Devoir Maison d'ITC N°1
pour la semaine du 04/11/2024
à rendre en séance de TP

Consignes

- La qualité, la lisibilité et l'efficacité du code entreront pour une part importante dans l'appréciation de la copie. L'indentation du code doit figurer clairement sur la copie, il est fortement conseillé d'y ajouter une barre verticale sur la gauche afin de préciser clairement les portions de code ayant la même indentation.
- **Le crayon à papier ne sera pas corrigé.**
- **Il est rappelé également que recopier une correction sur internet est complètement inutile pour tout le monde.**

Toutes les fonctions devront avoir *a minima* une signature. Il est attendu, dans chaque question, d'utiliser les fonctions des questions précédentes au maximum.

Problème **Calcul du produit épointé d'une liste** [Sol] On rappelle que le quotient q et le reste r de la division euclidienne d'un entier naturel a par un entier naturel non nul b sont définis comme les uniques entiers vérifiant :

$$a = b \times q + r \quad \text{et} \quad 0 \leqslant r < b.$$

Attention

Dans ce sujet, l'utilisation des opérateurs `//` et `%` n'est **pas** autorisée!

Même lorsque cela n'est pas précisé, `lst` désigne une liste non vide formée d'entiers strictement positifs.

1. Écrire une fonction `prod(lst)` qui renvoie le produit des éléments de la liste `lst`. Par exemple, `prod([1,2,3,4,5])` renverra `120`.
2. On souhaite écrire une fonction `div(a, b)` qui renvoie le quotient de la division euclidienne d'un entier naturel a par un entier naturel non nul b. Par exemple, `div(23, 6)` doit renvoyer 3 car :

   $$23 = 6 \times 3 + 5 \quad \text{et} \quad 0 \leqslant 5 < 6$$

   On propose le code suivant :

```
def div(a, b):
    q = 0
    v = 0
    while v <= a :
        q = q+1
        v = v+b
    return (...)
```

   2.1) Détailler dans un tableau les contenus successifs des variables `q` et `v` dans les différents passages dans la boucle `while` lors de l'appel `div(23, 6)`.

   2.2) Recopiez la fonction (en ajoutant signature et commentaire) et complétez la valeur à renvoyer.

   2.3) En utilisant la fonction précédente, écrire une fonction `mod(a, b)` qui renvoie le reste de la division euclidienne d'un entier naturel `a` par un entier naturel non nul `b`. Par exemple, `mod(23, 6)` doit renvoyer `5`.
3. On souhaite écrire une fonction `prodSauf(lst)` qui reçoit la liste `lst` et qui renvoie une liste de même taille dont l'élément d'indice `i` est le produit de tous les éléments de `lst` sauf celui d'indice `i`.
   Par exemple, `prodSauf([1,2,3,4,5])` renvoie la liste `[120,60,40,30,24]` car :

   $$120 = 2 \times 3 \times 4 \times 5, \quad 60 = 1 \times 3 \times 4 \times 5, \quad 40 = 1 \times 2 \times 4 \times 5, \quad \ldots.$$

   Une première version suit le principe suivant :

```
#1. calculer le produit p de tous les éléments de la liste lst
#2. initialiser une liste lps vide
#3. boucler pour chaque indice i de la liste lst sur :
    #3.1 insérer à droite de la liste lps l'entier égal au
    ↪ quotient de la division euclidienne de p par l'élément
    ↪ d'indice i de lst
#3. renvoyer la liste lps
```

   Écrire une fonction `prodSauf1(lst)` effectuant ce traitement.
4. Écrire une deuxième version `prodSauf2(lst)`, dont le principe n'est pas de calculer le produit de tous les éléments de la liste `lst` puis d'effectuer un calcul de quotient, mais seulement le produit des éléments d'indice différent de i.
5. Si $\ell = [\ell_0, \ldots, \ell_{n-1}]$ est une liste de n entiers, on définit la liste $p = [p_0, \ldots, p_{n-1}]$ des préfixes de ℓ et la liste $s = [s_0, \ldots, s_{n-1}]$ des suffixes de ℓ par :

   $$p_0 = \ell_0 \quad s_{n-1} = \ell_{n-1}, \quad \text{et :} \quad \forall i \in [\![1, n-1]\!] \quad \begin{cases} p_i = p_{i-1} \times \ell_i \\ s_{n-i-1} = s_{n-i} \times \ell_{n-i-1} \end{cases}$$

   5.1) Préciser les listes p et s obtenues si $\ell = [1,2,3,4,5]$.

   5.2) Écrire une fonction `pref_suff(lst)` qui reçoit la liste `lst` et qui renvoie le couple (p, s) des listes préfixe et suffixe de `lst`.

