**Beginselen van Programmeren**

**Practicum 2**

**November/December 2018**

**Spaarse matrices**

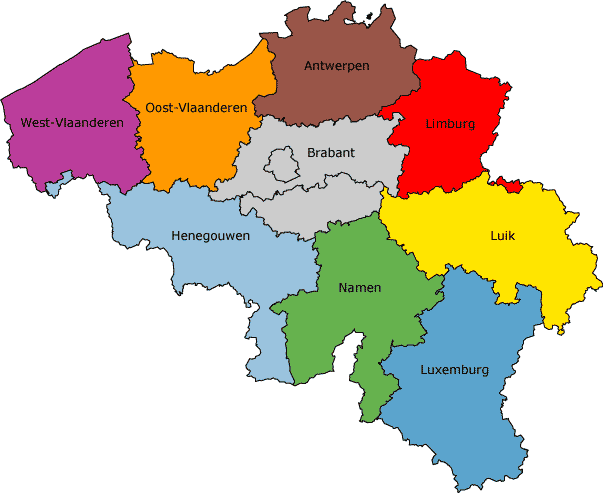
Het concept van een matrix in een programmeertaal laat toe om één-, twee-, of meer-dimensionale gegevensstructuren op te bouwen. De hoeveelheid geheugenruimte die zo’n matrix in een computer in beslag neemt, is natuurlijk afhankelijk van de grootte van de matrix. Een matrix van 100 op 200 kan 20.000 elementen bevatten, en beslaat bvb. ongeveer 400 keer meer geheugen dan een matrix van 5 op 10, dat is logisch.

Nu, voor heel wat, vaak wiskundige problemen wil men gebruik maken van gigantisch grote matrices, maar waarbij men vaak op voorhand weet dat er zich slechts een beperkt aantal elementen in de matrices zullen bevinden. Dit worden **spaarse matrices** genoemd.

Het kan bvb. nodig zijn om een matrix te gebruiken van 10.000 op 10.000, en men schat dat er zich effectief ongeveer 1.000 elementen in de matrix zullen bevinden. De informatie op de niet-gebruikte plaatsen in de matrix is bvb. voor dit probleem onbelangrijk. Een eenvoudige implementatie van een dergelijke matrix (a.h.v. een twee-dimensionale array) heeft enorm veel geheugen nodig, nl. geheugen dat plaats biedt aan 10.000 x 10.000 (dus 100.000.000) elementen. Als men schat dat er ongeveer 1.000 elementen effectief in de matrix zullen worden ingevuld, komt dit overeen met het nutteloos reserveren van 99.999.000 geheugenplaatsen. Het kan zelfs zijn dat programma’s die van dergelijke grote structuren gebruik willen maken, niet *kunnen* uitgevoerd worden op bepaalde computers omdat dit de maximum geheugencapaciteit van de computer kan overstijgen, ook al staan er effectief slechts een relatief klein aantal (bvb. 1000) elementen in de matrix. Voor deze opgave moet je een gegevensstructuur bedenken en implementeren die een dergelijke grote spaarse matrix qua geheugen op een veel efficiëntere manier kan voorstellen.

**Bankovervallen in België**

Een fictief voorbeeld is het volgende. Onderstel dat men een beeld wil krijgen van waar in ons land zich de meeste bankovervallen voordoen. Om de gegevens bij te houden legt men als het ware een 2-dimensionale matrix over een kaart van het land, waarbij elk element in de matrix 1 vierkante kilometer voorstelt, en waarin bijgehouden wordt hoeveel bankovervallen er in die bepaalde km2 in de afgelopen 5 jaar gebeurd zijn.



De matrixstructuur zou bij benadering uit 300 kolommen en 250 rijen bestaan, hetzij 75.000 elementen. Een (niet-gefundeerde) schatting van het aantal bankovervallen zou 100 kunnen zijn. Dit betekent dat de matrix 75.000 elementen bevat waarvan de meeste elementen de waarde 0 bevatten en maximaal ongeveer een 100-tal elementen een waarde groter dan 0 bevatten. Het is duidelijk dat het gebruiken van een gewone twee-dimensionale array in python niet efficiënt is qua geheugengebruik (100 nuttige elementen t.o.v. 75.000 elementen in de matrix). Dit is duidelijk een spaarse matrix.

**Het programma ...**

De FOD Binnenlandse Zaken wil dus een programma waarin gegevens rond bankovervallen bijgehouden kunnen worden. Per bankoverval geeft men aan waar (op welke coördinaten) de overval heeft plaatsgevonden. Het programma moet toelaten om informatie op te vragen, zoals het totaal aantal overvallen, het aantal overvallen op bepaalde coördinaten, het aantal overvallen in de ‘gevaarlijkste regio’, etc. Een regio wordt gedefinieerd als een vierkant deel van het land van 10 km op 10 km.

Voor deze opgave moet je een aantal functies definiëren, zodat minstens het onderstaande voorbeeld-programmaatje correct werkt.

Het kan handig zijn – om te testen bvb. – om ook andere functies te schrijven, bvb. een functie die een tekstueel beeld geeft van waar overvallen hebben plaatsgegrepen. Zo’n “tekstueel beeld” is een eenvoudige voorstelling van de inhoud van de matrix, bvb.

0000000000000000000000000000000000000...

0000000000000000030000000000000000000...

0000000000000030000000004500000000000...

0000000300000100000020000000000000000...

0000000000000000000000000000000000000...

0000000000000000000000000000000000000...

0000000000000000000000000000000000000...

0000000000000000000000000000000000000...

0000000000000000000000000000067000000...

...

