Opgave 2: De reisplanner

Leerdoelen

Na afloop van deze opdracht ben je in staat om:

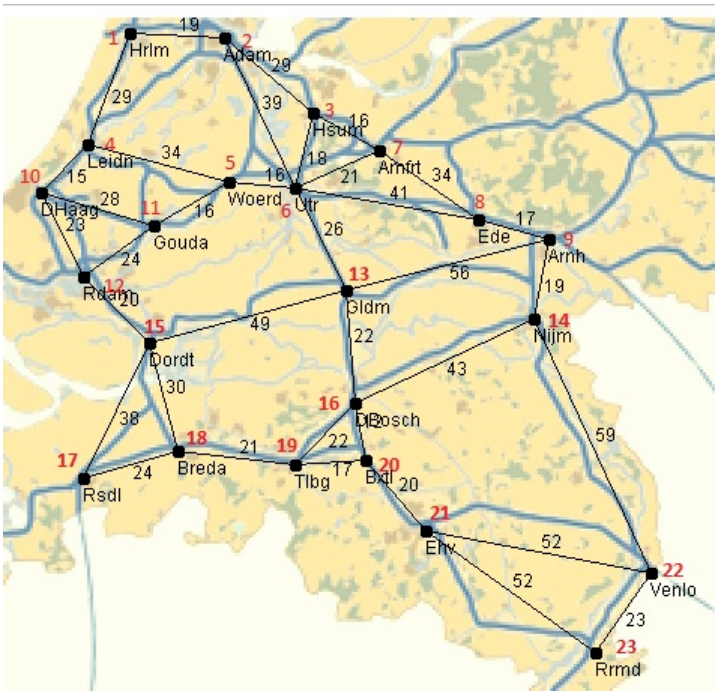
- in Java een abstracte klasse of interface te definiëren;
- instanties hiervan te definiëren en te gebruiken;
- lists en iterators in Java te gebruiken;
- een greedy algoritme te implementeren.

Achtergrond

Vanaf deze opgave zullen we in Java werken omdat dit simpeler is dan in C++. Houd je aan de layoutregels voor Java zoals die op de site staan. Deze opgave is voornamelijk bedoelt om je Java kennis op te frissen.

Probleemschets

Toepassingen als de TomTom, OV-9292, en de NS-reisplanner maken gebruik van algoritmen om de kortste route te vinden. Er bestaan verschillende algoritmen om het kortste route te vinden. In deze opgave zullen we een greedy algoritme gebruiken om voor het een netwerk van verbindingen de kortste route te vinden.



1 Representatie van de verbindingen

In deze opgave zullen we de lengte van verbindingen representeren als integers. Als er geen verbinding (bekend) is tussen twee stations zullen we een groot getal gebruiken om 'oneindig' aan te geven, b.v.:

```
public final int INF = 1000000;
```

De simpelste representatie van de een netwerk is een twee dimensionale matrix met afstanden:

```
private int [ ] [ ] afstand;
```

Voor een netwerk van behoorlijke afmetingen is het onhandig om de inhoud van deze matrix in je programma te coderen. Op Bb vind je code die zo'n matrix inleest uit een gegeven file.

De file `sporen.txt` bevat de afstanden tussen de boven getekende Nederlandse stations. Verbindingen worden slechts één maal genoemd in deze file, maar de aanname is dat alle sporen in twee richtingen te gebruiken zijn. De stations zijn genummerd om het zo simpel mogelijk te houden. Helaas lopen de nummers in deze file (en het corresponderende plaatje hierboven) vanaf 1, en de indices in de matrix vanaf 0. We zullen daarom van elk stationnummer in het interface 1 aftrekken voor intern gebruik in het programma.

Als er iets fout gaat met het inlezen van de file, of het raadplegen van de verbindingen wordt er een exception gegooid. Het is de bedoeling dat je programma deze vangt en de boodschap netjes afdrukt.

Deze representatie is natuurlijk erg inefficiënt. Voor N stations hebben we $O(N^2)$ ruimte nodig. Om dit later eenvoudig te kunnen verbeteren dien je aan abtsracte klasse of een interface `Netwerk` te definiëren. In je kortste-route algoritme gebruik je deze abstracte klasse/interface zodat het eenvoudig is een efficiëntere representatie van het netwerk te gebruiken.

2 Kortste route

Voor alle stations houden we de kortst bekende afstand tot het begin station bij. We gebruiken hier natuurlijk een rij van afstanden voor. Initieel zijn deze allemaal oneindig. Alleen voor het beginstation weet je natuurlijk dat die afstand 0 is.

