Projet de mathématiques : Automate déterminsite et non déterministe

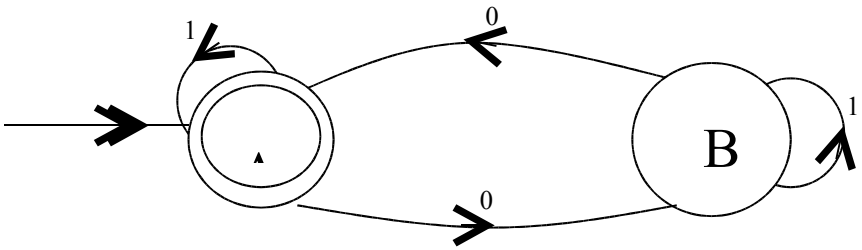
1 - Automate déterministe :

1.1 Présentation de l'automate

On choisit d'implémenter un automate qui est capable de déterminer à partir d'un mot composé de 0 et de 1 si ce mot contient un **nombre pair de 0**.

Graphe :

$\Sigma = \{0,1\}$
$Q = \{A,B\}$
$q_0 = A$
$F = \{A\}$



A chaque état, on applique une transition. On choisit de représenter ceci par un tableau à 2 dimensions
1ère entrée : état : A est codé par 0 et B par 1
2ème entrée : transition (0 ou 1)

		Transitions	
		0	1
Etats	A : 0	Impair	Pair
	B :1	Pair	Impair

1.2 Source

```
#include <string.h>
#include <stdio.h>

#define NBCARMAX       10
#define MESSAGE_PAIR   "Le mot saisi contient un nombre pair de 0"
#define MESSAGE_IMPAIR "Le mot saisi contient un nombre impair de 0"

void main ()
{
      int  tab[2][2];          //Tableau d'implémentation de l'automate
      int  etat = 0;           //Etat courant
      int  transition;         //Transition à appliquer à l'état

      char chaine_lu[NBCARMAX];     //Chaine de caractères saisie
```

```
      int  lg = 0;                  //Longueur de la chaîne lue
      int  i;                       //Indice de tableau

      printf("-= Simulation Automate deterministe 1  =-\n\n\n\n");

      //Lecture du mot composé de 0 et de 1
      printf("Saisissez un mot de 0 et de 1 :");
      scanf("%s", chaine_lu);
      lg = strlen ( chaine_lu );
      while (lg > NBCARMAX)
      {
            printf("\nErreur de saisie : le mot doit etre composé de moins de %d
caracteres\nRessaisir le mot : ",NBCARMAX);
            scanf("%s", chaine_lu);
            lg = strlen ( chaine_lu );
      }

      // Codage de l'automate
      tab[0][0] = 1;  //impair
      tab[1][0] = 0;  //pair
      tab[0][1] = 0;
      tab[1][1] = 1;

      // parcours du tableau --> simulation de l'automate
      for ( i = 0; i < lg; i++)
      {
            if (chaine_lu[i]=='0')
                  transition=0;
            if (chaine_lu[i]=='1')
                  transition=1;

            //on passe a l'etat suivant
            etat = tab[etat][transition];

            //Trace à l'ecran
            if ( etat == 1 ) printf("  nombre de 0 impair \n");
            if ( etat == 0 ) printf("  nombre de 0 pair \n");
      }

      //Affichage du résultat :pair ou impair
      if ( etat == 0)
            printf("\n\n%s\n",MESSAGE_PAIR);
      else
            printf("\n\n%s\n",MESSAGE_IMPAIR);
}
```

1.3 TRACE :

CAS 1 :

```
Saisissez un mot de 0 et de 1 :000000000000
  nombre de 0 impair
  nombre de 0 pair
  nombre de 0 impair
  nombre de 0 pair
  nombre de 0 impair
  nombre de 0 pair
  nombre de 0 impair
  nombre de 0 pair
  nombre de 0 impair
  nombre de 0 pair
  nombre de 0 impair
  nombre de 0 pair
```

```
Le mot saisi contient un nombre pair de 0
```

