

INFO0947: Construction de programme

Groupe 27: Alexandru DOBRE, Sami OUAZOUZ

Table des matières

1	Introduction	3
1.1	Contexte	3
1.2	Fonctionnement de la fonction	3
1.3	Exemple d'utilisation	3
2	Formalisation du Problème	3
3	Définition et Analyse du Problème	3
3.1	Définition du Problème	3
3.1.1	Input	3
3.1.2	Output	4
3.1.3	Caractérisation des inputs	4
3.2	Analyse du Problème	4
4	Specifications	4
4.1	Sous-problème 1	4
4.2	Sous-problème 2	5
5	Invariants	6
5.1	Explications	6
5.2	SP1	6
5.2.1	Invariant Graphique	6
5.2.2	Invariant Formel	6
5.3	SP2	6
5.3.1	Invariant Graphique	6
5.3.2	Invariant Formel	7
6	Approche Constructive	7
7	Code Complet	7
8	Complexité	7
9	Conclusion	7

1 Introduction

1.1 Contexte

Nous voulons construire une fonction dans laquelle nous allons passer en argument un tableau de nombres entiers et un nombre entier positif qui représente la taille du tableau. Nous voudrions que cette fonction nous retourne la taille du nombre d'éléments étant à la fois préfixe et suffixe de ce tableau.

1.2 Fonctionnement de la fonction

Soit un tableau T , contenant N valeurs entières ($N \geq 0$). Nous voulons construire une fonction qui va retourner un entier k ($k \in [0, \dots, N - 1]$) tel que le sous tableau $T[0, k - 1]$ est un préfixe de T et $T[N - k, N - 1]$ est un suffixe de T , c'est-à-dire que leurs éléments soient identiques.

Extrait du prototype de la fonction :

```
1 int prefixe_suffixe(int *T, const unsigned int N);
```

Extrait de Code 1 – Fonction souhaitée

où T et N ne sont pas modifiés.

1.3 Exemple d'utilisation

Soit F un tableau de 10 entiers et lg contient la taille du préfixe-suffixe :

```
1 int F[10] = {1, 2, 3, 1, 2, 3, 1, 2, 3, 1};  
2 unsigned int lg = prefixe_suffixe(F, 10);  
3 printf("Le plus long préfixe-suffixe du tableau est de taille %u.\n", lg);
```

Extrait de Code 2 – Exemple d'utilisation

Le printf va nous afficher : Le plus long préfixe-suffixe du tableau est de taille 4.

2 Formalisation du Problème

Soit $\text{prefixe_suffixe}(*T, N)$, une notation telle que :

- T un tableau d'entiers.
- N est la taille du tableau ($N \geq 0$).

On a $\text{prefixe_suffixe}(*T, N) \equiv 0 \leq k \leq N - 1, T[0, k] == T[N - k, N - 1]$.

Soit $\text{max_prefixe_suffixe}(*T, i, N)$ une notation pour le plus long préfixe-suffixe de T .

$\text{max_prefixe_suffixe}(*T, i, N) \equiv 0 \leq i \leq k < N, \text{max}(T[0, i - 1] == T[N - k + i, N - 1])$.

3 Définition et Analyse du Problème

3.1 Définition du Problème

3.1.1 Input

- T : Un tableau d'entiers
- N : La taille du tableau

3.1.2 Output

- La taille k du plus long préfixe qui est aussi un suffixe du tableau

3.1.3 Caractérisation des inputs

- **T** est un tableau non nul d'entiers
 - `int *T`;
- **N** est un entier non signé tel que $N \geq 0$
 - `unsigned int N`;

3.2 Analyse du Problème

On va découper le problème en sous-problèmes (SPs) afin de mieux l'analyser. Un sous-problème est une partie du problème qui peut être résolu indépendamment et qui peut être combiné avec d'autres sous-problèmes pour résoudre le problème global. Ils peuvent être de plusieurs types :

- Lecture au clavier.
- Affichage à l'écran
- Réaliser une action (Vérifier une propriété, une condition, sommer,...)
- Énumérer des valeurs et effectuer une action

Dans notre cas, nous avons affaire à deux sous-problèmes :

- **SP1** : Énumération des tailles possibles de préfixe-suffixe.
- **SP2** : Comparer les valeurs de préfixe et de suffixe.

4 Specifications

4.1 Sous-problème 1

Nous voulons écrire une boucle dans laquelle k balaye toutes les valeurs possibles pour le préfixe-suffixe. Pour ceci définissons la fonction :

- **Input :**
 - N , la taille du tableau
- **Output :**
 - k , la taille du préfixe-suffixe
- **Caractérisation de l'input :**
 - N est un entier non signé tel que $N \geq 0$
 - `unsigned int N`

Il nous faut donc écrire une boucle while qui va balayer toutes les valeurs possibles de k de $N - 1$ à 0.

