INFO0947: Complement de Programmation Projet 1

Groupe 26: Franck Duval HEUBA BATOMEN, Bilali Alassani

Contents

1 Introduction

Nous avons un tableau **T[N]** de **N** éléments **entiers** dans lequel nous souhaitons déterminer le minimun et le maximun. Ainsi calculer la somme des éléments entre le minimun et le maximun.

La solution évidente consiste à determiner le minimun et la maximun dans un premier temps puis de calculer la somme entre ces derniers.

Cependant la solution que nous voulons obtenir doit avoir une complexité de $\mathrm{O}N$ dans la pire des cas.

Le travail que nous allons mener, consiste à formaliser le, problème, produire un invariant graphique puis un invariant formel, et enfin produire le code, montrer qu'il fonctionne et prouver que sa complexité est en $\mathrm{O}N$.

2 Formalisation du Problème

Nom	Op
ET	$\wedge$
OU	$\vee$
Minimun d'un tableau	min
Maximun d'un tableau	max
Superieure	$>$
Inferieure	$<$
Superieure ou égale	$\geq$
Inferieure ou égale	$\leq$
Somme d'éléments de 0 à k	$\sum_0^k$
Quantification universelle	$\forall$
Quantification existentielle	$\exists$

Table 1: Opérateurs les plus usuels en logique

3 Définition et Analyse du Problème

3.1 Input:

Le nombre d'éléments à analyser:

```
int N;
```

Un tableau d'entier de:

```
int T[N];
```

3.2 Output:

cette fonction retourne la somme des éléments du tableau entre la position du minimum *min_pos* et la position du maximun *max_pos*.

Ainsi soient: $a = \min(min\_pos, max\_pos) \wedge b = \max(min\_pos, max\_pos)$

$somme = \sum_{i=a}^{b}(T[i]) \wedge 0 \leq a \leq b < N$

3.3 Objets Utilisés:

min_pos: Contient l'indice de $\min T$

```c
int min_pos;
```

max_pos: Contient l'indice de $\max T$

```c
int max_pos;
```

resultat: Contient la somme des élément du tableau $\max T$

```c
int resultat;
```

3.4 Sous-Problème:

Vu que le l'objectif de la fonction est d'écrire un programme de complexité $O(N)$, on aura un seul sous-problème qui va consister en:

- Determiner la position du maximun
- Determiner la position du minimun
- Calculer la somme des éléments du tableau entre le maximun et le minimum

4 Specifications

/**
 * @Pre: $N = N_0 > 0 \wedge T = T_0$
 * @post: $T = T_0 \wedge 0 \leq min\_pos \leq max\_pos \leq N-1 \wedge T[min\_pos] = \min T \wedge T[max\_pos] = \max T$
*/

```c
    int somme(int *T, int N,  int *min_pos, int *max_pos);
```

