Gedragscode

Dit practicum wordt gequoteerd, het examenreglement is dan ook van toepassing. Soms is er echter wat onduidelijkheid over wat toegestaan is, en wat niet, inzake samenwerking bij opdrachten zoals deze.

De oplossing die je voorlegt, moet volledig het resultaat zijn van het werk dat jij zelf gepresteerd hebt. Je mag je werk uiteraard bespreken met andere mede-studenten, in de zin dat je praat over algemene oplossingsmethoden of algoritmen, maar de bespreking mag niet gaan over specifieke code of een specifieke oplossing die je aan het schrijven bent.

Als je het met anderen over je practicum hebt, mag dit er dus NOOIT toe leiden, dat je op om het even welk moment in het bezit bent van een geheel of gedeeltelijke kopie van het opgeloste practicum van anderen, onafhankelijk van of die code nu op papier staat of in elektronische vorm beschikbaar is, en onafhankelijk van wie die code geschreven heeft (mede-studenten, eventueel uit andere studiejaren, volledige buitenstaanders, e.d.). Dit houdt tevens ook in dat er geen enkele geldige reden is om de code van je practicum door te geven aan mede-studenten of te posten op welk forum dan ook.

Elke student is verantwoordelijk voor de code en het werk dat hij indient. Indien je Python code verkrijgt op een dubieuze wijze en die in het practicum inbrengt, word je zelf verantwoordelijk geacht. Als tijdens de demonstratie van het practicum de assistent twijfels heeft over het feit of het practicum zelf gemaakt is (bijv. gelijkaardige code met andere studenten), zal de docent worden ingelicht en de quotering in beraad gehouden worden. Dit zal ook gebeuren als achteraf blijkt dat jouw code te hard lijkt op die van andere studenten (bv. bij automatische vergelijking). Na het horen van de studenten, en indien dit de twijfels niet wegwerkt, zal er overgegaan worden tot het melden van een onregelmatigheid (fraude), zoals voorzien in het examenreglement.

Examenpracticum Methodiek van de Informatica
Routeplanning

20 maart 2015

Inleiding

Het gebruik van papieren landkaarten ligt al jaren achter ons. In de auto vertrouwen we op onze GPS, en wanneer we zelf een route moeten uitstippelen vragen we hulp aan Google Maps. Zo toont Figuur 1 bijvoorbeeld de snelste fietsroutes van de Arenberg campus naar het station in Leuven.

In dit practicum gaan jullie zelf aan de slag om enkele algoritmen te ontwikkelen die kunnen gebruikt worden om automatisch de snelste (of kortste) weg te berekenen tussen een aantal locaties.

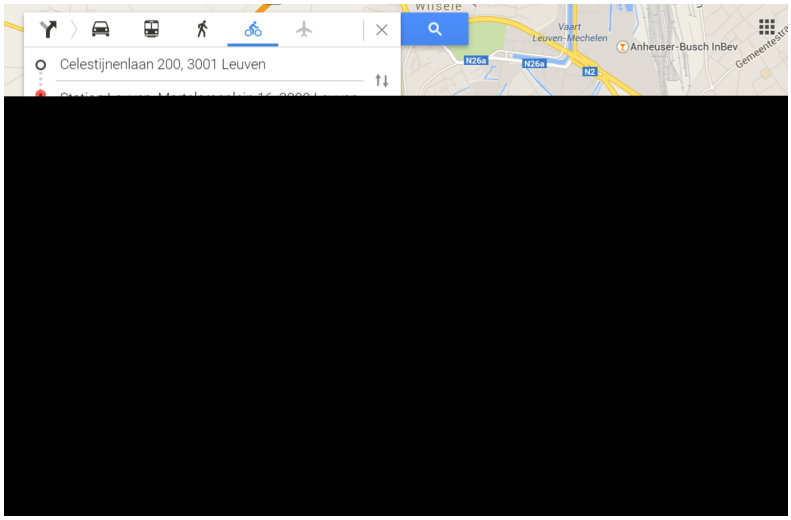
1 Een netwerk van wegen

Een vereiste om routes te kunnen berekenen is kennis van de beschikbare wegen.

Een eerste functie die je moet voorzien in dit practicum bestaat dus uit het initialiseren van dergelijk netwerk op basis van een gegeven verzameling tuppels. Elk tuppel bestaat uit: een uniek wegnummer (dit nummer bestaat uit een combinatie van cijfers en letters), de stad waaruit de weg vertrekt, de stad waar de weg aankomt, de lengte van de weg, en de gemiddelde snelheid.

