LENGUAJES DE COMPUTACIÓN

Tarea No.: Mini Proyecto 1	Fecha: 28/02/2021
Alumno: Lilia Arceli Lobato Martínez	Id alumno: IE706937
Palabras clave: AFD, Mini Proyecto	

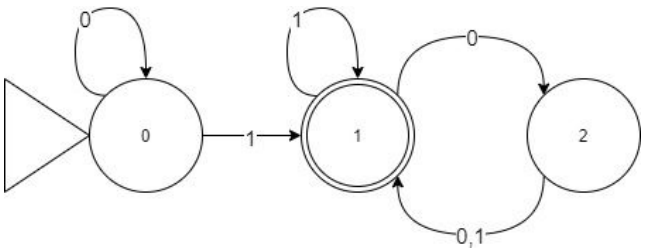
Objetivo general.

Simular un autómata finito determinista(AFD).
Entrada: archivo con la especificación de la cadena a procesar y la entrada

Cadena de entrada
Alfabeto: símbolos separados por punto y coma
Estado inicial
Estados finales
Matriz con la función de transiciones

Salida: mostrar la secuencia de estados visitados para procesar la cadena e indicar si es aceptada o no por el autómata.

Ejemplo para el autómata M4.



L(M4)={w|Conjunto de palabras que tienen al menos un 1 y un número par de 0s siguen al último 1}
Σ={0,1}

Entrada:

0010100
0;1;
0
1;
0;1;
2;1;
1;1;

Salida:

Aceptada
Secuencia de estados: 0/0/0/1/2/1/2/1/

Descripción algoritmo.

Describir de forma general cómo funciona su algoritmo, cómo se implementó.
Tenemos la opción de subir un archivo o describirlo manualmente, independiente de la forma que elijamos para obtener los datos, el algoritmo es el mismo.
El código se puede dividir en 2 partes: Obtener la información necesaria del AFD y su análisis respecto a la entrada.

Para obtener la información creamos estas funciones:

- Obtenemos la cadena a evaluar y la guardamos en una cadena.

```
//Recolectamos el caso a evaluar
char testCase[65]={};
printf("Cadena a Evaluar: ");
fgets(testCase, 64, stdin);
```

- Obtenemos el alfabeto con la función "alfabeto", dentro de esta función revisamos cómo está construida, eliminamos los nulos y los saltos de línea de la cadena obtenida para limpiarla y poder utilizarla.

```
//Obtenemos el alfabeto
char alfSize=alfabeto(letters);
```

```
int alfabeto(int matrix[2][32]){
    for(int iter=0; iter<32; iter++){
        matrix[0][iter]=iter;
    }

    char dirty[65];
    printf("Alfabeto: ");
    fgets(dirty, 64, stdin);
    int iter;
    for(iter=0; iter<65; iter+=2){
        if(dirty[iter]=='\0'||dirty[iter]=='\n')break;
        matrix[1][iter/2]=dirty[iter];
    }
    return iter/2;

}
```

- Decidimos cual es el nodo inicial y lo convertimos en una cadena.

```
//Decidimos cual es el nodo inicial
printf("Nodo incial: ");
fgets(number, 5, stdin);
node=atoi(number);
```

- Decidimos cuáles nodos se van a detectar como finales y lo convertimos en una cadena tras limpiarla.

```
//Recibimos los nodos finales
finalNode(automat);
```

```
void finalNode(char aut[64][64]){
    char dirty[129];
    printf("Nodo final: ");
    fgets(dirty, 128, stdin);
    int iter;
    for(iter=0; iter<129; iter+=2){
        if(dirty[iter]=='\0'||dirty[iter]=='\n')break;
        aut[atoi(&dirty[iter])][63]=1;
    }
}
```

- Creamos una tabla de transiciones por cada nodo. Es importante dejar la cadena con un formato limpio ya que es esta la que vamos a estar utilizando para saltar de un estado a otro al probar cualquier cadena.

```
//Recibimos las transiciones de cada nodo
printf("\t\t");
int num_cols = sizeof(letters[0]) / sizeof(letters[0][0]);
for (int alf_i = 0; alf_i <= (num_cols)-1; alf_i++) {
    if(letters[1][alf_i] != NULL){printf("%c;",letters[1][alf_i]);}
}
printf("\n-----------------------\n");
states(automat);
```

```
void states(char aut[64][64]){
    char dirty[129];

    for(int node=0; node<64; node++){
        memset(dirty,'\0',129);

        printf("Estado %d:\t", node);
        fgets(dirty, 128, stdin);

        if(dirty[0]=='\n')break;
        int iter;

        for(iter=0; iter<129; iter+=2){
            if(dirty[iter]=='\0')break;
            aut[node][iter/2]=atoi(&dirty[iter]);

        }
    }
}
```

Cuando obtenemos toda la información, podemos comenzar a probar la cadena inicial en el AFD definido.

