# *Feuille de Travaux Pratiques*
# Le Problème PLTC

Janvier-Février 2017

Six séances de TP seront consacrées à ce projet, lequel est à effectuer soit seul(e), soit en binôme.

La programmation se fera dans le langage de votre choix.

A l'issue des 6 séances, vous rendrez les sources de votre projet, sous la forme d'une archive NOM1-NOM2.zip contenant votre code, ainsi que votre compte-rendu. La décompression de l'archive doit produire un répertoire NOM1-NOM2 contenant tous les éléments de votre travail (compte-rendu, sources, exécutable, jeux d'essai le cas échéant) et un fichier texte contenant les instructions de compilation et d'exécution.
*Assurez-vous que vos programmes compilent et fonctionnent sous Linux dans les salles machine du CIE.*

Cette archive est à déposer sous Madoc. La date limite de restitution est fixée au Lundi 20 Février 2017 à 12h (aucun dépôt après cette heure ne sera possible).

Un malus pourra être appliqué, en fonction du non-respect de tout ou partie des consignes ci-dessus.

Note : dans ce mini-projet, la programmation ne présente aucune difficulté technique. La très grande majorité de la note sera donc donnée en fonction de :
- la qualité de vos tests
- la qualité de l'analyse que vous ferez de ces tests
- plus globalement, la qualité de votre compte-rendu

## 1 Introduction : le problème PLTC

On se donne deux suites de caractères $S$ et $T$, qu'on appellera des *textes*. Pour simplifier, on supposera que $S$ et $T$ sont de même longueur $n$ et qu'ils sont stockés dans des tableaux $S[]$ et $T[]$ indexés de 0 à $n-1$. Pour les tests, on se limitera à un alphabet de 4 caractères, comme on en trouve en bio-informatique dans les données issues de séquences ADN (avec les caractères A, T, C et G).

Dans ce projet, on veut rechercher la **longueur** du *plus long texte commun* à $S$ et $T$. Un texte commun $P$ entre deux textes $S$ et $T$ est une suite *consécutive* de caractères qui se situe dans $S$ (par exemple $S[i \dots j]$) et dans $T$ (par exemple $T[i' \dots j']$), et telle que $S[i \dots j] = T[i' \dots j'] = P$.
Par exemple, si $S =$"TAGACTCA" et $T =$"AGAGACTT", alors le plus long texte commun est $P = S[1 \dots 5] = T[2 \dots 6] =$"AGACT", et sa longueur est de 5.

Dans la suite, on appellera **PLTC** le problème consistant à déterminer la longueur du plus long texte commun à $S$ et à $T$. Le but de ce projet est d'étudier trois algorithmes qui résolvent **PLTC**, et plus précisément, pour chacun de ces algorithmes :

1. de l'implémenter
2. d'évaluer sa complexité de manière théorique
3. d'évaluer sa complexité de manière expérimentale à partir d'une série de tests
4. de comparer les résultats obtenus aux points 2. et 3. ci-dessus, et d'expliquer les différences s'il y en a

Le travail demandé est détaillé dans la Section 2. ci-après.

# 2 Travail demandé

Le travail demandé se décompose en deux parties :

**Partie I**. Proposer une interface à l'utilisateur, qui lui permet :
— de saisir deux textes $S$ et $T$
— de choisir parmi les 3 algorithmes proposés en Section 3
En sortie, l'utilisateur doit obtenir la longueur du PLTC de $S$ et de $T$ et le temps d'exécution de l'algorithme choisi. L'interface doit ensuite proposer (au moins) trois choix : quitter le programme, entrer d'autres textes $S$ et $T$, ou alors tester un autre algorithme sur les mêmes textes $S$ et $T$, qui ne seront donc pas ressaisis.

