**Dijkstran algoritmi**

Tomi Hyvärinen (N3652), TTV19SM

N3652@student.jamk.fi

Loppuraportti

TTZC0700 Tietorakenteet ja algoritmit/Antti Häkkinen

Joulukuu 2020

Tekniikan ala

Insinööri (AMK), Tieto- ja viestintätekniikka

**Sisältö**

[1 Johdanto 2](#_Toc58189182)

[2 Matrix taulu 2](#_Toc58189183)

[3 Dijkstra 2](#_Toc58189184)

[4 Dijkstra ja Matrix 3](#_Toc58189185)

[5 Matrixin koon vaikutus 9](#_Toc58189186)

[6 Pohdinta 13](#_Toc58189187)

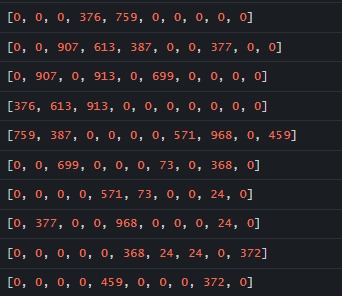
[7 Koodi 13](#_Toc58189188)

# Johdanto

Raportissa käydään läpi dijkstran algoritmiä, jolla saadaan laskettua lyhin reitti paikasta A paikkaan B. Myös matrix-taulua käydään läpi, jolla saamme määriteltyä montako eri reittä on ja mitkä ovat matkojen etäisyydet. Selvitetään myös miten reittien määrä vaikuttaa aikaan.

# Matrix taulu

Matrix-taulu on listoja listan sisällä. Listat tulee muodostaa listan pituuden mukaan. Jos lista sisältää 5 numeroa, on listan sisällettävä myös 5 listaa, eli matrix-taulusta tulee viiden numeron listoja 5 kappaletta. Näillä on vielä yksi ehto joka on esimerkiksi jos vaikka ensimmäisen listan paikalla 4 on numero (kuvassa numero 376) tämän numeron on pakko olla sama kuin 4 listan (kuvassa numero 376) ensimmäisellä paikalla, kuten alla olevasta matrix-taulussa nähdään. Lista on peilikuva missä listan korkeus ja leveys on sama.



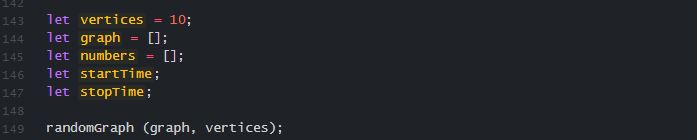
# Dijkstran algoritmi

Djikstran algoritmi on alkoritmi joka selvittää lyhimmä retin paikasta A paikkan B. Dijkstran algoritmissa voidaan käyttää esim. ylläolevan kuvan Matrix taulua josta valitaan lähtöpiste ja loppupiste. Sen avulla selvitetään yhteydet ja etäisyydet yhteyksissä , ja näiden avulla valitaan lyhin reitti.

# Dijkstran algoritmi ja Matrix

Aloitin suunnittelun sellaisen matrix-taulun luomisella jonka kokoa voidaan voidaan muuttaa, ja että myös mahdollisia yhteyksiä ja etäisyyksiä voitaisiin muuttaa tarvittaessa.

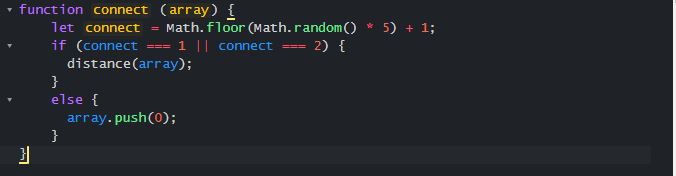
Loin muuttujan (vertices) jolla määritellään matrix-taulun koko ja myös itse matrix-taulun (graph) joka on aluksi tyhjä. Seuraavaksi tein funktion (randomGraph()) matrix-taulun luontiin, ja kutsuin funktiota randomGraph() ja tälle välitettiin matrix-taulu ja sen koko.



