TP 5 : Automates et recherche dans un texte

lionel.rieg@ens-lyon.fr

Dans ce dernier TP, nous allons voir une application des automates : la recherche de motifs dans un texte. On notera n la taille du texte et m celle du motif (utiles pour les calculs de complexité). Pour représenter les chaînes de caractères, on utilisera le type `string` de C++, disponible après avoir fait un `#include <string>`.

1 Recherche simple

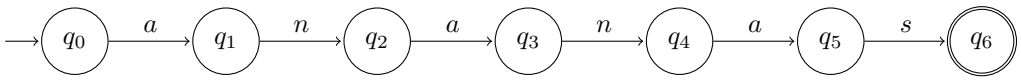
L'algorithme le plus simple pour trouver un motif est d'essayer de comparer les caractères du motif et du texte pour chaque position dans le texte. Dans ce cas simple, il n'y a pas besoin d'automate et toute l'information sur l'état de la recherche est contenu dans un couple (i, j) où i est la position courante dans le texte et j celle dans le motif. Il suffit alors d'imbriquer deux boucles `for` qui testent l'égalité des caractères pour faire la recherche. Si l'on ne recherche que la première occurrence du motif, bien penser à interrompre la boucle une fois qu'on l'a trouvée !

1. Quelle est la complexité de cet algorithme ?
2. L'implémenter.
3. Le tester avec le motif « ananas » et le texte « anatomie, ananana et ananas ».
4. Adapter votre algorithme pour qu'il renvoie le nombre d'occurrence du motif dans le texte.
5. Si maintenant on cherche le motif « nana » dans le même texte, combien trouvez-vous d'occurrences ?

Cet algorithme naïf est parfois appelé algorithme *brute force* car il teste exhaustivement toutes les possibilités sans essayer de tirer parti de la forme du motif. Dans toute la suite, on va faire un pré-traitement avec le motif pour accélérer la recherche dans le texte.

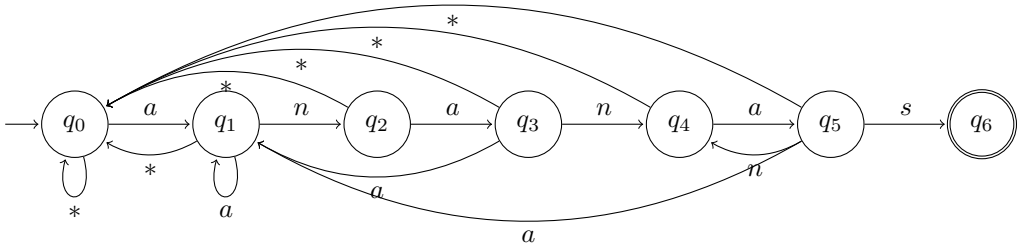
2 Automate optimal

Voici ci-dessous l'automate naïf pour rechercher « ananas » dans un texte, qu'on a implicitement utilisé avec l'algorithme précédent. Cet automate contient le motif et on le fait évoluer en lisant une lettre du texte. S'il n'y a pas de transition possible (pas de flèche avec la bonne lettre), cela signifie que le motif ne se trouve pas à cette position dans le texte. On redémarre alors l'automate en augmentant de 1 la position de départ dans le texte. Si on atteint l'état final (entouré par un double cercle), c'est qu'on a trouvé le motif.



1. Exécuter cet automate sur l'exemple et se convaincre qu'il recherche bien le motif « ananas ».

On peut améliorer cet automate pour qu'il gère lui-même les erreurs et qu'il n'y ait jamais besoin de relire un caractère deux fois ou de redémarrer l'automate. La méthode n'est pas très compliquée mais on ne la verra pas en détail ici. Il s'agit juste de bien tirer parti du fait que si l'on est dans l'état q_3, c'est qu'on a déjà lu les 3 premières lettres du motif. Voici ci-après l'automate optimal.



Les étiquettes « $*$ » correspondent à toutes les lettres qui n'ont pas d'autre transition possible.

2. Exécuter ce nouvel automate sur l'exemple et vérifier qu'il gère bien les erreurs et recherche le motif « ananas ».
3. Une fois l'automate optimal construit, quelle est la complexité de l'algorithme de recherche ?

3 Algorithme de Boyer-Moore

On va maintenant trouver un compromis : faire un automate plus simple qui permet tout de même d'avoir de bonne performance. En fait, si ce nouvel algorithme est aussi mauvais que l'algorithme naïf dans le pire cas, il est en moyenne *sous-linéaire* et n'a pas besoin de lire toutes les lettres du texte !

L'astuce consiste à lire le motif dans le texte depuis la fin. Ainsi, si on découvre une lettre qui n'apparaît pas dans le motif, on peut sauter toutes les lettres qui se trouvent avant cette position. De même si on a un conflit avec une lettre l qui se trouve ailleurs dans le motif, on décale le motif autant que possible de façon que l soit alignée avec une de ses positions dans le motif. Afin de ne pas sauter d'occurrence du motif, on aligne la lettre l du texte à la dernière apparition de l dans le motif (sauf si cela nous ferait reculer, auquel cas on avance de 1).

Pour mieux comprendre, voici par exemple l'exécution de l'algorithme lorsqu'on cherche le motif « annale » dans le texte « annuelle, anomalie, analyse, annale ».

```
annuelle, anomalie, analyse, annale
annale
```
⇝ on décale jusqu'à la dernière position de « l »
```
annuelle, anomalie, analyse, annale
 annale
```
⇝ idem
```
annuelle, anomalie, analyse, annale
  annale
```
⇝ décalage d'une case (dernière position de « l » passée)
```
annuelle, anomalie, analyse, annale
   annale
```
⇝ décalage de tout le motif car « , » n'y apparaît pas
```
annuelle, anomalie, analyse, annale
         annale
```
⇝ on décale jusqu'à la dernière position de « a »
```
annuelle, anomalie, analyse, annale
          annale
```
⇝ décalage de tout le motif
```
annuelle, anomalie, analyse, annale
                 annale
```
⇝ on décale jusqu'à la dernière position de « a »
```
annuelle, anomalie, analyse, annale
                   annale
```
⇝ décalage de tout le motif
```
annuelle, anomalie, analyse, annale
                         annale
```
⇝ on décale jusqu'à la dernière position de « n »
```
annuelle, anomalie, analyse, annale
                              annale
```
⇝ on décale jusqu'à la dernière position de « l »
```
annuelle, anomalie, analyse, annale
                                   annale
```
⇝ on a trouvé le motif !

Au total, 18 comparaisons de caractères $< 35 = m$.

1. Calculer la table des décalages qui à chaque lettre associe sa dernière position dans le motif.
2. À l'aide de cette table et de la position courante dans le motif, calculer le meilleur décalage possible.
3. Implémenter l'algorithme complet.
4. Donner un exemple de texte et de motif pour lesquels l'algorithme atteint le pire cas.
5. Adapter l'algorithme pour qu'il renvoie le nombre d'occurrence du motif dans le texte.

L'inconvénient de cet algorithme est qu'il nécessite de lire les caractères du texte dans un ordre quelconque, ce qui n'est pas toujours possible, notamment lorsque le texte et le motif sont très grands ou que le texte arrive au fur et à mesure (téléchargement en cours par exemple).