Université de Bordeaux
Semestre 1

2023-2024

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 —** Soit la fonction `indexSort(t, n)` où `t` est un tableau de `n` entiers :

```python
def indexSort(t, n) :
    i = 0
    while ((i + 1) < n) and (t[i] <= t[i + 1]) :
        i += 1
    return i
```

1. Que fait cet algorithme ?
2. Donner le nombre d'affectations effectuées sur la variable `i` dans la fonction `indexSort(t, n)` en fonction de la valeur de retour `i`
3. Déduire les complexités en nombre d'affectations, dans le meilleur et pire cas, de la fonction `indexSort(t, n)` en fonction de `n` (taille du tableau `t`) ?

** **Ex. 2 —** Soit la fonction `sort(t, g, d, n)` où `t` est un tableau de `n` entiers et `g` et `d` deux entiers compris entre 0 et $n-1$. La fonction `sort(t, g, d, n)` utilise la fonction du TD1 `swap(t, i, j)`.

```python
def sort(t, g, d, n):
    if g >= 0 and g < d and d <= (n -1):
        for i in range(g, d):
            for j in range(i+1, d+1):
                if t[i] > t[j]:
                    swap(t, i, j)
```

1. Si la fonction `swap` n,'a pas été écrite, en proposer une version.
2. Exécuter cet algorithme pour le tableau `[9, 5, 7, 2]` avec g=0 et d=3 , quel est le rôle de cet algorithme ?
3. Donner et justifier les complexités, dans le meilleur cas et pire cas, en nombre d'affectations sur le tableau `t` de la fonction `sort(t, g, d, n)` en fonction des paramètres `g` et `d`.

*** **Ex. 3 —** Soit la fonction `invalidSort(t, n)` où `t` est un tableau de `n` entiers :

```python
def invalidSort(t, n):
    i = indexSort(t, n)
    sort(t, i+1, n-1, n)
```

1. Expliquer pourquoi la fonction `invalidSort(t, n)` peut ne pas trier le tableau `t`. (On pourra prendre comme exemple le tableau `[2, 7, 5, 9]`)
2. Donner le nombre d'affectations effectuées sur la variable `i` et sur les variables `t[k]` du tableau lors d'un appel à la fonction `invalidSort(t, n)` en fonction de la variable `i` et de `n` (taille du tableau `t`)
3. En Déduire les complexités en nombre d'affectations de la fonction `invalidSort(t, n)` en fonction de `n` (taille du tableau `t`)

*** **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, 6)
# Résultat:
5
```

2.Quelle la complexité 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 :

```python
def mystere(t, n):
    m = 0
    cm = 0
    for i in range(n):
        cm += t[i]
        if cm < 0:
            cm = 0
        if m < cm:
            m = cm
    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]`.