Experiment - 1

# Deterministic Finite Automata (DFA)

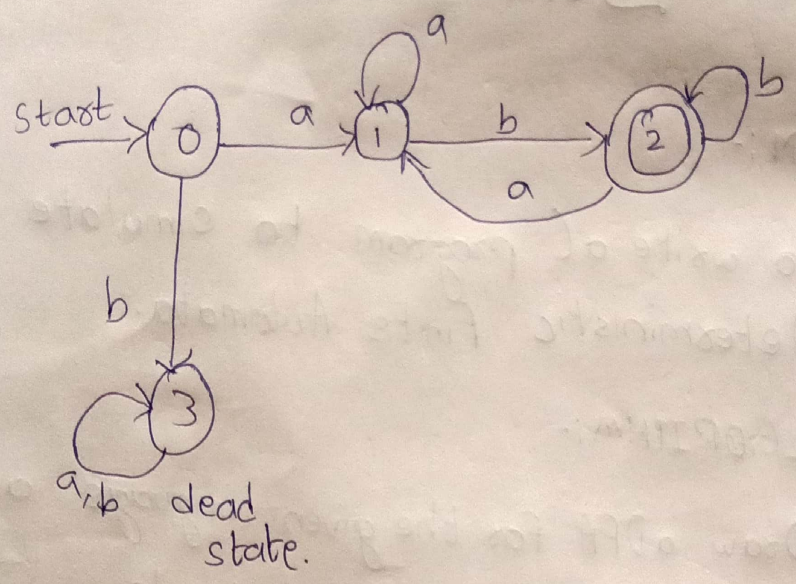
AIM :-

To write a C program to simulate a Deterministic Finite Automata.

ALGORITHM :-

1. Draw a DFA for the given language and construct the transition table.
2. Store the transition table in a two-dimensional array.
3. Initialize present_state, nextstate and final_state
4. Get the input string from the user.
5. Find the length of the input string.
6. Read the input string character by character.
7. Repeat step 8 for every character.

Design of the DFA




Transition Table:

| State/ Input | a | b |
|---|---|---|
| →0 | 1 | 3 |
| 1 | 1 | 2 |
| 2 | 1 | 2 |
| 3 | 3 | 3 |

PROGRAM:

```
# include<stdio.h>
# include<string.h>
# define max20
int main()
{
    int trans table[4][2] = {{1,3},{1,2},{1,2},{3,3}};
    int final_ state=2,i;
    int present_state=0;
    int next_ state=0;
    int invalid=0;
    char input _string[max];
    Printf("Enter a string:");
    scanf("%s",input_string);
    int l=strlen(input_string);
    for(i=0;i<l;i++)
    {
        if[input_string[i]==='a')
        next_state=trans_table
        else
        invalid=1;
        present_State=next_state
```

```
}
if(invalid==1)
{
        printf("Invalid input");
}
else if(present_state==final_state)
printf("Accept\n");
else
printf("Don't Accept\n");
}
```

Output

Enter a string : abaab
Accept

Process returned 0(0x0) executiontime: 7.5135

Press any key to continue.

# CHECKING WHETHER A STRING BELONGS TO A GRAMMAR

AIM:

To write a C program to check whether a string belongs to the grammar

$S \longrightarrow 0A1$

$A \longrightarrow 0A \mid 1A \mid \varepsilon$

Language defined by the Grammar:

set of all strings over $\Sigma = (0, 1)$ that start with 0 and end with 1

ALGORTIHM

1. Get the input string from the user
2. Find the length of the string.
3. check whether all the symbols in the input are either 0 or 1.
4. If the first symbol is 0 and the last symbol is 1, print "string accepted".

