Chapitre 4

Recherche d'un motif dans un texte

Trouver toutes les occurrences d'un motif dans un texte est un problème qu'on rencontre fréquemment dans les éditeurs de texte. Le plus souvent, le texte est un document en cours d'édition et le motif recherché est un mot particulier fourni par l'utilisateur. Puisqu'on a souvent recours à ce problème, on a recherché des algorithmes efficaces qui peuvent grandement améliorer la réaction de l'éditeur de texte. Ces algorithmes sont également utilisés pour trouver certains motifs dans des séquences d'ADN.

Nous allons voir la formalisation du problème et l'algorithme naïf pour la recherche d'un motif. Nous verrons ensuite l'algorithme le plus efficace en deux temps : d'abord l'idée fondamentale qui consiste à utiliser un automate fini et ensuite l'algorithme dit KMP qui construit l'automate fini à la volée.

4.1 Le problème

4.1.1 Notion de mot

Introduction.- On connaît intuitivement ce qu'est un mot. On sait qu'il existe d'autres alphabets que l'alphabet français. L'aspect abstrait de ceci se résume dans les deux définitions suivantes.

DÉFINITION 1.- *On appelle* **alphabet** *(*alphabet *également en anglais) tout ensemble fini non vide. Ses éléments sont appelés des* **lettres** *(*letter *en anglais).*

Exemple.- L'alphabet latin est l'ensemble suivant de vingt-six lettres :

{'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'}.

On remarquera que, pour faire la distinction entre la constante 'x' et la variable x, les lettres sont entourées par des apostrophes.

On remarquera aussi que l'on n'a fait pas la distinction entre les minuscules et les majuscules alors que ceci est indispensable en informatique (les codes ne sont pas les mêmes).

L'alphabet français comprend des *éléments diacritiques* (les accents, la cédille...).

DÉFINITION 2.- *On appelle* **mot** *(*word *en anglais) sur un alphabet* **A** *toute suite finie d'éléments de* **A**. *Soit* $w = (a_i)_{1 \leq i \leq n}$ *un tel mot, on le note souvent :*

$$w = a_1 a_2 ... a_n .$$

L'entier naturel n *s'appelle la* **longueur** *(*length *en anglais) du mot. Lorsque* $n = 0$ *on a le* **mot vide** λ *(*empty word *en anglais).*

Remarque.- Un mot en ce sens (dénomination maintenant traditionnelle en informatique) est plutôt un *texte* au sens de la langue vernaculaire. La notion de *mot*, séparé par des espaces, n'est pas utile en informatique. Il faut d'ailleurs prévoir explicitement le caractère **espace** dans l'alphabet pour pouvoir séparer certaines parties du texte d'autres si on le désire.

Pour ne pas faire collision avec le langage vernaculaire, on parle quelquefois de **caractère** (*character* en anglais) au lieu de lettre et de **chaîne de caractères** (*string* en anglais) au lieu de mot.

4.1.2 Recherche de motif

DÉFINITION 3.- *Soient $t = t_1 \dots t_n$ et $p = p_1 \dots p_m$ deux mots sur le même alphabet A, avec $m \leq n$. On dit que le* **motif** *(*pattern *en anglais) p apparaît dans le* **texte** *(*text *en anglais) t s'il existe $i \in [1, n-m+1]$ tel que*:

$$t_i = p_1, t_{i+1} = p_2, \dots, t_{i+m-1} = p_m.$$

Exemple.- Si $A = \{\text{a, b}\}$, $t =$ 'abaababa', $m_1 =$ 'ab' et $m_2 =$ 'abb' alors le motif m_1 apparaît dans le texte t (et même à trois endroits différents) alors que le motif m_2 n'apparaît pas.

DÉFINITION 4.- *L'indice i de la définition précédente s'appelle le* **décalage** *ou la* **position** *du motif.*

DÉFINITION 5.- *Le problème de la* **recherche d'un motif** *(*pattern matching *en anglais) m dans un texte t consiste à trouver tous les décalages pour lesquels le motif p apparaît dans le texte t.*

Exemple.- Dans l'exemple précédent, le motif m_1 apparaît aux positions 1, 4 et 6 alors que le motif m_2 n'apparaît à aucune position.