5 Invariants

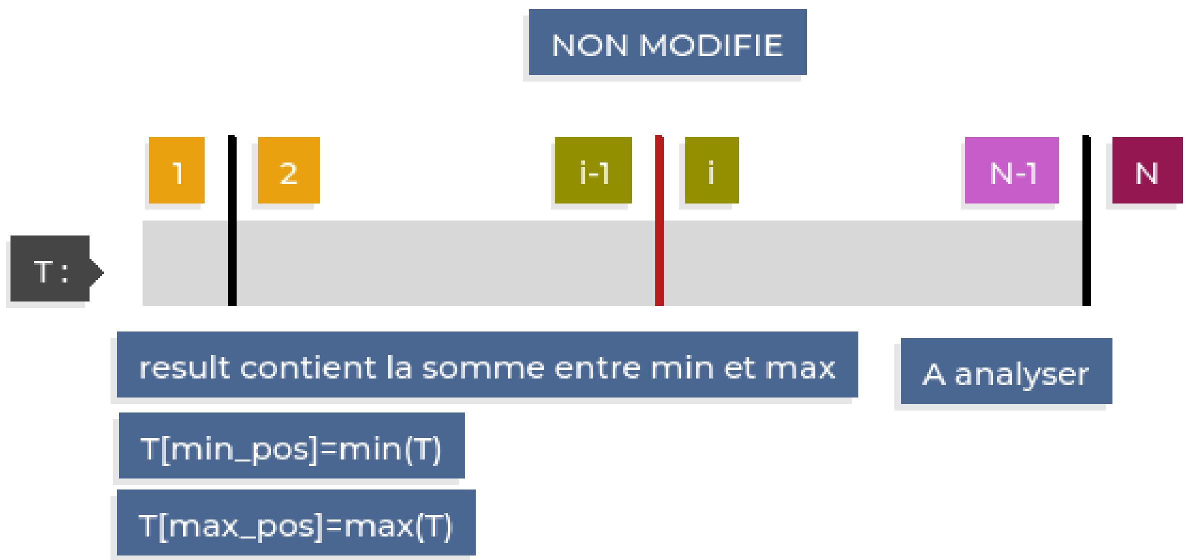

Figure 1: Invariant graphique

Invariant formel

$$INV \equiv N > 0 \wedge T = T_0 \wedge\ 0 \leq min\_pos \leq max\_pos \leq i \leq N-1 \wedge T[min\_pos] = \min T \\ \wedge T[max\_pos] = \max T$$

6 Approche Constructive

```
//$N = N_0 > 0 \wedge T = T_0$
int somme(int *T, int N,  int *min_pos, int *max_pos)
{
    //$N = N_0 > 0 \wedge T = T_0$
    assert(N>0 && T != NULL && min_pos != NULL && max_pos != NULL);
    //$N = N_0 > 0 \wedge T = T_0$

    int resultat, tmp1, tmp2, i;
    //$N = N_0 > 0 \wedge T = T_0$

    if (N==1)
        //$N = N_0 > 0 \wedge T = T_0 \wedge T[min\_pos] = \min T \wedge T[max\_pos] = \max T$
        return T[*min_pos = *max_pos = 0];

    //$N = N_0 > 0 \wedge T = T_0$
    if (T[0] > T[1])
        *min_pos = 1 + (*max_pos = 0);
        //$N = N_0 > 0 \wedge T = T_0 \wedge T[min\_pos] = \min T \wedge T[max\_pos] = \max T$
    else
        *max_pos = 1 + (*min_pos = 0);
        //$N = N_0 > 0 \wedge T = T_0 \wedge T[min\_pos] = \min T \wedge T[max\_pos] = \max T$

    //$\rightarrow N = N_0 > 0 \wedge T = T_0 \wedge T[min\_pos] = \min T \wedge T[max\_pos] = \max T$

    resultat = T[0] + T[1];
    //$N = N_0 > 0 \wedge T = T_0 \wedge T[min\_pos] = \min T \wedge T[max\_pos] = \max T \wedge resultat = \sum_{k=min\_pos}^{max\_pos}(T[k])$
    tmp1 = T[0];
    //$tmp1 = \sum^{\min(min\_pos,max\_pos)}(T[.])$
    //$N = N_0 > 0 \wedge T = T_0 \wedge T[min\_pos] = \min T \wedge T[max\_pos] = \max T \wedge resultat = \sum_{k=min\_pos}^{max\_pos}(T[k])$
    tmp2 = 0;
    //$tmp2 = \sum_{\max(min\_pos,max\_pos)}^{N-1}(T[.])$
    //$N = N_0 > 0 \wedge T = T_0 \wedge T[min\_pos] = \min T \wedge T[max\_pos] = \max T \wedge resultat = \sum_{k=min\_pos}^{max\_pos}(T[k])$
    i = 2;
    //$INV \equiv N = N_0 > 0 \wedge T = T_0 \wedge T[min\_pos] = \min T \wedge T[max\_pos] = \max T$
    //$\wedge resultat = \sum_{k=min\_pos}^{max\_pos}(T[k]) \wedge 1 \leq i \leq N$

    while(i<N)
    {
        //$INV \equiv N = N_0 > 0 \wedge T = T_0 \wedge T[min\_pos] = \min T \wedge T[max\_pos] = \max T$
        //$\wedge resultat = \sum_{k=min\_pos}^{max\_pos}(T[k]) \wedge 1 \leq i \leq N-1$
        if(T[i]<T[*min_pos])
        {
            if (*min_pos<*max_pos)
```

```
                resultat -= tmp1;
                //$resultat = \sum_{k=min\_pos}^{max\_pos}(T[k]$

            //$T[min\_pos] = \min T$
            *min_pos = i;
            //$T[min\_pos] = \min T$
            tmp1 = resultat;
            //$tmp1 = \sum^{\min(min\_pos,max\_pos)}(T[.])$
            //$tmp2 = \sum_{\max(min\_pos,max\_pos)}^{i-1}(T[.])$
            tmp2 = 0;
            //$tmp2 = \sum_{\max(min\_pos,max\_pos)}^{i}(T[.])$
        }
        else if(T[i]>T[*max_pos])
        {
            if (*max_pos<*min_pos)
                resultat -= tmp1;
                //$resultat = \sum_{k=min\_pos}^{max\_pos}(T[k]$

            //$T[max\_pos] = \max T$
            *max_pos = i;
            //$T[max\_pos] = \max T$
            tmp1 = resultat;
            //$tmp1 = \sum^{\min(min\_pos,max\_pos)}(T[.])$
            //$tmp2 = \sum_{\max(min\_pos,max\_pos)}^{i-1}(T[.])$
            tmp2 = 0;
            //$tmp2 = \sum_{\max(min\_pos,max\_pos)}^{i}(T[.])$
        }
        else
        {
            //$tmp2 = \sum_{\max(min\_pos,max\_pos)}^{i-1}(T[.])$
            tmp2 += T[i];
            //$tmp2 = \sum_{\max(min\_pos,max\_pos)}^{i}(T[.])$
        }

        resultat += T[i];
        //$resultat = \sum_{k=min\_pos}^{max\_pos}(T[k]$
        //$1 \leq i \leq N-1$
        i++;
        //$1 \leq i \leq N$
    }

    //$INV \equiv N = N_0 > 0 \wedge T = T_0 \wedge T[min\_pos] = \min T \wedge T[max\_pos] = \max T$
    //$\wedge resultat = \sum_{k=max\_pos}^{N-1}(T[k]) \wedge 1 \leq i \leq N$

    return resultat - tmp2;
    //$T = T_0 \wedge 0 \leq min\_pos \leq max\_pos \leq N-1 \wedge T[min\_pos] = \min T \wedge T[max\_pos] = \max T$
}
//$T = T_0 \wedge 0 \leq min\_pos \leq max\_pos \leq N-1 \wedge T[min\_pos] = \min T \wedge T[max\_pos] = \max T$
```