- **Déclaration du compteur :**
 - `unsigned int k = N - 1`
- **Nombres de tours dans la boucle :**
 - $N - 1$
- **Gardien de boucle :**
 - $k > 0$
- **Corps de Boucle :**
 - **Comparer les valeurs de préfixe et de suffixe :**
 - SP2
 - Décrémenter k

```

1 unsigned int k = N - 1;
2 while (k > 0) {
3     // SP2
4     k--;
5 }

```

Extrait de Code 3 – SP1

4.2 Sous-problème 2

Nous voulons écrire une boucle dans laquelle on compare les valeurs de préfixe et de suffixe. Pour ceci définissons la fonction :

- **Input :**
 - T , le tableau d'entiers
 - N , la taille du tableau
 - k , la taille du préfixe-suffixe
 - i , le compteur de la boucle
- **Output :**
 - k , la taille du préfixe-suffixe
- **Caractérisation des inputs :**
 - N est un entier non signé tel que $N \geq 0$
 - **unsigned int** N
 - k est un entier non signé tel que $0 \leq k < N$
 - **unsigned int** $k = N - 1$
 - i est un entier non signé tel que $0 \leq i < k$
 - **unsigned int** $i = 0$
 - T est un tableau d'entiers tel que $T[0, N - 1]$
 - **int** $T[N]$

Il nous faut donc écrire une boucle while qui va balayer toutes les valeurs possibles de i de 0 à $k-1$ et trouver des correspondances entre les valeurs de préfixe i et de suffixe $N - k + i$.

- **Déclaration du compteur :**
 - **unsigned int** $i = 0$
- **Nombres de tours dans la boucle :**
 - k
- **Gardien de boucle :**
- **Condition de sortie :**
 - $i < k \ \&\& \ T[i] == T[N - k + i]$
- **Corps de Boucle :**
 - **Vérifier si i et k sont égaux :**
 - if ($i == k$)
 - Incrémenter i

```

1 unsigned int i = 0;
2 while(i < k && T[i] == T[N - k + i]){
3     i++;
4 }
5 if (i == k){
6     return k;
7 }
8 else{

```

```

9      i = 0;
10 }

```

Extrait de Code 4 – SP2

5 Invariants

5.1 Explications

La raison d'être de cette section est de définir les invariants nécessaires à la construction du programme. Les invariants sont des propriétés qui doivent être vérifiées à chaque étape de l'exécution du programme. Ils permettent de garantir que le programme fonctionne correctement et produit les résultats attendus. Il y a une règle qui est que chaque boucle nécessite un invariant. Nous aurons donc 2 invariants graphiques et 2 invariants formels.

Nous allons donc définir les invariants graphiques. Le premier aura pour objectif de définir la boucle de décrementation de k et le seconde couvrira la boucle qui compare les valeurs du tableau entre $T[i]$ et $T[N - k + i]$.

5.2 SP1

5.2.1 Invariant Graphique

Pour le premier invariant, nous allons définir la boucle de décrementation de k . La valeur de k est initialisée à $N - 1$ et elle est décrementée jusqu'à ce qu'elle atteigne 0. Attention, elle atteint zéro uniquement si lors du SP2 nous n'avons trouvé aucune correspondance dans le tableau.

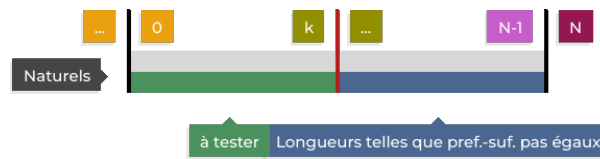


FIGURE 1 – Invariant graphique 1

Le critère d'arrêt de la boucle est $k == 0$. Donc, le gardien de boucle sera $k > 0$.

5.2.2 Invariant Formel

$$\begin{aligned}
 & N = N_0 \\
 & \wedge \\
 & 0 \leq k \leq N - 1 \\
 & \wedge \\
 & k = \text{max_prefixe_suffixe}(*T, i, N)
 \end{aligned}$$

5.3 SP2

5.3.1 Invariant Graphique

Pour le second invariant, nous allons définir la boucle de comparaison entre les valeurs du tableau entre $T[i]$ et $T[N - k + i]$. La valeur de i est initialisée à 0 et elle est incrémentée jusqu'à ce qu'elle atteigne k .

Le critère d'arrêt de la boucle est $i == k$. Donc, le gardien de boucle sera $i < k$.

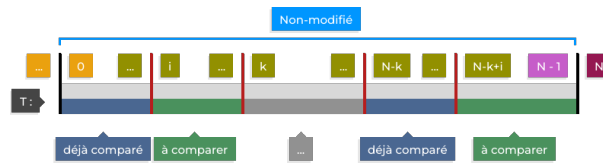


FIGURE 2 – Invariant graphique 2

5.3.2 Invariant Formel

$$\begin{aligned}
 N &= N_0 \wedge T = T_0 \\
 &\quad \wedge \\
 0 &\leq i < k < N \\
 &\quad \wedge \\
 T[i] &= T[N - k + i]
 \end{aligned}$$

6 Approche Constructive

```

1 int main(void)
2 {
3     // Les commandes Latex sont permises dans les commentaires sur une ligne. Exemple :  $x_i \leq a^b$ 
4     printf("Bonjour tout le monde !");
5     /*
6     Dans les commentaires sur plusieurs lignes, elles doivent être entourées
7     de symboles définis par l'option « escapeinside » de \lstset
8      $\sum_{i=1}^N 1 = N$ 
9     La commande « \coms » permet de colorer correctement le code latex ajouté.
10    Les accents et tous les autres diacritiques sont permis : àâççôëêëêëœ...
11    */
12    return 1;
13 }
```

Extrait de Code 5 – Un programme tout simple

Il est possible de faire référence à la ligne 12 de l'extrait de code.

7 Code Complet

8 Complexité

9 Conclusion