

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
2	Matrix taulu	2
3	Dijkstra	2
4	Dijkstra ja Matrix	3
5	Matrixin koon vaikutus	9
6	Pohdinta	13
7	Koodi.....	13

1 Johdanto

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

2 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.

```
[0, 0, 0, 376, 759, 0, 0, 0, 0, 0]
[0, 0, 907, 613, 387, 0, 0, 377, 0, 0]
[0, 907, 0, 913, 0, 699, 0, 0, 0, 0]
[376, 613, 913, 0, 0, 0, 0, 0, 0, 0]
[759, 387, 0, 0, 0, 0, 571, 968, 0, 459]
[0, 0, 699, 0, 0, 0, 73, 0, 368, 0]
[0, 0, 0, 0, 571, 73, 0, 0, 24, 0]
[0, 377, 0, 0, 968, 0, 0, 0, 24, 0]
[0, 0, 0, 0, 0, 368, 24, 24, 0, 372]
[0, 0, 0, 0, 459, 0, 0, 0, 372, 0]
```

3 Dijkstran algoritmi

Dijkstran algoritmi on algoritmi joka selvittää lyhimpiä reittejä paikasta A paikkaan B. Dijkstran algoritmista 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.

4 Dijkstran algoritmi ja Matrix

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

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

```
142  
143 let vertices = 10;  
144 let graph = [];  
145 let numbers = [];  
146 let startTime;  
147 let stopTime;  
148  
149 randomGraph (graph, vertices);
```

Funktio randomGraph() ottaa vastaan matrix-aulun 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-auluun. 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-aulusta kyseinen vasta numero oikealle paikalle ja asetetaan se tauluun. Jos tämäkään ei toteudu, käytämme connect()-funktia. Kun taulu on luotu, tämä lisätään matrix.auluun. Tätä toistetaan kunnes matrix-aulu on valmis.

```

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);
    }
}

```

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.

```

function connect (array) {
    let connect = Math.floor(Math.random() * 5) + 1;
    if (connect === 1 || connect === 2) {
        distance(array);
    }
    else {
        array.push(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.

```

function distance (array) {
    let distance = Math.floor(Math.random() * 1000) + 1;
    array.push(distance)
}

```

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

```
" Matrix size 10x10"
"-----"
"Random matrix"
[0, 0, 55, 0, 0, 310, 0, 0, 0, 0]
[0, 0, 0, 130, 185, 0, 380, 0, 0, 830]
[55, 0, 0, 0, 914, 0, 0, 964, 0, 0]
[0, 130, 0, 0, 156, 775, 0, 0, 966, 0]
[0, 185, 914, 156, 0, 0, 0, 133, 0, 0]
[310, 0, 0, 775, 0, 0, 0, 0, 151, 0]
[0, 380, 0, 0, 0, 0, 0, 0, 0, 30]
[0, 0, 964, 0, 133, 0, 0, 0, 0, 0]
[0, 0, 0, 966, 0, 151, 0, 0, 0, 0]
[0, 830, 0, 0, 0, 0, 30, 0, 0, 0]
```

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.

```

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]);
    }
  }
}

```

```

"-----"
"Connections in matrix"
"connect[1][3] = 55"
"connect[1][6] = 310"
"connect[2][4] = 130"
"connect[2][5] = 185"
"connect[2][7] = 380"
"connect[2][10] = 830"
"connect[3][5] = 914"
"connect[3][8] = 964"
"connect[4][5] = 156"
"connect[4][6] = 775"
"connect[4][9] = 966"
"connect[5][8] = 133"
"connect[6][9] = 151"
"connect[7][10] = 30"
"

```

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.

```

startTime = new Date();
dijkstra(graph, 0, vertices - 1, numbers);
stopTime = new Date();

```

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 muuttuja u jolle haetaan arvo minDistance funktiosta, vaihdetaan reitin arvo true kyseisessä kohdassa. Käydään samalla myös toinen for-looppi läpi, kunnes reitti muuttuja[v] on false, matrix-tilun indexit ja distance lisättynä matrix-tilun 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.

```

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;
  }

  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);
}

```



```

"Sortest path"
"  Edge      Distance      Path"
"  1 - 10      785"
"
"              1 - 9"
"              9 - 10"
"-----"

```

Tässä oli lyhyesti selitettynä Djikstran algoritmin toiminta.

5 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

```

"Sortest path"
"  Edge      Distance      Path"
"  1 - 10      999"
"
"              1 - 3"
"              3 - 10"
"-----"
" Matrix size 10x10"
"-----"
"Time to find sortest path"
"2 ms"

```

100x100 vieläkin nopea 5ms

"Sortest path"			
"	Edge	Distance	Path"
"	1 - 100	151"	
"			1 - 9"
"			9 - 23"
"			23 - 18"
"			18 - 87"
"			87 - 100"
"-----"			
" Matrix size 100x100"			
"-----"			
"Time to find sortest path"			
"5 ms"			

250x250 Edelleen nopea 12ms

"Sortest path"			
"	Edge	Distance	Path"
"	1 - 250	76"	
"			1 - 138"
"			138 - 153"
"			153 - 113"
"			113 - 216"
"			216 - 3"
"			3 - 250"
"-----"			
" Matrix size 250x250"			
"-----"			
"Time to find sortest path"			
"12 ms"			

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

"Sortest path"			
"	Edge	Distance	Path"
"	1 - 500	36"	
"			1 - 344"
"			344 - 28"
"			28 - 482"
"			482 - 150"
"			150 - 323"
"			323 - 500"
"-----"			
" Matrix size 500x500"			
"-----"			
"Time to find sortest path"			
"26 ms"			

1000x100 aika tuplaantui 54ms.

"Sortest path"			
"	Edge	Distance	Path"
"	1 - 1000	35"	
"			1 - 151"
"			151 - 535"
"			535 - 397"
"			397 - 270"
"			270 - 974"
"			974 - 926"
"			926 - 1000"
"-----"			
" Matrix size 1000x1000"			
"-----"			
"Time to find sortest path"			
"54 ms"			

2500x2500 nyt aika oli jo huomattavasti suurempi, 161ms

"Sortest path"			
"	Edge	Distance	Path"
"	1 - 2500	10"	
"			1 - 1651"
"			1651 - 1508"
"			1508 - 2390"
"			2390 - 170"
"			170 - 220"
"			220 - 2012"
"			2012 - 2500"
"-----"			
" Matrix size 2500x2500"			
"-----"			
"Time to find sortest path"			
"161 ms"			

5000x5000 meni jo huomattavasti kauemmin 632ms

"Sortest path"			
"	Edge	Distance	Path"
"	1 - 5000	9"	
"			1 - 1045"
"			1045 - 3799"
"			3799 - 2492"
"			2492 - 2934"
"			2934 - 3247"
"			3247 - 4404"
"			4404 - 5000"
"-----"			
" Matrix size 5000x5000"			
"-----"			
"Time to find sortest path"			
"632 ms"			


```

        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)

```