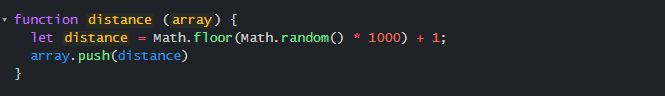
Funktio randomGraph() ottaa vastaan matrix-taulun ja sen koon. Luodaan muuttuja graphSize jolle annetaan taulun koko. Toistetaan for-looppia taulukon koolle määrätyn arvon mukaan, jonka avulla saadaan luotua kyseisen määrän tauluja matrix-tauluun. Luodaan tyhjä taulu. Seuraavaksi vertaillaan onko taulukon pituus sama kuin taulujen määrä, jos on, tauluun asetetaan arvo 0, tämä on risteyskohta. Jos edellinen ehto ei toteudu, verrataan onko tauluja luotu vähemmän kuin taulukon pituus tällä hetkellä. Luodaan kaksi muuttujaa, jotta saadaan jo valmiiksi luodusta matrix-taulusta kyseinen vasta numero oikealle paikalle ja asetetaan se tauluun. Jos tämäkään ei toteudu, käytämme connect()-funktiota. Kun taulu on luotu, tämä lisätään matrix.tauluun. Tätä toistetaan kunnes matrix-taulu on valmis.



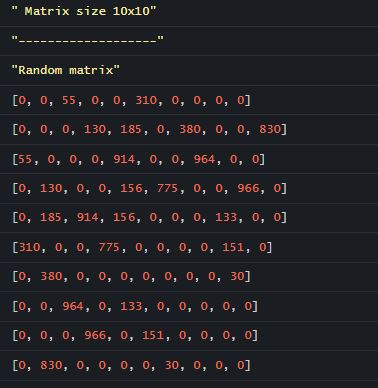
Edellisessä kuvassa connect-funktiota käytettäessä välitetty lista siirtyy connect-funktioon. Connect-funktiolla arvotaan luodaanko listaan yhteys vai ei, ja sen sisällä luodaan muuttuja ”connect” johon arvotaan numero 1-5. Jos numero on 1 tai 2 välitetään lista aina eteenpäin distance-funktiolle jos ei, niin asetetaan listaan 0.



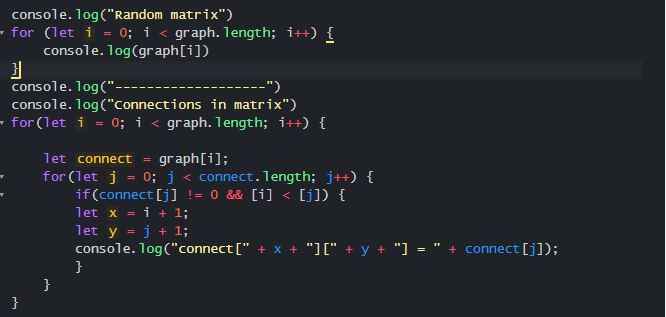
Distance-funktiossa arvotaan etäisyys pisteiden välillä. Funktio ottaa listan vastaan, luodaan muuttuja ”distance”, jolle arvotaan arvo väliltä 1-1000 ja asetetaan tämä numero listan paikalle.

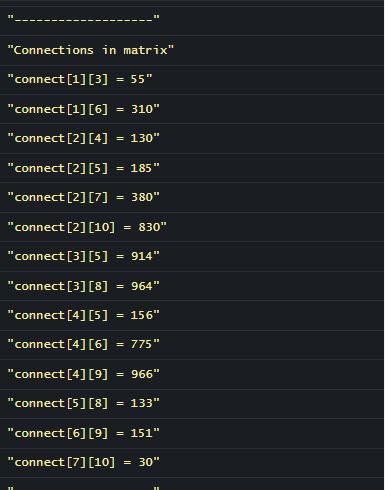


Kun nämä kaikki on suoritettu, on matrixin lista valmis. Alla on kuva valmiista 10x10 matrix-taulusta.

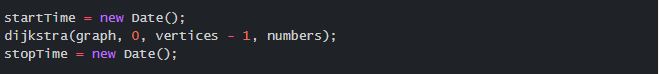


Seuraavaksi esitetään matrix kuten yllä olevassa kuvassa näkyy, käymme läpi for-loopilla taulukon, jonka jokaisen numeron tulostamme. Connection-matrix tulostaa kaikki yhteydet, joita matrixia luodessa on syntynyt. For-looppi käydään läpi taulukoiden määrän verran, luodaan uusi muuttuja jolle annetaan for-loopissa käytävän taulun numero, seuraavassa for-loopissa tarkistetaan matrixin-taulusta sisältääkö taulu numeron eri numeron kuin 0 ja myös tarkistetaan onko listan paikka pienempi kuin matrixissa käytävä lista. Jos nämä toteutuvat esitetään yhteys, tällä sain aikaiseksi että vain yksi yhteys esitettiin eikä esim. 1->2 ja 2<-1.





