

Faculty of computer and information

Automata &Language Theory(CS342)

* **Project name:**

**DFA Code.**

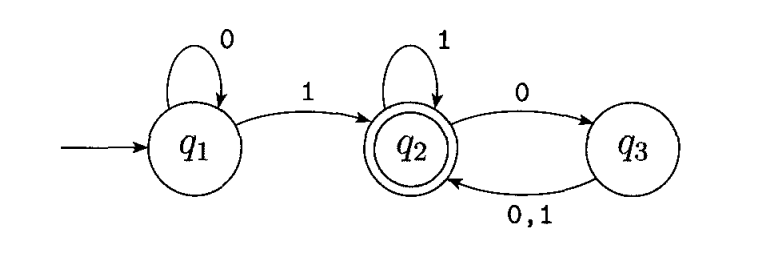
* **Team Members:**

1-عبدالرحمن عبدالناصر محمد عطيفي

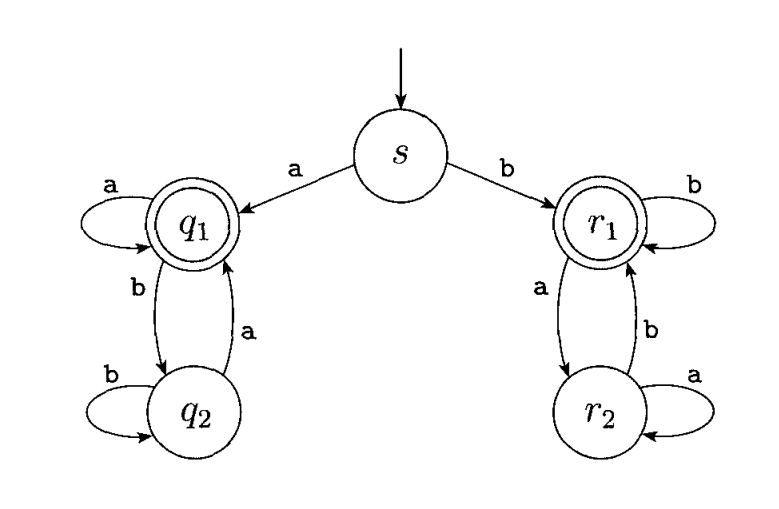
2-علاء الدين امين عبدالله نصير

3- فادى رفعت راغب مكسيموس

* **Introduction:**
* Our DFA accepts any language that contain an even number of “a” ‘s and any number of “b” ‘s that is greater than 1.
* The automatic door controller moves from state to another state, depending on the input that it received. Finite state machine consists of five things: [states, alphabet, transition function, start state, acceptance states]. It can be represented in a state diagram, the photo below is an example of a finite state machine:



* This machine accepts any string that ends with a 1 and also any string that ends with an even number of 0’s following the last 1. q2 represents the accept state, 1 and 0 are the alphabet of this language.
* The formal definition of the DFA is that it’s a list of five objects: set of states, input alphabet, rules for moving, start state and accept state. We also can define it as a 5-tuple consisting of these five parts.
* Here’s some examples of the DFA machines:



* **Machine Design:**
* FINITE AUTOMATA

Finite automata are good models for computers with an extremely limited

amount of memory. What can a computer do with such a small memory? Many

useful things! In fact, we interact with such computers all the time, as they lie at

the heart of various electromechanical devices.

The controller for an automatic door is one example of such a device. Often

found at supermarket entrances and exits, automatic doors swing open when

sensing that a person is approaching. An automatic door has a pad in front to

* (DFA), deterministic finite-state machine (DFSM), or deterministic finite-state automaton (DFSA)

is a [finite-state machine](https://en.wikipedia.org/wiki/Finite-state_machine) that accepts or rejects a given [string](https://en.wikipedia.org/wiki/String_(computer_science)) of symbols, by running through a state sequence uniquely determined by the string.[[1]](https://en.wikipedia.org/wiki/Deterministic_finite_automaton#cite_note-Hopcroft_2001-1) *Deterministic* refers to the uniqueness of the computation run. In search of the simplest models to capture finite-state machines, [Warren McCulloch](https://en.wikipedia.org/wiki/Warren_McCulloch) and [Walter Pitts](https://en.wikipedia.org/wiki/Walter_Pitts) were among the first researchers to introduce a concept similar to finite automata in 1943.

