Université de Lille 1 SCIENCES ET TECHNOLOGIES

TDM 3 - Automates et Langages

Recherche de motif

septembre 2015

Trouver la première occurrence d'un motif dans un texte est une fonctionnalité courante des éditeurs de texte. Dans ce contexte, le mot *motif* est synonyme de *facteur*. Le texte est un document en cours d'édition, le motif recherché est un mot particulier fourni par l'utilisateur. En java, la méthode `indexOf` de la classe `String` réalise aussi cette opération. En effet, `m1.indexOf(m2)` renvoie le plus petit entier i tel que l'on retrouve `m2` comme facteur de `m1` à partir de la position i, ou -1 si `m2` n'est pas un facteur de `m1`. Nous allons écrire des méthodes java équivalentes, en utilisant divers algorithmes, plus ou moins efficaces.

1 Documents disponibles

Vous trouverez sur le portail l'archive `tdm3_fichiers.zip` contenant tous les fichiers nécessaires à la réalisation des exercices. Recopiez cette archive dans votre répertoire personnel et décompressez la. Le répertoire `tdm3_fichiers` contient 3 sous-répertoires `src`, `classes` et `doc`.

1. Le répertoire `src` contient un fichier `Mot.java` que vous devez compléter avec les différents algorithmes de recherche de motif.
2. Le répertoire `classes` contient la classe `AutomateMotif.class`, décrite dans la documentation. Vous n'avez pas besoin de connaître le source java de cette classe.
3. Le répertoire `doc` contient la documentation des classes utilisées.

Quelque soit l'algorithme choisi, nous allons écrire dans la classe `Mot` une méthode `indiceMotif` qui prend en paramètre un mot m et renvoie le plus petit entier i tel que m soit facteur du mot courant (`this`) à partir de la position i, ou -1 si m n'est pas facteur du mot courant. La spécification de cette méthode est :

```
int indiceMotif(Mot motif) ;
```

L'alphabet X que l'on considérera est l'ensemble des lettres minuscules de 'a' à 'z'.

2 Description des algorithmes

Par la suite, on note `motif` le motif, et `T` le texte dans lequel on cherche le motif. Si `mot` est un mot, on note `mot[i..j]` le facteur de `mot` qui va de l'indice i à l'indice j inclus. Pour terminer, `mot[i]` représente la lettre de `mot` à la position i.
Les algorithmes décrits ci-après sont donnés dans un pseudo-langage impératif.

2.1 Algorithme naïf

```
n := longueur(T) ;
m := longueur(motif) ;
Pour i allant de 0 à n-m faire {
  si motif = T[i..i+m-1] alors retourner i ;
}
retourner -1
```

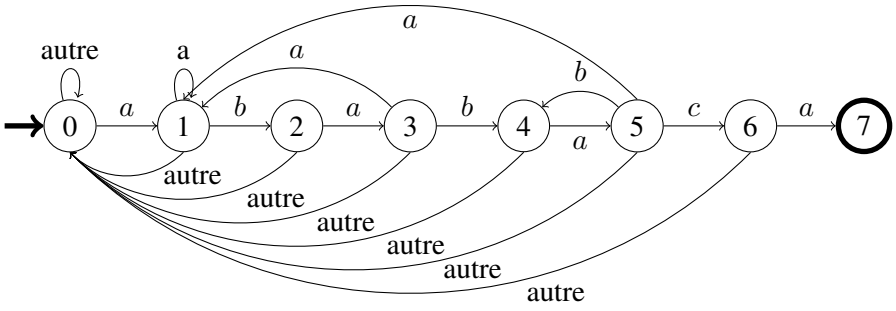
Q 1 . Programmez et testez l'algorithme naïf.

Cet algorithme est dit *naïf* car, après une comparaison infructueuse, la comparaison suivante débutera à l'indice $i + 1$ sans tenir compte de celles qui ont déjà eu lieu à l'itération précédente à la position i.

Chaque lettre du texte est donc comparée plusieurs fois aux lettres du motif, il faut donc effectuer des retours en arrière dans le texte afin de comparer les lettres aux lettres du motif recherché.

2.2 Algorithme basé sur les automates finis

A partir du mot `motif`, on construit un automate fini déterministe, qui permet de ne pas revenir en arrière lorsqu'on lit le texte `T`. On appelle *automate de motif du mot m* un automate déterministe qui reconnaît les mots qui ont m comme suffixe.
La classe `AutomateMotif` [1] permet de construire un tel automate, à partir d'un motif passé en paramètre du constructeur. L'automate ci-dessous est un automate de motif pour `abababa`. Comme tous les automates de motif que l'on manipulera, il a $l+1$ états, un état initial 0, un état final l, où l est la longueur du motif.



Une fois cet automate construit, on l'utilise pour rechercher la position du motif dans le texte :

```
Si longueur(motif) = 0 alors retourner 0 ; // => pas de transition
Sinon{
  autoMotif := automate motif pour le mot motif ;
  etat := etat initial de autoMotif ;
  Pour i allant de 0 à longueur(T)-1 faire {
    etat := delta(etat,T[i]) ; // delta fonction de transition de autoMotif
    Si (etat est final) alors retourner (i + 1 - longueur(motif)) ;
  }
  retourner -1
}
```

Q 2 . Ecrivez et testez cet algorithme, en utilisant la classe `automateMotif` fournie.

