Containing

Door:

Erwin Hager

Melvin Kool

Eelke Fierstra

Jeffery Verbeek

Klaas Bakker

Inhoudsopgave

[Evaluatie 2](#_Toc442098703)

### 

# Introductie

Project containing is een project van het tweede jaar Informatica van de NHL. Het doel van project containing is om een simulatie te maken van de Rotterdamse haven. Deze simulatie bestaat uit een app, controller en een 3D weergave. We hebben hier 8 weken de tijd voor gekregen.

In dit verslag zullen we jullie vertellen wat dit project precies inhoud, de techniek en algoritmes die er achter zitten, hoe het project verliep en een korte nabeschouwing. We hopen dat we jullie mee kunnen nemen in ons project.

**Samenvatting van de opdracht**

Zoals verteld in de introductie is de opdracht om een simulatie te maken met een 3d weergave, een app en een controller. Deze programma's moeten in Java en C++ geschreven worden. Ook moeten deze programma's getest worden, zo dat de opdrachtgever overtuigd raakt van de kwaliteit.

**Simulator**

De simulator moet gemaakt worden in 3D, waarbij gebruik moet worden gemaakt van de game-engine jMonkey.

**Controller**

De controller moet XML-bestanden verwerken die we aangeleverd krijgen. Deze bestanden bevatten informatie over containers, zoals aankomst- en vertrektijden, inhoud van de container enz. In deze XML-bestanden kunnen fouten zitten, waar wij op moeten controleren, zodat ons programma er geen last van heeft.

**App**

De app moet gemaakt worden voor een mobile platform (Android of iOS). In deze app moeten actuele aantallen containers per categorie getoond worden, dus waar ze zich op dat moment bevinden.

**Hoe wij het aanpakken**

**Simulator**

De simulator willen wij zo 'dom' mogelijk houden. Dit betekend dat deze alleen de huidige stand van zaken afbeeld. De simulator ontvangt commando's en gegevens via de server(controller). Een simulator kan elk moment verbinden met de server. De simulator is geschreven in java, met als game-engine jMonkey.

**Controller**

In onze controller stoppen we de server. Dit is het makkelijkst, omdat in ons geval alle logica in de controller zit en dan meteen de simulator kan aansturen. De controller is geschreven in c++. Deze bestaat uit verschillende parsers en schrijvers, een webserver, een server voor het versturen en ontvangen van data en vooral de logica die onder andere samenwerkt met onze database.

**App**

De app maken wij in HTML, CSS en JavaScript. Dit is een responsive webapp, die vriendelijk is voor alle platforms en alle devices. Hierdoor maakt het niet uit of je Linux of iOS gebruikt, of je een groot of een klein scherm hebt, en ook bijna niet welke webbrowser je gebruikt. Onze app laat live statistieken zien met behulp van AJAX en ChartJS. De app haalt al deze gegevens op van de webserver die in onze controller verwerkt zit.

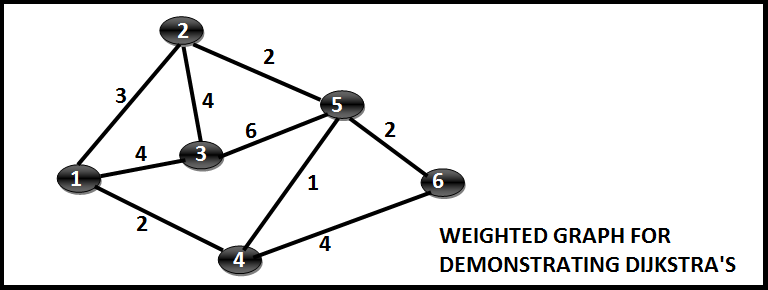
**Kortstepad**

In onze server voeren wij alle kortstepad berekeningen uit. Voor ons kortstepad algoritme gebruiken wij het algoritme van Dijkstra. Met dit algoritme reken je de kortste afstand uit tussen twee punten. Hier hebben we gekozen, omdat ons wegenstelsel vooral bestaat uit knooppunten, waar het algoritme van dijkstra zeer geschikt voor is! Dit levert je dus een snel algoritme op, wat door veel voertuigen tegelijk gebruikt kan worden.

