TP AppING1 & AppX1

E. Renault

1 Recherche de motifs et police scientifique

Par une longue soirée d'hiver, le docteur Lenoir a été tué dans la cuisine avec une raquette de tennis. Vous avez remarqué une tâche de sang qui a probablement été laissée par l'assassin. Cette tâche de sang ne vous fourni qu'un échantillon incomplet d'ADN. Dans un permier temps le labo vous indique que le ou les suspects possèdent 10 occurrences du motif ATACATAT dans leur ADN. Dans cet exercice nous nous intéressons à l'identification de l'assassin.

Question 1. Afin de pouvoir intentifier l'assassin, vous devez être en mesure de lire un fichier d'ADN. [1] Un tel fichier peut être vu comme une unique chaine de caractère composé des lettres A, T, G et C. Écrire la fonction permettant de lire un tel fichier et de le stocker dans un tableau alloué dynamiquement.

```
char* read_dna_file(char* filename);
```

Vous pouvez tester votre fonction sur les séquences ADN des différents suspects identifiés :

— Colonel Moutarde
`https://www.lrde.epita.fr/~renault/teaching/alga/dna-moutarde.txt`
— Mademoiselle Rose
`https://www.lrde.epita.fr/~renault/teaching/alga/dna-rose.txt`
— Docteur Olive
`https://www.lrde.epita.fr/~renault/teaching/alga/dna-olive.txt`
— Madame Leblanc
`https://www.lrde.epita.fr/~renault/teaching/alga/dna-leblanc.txt`

Question 2. Ecrivez la méthode naïve (vue en cours) permettant de rechercher un motif dans une chaîne. Cette fonction doit retourner le nombre d'occurence du motif.

```
unsigned naive_pattern_match(char* dna, unsigned dna_size,
                             char* pattern, unsigned pattern_size);
```

Question 3. Vous vous rendez maintenant compte que votre méthode n'est pas très efficace. Pour cela vous décider d'implémenter une version optimisée de votre algorithme. Cette nouvelle version est basée sur l'algorithme de Boyer-Moore (vu en cours). Implémentez cette fonction.

```
unsigned bm_pattern_match(char* dna, unsigned dna_size,
                          char* pattern, unsigned pattern_size);
```

Vérifier que cette fonction donne les même résultats que la méthode précédente. Pouvez vous maintenant réduire la liste des suspects ?

Question 4. Le labo vous rappelle maintenant pour vous dire que l'assassin possède nécessairement la séquence suivante : AGATAAGATA Qui est l'assassin ? Quel est son motif ?

1. Nous n'tulisons pas ici les fichiers FASTA qui sont généralement utilisés pour représenter des séquences d'ADN

2 Programmation dynamique et distance d'édition

Un groupe d'étudiants de l'EPITA décide de construire un éditeur de texte. Comme ils ont déjà réalisé l'exercice précédent, ils peuvent offrir la fonction *Rechercher* à l'utilisateur. Ils souhaitent maintenant offrir un correcteur orhtographique.

Pour réaliser cela, ils décident d'utiliser la distance de Levenshtein qui permet de savoir le plus petit nombre d'opérations nécessaires pour passer d'un mot a l'autre. Les opérations autorisées sont les suivantes :

- *Insérer*. Par exemple pour passer de *ingénu* à *ingénue*, il suffit d'insérer un 'e' à la fin du mot ;
- *Supprimer*. Par exemple pour passer de *considérer* à *sidérer* il suffit de supprimer les lettres 'n' 'c' 'o' ;
- *Remplacer*. Par exemple pour de *toto* à *tata* il faut remplacer les deux 'o' par des 'a'.

Question 1. Proposer une définition récursive de cette distance.

```
unsigned levenshtein_rec (char* s1, unsigned s1_size,
                          char* s2, unsigned s2_size);
```

Question 2. Proposer une amélioration de cette fonction en utilisant la programmation dynamique.

```
unsigned levenshtein_dyn(char* s1, unsigned s1_size,
                         char* s2, unsigned s2_size);
```