4.2 Algorithme naïf

Description.- L'algorithme naïf consiste à tester toutes les positions possibles : pour chaque position i on teste si t_i est égal à p_1, t_{i+1} est égal à p_2 et ainsi de suite jusqu'à t_{i+m-1} égal à p_m. Évidemment on peut arrêter le test dès qu'une lettre ne concorde pas, autrement dit on utilisera une boucle POUR pour décrire les positions possibles et une boucle TANT QUE pour le test.

Procédure.- Écrivons une fonction en langage C qui détermine les décalages, le texte et le motif étant des chaînes de caractères au sens du langage C :

```
void matching(char *t, int n, char *p, int m)
  {
  int i, j;
  int test;
  for (i = 0; i < n-m+1; i++)
    {
    test = 1;
    j = 0;
    while(test == 1 && j < m)
      {
      if (t[i+j] != p[j]) test = 0;
      else j++;
      }
    if (test == 1) printf("Le motif apparait avec un decalage %ld\n",
                          i+1);
    }
  }
```

Complexité.- L'indice i décrit $n - m + 1$ positions et, dans le pire des cas, pour chaque i l'indice j décrit m lettres. La complexité est donc $O((n - m + 1)m)$, soit $O(n^2)$ si on ne retient que la taille du texte et que l'on considère que la taille du motif est proche de celle du texte.

Exercice 1.- Écrire un programme C/C++ qui demande la longueur du motif, la longueur du texte, le motif (sur l'alphabet {a, b, c }) puis qui génère un texte de façon aléatoire, qui affiche les cent premiers caractères de ce texte, qui indique les positions où apparaît le motif et qui, enfin, indique le temps d'exécution en tics d'horloge.

4.3 Recherche de motif au moyen d'automates finis

Introduction.- La complexité $O(n^2)$ est beaucoup trop importante pour un problème utilisé très souvent. L'utilisation d'un automate fini va nous fournir un algorithme en temps $O(n)$, une fois l'automate construit : en effet l'automate examine chaque caratère du texte *une fois et une seule* (au lieu de m fois dans le pire cas pour l'algorithme naîf), en prenant un temps constant pour chaque caractère. Rappelons tout d'abord la définition formelle de ce qu'est un automate fini.

DÉFINITION 6.- *Un* **automate fini** *est un quintuplet* (Q, q_0, F, A, δ) *où :*

- Q *est un ensemble fini, dit des* **états** *;*
- q_0 *est un élément de* Q *(qui est donc non vide), appelé* **état initial** *;*
- F *est un sous-ensemble de* Q*, appelé ensemble des* **états terminaux** *;*
- A *est un alphabet ;*
- δ *est une application de* $Q \times A$ *dans* Q*, appelée* **fonction de transition**.

Rappel.- L'automate fini démarre à l'état q_0 et lit les caractères d'un mot sur l'alphabet A un par un. Lorsque l'automate se trouve dans l'état q et lit le caractère a, il se place dans l'état $\delta(q,a)$. Chaque fois que l'état q appartient à F, on dit que l'automate a **reconnu** la chaîne lue jusqu'à cet endroit.

Exemple.- Considérons le motif *aba* et le texte *aababab*. Recherchons les occurrences du motif dans ce texte :

- La première lettre du texte est également la première lettre du motif. Cela commence bien. Disons qu'on entre dans l'état q_1 de l'automate pour indiquer qu'on a trouvé la première lettre du motif.
- La seconde lettre du texte est un a. Ce n'est pas la seconde lettre du motif donc on n'entre pas dans l'état q_2. Il s'agit de la première lettre du motif, donc on est (on reste) dans l'état q_1.
- La troisième lettre du texte est un b, la seconde lettre du motif. Puisqu'on était dans l'état q_1, on entre dans l'état q_2 pour indiquer qu'on a trouvé les deux premières lettres du motif.
- La quatrième lettre du texte est un a, la première et troisième lettre du motif. Puisqu'on était dans l'état q_2, on passe dans l'état q_3. On a donc

trouvé une occurrence du motif, dont la position est 3 indices avant (la longueur du motif) : le motif apparaît donc à la position 1. On reprend tout depuis le début maintenant. *A priori* on entre dans l'état q_0. En fait on a la première lettre du motif, on entre donc dans l'état q_1.
- La cinquième lettre du texte est b, on entre donc dans l'état q_2.
- La sixième lettre du texte est a, on entre donc dans l'état q_3. On a une seconde occurrence du motif à la position 6 - 3, soit 3. On entre dans l'état q_1.