- Para esto creamos una función llamada “nodeTransition” donde, dependiendo del estado en que se está, se avanza al siguiente estado.

```
int nodeTransition(int matrix[2][32], char key){
    for(int a=0; a<32; a++){
        if(matrix[1][a]==key){
            return matrix[0][a];
        }
    }
    return -1;
}
```

- Se llama la función “nodeTransition” por tantos caracteres tenga la cadena a evaluar y se va creando un array donde se tiene el número del nodo. Al terminarlos, se busca si la cadena es válida (si el nodo existe como nodo final) y así es como se identifica si la cadena es válida

```
snprintf(cadena, sizeof(cadena)-1, "%d",node);
for(int c=0; c<strlen(testCase)-1; c++){
    node=automat[node][nodeTransition(letters, testCase[c])];
    helper=strdup(cadena);
    snprintf(cadena, sizeof(cadena)-1, "%s/%d", helper, node);
    free(helper);
}

if(automat[node][64]>0){
    printf("Aceptada\n");
}else{
    printf("rechazada\n");
}

printf("%s\n", cadena);
```

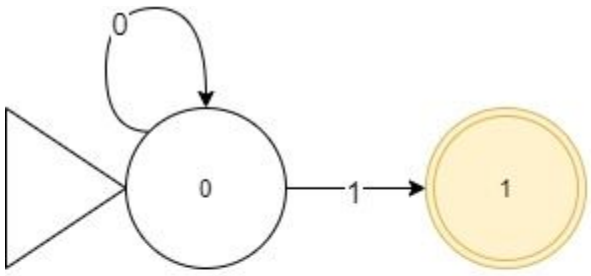
Ejemplos

Para demostrar el funcionamiento del AFD, generamos las corridas de algunos ejemplos y los comparamos con diagramas:

Con el autómata F4

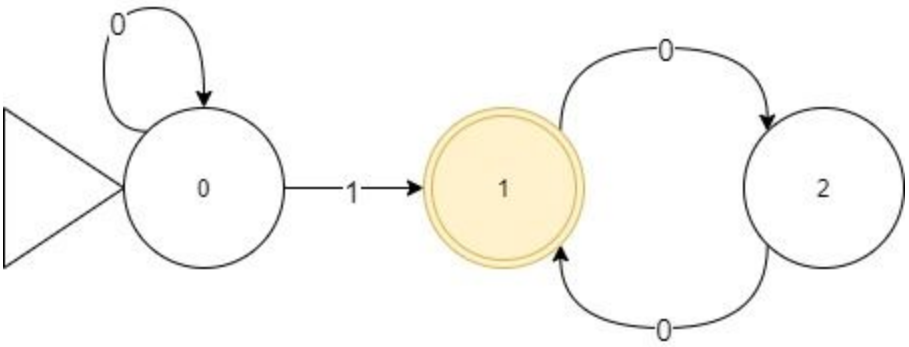
1. **00001**

```
AFD Simulation

------------------------
1. Upload a .txt file
2. Manually Upload each description
ANY OTHER KEY = Exit
1
Se va a avaluar la cadena asignada al inicio del documento
Para separar utilice ; y evite el uso de espacios
File name: new.txt
Cadena a Evaluar: 00001
Alfabeto: 0;1;
Nodo incial: 0
Nodo final: 1;

------------------------
;               0;1;
------------------------
Estado 0:       0;1;
Estado 1:       2;1;
Estado 2:       1;1;
ACEPTADA
0/0/0/0/0/1
'
```



