# INFO0947: Prefixe and Suffixe

Groupe 33: Aleksandr Pavlov, Alexandre Gendebien

# Table des matières

# 1 Introduction

!!!!!! text generé par le AI!!!!!!!!!

Ce projet s'inscrit dans le cadre du cours INFO0947 : Compléments de Programmation à l'Université de Liège. Nous devons résoudre le problème de recherche du plus grand sous-tableau qui soit à la fois préfixe et suffixe d'un tableau donné, sans utiliser de tableau intermédiaire.

Le problème présente un intérêt particulier en algorithmique des chaînes et trouve des applications dans divers domaines comme la bioinformatique ou le traitement de texte. Notre approche se base sur une analyse systématique des propriétés des préfixes et suffixes, avec une contrainte forte sur l'utilisation exclusive de boucles while et l'interdiction de structures de données auxiliaires.

# 2 Formalisation du Problème

$$\text{prefixe\_suffixe}(T, N) \equiv \max\{k \mid k \in 0 \dots N-1 \wedge \forall i \cdot (i \in 0 \dots k-1 \Rightarrow T[i] = T[N-k+i])\}$$

# 3 Définition et Analyse du Problème

## 3.1 Input/Output

- **Input** : Un tableau d'entiers $T$ de taille $N$, $N \geq 0$
- **Output** : Un entier $k$ représentant la longueur du plus grand sous-tableau préfixe et suffixe 0 si aucun sous-tableau non trivial ne satisfait la condition

## 3.2 Découpe en sous-problèmes

Nous décomposons le problème en deux Sp :

1. Recherche du plus grand prefixe-suffics de longeur $k$ possible de $N-1$ à 1
2. Vérification que le préfixe et suffixe de longueur $k$ sont égaux

# 4 Specifications

1. SP1 : Recherche du plus grand prefixe-suffics :
   - **Précondition** : $T$ pointer vers un tableau, $N \geq 0$
   - **Postcondition** : $T = T_0$
   - **Retour** : le plus grand $k \in [0, N-1]$ tel que $T[0..k-1] = T[N-k..N-1]$
2. SP2 : Vérification que le préfixe et sufixe de longueur $k$ sont égaux :
   - **Précondition** : $T$ pointer vers un tableau, $0 < N, 0 < k < N$
   - **Postcondition** : $T = T_0$
   - **Retour** : 1 si $T[0..k-1] = T[N-k..N-1]$ sinon 0

# 5 Invariants

1. SP1 :
   **Invariant formel :**
   $0 < k < N$
   $\wedge$
   $T[0..k-1] \neq T[N-k..N-1]$

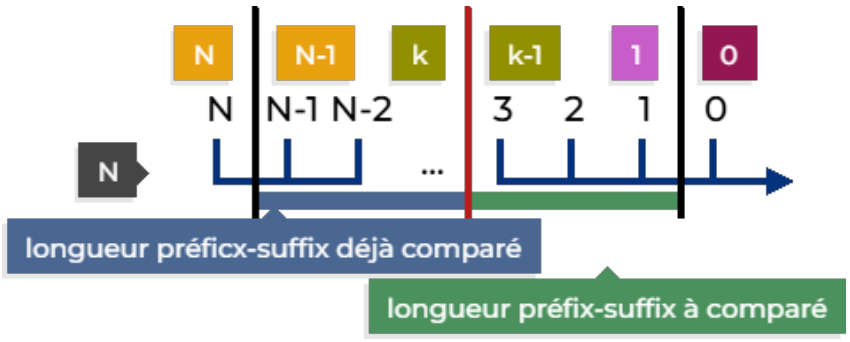


FIGURE 1 – Invariant graphique SP1

2. SP2 :
   **Invariant formel :**
   $T = T_0 \wedge k = k_0$
   $\wedge$
   $0 \leq i < k$
   $\wedge$
   $T[i] = T[N - k + i]$

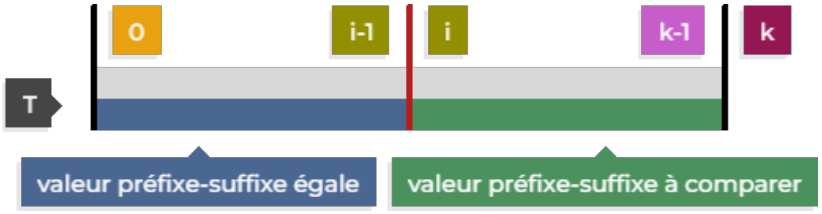


FIGURE 2 – Invariant graphique SP2

## 6 Approche Constructive

```
1  int main(void)
2  {
3      // Les commandes Latex sont permises dans les commentaires sur une ligne. Exemple : x_i ≤ 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      ∑_{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 : àÀçÇéÉèÈêÊœŒ...
11     */
12     return 1;
13 }
```

Extrait de Code 1 – Un programme tout simple

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

# 7 Code Complet

```c
#include <assert.h>
#include <stdlib.h>

#include "prefixe_suffixe.h"

// ===== Prototypes =====

/**
 * Checking if a prefix of length k is equal to the suffix.
 *
 * @param T: input array.
 * @param N: length of array T.
 * @param k: length of prefix and suffix.
 *
 * @pre: T != NULL, 0 < N, 0 < k < N
 * @post: T = T_0, N = N_0, k = k_0
 *
 * @return:
 *      0 Prefix and suffix are NOT equal
 *      1 Prefix and suffix are equal
 */
static int pref_equal_suff(int *T, const unsigned int N, unsigned int k);

// ===== Code =====

/**
 * Sp 1
 * Checking all prefixes starting from the longest one until we find a match
 * or exhaust all possibilities.
 */
int prefixe_suffixe(int *T, const unsigned int N) {
   assert((T != NULL) && (0 < N));

   for (unsigned int k = N - 1; k > 0; k--) {
      if (pref_equal_suff(T, N, k)) return k;
   }
   return 0;
}

/**
 * Sp 2
 * Comparing the prefix and suffix of the given length, element by element.
 */
static int pref_equal_suff(int *T, const unsigned int N, const unsigned int k) {
   assert((T != NULL) && (0 < N) && (0 < k && k < N));

   for (unsigned int i = 0; i <= k - 1; i++) {
      if (T[i] != T[N - k + i]) return 0;
   }
   return 1;
}
```

Extrait de Code 2 – Implémentation de prefixe_suffixe

# 8 Complexité

**Complexité :**

— Complexité de la fonction `pref_equal_suff` :
  — Dans le pire cas, la fonction effectue $k$ comparaisons
  — Dans cas maximal ($k = N - 1$), la complexité est $\mathcal{O}(N)$
— Complexité de la fonction `prefixe_suffixe` :
  — Dans le pire cas, itère sur toutes les valeurs de $k$ de $N - 1$ à 1
  — Appelle `pref_equal_suff` pour chaque $k$

**Complexité totale :**

— $\sum_{k=1}^{N-1} \mathcal{O}(k) = \mathcal{O}(N^2)$

# 9 Conclusion

C'est un cours difficile