Principe.- De façon générale, pour le motif $p = p_1 \dots p_m$, considérons l'automate fini dont :

- l'ensemble des états et $Q = \{0, 1, 2, \dots, m\}$;
- l'ensemble initial est 0 ;
- l'ensemble des éléments terminaux est le singleton $F = \{m\}$;
- l'alphabet est l'alphabet A sur lequel sont définis le motif et le texte ;
- la valeur de la fonction de transition $\delta(q, \sigma)$ est définie, pour $q \neq m$, comme la longueur du plus long préfixe de P qui soit aussi suffixe de $p_1 \dots p_q\sigma$. Pour $q = m$, seule $\delta(m, p_m)$ nous intéresse ; il s'agit de la longueur du plus long préfixe de P qui soit aussi suffixe de $p_2 \dots p_m$.

L'algorithme de recherche du motif est alors le suivant :

- On part de l'état initial 0.
- On décrit le texte avec un index i variant de 1 à n :
  - Le nouvel état est $\delta(q, \sigma)$ si σ est la i-ième lettre du texte.
  - Si le nouvel état q est l'état final m alors le motif apparaît à la position $i - m$ et le nouvel état est $\delta(m, p_m)$.

Procédure.- Écrivons une fonction en langage C/C++ qui détermine les positions et le nombre des décalages, le texte et le motif étant des chaînes de caractères au sens du langage C. Nous considérons que l'alphabet est { a, b, c} et que la longueur du motif est 99 au plus. Commençons par une fonction auxiliaire qui dit si une chaîne de caractères est suffixe d'une seconde chaîne de caractères :

```
int suffixe(char *suffix, int ls, char *mot, int lm)
  {
    int test = 1;
    int i = 0;

    while(test && i < ls)
      {
      if (suffix[ls-1-i] != mot[lm-1-i]) test = 0;
      else i++;
      }

    return test;
  }
```

```
void automate(char *t, long n, char *p, int m)
  {
  long i,
       cpt;           // nombre d'occurrences du motif
  int j;
  int delta[100][3];  // tableau de transition

  ////////////////////////////////////////
  // Calcul de la fonction de transition//
  ////////////////////////////////////////

  char s;             // decrit l'alphabet
  int q;              // decrit les etats
  int ls;             // longueur du suffixe
  char *mot;          // decrit les prefixes du motif
  int lm;             // longueur du mot
  int tests;          // test sur le suffixe

  for (q = 0; q < m; q++)
    {

      // definition du mot p_1 p_2 ... p_q a

      lm = q+1;
      mot = new char[lm];
      for (j=0; j < q; j++)
        mot[j] = p[j];
      for (s = 'a'; s <= 'c'; s++)
        {
          mot[q] = s;

          // determination de delta(q,s)

          ls = q+1;
          tests = 0;

          while(!tests)
            {
              if (suffixe(p, ls, mot, lm))
                {
                  tests = 1;
                  delta[q][s - 'a'] = ls;
                }
              else ls--;
            }
         }
    }

  // determination de delta(m, p_m)

  lm = m-1;
  mot = new char[lm];
  for (j=0; j < m-1; j++)
    mot[j] = p[j+1];
```

```
  tests = 0;
  ls = m-1;
  while(!tests)
    {
      if (suffixe(p, ls, mot, lm))
        {
          tests = 1;
          delta[m][p[m-1] - 'a'] = ls;
        }
      else ls--;
    }

  ////////////////////
  // Analyse du texte//
  ////////////////////

  q = 0;
  cpt = 0L;

  for (i = 0L; i < n; i++)
    {
      q = delta[q][t[i]-'a'];
      if (q == m)
        {
//      printf("Le motif apparat avec un decalage %ld\n", i-m+2);
        cpt++;
        q = delta[m][t[i]-'a'];
        }
    }

  printf("\nLe motif est apparu %ld fois.\n", cpt);
  }
```

Complexité.- Comme nous l'avons laissé entendre précédemment, pour l'analyse du texte proprement dit, l'indice i décrit n positions et, pour chaque caractère, une seule comparaison est réalisée. La complexité est donc $O(n)$.