Extrait de Code 1: somme.c

7 Code Complet

```
/**
 * \file somme.h
 * \brief Header pour la somme min-max d'un tableau
 * \author HEUBA BATOMEN Franck Duval, Bilali Assalni
```

```c
 * \version 0.1
 * \date 03/04/2023
 *
*/

#ifndef __SOMME__
#define __SOMME__

/*
 * @pre: N=N0>0 && T = T0
 * @post: T=T0 && N = N0 && T[min_pos]=min(T) && T[max_pos]=max(T)
 * && somme = min(min_pos, max_pos) + T[min(min_pos, max_pos) + 1]
 * + ... + max(min_pos, max_pos)
*/
int somme(int *T, int N,  int *min_pos, int *max_pos);

#endif
```

Extrait de Code 2: somme.h

```c
int somme(int *T, int N,  int *min_pos, int *max_pos)
{
    assert(N>0 && T != NULL && min_pos != NULL && max_pos != NULL);

    int resultat, tmp1, tmp2, i;

    if (N==1)
        return T[*min_pos = *max_pos = 0];

    if (T[0] > T[1])
        *min_pos = 1 + (*max_pos = 0);
    else
        *max_pos = 1 + (*min_pos = 0);

    resultat = T[0] + T[1];
    tmp1 = T[0];
    tmp2 = 0;
    i = 2;

    while(i<N)
    {
        if(T[i]<T[*min_pos])
        {
            if (*min_pos<*max_pos)
                resultat -= tmp1;

            *min_pos = i;
            tmp1 = resultat;
            tmp2 = 0;
        }
        else if(T[i]>T[*max_pos])
        {
            if (*max_pos<*min_pos)
                resultat -= tmp1;

            *max_pos = i;
            tmp1 = resultat;
            tmp2 = 0;
        }
        else
        {
```

```
42            tmp2 += T[i];
43        }
44
45        resultat += T[i];
46        i++;
47    }
48
49    return resultat - tmp2;
50 }
```