We zoeken een route door te beginnen bij de station waar de reiziger wil vertrekken. We 'kleuren' dit station (bijvoorbeeld door een boolean op **true** te zetten). Zolang we nog niet het doel-station gekleurd hebben herhalen we de volgende stappen:

1. Bekijk voor alle station of de route via het laatst gekleurde station korter is dan de lengte van de verbinding die tot nu toe bekend is. De lengte van de verbinding naar station s via het laatst gekleurde station is de som van de lengte van de route tot dat laatst gekleurde station (uit de rij met kortst bekende afstanden) en de afstand van dat laatst gekleurde station tot station s (te vinden in de verbindingsmatrix).

   Als de bekeken route korter is passen we de rij met kortst bekende afstanden aan: de nieuw bekeken route is immers korter.

2. We bepalen nu welk station we vervolgens gaan kleuren. Dit is het station dat het dichts bij het vertrekpunt ligt, maar dat zelf nog niet gekleurd is. Dit station kunnen we eenvoudig vinden door één keer door de rij met kortst bekende afstanden te lopen.

   Als we alle stations in de rij bekeken hebben, is het duidelijk welk station niet gekleurd is en het dichtst bij het vertrekpunt ligt. We kleuren dit station en gaan verder met deze herhaling.

Met dit greedy algoritme vinden we gegarandeerd de kortste route van begin naar- eind-station. De lengte van deze route kunnen we aflezen uit de rij met kortst bekende verbindingen.

Implementeer dit algoritme in Java. Het netwerk van verbindingen wordt uit een vaste file gelezen. De gebruiker wordt gevraagd om begin en eindpunt. Druk de lengte van de route af. Deze lengte kun je aflezen uit de rij met kortste verbindingen per station.

Merk op dat de afstandsmatrix zelf nooit verandert zal worden in dit algoritme. Je kunt dus veilig meerdere afstanden berekenen met een verbindingsmatrix.

3 De route

Om de route zelf af te drukken moeten we nog wat meer administratie bij houden. We kunnen dit efficient doen door in een rij van ieder station bij te houden op welk station we daarvoor waren. Initieel weten we natuurlijk niet waar we daarvoor waren. Codeer dit met een niet bestaand station nummer als -1.

Op het moment dat we in bovenstaande herhaling een kortere route naar stattion s vinden weten we ook dat deze route via het laatst gekleurde station gaat. We administreren dit laatste gekleurde station als

voorganger van s op de route. De rest van de route kun je vinden door de voorganger van de voorganger te bekijken.

Als bovenstaande loop termineert vinden we de route door vanaf het eindpunt steeds de voorganger te zoeken totdat we aan het beginpunt zijn.

Implementeer deze uitbreiding en zorg dat het programma behalve de lengte van de verbinding ook de route (natuurlijk in de juiste volgorde) afdrukt. Om de stationsnummers in de wereld van de gebruiker te krijgen moet je er steeds 1 bij optellen.

Hint: recursie is je vriend bij het in de juiste volgorde afdrukken van stations.

4 Complexiteit van het algoritme

Wat is de complexiteit van het geheugengebruik van dit programma als functie van het aantal stations N? Schrijf het antwoord met een korte motivatie als commentaar bij het algoritme.

Wat is de runtime complexiteit van dit algoritme?

5 Efficiente representatie van het netwerk

In de gebruikte representatie kunnen we voor ieder station N verbindingen op slaan. De meeste stations hebben slechts drie echte verbindingen. Alle andere verbindingen hebben de waarde oneindig (`INF`). Bovendien is het natuurlijk overbodig om een verbinding van station x naar station y zowel bij x als bij y op te slaan. We kunnen het aantal opgeslagen verbindingen halveren door de verbinding alleen op te slaan bij het station met het kleinste nummertje.

Maak een nieuwe implementatie van `Verbinding` die alleen alle echte verbindingen opslaat. Per station houden we een Java `List` van verbindingen bij. Om deze lijst te bekijken of te veranderen gebruiken we een `Iterator`.

6 Complexiteit van deze representatie

Wat is de geheugen complexiteit van deze representatie?

Inleveren van je producten

Als producten heb je (uitgeteste!) Java-code met het antwoord op de vragen over complexiteit als commentaar toegevoegd. Neem de uitvoer van je programma voor de kortste verbinding van Nijmegen (station 14) naar Den Haag (station 10) als commentaar op in je code.

Lever vóór maandag 14 februari 8.30 uur via Blackboard jullie Java code in. Vergeet niet om **in** je uitwerking duidelijk jullie namen te vermelden als JavaDoc.

Als je energie en tijd over hebt mag je natuurlijk de tabel met verbindingen uitbreiden met nieuwe stations en verbindingen.