Algorithmique du texte (2024-2025)

TP #1 : Recherche de motif

1 Un peu de vocabulaire

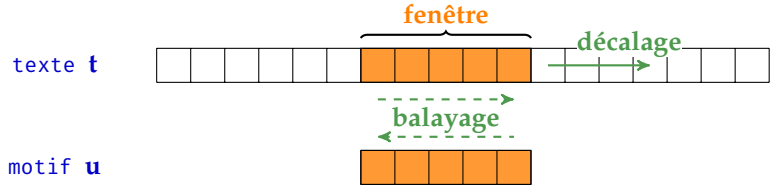
On rappelle :

- Un alphabet Σ : un ensemble fini de symboles, appelés lettres ou caractères ;
- un motif ou un texte sur l'alphabet Σ : une suite de lettres de Σ ;
- la longueur d'un mot w, notée $|w|$: le nombre de lettres du mot ;
  Par convention, on indice les lettres d'un mot à partir de 0 : $w = w[0]w[1]\dots w[n-1]$ *avec* $n = |w|$.
- le mot vide, i.e., le mot de longueur 0 : ε ;
- on note Σ^* l'ensemble de tous les mots sur Σ, et Σ^+ l'ensemble de tous les mots non vides sur Σ ;
- les préfixes de w constitue l'ensemble $\texttt{Pref}(w) = \{x :$ il existe $y \in \Sigma^*$ tel que $w = xy\}$;
- les suffixes de w consitue l'ensemble $\texttt{Suff}(w) = \{y :$ il existe $x \in \Sigma^*$ tel que $w = xy\}$;
- les facteurs de w constitue l'ensemble $\texttt{Fact}(w) = \{z :$ il existe $x, y \in \Sigma^*$ tel que $w = xzy\}$

Question 1. *Soit* $w = abbaac$. *Donner les langages (ensembles de mots)* $\texttt{Pref}(w)$, $\texttt{Suff}(w)$ *et* $\texttt{Fact}(w)$.

2 Stratégie de la fenêtre coulissante

Dans ce TP, nous allons comparer deux approches pour la recherche en utilisant la stratégie de « fenêtre coulissante ».
Le principe de cette stratégie consiste globalement à lire le texte au travers d'une fenêtre coulissante de la taille du mot.

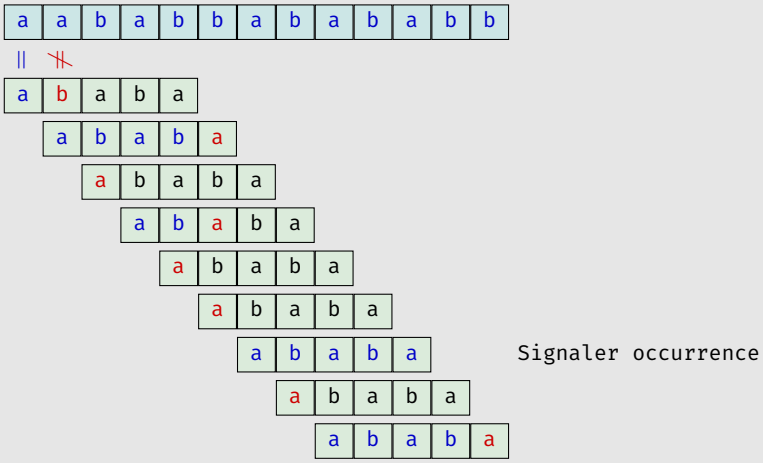


Positionner la fenêtre au début du texte;
while *(la fenêtre est sur le texte)* **do**
  if *(le motif est détecté dans la fenêtre)* **then** // **Balayage**
    Signaler occurrence du motif à la position courante;
  Décaler la fenêtre// **Décalage**;

2.1 Algorithme naïf

- La fenêtre est positionnée au début du texte.
- Le balayage consiste à examiner les symboles de gauche à droite.
- Le décalage décale la fenêtre d'une position à droite.

Sur un exemple :
Le texte : `aababbabababb`
Le motif cherché : `ababa`



Question 2. *Combien de comparaisons sont effectuées sur l'exemple ?*

Question 3. *Donner sur l'exemple au moins deux décalages qui semblent inutiles lors de l'exécution de l'algorithme naïf.*

Question 4. *Implémenter cet algorithme en `Python` dans une fonction `find(u, t)` qui retourne la liste des positions d'occurrences d'un motif `u` dans un texte `t` par la méthode naïve.* Attention, vous n'avez le droit d'utiliser que des comparaisons de symboles !

