# Notities bij Ontwerp 26-1-‘23

## Inleiding

Bijgevoegd een uitleg bij de code (oplevering 26-1-2023) voor het Globescope Assesment.

In het assesment wordt gevraagd om een programma te maken, waarmee minimaal 10 specifieke vragen beantwoord kunnen worden. De vragen zijn gericht op het zoeken van één of meerdere routes die aan bepaalde criteria voldoen. Een route is dan een georderde lijst van afwisselend nodes en edges. De criteria gaan over de volgorde van tussenpunten lengte van de route, en het aantal tussenpunten.

## Uitleg bij algoritme

De gemeenschappelijke deler tussen de 10 vragen is dat er routes gezocht moeten worden die aan bepaalde criteria voldoen. Dit vormde een uitgangspunt bij de opzet van de progamma code.

Het begint met het vastleggen van de eigenschappen van de nodes en de edges in objecten. Een Node is een plaats die geidentificeerd wordt met een letter van het alfabet. Een node heeft een ingang en een uitgang waar een edge op aansluit. Een edge vormt een unieke verbinding tussen twee nodes. De edge heeft de eigenschap afstand (weight).

Binnen een node- objecten wordt vastgelegd welke edges in- en uitgaand zijn en , per edge wordt de weight meegegeven. Deze node- en edge-objecten vormen de bouwblokken van een route wordt opgebouwd uit nodes en edges.

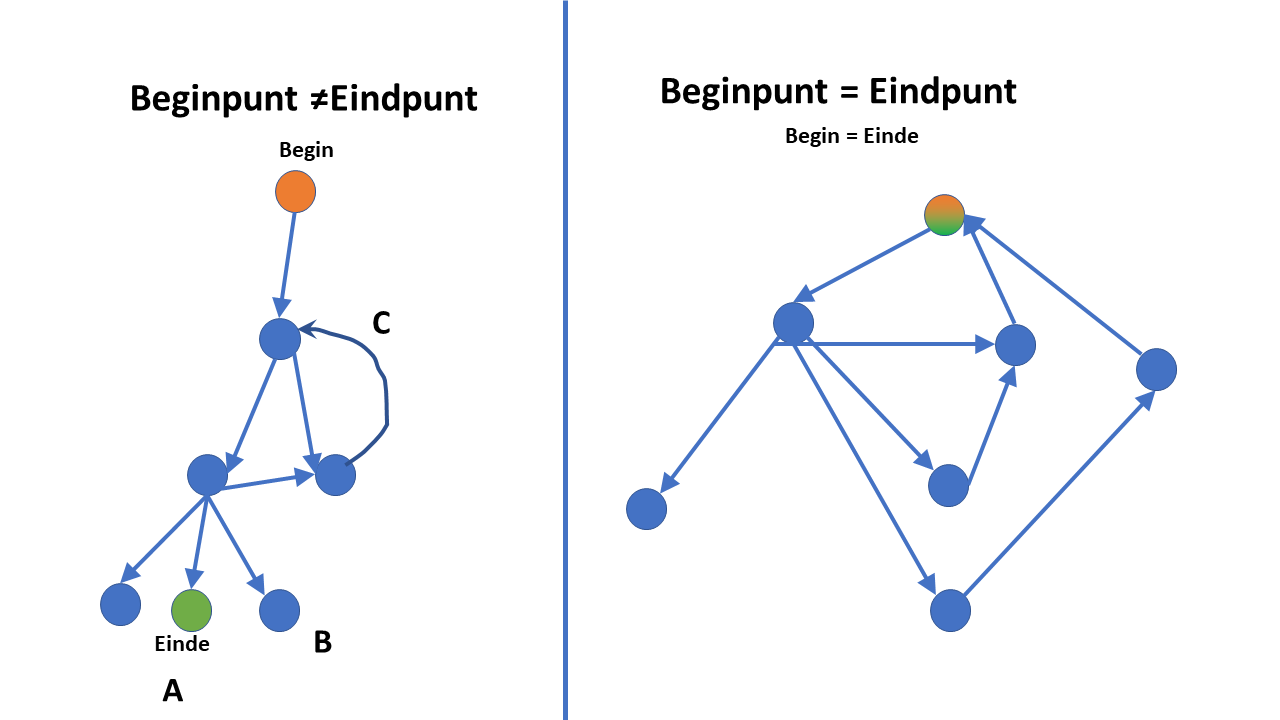
Het algoritme gaat verder met het vaststellen van mogelijke routes op basis van begin-, tussen- en eindpunten. Er worden twee verzamelingen routes gemaakt (Figuur 1).