* Formal definition

A deterministic finite automaton *M* is a 5-[tuple](https://en.wikipedia.org/wiki/N-tuple), (*Q*, Σ, *δ*, *q*0, *F*), consisting of

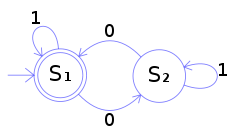
* a finite set of [states](https://en.wikipedia.org/wiki/State_(computer_science)) *Q*
* a finite set of input symbols called the [alphabet](https://en.wikipedia.org/wiki/Alphabet_(computer_science)) Σ
* a transition [function](https://en.wikipedia.org/wiki/Function_(mathematics)) *δ* : *Q* × Σ → *Q*
* an initial or [start state](https://en.wikipedia.org/wiki/Finite-state_machine#Start_state) {\displaystyle q\_{0}\in Q}
* a set of [accept states](https://en.wikipedia.org/wiki/Finite-state_machine#Accept_.28or_final.29_states) {\displaystyle F\subseteq Q}

Let *w* = *a*1*a*2…*an* be a string over the alphabet Σ. The automaton *M* accepts the string *w* if a sequence of states, *r*0, *r*1, …, *rn*, exists in *Q* with the following conditions:

1. *r*0 = *q*0
2. *ri*+1 = *δ*(*ri*, *ai*+1), for *i* = 0, …, *n* − 1
3. {\displaystyle r\_{n}\in F}

* **Example**

The following example is of a DFA *M*, with a binary alphabet, which requires that the input contains an even number of 0s.

[](https://en.wikipedia.org/wiki/File:DFAexample.svg)

The [state diagram](https://en.wikipedia.org/wiki/State_diagram) for *M*

*M* = (*Q*, Σ, *δ*, *q*0, *F*) where

* *Q* = {*S*1, *S*2}
* Σ = {0, 1}
* *q*0 = *S*1
* *F* = {*S*1} and
* *δ* is defined by the following [state transition table](https://en.wikipedia.org/wiki/State_transition_table):

|  |  |  |
| --- | --- | --- |
|  | **0** | **1** |
| ***S*1** | *S*2 | *S*1 |
| ***S*2** | *S*1 | *S*2 |

The state *S*1 represents that there has been an even number of 0s in the input so far, while *S*2 signifies an odd number. A 1 in the input does not change the state of the automaton. When the input ends, the state will show whether the input contained an even number of 0s or not. If the input did contain an even number of 0s, *M* will finish in state *S*1, an accepting state, so the input string will be accepted.

The language recognized by *M* is the [regular language](https://en.wikipedia.org/wiki/Regular_language) given by the [regular expression](https://en.wikipedia.org/wiki/Regular_expression) (1\*) (0 (1\*) 0 (1\*))\*, where \* is the [Kleene star](https://en.wikipedia.org/wiki/Kleene_star), e.g., 1\* denotes any number (possibly zero) of consecutive ones.

* **Machine Implementation:**

**- we use programing language c++ with visual studio.**

**- We make transition table vector to store rules of moving and accepted states container (vector) to mark final states as accepted**