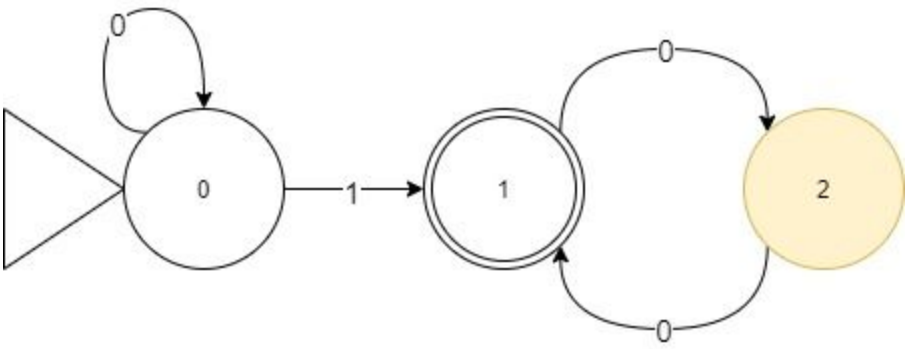
2. **0100**

```
AFD Simulation
-----------------------
1. Upload a .txt file
2. Manually Upload each description
ANY OTHER KEY = Exit
1
Se va a avaluar la cadena asignada al inicio del documento
Para separar utilice ; y evite el uso de espacios
File name: new.txt
Cadena a Evaluar: 0100
Alfabeto: 0;1;
Nodo incial: 0
Nodo final: 1;

------------------------
;               0;1;
------------------------
Estado 0:       0;1;
Estado 1:       2;1;
Estado 2:       1;1;
ACEPTADA
0/0/1/2/1
```



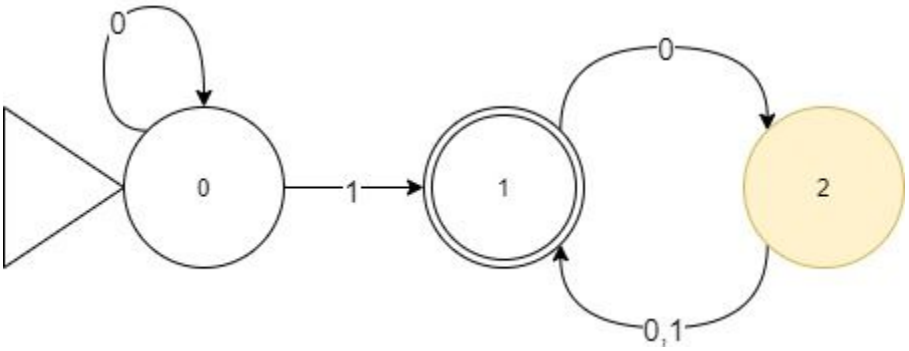
3. **00100000**

```
AFD Simulation
------------------------
1. Upload a .txt file
2. Manually Upload each description
ANY OTHER KEY = Exit
1
Se va a avaluar la cadena asignada al inicio del documento
Para separar utilice ; y evite el uso de espacios
File name: new.txt
Cadena a Evaluar: 00100000
Alfabeto: 0;1;
Nodo incial: 0
Nodo final: 1;

------------------------
;               0;1;
------------------------
Estado 0:       0;1;
Estado 1:       2;1;
Estado 2:       1;1;
RECHAZADA
0/0/0/1/2/1/2/1/2
```



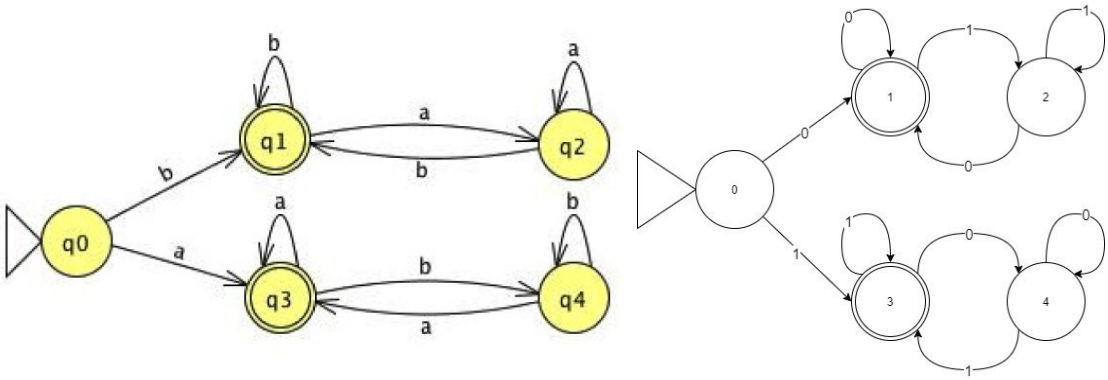
4. 010101010

```
AFD Simulation
-----------------------
1. Upload a .txt file
2. Manually Upload each description
ANY OTHER KEY = Exit
1
Se va a avaluar la cadena asignada al inicio del documento
Para separar utilice ; y evite el uso de espacios
File name: new.txt
Cadena a Evaluar: 010101010
Alfabeto: 0;1;
Nodo incial: 0
Nodo final: 1;

-----------------------
;               0;1;
-----------------------
Estado 0:       0;1;
Estado 1:       2;1;
Estado 2:       1;1;
RECHAZADA
0/0/1/2/1/2/1/2/1/2
```