Dijkstran algoritmiin syötetään yllä olevan mukaan luotu matrix-taulu, tälle annetaan parametreinä mikä on lähtöpiste (0) ja mikä on loppupiste (taulukon pituus, tässä tapauksessa 9), välitetään myös numerolista joka on luotu taulukon pituuden mukaan jotta saadaan selvä reittikuvaus.



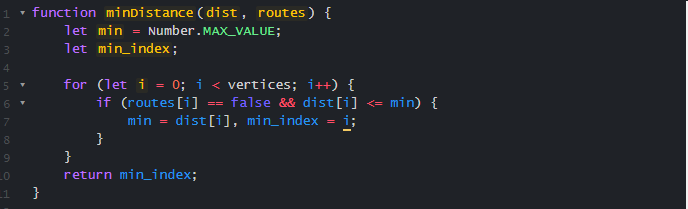
Dijkstra-funktiossa luodaan uusi tyhjä etäisyys-taulu, tyhjä reittitaulu ja parent-taulu.

Aloitetaan for-loopin pyörittäminen taulukon pituuden arvon verran. ”Parent” muuttujaan asetetaan arvo, joka on for-loopissa käytävän arvo -1, distance paikkaan, joka sijaitsee for-loopin käytävän arvon mukaiseen paikkaan, muuttuja haetaan max\_valuen avulla uusi arvo, reitti arvoon haetaan samalla tavalla arvo false.

Seuraavaa for-loopia kierrätetään kunnes taulukon pituus -1 on saman arvoinen kuin count. Luodaan muutuja u jolle haetaan arvo minDistance funktiosta, vaihdetaan reitin arvo true kyseisessä kohdassa. Käydään samalla myös toinen for-looppi läpi, kunnes reitti muutuja[v] on false, matrix-taulun indexit ja distance lisättynä matrix-taulun indexien arvon ovat pienempiä kuin etäisyys. Tämän for-loopin sisällä asetetaan parent for-loopin arvon kohtaan u, ja distance for-loopin arvon kohtaan distance u + taulukon u ja v. Näitä for-looppeja käydään läpi, kunnes ehdot ovat täyttyneet. Tämän jälkeen käytetään finalGraph-funktiota, jonne välitetään uudet arvot jotka on saatu for-looppeja käytäessä.

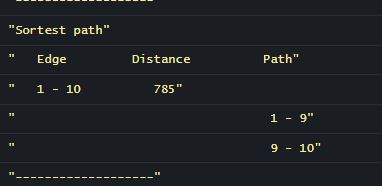


MinDistance-funktiolla haetaan uuteen min muuttujaan max\_value ja luodaan uusi min\_index muuttuja. For-loopia loopataan listan pituuden arvon verran. Jos reitti listan indeksi paikan numero on false ja distance indeksin paikka on pienempi tai yhtäsuuri kuin min, asetetaan uudeksi min-arvoksi etäisyyden indeksin arvo, ja min\_indexille annetaan uusi arvo käytävän indeksin mukaan. Lopuksi palautetaan uusi min\_indeksi.



FinalGraph-funktiossa for-looppia käydään taulukon pituuden arvon verran. Jos annetu päätepiste on sama kuin for-loopin indeksi, tätä tehdään kunnes ollaan saatu piirrettyä lyhin reitti ja reitti ohjeet paikan A ja B:n välillä.



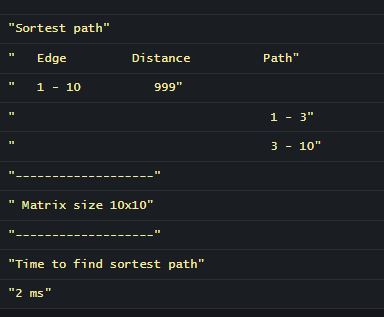


Tässä oli lyhyesti selitettynä Djikstran algoritmin toiminta.