PROGRAM;

```
#include <stdio.h>
# include <string.h>
int main(){
char s [100];
int i, Flag;
int I;
Print f("enter a string to check:");
scanf("%s", s);
I = strlen(s)
Flag = 1:
for(i=0;i<l;i++)
{
    if(s[i] != '0' && s[i] != '1')
    {
    Flag = 0;
    }
}
if(Flag != 1)
        Printf("string is Not valid\n");
if(Flag == 1)
```

{

iF(s[0] = = '0' && s[l-1] = = '1')
    Printf("string is accepted\n");
else
    printf("string is Not accepted\n");

}

}

<u>Output:</u>

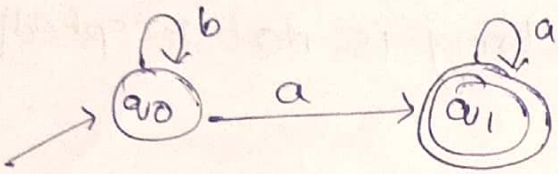
Enter a string to check : 0101011101

string is accepted.

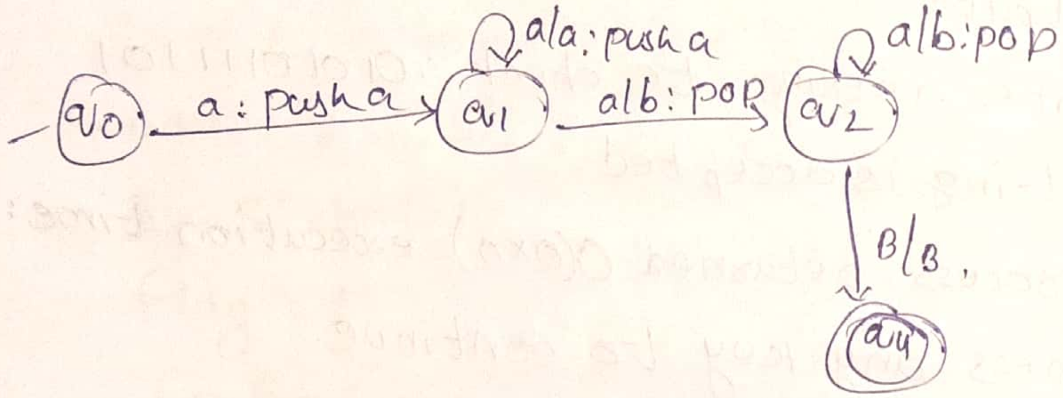
process returned 0(0x0) execution time : 2.5719s

press any key to continue.

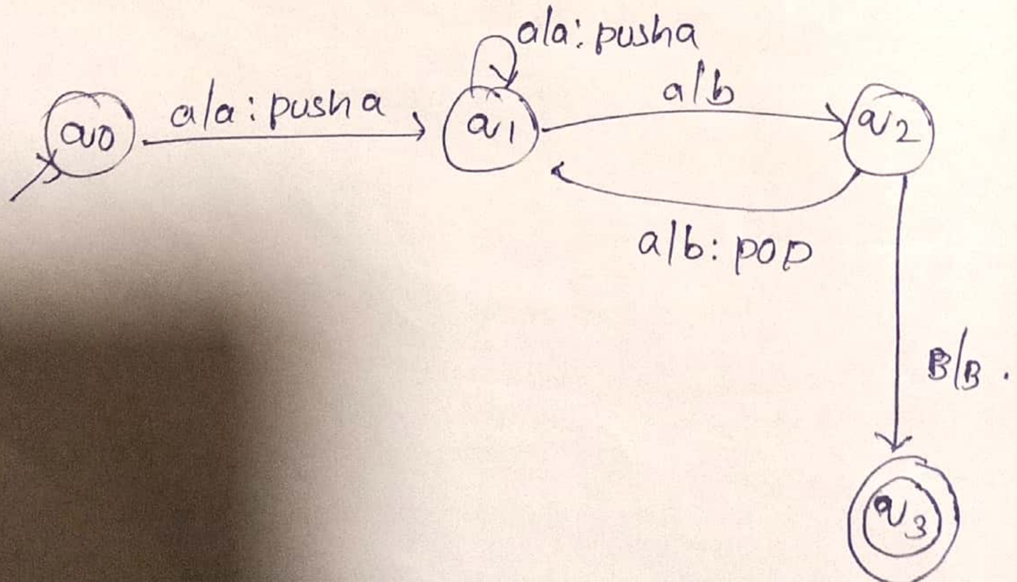
Ex3:- Design DFA using simulator to accept the input string "a", "ac", and "bac".



