

# REPORTE DE PRÁCTICA AFD

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO  
INSTITUTO DE CIENCIAS BÁSICAS E INGENIERÍA

RESUMEN LENGUAJES FORMALES

ALUMNO: Ordaz Rangel David

DOCENTE: Dr. Eduardo Cornejo Velázquez



# Reporte de Práctica: Implementación de un AFD en C++

Ordaz Rangel David

26 de febrero de 2025

## Introducción

Este documento presenta el análisis y la descripción de un código en C++ que simula el funcionamiento de un Autómata Finito Determinista (AFD). Se detallan la estructura general, las variables y estructuras de datos empleadas, así como el flujo de ejecución del programa.

## Descripción del Código

El programa implementa un AFD utilizando principalmente un `unordered_map` para representar las transiciones entre estados según los símbolos del alfabeto. Los principales componentes del código son:

- **Declaración de Variables:** Se definen variables para el número de estados ( $N$ ), número de símbolos del alfabeto ( $S$ ), número de transiciones ( $D$ ), estado inicial ( $q_0$ ), número de estados finales ( $T$ ) y la cantidad de cadenas a evaluar ( $C$ ).
- **Lectura de Datos:** Se ingresan los parámetros del autómata, el alfabeto, los estados finales y las transiciones entre estados.
- **Procesamiento de Cadenas:** Para cada cadena se simula el recorrido del AFD, verificando si al terminar la cadena el estado alcanzado se encuentra entre los estados finales.

## Estructura de Datos y Flujo de Ejecución

- **Mapa de Transiciones:** Se utiliza un `unordered_map<int, unordered_map<char, int>` para almacenar las transiciones. Cada clave del mapa representa un estado, y su valor es otro mapa que asocia cada carácter a un estado destino. Se inicializa la transición de cada símbolo a  $-1$  para indicar que, por defecto, no existe transición definida.

**Estados Finales:** Se guardan en un vector de enteros los estados que se consideran finales. Al terminar de procesar una cadena, si el estado actual está en este vector, la cadena se acepta.

**Procesamiento de la Cadena:**

- Si la cadena es vacía, se verifica si el estado inicial es final.
- Si la cadena tiene contenido, se recorre carácter a carácter. Para cada carácter se comprueba si existe una transición definida desde el estado actual. Si la transición existe, se actualiza el estado; en caso contrario, se marca la cadena como no aceptada.

## Código Fuente

A continuación, se muestra el código completo:

```

1 #include <bits/stdc++.h> // Librería general
2 using namespace std;
3
4 int main(){
5     // Declaraciones iniciales
6     int N, S, D, q0, T, C, Q;
7     int i, x, z;
8     char caracter, y;
9     string cadena;
10
11     // Ingreso de datos línea a línea
12     cin >> N >> S >> D >> q0 >> T >> C;
13     Q = q0; // Estado inicial
14
15     // Estados finales
16     vector<int> finales(T);
17
18     // Mapa columnas (símbolos) y filas (estados)
19     unordered_map<int, unordered_map<char, int>> alfabeto;
20
21     // Inicialización de transiciones en -1
22     for(i = 1; i <= S; i++){
23         cin >> caracter;
24         alfabeto[i][caracter] = -1;
25     }
26
27     // Registro de estados finales
28     for(i = 0; i < T; i++){
29         cin >> finales[i];
30     }
31
32     // Llenado de mapa con transiciones
33     for(i = 0; i < D; i++){
34         cin >> x >> y >> z;
35         alfabeto[x][y] = z;
36     }
37
38     cin.ignore(); // Ignora salto de línea residual
39
40     // Procesamiento de cada cadena
41     for(i = 0; i < C; i++){
42         getline(cin, cadena);
43         Q = q0; // Reiniciar estado inicial
44         // Caso cadena vacía
45         if (cadena.empty()) {
46             if (find(finales.begin(), finales.end(), Q) != finales.end
47             ())

```

```

47         cout << "ACEPTADA" << '\n';
48     else
49         cout << "RECHAZADA" << '\n';
50 } else {
51     // Procesamiento de cadena no vacía
52     for(int j = 1; j <= cadena.length(); j++){
53         // Verificar existencia de transición
54         if (alfabeto[Q].find(cadena[j-1]) != alfabeto[Q].end()
55             &&
56             alfabeto[Q][cadena[j-1]] != -1)
57             Q = alfabeto[Q][cadena[j-1]];
58         else{
59             Q = -1;
60             break;
61         }
62     }
63     if (Q != -1 && find(finales.begin(), finales.end(), Q) !=
64     finales.end())
65         cout << "ACEPTADA" << '\n';
66     else
67         cout << "RECHAZADA" << '\n';
68 }
69 return 0;
70 }

```

## Análisis del Funcionamiento

### Entrada de Datos

El programa inicia leyendo los siguientes parámetros:

1.  $N$ : Número total de estados (no se utiliza directamente en el código).
2.  $S$ : Número de símbolos en el alfabeto.
3.  $D$ : Número de transiciones definidas.
4.  $q_0$ : Estado inicial del AFD.
5.  $T$ : Número de estados finales.
6.  $C$ : Cantidad de cadenas a evaluar.

Luego, se leen los símbolos del alfabeto y se inicializan las transiciones a  $-1$ . Se registran los estados finales y las transiciones especificadas.

### Procesamiento de Cadenas

Por cada cadena ingresada, el autómata:

- Reinicia el estado actual al estado inicial  $q_0$ .
- Si la cadena es vacía, se verifica directamente si  $q_0$  es un estado final.

- Si la cadena contiene caracteres, se recorre cada uno. Para cada carácter, se comprueba la existencia de una transición definida desde el estado actual. Si la transición existe, se actualiza el estado; de lo contrario, se termina el proceso y se rechaza la cadena.
- Finalmente, si el estado final tras procesar la cadena se encuentra en el vector de estados finales, se acepta la cadena; si no, se rechaza.

## Consideraciones y Posibles Mejoras

- **Robustez:** El código asume que la entrada es correcta. Se podría agregar validación de datos para mayor robustez.
- **Manejo de Transiciones:** El uso de un `unordered_map` es eficiente, pero se debe tener cuidado al verificar la existencia de transiciones y evitar accesos a índices fuera de rango.
- **Optimización:** Se podrían implementar mejoras en la estructura de datos o en la lógica de procesamiento para manejar casos especiales o entradas grandes.

## Conclusión

El código presentado es una implementación sencilla y funcional de un AFD en C++. Permite evaluar múltiples cadenas de entrada y determinar si son aceptadas o rechazadas según la secuencia de transiciones definidas y los estados finales. La estructura del programa y el uso de estructuras de datos dinámicas, como el `unordered_map`, facilitan la simulación del comportamiento de un autómata. Este reporte ha proporcionado una visión general de la implementación, destacando su estructura, funcionamiento y posibles áreas de mejora.