

9

Algoritme van Dijkstra


Inleiding

Download het bestand `week9-Grafen-Dijkstra-opgave.zip` van Toledo. Importeer dit bestand in je IDE.

In deze oefenzitting leer je het algoritme van Dijkstra te implementeren in Java.


De eerste oefening is op papier. Hiermee leer je het algoritme goed begrijpen. Verder kan je deze papieren versie gebruiken om je zelfgeschreven code te testen. Houd de oplossing dus goed bij!

Oefening 9.1

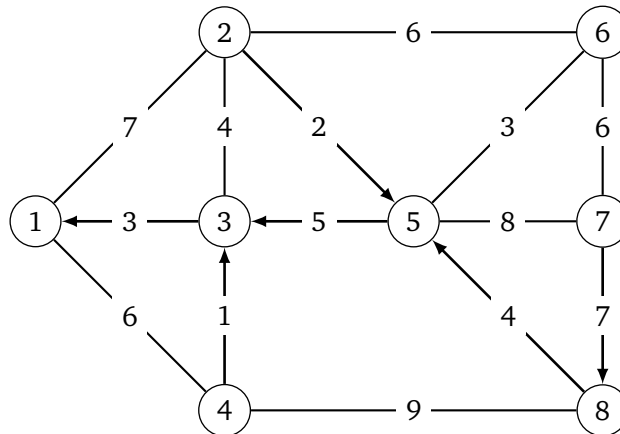
 Figuur 9.1 op pagina 50 toont een netwerk. De aangegeven getallen bij elke verbinding zijn het aantal km tussen beide knooppunten. Stel een lijst op met de kortste afstanden vanuit *knooppunt 3* naar elk ander punt. Geef ook telkens de kortste route aan. Gebruik de methode van Dijkstra zoals uitgelegd in hoofdstuk 3 van de cursusnota's "GrafenTheorie.pdf".

1. Pas het algoritme grafisch toe cfr. fig. 3.3 in de cursusnota's.
2. Gebruik de tabelmethode (tabel 3.3-3.11 in de cursusnota's).

Oefening 9.2

 In tegenstelling tot het algoritme van Floyd en BFS, is het algoritme van Dijkstra zoals je het vindt in de cursus niet 'reeds klaar voor Java-gebruik'. In deze eerste oefening maken we daarom de vertaling van het algoritme naar een alternatieve tabelmethode die geïmplementeerd kan worden in Java.

Lees het document 'oefeningAlternatieveTabelMethodeVoorDijkstra.pdf'. Los nu oefening 9.1 nogmaals op papier op, maar nu met deze methode.



Figuur 9.1 Netwerk bij oefening 9.1

Oefening 9.3



Je bent eindelijk klaar met het papierwerk. Je kan nu beginnen met het opzetten van de klasse Graph.

Schrijf de methode `int[][] initMatrixDijkstra(int vanKnoop)`. Deze methode modelleert de gewichtenmatrix zoals uitgelegd in stap 1 en stap 2 van het document 'oefeningAlternatieveTabelMethodeVoorDijkstra.pdf'. Let op de indices die je gebruikt!

- Er wordt een `int[][]` aangemaakt die evenveel kolommen telt als de gewichtenmatrix, maar één extra rij.
- Om aan te geven dat de elementen van de laatste rij leeg zijn, stellen we ze gelijk aan `Integer.MAX_VALUE`.
- De overige elementen krijgen de corresponderende waarde van de gewichtenmatrix waarbij infinity vervangen wordt door 0.
- Zet in de kolom die overeenkomt met de startknoop nullen.

Verwachte uitvoer bij het voorbeeld uit de cursustekst, waar een extra knoop 9 toegevoegd is cfr. main methode `UIDijkstra`:

Initiele matrix:

0	5	9	0	0	0	0	0	0
0	0	3	8	10	11	0	0	0
0	3	0	2	0	0	7	0	0
0	8	2	0	0	3	7	0	0
0	10	0	0	0	1	0	8	0
0	0	0	3	1	0	5	10	0
0	0	7	7	0	0	0	12	0
0	0	0	0	8	10	12	0	0
0	0	0	0	0	0	0	0	0
0	inf	inf	inf	inf	inf	inf	inf	inf

Oefening 9.4



Schrijf de methode `int[][] Dijkstra(int vanKnoop)`. Deze methode implementeert het algoritme van Dijkstra. Ze geeft de aangepaste gewichtenmatrix terug, met extra rij onderaan met de kleinste afstanden.

Verwachte uitvoer:

Resulterende matrix:

```
0 5 0 0 0 0 0 0 0
0 0 3 0 0 0 0 0 0
0 0 0 2 0 0 7 0 0
0 0 0 0 0 3 0 0 0
0 0 0 0 0 0 0 8 0
0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 5 8 10 14 13 15 22 inf
```

Oefening 9.5



Schrijf `ArrayList<Integer> vindPad(int vanKnoop, int naarKnoop, int[][] res)`, een methode die het kortste pad van `vanKnoop` naar `naarKnoop` berekent. De veranderlijke `int[][] res` is de aangepaste gewichtenmatrix zoals die teruggegeven wordt door de methode `Dijkstra`.

