EXAMEN: **Scriptingtalen**

1e Bachelor Informatica
prof. dr. Peter Dawyndt
groep 1

maandag 27-06-2011, 14:00
academiejaar 2010-2011
eerste zittijd

Opgave 1

Gegeven is een Python script `splits`, waaraan een woord als argument moet meegegeven worden. In wat volgt mag je veronderstellen dat dit woord enkel uit hoofdletters bestaat. Het script `splits` schrijft alle tripletten (substrings bestaande uit drie letters) van het gegeven woord naar standaard uitvoer, gesorteerd in lexicografische volgorde. Onderstaande sessie illustreert de werking van het script `splits` aan de hand van enkele voorbeelden.

```
$ splits BANAAN
AAN
ANA
BAN
NAA
$ splits ANANAS
ANA
ANA
NAN
NAS
$ splits APPELSIEN
APP
ELS
IEN
LSI
PEL
PPE
SIE
```

Gevraagd wordt om een `sed` script `reconstrueer.sed` te schrijven, dat op basis van de uitvoer van het Python script `splits` terug het originele woord samenstelt. De werking van dit script wordt geïllustreerd aan de hand van onderstaande sessie.

```
$ splits BANAAN | sed -f reconstrueer.sed
BANAAN
$ splits ANANAS | sed -f reconstrueer.sed
???
$ splits APPELSIEN | sed -f reconstrueer.sed
APPELSIEN
```

Om het originele woord te reconstrueren, moet het `sed` script `reconstrueer.sed` het volgende *greedy* algoritme implementeren:

1. bouw in de *pattern space* een lijst van tripletten op; na deze verwerkingsstap bevat de *pattern space* bij verwerking van het woord `BANAAN` bijvoorbeeld

   ```
   AAN,ANA,BAN,NAA
   ```

   je mag hierbij zelf bepalen hoe de tripletten van elkaar gescheiden worden (in bovenstaand voorbeeld werd een komma als scheidingsteken gebruikt)

2. voeg in de lijst een suffix/prefix paar samen; een suffix/prefix paar bestaat uit twee substrings waarvan de twee-letter suffix van de ene substring gelijk is aan de twee-letter prefix van de andere substring; indien er meerdere dergelijke suffix/prefix paren bestaan, dan mag je willekeurig kiezen welk paar wordt samengenomen (vandaar dat we zeiden dat dit een *greedy* algoritme was)

3. herhaal de vorige stap totdat er geen suffix/prefix paar meer gevonden wordt; bij verwerking van het woord `BANAAN` wordt hierdoor bijvoorbeeld de volgende reeks substrings samengenomen

   ```
   AAN,ANA,BAN,NAA
   AAN,BANA,NAA
   AAN,BANAA
   BANAAN
   ```

   Hint: voor het woord `BANAAN` kunnen de suffix/prefix paren ook in andere volgorden samengenomen worden, maar het resultaat levert altijd een lijst van één enkel woord op: `BANAAN`

4. indien de lijst na de vorige stap gereduceerd is tot één enkel woord, schrijf dan dit woord naar standaard uitvoer; schrijf anders drie vraagtekens (`???`) naar standaard uitvoer

Opmerking: onderstaand voorbeeld illustreert aan de hand van het woord `ANANAS` dat de volgorde waarin suffix/prefix paren worden samengenomen, bepaalt of het originele woord kan gereconstrueerd worden of niet; voor deze opgave is het **niet** de bedoeling dat je op zoek gaat naar een volgorde waarin het samennemen resulteert in één enkel woord

```
ANA,ANA,NAN,NAS
ANA,ANAN,NAS
ANANA,NAS
ANANAS
```

```
ANA,ANA,NAN,NAS
ANA,NANA,NAS
ANAS,NANA
```

indien de substrings in de linker volgorde samengevoegd worden, wordt het woord `ANANAS` uitgeschreven; in geval de rechter volgorde gevolgd wordt, worden drie vraagtekens uitgeschreven door het `sed` script