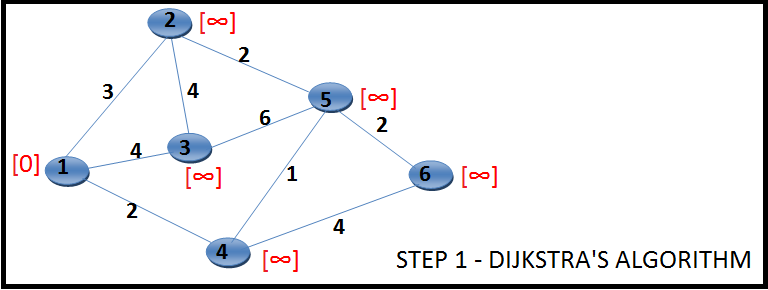
Hier een uitleg hoe het algoritme van dijkstra ongeveer werkt en hoe dit algoritme bij ons gebruikt wordt:

Om het algoritme van Dijkstra schematisch weer te geven zie je hier een voorstelling van een wegenstelsel. Je hebt hier punt 1,2 .. t/m 6. De afstand tussen punt 1 en punt 2 is bijvoorbeeld 3 en de afstand tussen punt 1 en punt 3 is 4.

Laten we als voorbeeld nemen de kortste route tussen punt 1 en punt 6.

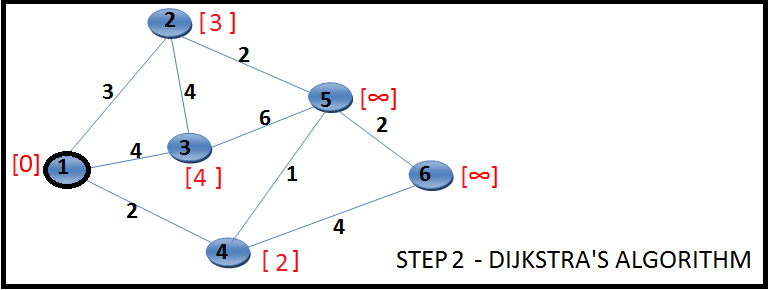
Bron afbeelding: <http://techieme.in/shortest-path-using-dijkstras-algorithm/>

Als eerste kies je je startpunt, in ons geval punt 1. De afstand naar punt 1 kun je op 0 zetten, aangezien dit ons startpunt is. Punt 1 is dan klaar, want je weet de kortste route naar punt 1. Hier kun je dan bijvoorbeeld een cirkel omheen tekenen, wat betekent dat dit punt klaar is. Vervolgens zet je de afstanden naar alle punten op oneindig, aangezien de route van punt 1 naar punt 2 langer had kunnen zijn dan de route van punt 1 naar punt 2 via punt 3.



Bron afbeelding: <http://techieme.in/shortest-path-using-dijkstras-algorithm/>

Vervolgens bereken je de afstand van alle aanliggende punten vanaf het punt met de kleinste afstand (als je begint punt 1 met de afstand van 0). De afstand tussen punt 1 en punt 2 is dan de kleinste afstand naar het huidige punt(punt 1 met afstand 0) 0 + de afstand naar het punt(punt 2) 3 = 3. De voorganger zet je dan op je huidige punt(punt 1) als de afstand naar het punt dat we bekijken(punt 2, afstand = 3) kleiner is vanaf het huidige punt dan de eerder berekende afstand naar het punt(afstand = oneindig), dit is later nodig voor het herleiden van de route vanaf het eindpunt(punt 6). De afstand tussen punt 1 en punt 3 is 4, de voorganger is punt 1. De afstand tussen punt 1 en punt 4 is 2, de voorganger is hier ook weer punt 1.