Question 5. *Comparer l'efficacité de votre algorithme sur plusieurs fichiers d'exemple. On comptera le nombre de comparaisons de caractères et le temps d'exécution (avec la commande du shell `/usr/bin/time -v` par exemple).*
Les fichiers suivants sont mis en forme et contiennent une seule ligne. (Il n'est pas utile de stocker ces fichiers dans votre répertoire : pour une utilisation temporaire, on peut les stocker dans `/tmp/`).

- *War and peace (Léon Tolstoï) (taille 3,1M)*
  `https://clementjo1.users.greyc.fr/algo-texte/ressources/war_and_peace.ascii.txt`
  Motifs à tester : "anana","the", "the the", "that", "you"
- *Escherichia coli (E. Coli) (taille 4,5M)*
  `https://clementjo1.users.greyc.fr/algo-texte/ressources/e-coli.ascii.txt`
  Motifs à tester : "GCTGGTGG", "TATATATAT"
- *aaaaaaaa . . . (taille 1,1M)*
  `https://clementjo1.users.greyc.fr/algo-texte/ressources/aaa.ascii.txt`
  Motifs à tester : "aaa", "aab", "aaaaaaaaa", "aaaaaaaab",
  $a^{50} =$ *"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"*

Que pouvez-vous conclure de ces essais ?

Question 6. *Proposer une analyse de pire cas pour le nombre de comparaisons de symboles de l'algorithme naïf prenant un texte t de taille n et un motif de taille m. Quelle est la complexité de l'algorithme naïf dans le pire des cas ?*

2.2 Algorithme de Morris-Pratt

L'algorithme de Morris-Pratt pré-calcule la table des bords du motif u pour optimiser le décalage : on mémorise les caractères du mot appariés à ceux du texte. Et on utilise la notion de *bord* pour calculer le décalage.
Un *bord* d'un mot w est un mot différent de w qui est à la fois préfixe et suffixe de w.
Le *bord maximal* de w, noté $\texttt{Bord}(w)$, est le plus long bord de w.

Question 7. *Calculer les ensembles de bords pour les mots suivant :*
- $w = ababa$;
- $w = abaababa$;
- $w = aabaabaa$.

La table des bords d'un mot w est un tableau de $|w| + 1$ entiers, noté `TB`, et défini par :

$$\texttt{TB}[i] = \begin{cases} -1 & \text{si } i = 0 \\ |\texttt{Bord}(w_0 \dots w_{i-1})| & \text{si } 0 < i \leq |m| \end{cases},$$

où $w_0 \dots w_{i-1}$ est le préfixe de longueur i de w.

Question 8. *Calculer la tables de bords pour* $w = abaababa$

i	0	1	2	3	4	5	6	7	8
w[i-1]		a	b	a	a	b	a	b	a
TB[i]									

Question 9. *Calculer la tables de bords pour* $w = aabaabaa$

i	0	1	2	3	4	5	6	7	8
w[i-1]		a	a	b	a	a	b	a	a
TB[i]									

Question 10. *Calculer la table des bords de* `w = ababa` *(le motif de l'exemple).*

Question 11. *Comment utiliser cette table pour calculer un meilleur décalage que le naïf et éviter des comparaisons inutiles ? Quelle est la propriété des bords utilisée ici ?*

Question 12. *Implanter l'algorithme de calcul de la table des bords qui prend en argument un motif et renvoie la liste (ou le tuple) correspondant à sa table des bords.*

L'algorithme se base sur le principe suivant :
Pour tout mot $u \in \Sigma^+$ *et tout caractère* $a \in \Sigma$ *:*

$$\texttt{Bord(ua)} = \begin{cases} \texttt{Bord(u)a} & \textit{si } \texttt{Bord(u)a} \textit{ est un préfixe de } \texttt{u} \\ \texttt{Bord(Bord(u)a)} & \textit{sinon} \end{cases}$$

Question 13. *Écrire une fonction de recherche* `findMP` *qui utilise la table des bords pour calculer son décalage. Tester sur les mêmes exemples que pour l'algorithme naïf (Question 5). Comparer les résultats avec l'algorithme naïf.*

Question 14. *(Bonus) Pour un texte de taille* n *et un motif de taille* m*, quelle est la complexité dans le pire des cas en nombre de comparaisons de l'algorithme de Morris-Pratt ? (Solution : cette complexité est linéaire, c'est-à-dire* $O(n + m)$*.*