**Dibujo de una persona

Descripción generada automáticamente con confianza bajaINSTITUTO POLITÉCNICO NACIONAL**

**ESCUELA SUPERIOR DE CÓMPUTO**

Práctica 1. Programar un AFD

**PRESENTA**

REYES FRANCISCO PAOLA

5CM4 | COMPILADORES | CECILIA ALBORTANTE MORATO



Introducción.

Teoría general y básica acerca del tema de la práctica.

¿qué es una autómata?

# Desarrollo.

Para el desarrollo de esta práctica, se crearon tres archivos de texto que contenían la información de la tupla que define el autómata que queremos utilizar. Para ello, el archivo contenía la siguiente información:

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

Transiciones

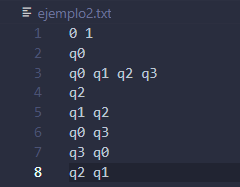
Para las transiciones sólo se consideró el contenido de la tabla de transiciones.

**Ejemplo 1.** *L = {w | w inicia con a}*

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

**Ejemplo 2.** *L = {w | w tiene un número par de ceros y un número impar de unos}*



**Ejemplo 3.** *L = {w | w es un número entero o un número real, ambos con o sin signo}*

Imagen de la pantalla de un computador

Descripción generada automáticamente con confianza baja

Primero, como variables globales se declararon las siguientes.

|  |
| --- |
| vector<vector<string>> transiciones;  vector<char> alfabeto;  vector<string> estados;  string estadoInicial;  vector<string> estadosFinales;  string cadena,nombreArchivo; |

El programa al momento de ejecutarse va a recibir dos argumentos, el primer argumento es el archivo de texto, es decir, el autómata que queremos utilizar y el segundo es la cadena que vamos a probar con dicho autómata.

|  |
| --- |
| int main(int argc, char \*argv[]) {  if (argc > 2) {  nombreArchivo = argv[1];  cadena = argv[2];  } else {  cout << "nombreArchivo.txt cadena" << endl;  }  ifstream archivo(nombreArchivo);  if(!archivo.is\_open()){  cout << "No se pudo abrir el archivo" << endl;  return 1;  } |

Después, se lee cada línea del archivo con nuestra función *getline*, en la primera línea leemos el alfabeto y mandamos a llamar a nuestra función *agregarAlfabeto*, pasa lo mismo para la siguiente línea, leemos el estado inicial y mandamos a una función que agrega el estado. Lo mismo ocurre para agregar el conjunto de estados y los estados finales.

Para las transiciones se ocupó un vector auxiliar donde se fueron añadiendo los datos que el programa iba leyendo en cada línea y después se agregaban a nuestro vector de transiciones. Se hizo una condicional para que no agregara filas vacías, ya que se tenía este problema al principio.

|  |
| --- |
| *// para almacenar cada línea leída del archivo*  string linea;  *// para saber el número de línea actual*  int num\_linea = 1;  *// lee cada línea del archivo*  while(getline(archivo,linea)){  if(num\_linea == 1){ *// alfabeto*  istringstream alfa(linea);  char simbolo;  *// para cada palabra de la línea*  while(alfa >> simbolo) agregarAlfabeto(simbolo);  }else if(num\_linea == 2){ *// estado inicial*  istringstream inicial(linea);  string estado;  while(inicial >> estado) agregarInicial(estado);  }else if(num\_linea == 3){ *// conjunto de estados*  istringstream estado(linea);  string estados;  while(estado >> estados) agregarEstados(estados);  }else if(num\_linea == 4){ *// estados finales*  istringstream final(linea);  string finales;  while(final >> finales) agregarFinales(finales);    }else{ *// transiciones*  vector<string> aux;  istringstream fila(linea);  string casilla;  while(fila >> casilla) aux.push\_back(casilla);  *// para no agregar filas vacías*  if(aux.size() != 0) transiciones.push\_back(aux);  }  num\_linea++;  }  imprimir();  automata();  cout << "\n";  archivo.close();  return 0;  } |

Funciones para agregar la tupla del autómata (excepto las transiciones).

|  |
| --- |
| void agregarAlfabeto(char simbolo){  alfabeto.push\_back(simbolo);  }  void agregarInicial(const string& inicial){  estadoInicial = inicial;  }  void agregarEstados(const string& simbolo){  estados.push\_back(simbolo);  }  void agregarFinales(const string& simbolo){  estadosFinales.push\_back(simbolo);  } |

Posteriormente, se creó una función para evaluar la cadena de entrada, al final de la evaluación se indica si la cadena fue aceptada, si no fue aceptada o si no es válida (cuando tenemos un símbolo que no es parte del alfabeto). Se utilizaron otras dos funciones auxiliares para obtener el índice de la fila y la columna para la tabla de transiciones.