5.3) En utilisant la fonction `pref_suff`, en déduire une fonction `prodSauf3(lst)` qui renvoie le même résultat que les fonctions `prodSauf1` et `prodSauf2`. *on expliquera bien entendu la méthode utilisée!*

6. Comparer les performances en terme de temps de calcul des fonctions `prodSauf1`, `prodSauf2` et `prodSauf3`.

La réponse sera expérimentale (en attendant le cours de complexité), en mesurant les temps de calcul mis par ces fonctions sur des listes de grandes tailles (augmentez progressivement cette taille) remplies par des entiers choisis aléatoirement. Pour cela, vous utiliserez d'une part la fonction `time()` (à importer du module time) qui permet de mesurer un temps de calcul, et d'autre part la fonction `randint(a, b)` (à importer du module `random`) permettant d'obtenir un entier choisi au hasard de manière équiprobable dans l'intervalle d'entiers $[\![a, b]\!]$. Vous trouverez facilement sur internet plus d'information sur l'utilisation de ces fonctions. Par exemple, pour afficher le temps d'exécution de `prodSauf1` sur une liste `lst`, on écrira :

```
t1 = time()
lps = prodSauf1(lst)
t2 = time()
print("Durée de prodSauf1 : ",t2-t1)
```

Votre réponse fera apparaître :

6.1) les lignes de code permettant de remplir une liste `lst` de taille n avec des valeurs aléatoires, et affichant à l'écran les temps de calculs respectifs aux appels des fonctions `prodSauf1`, `prodSauf2` et `prodSauf3` pour cette même liste ;

6.2) un tableau récapitulant les temps d'exécution des trois fonctions pour les différentes valeurs de n ci-dessous :

n	prodSauf1	prodSauf2	prodSauf3
5			
10			
20			
50			
100			
1000			
10000			

Il est possible que certaines exécutions ne se fassent pas en un temps raisonnable, dans ce cas on laissera la case vide.

6.3) un texte analysant qualitativement les résultats précédents.

Correction
du Devoir Maison d'ITC N°1

Solution

1.
```
def prod(lst : list) -> int :
    """
    Renvoie le produit des éléments de lst.
    """
    p = 1
    for e in lst :
        p *= e
    return p
```

```
>>> prod([1, 2, 3, 4, 5])
120
```

2. 2.1)

i	q_i	v_i
0	0	0
1	1	6
2	2	12
3	3	18
4	4	24

2.2)
```
def div(a : int, b : int) -> int :
    """
    Renvoie le quotient de la division euclidienne de a \
    ↪ par b.
    """
    q = 0
    v = 0
    while q*b <= a:
        q += 1
        v += b
    return q-1
```

```
>>> div(23, 6)
3
```

2.3)
```
def mod(a : int, b : int) -> int :
    """
    Renvoie le reste de la division euclidienne de a par b.
    """
    return a-div(a,b)*b
```

```
>>> mod(23, 6)
5
```

3.
```
def prodSauf1(lst : list) -> list :
    """
        Renvoie une liste de même taille que lst dont le terme \
        ↪ d'indice i
        contient le produit de tous les éléments de lst sauf \
        ↪ celui d'indice i.
    """
    p = prod(lst)
    lps = []
    for e in lst :
        lps.append(div(p, e))
    return lps
```

```
>>> prodSauf1([1, 2, 3, 4, 5])
[120, 60, 40, 30, 24]
```

4.
```
def prodSauf2(lst : list) -> list :
    """
        Renvoie une liste de même taille que lst dont le terme \
        ↪ d'indice i
        contient le produit de tous les éléments de lst sauf \
        ↪ celui d'indice i.
    """
    lps = []
    for i in range(len(lst)) :
        lps.append(prod(lst[:i]+lst[i+1:]))
    return lps
```

Dans ce code, notons que `lst[:i]+lst[i+1:]` construit la sous-liste de `lst` en enlevant l'élément d'indice i (sans modifier la liste `lst`).

```
>>> prodSauf2([1, 2, 3, 4, 5])
[120, 60, 40, 30, 24]
```

5. 5.1) On obtient successivement

$$p_0 = 1 \quad \text{et} \quad s_4 = 5$$

$$p_1 = p_0 \times 2 = 2 \quad \text{et} \quad s_3 = s_4 \times 4 = 20$$
$$p_2 = p_1 \times 3 = 6 \quad \text{et} \quad s_2 = s_3 \times 3 = 60$$
$$p_3 = p_2 \times 4 = 24 \quad \text{et} \quad s_1 = s_2 \times 2 = 120$$
$$p_4 = p_3 \times 5 = 120 \quad \text{et} \quad s_0 = s_1 \times 1 = 120,$$

donc :

$$\boxed{p = [1,2,6,24,120]} \quad \text{et} \quad \boxed{s = [120,120,60,20,5]}$$