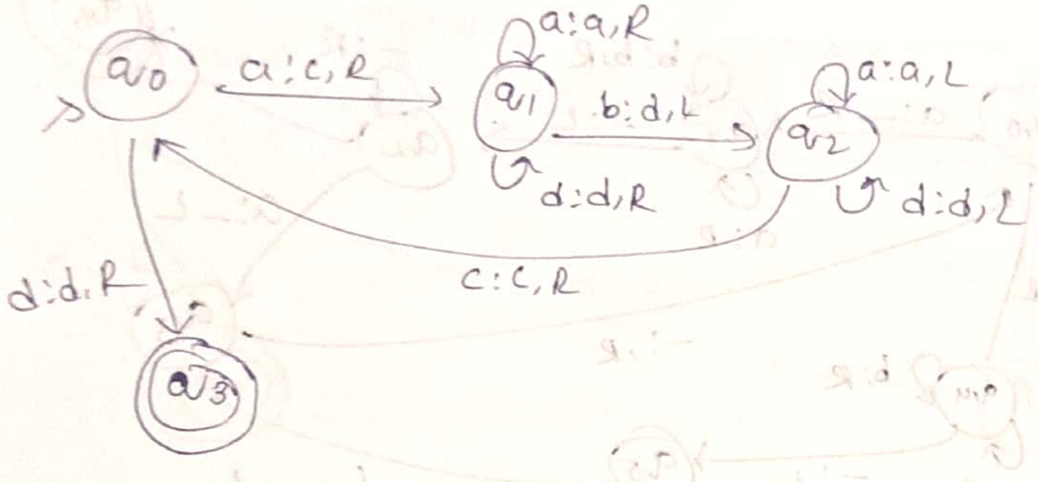

Ex4: Design PDA using simulator to accept the input string aabb



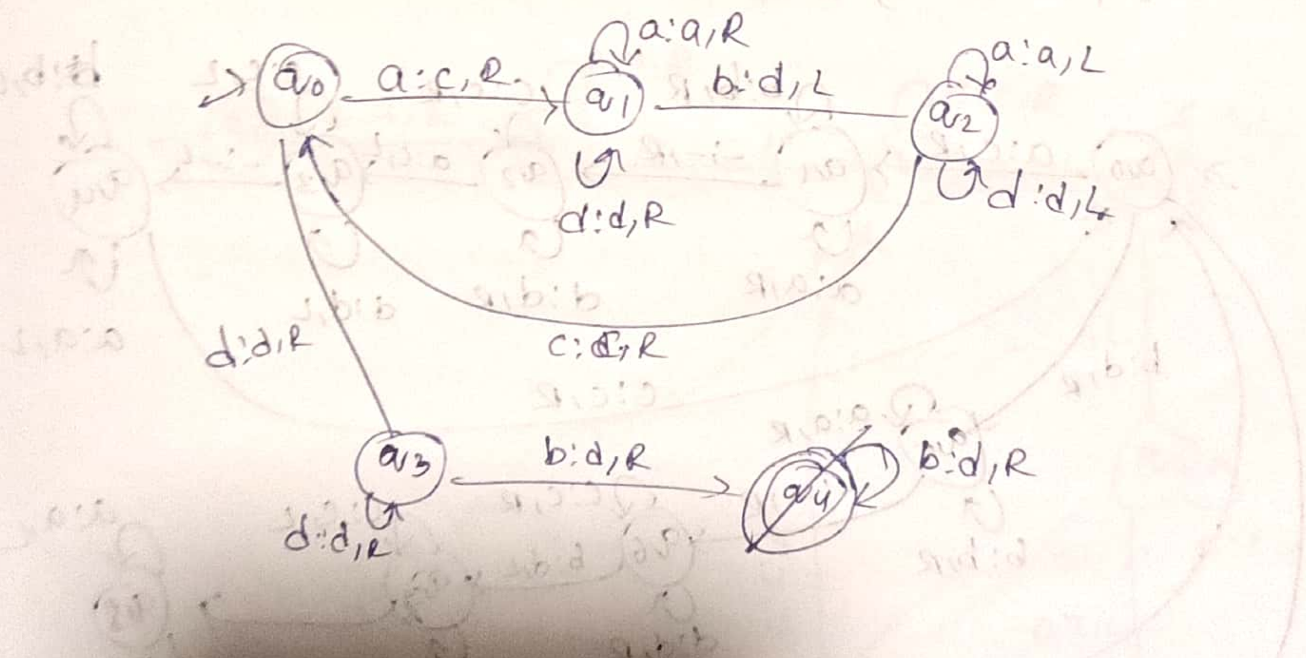

