1I001 – MIPI 14.2 TME Solitaire

le 28 Octobre 2014

Le seul document permis est la carte de référence.

Exercice 1 (4 points)

Question 1 : Ecrivez une fonction **somme** qui prend en entrée un entier naturel n et renvoie la valeur de

$$\sum_{0 \leq i < j \leq n} 2i + 3j.$$

Correction :

```
def somme(n):
    """ int -> int
    Hypothese: n >= 0
    Renvoie la somme  des (2i+3j) pour tous les couples (i,j) dans
    l'intervalle [0,n], i étant différent de j"""
    #sum : int
    sum=0 # sum est la somme finale que l'on cherche a calculer
    #i : int
    for i in range (0,n+1):
        #j : int
        for j in range (i+1, n+1):
            sum=sum+2*i+3*j

    return sum
```

Question 2 : Ecrivez un jeu de tests pour la fonction précédente.

Correction :

```
#jeu de tests
assert somme(0) == 0
assert somme(1) == 3
assert somme(2) ==  17
assert somme(6) == 343
```

Exercice 2 : Anagrammes (8 points) Le but de cet exercice est de faire appel à des notions d'itérations et de fonctions partielles sur des chaines de caractères. Au sens mathématique, deux mots de même longueur $a_1, a_2, \cdots, a_n$ et $b_1, b_2, \cdots, b_n$ sont anagrammes s'il existe une permutation σ de l'ensemble des entiers $[1, n]$ tel que pour tout $i : b_i = a_{\sigma(i)}$. Cela signifie que deux mots sont anagrammes quand l'un peut être obtenu depuis l'autre par permutation des lettres. Par exemple, "parisien" est un anagramme de "aspirine" et "amal" est un anagramme de "lama".

Dans la suite, on considère qu'un mot est une chaine de caractère qui ne contient pas d'espacement.

Question 1 : Donnez la définition d'une fonction partielle **moins_lettre**(c,a) qui renvoie :

- la chaîne obtenue à partir de la chaine c en supprimant la première occurrence de la lettre a dans c, si c contient au moins une fois a.
- None si c ne contient pas a.

Correction :

```
def moins_lettre (c,a):
    """ str*str  -> str + NoneType
    Hypothese: len(a)=1
    Renvoie le mot c prive de la première occurrence du
    caractère a si c contient a, et None sinon. """
    #premiere_trouvee : bool
    premiere_trouvee=False #Vaut True ssi une occurrence au moins a été vue.
    #res : str
    res="" #pour garder en mémoire le résultat
    #d : str
    for d in c:
        # le seul cas ou ne rajoute pas la lettre au mot res, c'est lorsque
        # l'on tombe sur le caractere cherche pour la premiere fois
        if d==a and not premiere_trouvee:
            premiere_trouvee=True
        else:
            res=res+d
```

```
    if premiere_trouvee:
        return res
    else:
        return None

#jeu de tests
assert moins_lettre("aspirine","i") == "asprine"
assert moins_lettre("aspirine","z") == None
assert moins_lettre("aspirine","p") == "asirine"
assert moins_lettre("","i") == None
```

Question 2 : Ecrivez la fonction **anagramme** qui étant donnés deux mots m_1 et m_2 renvoie True si les mots sont anagrammes et False sinon.

Par exemple :

```
>>> anagramme("cecile","frederic")
False
>>> anagramme("amal","lama")
True
>>> anagramme("alexandra","axel")
False
```

Correction :

```
def anagramme (m1,m2):
    """ str*str  -> bool
    Renvoie True si les mots sont anagrammes, et False sinon."""
    #cm2 : str
    cm2 = m2 #copie de m2 à laquelle on va retirer des lettres
    #res : str + NoneType
    res="" #resultat de la fonction moins_lettre
    #a : str
    for a in m1:
        res = moins_lettre(cm2,a) # on retire une lettre, si on peut
        if res==None: # si on n'a pas réussi, c'est qu'ils ne sont pas anagrammes
            return False
        else:
            cm2 = res # autrement on continue avec une lettre en moins
    return cm2 =="" # si on n'a enlevé toutes les lettres, c'est bon !

#jeu de tests
assert not anagramme("cecile","frederic")
```

```
assert anagramme("amal","lama")
assert anagramme("leon","noel")
assert not anagramme("alexandra","axel")
assert not anagramme("lyse","ulysse")
assert anagramme("","")
```

Exercice 3 : Le problème du logarithme discret (8 points) Le problème du logarithme discret est un problème cousin de la factorisation qui est, lui aussi, utilisé en cryptologie. Le problème est le suivant. Etant donné un entier naturel h et un entier naturel g il s'agit de retrouver l'entier x (en supposant ici qu'il existe) tel que $h = g^x$. L'entier x est appelé le logarithme discret de h en base g. Par exemple, le logarithme discret de 8 en base 2 est $x = 3$.

Question 1 : Ecrivez une fonction **puissance_naive** qui étant donnes deux entiers naturels g et n retourne la liste des n premieres puissances successives de g : $g, g^2, g^3, \cdots, g^n$.

Par exemple :

```
>>> puissance_naive(2,6)
[2,4,8,16,32,64]
```

Question 2 : Complétez votre fonction précédente pour que soit utilisé l'élément précédent g^{i-1} déjà calculé pour le calcul de l'élément suivant g^i.

Correction Questions 1 et 2 :

```
def puissance_naive(g,n):
    """ int*int  -> list[int]
    Hypotheses: n >= 0, g >= 0
    Renvoie la liste des n premieres puissances successives de g """
    #L : list[int]
    L=[] # la liste finale
    #e : int
    e = 1 # e est l'element courant, on commence par l'element neutre
    #i : int
    for i in range (1,n+1):
        e=e*g
        L.append(e)
    return L

#jeu de tests
assert puissance_naive(2,6)==[2,4,8,16,32,64]
assert puissance_naive(2,0)==[]
assert puissance_naive(0,6)==[0,0,0,0,0,0]
```

Question 3 : Ecrivez une fonction **logarithme_discret** qui, étant donnés trois entiers naturels g, h et n retourne le logarithme discret de h en base g, s'il existe, False sinon. Vous ferez l'hypothèse que $h <= g^n$.

Par exemple :

```
>>> logarithme_discret(2,32,6)
5

>>> logarithme_discret(3,10460353203,25)
21
```

Indication : utilisez la fonction puissance_naive !

Correction :

```
def logarithme_discret(g,h,n):
    """ int*int*int  -> int+NoneType
    Hypotheses: n >= 0, g >= 0, h<=g^n
    Renvoie le logarithme discret de h en base g, s'il existe, False sinon"""
    #L : list[int]
    L=puissance_naive(g,n)
    #i : int
    #i est la puissance de l'element courant dans la liste que l'on compare avec h
    for i in range(0,len(L)):
        if L[i]==h:
            return i+1
    return False


#jeu de tests
assert logarithme_discret(2,32,6)==5
assert logarithme_discret(3,10460353203,25)==21
assert logarithme_discret(2,10460353203,35)==False
```

Question 4 : Quel est le logarithme discret de 96889010407 en base 7 ?

Correction :

```
>>> logarithme_discret(7,96889010407,25)
13
```

Donc le logarithme discret cherché est 13.