Sorting and Searching, practicum 2

Zoek het kortste pad in een 'tile based world'

Kostis Thanos
IS-204

Inhoudsopgave

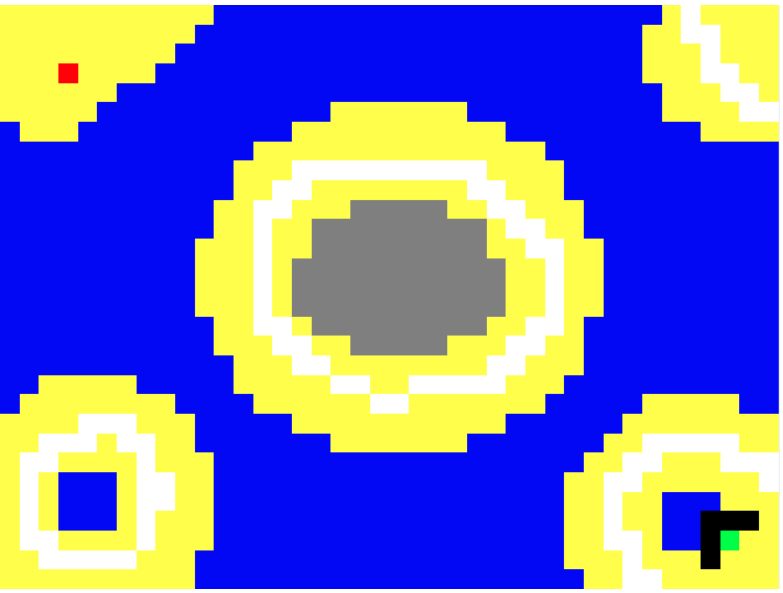
Inleiding

In opdracht van de Hogeschool van Amsterdam moet ik (Kostis Thanos) voor het vak Sorting and Searching een aantal opdrachten maken en die vervolgens weer verwerken in verslagen. Dit verslag betreft opdracht 2, waarin ik twee algoritmes test die allebei d.m.v. wereld- en blokklassen een plaatje kunnen herkennen en kunnen verwerken tot een wereld van blokken, die vervolgens ge-analyseerd kan worden om zo het kortste pad te vinden in die wereld. Ik zal in dit verslag beschrijven welke code ik in beide algoritmes toevoeg om de lengte en de kosten van het kortste pad te vinden, en om het aantal onderzochte knopen te vinden. Veel plezier met lezen!

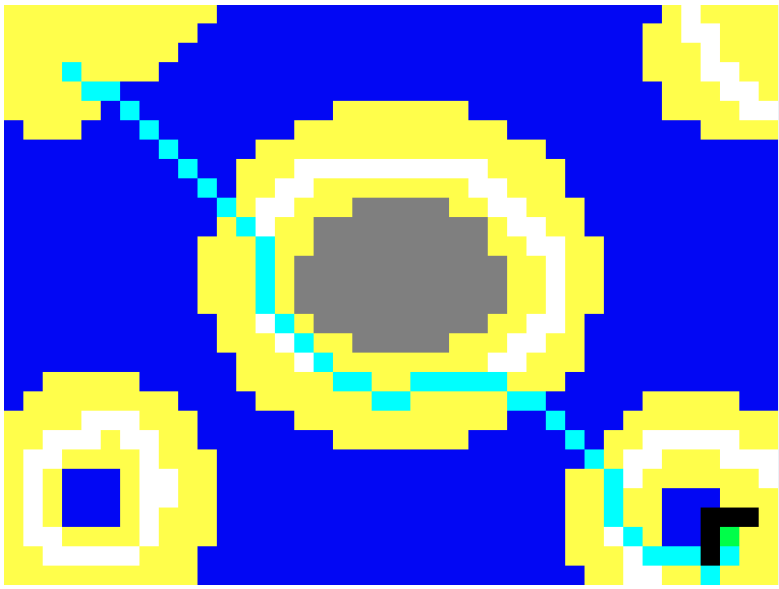
Dijkstra - Oplossingen bijgeleverde testset & 3 eigengemaakte werelden

Hieronder zijn 3 oplossingen voor 3 willekeurige tileworlds van de bijgeleverde testset.