|  |
| --- |
| bool automata(){  *// Primero se busca el índice del estado inicial (fila)*  size\_t estadoActual = getFila(estadoInicial);  *// itera sobre cada símbolo de la cadena de entrada*  for(int i=0; i<cadena.size(); i++){  *// busca el símbolo en la posición i de la cadena a través*  *// de nuestro vector que contiene el alfabeto*  if(find(alfabeto.begin(), alfabeto.end(), cadena.at(i)) != alfabeto.end()){  *// se busca el estado al que transita en la fila de nuestro*  *// estado actual y en la columna en donde se encuentra el*  *// símbolo dentro de nuestro alfabeto, se guarda en una variable*  *// ese estado*  string s = transiciones.at(estadoActual).at(getColumna(cadena.at(i)));  *// se actualiza el valor del índice de la fila de*  *// nuestro estado actual*  estadoActual = getFila(s);  }else{  *// si no encuentra el símbolo dentro del alfabeto*  cout << "\n\033[91mCadena NO valida\033[0m" << endl;  return false;  }  }  *// Una vez que termina de evaluar la cadena, iteramos a lo*  *// largo de nuestro vector de estados finales para*  *// verificar si el estado actual es uno de los estados finales*  for(int i=0; i<estadosFinales.size(); i++){  if(estados.at(estadoActual) == estadosFinales.at(i)) {  cout << "\n\033[92mCadena aceptada\033[0m";  return true;  }  }  *// si no es parte de los estados finales*  cout << "\n\033[91mCadena NO aceptada\033[0m";  return false;  } |

Funciones para obtener el índice de la fila y la columna.

|  |
| --- |
| *// para obtener el índice de la fila*  size\_t getFila(string estadoActual) {  *// busca el estado en el vector*  auto estado = find(estados.begin(), estados.end(), estadoActual);  *// calcula la distancia (el índice)*  *// entre el inicio del vector y el estado*  return distance(estados.begin(), estado);  }  *// para obtener el índice de la columna*  size\_t getColumna(char c) {  auto caracter = find(alfabeto.begin(), alfabeto.end(), c);  return distance(alfabeto.begin(), caracter);  } |

Finalmente, como extra se creó una función que imprime la tupla del autómata, a excepción de las tabla de transiciones.

|  |
| --- |
| void imprimir() {  cout << "\n\033[96mAlfabeto:\033[0m";  for(int i=0; i<alfabeto.size(); i++) {  cout << " " << alfabeto.at(i);  }  cout << "\n\033[93mEstado Inicial:\033[0m " << estadoInicial;  cout << "\n\033[93mEstados:\033[0m";  for(int i=0; i<estados.size(); i++) {  cout << " " << estados.at(i);  }  cout << "\n\033[93mEstados Finales:\033[0m";  for(int i=0; i<estadosFinales.size(); i++) {  cout << " " << estadosFinales.at(i);  }  } |

# Resultados.

**Ejemplo 1.** *Cadenas que inician con a.*

Texto

Descripción generada automáticamente

Como se puede observar, tenemos un ejemplo al inicio de una palabra aceptada, otra no aceptada y la última es para cuando ingresamos una cadena con algún símbolo que no forma parte de nuestro alfabeto.

**Ejemplo 2.** *Cadena con un número par de ceros e impar de unos.*

Texto

Descripción generada automáticamente

En los dos primeros ejemplos se probó con cadenas con un número impar de ceros.

Texto

Descripción generada automáticamente

Otros ejemplos donde se probó también con la cadena *1*, como el número de ceros es par, puede no estar en la cadena.

**Ejemplo 3.** *Cadena con un número entero o un número real, con o sin signo.*

Para esta última prueba se comentó la función imprimir con la finalidad de mostrar más ejemplos.

Texto

Descripción generada automáticamente

En el diseño del AFD se pensó en que, si el usuario ingresaba un número entero con 0 al inicio, se iba a marcar como inválida la cadena, es decir, no se puede poner 019 por ejemplo, solo 19. De igual manera, si se ingresa un 0 y seguido de este un punto para después poner solo ceros (0.0, 0.00, etc.), también te lo marca como inválido, te lo marca como válido si está el 0 solo:

Texto

Descripción generada automáticamente

También se pensó en sí, el usuario ingresa un número decimal como en el ejemplo (0.100290), el autómata lo toma como una cadena no aceptada, pues pasa lo mismo con los enteros, puede agregarse un 0 al inicio (019) y no afecta, pero no es común. En el caso de los decimales se realizó el diseño así para evitar cadenas como 0.10000 que equivale a 0.1.

Sin embargo, esa parte de los decimales puede cambiarse en la tabla de transiciones del autómata.

Texto

Descripción generada automáticamente

Cambiando el diseño del autómata para que acepte las cadenas con decimales que terminan en 0 tenemos:

Texto

Descripción generada automáticamente

Se puede apreciar también, que las cadenas como 0.000 siguen sin ser aceptadas, pues para que sea aceptada tiene que ser sólo 0.

Imagen de la pantalla de un computador

Descripción generada automáticamente con confianza baja