaa $O(|V|)$, sillä se käyttää useita aputauluja, joissa on alkio solmua kohden kussakin. Toteuttamani algoritmi käyttää minimikekoa, joka toteutettiin tavallista `in[]`-taulukkoa käyttäen. Aluksi taulukon koko on 14. Taulukon koko kaksinkertaistetaan, jos alkioita on enemmän kuin siihen mahtuisi. Taulukon koko voi siis kasvaa suuremmaksi kuin $|V|$, muttei kuitenkaan suuremmaksi kuin $2|V|$. Sekin siis pysyy tavoitellussa kertaluokassa.

Dijkstra:

Algoritmia testattiin samoin kuin `A*`-algoritmia. Algoritmin tulisi toimia samassa kertaluokassa kuin A*:in, joskin hieman hitaammin, sillä se tutkii kaikki löydetyn lyhyimmän polun kanssa samanpituiset polut. Samoilla labyrinteilla algoritmin suoritukseen menin vajaa kaksinkertainen aika. Algoritmit näyttäisivät siis olevan samaa kertaluokkaa.

Ajettiin useita testejä erikokoisilla labyrinteilla. Kuvassa saadut tulokset:



Kuvassa x-akseli kuvaa verkon solmujen lukumäärää ja y käytettyä aikaa. Sinisellä on merkitty Dijkstra-algoritmin käyttämä aika. Algoritmin käyttämä aika on tässä jaettu kahdella; todellinen aika on siis aina kaksinkertainen merkittyyn nähden. Kertaluokkaanhan tämä ei vaikuta. Näin tehtiin yksinkertaisesti käytännön syistä: kuvasta tuli näin siistimpi. Varianssi yksittäisillä labyrinteilla on siis tosiasiassa hieman suurempi.

Dijkstra käyttää A*:ia vastaavia aputauluja, joten sen kertaluokka on muistinkäytön kannalta samainen $O(|V|)$.