Cas 2 :

```
Saisissez un mot de 0 et de 1 :111111111
  nombre de 0 pair
  nombre de 0 pair
  nombre de 0 pair
  nombre de 0 pair
  nombre de 0 pair
  nombre de 0 pair
  nombre de 0 pair
  nombre de 0 pair
  nombre de 0 pair

Le mot saisi contient un nombre pair de 0
```

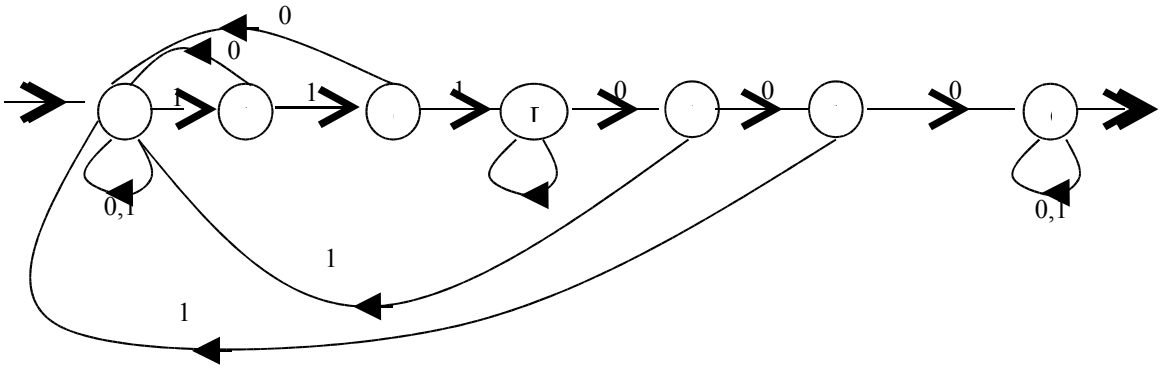
Cas 3 :

```
Saisissez un mot de 0 et de 1 :110010
  nombre de 0 pair
  nombre de 0 pair
  nombre de 0 impair
  nombre de 0 pair
  nombre de 0 pair
  nombre de 0 impair

Le mot saisi contient un nombre impair de 0
```

2 - Automate non déterministe :

2.1 Présentation de l'automate

Automate non déterministe :
En appliquant une transition, on ne tombe pas forcément sur un état unique mais sur un ensemble d'états. La difficulté est donc d'implémenter cela. Pour l'illustrer, on choisit d'implémenter un automate qui est capable de reconnaître la chaîne 111000. La difficulté est de parcourir exhaustivement tous les parcours possibles. Pour cela, on va transformer cet automate non déterministe en un automate déterministe dont la matrice est donnée ci-dessous.



		Transitions	
		0	1
Etats	A	A	A,B
	B	A	C
	C	A	D
	D	E	D
	E	F	A
	F	G	A
	G	G	G

Le programme affiche la liste des états accédés et si la chaîne 111000 a été reconnue ou non dans le mot tapé en entrée.