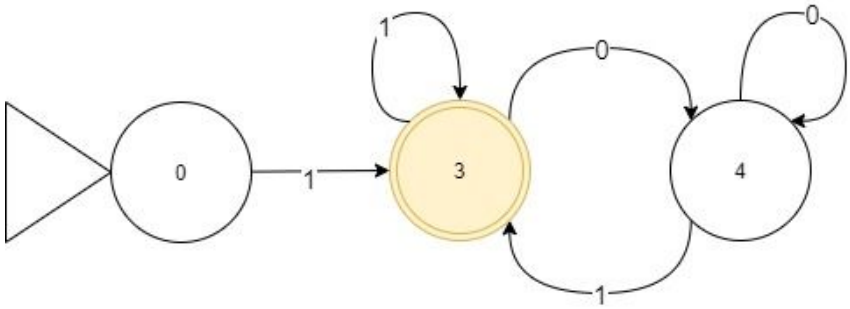
Con el autómata F5

Como restricción, el AFD solamente puede ser descrito por números del 0-9 por lo que vamos a traducir las cadenas a=1 y b=0. Es exactamente lo mismo, el resultado regresará lo mismo que si se trabajara con caracteres.



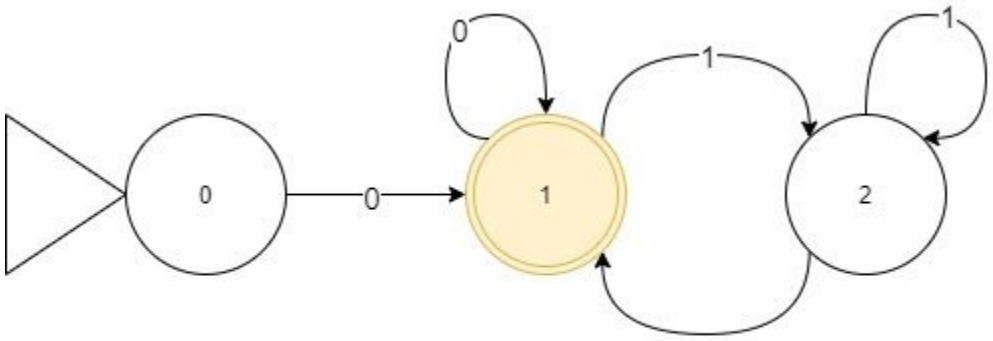
1. **aabba = 11001**

```
AFD Simulation
-----------------------
1. Upload a .txt file
2. Manually Upload each description
ANY OTHER KEY = Exit
1
Se va a avaluar la cadena asignada al inicio del documento
Para separar utilice ; y evite el uso de espacios
File name: new.txt
Cadena a Evaluar: 11001
Alfabeto: 0;1;
Nodo incial: 0
Nodo final: 1;3;

-----------------------
;               0;1;
-----------------------
Estado 0:       1;3;
Estado 1:       1;2;
Estado 2:       1;2;
Estado 3:       4;3;
Estado 4:       4;3;
ACEPTADA
0/3/3/4/4/3
```