Ex5:- Design PDA using Simulator to accept the input string a^n b^2n



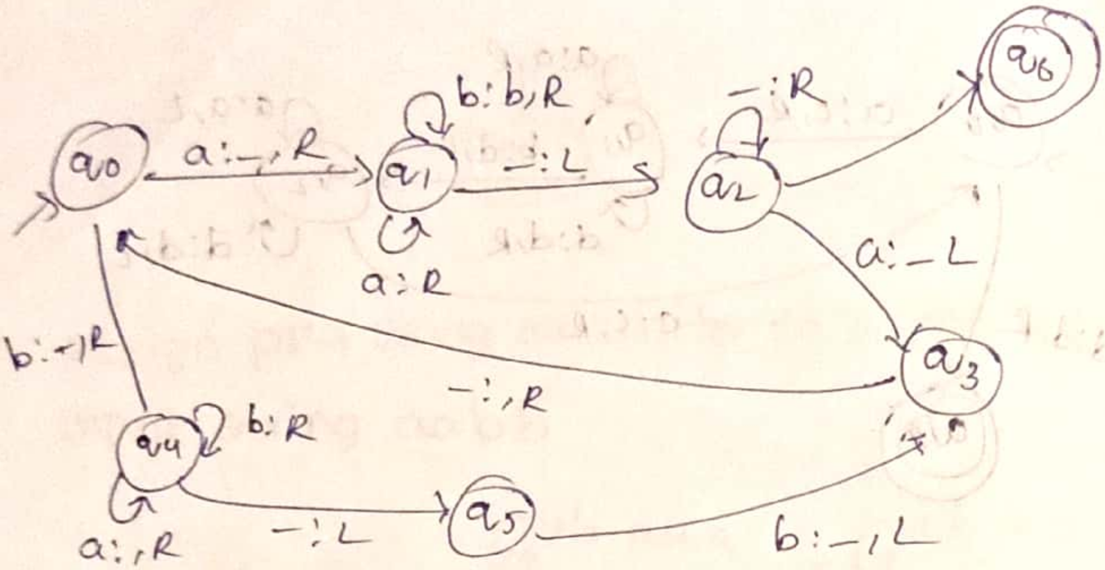
Ex6:- Design TM using simulator to accept the input A^n B^n