2.2 Source

```
#include <string.h>
#include <stdio.h>
#include <conio.h>


//Variables globales
#define LG_TAB 50

char tab_etat   [LG_TAB][3];  //Tableau des etats
char traitement0[LG_TAB];
char traitement1[LG_TAB];
int indice1;
int indice2;

//*********************************************************
void etat_suivant1 ( char etat )
{
  int i;

  for ( i=0;i<=LG_TAB;i++)
  {
        if (tab_etat[i][0] == etat && tab_etat[i][1] == '0')
        {
              traitement1[indice2] = tab_etat[i][2];
              indice2 ++;
        }
  }
}

//*********************************************************
void etat_suivant2 ( char etat )
{
  int i;

  for ( i=0;i<=LG_TAB;i++)
  {
        if (tab_etat[i][0] == etat && tab_etat[i][1] == '1')
        {
```

```
                traitement1[indice2] = tab_etat[i][2];
                indice2 ++;
          }
    }
}



void main ( )
{
      char chaine [50];
      char etat_final;
      char etat_courant;

    int i, j, lg;

      printf("-= Simulation Automate non deterministe  =-\n");
      printf("      reconnaissance du mot 111000\n\n\n\n");
      printf("Saisissez un mot de 0 et de 1 :");

      indice1 = 0;
      indice2 = 0;

      //Definition de l'automate
      tab_etat[0][0] = 'a';
      tab_etat[0][1] = '0';
      tab_etat[0][2] = 'a';

      tab_etat[1][0] = 'a';
      tab_etat[1][1] = '1';
      tab_etat[1][2] = 'a';

      tab_etat[2][0] = 'a';
      tab_etat[2][1] = '1';
      tab_etat[2][2] = 'b';

      tab_etat[3][0] = 'b';
      tab_etat[3][1] = '1';
      tab_etat[3][2] = 'c';

      tab_etat[4][0] = 'c';
      tab_etat[4][1] = '1';
      tab_etat[4][2] = 'd';

      tab_etat[5][0] = 'd';
      tab_etat[5][1] = '0';
      tab_etat[5][2] = 'e';

      tab_etat[6][0] = 'e';
      tab_etat[6][1] = '0';
      tab_etat[6][2] = 'f';

      tab_etat[7][0] = 'f';
      tab_etat[7][1] = '0';
      tab_etat[7][2] = 'g';

      tab_etat[8][0] = 'g';
      tab_etat[8][1] = '0';
      tab_etat[8][2] = 'g';

      tab_etat[9][0] = 'g';
      tab_etat[9][1] = '1';
      tab_etat[9][2] = 'g';

      etat_courant = 'a';
      etat_final = 'g';
```

```
      traitement0[0] = etat_courant;
      indice1 = 1;

      fflush(stdin);
      //Lecture de la chaine d'entree
      scanf("%s",chaine);
      lg = strlen(chaine);

   // parcours du tableau --> simulation de l'automate
      printf("Liste des etats accedes\n");
      for(i=0;i<lg;i++)
      {
            if (chaine[i]=='0')
            {
                  for(j=0;j<indice1;j++)
                        etat_suivant1(traitement0[j]);
            }

            if (chaine[i]=='1')
            {
                  for(j=0;j<indice1;j++)
                        etat_suivant2 ( traitement0[j] );
            }

            for (j=0;j<indice2;j++)
            {
                  traitement0[j]=traitement1[j];

            }
            printf(" %c,", traitement1[indice2-1]);
            indice1 = indice2;
            indice2 = 0;
      }
      indice2 = 0;


      for (i=0;(i<indice1)&&(indice2==0);i++)
      {
            if (traitement0[i] == etat_final)
                  indice2 = 1;
      }

      // Affichage du résultat
      if (indice2 == 1)
            printf ("\n\n   Chaine reconnue\n\n\n");
      else
            printf("\n\n    Chaine non reconnue\n\n");
}
```

2.3 TRACE :

Cas 1 : paquets de 0 et de 1 avant la séquence 111000

```
-= Simulation Automate non deterministe  =-
      reconnaissance du mot 111000


Saisissez un mot de 0 et de 1 :1100111000
Liste des etats accedes
 b, c, a, a, b, c, d, e, f, g,

   Chaine reconnue
```

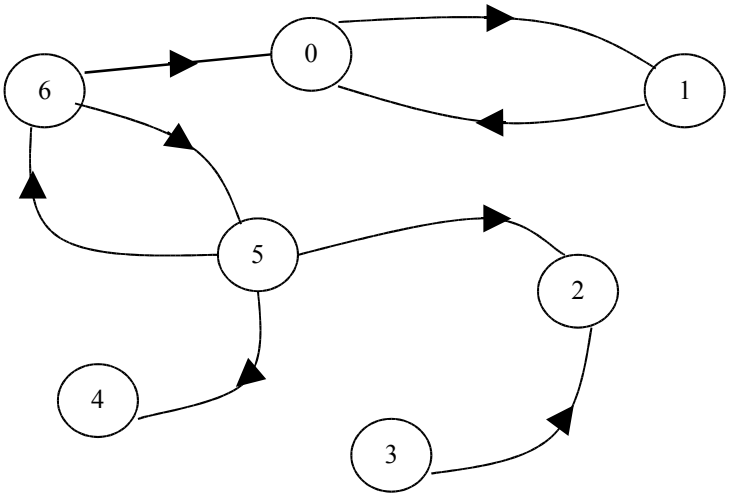
Cas 2 : paquets de 0 et de 1 après la séquence 111000

```
-= Simulation Automate non deterministe  =-
      reconnaissance du mot 111000


Saisissez un mot de 0 et de 1 :111000100100
Liste des etats accedes
 b, c, d, e, f, g, g, g, g, g, g, g,

   Chaine reconnue
```

