Mathématiques supérieures — le 7 Novembre 2025

Corrigé du TP n°2 - Recherches séquentielles (suite)

Dans tout ce qui suit, l'appellation "tableau" pour une variable Python désignera indéferemment une liste, un tuple ou une chaîne de caractères.

1. Recherche d'un maximum

Dans cette partie, nous allons nous intéresser à la recherche d'un maximum dans une liste d'éléments qui peuvent être comparés par un relation d'ordre.

Principe de recherche séquentielle d'un maximum dans un tableau. On suppose que le tableau contient au moins un élément.

- On crée une variable `max_en_cours` à laquelle on affecte la valeur du premier élément du tableau.
- On parcourt un à un les éléments du tableau et pour chacun des élément, si sa valeur a est (strictement) plus grande que la valeur de `max_en_cours`, on affecte à `max_en_cours` la valeur a ; et sinon, on ne fait rien
- Une fois le parcours du tableau fini, on renvoie la valeur de `max_en_cours`.

Remarque. Important !

- On prouvera plus tard dans l'année que l'algorithme décrit précédemment renvoie bien la valeur maximale d'un tableau grâce à un *invariant de boucle* : ici, on peut montrer que l'assertion "*à la fin du i-ème tour de boucle, `max_encours` vaut le maximum des i premiers éléments du tableau*" est vrai à chaque tour i du parcours. On dira que cette assertion est un invariant de boucle et il nous permettra, une fois démontré, de prouver que notre algorithme fait bien ce qu'on lui demande (on parle de *correction* de l'algorithme).
- On remarque qu'avec cet algorithme, on parcourt une seule fois le tableau (i.e. on "regarde" chaque élément du tableau une seule fois), dans ce cas, on dit que la **complexité temporelle** de l'algorithme est **linéaire** : le tableau est de taille n et on fait n "opérations". En fait, on dira de même que la compléxité est linéaire si la nombre d'opérations est une fonction linéaire de n (ou même affine).
  Si on avait fait n^2 "opérations", on aurait dit que la complexité temporelle est **quadratique** (nous en verrons bientôt).
  Nous étudierons plus rigoureusement la notion de complexité temporelle plus tard dans l'année.

Dans ce T.P., on s'efforcera à programmer des fonctions de complexité linéaire.

Q1 : Écrire une fonction `maximum(L)` qui renvoie la valeur maximale des éléments du tableau `L` si `L` est non vide et `None` sinon.
Exemple de comportement attendu :

```
>>>maximum((3,9,152,1,4))
152
```

```
>>>maximum([])

>>>maximum('avion')
'v'
```

Q2 : Écrire une fonction `mini_maxi(L)` qui, si `L` est non vide, renvoie un tuple `(m,M)` où `m` est la valeur minimale des éléments du tableau `L`, `M` est la valeur maximale des éléments du tableau `L` ; et `None` sinon.

Q3 : **a)** Écrire une fonction `indice_maximum(L)` qui renvoie l'indice de la première occurrence du maximum du tableau `L` si `L` est non vide et `None` sinon.

b) Que faut-il changer pour obtenir cette fois-ci l'indice de la dernière occurrence du maximum (programmer ceci dans une fonction `indice_maximum2(L)`)
Exemples de comportement attendu :

```
>>>indice_maximum((3,3,152,1,152))
2
>>>indice_maximum2((3,3,152,1,152))
4
>>>indice_maximum('blabla')
1
```

c) Écrire une fonction `liste_indices_maximum(L)` qui renvoie la liste des indices de chaque occurence du maximum du tableau `L`.

Q4 : **a)** Écrire une fonction `deuxieme_maximum(L)` qui renvoie la deuxième valeur maximale de `L` (donc pas la valeur maximale mais celle juste strictement inférieure). On supposera pour cette question que `L` possède au moins 2 éléments dont les deux premiers sont différents)

b) Écrire une fonction `deuxieme_maximum2(L)` qui renvoie la deuxième valeur maximale de `L` s'il existe deux éléments différents dans `L` et la valeur maximale si tous les éléments sont égaux. (On supposera seulement ici que la liste possède au moins deux éléments).

