

TP 1 : Recherche exacte

S  verine B  rard - ISE-M, Facult   des Sciences, Universit   de Montpellier

Severine.Berard@umontpellier.fr

Le but de ce TP est de programmer tous les algorithmes vus en cours et d  tudier et de comparer leurs performances. Lisez bien l  nonc   et respectez bien toutes les consignes, la note de votre TP en d  pendra.

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

1 R  alisations

1.1 Premi  re partie

Vous devez programmer les quatre m  thodes de recherche de motif exact suivantes (vues ou    voir en cours) :

1. Recherche na  ve, fonction nomm  e *naif*
2. MP, fonction nomm  e *MP*
3. KMP, fonction nomm  e *KMP*
4. Boyer-Moore, fonction nomm  e *BM*

Les algorithmes n  cessaires sont rappelés dans la section 2.

Remarque importante : Nous rappelons que nous consid  rons que la num  rotation des cha  nes de caract  res commence    1. Or dans les langages de programmation la premi  re lettre d  une cha  ne est stock  e    la position 0. Pour   viter des manipulations d  indices fastidieuses, *nous rangerons donc la premi  re lettre d  une cha  ne    la position 1* en ajoutant un caract  re arbitraire devant nos cha  nes. Nous aurons besoin d  utiliser des tableaux, comme *MP_next*, dont les cases correspondent aux lettres de nos cha  nes. En cons  quence, nous n  utiliserons jamais les cases 0 de nos tableaux et nous devrons cr  er des tableaux de taille 1 de plus que celle indiqu  e dans les algorithmes.

On veut un seul fichier de code avec toutes les fonctions    l  int  rieur. Les langages autoris  s sont C, C++ et java avec une forte pr  f  rence pour C++. Ce fichier sera bien s  r propre et comment  .

Le programme principal appellera successivement les 4 m  thodes ci-dessus pour chercher un motif *P*, lu dans un fichier pass   en ligne de commande, dans un texte *T*,   galement lu dans un fichier pass   en ligne de commande. On veut donc taper dans un terminal : `NomExe fichierMotif fichierTexte`.

Pour chacune des m  thodes vous devrez faire afficher *le nombre de comparaisons de caract  res r  ellement effectu  es*, pour cela on tiendra compte de l   valuation paresseuse des op  rateurs bool  ens.    savoir :

-    l  valuation de `(cond1) ET (cond2)` avec `(cond1)` faux n   value pas `(cond2)`
-    l  valuation de `(cond1) OU (cond2)` avec `(cond1)` vrai n   value pas `(cond2)`

Ajouts sp  cifiques :

1. Pour MP vous afficherez la table *MP_next*
2. Pour KMP vous afficherez la table *KMP_next*
3. Pour Boyer-Moore vous afficherez les tables *Suff*, *D* et *R* (on consid  re pour le moment l  alphabet classique    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 ≤ n - m + 1 faire
    i := 1 ;
    tant que (i ≤ 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] ≠ 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 ≤ n faire
    tant que (i > 0 et P[i] ≠ 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 $P[i] \neq P[j]$) **faire** $j := KMP_next[j];$

 ;

$j := j + 1;$

si ($i = m$ ou $P[i + 1] \neq P[j]$) **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 $j \leq n$ **faire**

tant que ($i > 0$ et $P[i] \neq T[j]$) **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*] ≠ *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* ≤ *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* ≤ *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]]) ;