M1 Bio-Info, Info et Math-Info
Algorithmique du texte

Année 2016-2017

TP 1 : Algorithmique du texte

Sèverine Bérard - Université de Montpellier, ISE-M

Severine.Berard@umontpellier.fr

Le but de ce TP est de programmer tous les algorithmes vus en cours et d'étudier et de comparer leurs performances. Lisez bien l'énoncé et respectez bien toutes les consignes, la note de votre TP en dépendra.

Un canevas de programmation de ce TP en C++ est disponible sur l'ENT : `Canevas_TP.cc`. À utiliser sauf bonne raison contraire. La ligne pour compiler est `> g++ Canevas_TP.cc -o NomExe`.

1 Réalisations

1.1 Première partie

Vous devez programmer les quatre méthodes de recherche de motif exact vues en cours :

1. Recherche naïve, fonction nommée $naif$
2. MP, fonction nommée MP
3. KMP, fonction nommée KMP
4. Boyer-Moore, fonction nommée BM

Les algorithmes nécessaires sont rappelés dans la section 2.

Remarque importante : Nous rappelons que nous considérons que la numérotation des chaînes de caractères commence à 1. Or dans les langages de programmation la première lettre d'une chaîne est stockée à la position 0. Pour éviter des manipulations d'indices fastidieuses, *nous rangerons donc la première lettre d'une chaîne à la position 1* en ajoutant un caractère arbitraire devant nos chaînes. Nous aurons besoin d'utiliser des tableaux, comme MP_next, dont les cases correspondent aux lettres de nos chaînes. En conséquence, nous n'utiliserons jamais les cases 0 de nos tableaux et nous devrons créer des tableaux de taille 1 de plus que celle indiquée dans les algorithmes.

On veut un seul fichier de code avec toutes les fonctions à l'intérieur. Les langages autorisés sont `C`, `C++` et `java` avec une forte préférence pour `C++`. Ce fichier sera bien sûr propre et commenté.

Le programme principal appellera successivement les 4 méthodes ci-dessus pour chercher un motif P, lu dans un fichier passé en ligne de commande, dans un texte T, également lu dans un fichier passé en ligne de commande. On veut donc taper dans un terminal : `> NomExe fichierMotif fichierTexte`.

Pour chacune des méthodes vous devrez faire afficher *le nombre de comparaisons de caractères réellement effectuées*, pour cela on tiendra compte de l'*évaluation paresseuses* des opérateurs booléens. À savoir :

— l'évaluation de `(cond1) ET (cond2)` avec `(cond1)` faux n'évalue pas `(cond2)`
— l'évaluation de `(cond1) OU (cond2)` avec `(cond1)` vrai n'évalue pas `(cond2)`

Ajouts spécifiques :

1. Pour MP vous afficherez la table MP_next
2. Pour KMP vous afficherez la table KMP_next
3. Pour Boyer vous afficherez les tables $Suff$, D et R (on considère pour le moment l'alphabet classique à 26 lettres en minuscule)

1.2 Deuxième partie

Une fois la première partie réalisée, vous implémenterez la règle améliorée du mauvais caractère (*cf.* exercice 2 du TD3) qui calcule une table S. Et vous adapterez la phase de recherche de l'algorithme Boyer-Moore en conséquence, cette nouvelle version sera nommée $BM2$ sans écraser la précédente car on fera des comparaisons entre les 2.

Lors de l'examen de TP, il vous sera demandé des exécutions sur des paires de motifs et textes spécifiques ainsi que des modifications de votre code. Vous devrez rendre les réponses aux questions posées ainsi que votre fichier de TP.

2 Rappels des algorithmes

2.1 Algorithme naïf

Algorithme 1: Recherche naïve

Données : Deux chaînes T et P de longueurs respectives n et m.