Opgave 2

Gevraagd wordt om een `awk` script `expy.awk` te schrijven, dat kan gebruikt worden om functie- en klassedefinities te extraheren uit een Python script. Aan het `awk` script moeten op de commandolijn twee variabelen meegegeven worden: de `type`-variabele geeft aan of er functiedefinities (waarde `def`) dan wel klassedefinities (waarde `class`) moeten geëxtraheerd worden; de `naam`-variabele geeft de naam aan van de functies of klassen die moeten geëxtraheerd worden. Onderstaande sessie illustreert de basiswerking van het `awk` script `expy.awk` op basis van het voorbeeld Python script `nested.py`, dat ook werd meegeleverd bij het examen.

```
$ cat nested.py
class A(object):

  def __init__(self):
    self.y = 0

  class B(object):

    def __init__(self):
      self.x = 0

    def f(self):
      pass

  class C(object):

    def __init__(self):
      self.x = 0
      self.y = 0

      pass

    def f(self):
```

```
      def g():
        pass

      pass

def f():
  pass
$ awk -f expy.awk -v type="def" -v naam="g" nested.py
def g():
  pass
$ awk -f expy.awk -v type="class" -v naam="B" nested.py
class B(object):

  def __init__(self):
    self.x = 0

  def f(self):
    pass
$
```

Python scripts maken gebruik van witruimte (spaties en tabs) aan het begin van een regel om het insprongniveau van de regel te bepalen. Het insprongniveau wordt op zijn beurt gebruikt om statements op verschillende regels te groeperen. Functie- en klassedefinities zijn samengestelde statements die bestaan uit een *header* en een *suite*. De header is een regel die eventueel start met witruimte, gevolgd door het sleutelwoord `def` (voor functiedefinities) of `class` (voor klassdefinities), en de naam van de functie of de klasse (bestaande uit letters, cijfers en underscores). Na de naam volgt optioneel nog witruimte en bijkomende informatie tussen ronde haken. Een header-regel wordt altijd afgesloten door een dubbelpunt (`:`).
Alle niet-lege regels van de suite die horen bij de header-regel van een functie- en klassedefinitie hebben een insprongniveau dat dieper is dan dat van de header-regel. Een functie- of klassedefinitie start dus altijd met een header-regel en eindigt op de eerstevolgende niet-lege regel met een insprongniveau dat niet dieper is dan dat van de header. Merk op dat regels die enkel bestaan uit spaties en tabs (zogenaamde lege regels) niet gebruikt worden om het einde van functie- en klassedefinities te bepalen. Zorg ervoor dat het `awk` script `expy.awk` bij het extraheren van functie- en klassedefinities wel tussenliggende lege regels meeneemt, maar geen lege regels op het einde van de definitie. Geëxtraheerde functie- en klassedefinities mogen op het einde dus nooit lege regels bevatten. Zorg er ook voor dat de witruimte die het insprongniveau van de header-regel bepaalt, wordt verwijderd uit alle regels van de geëxtraheerde definitie (door die als prefix te verwijderen).
Naast een vaste naam moet het ook mogelijk zijn om aan het `awk` script `expy.awk` een reguliere expressie als naam mee te geven. Deze reguliere expressie omschrijft de volledige naam[1]. Door het feit dat in een Python script verschillende functies en klassen met dezelfde naam kunnen gedefinieerd worden, en door het feit dat namen als reguliere expressies kunnen opgegeven worden, kan het voorkomen dat het `awk` script `expy.awk` verschillende definities moet extraheren. In dat geval moeten de definities van elkaar gescheiden worden door een regel die enkel koppeltekens (`-`) bevat. Dit wordt geïllustreerd aan de hand van onderstaande sessie:

```
$ awk -f expy.awk -v type="def" -v naam="f" nested.py
def f(self):
  pass
----------------------------------------
def f(self):

  def g():
    pass

  pass
----------------------------------------
def f():
  pass
```

[1] De reguliere expressie `xy?z` zoekt dus naar functies of klassen met de naam `xyz` of `xz`, en niet naar functies of klassen waarvan de naam de substring `xyz` of `xz` bevat.

```
$ awk -f expy.awk -v type="class" -v naam="[BC]" nested.py
class B(object):

  def __init__(self):
    self.x = 0

  def f(self):
    pass
---------------------------------------
class C(object):

  def __init__(self):
    self.x = 0
    self.y = 0

    pass

  def f(self):

    def g():
      pass

    pass
$
```

Opmerking: Merk op dat functies en klassen in Python makkelijk kunnen genest worden binnen andere functies en klassen door het gebruik van het insprongniveau. Het voorbeeldbestand `nested.py` bevat hiervan verschillende voorbeelden. Let er dus op dat de volledige functie- en klassedefinities moeten geëxtraheerd worden, inclusief alle geneste definities. Indien zowel de naam van een definitie als de naam van een daarin geneste definitie overeenkomen met de opgegeven naam, moet enkel de buitenste definitie geëxtraheerd worden (maar dus wel inclusief de geneste definities).