* Hoofdroutes, met een begin- en een eindnode, gelijk aan begin- en eindnode in de code.
* Extra routes, met een begin en eindnode, gelijk aan het eindnode in de input. de extra routes zijn bedoeld om achter de hoofdroutes aan te plakken.

De twee verzamelingen routes worden gemaakt met *routing* functie. De functie bouwt routes op startend met een beginnode. Voor node-objecten is bekend met welke edges deze in verbinding staan. Op deze informatie wordt een route uitgebreid met edges en nodes en worden eventueel nieuwe route, als aftakking, gestart. Een route loopt door totdat:

* Eindnode bereikt is (A in Figuur 1).
* De knoop geen verdere verbinding heeft (B in Figuur 1).
* Een knoop bereikt is die al eerder gepasseerd in de route (C in Figuur 1)

Routes die de eindnode bereiken worden meegenomen in een verdere verwerking.

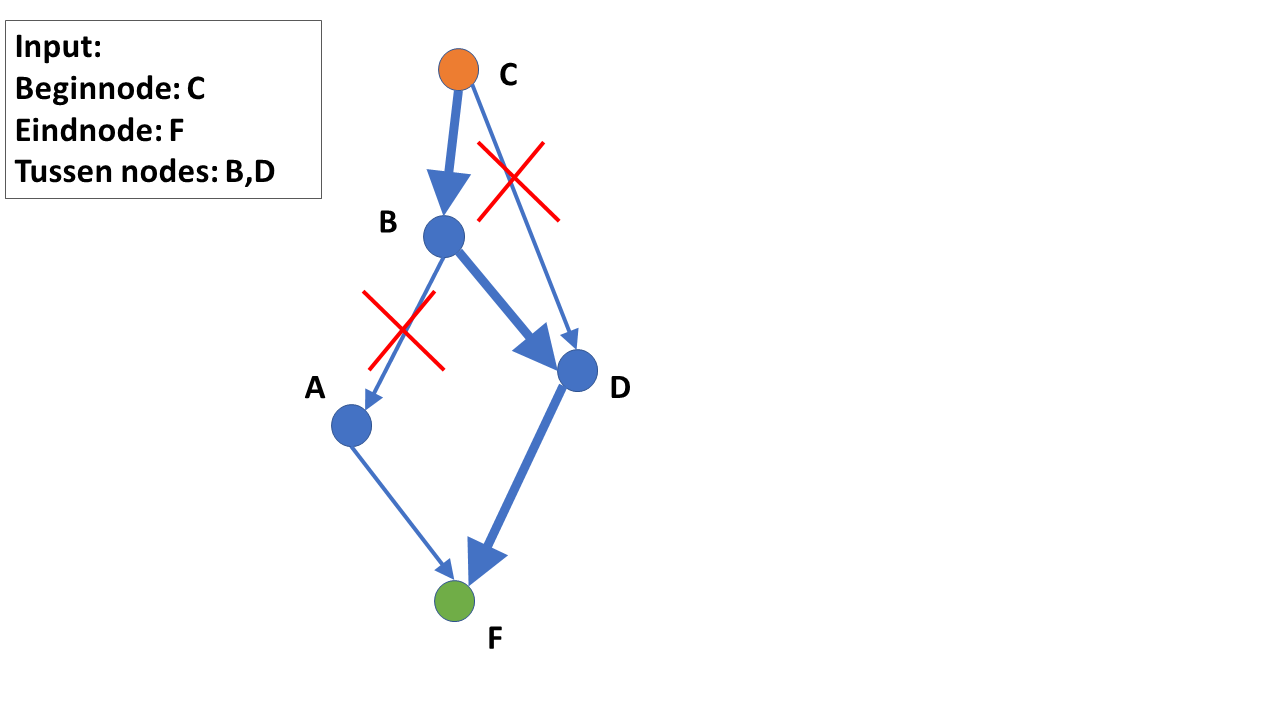
****

Figuur 1 Tweedeling in routes voor het algoritme, Hoofdroute (links), extra routes (rechts)

Voor de hoofdroutes geldt verder: routes worden eruit gefilterd waarbij tussennodes niet overeenkomen met de input (Figuur 2).