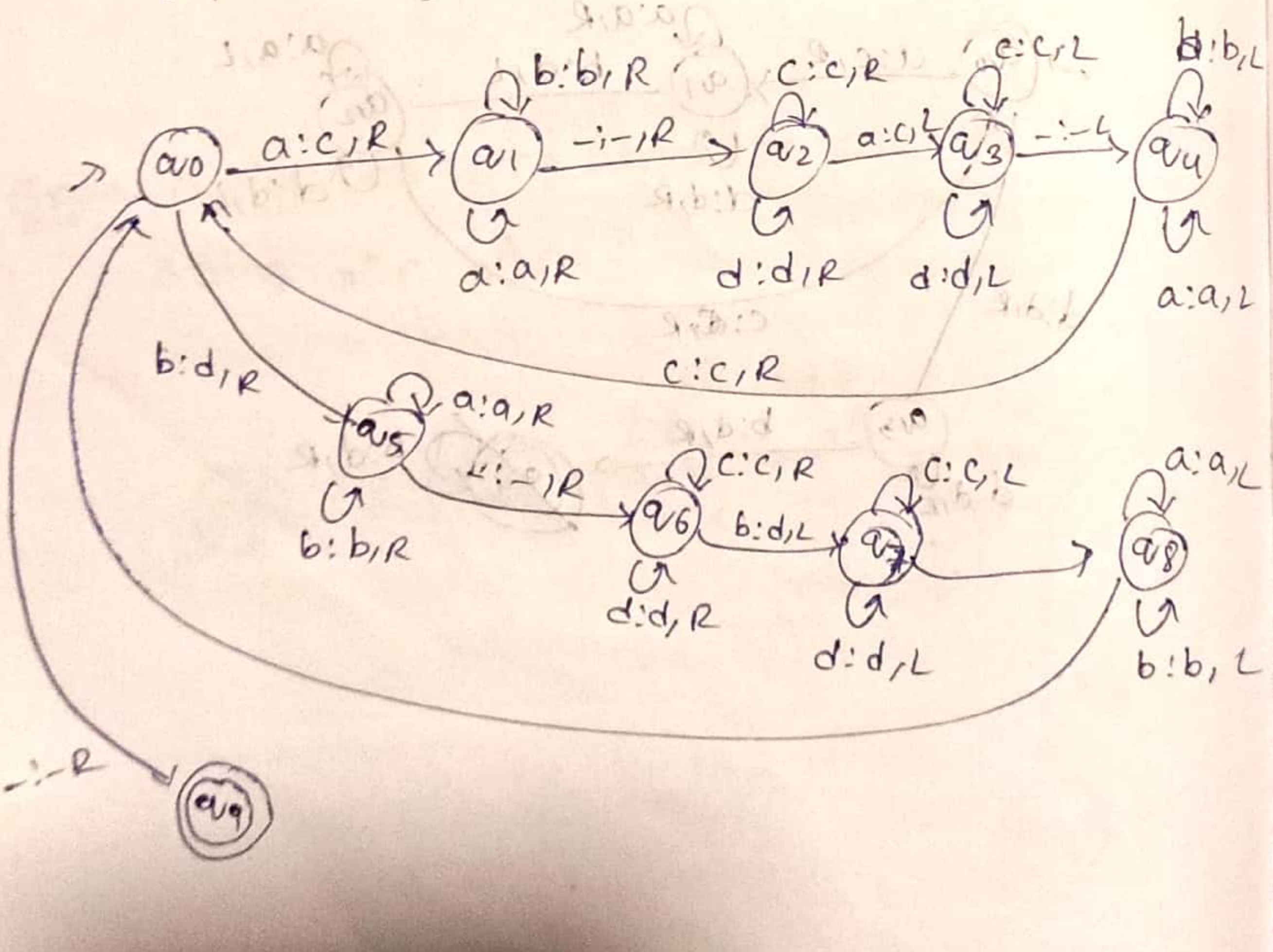

Ex7:- Design TM using Simulator to accept the Input String A^n B^2n



Ex8:- Design TM using Simulator to accept the input string palindrome ababa

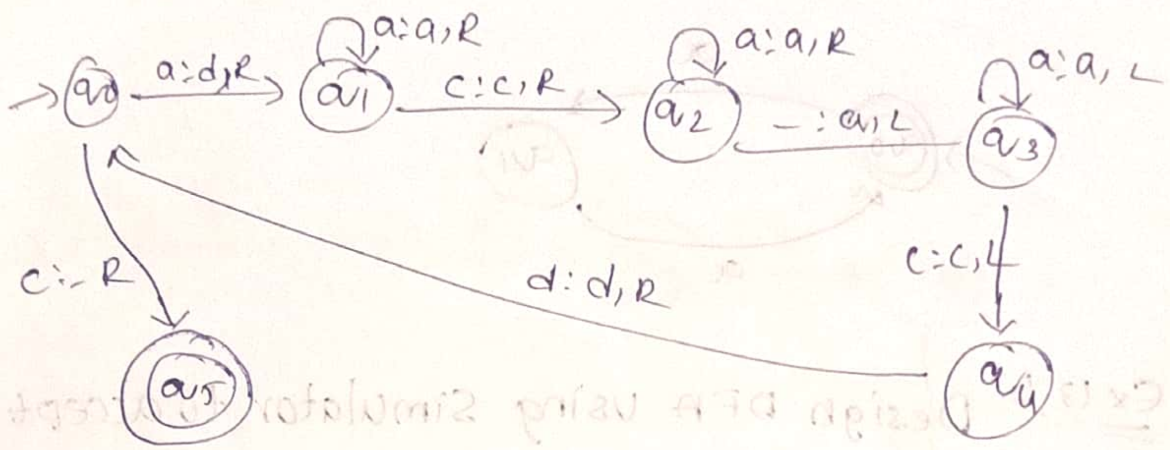



Ex9:- Design TM using Simulator to accept the Input String ww