5.2)

```
def pref_suff(lst : list) -> tuple :
    """
        Renvoie le couple (p,s) où p et s sont la \
        ↪ respectivement la liste
        des préfixes et des suffixes de lst.
    """
    p = [lst[0]]
    s = [lst[-1]]
    for i in range(1, len(lst)) :
        p.append(p[-1]*lst[i])
        s.insert(0,s[0]*lst[-i-1])
    return (p,s)
```

```
>>> pref_suff([1, 2, 3, 4, 5])
([1, 2, 6, 24, 120], [120, 120, 60, 20, 5])
```

5.3) On constate que $p[i] = \ell_0 \times \cdots \times \ell_i$ et que $s[i] = \ell_i \times \cdots \times \ell_{n-1}$, donc :

$$\ell ps[i] = \ell_0 \times \cdots \times \ell_{i-1} \times \ell_{i+1} \times \cdots \times \ell_{n-1} = p[i-1] \times s[i+1].$$

Ceci pour $i \in [\![1\,,\, n-2]\!]$, avec les cas particuliers

$$\ell ps[0] = \ell_1 \times \cdots \times \ell_{n-1} = s[1] \quad \text{et } \ell ps[n-1] = \ell_0 \times \cdots \times \ell_{n-2} = p[n-2].$$

On déduit alors le script suivant :

```
def prodSauf3(lst) :
    """
        Renvoie la liste de même taille que lst dont le \
        ↪ terme d'indice
        contient le produit de tous les éléments de lst \
        ↪ sauf celui d'indice i
    """
    p, s = pref_suff(lst)
    lps = [s[1]]
    for i in range(1, len(lst)-1) :
        lps.append(p[i-1]*s[i+1])
    lps.append(p[-2])
    return lps
```

```
>>> prodSauf3([1, 2, 3, 4, 5])
[120, 60, 40, 30, 24]
```

6. Voici par exemple le script permettant de remplir une liste de 5 entiers naturels tirés au sort entre 1 et 9, puis appeler la fonction `prodSauf1(lst)` et afficher son temps de calcul :

```
>>> import random as rd
>>> import time as ti
>>> n = 10    # taille de la liste
>>> max = 9    # les entiers de la liste
>>>         # sont choisis entre 1 et max inclus
>>> lst = []
>>> for _ in range(n) :
...     lst.append(rd.randint(1, max))
...
>>> t1 = ti.time()
>>> lps = prodSauf1(lst)
>>> t2 = ti.time()
>>> print("Durée de prodSauf1 : ",t2-t1)
Durée de prodSauf1 :  2.5365891456604004
>>> t1 = ti.time()
>>> lps = prodSauf2(lst)
>>> t2 = ti.time()
>>> print("Durée de prodSauf2 : ",t2-t1)
Durée de prodSauf2 :  4.1961669921875e-05
>>> t1 = ti.time()
>>> lps = prodSauf2(lst)
>>> t2 = ti.time()
>>> print("Durée de prodSauf3 : ",t2-t1)
Durée de prodSauf3 :  3.1948089599609375e-05
```

Voici les durées (en secondes) mis par les appels des trois versions, calculés à chaque fois sur une même liste de taille n, pour différentes valeurs de n et pour `max = 9` (à partir de $n = 20$, le temps de calcul de `prodSauf1` n'est plus indiqué car il est trop long).

n	`prodSauf1`	`prodSauf2`	`prodSauf3`
5	$1.3\,10^{-3}$	$2.9\,10^{-5}$	$2.7\,10^{-5}$
10	13.3	$1.0\,10^{-4}$	$3.3\,10^{-5}$
20		$2.0\,10^{-4}$	$8.8\,10^{-5}$
50		$5.3\,10^{-4}$	$9.5\,10^{-5}$
100		$3.8\,10^{-3}$	$2.4\,10^{-4}$
1000		$3.7\,10^{-1}$	$1.5\,10^{-2}$
10000		164	2.1

On constate que la première version a un temps de calcul beaucoup plus long que les deux autres, même pour de petites listes, et devient vite prohibitif. Les deux autres versions ont un temps de calcul bien meilleur, plus court d'un facteur 10 environ pour la dernière version, et ce facteur semble s'accroître pour de très grandes listes (les temps de calcul peuvent évoluer légèrement sur plusieurs exécutions pour la même taille de liste puisque les valeurs remplissant ces listes sont choisies aléatoirement).