INFO0947: Projet 1 - Multiplicite

Groupe 4: SeyedPouria SALEHI KATOZI,

20/06/2021

Table des matières

1 Introduction :

L'objectif principal de ce projet est de créer une fonction nommée "multiplicite" qui nous permettra de trouver la valeur maximale présente à l'intérieur d'un tableau d'entiers et de compter le nombre d'occurences de cette même valeur dans ce tableau.
Notons que ce programme doit respecter certaines contraintes :

- Il doit respecter une complexité meilleure que linéaire O(N)
- Il ne peut contenir au maximum qu'une boucle de type while.

2 Formalisation du problème :

2.1 Notations :

Pour commencer, afin d'aborder le projet sous le meilleur angle possible, nous allons choisir les notations les plus adéquates pour représenter les parties du problème.
Ces notations et leur définition (prédicats) sont introduits ci-dessous conformément aux bonnes règles de procédure :

Soit Max(x,y) une notation telle que, retourne le maximum entre d'entier x et y (Référence : Université de Liège INFO 0947 - Slide 22-24 - Autor : Bénoit Donnet).

Soit le prédicat MaximumSSTab(T, i, j, N) pour obtenir la valuer maximale du tableau :
MaximumSSTab(T, i, j, N), $\equiv 0 \leqslant$ i $\leqslant$ j $<$ N, max = max($max_{T[0...i]}$, $max_{T[j...N[}$).

Soit le prédicat NbMaxSSTab ab(T, i, j, N, max) pour obtenir l'occurance de valuer maximale du tableau :
NbMaxSSTab ab(T, i, j, N, max) $\equiv$ occ = #i $\bullet$(0 $\leqslant$ i $<$ N | max = T[i]).

3 Analyse du problème et découpe en sous-problèmes :

Pour rappel, le problème général consiste à trouver le maximum et son occurence dans le tableau de sorte qu'à la fin, la post condition de cette fonction soit le prédicat suivant :
T = T_0 $\wedge$ N = N_0 $\wedge$ 0 $\leqslant$ j $\leqslant$ i $\leqslant$ N-1 $\wedge$
occ = multiplicite(T, N, max) $\wedge$
max = Max(MaximumSSTab(T, 0, i), MaximumSSTab(T, j, N-1))

3.1 SP1 :

Trouver la valuer maximale du tableau

3.1.1 Input :

- valeur1, un entier.
- valeur2, un entier.
- occ, un entier.
- *max, un pointeur vers un entier.

3.1.2 Output :

- retourne l'occurance de valeur maximale

3.1.3 Objet Utilisé :

- int valeur1, est un entier.
- int valeur2, est un entier.
- int occ, un entier.
- int *max, pointe vers un entier.

3.1.4 Spécification :

```
/*PréCondition :  max initialisé ∧ occ ⩾ 0
 *
 *PostCondition : max = MaximumSSTab(T, i, j, N-1) ∧
 *                 occ = count_occ(T[i],T[j],max,occ)
 */
 static int counr_occ(int valeur1, int valeur2, int *max, int occ);
```

3.2 SP2 :

Déterminer l'occurance de la valeur maximale

3.2.1 Input :

- T, tableau.
- N, la taille du tableau.
- *max, un pointeur vers un entier.

3.2.2 Output :

- retourner l'occurence de maximum du tableau.

3.2.3 Objet Utilisé :

- int *T, est un tableau d'entier.
- const int N, est une valeur entier N > 0.
- int *max, un pointeur d'entier.

3.2.4 Spécification :

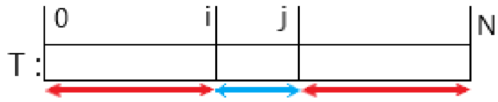
```
/*PréCondition :  T initialisé ∧ N>0 ∧ max pointe vers un entier
 *
 *PostCondition : T = T₀ ∧ N = N₀
 *               ∧ max = Max(MaximumSSTab(T, 0, i), MaximumSSTab(T, j, N-1))
 *               ∧ occ = multiplicite(T, N, max)
 */
 int multiplicite(int *T, const int N, int* max);
```

4 Construction :

4.1 Invariant :

4.1.1 Graphique :

Comme la complexité devant être meilleure que linéaire, il est souhaitable de parcourir le tableau à partir de plusieurs positions à la fois. La technique pour laquelle nous opterons ici consiste à parcourir les différents éléments depuis les extrémités vers le centre du tableau.