**We make 7 methods:**

**1- defineDFA(): Take the 5-tuples of DFA , [states,input alphapet, rules of moving,start state and final states].**

**2- extractAlphapit(): Remove extra characters and spaces from the alphapit string**

**3- takeTransitionTable(): Takes rules of moving and stores every move for each state**

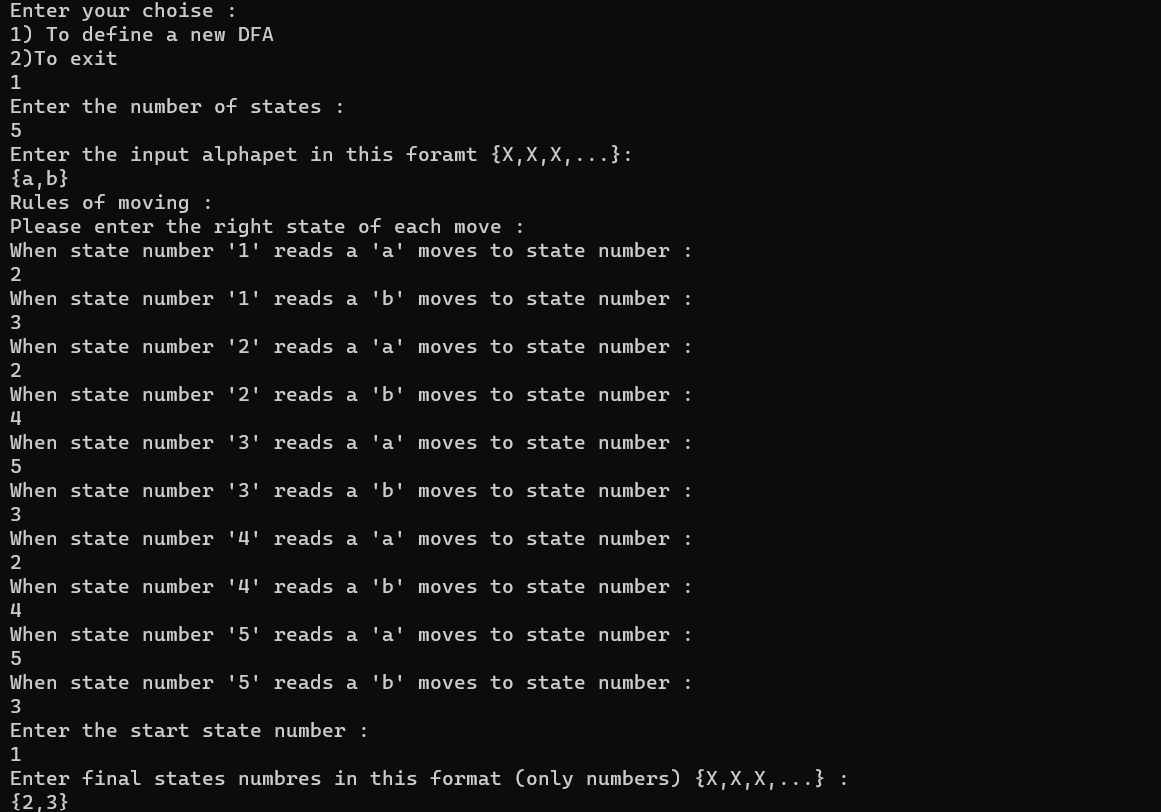
**4- exractFinalStates(): Extract final states numbers and mark tehm as accepted states**

**5- testString(): Takes a string and determines it will be accepted or rejected, due to the defined DFA machnine**

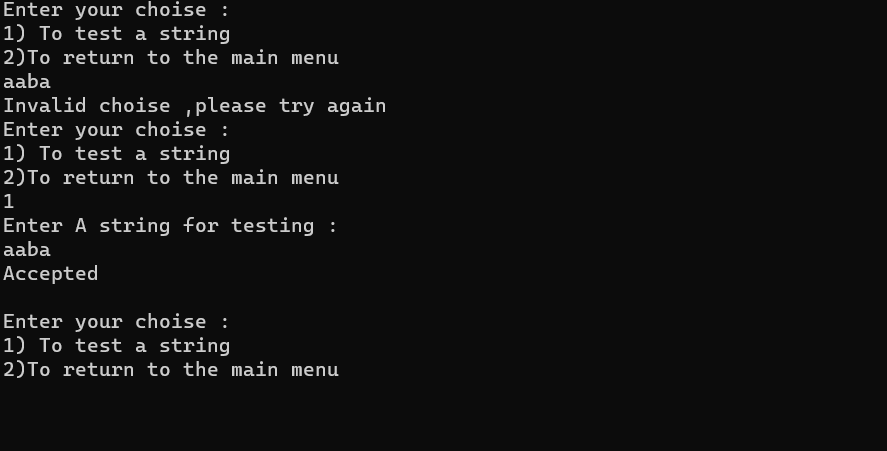
**6-** **takeInteger():** **To insure that the taken number is a valid one and in specific range and handels any wrong input**

**7- menu(): Provides options to the user to choose between progaram features**

**we can define any DFA machine .We show example for our test code**

****

Here we define DFA machine

****

Here we test DFA machine by write any path

**Text

Description automatically generated**

**Text

Description automatically generated**

* **Appendix:**

#include<iostream>

#include<string>

#include<vector>

#include<algorithm>

using namespace std;

const int maxSize = 500;

//Transition table to store rules of moving.

vector<vector<int>>transitionTable;

vector<bool>acceptedStates;// To mark final states as accepted.

string inputAlphapet, finalStates;

int numberOfStates, startState;

int takeInteger(int start, int end)

//To insure that the taken number is a valid one and in specific range

{

int number;

while (true)

{

string input; bool validNumber = true;

cin >> input;

int size = input.size();

for (int i = 0; i < size; i++) {

if (!isdigit(input[i])) {

cout << "Invalid number ,please try again \n";

validNumber = false;

break;

}

}

if (validNumber) {

number = stoi(input);

if (!(number >= start && number <= end)) {

cout << "Invalid number ,please try again \n";

continue;

}

else return number;;

}

else

continue;

}

return number;

}

void takeTransitionTable()

// Takes rules of moving and stores every move for each state

{

char letter; int nextMove;

cout << "Rules of moving :\nPlease enter the right state of each move :\n";

for (int i = 1; i <= numberOfStates; i++) {

for (int j = 0; j < inputAlphapet.size(); j++) {

letter = inputAlphapet[j];

cout << "When state number '" << i << "' reads a '"

<< letter << "' moves to state number :\n";

nextMove = takeInteger(1, numberOfStates);

//cin >> nextMove;

transitionTable[i][letter - '0'] = nextMove;

}

}

}

void extractAlphapit()

//Remove extra characters and spaces from the alphapit string

{

string temp = "";//To temporary store the alphapet

int size = inputAlphapet.size();

for (int i = 0; i < size; i++) {

char letter = inputAlphapet[i];

if (isalpha(letter) || isdigit(letter))

temp.push\_back(letter);

}

inputAlphapet = temp;

}

void exractFinalStates()

//Extract final states numbers and mark tehm as accepted states

{

int size = finalStates.size();

for (int i = 0; i < size; i++) {

char letter = finalStates[i];

if (isdigit(letter))

acceptedStates[letter - '0'] = true;

}

}

void testString()

//Takes a string and determines it will be accepted or rejected

// due to the defined DFA machnine

{

string stringForTesting;

cout << "Enter a string for testing :\n";

cin >> stringForTesting;

int currentState = startState;

int size = stringForTesting.size();

for (int i = 0; i < size; i++) {

char letter = stringForTesting[i];

currentState = transitionTable[currentState][letter - '0'];

//checl if the letter in the input alphapet.

if (currentState == -1) {

cout << "Rejected\n\n";

return;

}

}

if (acceptedStates[currentState])

cout << "Accepted\n\n";

else

cout << "Rejected\n\n";

}

void defineDFA()

//Take the 5-tuples of DFA

//[states,input alphapet,rules of moving,start state and final states].

{

//Return the transition table to it's primary state with initial values equal to -1

transitionTable = vector<vector<int>>(maxSize, vector<int>(maxSize, -1));

//Return the acceptedStates array to it's primary state with initial values equal to fasle

acceptedStates = vector<bool>(maxSize, false);

cout << "Enter the number of states : \n";

numberOfStates = takeInteger(1, maxSize);

//cin >> numberOfStates;

cin.ignore();

cout << "Enter the input alphapet in this foramt {X,X,X,...}: \n";

getline(cin, inputAlphapet);

extractAlphapit();

takeTransitionTable();

cout << "Enter the start state number :\n";

cin >> startState;

cin.ignore();

cout << "Enter final states numbres in this format (only numbers) {X,X,X,...} :\n";

getline(cin, finalStates);

exractFinalStates();

}

void menu()

//Provides options to the user to choose between progaram features

{

while (true)

{

string choise;

cout << "Enter your choise :\n";

cout << "1) To define a new DFA\n"

<< "2)To exit\n";

cin >> choise;

if (choise == "1") {

defineDFA();

while (true)

{

string choise;

cout << "Enter your choise :\n";

cout << "1) To test a string\n"

<<

"2)To return to the main menu\n";

cin >> choise;

if (choise == "1")

testString();

else if (choise == "2")

break;

else

cout << "Invalid choise ,please try again\n";

}

}

else if (choise == "2")

break;

else

cout << "Invalid choise ,please try again\n";

}

cout << "\n\*\*\*\*\*\*\*\*\*\*\*THANK YOU\*\*\*\*\*\*\*\*\*\*\*\n";

}

int main() {

menu();

}

* **References:**
* **Michael Sipser, Introduction to the Theory of Computation,2nd**

**Edition, Thomson Course Technology, 2006.**

* **https://ar.wikipedia.org**