3 – Parcours d'un graphe

2.1 Présentation de l'automate

Le graphe que l'on a choisit d'implémenter est le suivant :



Indice	0	1	2	3	4	5	6
contenu			5	2	5	6	-1

2.2 Source

```
//Ecrire le programme qui parcours le graphe a partir d'un etat donné
//et affiche la liste des sommets pouvant etre atteint

#include <string.h>
#include <stdio.h>
#include <conio.h>

#define NB_ARC 8
#define NB_SOMMET 7

int arc   [NB_ARC][2];  //Tableau des arcs
int bcktrack[NB_SOMMET];//Tableau des sommets atteints
```

```
//*********************************************************
void main ( )
{
      // Déclaration des variables
      int cpt_cours, cpt_suivants, i, j, depart;
      int sommet_cours[NB_SOMMET];
      int sommet_suivants[NB_SOMMET];

      printf("-= Simulation d'un graphe  =-\n");
      printf("     \n\n");
      printf("Saisissez un sommet:");

      //Definition de l'automate
      arc[0][0] = 0;
      arc[0][1] = 1;

      arc[2][0] = 0;
      arc[2][1] = 6;

      arc[1][0] = 1;
      arc[1][1] = 0;

      arc[3][0] = 5;
      arc[3][1] = 6;

      arc[4][0] = 5;
      arc[4][1] = 4;

      arc[5][0] = 2;
      arc[5][1] = 3;

      arc[6][0] = 6;
      arc[6][1] = 5;

      arc[7][0] = 5;
      arc[7][1] = 2;

      fflush(stdin);
      // Lecture de la chaine d'entree
      cpt_cours = 1; // nombre de sommets accessibles
      scanf("%d",&sommet_cours[0]);

      // Initialisation du tableau de back tracking
      bcktrack[sommet_cours[0]] = -1;

      // Remplissage du back track tant que des sommets peuvent être atteints
      while (cpt_cours > 0)
      {
            // remplissage du backtrack à partir des sommets en cours
            for (i=0; i<cpt_cours; i++)
            {
                  for (j=0;j<NB_ARC;j++)
                  {
                        if ( arc[j][0] == sommet_cours[i] && bcktrack[arc[j][1]]
== NULL)
                        {
                              bcktrack[arc[j][1]] = arc[j][0];
                        }
                  }
            }
            // mémorisation des sommets pouvant être atteints
            cpt_suivants = 0;
            for (i=0; i<cpt_cours; i++)
            {
                  for (j=0; j<NB_SOMMET; j++)
```

```
                  {
                        if (bcktrack[j] == sommet_cours[i])
                        {
                              sommet_suivants[cpt_suivants] = j;
                              cpt_suivants++;
                        }
                  }
            }
            // initialisation des sommets en cours
            cpt_cours = cpt_suivants;
            for (i=0; i<cpt_cours; i++)
            {
                  sommet_cours[i]=sommet_suivants[i];
            }
      }

      // affichage du backtrack
      for (i=0; i<NB_SOMMET; i++)
      {
            printf("| %d |",bcktrack[i]);
      }

      // affichage des sommets parcourus
      printf ("\nSommets parcourus \n");
      for (i=0; i<NB_SOMMET; i++)
      {
            if (bcktrack[i] != 0)
                  printf("| %d |",i);
      }

      // affichage d'un chemin
      printf ("\nChemin le plus court \n");
      depart = -1;
    j=NB_SOMMET;
      while (j>0)
      {
            for(j=NB_SOMMET;j>0;j--)
            {
                  if (bcktrack[j] == depart)
                  {
                        depart = j;
                        printf("%d",depart);
                  }
            }
      }
}
```

3.3 TRACE :

```
-= Simulation d'un graphe  =-


Saisissez un sommet:6
| 0 || 0 || 5 || 2 || 5 || 6 || -1 |
Sommets parcourus
| 2 || 3 || 4 || 5 || 6 |
Chemin le plus court
654
```