Mais il faut aussi, bien évidemment, tenir compte de la détermination de l'automate, ce qui représente d'ailleurs l'essentiel du programme. Pour déterminer la fonction de transition, l'état décrit les $m+1$ positions et, pour chaque lettre de l'alphabet, on vérifie s'il s'agit d'un suffixe pour, dans le pire cas, $m+1$ valeurs. Pour la détermination du suffixe, là encore, dans le pire cas, on effectue $m+1$ tests. La complexité est donc $O(m^3.|A|)$.

Si m est proche de n, l'algorithme est donc pire que l'algorithme naïf mais, en pratique, n est grand et m petit. L'algorithme est donc plus efficace.

Exercice 2.- Écrire un programme C/C++ qui demande la longueur du motif, la longueur du texte, le motif (sur l'alphabet {a, b, c }) puis qui génère un texte de façon aléatoire, qui affiche les cent premiers caractères de ce texte puis qui, pour la méthode naïve d'abord et pour la méthode avec automate ensuite, indique le nombre d'occurrences du motif dans le texte et le temps d'exécution en tics d'horloge.

Corrigé des exercices

Exercice 1.- Rappelons tout d'abord que la constante entière :

`RAND_MAX`

est la valeur maximum renvoyé par la fonction :

`int rand(void)`

qui renvoie un entier pseudo-aléatoire de l'intervalle [0, RAND_MAX]. Toutes les deux sont déclarées dans le fichier en-tête :

`stlib.h`

Rappelons ensuite que la fonction :

`clock_t clock(void)`

permet de déterminer le temps machine utilisé, le type `clock_t` n'étant rien d'autre qu'un entier. Ces deux entités sont déclarées dans le fichier en-tête :

`time.h`

Ici, pour pouvoir effectuer des tests sur de longs textes, on utilisera des entiers de type `long`. Ceci nous conduit par exemple au programme suivant :

```
// test.c
// Programme pour tester les algorithmes
// sur la recherche de motifs
//
// pour compiler : g++ -lg++ test.c

#include <stdio.h>
#include <stdlib.h>    // pour rand() et RAND_MAX
#include <time.h>      // pour clock() et clock_t

void matching(char *t, long n, char *p, int m)
  {
  long i,
       cpt; // nombre d'occurrences du motif
  int j;
  int test;
  cpt = 0L;
  for (i = 0L; i < n-m+1; i++)
    {
    test = 1;
    j = 0;
    while(test == 1 && j < m)
      {
      if (t[i+j] != p[j]) test = 0;
      else j++;
      }
    if (test)
      {
      printf("Le motif apparait avec un decalage %ld\n", i+1);
      cpt++;
      }
```

```
    }
  printf("\nLe motif est apparu %ld fois.\n", cpt);
  }

int main()
     {
      //declarations

      int m,              // longueur du motif
          j;              // indice dcrivant le motif
      long n,             // longueur du texte
           i;             // indice dcrivant le texte
      char * motif;       // le motif
      char * texte;       // texte
      clock_t t0, t1, t2;


      // Saisie des parametres

      printf("Longueur du motif : ");
      scanf("%d", &m);
      printf("Longueur du texte : ");
      scanf("%ld", &n);

      // Saisie du motif

      motif = new char[m];
      printf("Donnez le motif (alphabet 'a', 'b', 'c') : ");
      scanf("%s", motif);

      // Cration aleatoire du texte

      texte = new char[n+1];

      for (i=0L; i<n; i++)
        switch(rand()/(RAND_MAX/3))
          {
          case 0 : texte[i] = 'a'; break;
          case 1 : texte[i] = 'b'; break;
          case 2 : texte[i] = 'c'; break;
          }

      texte[n] = '\0';

      printf("\nVoici le debut du texte :\n");
      for (i = 0L; (i < 100L) && (i < n) ; i++)
          {
           putchar(texte[i]);
           if (i%30 == 29L) {printf("\n");}
          }
```

```
      // Methode naive

      printf("\nDebut de la methode naive\n\n");
      t0 = clock();
      matching(texte, n, motif, m);
      t1 = clock();
      printf("Temps passe : %lu.\n", t1-t0);

      // Liberation des tableaux

      delete [] motif;
      delete [] texte;

     return 0;
     }
```