Opgave 3

In deze opgave gaan we aan de hand van `bash` shell scripts (die geen `sed` of `awk` scripts bevatten) bepalen of een gegeven getal een *superpriem* is of niet. We doen dit in twee stappen. Gevraagd wordt om `bash` shell scripts te schrijven die voldoen aan onderstaande beschrijvingen:

1. Een natuurlijk getal n ($n \geq 2$) is een priemgetal als het niet deelbaar is door de getallen 2 tot en met $\lfloor \sqrt{n} \rfloor$. Schrijf een `bash` shell script `genereerPriemgetallen.sh` dat alle priemgetallen tot en met een zekere bovengrens naar een bestand uitschrijft. Aan dit shell script moeten exact twee argumenten meegegeven worden. Het eerste argument geeft de relatieve of absolute locatie aan van het bestand waarnaar de priemgetallen moeten uitgeschreven worden. Het tweede argument geeft de bovengrens aan van de priemgetallen die moeten uitgeschreven worden.

   Indien het opgegeven bestand reeds bestaat, mag je aannemen dat het de uitvoer van een vorige uitvoering van het shell script `genereerPriemgetallen.sh` bevat. In dit geval moet het shell script enkel de lijst van priemgetallen aanvullen tot en met de opgegeven bovengrens. Er moet dus niets gebeuren indien de bovengrens kleiner is dan het grootste reeds aanwezige priemgetal in het bestand. Onderstaande sessie illustreert de werking van het shell script `genereerPriemgetallen.sh`.

   ```
   $ genereerPriemgetallen.sh priem.txt 5
   $ cat priem.txt
   2
   3
   5
   $ genereerPriemgetallen.sh priem.txt 15
   $ cat priem.txt
   2
   3
   5
   7
   ```

```
11
13
$ genereerPriemgetallen.sh priem.txt 10
$ cat priem.txt
2
3
5
7
11
13
```

Zorg er voor dat het shell script `genereerPriemgetallen.sh` een shell functie `isPriem` bevat, waaraan een natuurlijk getal als argument moet meegegeven worden. De shell functie `isPriem` moet teruggeven of het gegeven natuurlijk getal een priemgetal is of niet.

Het shell script `genereerPriemgetallen.sh` moet controleren of er daadwerkelijk twee argumenten worden meegegeven, en dat het tweede argument een natuurlijk getal is. Indien niet aan deze voorwaarden voldaan is, moet het shell script een gepaste foutboodschap naar standaard error schrijven.

2. Een natuurlijk getal n is een *superpriem* als het een priemgetal is waarvan het rangnummer in de lijst van priemgetallen zelf ook een superpriem is. 127 is bijvoorbeeld het 31^{e} priemgetal. 31 is zelf ook een priemgetal, namelijk het 11^{e}. 11 is het 5^{e} priemgetal, 5 het derde, 3 het tweede en 2 het eerste. Merk op dat 2 per definitie een superpriem is. Uit voorgaande blijkt dus dat 127 een superpriem is, net zoals 31, 11, 5, 3 en 2.

   Schrijf een `bash` shell script `superpriem.sh` dat in staat is om voor een gegeven reeks natuurlijke getallen te bepalen of ze superpriem zijn of niet. Het shell script moet twee invoermethoden ondersteunen die geïllustreerd worden door onderstaande sessies. Deze sessies geven meteen ook aan hoe de uitvoer er moet uitzien die door het shell script gegeneerd wordt op standaard uitvoer. Ofwel worden de te controleren natuurlijke getallen als een reeks argumenten aan het shell script meegegeven.

```
$ superpriem.sh 127 131
127 is het 31e priemgetal
31 is het 11e priemgetal
11 is het 5e priemgetal
5 is het 3e priemgetal
3 is het 2e priemgetal
2 is het 1e priemgetal

*** 127 is een superpriem ***

131 is het 32e priemgetal
32 is geen priemgetal

*** 131 is geen superpriem ***

$
```

   Ofwel wordt na de optie `-f` de locatie van een bestand meegegeven dat op afzonderlijke regels een lijst van te controleren getallen bevat. In dit geval moet het shell script `superpriem.sh` ook nagaan of het opgegeven bestand bestaat en leesbaar is.

```
$ cat invoer.txt
109
11
$ superpriem.sh -f invoer.txt
109 is het 29e priemgetal
29 is het 10e priemgetal
10 is geen priemgetal

*** 109 is geen superpriem ***
```