Encore à traiter

La valeure maximale est déjà inséré dans *max et occ contient l'occurance de cette valeur maximale.

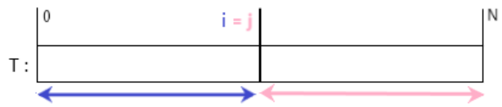
4.1.2 Formel :

Par conséquent, les parties extérieures aux deux zones d'observation du tableau T seront donc déjà parcourue (les zones d'observation sont les cases d'indice i et j), on en connaît la plus grande valeur et son occurrence sur cette zone.
Cette zone dès lors parcourue sera représentée mathématiquement par le prédicat suivant :

$\text{INV} \equiv 0 \leqslant \text{i} \leqslant \text{j} < \text{N} \wedge \text{T} = T_0 \wedge \text{N} = N_0 \wedge \text{occ} = \text{multiplicite(T, N, max)} \wedge$
$\text{max} = \text{Max(MaximumSSTab(T, 0, i), MaximumSSTab(T, j, N-1))}$

4.2 INIT :

4.3 Critère d'arrêt :



$(\neg \text{B}) \equiv \text{j} \leqslant \text{i}$

4.4 Gardien-Boucle :

$(\neg(\neg \text{B})) \equiv \neg(\text{j} \leqslant \text{i}) = \text{j} > \text{i}$

