

INFO0947: Multiplicité

Groupe 21: Maxime DERAUVET, Luca MATAGNE

Table des matières

1	spécification du module "multiplicité"	3
2	Nos Sous-Problèmes	3
2.1	SP 1 : Obtenir le max du tableau	3
2.2	SP 2 : Compter le nombre d'occurence du max	3
3	Vérification de la complexité	4
4	Code	4
4.1	multiplicite.h	4
4.2	main.c	4
4.3	multiplicite.c	5

1 spécification du module "multiplicité"

```
1 /**
2  * multiplicite
3  *
4  * @pre : N > 0, T est un tableau d'entier de taille
5  *
6  * @post: max contient le maximum du tableau
7  *
8  * @return: le nombre d'occurrence de max
9  *
10 */
11
12
13 int multiplicite(int *T, const int N, int *max);
```

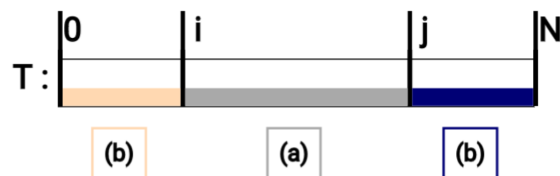
Extrait de Code 1 – Spécifications

2 Nos Sous-Problèmes

2.1 SP 1 : Obtenir le max du tableau

La notation du sous-problème consistant à obtenir le maximum du tableau est la suivante : $\text{MaximumSSTab}(T, i, j, N) \equiv 0 \leq i \leq j < N$

Dans ce sous problème, nous allons introduire notre boucle while. Voci donc l'invariant de boucle qui nous a permis d'écrire notre code :



$b = \text{Maximum trouvé} \parallel a = \text{Éventuel maximum à trouver}$

L'invariant formel qui découle de cet invariant graphique est naturellement en lien avec la notation de notre sou-problème(en début de sous-section), le voici :

$\text{max} = \text{MaximumSSTab} \wedge 0 \leq i \leq j < N$

Fonction de terminaison : $j - i$

2.2 SP 2 : Compter le nombre d'occurrence du max

Il est important de comprendre que ce sous problème n'est pas la suite du premier mais est bel et bien inclus dans le premier sous-problème. On va donc faire attention à exécuter les instructions de ce sous problème À CHAQUE ITÉRATION DE NOTRE BOUCLE (dont l'invariant est rappelé dans le SP1).

Le principe va être de créer une variable "occurrence" et d'incrémenter cette variable de 1 à chaque fois qu'un même maximum est détecté. Dans le cas ou un nouveau maximum est trouvé, cette variable est naturellement ramenée à 1.

3 Vérification de la complexité

Pour ce programme, notre complexité est de l'ordre de $N/2$. Voici notre raisonnement pour arrivé à ce résultat. $T(\text{multiplicité}) = T(A) + t(B)$

où $T(A)$ est la boucle while et $T(B)$ est la seule instruction en dehors de la boucle

$$= T(A) + 1$$
$$= 1 + \sum_{i=0}^{N/2} (A_1 + A_2 + A_3 + A_4 + A_5)$$

où A_1, A_2, A_3, A_4 sont les instructions conditionnelles dans la boucle et A_5 est l'incréméntation du "i" et la décréméntation de "j".

$$= 1 + \sum_{i=0}^{N/2} (1 + 1 + 1 + 1 + 2)$$
$$= 1 + N/2 \in O(N/2)$$

4 Code

4.1 multiplicite.h

```
1
2 #ifndef __MULTI__
3 #define __MULTI__
4
5 /**
6  * multiplicite
7  *
8  * @pre : N > 0, T est un tableau d'entier de taille
9  *
10 * @post: max contient le maximum du tableau
11 *
12 * @return: le nombre d'occurence de max
13 *
14 **/
15
16
17 int multiplicite(int *T, const int N, int *max);
18
19
20 #endif
```

Extrait de Code 2 – Header

4.2 main.c

```
1
2 #include <stdio.h>
3 #include "multiplicite.h"
4
5
6
7 int main(){
8
9     int T[10] = {16, 16,10, 16,160, 16, 16, 16, 16, 16};
10    int max = 0;
11    multiplicite(T,10,&max);
12    printf("%d - %d\n",multiplicite(T,10,&max),max );
13}
```

14
15
16 }

Extrait de Code 3 – Main

4.3 multiplicite.c

```
1
2 #include "multiplicite.h"
3 #include <assert.h>
4
5 int multiplicite(int *T, const int N, int *max){
6
7     assert (N>0);
8     assert (T!=0);
9
10    int i = 0;
11    int j = N-1;
12    int temp = 0;
13    int occurrence = 0;
14    int nbri;
15    int nbrj;
16    int maxi;
17
18
19    while (i <= j){
20        nbri = T[i];
21        nbrj = T[j];
22
23
24        //Nombre le plus grand entre les deux valeurs actuelles
25
26        if (i!=j){
27            if (nbri > nbrj){
28                temp = nbri;
29            }//fin if i>j
30
31            else{
32                temp = nbrj;
33            }//fin else
34        }
35        else
36            temp = nbri;
37
38
39        //Changement du maximum et du nombre d'occurrence
40
41        if (i ==0){ //Premier tour de boucle
42            maxi = temp;
43            if ((nbri == nbrj)&&(i!=j))
44                occurrence = 2;
45            else
46                occurrence = 1;
47        }//fin if i ==0
48
49        if (maxi < temp){ //si le nouveau nombre est plus grand, changement et reinitialisation
50            de l'occurrences
            maxi = temp;
```

```

51         if ((nbri == nbrj)&&(i!=j)) // si les deux valeurs sont égale et qu'on regarde deux
valeur différente => (Occurence ==2) sinon occurrence ==1
52             occurrence = 2;
53         else
54             occurrence = 1;
55     }//fin if max<temp
56
57
58     else if ((maxi == temp) && (i!=0)){//Si le nouveau nombre est égal, augmentation de
l'occurence
59
60         if ((nbri == nbrj)&&(i!=j)) // si les deux valeurs sont égale et qu'on regarde deux
valeur différente => (Occurence +2) sinon occurrence +1
61             occurrence += 2;
62         else
63             occurrence += 1;
64     }//fin if maxi == temp
65
66
67     //incréméntation des compteurs
68     i++;
69     j--;
70
71
72
73 }//fin while
74
75 *max = maxi;
76
77 return occurrence;
78
79 }//fin multiplicite

```

Extrait de Code 4 – multiplicite.c