$pos := 1$;
tant que $pos \leqslant n - m + 1$ **faire**
  $i := 1$;
  tant que *(*$i \leqslant m$ **et** $P[i] = T[pos + i - 1]$*)* **faire** $i := i + 1$;
  si $i = m + 1$ **alors**
    Écrire("P apparaît à la position ", pos) ;
  $pos := pos + 1$;

2.2 Algorithme MP

Algorithme 2: Calcule MP_next

Données : Un mot P de longueur m

$MP\_next[1] := 0$; $j := 0$;
pour *(*i *de* 1 *à* m*)* **faire**
  `/* À chaque entrée de boucle on a` $j := MP\_next[i]$ `*/`
  tant que *(*$j > 0$ **et** $P[i] \neq P[j]$*)* **faire** $j := MP\_next[j]$;
  $j := j + 1$;
  $MP\_next[i + 1] := j$;

Algorithme 3: Algorithme MP

Données : Deux chaînes T et P de longueurs respectives n et m.

$i := 1$; $j := 1$;
tant que $j \leqslant n$ **faire**
  tant que *(*$i > 0$ **et** $P[i] \neq T[j]$*)* **faire**
    $i := MP\_next[i]$;
  $i := i + 1$; $j := j + 1$;
  si $i = m + 1$ **alors**
    Écrire("P apparaît à la position ", $j - i + 1$) ;
    $i := MP\_next[i]$;

2.3 Algorithme KMP

Algorithme 4: Calcule KMP_next

Données : Un mot P de longueur m

```
KMP_next[1] := 0 ; j := 0 ;
pour (i de 1 à m) faire
    tant que (j > 0 et P[i] ≠ P[j]) faire j := KMP_next[j] ;
    j := j + 1 ;
    si (i = m ou P[i + 1] ≠ P[j]) alors
        KMP_next[i + 1] := j ;
    sinon
        KMP_next[i + 1] := KMP_next[j]
```

Algorithme 5: Algorithme KMP

Données : Deux chaînes T et P de longueurs respectives n et m.

```
i := 1 ; j := 1 ;
tant que j ⩽ n faire
    tant que (i > 0 et P[i] ≠ T[j]) faire
        i := KMP_next[i] ;
    i := i + 1 ; j := j + 1 ;
    si i = m + 1 alors
        Écrire("P apparaît à la position ", j − i + 1) ;
        i := KMP_next[i] ;
```

2.4 Boyer-Moore

Algorithme 6: Calcule $suff$

Données : Un mot P de longueur m

$suff[m] := m$; $g := m$;
pour *(i de* $m-1$ *à* 1*)* **faire**
  si *(*$i > g$ **et** $suff[i+m-f] \neq i-g$*)* **alors**
    $suff[i] := \min(suff[i+m-f], i-g)$;
  sinon
    $f := i$; $g := \min(g, i)$;
    tant que *(*$g > 0$ **et** $P[g] = P[g+m-f]$*)* **faire**
      $g := g-1$;
    $suff[i] := f-g$;

Algorithme 7: Calcule D

Données : Un mot P de longueur m, la table $suff$
`/* Initialisation */`
pour *(*i *de* 1 *à* m*)* **faire** $D[i] := m$;
`/* Cas 2 */`
$i := 1$;
pour *(*j *de* $m-1$ *à* 0*)* **faire**
  si *(*$j = 0$ **ou** $suff[j] = j$*)* **alors**
    tant que *(*$i \leqslant m-j$*)* **faire**
      $D[i] := m-j$;
      $i := i+1$;
`/* Cas 1 */`
pour *(*j *de* 1 *à* $m-1$*)* **faire** $D[m-suff[j]] := m-j$;

Algorithme 8: Calcule R

Données : Un mot P de longueur m, la table R
`/* À vous de l'écrire (cf. exercice 1 du TD3) */`

Algorithme 9: Algorithme de Boyer-Moore

Données : Deux chaînes T et P de longueurs respectives n et m.
$R :=$ table issue de la règle simple du mauvais caractère ;
$D :=$ table issue de la règle forte du bon suffixe ;
$pos := 1$;
tant que $pos \leqslant n-m+1$ **faire**
  $i := m$;
  tant que *(*$i > 0$ **et** $P[i] = T[pos+i-1]$*)* **faire** $i := i-1$;
  si $i = 0$ **alors**
    Écrire("P apparaît à la position ", pos) ;
    $pos := pos + D[1]$;
  sinon
    $pos := pos + \max(D[i], i - R[T[pos+i-1]])$;