Q5 : Importer la fonction `randint` du module `random`

```
1 from random import randint
```

a) Écrire une fonction `liste_aleatoire(n,k)` qui renvoie une liste de longueur `n` dont les éléments sont des entiers tirés aléatoirement et uniformément dans $[\![1,k]\!]$.

b) Écrire une fonction `moyenne_maximum(n,k,nb)` qui renvoie la moyenne des maximums de `nb` listes aléatoires créées avec `liste_aleatoire(n,k)`.
Quelle valeur semble prendre `moyenne_maximum(3,10,nb)` quand `nb` devient grand ? Peut-on retrouver cette valeur mathématiquement ?

Correction.

Q1 :

```python
def maximum(L):
    if len(L) == 0:
        return None
    max_en_cours = L[0]
    for i in range(1,len(L)):
        if L[i] > max_en_cours:
            max_en_cours = L[i]
    return max_en_cours
```

Q2 :

```python
def mini_maxi(L):
    if len(L) == 0:
        return None
    min_en_cours,max_en_cours = L[0],L[0]
    for i in range(1,len(L)):
        if L[i] < min_en_cours:
            min_en_cours = L[i]
        if L[i] > max_en_cours:
            max_en_cours = L[i]
    return min_en_cours,max_en_cours
```

Q3 : a)

```python
def indice_maximum(L):
    if len(L) == 0:
        return None
    max_en_cours = L[0]
    ind_en_cours = 0
    for i in range(1,len(L)):
        if L[i] > max_en_cours:
            max_en_cours = L[i]
            ind_en_cours = i
    return ind_en_cours

# ou plus concis (mais doit accéder à L[ind_en_cours] à chaque tour) :

def indice_maximum(L):
    if len(L) == 0:
        return None
    ind_en_cours = 0
    for i in range(1,len(L)):
        if L[i] > L[ind_en_cours]:
            ind_en_cours = i
    return ind_en_cours
```

b) Il suffit de changer le `>` de `L[i] > max_en_cours` par un `>=` :

```python
def indice_maximum2(L):
    if len(L) == 0:
        return None
    max_en_cours = L[0]
    ind_en_cours = 0
    for i in range(1,len(L)):
        if L[i] >= max_en_cours:
            max_en_cours = L[i]
            ind_en_cours = i
    return ind_en_cours
```

c)

```python
def indice_maximum(L):
    if len(L) == 0:
        return None
    max_en_cours = L[0]
    liste_en_cours = [0]
    for i in range(1,len(L)):
        if L[i] == max_en_cours:
            liste_en_cours.append(i)
        if L[i] > max_en_cours:
            max_en_cours = L[i]
            liste_en_cours = [i]
    return liste_en_cours
```

Q4 : a)

```python
def deuxieme_maximum(L):
    m1,m2=L[0],L[1]
    if m1 < m2:
        m1,m2 = m2,m1
    for i in range(2,len(L)):
        if L[i] > m1:
            m2 = m1
            m1 = L[i]
        elif L[i]!=m1 and L[i] > m2:
            m2 = L[i]
    return m2
```

b)

```python
def deuxieme_maximum2(L):
    m1,m2=L[0],L[1]
    if m1 < m2:
        m1,m2 = m2,m1
    for i in range(2,len(L)):
        if L[i] > m1:
            m2 = m1
            m1 = L[i]
        elif m1 == m2 or (L[i]!=m1 and L[i] > m2):
            m2 = L[i]
    return m2
```

Q5 : Importer la fonction `randint` du module `random`

```python
from random import randint
```

a)

```python
def liste_aleatoire(n,k):
    L=[]
    for i in range(n):
        L.append(randint(1,k))
    return L

#ou, plus concis avec la syntaxe Python

def liste_aleatoire(n,k):
    return [randint(1,k) for i in range(n)]
```

b)

```python
def moyenne_maximum(n,k,nb):
    somme = 0
    for i in range(nb):
        somme += maximum(liste_aleatoire(n,k))
    return somme/nb
```