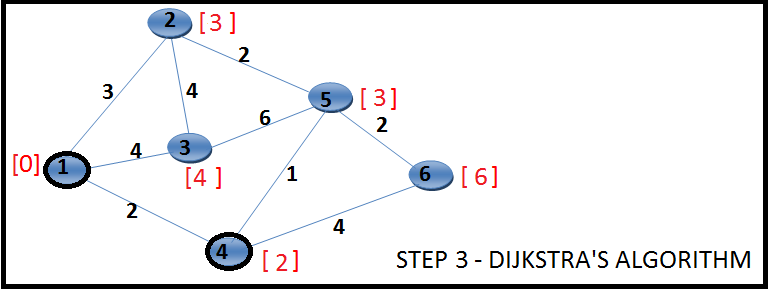
Bron afbeelding: <http://techieme.in/shortest-path-using-dijkstras-algorithm/>

Vanaf 1

Vanaf 1

Vanaf 1

Vervolgens kijk je welk punt de kleinste totale afstand heeft vanaf het beginpunt en nog niet klaar is, dat is in ons geval punt 4 met een totale afstand van 2. Dit punt is dan klaar, dus zet je er een cirkel omheen. Dit punt wordt dan je huidige punt. Hierna ga je de vorige stappen steeds herhalen. We bekijken eerst weer de afstanden naar de andere punten(punt 5 en punt 6). Deze afstanden zijn kleiner dan oneindig, dus stellen we de voorganger in op het huidige punt(punt 4). Hierna zoeken we het punt met de kleinste totale afstand uit en stellen dit punt in als ons huidige punt. Je ziet nu dat er twee mogelijkheden zijn, punt 2 of punt 5. In ons geval kiezen we punt 2.

Bron afbeelding: <http://techieme.in/shortest-path-using-dijkstras-algorithm/>

Vanaf 4

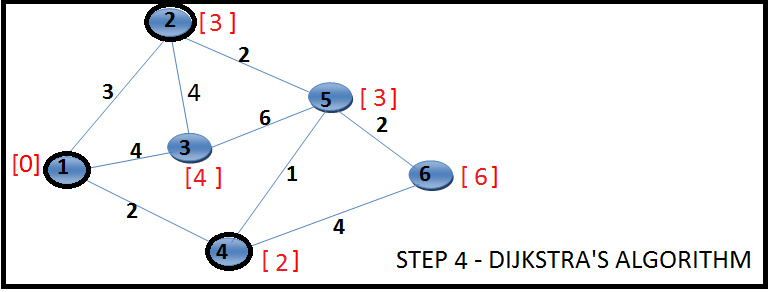
Vanaf 4

Vanaf 1

Vanaf 1

Vanaf 1

Punt 2 is nu klaar. De afstand van punt 2 (3) naar punt 5 (3 + 2 = 5) is groter dan 3, dus stellen we hier geen nieuwe voorganger in. Met punt 3 hetzelfde.



Vanaf 1

Vanaf 1

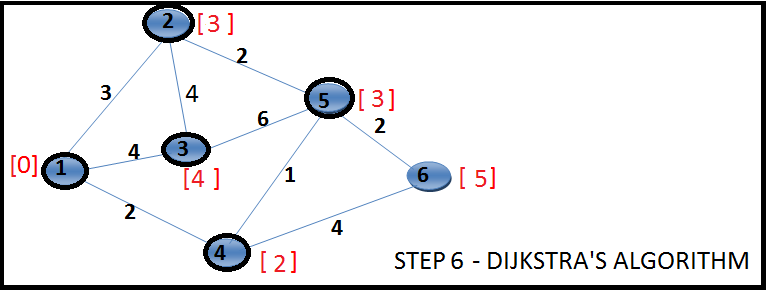
Vanaf 1

Vanaf 4

Vanaf 4

Bron afbeelding: <http://techieme.in/shortest-path-using-dijkstras-algorithm/>

Hierna zoeken we het punt dat nog niet klaar is en de kleinste totale afstand heeft. Dit is punt 5. Punt 5 is dan ook klaar. De afstand van punt 5 naar punt 3(3 + 6 = 9) is langer dan de huidige afstand(4), dus de voorganger veranderen we niet. De afstand van punt 5 naar punt 6(3 + 2 = 5) is korter dan de huidige afstand(6), dus we zetten de voorganger van punt 6 op punt 5 en de afstand op 5.