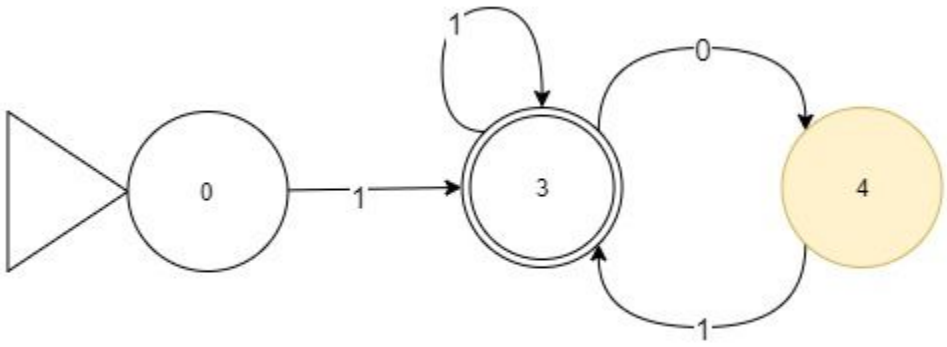
3. **bababab = 0101010**

```
AFD Simulation
-----------------------
1. Upload a .txt file
2. Manually Upload each description
ANY OTHER KEY = Exit
1
Se va a avaluar la cadena asignada al inicio del documento
Para separar utilice ; y evite el uso de espacios
File name: new.txt
Cadena a Evaluar: 0101010
Alfabeto: 0;1;
Nodo incial: 0
Nodo final: 1;3;

-----------------------
;               0;1;
-----------------------
Estado 0:       1;3;
Estado 1:       1;2;
Estado 2:       1;2;
Estado 3:       4;3;
Estado 4:       4;3;
ACEPTADA
0/1/2/1/2/1/2/1
```



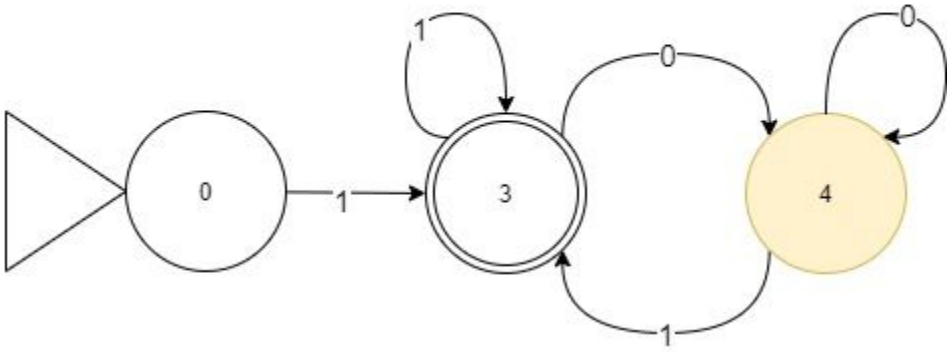
5. **abaaab = 101110**

```
AFD Simulation
-----------------------
1. Upload a .txt file
2. Manually Upload each description
ANY OTHER KEY = Exit
1
Se va a avaluar la cadena asignada al inicio del documento
Para separar utilice ; y evite el uso de espacios
File name: new.txt
Cadena a Evaluar: 101110
Alfabeto: 0;1;
Nodo incial: 0
Nodo final: 1;3;

-----------------------
;               0;1;
-----------------------
Estado 0:       1;3;
Estado 1:       1;2;
Estado 2:       1;2;
Estado 3:       4;3;
Estado 4:       4;3;
RECHAZADA
0/3/4/3/3/3/4
```



7. **aababb = 110100**

```
AFD Simulation
-----------------------
1. Upload a .txt file
2. Manually Upload each description
ANY OTHER KEY = Exit
1
Se va a avaluar la cadena asignada al inicio del documento
Para separar utilice ; y evite el uso de espacios
File name: new.txt
Cadena a Evaluar: 110100
Alfabeto: 0;1;
Nodo incial: 0
Nodo final: 1;3;

------------------------
;               0;1;
------------------------
Estado 0:       1;3;
Estado 1:       1;2;
Estado 2:       1;2;
Estado 3:       4;3;
Estado 4:       4;3;
RECHAZADA
0/3/3/4/3/4/4
```



Código

El código utilizado en esta práctica junto con los archivos de las pruebas se encuentra en el siguiente repositorio: https://github.com/LiliaLobato/LF_Practicas

Conclusiones.

Esta es una de las tareas más interesantes que hemos tenido ya que estamos aplicando nuestra habilidad para aprender a programar junto con el conocimiento teórico de los AFD.
Al realizar el ejemplo, sin crear un AFD variable, el código parecía más una máquina de estados muy muy simple, me pareció muy interesante ver como mis compañeros habían realizado un evaluador de arrays en lugar de una máquina con N estados.

La parte complicada fue dar un formato que pudiera abarcar un AFD general.
Entre las mejoras que le hice al código está poder tomar directamente de una archivo la descripción del AFD.
Aun así, tiene muchas áreas de mejora como que tome también los estados con un nombre distinto a números, tener una interfaz gráfica o que permita tomar nuevas cadenas a evaluar sobre un mismo AFD sin tener que volver a describirlo.