TD RECHERCHE TEXTUELLE

Dans ce TD, nous allons étudier une version simple de l'algorithme de Boyer-Moore (version Horspool). Cet algorithme assez complexe est utilisé pour détecter la présence d'une courte chaîne de caractères à l'intérieur d'une autre chaîne de caractère, bien plus longue.
Ce Td se décompose en deux parties : une partie papier et une partie programmation dans un notebook.
On a donc à notre disposition :

- un **motif** à rechercher `"AT-THAT"` ;
- dans un **texte** `"WHICH-FINALLY-HALTS.--AT-THAT-POINT"`.

La longueur du motif est notée *n* et la longueur du texte *N*.
Remarque : Votre professeur a lu et entendu des choses contradictoires à propos de cet algorithme, notamment qu'il était efficace pour chercher des séquences d'ADN dans un génome (donc avec 4 lettres A C G T), et aussi qu'il n'était efficace que lorsque l'alphabet utilisé comporte beaucoup de lettres… ceci étant un peu gênant pour l'alphabet du génome et ses 4 lettres.

1. RECHERCHE NAIVE

Dans la recherche naïve, on parcourt le texte lettre à lettre, en comparant le motif à une fenêtre glissante dans le texte. On effectue la comparaison de droite à gauche (pour des raisons techniques que seuls ceux qui veulent faire le notebook en entier verront). On a donc l'algorithme suivant :

Données :
- texte t de longueur N
- motif x de longueur n

Renvoie :
- toutes les positions de x dans t

```
Début
    P ← liste vide des positions de x dans t
    pour i de 0 à N − n + 1 :
        j ← n − 1
        Tant que j ≥ 0 et x_j = t_{i+j} :
            j ← j − 1
        Si j = −1 :
            ajouter i à P
    Renvoyer P
Fin
```

A. SUR PAPIER

- Itération $i = 0$:
  texte (avec la fenêtre flottante) : WHICH-FINALLY-HALTS.--AT-THAT-POINT
  motif : AT-THAT
  Pour $j = 6$, $x_j = x_6 = T \neq F = t_6 = t_{i+j}$. La boucle **tant que** de l'algorithme s'arrête et le motif n'est pas trouvé.
- Itération $i = 1$:
  texte (avec la fenêtre flottante) : WHICH-FINALLY-HALTS.--AT-THAT-POINT
  motif : AT-THAT
  Pour $j =$, $x_j = x = T \neq F = t = t_{i+j}$. La boucle **tant que** de l'algorithme s'arrête et le motif n'est pas trouvé.
- …
- Itération $i =$:
  texte (avec la fenêtre flottante) : WHICH-FINALLY-HALTS.--AT-THAT-POINT
  motif : AT-THAT

Pour `j =` , $x_j = x = T \neq F = t = t_{i+j}$. La boucle **tant que** de l'algorithme s'arrête et le motif n'est pas trouvé.

- …
- Itération `i =` :
  texte (avec la fenêtre flottante) : `WHICH-FINALLY-HALTS.--AT-THAT-POINT`
  motif : `AT-THAT`
  Pour `j =` , la boucle **tant que** de l'algorithme s'arrête et le motif est trouvé.
- …
- Itération `i =` :
  texte (avec la fenêtre flottante) : `WHICH-FINALLY-HALTS.--AT-THAT-POINT`
  motif : `AT-THAT`
  Pour `j =` , $x_j = x = T \neq F = t = t_{i+j}$. La boucle **tant que** de l'algorithme s'arrête et le motif n'est pas trouvé. La boucle pour s'arrête également
- L'algorithme renvoie *P* =

Que va renvoyer cet algorithme pour :

- texte = "`acaabbabaabaaab`" et motif = "`abaa`" ?
- texte = "`acaabbabababbaaab`" et motif = "`abaa`" ?

B. Programmer cet algorithme dans le notebook

C. Complexite

Compter le nombre de comparaisons pour :

- texte = "`aaaaa`" et motif = "`aa`" ;
- texte = "`aaaaaa`" et motif = "`aaa`".

Généraliser à un texte de longueur *N* et un motif de longueur *n*. On remarque que l'on a une complexité dépendant de deux paramètres.

2. Le mauvais caractere surtout et un peu le bon suffixe

On va introduire dans ce paragraphe les règles utilisées utilisée par l'algorithme de Boyer-Moore :

- Le mauvais caractère est le premier caractère de la fenêtre ne correspondant pas à un caractère du motif. C'est la règle utilisée par l'algorithme de Horspool
- Le bon suffixe est constitué des caractères situés après le mauvais caractère.

