INFO0947: Projet 1 Rapport

Groupe 10: Cyril Russe, Martin Randaxhe

Table des matières

1 Définition du problème

1.1 Introduction

Soit T, un tableau à N valeurs entière ($N \geq 0$). On veut construire une fonction qui permet d'obtenir, pour T, le plus grand entier $k(k \in [0, \ldots, N-1])$ tel que le sous-tableau $T[0 \ldots k-1]$ est à la fois préfixe et suffixe de T. Si un tel sous-tableau n'existe pas, la fonction doit renvoyer la valeur 0. Attention, on fait l'hypothèse que $k \neq N$ sinon le problème devient trivial.

1.2 Input/Output

1.2.1 Input

— T : un tableau d'entier de taille N
— N : la taille du tableau T

1.2.2 Output

— $taille\_sous\_tableau$: la taille du plus grand sous tableau à la fois préfixe et suffixe de T

1.3 Objets utilisés

— int $*T$: un tableau d'entier de taille N
— unsigned int N : la taille du tableau d'entier
— int $taille\_sous\_tableau$: la taille du plus grand sous tableau à la fois préfixe et suffixe de T
— int i, j : des compteurs de boucle

2 Formalisation du problème

2.1 Description de notation

— $MemeSSTableau(T, N, i) \equiv \exists j, 0 \leq j \leq i, T[j] \neq T[N-1-i+j] \Rightarrow 0$, sinon 1
— $PrefixeSuffixe(T, N) \equiv i+1$ si $\forall i, 0 \leq i < N-1, MemeSSTableau(T, N, i) = 1$

2.2 Spécifications

```c
/**
*
* int prefixe_suffixe(int *T, const unsigned int N)
*
* Fonction qui renvoit la taille du plus grand sous tableau
* étant préfixe et suffixe
*
* @param  T un tableau d'entiers initialisé au préalable
* @param  N un entier définissant la taille de T
*
* @pre : T=T_0 && T!=NULL && N>1 && i=0 && taille_sous_tableau=0
* @post : T=T_0, N=N_0, taille_sous_tableau = PrefixeSuffixe(T, N)
*
* @return : taille_sous_tableau
*/
int prefixe_suffixe(int *T, const unsigned int N);

```

Extrait de Code 1 – Spécification fonction prefixe_suffixe

2.3 Décomposition en sous-problèmes

- **Sous-problème 1** : Teste les préfixes/suffixes de T en allant de N-2 à 0
- **Sous-problème 2** : Teste si chaque élément préfixe est égal à l'élément correspondant suffixe

$$SP2 \subset SP1$$

2.4 Spécification des sous-problèmes

- **SP1** :
  - @pre :

$$Tinit \wedge T \neq NULL \wedge N > 1 \wedge i = 0 \wedge taille\_sous\_tableau = 0$$

  - @post :

$$T = T_0 \wedge N = N_0 \wedge taille\_sous\_tableau = PrefixeSuffixe(T, N)$$

- **SP2** :
  - @pre :

$$T = T_0 \wedge N = N_0 \wedge j = 0 \wedge 0 < i \leq N - 2$$

  - @post :