4.5 Fonction de terminaison :

$\text{F} \equiv \text{j - i}$

5 Code complet :

```
static int count_occ(int valeur1, int valeur2, int *max , int occ){
   assert(max != NULL && occ >= 0);
   {Pré = max initialisé ∧ occ ⩾ 0}
   if(valeur1 == *max){   //ZONE A
   occ++; {max = valeur1 ∧ occ = (occ + 1)} //ZONE A
   }
   if(valeur2 == *max){ //ZONE B
   occ++; {max = valeur2 ∧ occ = (occ + 1)} //ZONE B.1
   }
   else if(valeur1 > *max && valeur2 > *max && valeur1 == valeur2){ //ZONE B.2
   occ = 2; {valeur1 = valeur2 ∧ valeur1 > max ∧ valeur2 > max ∧ occ = 2}
   }
   else if(valeur1 > *max || valeur2 > *max){ //ZONE B.3
   occ = 1; {valeur1 > max ∨ valeur2 > max ∧ occ = 1}
   }
   return occ;{Post: max initialisé ∧ valeur1 = valeur1_0 ∧ valeur2 = valeur2_0 ∧
   occ = count_occ(valeur1, valeur2, max, occ)}
   }
```

```
int multiplicite(int *T, const int N, int* max){
   assert(T != NULL && N > 0 && max != NULL);
   {Pré ≡ T = T_0 ∧ N = N_0 ∧ max initialisé} //ZONE A
   unsigned int i = 0;
   unsigned int j = N-1;
   int occ = 0;
   *max = T[0];
   {T = T_0 ∧ N = N_0 ∧ max initialisé ∧ i = 0 ∧ j = N-1 ∧ occ = 0 ∧ *max = T[0]}
//ZONE A
   while(i < j){ //ZONE B
   {INV ∧ B ≡ T = T_0 ∧ N = N_0 ∧ 0 ⩽ i < j ⩽ N-1 ∧ occ = multiplicite(T,N,max)}
   if(T[i] > T[j]){ //ZONE B.1
      if(T[i] > *max){
{T = T_0 ∧ N = N_0 ∧ 0 ⩽ i < j ⩽ N-1 ∧ T[i]>T[j] ∧ occ = multiplicite(T,N,max)}
         occ = 0; //ZONE B.1.1
         *max = T[i]; //ZONE B.1.2
{T = T_0 ∧ N = N_0 ∧ 0 ⩽ i < j ⩽ N-1 ∧ *max = T[i] ∧ occ = 0}
         }
         occ = count_occ(T[i],T[j], max, occ); //ZONE B.1.3
{T = T_0 ∧ N = N_0 ∧ 0 ⩽ i < j ⩽ N-1 ∧ *max = T[i] ∧ T[i] = T[i]_0 ∧ T[j] = T[j]_0
∧ occ = count_occ(T[i], T[j], max, occ)} //ZONE B.1
      } //ZONE B.2
      else if(T[i] == T[j]){
      if(T[i] == *max){ //B.2.1
          occ = count_occ(T[i],T[j], max, occ); //ZONE B.2.1
{T = T_0 ∧ N = N_0 ∧ 0 ⩽ i < j ⩽ N-1 ∧ *max = T[i] ∧ T[i] = T[j]
∧ occ = count_occ(T[i],T[j],max,occ)}
      }
      if(T[i] > *max){ //ZONE B.2.2
        occ = 0;      //ZONE B.2.2.1
        *max = T[i];  //ZONE B.2.2.2
        occ = count_occ(T[i],T[j], max, occ); //ZONE B.2.2.3
{T = T_0 ∧ N = N_0 ∧ 0 ⩽ i < j ⩽ N-1 ∧ T[i] > *max ∧ max = T[i] ∧
occ = count_occ(T[i],T[j],max,occ)}
      } //ZONE B.2
    } //ZONE B.3
    else if(T[i] < T[j]){
      if(T[j] > *max){
        occ = 0; //ZONE B.3.1
```

```
70          *max = T[j]; //ZONE B.3.2
71 {T = T0 ∧ N = N0 ∧ 0⩽i<j⩽N-1 ∧ T[i] < T[j] ∧ T[j] > *max ∧ *max = T[j] ∧occ=0}
72        }
73        occ = count_occ(T[i],T[j], max, occ); //ZONE B.3.3
74 {T = T0 ∧ N = N0 ∧ 0 ⩽ i < j ⩽ N-1 ∧ T[j] > *max
75 ∧ occ = count_occ(T[i],T[j],max,occ)} //ZONE B.3
76      }
77      i++; //ZONE B.3.4
78      j--; //ZONE B.3.5 //ZONE B
79 {T = T0 ∧ N = N0 ∧ 0 ⩽ i < j ⩽ N-1
80 ∧ occ = count_occ(T[i],T[j],max,occ) ∧ i = 1+i ∧ j = j-1}
81    }//fin while
82 //ZONE C
83    if(i == j){
84      if(T[i] > *max){ //ZONE C.1
85        occ = 0; //ZONE C.1.1
86 {T = T0 ∧ N = N0 ∧ 0⩽i<j⩽N-1 ∧ i = j ∧ T[i] > *max ∧ occ = 0}
87        *max = T[i]; //ZONE C.1.1
88 {T = T0 ∧ N = N0 ∧ 0⩽i<j⩽N-1 ∧ i = j ∧ T[i] > *max ∧ *max = T[i] ∧ occ=0}
89        occ = 1;//ZONE C.1.1
90 {T = T0 ∧ N = N0 ∧ 0⩽i<j⩽N-1 ∧ i = j ∧ T[i] > *max ∧ *max = T[i] ∧ occ=1}
91      }//ZONE C.1 //ZONE C.2
92      else if(T[i] == *max){
93        occ++;
94 {T = T0 ∧ N = N0 ∧ 0⩽i<j⩽N-1 ∧ i = j ∧ T[i] = *max ∧ *max = T[i] ∧ occ=occ+1}
95      } //ZONE C.2
96    } //ZONE C
97    return occ;
98 {Post: T = T0 ∧ N = N0 ∧ 0 ⩽ j ⩽ i ⩽ N-1 ∧ occ = multiplicite(T, N, max) ∧
99 max = Max(MaximumSSTab(T, 0, i), MaximumSSTab(T, j, N-1))} ⇒ {PostCondition}
100 }
```

6 Complexité :

Pour avoir plus de facilité à trouver la complixité de mon code je définis des zones en indiquant à quel endroit du code coresspond et se situe à partir de quelle ligne jusqu'au quelle ligne (Début-Fin).