I-03

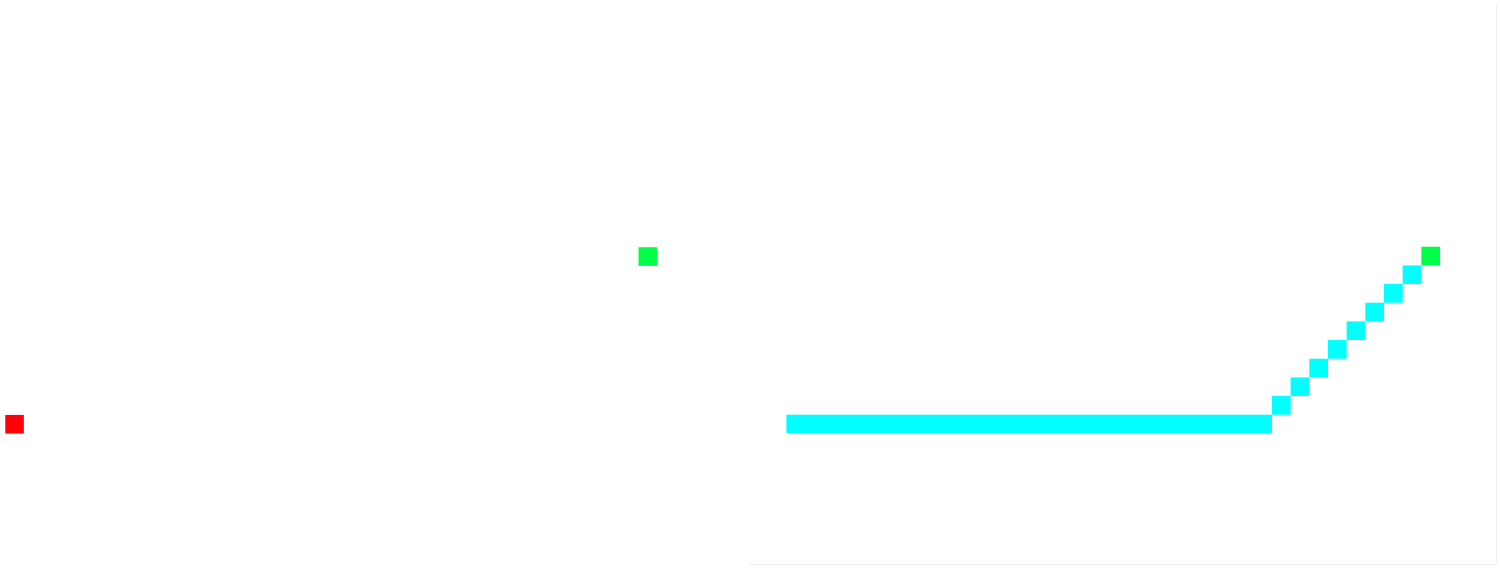


I-03 uitwerking Dijkstra



I-14

I-14 uitwerking Dijkstra



Code Aantal onderzochte knopen & en lengte kortste pad

De relax methode onderzoekt of de huidige kortste afstand naar het punt waar je naartoe gaat (w) vanaf het begin groter is dan de afstand van het nieuwe punt waar je vandaan komt plus het gewicht daarvan. Dan wordt de kortste afstand daarop gezet en de kortste connectie ook (edgeTo).

Ik zou zeggen dat in dit if statement dus twee knopen worden onderzocht.

```
private void relax(DirectedEdge e) {
   aantalOnderzochteKnopen++;
    int v = e.from(), w = e.to();
    if (distTo[w] > distTo[v] + e.weight()) {
        distTo[w] = distTo[v] + e.weight();
        edgeTo[w] = e;
        if (pq.contains(w)) pq.decreaseKey(w, distTo[w]);
        else                pq.insert(w, distTo[w]);
    }
}
```

In de code hieronder (die in de main staat) wordt de methode hasPath aangeroepen om te zien of er dus een pad is naar het einde. Als het pad eenmaal is gevonden, dan ga ik de lijst af met de hasNext()-methode tot het einde om te zien hoeveel DirectedEdges (tiles) er in het kortste pad zitten (de waarde staat al op één omdat als je begint met het vergelijken van twee tiles er dus ook al twee zijn).