Exemples :

- Avec :
  `WHICH-FINALLY-HALTS.--AT-THAT-POINT`
  `AT-THAT`
  Mauvais caractère `F`, pas de bon suffixe.
- Avec :
  `WHICH-FINALLY-HALTS.--AT-THAT-POINT`
  `AT-THA`**`T`**
  Mauvais caractère `L`, bon suffixe `T`.
- Avec :
  `WHICH-FINALLY-HALTS.--AT-THAT-POINT`
  `AT-TH`**`AT`**
  Mauvais caractère `-`, bon suffixe `AT`.

A. Exercice sur papier

Trouver le mauvais caractère et le bon suffixe dans les cas suivants (on ne tient pas compte des accents éventuellement).

- Ces airs de valse triste avaient une saveur de bleu et de jaune
  alsace
- Ces airs de valse triste avaient une saveur de bleu et de jaune
  alpiniste
- Ces airs de valse triste avaient une saveur de bleu et de jaune
  béjaune

B. Principe de l algorithme de Horspool

L'idée centrale de l'algorithme est de décaler la fenêtre non pas d'une unité, mais d'autant que possible lorsque le motif n'est pas trouvé. Pour cela on va construire un tableau associatif (un dictionnaire) donnant la dernière position de chaque caractère du motif (sauf pour la dernière lettre, cf. ci-après). On pourra ainsi décaler la fenêtre glissante sur le texte en « collant » au mauvais caractère lors de l'itération précédente.

Exemple 1 :

```
WHICH-FINALLY-HALTS.--AT-THAT-POINT
       AT-THAT
```

Le mauvais caractère est le tiret - , qui est présent dans le motif : on décale la fenêtre de manière à ce que les deux tirets coïncident. On obtient :

```
WHICH-FINALLY-HALTS.--AT-THAT-POINT
          AT-THAT
```

Exemple 2 :

```
WHICH-FINALLY-HALTS.--AT-THAT-POINT
AT-THAT
```

Le mauvais caractère est le F, qui n'est pas dans le motif : on décale la fenêtre juste après le mauvais caractère. On obtient :

```
WHICH-FINALLY-HALTS.--AT-THAT-POINT
       AT-THAT
```

Exemple 3 :

```
WHICH-FINALLY-HALTS.--AT-THAT-POINT
          AT-THAT
```

Le mauvais caractère est le L, qui n'est pas dans le motif : on décale la fenêtre juste après le mauvais caractère. On obtient :

```
WHICH-FINALLY-HALTS.--AT-THAT-POINT
                 AT-THAT
```

Exemple 4 :

```
WHICH-FINALLY-HATTTHTKT--AT-THAT-POINT
          ATTTHTKT
```

Le mauvais caractère est T, qui est présent plusieurs fois dans le motif. De combien décaler ? Si on décale en se fixant sur le T final, on repart en arrière en faisant reculer la fenêtre. Si on décale en se basant sur la première occurrence du T dans le motif, on risque de rater le motif.

Avec un décalage sur la dernière lettre qui est T on recule :

```
WHICH-FINALLY-HATTTHTKT--AT-THAT-POINT
           ATTTHTKT
```

Avec un décalage sur le premier T on rate le motif :

```
WHICH-FINALLY-HATTTHTKT--AT-THAT-POINT
                  ATTTHTKT
```

Avec un décalage sur le dernier T, qui n'est pas en position finale, on a toujours la possibilité de trouver le motif :

```
WHICH-FINALLY-HATTTHTKT--AT-THAT-POINT
             ATTTHTKT
```

Remarquez que dans ce dernier cas la prochain décalage trouvera le motif.

On choisit finalement un décalage correspondant à l'occurrence la plus à droite du caractère, non finale.

Exemple 5 :

```
JOLIES BAILADES EN OCCITANIE
        MALADES
```

Si on aligne le mauvais caractère L avec son occurrence la plus à droite non finale dans le motif, on recule :

```
JOLIES BALLADES EN OCCITANIE
       MALADES
```

Dans ce cas on fait le choix de la recherche simple : on avance d'un caractère.

```
JOLIES BALLADES EN OCCITANIE
         MALADES
```

C. UN EXEMPLE SUR PAPIER

Faire tourner cet algorithme (même s'il n'est pas formalisé) avec le texte et le motif suivants. Vous remarquerez qu'il y a juste le bon nombre de lignes à compléter. *On ne tiendra pas compte des accents.*

```
LES PÉCHEURS RUDOYÉS DE CET OUEST BATTU DES VENTS FONT DES PILOTES HABILES
PILOTES
```

- 1ère étape : décalage de 7 caractères

```
LES PÉCHEURS RUDOYÉS DE CET OUEST BATTU DES VENTS FONT DES PILOTES HABILES
       PILOTES
```

- 1ère étape : décalage de caractères

```
LES PÉCHEURS RUDOYÉS DE CET OUEST BATTU DES VENTS FONT DES PILOTES HABILES
```

- 2ème étape : décalage de caractères

```
LES PÉCHEURS RUDOYÉS DE CET OUEST BATTU DES VENTS FONT DES PILOTES HABILES
```