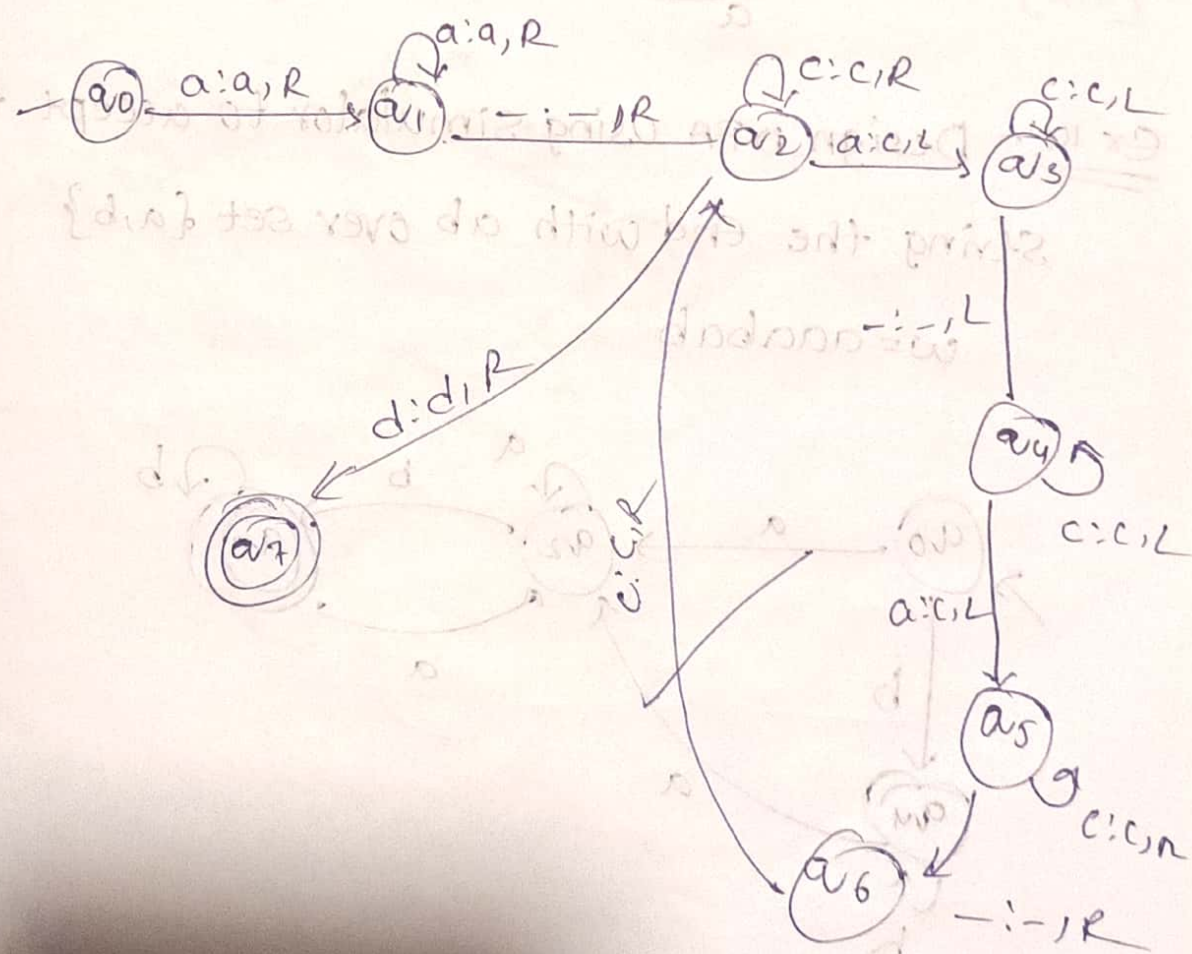



Scanned with CamScanner

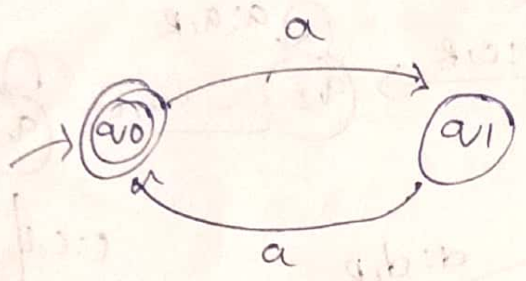
Ex10:- Design TM using Simulator to perform addition of "aa" and "aaa"



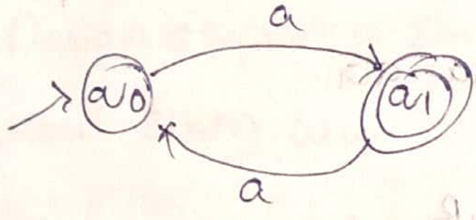

Ex11:- Design TM using simulator to perform subtraction of aaa-aa