De kosten worden gevonden door ook in de hasNext()-methode het gewicht van de huidige DirectedEdge op te tellen tot de loop stopt.

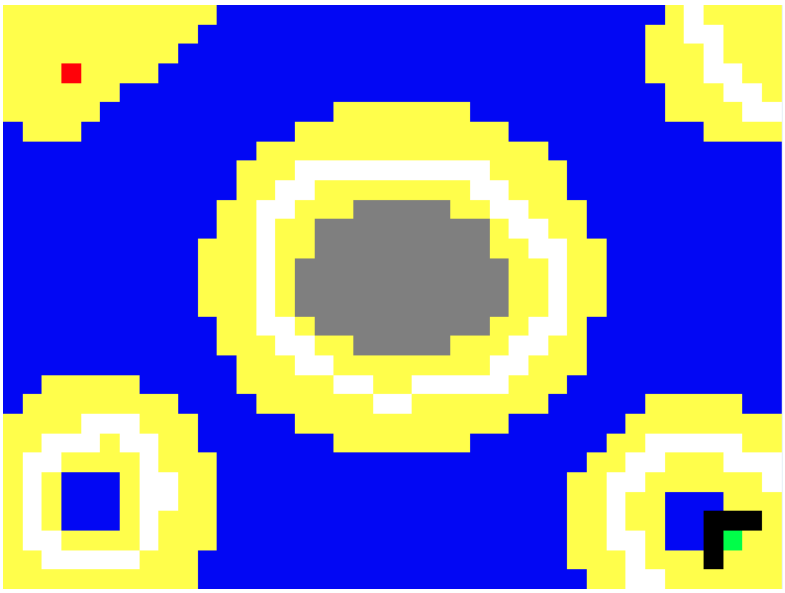
```
EdgeWeightedDigraph eD = new EdgeWeightedDigraph("i" + i);
Dijkstra dijkstra = new Dijkstra(eD, eD.getStart());


if (dijkstra.hasPathTo(eD.getEnd())){
	Iterable<DirectedEdge> antwoord = dijkstra.pathTo(eD.getEnd());
	System.out.println("Aantal onderzochte knopen: " +
	dijkstra.getAantalOnderzochteKnopen());
	Iterator it = antwoord.iterator();
	int totalWeightDijkstra = 0;
	int totalLengthDijkstra = 1;
	while(it.hasNext()){
		DirectedEdge dE = (DirectedEdge)it.next();
		totalWeightDijkstra += dE.getWeight();
		totalLengthDijkstra ++;
	}
	System.out.println("Lengte van het kortste pad" + i + "(met Dijkstra)
	is: " + totalLengthDijkstra);
	System.out.println("Kosten van het kortste pad" + i + "(met Dijkstra)
	zijn: " + totalWeightDijkstra);
	eD.tekenPad(antwoord);
	eD.save("i" + i + "-Dijkstra");
}
```

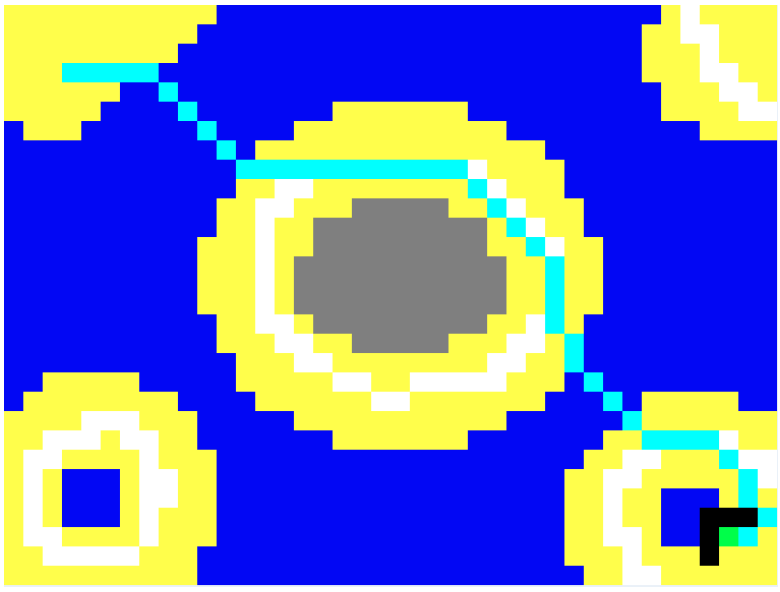
Floyd-Warshall - Oplossingen bijgeleverde testset & 3 eigengemaakte werelden

De Floyd-Warshall komt soms met een ander uitkomst plaatje (omdat het achteraf blijkt dat de kosten hetzelfde zijn als bij de Dijkstra uitwerking, dus dat is correct en interessant!)