Cet algorithme est beaucoup plus efficace que l'algorithme naïf, mais présente comme défaut la construction au préalable de l'automate de motif. En particulier, la construction de la table de transition est dépendante de l'alphabet X. Le temps de calcul de la fonction de transition `delta`, lorsqu'on construit l'automate de motif, est en $\mathcal{O}(m^3 \times |X|)$, où m est la longueur du motif. Il peut être amélioré en $\mathcal{O}(m \times |X|)$, mais on ne se débarrasse pas du facteur $|X|$. Ainsi, avec la procédure améliorée, le temps d'exécution de la recherche de motif est en $\mathcal{O}(n + m \times |X|)$, où m est la longueur du motif et n la longueur du texte.

2.3 Algorithme de Knuth-Morris-Pratt

Pour terminer, voici maintenant un algorithme de recherche de motif dû à Knuth, Morris et Pratt. Il permet d'atteindre un temps d'exécution en $\mathcal{O}(n+m)$, où m est la longueur du motif et n est la longueur du texte, en évitant le calcul de la fonction de transition.
L'algorithme de Knuth-Morris-Pratt utilise une fonction auxiliaire appelée fonction préfixe, précalculée en fonction du motif.

1. On rappelle que cette classe est disponible dans le répertoire `classes`.

2.3.1 La fonction préfixe d'un motif

La fonction préfixe d'un motif exprime les correspondances entre le motif et ses propres décalages. Cette information peut être utilisée pour éviter de tester des décalages inutiles dans l'algorithme naïf, ou le précalcul de la fonction de transition pour un automate.
Prenons un exemple :

```
T       =  bacbababaabcbab
motif   =       ababaca
```

On recherche le motif dans `T` à partir de l'indice $i = 4$. La correspondance est établie pour $q = 5$ caractères, mais le 6ème caractère du motif ne correspond pas au 10ème caractère du texte `T`. Le fait de savoir que les 5 premiers caractères concordent nous permet de savoir quels décalages sont invalides.
Si on décale le motif de 1, le premier 'a' du motif va être en face d'un 'b' dans le texte, donc ce décalage est invalide :

```
T       =  bacbababaabcbab
motif   =        ababaca
```

Si on décale le motif de 2, on sait qu'on a déjà le préfixe 'aba' du motif qui concorde :

```
T       =  bacbababaabcbab
motif   =         ababaca
```

De manière générale, il faut répondre à la question suivante :
Sachant que les caractères `motif[0..q-1]` du motif correspondent aux caractères `T[i .. i+q-1]` du texte, quel est le plus petit décalage $j > i$ tel que `motif[0..k-1]` = `T[j .. j+k-1]` si $j + k = i + q$?
Au nouveau décalage j, on n'a pas besoin de comparer les k premiers caractères du motif avec les caractères correspondants dans le texte, puisqu'on est sûr qu'ils concordent.
Cette information peut être précalculée en comparant le motif avec lui-même. En fait, `T[j .. j+k-1]` est un suffixe de `motif[0..q-1]`. On recherche le plus petit décalage j, donc le plus grand k.

Le précalcul nécessaire peut être formalisé de la façon suivante : étant donné un motif `motif[0..m-1]` de longueur m, la fonction motif associée est la fonction `pref` de $\{1, 2, \ldots, m\} \rightarrow \{0, 1, \ldots, m-1\}$ telle que

$$\texttt{pref}(q) = \max\{k : k < q \text{ et } \texttt{motif}[0..k-1] \text{ suffixe de } \texttt{motif}[0..q-1]\}$$

Voici à titre d'exemple le résultat du calcul de `pref` pour le motif "abcdabd" :

0	0	0	0	1	2	0
1	2	3	4	5	6	7

Et le résultat du calcul de `pref` pour le motif "ababaca" :

0	0	1	2	3	0	1
1	2	3	4	5	6	7

Q 3 . Ecrivez une méthode privée `calculerFonctionPrefixe` qui prend en paramètre un mot `motif` et retourne un tableau d'entiers, résultat du calcul de `pref` pour ce motif.
Voici l'algorithme de calcul de ce tableau :

```
pref[1] := 0 ;
k := 0 ;
Pour q allant de 2 à longueur(motif){
  Tant que k > 0 et motif[k] != motif[q-1] faire { k := pref[k] ; }
  Si motif[k] = motif[q-1] alors  k := k+1 ;
  pref[q] := k ;
}
retourner pref ;
```

Une fois que la fonction préfixe est calculée et stockée dans un tableau, l'algorithme de Knuth-Morris-Pratt recherche le motif dans le texte `T` de la manière suivante :

```
Si longueur(motif) = 0 alors retourner 0 ;
Sinon{
  pref := calculerFonctionPrefixe(motif) ;
  q := 0 ;
  pour i allant de 0 à longueur(T)-1 faire {
     Tant que q > 0 et motif[q] != T[i] faire { q = pref[q]; }
     Si motif[q] = T[i] alors q := q+1 ;
     Si q = longueur(motif) alors retourner (i - longueur(motif) + 1) ;
  }
  retourner -1 ;
}
```

Q 4 . Ecrivez et testez la fonction de recherche de motif basé sur l'algorithme de Knuth-Morris-Pratt.