TD2 : Algorithmique des tableaux

Compétences

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

Exercices

** **Ex. 1 —** Deux entiers x et y sont opposés si leur somme est égale à 0. Dans cet exercice on appelle couple d'opposés dans un tableau t d'entiers, deux éléments de `t` d'indices `i` et `j` tels que `t[i]+t[j]` est égale à 0 ($0 <= i < n$ et $0 <= j < n$).
Soit `t` un tableau d'entiers, écrivez une fonction `sansOppose(t: array, n: int)->boolean:` qui prend en entrée un tableau `t` de taille `n` et retourne `True` si `t` ne contient pas de couple d'opposés et `False` sinon.
Dans le pire des cas combien de comparaisons seront nécessaires pour un tableau `t` de `n` entiers? Justifiez votre réponse et précisez quel est le pire des cas.

** **Ex. 2 —** Algorithme du nombre maximum d'occurrences dans un tableau. On désire connaître l'élément qu'on retrouve le plus dans un tableau. Il s'agit donc de trouver le nombre maximum d'occurrences des éléments d'un tableau. En cas d'égalité, on choisit celui trouvé en premier. Par exemple, si nous avons les tableaux suivants :

- [0, 1, 4, 2, 3, 2, 2] , l'élément le plus présent est 2 avec 3 occurrences.
- [6, 10, 6, 10, 7, 3] , l'élément le plus présent est 6 avec 2 occurrences.
- [5, 8, 5, 5, 8, 8] , l'élément le plus présent est 5 avec 3 occurrences.

1. Écrire une fonction `occurrenceMax(t:array,n:int)->int` qui prend en argument un tableau `t`, son nombre d'éléments `n` et qui implémente un algorithme naïf renvoyant l'élément ayant le plus grand nombre d'occurrences.
2. Combien d'itérations comporte votre algorithme?

** **Ex. 3 —** Soit `t` un tableau de `n` entiers. On considère la fonction mystere ci-dessous.

```
1  def mystere(t, n):
2      c, v = -1, 0
3      for i in range(n):
4          if v == 0:
5              c = t[i]
6              v = 1
7          else:
8              if t[i] == c:
9                  v = v + 1
10             else:
11                 v = v - 1
12     compteur = 0
13     for j in range(n):
14         if t[j] == c:
15             compteur += 1
16     if compteur > n // 2:
17         return c
18     else:
19         return -1
```

1. Complétez le tableau ci-dessous en indiquant les valeurs successives des variables lorsqu'on appelle la fonction mystere pour `t = array('l', [5,2,1,2,8,2,2,7,2,5,2,2,5])` et `n=13`.

i	/	...	
c	-1	...	
v	0	...	
j		...	
compteur		...	

2. Quelle sera la valeur retournée?
3. Quelle sera la valeur retournée lorsqu'on appelle la fonction mystere pour `t=[5,2,1,2,8,3]` et `n=6`?

4.Dans le cas général que fait la fonction `mystere (t,n)` ? Quelle est sa complexité en temps ?

*** **Ex. 4 —** Algorithme du sous-tableau maximum : le problème du sous-tableau maximum est la méthode permettant de trouver le sous-tableau contigu dans un tableau d'entiers relatifs ayant la plus grande somme. Le problème a été proposé à l'origine par Ulf Grenander de l'Université Brown en 1977.
Par exemple, pour les tableaux suivants :

- `[0, -1, 2, -2, 3, 2]`, la somme est de **5** avec le sous-tableau maximal `[3, 2]`,
- `[1, -10, 9, 6, 8, -6]`, la somme est de **23** avec le sous-tableau maximal `[9, 6, 8]` ,
- `[6, 10, -6, -1, 7, 3]` la somme est de **19** avec le sous-tableau maximal `[6, 10, -6, -1, 7, 3]`,
- `[-4, 9, -3, -5, -8, 5]`, la somme est de **9** avec le sous-tableau maximal `[9]`.

1.Écrire une fonction `sequenceMax(t: array, n: int)->int:` qui prend en argument un tableau, son nombre d'éléments et qui implémente un algorithme naïf renvoyant la somme max en testant toutes les sous séquences d'éléments consécutifs possibles.

```
Exemple :

t = array('h', [0, -1, 2, -2, 3, 2])
sequenceMax(t, len(t))
# Résultat:
5
```

2.Quelle la compleité en temps de la fonction `sequenceMax` ?

** **Ex. 5 —** Soit la fonction `mystere` qui prend en paramètres un tableau **t** contenant un nombre **n** d'éléments :

```
1  def mystere(t, n):
2      m = 0
3      cm = 0
4      for i in range(n):
5          cm += t[i]
6          if cm < 0:
7              cm = 0
8          if m < cm:
9              m = cm
10     return m
```

1.Remplir le tableau ci-dessous avec l'appel

```
mystere(array('l',[5, 9, -6, -9, -5, 3, 15, -4, 5, -11]), 10)
```

i										
cm										
m										

2.Que fait la fonction `mystere` dans le cas général ? Préciser dans votre réponse ce que représente **cm**.

3.Donner le nombre d'affectations sur **cm** et **m** dans le pire et le meilleur cas.

** **Ex. 6 —** En modifiant la fonction `isSection` de l'exercice `15` de la feuille du `TD1`, écrire une nouvelle fonction `numberSections(t, nt, s, ns)` qui calcule et retourne le nombre de fois où le tableau non vide `s` apparaît comme section du tableau `t`.

Par exemple, le tableau `[1, 2]` correspond à deux sections du tableau `[5, 1, 2, 3, 1, 2, 1]`. Autre exemple, le tableau `[1, 2, 1]` correspond à deux sections du tableau `[5, 1, 2, 1, 2, 1]`. Dans ce dernier cas, les deux sections se chevauchent : `[5, 1, 2, 1, 2, 1]` et `[5, 1, 2, 1, 2, 1]`.

** **Ex. 7 —** Soit la fonction mystere qui prend en paramètres un tableau `t` contenant un nombre `n` d'éléments.

```
1  def mystere(t:array,n:int)->int:
2      a = 0
3      b = 1
4      c = t[0]
5      for i in range(1,n) :
6          if t[i] == c:
7              b += 1
8          else:
9              if b > a:
10                 a = b
11             b = 1
12             c = t[i]
13     if b > a:
14         a = b
15     return a
```

1. Remplir le tableau ci-dessous avec l'appel `mystere(array('l',[8, 8, 7, 7, 7, 3, 7, 7, 7, 7]), 10)`

i										
a										
b										
c										
t[i] == c										
b > a										

2. Que fait la fonction mystere dans le cas général ? Préciser dans votre réponse ce que représentent a et b.
3. Donner le nombre d'affectations sur a et b dans le pire et le meilleur cas.