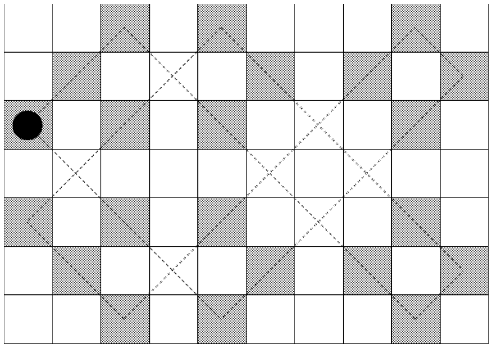
```
11 is het 5e priemgetal
5 is het 3e priemgetal
3 is het 2e priemgetal
2 is het 1e priemgetal

*** 11 is een superpriem ***

$
```

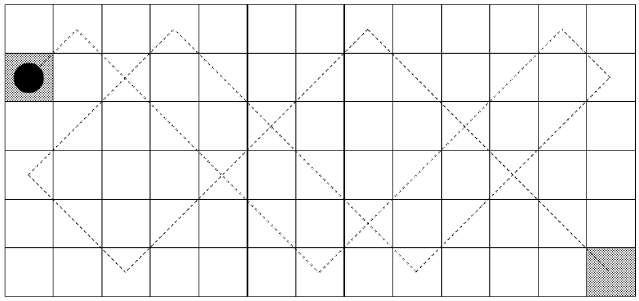
Om na te gaan of een natuurlijk getal een priemgetal is, moet het shell script `superpriem.sh` controleren of dit getal voorkomt in een lijst van priemgetallen die op afzonderlijke regels in een bestand staan. Om een dergelijk bestand te genereren moet het shell script uiteraard gebruik maken van het shell script `genereerPriemgetallen.sh`, op een manier waarbij gegarandeerd wordt dat alle nodige priemgetallen in het bestand aanwezig zijn. Het bestand dat de priemgetallen bevat is een tijdelijk bestand dat door het shell script `superpriem.sh` moet aangemaakt worden, maar ook weer moet verwijderd worden vooraleer het shell script eindigt.

Hint: indien je (nog) geen werkend shell script `genereerPriemgetallen.sh` hebt, dan kan je voor het schrijven van het shell script `superpriem.sh` gebruik maken van het meegeleverde bestand `priem.txt`, dat de lijst van alle priemgetallen kleiner dan 1000 bevat.

Opgave 4

Op een rechthoekig rooster bestaande uit vierkante velden, gerangschikt in n rijen en m kolommen, plaatsen we een bal in het midden van een willekeurige veld. De diameter van de bal is gelijk aan de hoogte (en breedte) van de velden in het rooster. Daarna stoten we de bal richting rechterbovenhoek van het rooster, met een hoek van 45° ten opzichte van de randen van het rooster. De bal weerkaatst tegen de randen van het rooster: indien de bal tegen een rand aanbotst, wordt elke snelheidcomponent loodrecht op de rand waartegen gebotst wordt, omgekeerd.





Gevraagd wordt om een subprocedure `flipperkast` te schrijven in VBA. Aan deze subprocedure moeten vier argumenten doorgegeven worden. De eerste twee argumenten geven aan hoeveel rijen n en kolommen m het rechthoekig rooster bevat. De laatste twee argumenten geven het rij- en kolomnummer aan van het veld waar de bal initieel geplaatst wordt. Rijen en kolommen worden van boven naar onder en van links naar rechts genummerd, startend vanaf 1. Op basis van deze configuratie moet een simulatie gemaakt worden van het traject dat de bal beschrijft doorheen het rooster. Het traject is voltooid als de bal terug zijn startpositie bereikt. Telkens als de bal over het midden van een veld rolt, wordt aan dat veld een extra punt toegekend (initieel heeft elk veld nul punten).
De subprocedure moet een nieuw werkblad aanmaken in Excel, waarop de resultaten van de simulatie als volgt worden weergegeven. Het rooster dat gebruikt wordt voor de simulatie correspondeert met het $n \times m$ rooster in de linkerbovenhoek van het werkblad. De cellen van het werkblad die corresponderen met velden van het rooster die na de simulatie een oneven aantal punten hebben, moeten een zwarte achtergrondkleur krijgen. Voorts moeten de cellen van het werkblad die niet in het rooster liggen, verborgen worden. De cellen van het werkblad die wel in het rooster liggen moeten hoogte 20.5 en

breedte 3.5 krijgen. Voor een 7×10 rooster met startpositie $(3, 1)$ (links) en voor een 6×13 rooster met startpositie $(2, 1)$ (rechts) moet de subprocedure de volgende werkbladen aan- en opmaken: