|  |  |
| --- | --- |
| Simulación AFD | Fecha: 30/11/2017 |
| Alumno: David Alfonso Velasco Sedano | Id alumno: 671002 |

## Rúbrica.

Entregar  en un archivo zip :

1. código fuente

2. reporte de tarea

Rúbrica:

40 puntos menos -  Si no anexan el código fuente

40 puntos menos -  Si no anexan reporte de tareas

20 puntos menos -  Si esta incompleto su reporte, recuerden poner referencias y conclusiones.

20 puntos menos -  Si  la simulación  es incompleto o incorrecta su simulación.

## 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.



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/

Ejemplo para el autómata M4.

## Contenido.

Algoritmo:

Describir de forma general como funciona su algoritmo, como se implemento

Es un algoritmo de complejidad temporal lineal O(n). Donde n es igual al largo de la cadena a validar. Aquí vamos validando a que nuevo estado nos mueve las transiciones, si por alguna razón hay una transición inválida, nos quedamos en el estado anterior.

En pseudo-código:

Nodo temporal es igual al nodo inicial

Mientras contador sea menor al largo de la cadena de entrada

Validar que la transición actual sea válida

Si no, continúa

Si es así

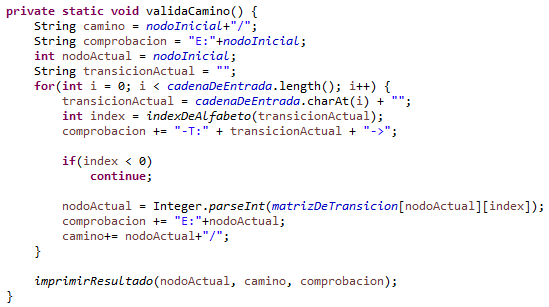
Nodo temporal es igual al nodo que lleva la transición

Nodo temporal es igual al nodo final

Si es así, aceptados la camino

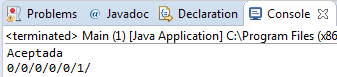
Si no, rechazamos el camino

Código:

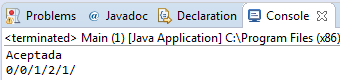


Para el autómata M4 mostrar la salida con impresiones de pantalla para las cadenas:

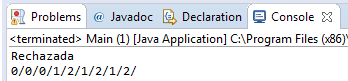
1. **00001**



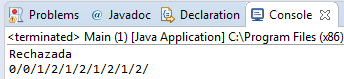
1. **0100**

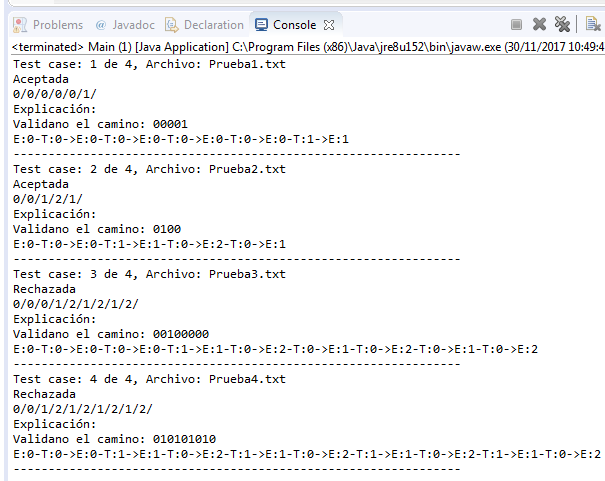


1. **00100000**



1. **010101010**







Define la entrada de tu programa para la M5, con la cadena de entrada **aabba**.

Entrada:

aabba

a;b;

0

1;3;

3;1;

2;1;

2;1;

3;4;

3;4;

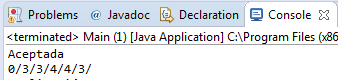
Salida:

Aceptada

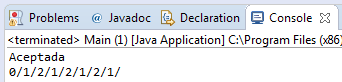
Secuencia de estados: 0/3/3/4/4/3/

Para el autómata M5 probar con las cadenas:

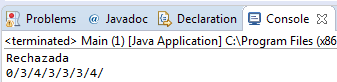
1. **aabba**



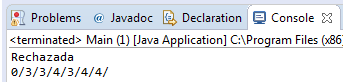
1. **bababab**

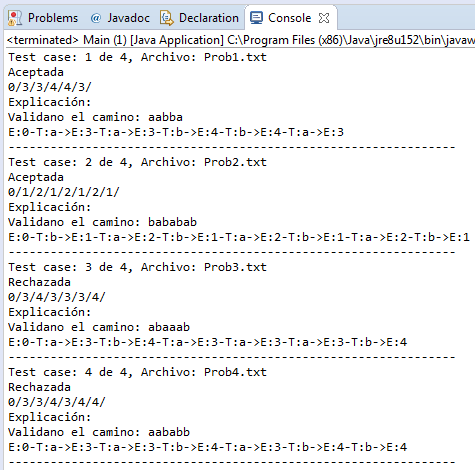


1. **abaaab**



1. **aababb**





## Conclusiones.

La tarea gira alrededor de comprender cómo se puede traducir una autómata en código. En mi caso decidí usar una matriz que reflejara las transiciones. De esta forma mis tiempos de búsquedas es constante, aunque puede hacer que en uso de memoria sea algo intensivo dado que estoy usando 2 HashMaps y la matriz será igual de grande que nodos multiplicado por los elementos dentro de nuestro alfabeto.

Tenía otra forma planeada para resolverlo, que era utilizando clases “Nodos”. Estas clases tendrían un arreglo que apunte al siguiente nodo dependiendo de la transición. Actuaría como una estructura tipo árbol una vez que todos los nodos fueran agregados. El conflicto que vi fue el tiempo de implementación y los tiempos que ya tenía encima. Considero que usando objetos hubiera sido una solución más limpia.

Haciendo una relación entre análisis y diseño de algoritmos junta con esta, he estado viendo mucho potencial en cambiar ciertas implementaciones en mi trabajo a utilizar grafos. Debo ver cómo cambiar algunos procesos para que funcionen utilizando la lógica de una autónoma no determinista. Así podría ver si puedo comenzar a automatizar aún más procesos.

## Referencias

Material de clases