

INFO0947: Construction de Programme

Groupe 34: Timothy SMEERS, Soline LÈBRE

29 mars 2021

Table des matières

| | | |
|-----|---|---|
| 1 | Sous-problèmes | 3 |
| 2 | Notations | 3 |
| 3 | Précondition & postcondition | 3 |
| 4 | Invariant graphique | 4 |
| 4.1 | Règle de complexité | 4 |
| 4.2 | Calcul de la complexité du code | 5 |
| 5 | Invariant formel | 7 |
| 6 | Code | 7 |
| 6.1 | main.c | 7 |
| 6.2 | multiplicite.h | 8 |
| 6.3 | multiplicite.c | 9 |
| 7 | Documentation | 9 |

1 Sous-problèmes

SP_1 : Balayer le tableau

- Input : Un tableau d'entier de taille N
- Output : \
- Objets utilisé(s) :
 - ★ unsigned int i ;
 - ★ unsigned int j ;
 - ★ int $T[N]$;

SP_2 : Trouver le plus grand nombre présent dans le tableau et incrémenté une variable de 1 pour chaque occurrence trouvée

- Input : Un tableau d'entier de taille N
- Output : Le maximum et le nombre d'occurrence(s)
- Objets utilisé(s) :
 - ★ int max ;
 - ★ unsigned int $occurrence$;
 - ★ unsigned int $T[N]$;

Lien entre les SP_s : $SP_1 \subset SP_2$

2 Notations

$$\#i \bullet (i \in [0, N - 1]) \mid (max_i T[N])$$

3 Précondition & postcondition

Précondition : $T[N] \neq NULL \ \&\& \ N > 0 \rightarrow assert(T \neq NULL \ \&\& \ N > 0)$;

Postcondition :

- $T = T_0 \ \&\& \ N = N_0 \ \&\& \ max \neq 0 \ \&\& \ occurrence > occurrence_0$
- max contient la valeur du plus grand nombre présent dans $T[N]$
- $occurrence$ contient le nombre d'occurrence de max présent dans $T[N]$

4 Invariant graphique

| | | | | |
|-----|---|-----------------|---|---|
| T : | 0 | i | j+1 | N |
| | Déjà balayer | Reste à balayer | Déjà balayer | |
| | - max contient la valeur maximum | | - max contient la valeur de maximum | |
| | - occurrence contient le nombre d'occurrence de max | | - occurrence contient le nombre d'occurrence de max | |

FIGURE 1 – Invariant graphique

1. $G.B = i \leq j$
2. $C.A = i > j$
3. $F_t = j - i$

4.1 Règle de complexité

R_1 : Instruction de base

$$T(.) = 1$$

R_2 : Séquence d'instruction

$$T(.) = T_1(.) + T_2(.) + \dots + T_k(.)$$

R_3 : if

$$T(.) = \max(T_1(.), T_2(.))$$

R_4 : Switch

$$T(.) = \max(T_1(.), T_2(.), \dots, T_k(.), T_{k+1}(.))$$

R_5 : Boucle

$$T(.) = T_1(.) + T_2(.) + \dots + T_k(.)$$

$$= \sum_{i=1}^k \underbrace{T_i(.)}_{\text{gardien de boucle}}$$

R_6 : Programme

4.2 Calcul de la complexité du code

```

int multiplicite ( int *T , const int N , int *max ) {
assert(T != NULL && N > 0);

unsigned int i = 0;
unsigned int j = N-1;
unsigned int occurence = 0;
*max = T[0];
while (i <= j) {

    if (T[i] > *max || T[j] > *max){
        if( T[i] == T[j]){
            *max = T[i];
            occurence = 2;
        }
        }else if(T[i] > T[j]){
            *max = T[i];
            occurence = 1;
        }
        }else{
            *max = T[j];
            occurence = 1;
        }
    }

    }else if(*max == T[i] || *max == T[j]){
        if(i == j)
            occurence++;
        else if(T[i] == T[j])
            occurence +=2;
        else
            occurence++;
    }

    i++;
    j--;
}
return(occurence);
}

```

Diagram illustrating the complexity analysis of the code using curly braces:

- The first group of lines (initialization and while loop condition) is grouped under $T(A)$.
- The nested if-else structure is grouped under $T(B')$.
- The innermost if-else blocks are grouped under $T(B'')$.
- The final increment/decrement and return statement are grouped under $T(B_c')$.
- The entire while loop body is grouped under $T(B)$.