Het tuppel (("BL", "Brussel", Leuven", 30.0, 120), ("LA", "Leuven", "Antwerpen", 61.0, 120), ("AB", "Antwerpen", "Brussel", 44.0,120)) stelt bijvoorbeeld de wegen in Figuur 2 voor. Steden worden gekenmerkt door hun naam (in de vorm van een string), de lengte van de weg wordt uitgedrukt in kilometers (in de vorm van een floating point getal), en de gemiddelde snelheid wordt uitgedrukt in kilometers per uur (in de vorm van en geheel getal).

Het resultaat van je initialisatie-functie is een netwerk van wegen dat in alle andere functies van je programma zal gebruikt worden (meegegeven als parameter). Je bepaalt zelf de beste voorstelling van dit netwerk, maar je dient er voor te zorgen dat je in constante tijd de eigenschappen van een weg op basis van zijn wegnummer kan opvragen (bv. de gemiddelde snelheid van de weg met uniek nummer 'E314').



Figuur 1: Voorbeeld van route in Google Maps

Figuur 2: Weergave van de wegen (("BL", "Brussel", Leuven", 30.0, 120), ("LA", "Leuven", "Antwerpen", 61.0, 120), ("AB", "Antwerpen", "Brussel", 44.0,120))

Uitbreiding (16+). Je mag je netwerk-voorstelling ook uitbreiden om het ook mogelijk te maken om in constante tijd alle wegen vertrekkende uit een bepaalde stad weer te geven. Dit betekent dus dat je initialisatie-functie een tuppel van weergaves zal teruggeven. Probeer er wel voor te zorgen dat je redundante opslag zoveel mogelijk beperkt. Deze uitbreiding is niet verplicht, maar wordt wel verwacht indien je minstens 16/20 nastreeft voor dit practicum.

2 Een route

Een route wordt voorgesteld als een opeenvolging van wegnummers. In dit practicum mag een route geen lussen bevatten, wat betekent dat elke weg hooguit 1 maal mag gebruikt worden.

3 Kortste route

Dit algoritme[1] is gebaseerd op het principe dat het kortste pad van a naar b bestaande uit een subset van de eerste k steden, gelijk is aan het minimum van kortste pad van a naar b bepaald door een subset van de eerste k-1 steden, en de som van het kortste pad van a naar de k^e stad bestaande uit een subset van de eerste k-1 steden en het kortste pad van de k^e stad naar b bestaande uit een subset van de eerste k-1 steden, of nog:

$kortste\_pad_{steden[0:k]}(a,b) = min(kortste\_pad_{steden[0:k-1]}(a,k) + kortste\_pad_{steden[0:k-1]}(k,b))$

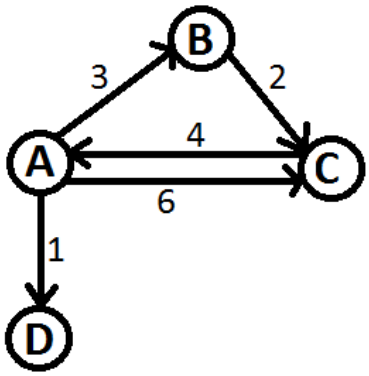
met als basisgeval: $kortste\_pad_{steden[0:0]}(a,b) = lengte\_weg(a,b)$

Het algoritme dat we van jullie verwachten bestaat uit 3 onderdelen: initialisatie, berekening, opzoeking.

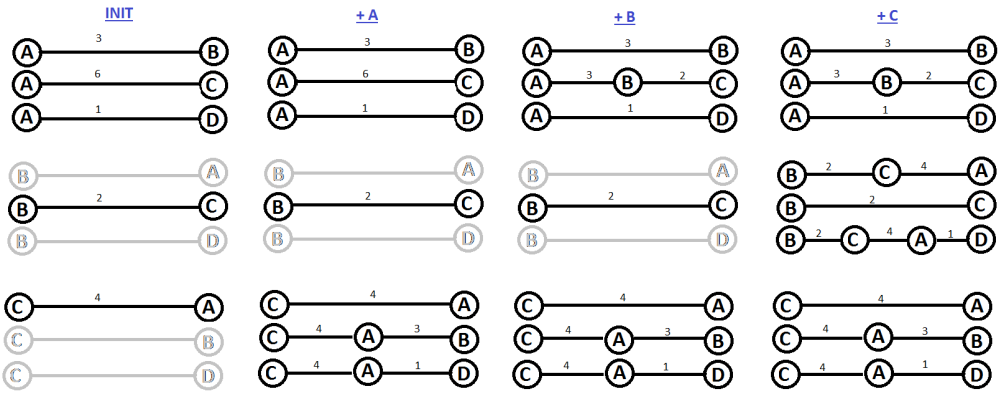
Initialisatie. Je creëert een rooster dat de rechtstreekse afstanden tussen steden (op basis van alle wegen in het netwerk) voorstelt. Om achteraf gemakkelijk de route te kunnen opvragen, moet je ook een rooster opstellen dat de te nemen weg registreert van een bepaalde stad naar een andere stad.