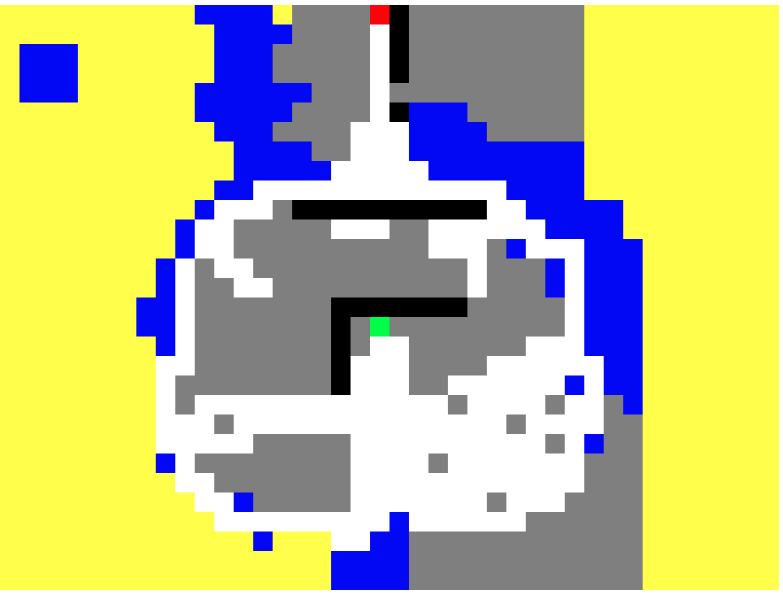
I-03



I-03 Uitwerking Floyd Warshall



I-14



I-14 Uitwerking Floyd Warshall

Code Aantal onderzochte knopen & en lengte kortste pad

In Floyd Warshall gebeurt in het algemeen hetzelfde. De hasPath-methode wordt aangeroepen met als input de start en de eind tiles. Ik heb ook hier het aantalTiles al op één gezet.

```
EdgeWeightedDigraph eDFloyd = new EdgeWeightedDigraph("i" + i);
AdjMatrixEdgeWeightedDigraph adjMEWD =
eDFloyd.createAdjMatrixEdgeWeightedDigraph();

FloydWarshall fW = new FloydWarshall(adjMEWD);
      if(fW.hasPath(eDFloyd.getStart(), eDFloyd.getEnd())){
            int aantalTiles = 1;
            Iterable<DirectedEdge> antwoord = fW.path(eDFloyd.getStart(),
            eDFloyd.getEnd());
            Iterator it = antwoord.iterator();
            while(it.hasNext()){
                  DirectedEdge dE = (DirectedEdge)it.next();
                  aantalTiles++;
            }
```

Resultaten & conclusies

Dijkstra

(De plaatjes zijn in de vorige pagina's al laten zien)

Bitmap	Knopen	Lengte	Kosten
I03.png	9184	41	656
I14.png	9184	23	320
I15.png	9184	35	376

Floyd Warshall

Bitmap	Knopen	Lengte	Kosten
I03.png	1836302400	44	656
I14.png	1757887200	23	320
I15.png	1840728000	35	376

Wat opvalt is dat het Dijkstra algoritme drie keer hetzelfde aantal knopen geeft. (of dit goed is weet ik niet, maar bij andere bitmaps geeft hij ook andere getallen aan). En natuurlijk worden er bij Floyd Warshall veel meer knopen vergelijken omdat hij vanaf het beginpunt naar elk ander punt het kortste pad zoekt, dus dat is logisch. De conclusie is dat voor dit soort bitmaps Dijkstra sneller en dus efficienter is.