In de figuur zijn ook, bij wijze van voorbeeld, 2 regio’s (die omwille van de beperkte ruimte hier 4 km op 4 km groot zijn i.p.v. 10 op 10) aangeduid. In werkelijkheid zijn er natuurlijk enkele duizenden regio’s te definiëren. De gevaarlijkste regio in dit voorbeeld bevat 13 overvallen.

**Opgave**

Bedenk eerst een gepaste gegevensstructuur die een dergelijke spaarse matrix kan voorstellen. Deze gegevensstructuur moet dus op een geheugen-efficiënte manier een spaarse matrix kunnen voostellen.

Dan, de spaarse matrix die we voor deze opgave nodig hebben heeft een vaste hoogte (aantal rijen) en breedte (aantal kolommen), maar moeten kunnen ingesteld worden (dit zijn dus geen constanten, maar kunnen eender welke natuurlijke getallen zijn). Bij het opstarten van het programma is deze matrix leeg. Daarna kunnen gegevens worden toegevoegd, en kan de data-structuur worden ‘ondervraagd’. Daarvoor wordt gebruik gemaakt van coördinaten (x, y) – x geeft de rij weer (een geldige x-coördinaat is dus begrensd door de hoogte), y de kolom (begrensd door de breedte van de matrix).

Definieer daarvoor de onderstaande functies.

Voor deze opgave moet je de interactie met de gebruiker dus verder niet zelf implementeren.

def creeer\_wereld (hoogte, breedte):

… # deze functie geeft een data-structuur die nog geen  
 # informatie van overvallen bevat; hoogte en breedte zijn  
 # geldige natuurlijke getallen (> 0)

def geef\_aantal\_overvallen (wereld):

… # deze functie geeft een geheel getal terug dat aangeeft  
 # van hoeveel overvallen informatie in deze wereld  
 # al werd ingevoerd

def geef\_aantal\_overvallen\_op\_positie (wereld, x, y):

… # deze functie geeft een geheel getal terug dat aangeeft  
 # van hoeveel overvallen informatie in deze wereld  
 # is ingevoerd voor de positie (x,y);  
 # (x,y) is een geldige positie in de wereld

def er\_is\_een\_overval\_gebeurd\_op\_positie (wereld, x, y):

… # deze functie geeft een Boolean terug die aangeeft  
 # of voor de gegeven positie een overval werd ingevoerd;  
 # (x,y) is een geldige positie in de wereld

def meeste\_overvallen\_in\_regio (wereld):

… # deze functie geeft weer hoeveel overvallen voorkwamen  
# in de ‘gevaarlijkste’ regio, dat is de regio met de  
# meeste overvallen

def voeg\_overval\_toe (wereld, x, y):

… # er is een overval gebeurd op positie (x,y), deze  
# functie voegt die informatie toe aan de wereld en geeft  
# als resultaat True terug; indien (x,y) zich buiten de  
# ‘grenzen’ van de wereld bevindt, geeft de functie  
# False terug

def main ()

wereld = creeer\_wereld (250, 300)

print(geef\_aantal\_overvallen (wereld)) # 0 print(er\_is\_een\_overval\_gebeurd\_op\_positie (wereld, 19, 52)) # False

voeg\_overval\_toe (wereld, 1, 5)  
voeg\_overval\_toe (wereld, 27, 95)  
voeg\_overval\_toe (wereld, 112, 225)  
voeg\_overval\_toe (wereld, 19, 52)  
voeg\_overval\_toe (wereld, 1, 225)  
voeg\_overval\_toe (wereld, 51, 185)  
voeg\_overval\_toe (wereld, 231, 5)  
voeg\_overval\_toe (wereld, 19, 52)

print(geef\_aantal\_overvallen (wereld)) # 8  
print(er\_is\_een\_overval\_gebeurd\_op\_positie (wereld, 19, 52)) # True  
print(er\_is\_een\_overval\_gebeurd\_op\_positie (wereld, 319, 520))# False

print(meeste\_overvallen\_in\_regio (wereld)) # …

if \_\_name\_\_ == "\_\_main\_\_": # gebruik dit letterlijk zo  
 main() # in je eigen code

**Praktisch**

Noem het bestand van je practicum “**bvp\_practicum2.py**”.

Op Toledo vind je een bestand ‘**Test.py**’ dat je kan gebruiken om een aantal gevallen te testen. Bemerk dat (1) je eerste zelf je code grondig moet testen voor je met deze Test-file aan de slag gaat; en (2) dat die Test-file slechts een beperkt aantal gevallen test – je kan en moet zelf je code uitgebreider testen. De Test-file die gebruikt zal worden om deels de score te bepalen zal ook extra test-gevallen bevatten. Het is dus perfect mogelijk dat jouw programma voor alle testen in Test.py slaagt, maar dat je niet alle punten scoort.

**Indienen**

Je dient één .py file (niet de .pyc file) in via Toledo onder “Practicum 2”, en dat vóór

**donderdag 13 december 2018, 18u.**

**Uitbreiding**

In de week van 17 tot 21 december krijg je, tijdens het tijdslot van je oefenzitting, 2 uur de tijd om één of meerdere kleine uitbreidingen aan je eigen code toe te voegen. Daarna worden je uitbreidingen gequoteerd op correctheid.

Zorg dat je eigen code leesbaar en duidelijk gestructureerd is, en schrijf commentaar. Dat zal het implementeren van een uitbreiding sowieso een pak eenvoudiger maken.

**BEMERK: INTEGRITEIT**

Bemerk dat het NIET toegelaten is om programma-code van elkaar te kopiëren! Met anderen mondeling een oplossingsstrategie bespreken kan uiteraard wel. Code delen NIET! We gebruiken fraude-detectie-tools voor Python-code om dit te checken. Die tools zijn overigens ‘slim’ genoeg om zich niet te laten misleiden door kleine variaties (bvb. andere variabele-namen).

Veel succes!

Het BVP-team