Berekening. Bij de berekeningsfase, ga je voor elke bestaande route bekijken of er geen kortere bestaat door bovenstaande formule toe te passen. Hou er rekening mee dat je niet enkel de afstanden moet bijhouden, maar ook de route zelf. Meer in het bijzonder zal nagegaan worden of de huidige kortste verbinding tussen stad A en stad B niet kan verbeterd worden door een verbinding van stad A naar een andere stad C (afstand terug te vinden in het rooster) gecombineerd met de verbinding van stad C naar stad B. In het rooster van de wegen, wordt de informatie omtrent de verbinding tussen A en C overgenomen in de verbinding tussen A en B. Daarmee kan later de route gereconstrueerd worden. Figuur 3 toont een voorbeeld wegennet. Figuur 4 illustreert het algoritme. Na de initialisatie zijn enkel de wegen gekend die steden rechtstreeks verbinden. In de volgende stappen, worden het aantal wegen telkens uitgebreid door tussenstops (steden A, B, C) systematisch toe te voegen. Na uitvoering van het algoritme, zijn voor alle mogelijke onderlinge verbindingen tussen steden de kortste paden gekend. Merk op dat mogelijk een combinatie van wegen korter is dan een rechtstreekse verbindingen (zoals

[1] Meer informatie over het Floyd-Warshall algoritme kan je vinden op wikipedia: http://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm



Figuur 3: Steden A, B, C, en D verbonden door wegen



Figuur 4: Voorbeeld uitvoering van het korste pad algoritme (Floyd-Warshall). Na de initialisatie zijn enkel de rechtstreekse wegen gekend. Deze worden stap voor stap uitgebreid door systematisch verbindingen naar een andere stad (als tussenstop) toe te voegen.

geïllustreerd in stap '+B' in Figuur 4. De rechtstreekse verbinding van A naar C is langer dan de combinatie van verbindingen van A naar B, en van daaruit naar C.

Opzoeking. Na de berekening, kan eenvoudig, voor gelijk welke begin- en eindstad het kortste pad worden opgevraagd aan het rooster. Dit onderdeel van het algoritme moet als **recursieve** hulpfunctie uitgewerkt worden.

Opmerkingen. Hou er rekening mee dat er meerdere wegen kunnen bestaan tussen 2 steden. Je kan bijvoorbeeld zowel een weg tussen Brussel en Antwerpen hebben die de autostrade volgt, als een die over de binnenbanen loopt.

Er kunnen ook steden meegegeven worden die moeten vermeden worden (bv. bepaal een route van Leuven naar Mechelen waarbij je Brussel vermijdt). Zorg dat je algoritme dit ook ondersteunt.

4 Snelste route met tussenstops

Dit algoritme berekent de snelste route waarbij er een aantal tussenstops (vias) in volgorde worden aangedaan. Werk hiervoor een backtracking algoritme uit. Dit algoritme gaat telkens op zoek naar de snelste route tussen 2 tussenstops tot de doelstad bereikt is.

Merk op dat in dit algoritme nog steeds geldt dat er geen lussen mogen voorkomen, maar twee steden kunnen wel verbonden zijn door meerdere wegen.

5 Gevraagd

Schrijf een programma dat een netwerk van wegen initialiseert op basis van gegeven waarden. Zorg voor functies om de kortste route te berekenen, en de snelste route via een aantal tussenstops (op basis van de algoritmen die hierboven beschreven zijn). Zorg dat je alle gevraagde functies implementeert in het skelet (de functie-hoofding moet ongewijzigd blijven) aangezien deze automatisch getest zullen worden.

5.1 Hulpfuncties

We vragen om ook een aantal verplichte hulpfuncties te implementeren. Deze zullen enerzijds jullie op weg helpen om je programma op te splitsen in deelproblemen, anderzijds zullen wij tijdens de evaluatie een aantal functies hiervan gebruiken om jullie programma automatisch te testen. Je vindt alle details omtrent die hulpfuncties terug in het skelet.

Je mag uiteraard het bestand nog uitbreiden met extra hulpfuncties. We vragen je wel om de volgorde van de gegeven functies niet te wijzigen. Je mag extra hulpfuncties toevoegen waar je dat nodig acht, maar de onderlinge volgorde van de gegeven functies dient onveranderd te blijven.

5.2 Invarianten en documentatie

Werk een invariant uit voor elke lus die je schrijft. De invarianten worden informeel geschreven (gewoon in tekst), zoals dat veelvuldig werd toegepast in de voorbeelden. Zorg wel voor een zo scherp mogelijke formulering van de invariant, die de verbanden tussen de variabelen betrokken in de iteratie afdoende beschrijft.