Extrait de Code 3: somme.c

8 Complexité

determinons la complexité de notre code:
On peut diviser notre code en 05 parties:

- De la ligne 1–2: P_a

```
assert(N>0 && T != NULL && min_pos != NULL && max_pos != NULL);
int resultat, tmp1, tmp2, i;
```

- De la ligne 4–9: P_b

```
if (N==1)
    return T[*min_pos = *max_pos = 0];
elif (T[0] > T[1])
    *min_pos = 1 + (*max_pos = 0);
else
    *max_pos = 1 + (*min_pos = 0);
```

- De la ligne 11–14: P_c

```
resultat = T[0] + T[1];
tmp1 = T[0];
tmp2 = 0;
i = 2;
```

- De la ligne 16–43: P_d

```
while(i<N)
{
    if(T[i]<T[*min_pos])
    {
        if (*min_pos<*max_pos)
            resultat -= tmp1;

        *min_pos = i;
        tmp1 = resultat;
        tmp2 = 0;
    }
    else if(T[i]>T[*max_pos])
    {
        if (*max_pos<*min_pos)
            resultat -= tmp1;

        *max_pos = i;
```

```
        tmp1 = resultat;
        tmp2 = 0;
    }
    else
    {
        tmp2 += T[i];
    }

    resultat += T[i];
    i++;
}
```

- De la ligne 45: P_e

```
return resultat - tmp2;
```

8.1 P_a.

$T(.) = 1$

8.2 P_b.

Ici la complexité est égale au maximun des complexités de chaque branche c'est à dire: $T(.) = 1$

8.3 P_c.

Ici toutes les instructions sont élémenetaires, ainsi: $T(.) = 1$

8.4 P_d.

Ici le code peut être diviser en quatre partie:

- De la ligne 19 à 25: P_d1

```
if(T[i]<T[*min_pos])
{
    if (*min_pos<*max_pos)
        resultat -= tmp1;

    *min_pos = i;
    tmp1 = resultat;
    tmp2 = 0;
}
```

- De la ligne 29 à 34: P_d2

```
else if(T[i]>T[*max_pos])
{
    if (*max_pos<*min_pos)
        resultat -= tmp1;

    *max_pos = i;
    tmp1 = resultat;
    tmp2 = 0;
}
```

- De la ligne 38 à 38: P_d3

```
else
{
    tmp2 += T[i];
}
```

- De la ligne 41 à 42: P_d4

```
resultat += T[i];
i++;
```

8.4.1 P_d1

Ici la complexité est $T(.) = 1$

8.4.2 P_d2

Ici la complexité est $T(.) = 1$

8.4.3 P_d3

Ici la complexité est $T(.) = 1$

8.4.4 P_d4

Ici la complexité est $T(.) = 1$
Ainsi la complexité de notre programme est $T_d(.) = \sum_{i=1}^{N}(\max(T_d1(.), T_d2(.), T_d3(.)) + T_d4(.))$.
Ainsi $T_d(.) = 2N$

8.5 P_e.

Enfin $T(.) = 1$

$T(N) = T_a(.) + T_b(.) + T_c(.) + T_d(.) = 3 + 2N$

$T \in O(N) \leftrightarrow \exists n_0 \in N, c \in R^+ : \forall n > n_0 : T(n) \leq c * n$
Ainsi pour $c \geq 4, T \in O(N)$.

La complexite de notre code est donc de: $O(N)$

9 Conclusion

Arrivé à l'épilogue de notre travail, il a été question de mener à bien un projet en respectant, la méthodologie de développement.

Ainsi dans un premier temps, nous avons formalisez le problème et donnez une définition du problème. Ensuite nous avons données les spécifications à partie desquelles nous avons construit l'Invariant graphique et ainsi nous avons déterminé l'invariant formel.

Ce dernier nous a permis d'écrire notre code et Ensuite de montrer qu'il est correcte; et enfin de determiner la complexité de notre code.