Verwachte uitvoer:

```
Kortste afstand van 1 naar 2 = 5
via [1, 2]
Kortste afstand van 1 naar 3 = 8
via [1, 2, 3]
Kortste afstand van 1 naar 4 = 10
via [1, 2, 3, 4]
Kortste afstand van 1 naar 5 = 14
via [1, 2, 3, 4, 6, 5]
Kortste afstand van 1 naar 6 = 13
via [1, 2, 3, 4, 6]
Kortste afstand van 1 naar 7 = 15
via [1, 2, 3, 7]
Kortste afstand van 1 naar 8 = 22
via [1, 2, 3, 4, 6, 5, 8]
```

9 *Algoritme van Dijkstra*

Er is geen pad van 1 naar 9

Oplossingen

Oplossing 9.1

Stad	Kortste afstand vanuit 3	Route
1	3	3 → 1
2	4	3 → 2
3	0	
4	9	3 → 1 → 4
5	6	3 → 2 → 5
6	9	3 → 2 → 5 → 6
7	14	3 → 2 → 5 → 7
8	18	3 → 1 → 4 → 8

Oplossing 9.4

Listing 2 Dijkstra

```
private int getAantalKnopen() {
    return gewichtenMatrix.length;
}

private int[][] initMatrixDijkstra(int vanKnoop) {
    int[][] res = new int[this.gewichtenMatrix.length + 1][this.gewichtenMatrix.length];
    // laatste rij is rij met kortste lengtes vanuit vanKnoop

    // oefening 3.3
    for (int i = 0; i < getAantalKnopen(); i++) {
        for (int j = 0; j < getAantalKnopen(); j++)
            res[i][j] = gewichtenMatrix[i][j] != inf ? gewichtenMatrix[i][j] : 0;
        res[getAantalKnopen()][i] = inf;
    }
    for (int i = 0; i <= getAantalKnopen(); i++) {
        res[i][vanKnoop - 1] = 0;
    }
    return res;
}

public int[][] Dijkstra(int vanKnoop) {
    int[][] res = initMatrixDijkstra(vanKnoop);

    System.out.println("Initiele matrix: \n");
    printIntMatrix(res);
}
```

Oplossingen

```
// oefening 3.4
// herhaal voor alle knopen
for (int i = 0; i < getAantalKnopen() - 1; i++) {
    // zoek nieuwe minimale afstand
    int min = inf;
    int[] knopenpaar = {inf, inf}; // index die het nieuwe minimum is
    for (int j = 0; j < getAantalKnopen(); j++) {
        // herhaal voor alle knopen die al bezocht zijn
        if (res[getAantalKnopen()][j] != inf) {
            for (int k = 0; k < getAantalKnopen(); k++) {
                // als knoop k+1 nog niet gevonden is,
                // als er een verbinding is tussen knoop j+1 en knoop k+1
                // en als de verbinding tussen deze knopen korter is
                // dan het minimum tot nog toe
                if (res[getAantalKnopen()][k] == inf && res[j][k] != 0 &&
                    res[getAantalKnopen()][j] + res[j][k] < min) {
                    // onthoud (index van) dit knopenpaar en hun minimum
                    knopenpaar[0] = j;
                    knopenpaar[1] = k;
                    min = res[getAantalKnopen()][j] + res[j][k];
                }
            }
        }
    }
    // tussenresultaat wegschrijven indien er verbetering is
    if (knopenpaar[0] != inf && knopenpaar[1] != inf) {
        // nieuwe minimum
        res[getAantalKnopen()][knopenpaar[1]] = min;
        for (int j = 0; j < getAantalKnopen() - 1; j++) {
            // kolom op nul zetten, maar niet op de plaats die het minimum aanlevert
            if (j != knopenpaar[0])
                res[j][knopenpaar[1]] = 0;
        }
    }
}
return res;
}
```

Oplossing 9.5

Listing 3 vindPad

```
private ArrayList<Integer> vindPad(int vanKnoop, int naarKnoop, int[][] res) {
    ArrayList<Integer> pad = new ArrayList<>();
    // oefening 3.5
    // naarKnoop, vanKnoop en k zijn namen van knopen
    // hun index in de matrix is altijd eentje minder want de rijen/kolommen tellen vanaf 0
    pad.add(naarKnoop);

    while (naarKnoop != vanKnoop) {
        int k = 1;
        while (k - 1 < getAantalKnopen() && res[k - 1][naarKnoop - 1] == 0)
            k++;
    }
}
```

```
        pad.add(0, k);  
        naarKnoop = k;  
    }  
    return pad;  
}
```