Ex 12:- Design DFA using simulator to accept even number of a's

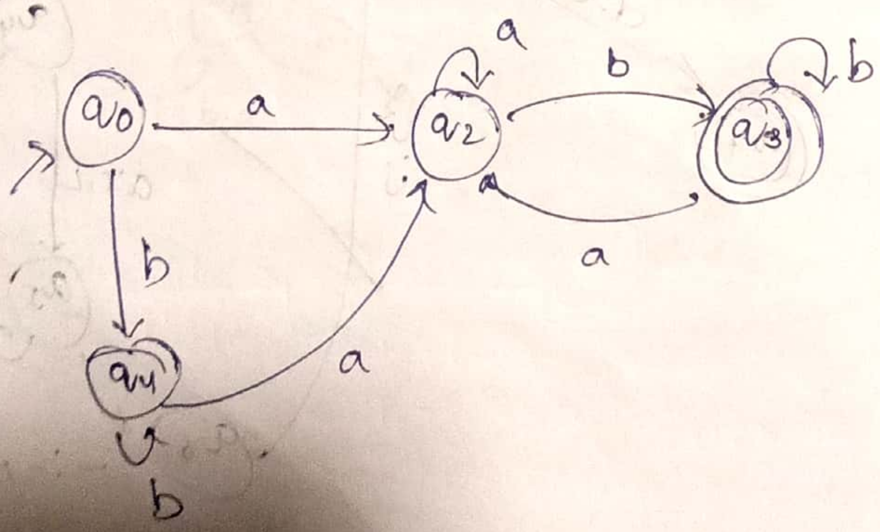



Ex 13:- Design DFA using simulator to accept odd number of a's

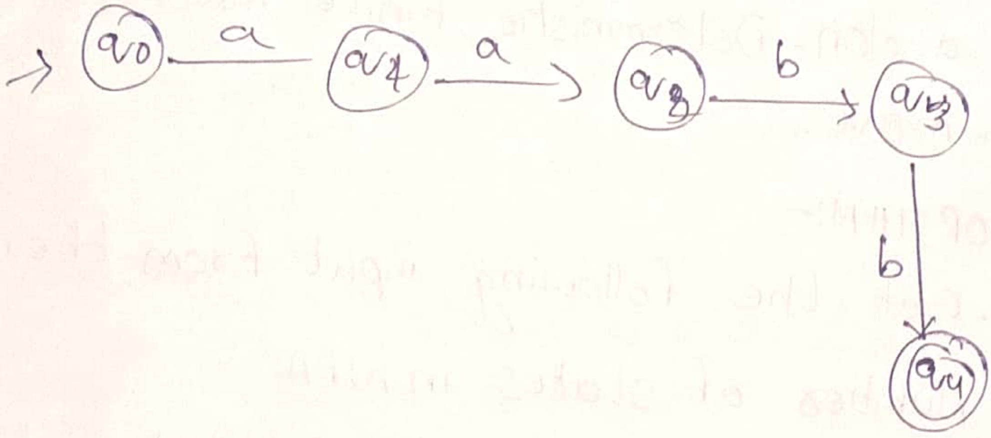



Ex 14:- Design DFA Using simulator to accept the string the end with ab over set {a,b}

