

8

Floyd


Inleiding

Download het bestand `week08-Grafen-Floyd-opgave.zip` van Toledo. Importeer dit bestand in je IDE.

In deze oefenzitting leer je het algoritme van Floyd 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 8.1

 Een graaf bestaat uit zes knooppunten en wordt gegeven door zijn gewichtenmatrix

$$D^{(0)} = \begin{bmatrix} 0 & 7 & 3 & \infty & 5 & \infty \\ 2 & 0 & \infty & 12 & \infty & \infty \\ \infty & 3 & 0 & \infty & 1 & \infty \\ \infty & \infty & \infty & 0 & \infty & 6 \\ 5 & \infty & \infty & \infty & 0 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 \end{bmatrix}$$

1. Teken het netwerk.
2. Gebruik de methode van Floyd om de kortste paden tussen de verschillende punten van het netwerk te berekenen.

Oefening 8.2

  Bestudeer de klasse `WeightedGraph` die je in je IDE importeerde.

1. In tegenstelling tot de klasse `Graph` die we schreven bij BFS, is de instantieveranderlijke nu een matrix van `double` en niet van `boolean`. Waarom?

2. Wanneer is een gewichtenmatrix geldig?

Oefening 8.3



Schrijf de methode `int[][] findDistances()`. Deze methode berekent de pointermatrix P .

- De (lege) pointermatrix wordt gedeclareerd.
- Er wordt een kloon van de gewichtenmatrix gemaakt. Dit is een kopie die verwijst naar een andere geheugenplaats. Deze kloon is de D -matrix van de cursustekst.
- Voeg de knopen één voor één toe als tussenstation en pas de D -matrix aan zoals aangegeven in formule (2.3). Update de pointermatrix.

Verwachte uitvoer:

```
p_matrix:
0 0 4 0 4
5 0 0 0 4
5 5 0 0 4
5 5 0 0 0
0 1 4 1 0
```

Oefening 8.4



Schrijf `ArrayList<Integer> getShortestPath(int i, int j, int[][] path)`. Deze methode berekent uit de pointermatrix $path$ het kortste pad tussen knoop i en knoop j .

In deze oefening geldt dezelfde opmerking over de nummering van de knopen als we maakten in oefening 7.4. Het knooppunt dat laatst toegevoegd werd als tussenstation bij de berekening van het kortste pad tussen knooppunten i en j , is $path[i - 1][j - 1]$.

(Deel van) de verwachte uitvoer:

Kortste paden:

Er is geen pad van 1 naar 1

Kortste pad van 1 naar 2 lengte = 0 via : [1, 2]

Kortste pad van 1 naar 3 lengte = 0 via : [1, 4, 3]

Kortste pad van 1 naar 4 lengte = 0 via : [1, 4]

Kortste pad van 1 naar 5 lengte = 0 via : [1, 4, 5]

Kortste pad van 2 naar 1 lengte = 0 via : [2, 4, 5, 1]

Er is geen pad van 2 naar 2

Kortste pad van 2 naar 3 lengte = 0 via : [2, 3]

Kortste pad van 2 naar 4 lengte = 0 via : [2, 4]

Kortste pad van 2 naar 5 lengte = 0 via : [2, 4, 5]

Kortste pad van 3 naar 1 lengte = 0 via : [3, 4, 5, 1]

Kortste pad van 3 naar 2 lengte = 0 via : [3, 4, 5, 1, 2]

Er is geen pad van 3 naar 3

Kortste pad van 3 naar 4 lengte = 0 via : [3, 4]

Kortste pad van 3 naar 5 lengte = 0 via : [3, 4, 5]

Oefening 8.5



Schrijf de methode `int berekenLengte(ArrayList<Integer> pad)`. Invoer is `pad`: een opeenvolging van knopen die resulteert in het kortste pad tussen eerste en laatste element van `pad`. Uitvoer is de lengte van dit pad. Deze wordt berekend met behulp van de instantie- veranderlijke `gewichtenmatrix`. (Deel van) de verwachte uitvoer:

Kortste paden:

Er is geen pad van 1 naar 1

Kortste pad van 1 naar 2 lengte = 1 via : [1, 2]

Kortste pad van 1 naar 3 lengte = 3 via : [1, 4, 3]

Kortste pad van 1 naar 4 lengte = 1 via : [1, 4]

Kortste pad van 1 naar 5 lengte = 4 via : [1, 4, 5]

Kortste pad van 2 naar 1 lengte = 8 via : [2, 4, 5, 1]

Er is geen pad van 2 naar 2

Kortste pad van 2 naar 3 lengte = 3 via : [2, 3]

Kortste pad van 2 naar 4 lengte = 2 via : [2, 4]