- 3ème étape : décalage de caractères

```
LES PÉCHEURS RUDOYÉS DE CET OUEST BATTU DES VENTS FONT DES PILOTES HABILES
```

- 4ème étape : décalage de caractères

```
LES PÉCHEURS RUDOYÉS DE CET OUEST BATTU DES VENTS FONT DES PILOTES HABILES
```

- 5ème étape : décalage de caractères

```
LES PÉCHEURS RUDOYÉS DE CET OUEST BATTU DES VENTS FONT DES PILOTES HABILES
```

- 6ème étape : décalage de caractères

```
LES PÉCHEURS RUDOYÉS DE CET OUEST BATTU DES VENTS FONT DES PILOTES HABILES
```

- 7ème étape : décalage de caractères

```
LES PÉCHEURS RUDOYÉS DE CET OUEST BATTU DES VENTS FONT DES PILOTES HABILES
```

- 8ème étape : décalage de caractères

```
LES PÉCHEURS RUDOYÉS DE CET OUEST BATTU DES VENTS FONT DES PILOTES HABILES
```

- 9ème étape : décalage de caractères

```
LES PÉCHEURS RUDOYÉS DE CET OUEST BATTU DES VENTS FONT DES PILOTES HABILES
```

- 10ème étape : décalage de caractères

```
LES PÉCHEURS RUDOYÉS DE CET OUEST BATTU DES VENTS FONT DES PILOTES HABILES
```

- 11ème étape : décalage de caractères

```
LES PÉCHEURS RUDOYÉS DE CET OUEST BATTU DES VENTS FONT DES PILOTES HABILES
```

- 12ème étape : décalage de caractères

```
LES PÉCHEURS RUDOYÉS DE CET OUEST BATTU DES VENTS FONT DES PILOTES HABILES
```

- 13ème étape : décalage de caractères

```
LES PÉCHEURS RUDOYÉS DE CET OUEST BATTU DES VENTS FONT DES PILOTES HABILES
```

- 14ème étape : décalage de caractères

```
LES PÉCHEURS RUDOYÉS DE CET OUEST BATTU DES VENTS FONT DES PILOTES HABILES
```

- 15ème étape : décalage de caractères

LES PÉCHEURS RUDOYÉS DE CET OUEST BATTU DES VENTS FONT DES PILOTES HABILES

D. Approche intuitive de la complexité

Sans avoir à faire de calcul compliqué, on peut remarquer que dans le cas du texte = "aaaaa" et motif = "aa", l'algorithme n'apporte pas d'amélioration, puisque l'on avance que d'un caractère à chaque étape. Sur l'exemple précédent à la question c/, on passe de 87 comparaisons avec l'algorithme naïf à 28 comparaisons, soit un gain conséquent.

Néanmoins la complexité de l'algorithme n'a pas changé, elle reste en $O(n \times N)$ dans les cas « pathologiques ». Par contrer on dispose d'un algorithme plus rapide en pratique, la complexité moyenne étant $O(N)$

E. Algorithme

L'algorithme de Horspool ci-dessous tient compte des modifications proposées par la règle du mauvais caractère. Comme vous pouvez le voir, il y a peu de différences –elles sont signalées par partie rajoutée ou modifiée–, et pourtant le gain est conséquent. Programmez-le dans le notebook.

Données :
- texte t de longueur N
- motif x de longueur n

Renvoie :
- toutes les positions de x dans t

```
Calculs préliminaires :    # partie rajoutée
    pos_droite ← dictionnaire de clés les caractères du motif, la
    valeur étant la position la plus à droite du caractère dans le
    motif. La dernière lettre du motif est exclue.
Début
    P ← liste vide des positions de x dans t
    Tant que i ≤ N - n :    # partie modifiée : on arrive plus vite
                            # à la fin de t qu'avec un « pour »
        j ← n - 1
        Tant que j ≥ 0 et x_j = t_{i+j} :
            j ← j - 1
        Si j = -1 :
            ajouter i à P
        Si t_{i+j} est dans pos_droite :              # partie ajoutée
            i ← i + max(1, j - pos_droite[t_{i+j}])   # cf. ci-dessous
        Sinon :
            i ← i + max(1, j + 1)
    Renvoyer P
Fin
```

Remarque : le `si-sinon` final ajouté correspond à l'application de la règle du mauvais caractère. On peut le simplifier en une seule instruction $i \leftarrow i + \max(1, j - \text{droite}(t_{i+j}))$, où `droite` est une fonction qui renvoie $pos\_droite[t_{i+j}]$ si t_{i+j} est dans le dictionnaire et -1 sinon.

F. Algorithme de Boyer Moore

Ceux qui le souhaitent peuvent d'abord le comprendre, puis le programmer dans le notebook.