Bron afbeelding: <http://techieme.in/shortest-path-using-dijkstras-algorithm/>

Vanaf 1

Vanaf 1

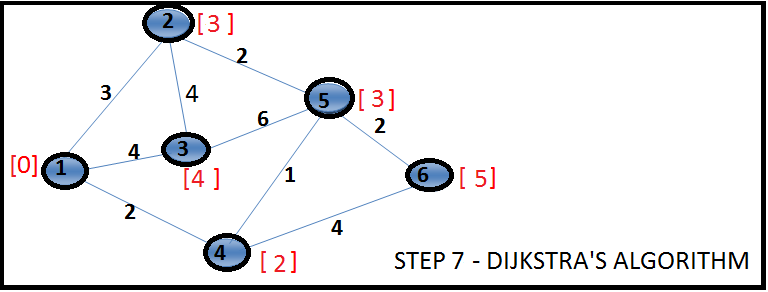
Vanaf 1

Vanaf 4

Vanaf 5

Nu zoeken we weer het punt met de kortste afstand die nog niet klaar is, dat is punt 3. Punt 3 zetten we op de huidige en die is klaar. Punt 3 heeft geen aanliggende punten die nog niet klaar zijn, dus kunnen we verder.

Nu zoeken we het kleinste punt dat nog niet klaar is en de kleinste totale afstand heeft.

Dit is punt 6, dus stellen we punt 6 in als ons huidige punt. Punt 6 is klaar, we hebben ons doel nu bereikt.

Vanaf 1

Vanaf 4

Vanaf 5

Vanaf 1

Vanaf 1

Bron afbeelding: <http://techieme.in/shortest-path-using-dijkstras-algorithm/>

We herleiden nu de kortste route. We zijn naar 6 gekomen vanaf 5 → 5,6. We zijn naar 5 gekomen via 4 → 4,5,6. We zijn naar 4 gekomen vanaf 1 → 1,4,5,6. We zijn nu bij ons beginpunt, dus hebben we de kortste route gevonden: 1,4,5,6.

Hier een voorbeeld in pseudocode hoe je dit zou kunnen programmeren:

|  |  |  |
| --- | --- | --- |
| 1: | **function** Dijkstra(Graph, source): |  |
| 2: | **for each** vertex v in Graph: | // Initialization |
| 3: | dist[v] := infinity | // initial distance from source to vertex v is set to infinite |
| 4: | previous[v] := undefined | // Previous node in optimal path from source |
| 5: | dist[source] := 0 | // Distance from source to source |
| 6: | Q := the set of all nodes in Graph | // all nodes in the graph are unoptimized - thus are in Q |
| 7: | **while** Q **is not** empty: | // main loop |
| 8: | u := node in Q with smallest dist[ ] |  |
| 9: | remove u from Q |  |
| 10: | **for each** neighbor v of u: | // where v has not yet been removed from Q. |
| 11: | alt := dist[u] + dist\_between(u, v) |  |
| 12: | **if** alt < dist[v] | // Relax (u,v) |
| 13: | dist[v] := alt |  |
| 14: | previous[v] := u |  |
| 15: | **return** previous[ ] |  |

Bron: <http://www.gitta.info/Accessibiliti/en/html/Dijkstra_learningObject1.html>

# Evaluatie

Het doel van het project was om een werkende simulatie te maken van een haven. Hierbij moet XML worden ingeladen. Uit deze XML moeten de voertuigen worden aangestuurd. De agv’s moeten de containers bij de schepen, trucks, en treinen weghalen. De containers worden via kranen geladen en gelost. De containers worden opgeslagen op het opslagterrein. Deze moeten op datum worden gesorteerd.

Tijdens Containing liepen we tegen enkele problemen aan. Deze konden opgelost worden door de hulp van een “expert” of medestudent.

Bijna iedere week hebben we een gesprek met onze tutor/opdrachtgever(Wouter) gehad. Tijdens het gesprek besproken we de voortgang, de sprints, de backlog en de samenwerking in de groep.

Het gebruik van Git via de commandline is een leer geweest voor ons allemaal. Daarnaast hebben we natuurlijk ook geleert van het programmeren in Java en C++. Ook het samenwerken in een groep heeft ons allemaal weer wat bijgebracht.

Al met al zijn we straks blij als we het project klaar hebben aangezien het niet ons favoriete is. Verder ging het project wel goed. Het was alleen te groot om in 8 weken af te krijgen.