Kortste pad van 2 naar 5 lengte = 5 via : [2, 4, 5]

Kortste pad van 3 naar 1 lengte = 10 via : [3, 4, 5, 1]

Kortste pad van 3 naar 2 lengte = 11 via : [3, 4, 5, 1, 2]

Er is geen pad van 3 naar 3

Kortste pad van 3 naar 4 lengte = 4 via : [3, 4]

Kortste pad van 3 naar 5 lengte = 7 via : [3, 4, 5]

Kortste pad van 4 naar 1 lengte = 6 via : [4, 5, 1]

Kortste pad van 4 naar 2 lengte = 7 via : [4, 5, 1, 2]

Kortste pad van 4 naar 3 lengte = 2 via : [4, 3]

Er is geen pad van 4 naar 4

Kortste pad van 4 naar 5 lengte = 3 via : [4, 5]

8 Floyd

Kortste pad van 5 naar 1 lengte = 3 via : [5, 1]

Kortste pad van 5 naar 2 lengte = 4 via : [5, 1, 2]

Kortste pad van 5 naar 3 lengte = 6 via : [5, 1, 4, 3]

Kortste pad van 5 naar 4 lengte = 4 via : [5, 1, 4]

Er is geen pad van 5 naar 5

Oplossingen

Oplossing 8.1 Gewichtenmatrix $D^{(6)}$ en bijhorende pointermatrix P :

$$D^{(6)} = \begin{bmatrix} 0 & 6 & 3 & 7 & 4 & 6 \\ 2 & 0 & 5 & 9 & 6 & 8 \\ 5 & 3 & 0 & 4 & 1 & 3 \\ \infty & \infty & \infty & 0 & \infty & 6 \\ 5 & 11 & 8 & 3 & 0 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 \end{bmatrix} \quad P = \begin{bmatrix} 0 & 3 & 0 & 6 & 3 & 5 \\ 0 & 0 & 1 & 6 & 3 & 5 \\ 2 & 0 & 0 & 6 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 1 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Enkele paden: $2 \rightarrow 1 \rightarrow 3 \rightarrow 5 \rightarrow 6$ met lengte 8; $5 \rightarrow 1 \rightarrow 3 \rightarrow 2 \rightarrow 4$ met lengte 3

Oplossing 8.2

1. Bij het algoritme van Floyd berekenen we paden met minste gewicht. We hebben daarom de gewichtenmatrix nodig. De elementen van die gewichtenmatrix zijn niet meer binair (0 of 1), maar kunnen alle positieve waarden aannemen.
2. Een gewichtenmatrix is geldig als er elementen in zitten en als hij vierkant is.

Oplossing 8.3

Listing 2 findDistances

```
public int[][] findDistances() {
    int aantal = this.gewichtenMatrix.length;
    int[][] P = new int[aantal][aantal];
    //double[][] D = this.gewichtenMatrix.clone(); fout = shallow clone
    //http://stackoverflow.com/questions/9106131/how-to-clone-a-multidimensional-array-in-java
    //of manuele versie in de nieuwe opgave op toledo
    //argument voor deze clone: is gezien in OOP
    double[][] D = this.gewichtenMatrix.clone();
    for (int i = 0; i < D.length; i++) {
        D[i] = D[i].clone();
    }

    for (int k = 0; k < aantal; k++) {
        for (int i = 0; i < aantal; i++) {
            for (int j = 0; j < aantal; j++) {
                if (D[i][k] + D[k][j] < D[i][j]) {
                    D[i][j] = D[i][k] + D[k][j];
                    P[i][j] = k + 1;
                }
            }
        }
    }
}
```

Oplossingen

```
    return P;
}
```

Oplossing 8.4

Listing 3 getShortestPath

```
public List<Integer> getShortestPath(int van, int tot, int[][] P) {
    List<Integer> pad = new ArrayList<>();
    if (van == tot) {
        return pad;
    } else {
        int via = P[van - 1][tot - 1];
        if (via == 0){
            pad.add(van);
            pad.add(tot);
        } else {
            pad = getShortestPath(van, via, P);
            pad.remove(pad.size() - 1); //anders dubbel
            pad.addAll(getShortestPath(via, tot, P));
        }
    }

    return pad;
}
```

Oplossing 8.5

Listing 4 berekenLengte

```
public int berekenLengte(List<Integer> pad) {
    int som = 0;
    int aantalKnopen = pad.size();
    int huidigeKnoop, volgendeKnoop;

    for (int i = 0; i < aantalKnopen - 1; i++) {
        huidigeKnoop = pad.get(i);
        volgendeKnoop = pad.get(i + 1);
        som += this.gewichtenMatrix[huidigeKnoop - 1][volgendeKnoop - 1];
    }

    return som;
}
```