TD1 : Algorithmique des tableaux

Compétences

- Itérer dans un tableau avec les structures de contrôle for/while
- Lire un programme et prévoir le résultat à l'aide d'un raisonnement structuré
- Modifier les données d'un tableau
- Être capable de compter les affectations ou les comparaisons d'un programme.

Introduction

Pour tous les exercices manipulant des tableaux, on distinguera la taille $nMax$ d'un tableau et le nombre d'éléments n qu'il contient. On entend par taille d'un tableau le nombre de cases de mémoire qui le composent. Un tableau ne peut pas contenir plus d'élements que sa taille. Un tableau pourra par exemple être de taille 10 (c'est à dire qu'il dispose de 10 cases) et contenir moins de 10 éléments. Les éléments seront systématiquement rangés dans les cases d'indices les plus petits possibles, soit de 0 à $n-1$. Par exemple, si un tableau a une taille égale à 10 et contient seulement 4 éléments, ils seront rangés dans les cases d'indice 0 à 3.

Pour connaître la taille $nMax$ d'un tableau on fera appel en Python à la fonction `len` qui prend en paramètre un objet itérable (un tableau est un objet itérable) et renvoie sa taille. Puisque le nombre d'éléments n contenus dans un tableau t est inférieur ou égal à sa taille `len(t)`, il sera nécessaire, chaque fois que l'on passe un tableau en paramètre à une fonction, de passer aussi en paramètre le nombre n de ses éléments.

Exercices

** **Ex. 1 —** Soit la fonction `toStartWithArray` suivante :

```
1 def toStartWithArray (t, n):
2     m = t[0]
3     for i in range(n-1):
4         if t[i] < m :
5             m = t[i]
6     return m
```

1. En utilisant `t =` | 3 | 5 | 7 | 1 | 10 | 2 | 7 | -3 | comme tableau de valeurs, simulez l'exécution de la fonction `to_start_with_array` en complétant le tableau ci-dessous qui permet de suivre l'évolution des variables : `m, n, i` et `t[i]`.

	`m`	`n`	`i`	`t[i]`
Avant la boucle				
Dans la boucle				
	...	...	...	...

2. Quel sera en général le résultat de l'exécution de la fonction `to_start_with_array` pour `n>1` ?
3. Quel est le nombre de comparaisons effectuées par la fonction ?
4. Quel est le nombre d'affectations effectuées par la fonction ?

** **Ex. 2 —** Écrire une fonction `insert(t, n, elt, k)` qui, étant donné un tableau `t` contenant `n` éléments (avec `n<len(t)`) insère un élément `elt` à la position `k` avec $k \geqslant 0$. Si $k \geqslant n$, l'élément sera ajouté à l'indice `n`. L'ordre initial des éléments du tableau sera conservé. La fonction doit renvoyer le nouveau nombre d'éléments du tableau.

** **Ex. 3 —** Écrire une fonction `delete(t, n, k)` qui supprime l'élément situé à la position `k` du tableau `t` contenant `n` éléments en supposant $0 \leqslant k < n$. L'ordre initial des éléments du tableau sera conservé. La fonction doit renvoyer le nouveau nombre d'éléments dans le tableau.

* **Ex. 4 —** Écrire une fonction `amplitude(t,n)` qui, en parcourant **une seule fois** le tableau `t`, calcule et renvoie la différence entre le plus grand et le plus petit élément parmi les n premiers éléments du tableau t.

Exemple : Soit `t = [2, 8, 11, 5, 9, 3, 1]` alors `amplitude(t, 7)` renvoie 10 et `amplitude(t, 4)` renvoie 9.

** **Ex. 5 —** On souhaite écrire une fonction `max2(t, n)` qui calcule et renvoie la deuxième plus grande valeur parmi les `n` premiers éléments d'un tableau `t`..

Exemple : Soit `t = [5, 3, 4, 6, 1, 10, 2, 10, 4]`, `max2(t, 5)` doit renvoyer 5, mais `max2(t, 9)` doit renvoyer 10.

1. Proposer un algorithme
2. Quel est le nombre de comparaisons effectuées par votre algorithme ?
3. Implémenter une fonction `max2(t, n)`

** **Ex. 6 —** Écrire une fonction `deleteFirstInstance(t, n, elt)` permettant d'enlever du tableau `t` la première occurrence d'un élément `elt` passé en paramètre (le tableau `t` ne sera pas modifié si `elt` n'appartient pas à `t`). L'ordre initial des éléments du tableau sera conservé. La fonction doit renvoyer le nouveau nombre d'éléments dans le tableau.