**Partie II**. Pour chacun des trois algorithmes décrits dans la Section 3, répondre aux questions suivantes :

1. Implémenter l'algorithme.
2. Déterminer sa complexité de manière théorique, en donnant tous les détails de vos calculs dans le rapport. On attend ici une complexité en notation de Landau, en étant le plus précis possible.
3. Exécuter l'algorithme sur des paires de textes $(S, T)$ de plus en plus longues, et en particulier remplir le tableau ci-dessous, qui indique les temps d'exécution de chacun de vos tests. **Ce temps d'exécution doit systématiquement être donné en *secondes (s)*, avec trois chiffres significatifs**. Attention, le temps d'exécution ne doit prendre en compte que l'algorithme qui résout PLTC, pas la génération des textes.

| Algorithme $A_i$ - Temps d'exécution | | | | | |
|---|---|---|---|---|---|
| $n$ | Nombre de tests effectués | Analyse exhaustive ? (Oui/Non) | Temps d'exécution au mieux | en moyenne | au pire |
| 1 | | | | | |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |
| 5 | | | | | |
| 6 | | | | | |
| 7 | | | | | |
| 8 | | | | | |
| 9 | | | | | |
| 10 | | | | | |
| etc. | | | | | |

**Remarques** :
— Les valeurs de $n$ à tester doivent être les suivantes : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500, etc.
— Pour les petites valeurs de $n$, les textes $S$ et $T$ seront générés de manière exhaustive (c'est-à-dire que toutes les paires de textes de longueur $n$ sur un alphabet de taille 4 seront testées), puis de manière aléatoire pour les valeurs plus grandes. Pour savoir à partir de quelle valeur de $n$ il faut passer d'une génération exhaustive à une génération aléatoire, voir le point suivant.
— Pour chaque valeur de $n$, le nombre de tests effectués doit être le plus grand possible, tout en gardant un temps d'exécution inférieur à 3 minutes pour l'ensemble de la ligne. On arrêtera donc les tests exhaustifs quand une ligne prend plus de 3 minutes, et on continuera alors avec un échantillon aléatoire, le plus grand possible, tout en restant sous les 3 minutes par ligne.
— On arrêtera les tests à la première valeur de $n$ pour laquelle *un seul test* dépasse 3 minutes.
— Si le temps d'exécution pour un seul test dépasse 3 minutes, indiquer NRP (pour "Ne Répond Pas") dans la case du tableau qui correspond. Si un dépassement mémoire apparaît, indiquer DM dans le tableau.

4. Sur un graphique, tracer les courbes représentant les temps d'exécution au mieux, en moyenne et au pire en fonction de $n$ (un graphique par algorithme).
5. Déduire de façon expérimentale (c'est-à-dire sur la base des Questions 3. et 4.) la complexité au pire de l'algorithme étudié. On attend ici une complexité d'abord (1) sous la forme d'une fonction précise (la plus proche possible des courbes obtenues), puis (2) en notation de Landau. La méthode qui vous a permis de déterminer la complexité (1) doit être expliquée en détail dans le rapport.
6. Pour chaque algorithme, comparer les complexités au pire expérimentale et théorique obtenues précédemment. Si une différence apparaît entre ces deux complexités, proposer une (ou des) explication(s) à ce phénomène.

# 3 Les trois algorithmes à implémenter et tester

## 3.1 Algorithme $A_1$

On s'intéresse dans un premier temps à la longueur du plus long **suffixe** commun des deux textes $S$ et $T$. On appellera cette longueur $pls(S, T)$. On rappelle qu'un suffixe d'un texte $S$ est un texte $Q$ tel que $Q$ représente la fin de $S$. Par exemple, si $S =$"TGCGCAACGCAA", alors $Q =$"CAA" est un suffixe, mais aussi $Q =$"ACGCAA". Le plus long suffixe commun à deux textes $S$ et $T$ est un texte $Q$ qui est à la fois suffixe de $S$ et de $T$, et de longueur maximum. Par exemple, si $S =$"TGCGCAACGCAA" et $T =$"ACCGCAA", alors $Q =$"CGCAA" est le plus long suffixe commun, et il est de longueur $pls(S, T) = 5$.

