Sorting and Searching, practicum 2

Zoek het kortste pad in een 'tile based world'

Inleiding

De tile based world

De 'tile based world' is een twee dimensionale kaart, die wordt weergegeven door een bitmap (.png) van 40x30 tegels (=tiles). De wereld heeft begaanbare en onbegaanbare tegels. In dit laatste geval betekent het dat er een obstakel is dat niet te nemen is. In dat geval kun je deze tegel dus niet opnemen in je kortste pad. In de bitmap hebben de tegels verschillende kleuren, afhankelijk van de energie (=cost) die het kost om deze tegel te doorlopen. Er zijn vier kleuren: een weg (*White*), zand (*Yellow*), water (*Blue*) en bergen (*Grey*). In de onderstaande tabel staat hoe je deze kleuren kunt vertalen naar cost.

Walkable type	Color	Cost	Diagonal
Road	White	10	14
Sand	Yellow	14	20
Water	Blue	20	28
Mountain	Grey	24	34

De energie die het kost om van *punt A* naar het naastliggend *punt B* te reizen, hangt alleen af van de cost van *punt B* (=het punt waar je naar toe gaat). Als je horizontaal of verticaal door de wereld reist gebruik je de cost uit kolom 3 (Cost), ga je diagonaal (bijvoorbeeld van *(x,y)* naar *(x+1,y+1)*) dan gebruik je de cost uit kolom 4 (Diagonal).

Het startpunt is *Red* in de bitmap, het eindpunt *Green*. Als je een kortste pad hebt gevonden teken je dit in de bitmap in *Cyan* (zie onderstaande tabel).

Tile type	Color
Non-walkable	Black
Start	Red
End	Green
Node on the path	Cyan

Aangeleverde resources

Om je te helpen is er een Maven project `PathFinding-Student.zip` op de VLO met daarin alles wat je nodig hebt om te kunnen beginnen.
In de folder `src/main/resources/input` staan alle 21 testwerelden die je bij je experiment moet gebruiken.

Een aantal opmerkingen/tips over de aangeleverde classes

- **De class `EdgeWeightedDigraph` is in staat om de maps direct in te lezen**! Je hoeft zelf geen code te schrijven om de graaf op te bouwen.
- De graaf met de bijbehorende adjacency list wordt beschreven in de API (javadoc)van de klasse `EdgeWeightedDigraph`. Deze maakt een Array aan van het type `LinkedList<DirectedEdge>`.
- In de vertaling van de (x,y)-coordinaten en terug wordt voorzien door de methodes `getX()`, `getY()` en `getIndex()`.
- Het begin- en eindpunt van je pad kun je opvragen door respectievelijk `getStart()` en `getEnd()`.
- Verder kan de graaf worden gegenereerd met de methode `load()`, bewaard met de methode `save()` en op het scherm worden getoond met de methode `show()`. **N.B. Voordat je een graaf gaat opslaan als afbeelding zorg dat `TileworldUtil.outputDir` naar een bestaande folder verwijst!**
- `Dijkstra` bepaalt het kortste pad vanuit één knoop naar alle andere knopen, `FloydWarshall` bepaalt de kortste paden van alle knopen naar alle andere knopen.
- In de beide classes zijn methodes opgenomen op het juiste pad van het startpunt naar het eindpunt op te vragen. Dit pad kun je in de graaf tekenen in de juiste kleur met de methode `tekenPad()`.
- De constructor van `Dijkstra` heeft een `EdgeWeightedDigraph` nodig. De constructor van `FloydWarshall` een `AdjMatrixEdgeWeightedDigraph`, die je kunt genereren met de methode `createAdjMatrixEdgeWeightedDigraph`.
- Er is een methode om een (kortste) pad te tekenen in de graaf en er zijn twee methodes om de gehele adjacency list en om de `DirectedEdges` op te vragen die vanuit een knoop te benaderen zijn.

De opdracht

De opdracht bestaat uit twee delen.

a. Dijkstra

Gebruik het algoritme van Dijkstra om het kortste (=goedkoopste) pad te bepalen van een aantal werelden uit de bijgeleverde testset. (i1.png,...i21.png) en een aantal (minimaal 3) zelfgemaakte werelden. Bepaal ook het aantal onderzochte knopen (hiertoe dien je de code aan te passen door een teller toe te voegen) en de lengte (aantal tiles) van het kortste pad.

De geleverde werelden in de testset zijn allemaal 30x40 (=1200 knopen). De werelden die je zelf maakt kunnen even groot zijn, maar ook grotere en kleinere werelden mogen. Deze werelden kun je bijvoorbeeld ontwerpen door een aantal werelden uit de testset te combineren. Ook kan je van b.v. https://www.openstreetmap.org/ gegevens exporteren en daar een kaart van maken. En vervolgens het algoritme de optimale route van je huis naar de HvA laten bepalen.

b. Floyd-Warshall en/of A*

Doe dezelfde exercitie als bij Dijkstra voor

- ofwel Floyd-Warshall (te vinden in de bijgeleverde bibliotheek)
- ofwel A* (deze code zal je zelf moeten schrijven)
- ofwel nog mooier: voor beide algoritmen

Resultaten en conclusies

Vergelijk de resultaten van alle algoritmes met elkaar (in tabellen en in drie grafieken).

Toon je resultaten per algoritme in de volgende tabel.

Bitmap	Knopen	Lengte	Kosten	Oplossing
i01.png				
i02.png				
…				
i21.png				
Eigen bitmaps				

Bitmap is de naam van de onderzochte bitmap.
Knopen is het totaal aantal onderzochte knopen.
Lengte is de lengte (aantal tiles) van het kortste pad.
Kosten zijn de kosten van het kortste pad.
Oplossing is een verkleind plaatje van de bitmap met het kortste pad erin getekend.

Om de onderzochte algoritmes goed te kunnen vergelijken maak je ook drie grafieken:

1. Een grafiek voor *Knopen*,
2. Een grafiek voor *Lengte* kortste pad,
3. Een grafiek voor *Kosten* kortste pad.

Voor iedere grafiek geldt: x-as bevat de bitmaps, y-as bevat de gemeten waarde, en in de grafiek staan lijnen voor de onderzochte algoritmes.

Trek conclusies over je meetresultaten. Vergelijk de algoritmes met elkaar.