** **Ex. 7 —** Soit la fonction Python suivante :

```
1 def myFunction(t, n, x):
2     stop = False
3     while n >0 and not stop :
4         numberOfElement = deleteFirstInstance(t, n, x ) # Ex. 4
5         if numberOfElement == n :
6             stop = True
7         else :
8             n = numberOfElement
9     return n
```

1. Soit `t = [2, -7, 4, 5, 12, 10, 4, 2, 4, -18]` un tableau contenant 10 éléments. Simulez l'exécution de `myFunction(t, 10, 4)` en complétant le tableau suivant pour montrer l'évolution des variables `n`, `stop`, et `numberOfElement`. Précisez également le contenu de `t` à chaque étape.

`n`		
`stop`		
`numberOfElement`		

2. Quel est le nombre d'éléments du tableau décalés dans le meilleur et dans le pire des cas ?
3. Quel est le nombre de comparaisons concernant des éléments du tableau dans le meilleur et dans le pire des cas ?

*** **Ex. 8 —** Écrire une fonction `deleteInstances(t, n, elt)` permettant d'enlever du tableau `t` à `n` éléments toutes les occurrences de l'élément `elt`. La fonction devra renvoyer le nombre d'éléments du tableau à la fin du traitement et ne devra effectuer **qu'un seul parcours** du tableau.
Quel est le nombre de décalages d'éléments du tableau nécessaires dans le meilleur et dans le pire des cas ?

Pour s'entraîner sur quelques fondamentaux

** **Ex. 9 —** Soit `t` un tableau d'entiers. On appelle monotonie de `t`, une partie de `t` triée dans l'ordre croissant. Si `t` est trié dans l'ordre croissant, sa plus longue monotonie est lui-même. Si T est trié dans l'ordre décroissant, sa plus longue monotonie ne contient qu'un élément.
Écrire une fonction `monotonicity(t, n)` qui renvoie la longueur de la plus longue monotonie d'un tableau d'entiers `t` de `n` éléments ainsi que l'indice auquel commence cette monotonie.

* **Ex. 10 —** Écrire une fonction `search`, prenant en paramètre un tableau non vide `tab` d'entiers, un entier `n` correspondant au nombre d'éléments dans le tableau, un entier `p` et qui renvoie l'indice de la dernière occurrence de `p`. Si l'entier `p` n'est pas présent, la fonction renvoie : `-1`

* **Ex. 11 —** Écrire une fonction `swap(tab, i, j)` qui permute deux éléments d'un tableau non vide `tab`, dont les positions sont `i` et `j`.

** **Ex. 12 —** Soit `t` un tableau contenant `n` nombres rangés dans l'ordre croissant, avec `n < len(t)`. Écrire une fonction `insert_order(t, n, elt)` qui insère un nouvel élément `elt` dans `t` en respectant l'ordre croissant et qui renvoie le nouveau nombre d'éléments dans le tableau.

Exemple : Soit `t =[2, 4, 7, 10, 15, 20, 25, None, None, None]`, l'appel `insert_order(t, 7, 5)` va renvoyer 8 (nombre d'éléments dans t après l'insertion) et va aussi modifier le contenu de **`t`** en `[2, 4, 5, 7, 10, 15, 20, 25, None, None]`.

** **Ex. 13 —** Écrire une fonction **`separate`** qui prend en argument un tableau **`t`** dont les éléments sont des 0 et des 1 et qui sépare les 0 des 1 en plaçant les 0 au début de tableau et les 1 à la suite.

*** **Ex. 14 —** Écrire une fonction **`isPortion(t, nt, s, ns)`** qui prend en paramètre deux tableaux d'entiers **`t`** et **`s`** contenant respectivement **`nt`** et **`ns`** éléments et qui renvoie True si **`s`** correspond à une section du tableau **`t`** et False sinon. Une section est une suite éventuellement vide d'éléments contigus dans un tableau.

Exemple : Soient `t = [5, 1, 2, 3, 1, 2, 1]` et `s = [1, 2]` dans ce cas la fonction renvoie True. Elle renvoie False pour `t = [1, 2, 3, 4, 1]` et `s = [1, 3]`.

** **Ex. 15 —** Soient les variables **`student`** et **`score`** qui référencent deux tableaux de même longueur. Ces tableaux contiennent respectivement le nom des élèves de la promo et les notes obtenues au dernier devoir d'informatique. Écrire une fonction **`highestScore`** qui renvoie les noms des élèves ayant obtenus la note maximale.