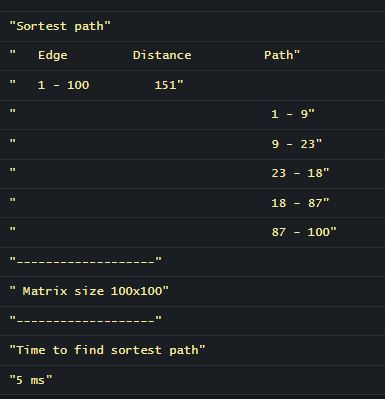
# Matrixin koon vaikutus

Tässä huomataan hyvin, että reitin aika kasvaa huomattavasti mitä isompi matrix-taulu on. Ajan mittaamiseen käytettiin 10, 100, 250, 500, 1000 ja 2500 listan matrix-taulukkoa:

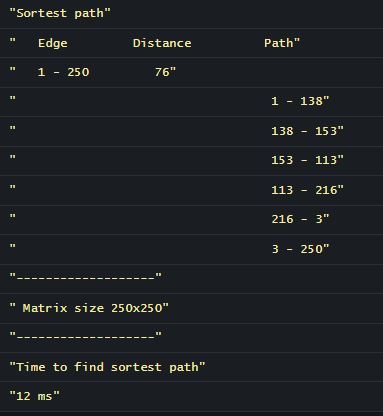
10x10 hyvin nopea vain 2ms



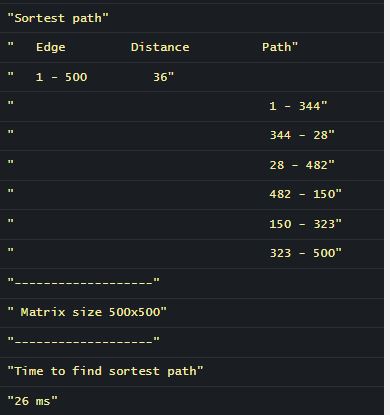
100x100 vieläkin nopea 5ms



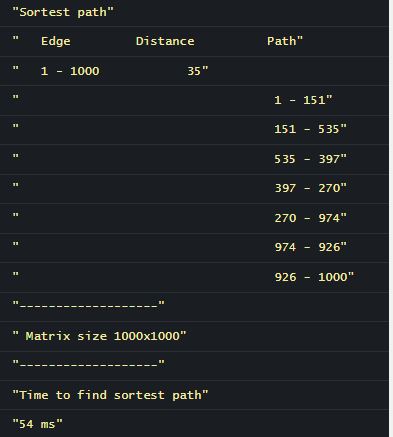
250x250 Edelleen nopea 12ms



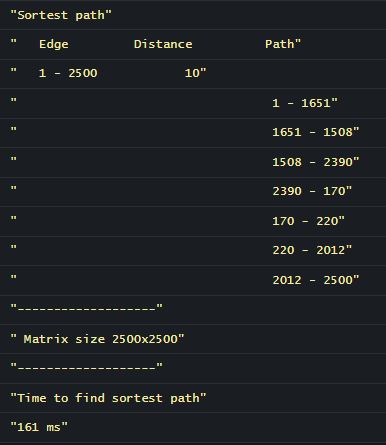
500x500 Rupeaa aika kasvamaan jo vähän reilummin, 26ms.



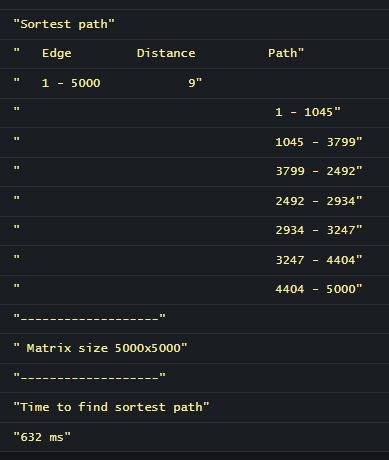
1000x100 aika tuplaantui 54ms.



2500x2500 nyt aika oli jo huomattavasti suurempi, 161ms



5000x5000 meni jo huomattavasti kauemmin 632ms



Kuten mittauksista huomasimme, aika nousi eksponentiaalisesti, koska reittejä rupesi olemaan jo niin paljon.