- $T(A) = 1$ Par application de la règle 1
- $T(B''_{a1}) = 2$ Par application de la règle 1, 2, 3
- $T(B''_{a2}) = 2$ Par application de la règle 1, 2, 3
- $T(B''_{a3}) = 2$ Par application de la règle 1, 2, 3
- $T(B'_a) = \max(T(B''_{a1}), T(B''_{a2}), T(B''_{a3}))$ Par application de la règle 2 et 3
 $T(B'_a) = 2$
- $T(B''_{b1}) = 1$ Par application de la règle 1 et 2 et 3
- $T(B''_{b2}) = 1$ Par application de la règle 1 et 2 et 3
- $T(B''_{b3}) = 1$ Par application de la règle 1 et 2 et 3

- $T(B'_b) = \max(T(B''_{b1}), T(B''_{b2}), T(B''_{b3}))$ Par application de la règle 2 et 3
 $T(B'_b) = 1$
- $T(B'_c) = 2$ Par application de la règle 1
- $T(B') = \max(T(B'_a), T(B'_b))$ Par application de la règle 2 et 3
 $T(B') = 2$
- $T(B) = \sum_{i=1}^k T_i(.)$ Par application de la règle 5
 $T(B) = n/2$
- $T(n) = T(A) + T(B)$ Par application de la règle 2 et 1 $T(n) = (n/2) + 2$

5 Invariant formel

$$0 \leq i \leq N - 1 \wedge \max = \max_i T[N] \wedge \text{occurrence} = \#i \bullet (i \mid \max)$$

6 Code

6.1 main.c

```
1 #include <stdio.h>
2 #include "multiplicite.h"
3
4 int main(){
5
6     int T [9] = {13 , 16 , 19 , 17 ,19 , 12 , 2, 4, 10};
7     int max ;
8
9     int occurrences = multiplicite(T, 9, &max);
10    printf("%d - %d\n", occurrences, max);
11 }
```

Codingstyle 1 – main.c

6.2 multiplicite.h

```
1 #ifndef MULTIPLICITE_H_
2 #define MULTIPLICITE_H_
3
4 /**
5  * @fn int multiplicite(int*, const int, int*)
6  * @brief Obtain, for T, the greatest value of T as well as the number
7  * of occurrences of this value in T.
8  *
9  * @pre T != NULL && N > 0
10 * @post T = T0 && N = N0 && max != 0 && occurrence > occurrence0
11 *      max contains the value of the largest number present in T [N]
12 *      occurrence contains the number of occurrences of max present in T [N]
13 * @param T an array with N integer values
14 * @param N table size T
15 * @param max contains the value of the largest number present in T [N]
16 * @return The number of occurrences of the maximum in T.
17 */
18 int multiplicite ( int *T , const int N , int *max );
19
20 #endif MULTIPLICITE_H_
```

Codingstyle 2 – multiplicite.h

6.3 multiplicite.c

```
1 #include <stdio.h>
2 #include <assert.h>
3 #include "multiplicite.h"
4
5 int multiplicite ( int *T , const int N , int *max ) {
6     assert(T != NULL && N > 0);
7
8     unsigned int i = 0;
9     unsigned int j = N-1;
10    unsigned int occurrence = 0;
11    *max = T[0];
12
13    while (i <= j) {
14
15        if (T[i] > *max || T[j] > *max){
16            if( T[i] == T[j]){
17                *max = T[i];
18                occurrence = 2;
19
20            }else if(T[i] > T[j]){
21                *max = T[i];
22                occurrence = 1;
23            }else{
24                *max = T[j];
25                occurrence = 1;
26            }
27
28            }else if(*max == T[i] || *max == T[j]){
29                if(i == j)
30                    occurrence++;
31                else if(T[i] == T[j])
32                    occurrence +=2;
33                else
34                    occurrence++;
35            }
36            i++;
37            j--;
38        }
39        return(occurrence);
40    }
```

Codingstyle 3 – multiplicite.c

7 Documentation

Pour plus d'information sur le code vous pouvez consulter le [site internet](#) contenant la documentation doxygen.