Exercice 2.- Il suffit de reprendre une grande partie de l'exercice 1 et d'y incorporer le programme que nous avons vu à propos de la méthode avec automate :

```
// test2.c
// Programme pour tester les algorithmes
// sur la recherche de motifs
//
// pour compiler : g++ -lg++ test.c

#include <stdio.h>
#include <stdlib.h> // pour rand() et RAND_MAX
#include <time.h>   // pour clock() et clock_t

////////////////////
// Algorithme naif//
////////////////////

void matching(char *t, long n, char *p, int m);

/////////////////////////////
// Algorithme avec automate//
/////////////////////////////

int suffixe(char *suffix, int ls, char *mot, int lm);

void automate(char *t, long n, char *p, int m);

//////////////////////
//Programme principal//
//////////////////////

int main()
     {
```

```
//declarations

int m,              // longueur du motif
    j;              // indice dcrivant le motif
long n,             // longueur du texte
     i;             // indice dcrivant le texte
char * motif;       // le motif
char * texte;       // texte
clock_t t0, t1, t2;

// Saisie des parametres

printf("Longueur du motif : ");
scanf("%d", &m);
printf("Longueur du texte : ");
scanf("%ld", &n);

// Saisie du motif

motif = new char[m];
printf("Donnez le motif (alphabet 'a', 'b', 'c') : ");
scanf("%s", motif);

// Creation aleatoire du texte

texte = new char[n+1];

for (i=0L; i<n; i++)
  switch(rand()/(RAND_MAX/3))
    {
    case 0 : texte[i] = 'a'; break;
    case 1 : texte[i] = 'b'; break;
    case 2 : texte[i] = 'c'; break;
    }

texte[n] = '\0';

printf("\nVoici le debut du texte :\n");
for (i = 0L; (i < 100L) && (i < n) ; i++)
    {
     putchar(texte[i]);
     if (i%30 == 29L) {printf("\n");}
    }

// Methode naive

printf("\nDebut de la methode naive\n\n");
t0 = clock();
matching(texte, n, motif, m);
t1 = clock();
```

```
 printf("Temps passe : %lu.\n", t1-t0);

 // Methode avec automate

 printf("\nDebut de la methode avec automate\n\n");
 t0 = clock();
 automate(texte, n, motif, m);
 t1 = clock();
 printf("Temps passe : %lu.\n", t1-t0);

 // Liberation des tableaux

 delete [] motif;
 delete [] texte;

return 0;
}
```

4.4 L'algorithme KMP

Introduction.- L'idée de l'algoritme de Knuth, Morris et Pratt est celle de l'algorithme avec automate mais en calculant la fonction de transition *à la volée*.

Principe.- L'idée principale est d'utiliser un tableau des prédécesseurs de dimension la longueur du motif. Si on se trouve dans l'état q (qui indique, comme dans le cas de l'algorithme précédent, la longueur du préfixe du motif que l'on a rencontré jusqu'à maintenant) : si la lettre que l'on lit est la $q+1$-ième lettre du motif, on entre dans l'état $q+1$; si ce n'est pas le cas, le tableau des prédécesseurs donne à l'index q le nouvel état, c'est-à-dire la longueur du plus long préfixe du motif qui est un suffixe propre de $p_1 \dots p_q$.

Exemple.- Si le motif *abababacca*, le tableau des prédécesseurs est :

0	1	2	3	4	5	6	7	8	9	10
	a	b	a	b	a	b	a	b	c	a
0	0	0	1	2	3	4	5	6	0	1

Le tableau proprement dit est constitué de la troisième ligne seulement. Nous avons ici rappelé ici notre façon d'indexer le tableau ainsi que le motif. Nous avons également ajouté une colonne d'index 0 pour rappeler l'indexation en langage C.

Index