# Pohdinta

Tässä projektissa tuli tutuksi kyllä juuria myöten Matrix-taulu ja sen luonti, tämä oli todella mukava luoda. Välillä meinasi tulla vähän vaikeuksia, mutta viimein kun sain random matrix-taulun luotua olin todella tyytyväinen. Dijkstran algoritmi tuli myös projektin aikana tutuksi, todella tärkeä ja opettava algoritmi reittien etsimiseen. Kaiken kaikkiaan mukava ja haastava projekti mutta mielestäni hyvin onnistunut.

# Koodi

function minDistance(dist, routes) {

let min = Number.MAX\_VALUE;

let min\_index;

for (let i = 0; i < vertices; i++) {

if (routes[i] == false && dist[i] <= min) {

min = dist[i], min\_index = i;

}

}

return min\_index;

}

function printGraph(parent, j, numbers) {

if (j == null) {

return;

}

if (parent[j] == -1) {

return;

}

printGraph(parent, parent[j], numbers);

console.log("\t\t\t\t\t\t\t\t\t" + numbers[parent[j]] + ' - ' + numbers[j])

}

function finalGraph(dist, parent, src, end, numbers) {

for (let i = 0; i < vertices; i++) {

if(end == i){

console.log("\tEdge \t\t\ Distance \t\t Path");

console.log('\t' + numbers[src] + ' - ' + numbers[i] + '\t\t\t' + dist[i]);

printGraph(parent, i, numbers);

}

}

}

function dijkstra(graph, src, end, numbers) {

let dist = [];

let routes = [];

let parent = [-1];

for (let i = 0; i < vertices; i++) {

parent[i] = -1;

dist[i] = Number.MAX\_VALUE;

routes[i] = false;

}

dist[src] = 0;

for (let count = 0; count < (vertices - 1); count++) {

let u = minDistance(dist, routes);

routes[u] = true;

for (let v = 0; v < vertices; v++) {

if (!routes[v] && graph[u][v] && dist[u] + graph[u][v] < dist[v]) {

parent[v] = u;

dist[v] = dist[u] + graph[u][v];

}

}

}

finalGraph(dist, parent, src, end, numbers);

}

function distance (array) {

let distance = Math.floor(Math.random() \* 1000) + 1;

array.push(distance)

}

function connect (array) {

let connect = Math.floor(Math.random() \* 5) + 1;

if (connect === 1 || connect === 2) {

distance(array);

}

else {

array.push(0);

}

}

function randomGraph (graph, vertices) {

let graphSize = vertices;

for (let i = 0; i < graphSize; i++) {

let array = [];

for (let j = 0; j < graphSize; j++) {

if (array.length == [i]) {

array.push(0)

}

else if (graph.length > array.length) {

let graphIndex = ((graph.length - array.length - graph.length) \* -1)

let arrayIndex = graph.length;

let num = graph[graphIndex][arrayIndex]

array.push(num)

}

else {

connect(array);

}

}

graph.push(array);

}

}

function calculateTime(start, stop){

let time = stop.getMilliseconds() - start.getMilliseconds();

console.log(time + " ms")

}

let vertices = 10;

let graph = [];

let numbers = [];

let startTime;

let stopTime;

randomGraph (graph, vertices);

console.log("Random matrix")

for (let i = 0; i < graph.length; i++) {

console.log(graph[i])

}

console.log("-------------------")

console.log("Connections in matrix")

for(let i = 0; i < graph.length; i++) {

let connect = graph[i];

for(let j = 0; j < connect.length; j++) {

if(connect[j] != 0 && [i] < [j]) {

let x = i + 1;

let y = j + 1;

console.log("connect[" + x + "][" + y + "] = " + connect[j]);

}

}

}

for (let i = 0; i < vertices; i++){

numbers.push(i + 1);

}

console.log("-------------------")

console.log("Sortest path")

startTime = new Date();

dijkstra(graph, 0, vertices - 1, numbers);

stopTime = new Date();

console.log("-------------------")

console.log(" Matrix size " + vertices + "x" + vertices);

console.log("-------------------")

console.log("Time to find sortest path")

calculateTime(startTime, stopTime)