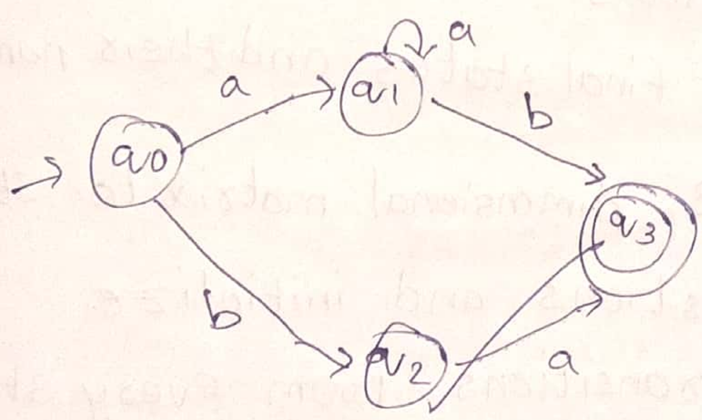
w= aaabab



Ex15:- Design DFA using Simulator to accept the string having "ab" as Substring over the set {a,b}




Ex16:- Design DFA Using Simulator to accept the String start with a or b over set {a,b}



Ex17

# FINDING ε-CLOSURE FOR NFA WITH ε-moves

AIM:-

To write a cprogram to find ε-closure of a Non-Deterministic Finite Automata with ε-move.

ALGORTIHM:-

1. Get the following input from the user.

(i) Number of states in NFA

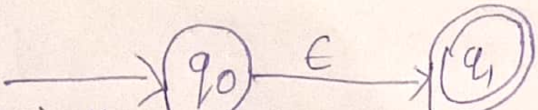
(ii) Number of symbol input alphabet includ; ε:

(iii) input symbols.

(iv) Number of final states and their names.

2. Declear a 3-dimensional matrix to store the transistions and initialize.

3. Get the transitions from every state for input symbol from the user and store.




4. Initialize of two-dimensional matrix c-closure with -1 in all Entries.

5. ε-closure of state q is defined at set state that can be reached.

ε - closure(0) = 0, 1

ε - closure(1) = 1

ε - closure(2) = 2,

6. for every state - print e-closure values.

**Program:-**

```
# include <stdio.h>
# include <string.h>
int trans_table[10][5][3];
char symbol[5], a;
int e-closure[10][10], ptr, state;
char symbol [5], a;
~~int trans_ta~~
int main()
{
int i, j, k, n, num_state, num_symbols;
for(i=0; i<10; i++)
{
for(j=0; i<10; i++)
{
for(k=0; k<3; k++)
{
trans_table[i][j][k] = -1;
}
}
}
num_states = 3;
num_symbols = 2;
```

```
symbol[10]='e';
n=1;
tran_table[0][0][0]=1;
for(i=0; i<10; i++)
{
  for(j=0; j<10; j++)
  {
    e_closure[i][j]=-1;
  }
}
for(i=0; i<num_state, i++)
  e_closure[i][0]=i;
for(i=0, i<num_state; i++)
{
 if(trans_table[i][0][0]==-1)
  continue;
  else
  {
    state=i;
    ptr=1;
    find_e_closure(i);
   }
 }
 for(i=0; i<num_state ; i++)
 {
 Printf(e-closure(-1 d)={a, i});
 for(j=0 ; j<num_states; j++)
 {
  if(e-closure[i][j] != -1)
  {
```

```
Print F("%d,", e_closure [i] [j]);
}
}print F("}\n");
}
}
void find-e-closure(int x)
{
int i, j, y [10], num_trans;
i=0;
while (trans_table [x) [0] [i] != -1)
{ y[i] = trans_table [x] (0) [i];
i = i+1;
}
num_trans = i
For (j=0; j<num_trans; j++)
{
e-closure[state) [ptr) = y [i];
ptr++;
find-e-closure (y[i]);
}
}
```

Output:-

e-closure (0) = {0,1,}
e - closure (1) = {1,}
e- closure (2) = {2}.