Pour tout $0 \leq i \leq n-1$, $S_i = S[0 \dots i]$ est le texte $S$ limité à ses $(i+1)$ premiers caractères. Pour tout $0 \leq j \leq n-1$, $T_j = T[0 \dots j]$ est le texte $T$ limité à ses $(j+1)$ premiers caractères. L'algorithme $A_1$ consiste alors à :

1. Remplir une matrice $M_{pls}$ à $n$ lignes et $n$ colonnes, où, pour tous $0 \leq i, j \leq n-1$, $M_{pls}[i][j] = pls(S_i, T_j)$. La valeur de $pls(S_i, T_j)$ sera calculée de la manière suivante :

   — Pour tout $0 \leq i \leq n-1$
   $pls(S_i, T_0) = 1$ si $S[i] = T[0]$
   $pls(S_i, T_0) = 0$ sinon
   — Pour tout $0 \leq j \leq n-1$
   $pls(S_0, T_j) = 1$ si $T[j] = S[0]$
   $pls(S_0, T_j) = 0$ sinon
   — Pour tous $1 \leq i \leq n-1$ et $1 \leq j \leq n-1$
   — $pls(S_i, T_j) = pls(S_{i-1}, T_{j-1}) + 1$ si $S[i] = T[j]$
   — $pls(S_i, T_j) = 0$ sinon

2. Parcourir $M_{pls}$ pour trouver son maximum $Max$.
3. Renvoyer $Max$ ($Max$ est en effet la longueur du PLTC entre $S$ et $T$).

## 3.2 Algorithme $A_2$

Soit $Max$ un entier, initialisé à zéro. L'algorithme $A_2$ est décrit sommairement ci-dessous :

```
Pour tout 1 ≤ l ≤ n faire
    Pour tous 0 ≤ a ≤ b ≤ n − 1 tels que b − a + 1 = l faire
        Pour tous 0 ≤ p ≤ q ≤ n − 1 tels que q − p + 1 = l faire
            Si S[a . . . b] = T[p . . . q] alors
                Max ← l
            FinSi
        FinPour
    FinPour
FinPour
```

Renvoyer $Max$ (c'est la longueur du PLTC entre $S$ et $T$).

**Remarque :** Pour le calcul de complexité théorique (Question 2. de la Section 2), on considérera sans démonstration que le test d'égalité de deux chaînes de longueur $l$ est un $\Theta(l)$.

## 3.3 Algorithme $A_3$

Pour toute paire de positions $0 \leq i \leq j \leq n-1$, on appellera $S_{ij} = S[i \dots j]$, et $s_{ij} = j - i + 1$ la longueur de ce texte. Dans l'algorithme $A_3$, $S_{ij}$ sera considéré comme un tableau indicé de 0 à $s_{ij} - 1$.

L'algorithme $A_3$ fonctionne alors comme suit : Soit $Max$ une variable entière initialisée à 0. Pour tous $0 \leq i \leq j \leq n-1$, faire :

— Remplir le tableau PMK_ij[] (***ce tableau est indicé de 0 à*** $s_{ij}$) en suivant l'algorithme donné ci-dessous :

```
 1:  a <-- 0
 2:  b <-- -1
 3:  PMK_ij[0] <-- -1
 4:
 5:  Tant que (a< s_ij) faire
 6:       Tant que (b > -1 && S_ij[a]!=S_ij[b]) faire
 7:           b <-- PMK_ij[b]
 8:       Fin Ttque
 9:
10:       a <-- a+1
11:       b <-- b+1
12:
13:       Si a<s_ij et S_ij[a]=S_ij[b] alors
14:            PMK_ij[a] <-- PMK_ij[b]
15:       Sinon
16:            PMK_ij[a] <-- b
17:      FinSi
18:  Fin Ttque
```

— Exécuter ensuite l'algorithme suivant :

```
20:  bool_res <-- FAUX
21:  a <-- 0
22:
23:  Pour b de 0 à n-1 faire
24:       Tant que (a > -1 && S_ij[a] != T[b]) faire
25:            a <-- PMK_ij[a]
26:       Fin Ttque
27:
28:       a <-- a+1
29:
30:       Si a >= s_ij alors
31:            bool_res <-- VRAI
32:            a <-- PMK_ij[a]
33:       Fin Si
34:  Fin Pour
35:
36:  Retourner bool_res
```

— Si l'algorithme ci-dessus renvoie VRAI, faire $Max \leftarrow \max\{Max, s_{ij}\}$

Renvoyer $Max$, qui est la longueur du PLTC entre $S$ et $T$.

**Remarque :** Pour le calcul de complexité théorique (Question 2. de la Section 2), on considérera sans démonstration que la complexité des boucles `Tant que` aux lignes 6–8 et 24–26 est constante (càd un $O(1)$).