$$T = T_0 \wedge N = N_0 \wedge est\_pref\_et\_suf = MemeSSTableau(T, N, i)$$

2.5 Invariant

2.5.1 Invariant SP1

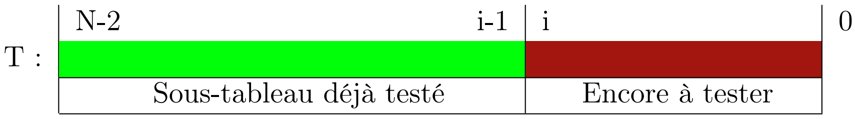

FIGURE 1 – Invariant SP1

$$Inv : N = N_0 \wedge T = T_0 \wedge 0 \leq i \leq N - 2$$

2.5.2 Invariant SP2

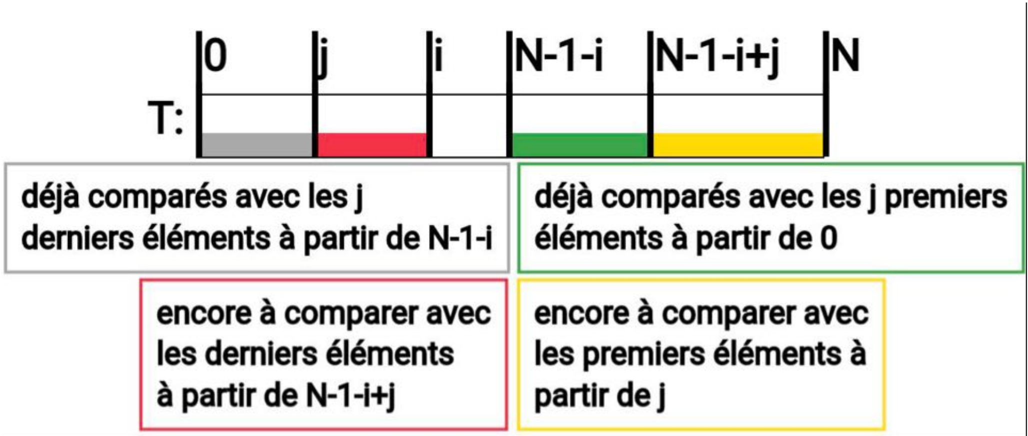

FIGURE 2 – Invariant SP2

$$Inv : N = N_0 \wedge T = T_0 \wedge 0 \leq j \leq i$$

2.6 Critères d'arrêt

— SP1 : $i = 0 \vee taille\_sous\_tableau \neq 0$
— SP2 : $j = i$

2.7 Gardiens de boucle

— SP1 : $i > 0 \wedge taille\_sous\_tableau = 0$
— SP2 : $j < i$

2.8 Fonctions de terminaison

— SP1 : i
— SP2 : $i - j$

3 Code

3.1 Code complet de la fonction "prefixe_suffixe"

```
 int prefixe_suffixe(int *T, const unsigned int N)
{
    assert(T!=0 && N>1);
    //Initialisation des variables
    unsigned int i=N-2, j;
    int taille_sous_tableau=0;

    //Début SP1
    while(taille_sous_tableau==0 && i!=0){
        //Début SP2
        j=0;
        while(j<i){
            if(T[j]!=T[N-i+j])
                j=i+1;
            j++;
        }
        //Fin SP2
        if(j==i)
            taille_sous_tableau=i;
        i--;
    }
    //Fin SP1


    return taille_sous_tableau;
}//Fin prefixe_suffixe
```

Extrait de Code 2 – Code complet de la fonction "prefixe_suffixe"

3.1.1 Initialisation des variables

Cet extrait de code présente le début de la fonction qui est constitué de l'initialisation de nos variables, afin d'arriver au stade où celles-ci vérifient les pré-conditions du SP 1.

```
  {Pré: T_init ∧ N > 1}
  assert(T!=NULL && N>1);
  unsigned int i=0, j;
  {T = T_0 ∧ N = N_0 ∧ i = 0 ∧ j_init}
  int taille_sous_tableau=0, est_pref_et_suf=1;
  {T = T_0 ∧ N = N_0 ∧ i = 0 ∧ j_init ∧ taille_sous_tableau = 0 ∧ est_pref_et_suf = 1}
```

Extrait de Code 3 – Initialisation des variables

3.1.2 Sous-problème 1

```
{Inv : N = N_0 ∧ T = T_0 ∧ 0 ≤ i ≤ N − 2}
while(taille_sous_tableau==0 && i!=0){
    {Inv ∧ B : T = T_0 ∧ N = N_0 ∧ 0 < i ≤ N − 2}

    //SP2

    if(j==i)
        taille_sous_tableau=i;
    i--;
    {N = N_0 ∧ T = T_0 ∧ 0 ≤ i ≤ N − 2}
}//fin while
{Post SP1 : T = T_0 ∧ N = N_0 ∧ est_pref_et_suf = MemeSSTableau(T, N, i)}
```

Extrait de Code 4 – Sous-problème 1

3.1.3 Sous-problème 2

```

j=0;
{Pré SP2: T = T_0 ∧ N = N_0 ∧ j = 0 ∧ 0 < i ≤ N − 2}

{Inv : N = N_0 ∧ T = T_0 ∧ 0 ≤ j ≤ i}
while(j<i){
    {Inv ∧ B : T = T_0 ∧ N = N_0 ∧ 0 ≤ j < i}
    if(T[j]!=T[N-i+j])
        j=i+1;
    j++;
    {T = T_0 ∧ N = N_0 ∧ 0 ≤ j ≤ i}
}
{Post SP2:T = T_0 ∧ N = N_0}
```

Extrait de Code 5 – Sous-problème 2

4 Complexité

On va découper la complexité en plusieurs segments et ensuite on va les additionner. La première partie est l'initialisation des variables et vu qu'on en déclare 4

$$T_A(N) = 4$$

Ensuite, on a la première boucle et par la règle 5, on a

$$T_B(N) = \sum_{i=0}^{N-1} (4 + T_C(i))$$

Dans ce cas-ci, le "4" représente l'initialisation des variables et le $T_C(i)$ est la deuxième boucle. Pour la deuxième boucle, on a

$$T_C(i) = \sum_{j=0}^{i} 1$$

On a donc

$$T_C(i) = i$$

On revient à la première boucle

$$T_B(N) = \sum_{i=0}^{N-1} (4 + i) \Leftrightarrow T_B(N) = \frac{N^2 + 7N}{2}$$

Enfin, en additionnant le tout, on obtient

$$T(N) = \frac{N^2 + 7N + 8}{2}$$

Dans ce cas ci, notre complexité $T(N)$ est majorée par $O(N^2)$. On peut donc en conclure que la fonction est de complexité quadratique.