Het oplossen van de specifieke vragen gebeurt in de class Rfinder. De functie *routing* wordt hierbinnen twee maal aangeroepen, voor de hoofdroutes en de extra routes. In de class zijn verschillende methodes gedefinieerd voor het oplossen van vragen.

Voor vragen over de kortste route hoef je logischerwijs niet verder te kijken dan de hoofdroutes. Voor de andere vragen geldt dat mogelijk extra routes achter de hoofdroutes geplakt moeten worden. In principe kunnen er oneindig aantal extra routes achter elkaar vastgeplakt worden.

****

Figuur 2 Filtering routes op basis van tussennodes

## Programma opzet

De code bestaat uit 4 bestanden:

* rf\_input.py: klaar maken input voor verdere verwerking,
* rf\_processing: algoritme voor oplossen vragen,
* main4.py: hoofdprogramma, met functies die direct aanroepbaar zijn,
* unittest\_gsa4.py: (unit)tests, op 10 vragen assesment en onderdelen uit het programma.

### Module rf\_input

In rf\_input worden variabelen voor rf\_processing gegenereerd:

1. nodeorder,
2. Nodescoll.

De variabele ***nodeorder*** iseen list-object, met daarin node-objecten op volgorde van een beginnode via tussennodes naar een eindnode. De variabele wordt gegenereerd met de functie *make\_nodeorder*.

Een ***Nodescoll***-object bevat een verzameling unieke nodes die meegegeven zijn in een graphdef.

Verder zijn in rf\_input de classes “Node” en “Edge” gedefinieerd.

### Module rf\_processing

In rf\_processing staat het algoritme voor het oplossen van de vragen. Het algoritme in rf\_processing is gesplitst in:

* functie routing:
* class Rfinder:

De functie ***routing*** maakt het mogelijk om een verzameling hoofdroutes of extra routes te maken (zie Uitleg bij algoritme). De class Rfinder dient voor het beantwoorden van specifieke vragen op basis van een criteria. Het gaat om de volgende methodes:

* return\_shortestdistance
* return\_countmaxstops
* return\_countnrstops
* return\_countmaxdist

Voor deze methodes moet minimaal een *nodeorder* en Nodescoll opgegeven.

In de module is de class Route gedefinieerd. Een Route-object heeft naast een aantal methodes, twee attributen. De attribuut *route* bevat een list met een afwisseling van Node en Edge objecten. Met de attribuut *end* wordt aangeven of het einddoel van de route al dan niet bereikt is.

### Module main4

Deze module is bedoeld om een programma op te zetten. De modules rf\_input en rf\_processing worden geïmporteerd en staan ter beschikking voor het invullen van het programma. Als voorbeeld heb ik een paar functies gemaakt.

### (Unit)test, unittest\_gsa4.py

De unittest wordt gestart door het aanroepen van het bestand unittest\_gsa4.py

Met de test worden de uitkomsten van de 10 vragen van de assesment getest en een paar testen voor rf\_input.

## Bekende beperkingen/bugs

|  |  |
| --- | --- |
| **Waar** | **Wat** |
| Rf\_processing- class Rfinder- methode\_no\_such\_route | Implementatie in methode return\_ shortestdistance verschilt van de andere methodes in Rfinder. |
| Rf\_processing- class Rfinder- | Het onbeperkt uitbreiden met extra routes is (alleen) verwerkt in methode return\_countmaxdist. Eigenlijk hoort dit in de routing functie geïmplementeerd te worden, zodat het toegankelijk is voor alle methodes in Rfinder. |
| Rf\_processing- class Rfinder-  Methode return\_countmaxdist | Methode bevat (te) veel indentatie. |
| unittest\_gsa4 | Nog geen specifieke testen voor rf\_processing ontwikkeld. |