6.1 Complexité de la fonction count_occ() :

Pour obtenir la complexité de la fonction count_occ() je divise en 2 parties :

6.1.1 ZONE A - (17-18) :

Par application de la régle 1 la complexité de la zone A :

$$T = 1$$

Par quoi borner T ?

O(1)

6.1.2 ZONE B - (20-27) :

Par application de la régle 3 la complexité de la zone B :

T(B) = Max(B.1, B.2, B.3)
T = 1

Par quoi borner T ?

O(1)

Par application de la régle 2 du code :
T = T(A) + T(B)
T = 1 + 1
T = 2
Par quoi borner T ?

O(1)

La compléxité de la fonction count_occ() est **Constante**.

6.2 Complexité des différentes zones de la fonction multiplicite() :

Pour obtenir la complexité de la fonction multiplicite() j'ai dévisé le code en 3 parties, zones A,B,C.

6.2.1 ZONE A - (34-39) :

Par application de la régle 1 la complexité de la zone A veut T(1) donc T est borné par O(1).

6.2.2 ZONE B.1 - (42-51) :

Par application de la régle 1 et 2 :

T(B.1) = T(B.1.1) + T(B.1.2) + T(B.1.3)
3 = 1 + 1 + 1

Par quoi borner T ?

O(1)

6.2.3 ZONE B.2 - (53-65) :

Par application de la rélgle 1 la compléxité de la zone B.2.1 veut O(1).
Par application de la régle 1 et 2 la compléxité de la zone B.2.2 veut :

T(B.2.2) = T(B.2.2.1) + T(B.2.2.2) + T(B.2.2.3)
3 = 1 + 1 + 1

Par quoi borner T ?

O(1)

Donc la compléxité de la zone B.2.1 et B.2.2 est O(1), alors la complexité de la zone veut :

T(B.2) = T(B.2.1) + T(B.2.2)
3 = 1 + 1 + 1

Par quoi borner T ?

O(1)

6.2.4 ZONE B.3 - (66-78) :

Par application de la régle 1 et 2 la compléxité de la zone B.3 veut :

T(B.3) = T(B.3.1) + T(B.3.2) + T(B.3.3) + T(B.3.4) + T(B.3.5)
5 = 1 + 1 + 1 + 1 + 1

Par quoi borner T ?

O(1)

6.2.5 ZONE B - (42-78) :

Pour obtenir la complexité totale de la zone B, je procède de façon suivant :

$$T(B) = \sum_{i=0}^{\frac{n}{2}}((B.1) + (B.2) + (B.3)) \tag{1}$$

$$= \sum_{i=0}^{\frac{n}{2}}(1 + 1 + 1) \tag{2}$$

$$= 3 \cdot (\frac{n}{2}) \tag{3}$$

Par quoi borner T ?

$$= O(\frac{n}{2}) \tag{4}$$

6.2.6 ZONE C - (82-96) :

En utilisant les régles 1 et 2 la complixité de la zone C.1 veut :

T(C.1) = T(C.1.1) + T(C.1.2) + T(C.1.3)
3 = 1 + 1 + 1

Par quoi borner T ?

O(1)

En applicant la régle 3 pour la zone C la complexité veut :

T(C) = Max(T(C.1) , T(C.2))   T(C) = 1

Par quoi borner T ?

O(1)

6.3 Complexité de la fonction multiplicite() :

Pour obtenir la complexité totale de la fonction multiplicite on combine la compléxité de chaque zones :

Comme la complexité de la fonction dépend de nombre itération alors on exprime T en fonction N/2

$$T(\frac{n}{2}) = T(A) + T(B) + T(C) \tag{5}$$

$$T(\frac{n}{2}) = 1 + (\frac{n}{2}) + 1 \tag{6}$$

$$T(\frac{n}{2}) = (\frac{n}{2}) + 2 \tag{7}$$

Par quoi borner T ?

$$O(\frac{n}{2}) \tag{8}$$

On peut prouver que la complexité de la fonction complexite est O(N/2) qui est meilleure de la complexité linéaire qui est O(n).