Examen Python M1

November 7, 2012

Sous-mots et générateurs

Le but de cet exercice est développer un module python élémentaire (`stringtools`) permettant d'itérer (sous la forme d'un générateur) sur les sous-séquences, préfixes, suffixes, ... d'une chaîne de caractères.

Quelques définitions/rappels. Soit $u = u_1 u_2 \dots u_n$ un mot.

- L'ensemble de **préfixes** de u est l'ensemble $\{u[0 : j] : 0 \leq i \leq n\}$. Le préfixe $u[0 : j]$ est **propre** si $j \leq n$, *i.e.*, $u[i : j] \neq u$.
- L'ensemble de **suffixes** de u est l'ensemble $\{u[i : n] : 0 \leq i \leq n\}$. Le suffixe $u[0 : j]$ est **propre** si $i \leq n$, *i.e.*, $u[0 : j] \neq u$.
- L'ensemble des **sous-séquences** de u est l'ensemble $\{u[i : j] : 0 \leq i \leq j \leq n\}$. La sous-séquence $u[i : j]$ est **propre** si $u[i : j] \neq i$.

Tout au long de cet exercice, nous allons développer les classes suivantes:

- `StringIterator`,
- `PrefixIterator`, `ProperPrefixIterator`,
- `SuffixIterator`, `ProperSuffixIterator`,
- `SubstringIterator`, `ProperSubstringIterator`,

puis les classes

- `DistinctSubstringIterator`, et
- `FixedLengthSubstringIterator`,

plus quelques classes additionnelles au seins d'un module nommé `stringtools`. Considérons dans un premier temps le diagramme de classes indiqué Figure 1. La classe `StringIterator` doit être considérée comme un classe abstraite et donc non instanciable.

Pour illustrer notre propos, considérons le programme suivant:

```
from stringtools import PrefixIterator
from stringtools import SuffixIterator
from stringtools import SubstringIterator

u = 'abc'

# On itere sur tous les prefixes de u
print '(1) ' + ','.join(str(v) for v in PrefixIterator(u))
```

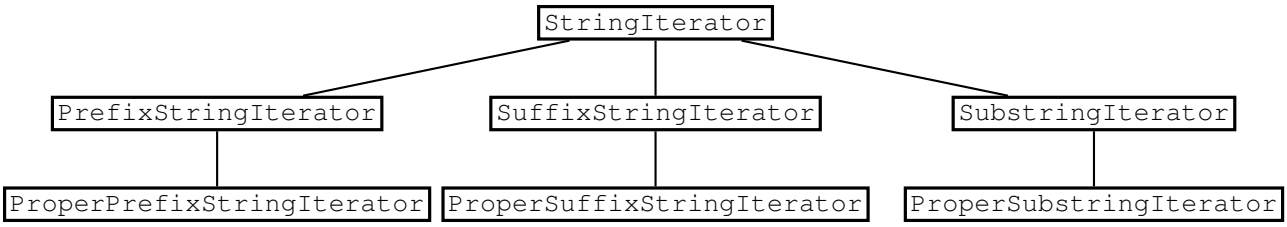

Figure 1: Diagramme de classes.

```
# On itere sur tous les suffixes de u
print '(2) ' + ','.join(str(v) for v in SuffixIterator(u))

# On itere sur toutes les sous-chaines de u
print '(3) ' + ','.join(str(v) for v in SubstringIterator(u))

# Nous manipulons des generateurs !
u = 'ab'
g = iter(SubstringIterator(u))
print '(4) ' + g.next()
print '(5) ' + g.next()
print '(6) ' + g.next()
print '(7) ' + g.next()
```

qui fournit ce résultat (ce résultat n'est pas unique puisque l'ordre des préfixes, suffixes et sous-chaînes n'est pas imposé):

```
$ python test_stringtools.py
(1) ,a,ab,abc
(2) ,c,bc,abc
(3) ,a,ab,abc,b,bc,c
(4) a
(5) ab
(6) b
Traceback (most recent call last):
  File "test.py", line 23, in <module>
    print '(7) ' + g.next()
StopIteration
$
```

Question 1

Écrire la classe mère (abstraite) `StringIterator`, ainsi que les classes `PrefixIterator`, `SuffixIterator` et `SubstringIterator`.

Question 2

Écrire les classes `ProperPrefixIterator`, `ProperSuffixIterator` et `ProperSubstringIterator` permettant d'itérer sur les préfixes, suffixes et sous-séquences propres d'une chaîne de caractères.

Question 3

La chaîne de caractères "ababa" contient trois occurrences de la sous-séquence "a", deux occurrences de la sous-séquence "b", deux occurrences de la sous-séquence "ab", deux occurrences de la sous-séquence "ba", deux occurrences de la sous-séquence "aba", une occurrence de la sous-séquence "bab", une occurrence de la sous-séquence "abab", une occurrence de la sous-séquence "baba", et une occurrence de la sous-séquence "ababa". Considérons le problème consistant à itérer sur les sous-séquences distinctes d'une chaîne de caractères. Une première solution serait

```
from stringtools import SubstringIterator

u = 'ababa'
seen = []
for v in SubstringIterator(u):
    if v not in seen:
        seen.append(v)
print ','.join(str(v) for v in seen)
```

Écrire la classe `DistinctSubstringIterator` (et indiquer où se trouve cette classe dans la hiérarchie présentée Figure 1) permettant d'itérer (sous la forme d'un générateur) sur les sous-séquences distinctes d'une chaîne de caractères. C'est-à-dire:

```
from stringtools import DistinctSubstringIterator

u = 'ababa'
print ','.join(str(v) for v in DistinctSubstringIterator(u))
```

Question 4

Écrire la classe `FixedLengthSubstringIterator` (et indiquer où se trouve cette classe dans la hiérarchie présentée Figure 1) permettant d'itérer (sous la forme d'un générateur) sur les sous-séquences d'une longueur donnée d'une chaîne de caractères. Par exemple

```
from stringtools import FixedLengthSubstringIterator

u = 'ababa'
print ','.join(str(v) for v in FixedLengthSubstringIterator(u, 3))
# affiche (par exemple, l'ordre n'est pas fixe): aba, bab, aba
```

Question 5

Un **bord** dans un mot u est un mot v qui est à la fois préfixe et suffixe de u. Par exemple, il existe deux bords pour `abab`: `ab` et `abab`. Proposer une solution pour le problème suivant: Itérer (sous la forme d'un générateur) sur les bords d'une chaîne de caractères donnée.

Question 6

Écrire la classe `SplitIterator` qui permet d'itérer (sous la forme d'un générateur) sur toutes les décompostions préfixe–suffixe d'une chaîne de caractères donnée. Par exemple,

```
from stringtools import SplitIterator

u = 'ababa'
print ','.join(str(v) for v in SplitIterator(u))
# affiche (par exemple, l'ordre n'est pas fixe)
# ('',ababa),(a,baba),(ab,aba),(aba,ba),(abab,a),(ababa,'')
```

En déduire la classe `ConjugateIterator` qui permet d'itérer (sous la forme d'un générateur) sur tous les *conjugués* d'une chaîne de caractères. Les mots u et v sont conjugués s'il existe les mots z et t tels que $u = zt$ et $v = tz$. Par exemple,

```
from stringtools import ConjugateIterator

u = 'ababa'
print ','.join(str(v) for v in ConjugateIterator(u))
# affiche (par exemple, l'ordre n'est pas fixe)
# ababa,babaa,abaab,baaba,aabab
```