Voor zelf-gedefinieerde functies verwachten we ook een documentatiestring.

6 Praktische informatie

Hieronder geven we alle praktische informatie i.v.m. dit examenpracticum wat betreft deadline, de uiteindelijke verdediging en quotering, enz.

6.1 Alleen of in groep?

Het practicum zal **individueel** worden gemaakt.

De verdedigingen van het practicum vinden plaats tussen 11 mei en 22 mei. Per reeks zijn er een aantal slots voorzien voor de verdediging van het practicum (zie uurrooster en toledo). Zorg dus zeker dat je op deze momenten jezelf kan vrijmaken! Als er een *gegronde* reden is waarom een bepaald practicumslot in jouw uurrooster niet past (bijvoorbeeld een vak uit het 2e jaar op hetzelfde moment), laat je dit zo snel mogelijk weten aan Koen Paes (koen.paes@mirw.kuleuven.be). Het precieze moment van jouw verdediging wordt op 6 mei op Toledo beschikbaar gemaakt.

Lees ook zeker de gedragscode na op de eerste pagina van deze opgave en op Toledo. Op plagiaat staan zware sancties. Je mag over de opgave discussiëren met andere studenten, maar het is absoluut *not done* om code, op welke manier ook, door te geven.

6.2 Wat in te dienen?

Er wordt verwacht dat je een werkend programma schrijft. Je moet de oplossing van je practicum elektronisch **indienen tegen maandag 4 mei 2015 om 12u** (harde deadline; niet op tijd ingediend is geen verdediging). Stel het indienen niet uit tot de laatste minuut. Er kunnen steeds dingen foutlopen ...! Met oplossing bedoelen we alle Python-bestanden die je gemaakt hebt gebundeld in 1 zip-bestand (zelfs al heb je maar 1 python-bestand). Bekijk hiervoor de instructies op Toledo en volg deze strikt, zoniet zal je niet gepland worden voor ondervraging.

6.3 Verwachte inspanning

We verwachten dat je 15 uur nodig hebt om dit practicum tot een goed einde te brengen. In die schatting gaan we uit van een perfecte kennis van het cursusmateriaal. Het is met andere woorden de tijd die een ervaren Python programmeur nodig zou hebben om deze opgave uit te werken.

6.4 Waar word je op beoordeeld?

Je code zal automatisch getest worden op functionaliteit. Zorg dus dat je oplossing voldoet aan de gevraagde interface. Je krijgt van ons al een subset van het testbestand dat we zullen gebruiken voor de automatische testen zodat je kan controleren of je oplossing voldoet aan de vereisten. Naast functionaliteit, zal je oplossing ook (en vooral) op algemene concepten worden beoordeeld.

16 of meer Indien je een hogere score (16 of meer) wenst op je practicum, verwachten we dat je "slimme" code schrijft. Zo zal je er voor moeten zorgen dat je ook in constante tijd alle wegen vertrekkend uit een bepaalde stad kan opvragen. Uiteraard moet je er ook nog steeds voor zorgen dat de rest van je programma voldoet aan de opgave, en geschreven is volgens de regels van de kunst.

Dit extra deel zal enkel in rekening gebracht worden als de rest van je oplossing inderdaad van hoog niveau is. Werk daarom eerst een oplossing uit zonder rekening te houden met deze extra vereiste. Pas wanneer je volledig tevreden bent over die oplossing, kan je denken aan dit extra stuk.

6.5 Wat te doen bij onduidelijkheden?

Je kan steeds uitleg vragen over de opgave via het forum op Toledo. Indien je vragen stelt op Toledo word je verondersteld te controleren of de vraag nog niet aan bod kwam. Geef daarom een zinnig onderwerp aan je bericht.

6.6 Wat wel en niet gebruiken?

Dit practicum zal toetsen of je de leerstof tot aan de paasvakantie beheerst. Alles wat na de paasvakantie aan bod komt in de les (o.a. functioneel programmeren en OO) mag niet gebruikt worden. Je mag ook geen gebruik maken van list comprehension of set comprehension, behalve bij het initialiseren van gegevensstructuren, zoals dat ook in de cursus gebeurde.

Het practicum moet uitgewerkt worden in versie 2.7.9 van Python. Je mag gebruik maken van de standaard Python modules, de Python-bestanden die door ons gegeven worden en natuurlijk de zelf geschreven bestanden. **Het gebruik van andere modules en/of packages is niet toegelaten.** Bij twijfel, kan je het forum van Toledo raadplegen.

Heel veel succes!
het MI-team