DETERMINISTIC FINITE AUTOMATA (DFA)

Exp No:1.

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, next_state 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.

8.) Refer the transition table for the entry corresponding to the present state and the current input symbol and update the next state.

9.) When we reach to end of the input, if the final state is reached, the input is accepted. Otherwise the input is not accepted.

Program:

```
#include<stdio.h>
#include<string.h>
#define max 20
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[present_state][0];
    else if(input_string[i]=='b')
    next_state = trans_table[present_state][1];
    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:

i.) Enter a String: abaab
Accept.

ii.) Enter a String: abbbaaaba
Don't Accept.

FINDING ε-CLOSURE FOR NFA WITH ε-MOVES.

Exp No: 2 (Typo: Exp No: 3)

AIM:

To write a C program to find ε-closure of a Non-Deterministic finite Automata with ε-moves.

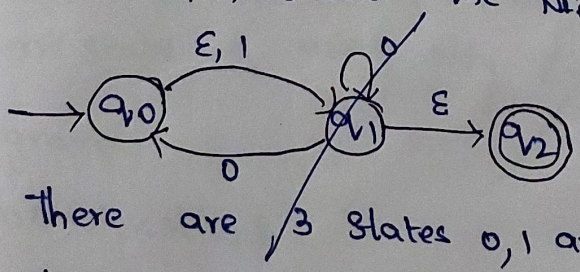
Algorithm:

1.) Get the following as input from the user.
   i.) Number of states in the NFA.
   ii.) Number of symbols in the input alphabet including ε.
   iii.) Input symbols.
   iv.) Number of final states and their names.

2.) Declare a 3-dimensional matrix to store the transitions and initialize all the entries with -1.

3.) Get the transitions from every state for every input symbol from the user and store it in the matrix

for example, consider the NFA shown below:



There are 3 states 0, 1 and 2.

There are three input symbols ε, 0 and 1. As the array index always starts with 0, we assume 0th symbol is ε, 1st symbol is 0 and 2nd symbol is 1.

The transitions will be stored in the matrix as follows:

from state 0, for input ε, there is one transition to state 1, which can be stored in the matrix as

m[0][0][0] = 1

from state 0, for input 0, there is no transition.

from state 0, for input 1, there is one transition to state 1, which can be stored in the matrix as

m[0][2][0] = 1

Similarly, the other transitions can be stored as follows:

m[1][0][0] = 2

m[1][1][0] = 1

All the other entires in the matrix will be -1 indicating no moves.

4.) Initialize a two-dimensional matrix e_closure with -1 in all the entires.

5.) ε-closure of a state q is defined as the set of all states that can be reached from state q using only ε-transitions.

Program:

```
#include<stdio.h>
#include<string.h>
int trans_table[10][5][3];
char symbol[5], a;
int e_closure[10][10], ptr, state;
```

```
void find e-closure (int x);
int main()
{
   int i, j, k, n, num_states, num-symbols;
   for (i=0; i<10; i++)
   {
      for (j=0; j<5; j++)
      {
         for (k=0; k<3; k++)
         {
            trans_table [i][j][k] = -1;
         }
      }
   }
printf("How many states in the NFA with e-
                                    moves:");
scanf("%d", &num_states);
printf("How many symbols in the input alphabet
                        including e:");
scanf("%d", &num_symbols);
printf("Enter the symbols without space.
                    Give 'e' first:");
scanf("%s", symbol);
for (i=0; i<num_states; i++)
{
   for (j=0; j<num_symbols; j++)
   {
```

```
printf("How many transitions from state
%d for the input %c:", i, symbol[j]);
scanf("%d",&n);
for(k=0; k<n; k++)
{
printf("enter the transitions %d from state
%d for the input %c:", k+1, i, symbol[j]);
scanf("%d", &trans_table[i][j][k]);
}
}
}
for(i=0;i<10;i++)
{
for(j=0; j<10; j++)
{
e_closure[i][i]=-1;
}
}
for(i=0; i< num_states; i++)
e_closure[i][0]=i;
for(i=0; i< num_states; i++)
{
if(trans_table[i][0][0]==-1)
continue;
else
{
state=i;
ptr=1;
find_e_closure(i);
```

```
    }
  }
    printf("}\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[j];
   ptr++;
   find_e_closure[y[j]);
 }
}
```

Output:

How many states in the NFA with e_moves:3
How many symbols in the input alphabet including e:3
How Enter the symbols without space. Give 'e' first: e01.

How many transitions from State 0 for the input e:1
Enter the transitions 1 from State 0 for the input e:1
How many transitions from State 0 for the input 0:0
How many transitions from state 0 for the input 1:1
Enter the transitions 1 from State 0 for the input 1:1
How many transitions from state 1 for input e:1
Enter the transtions 1 from State 0 for the input e:2
How many transitions from state 1 for the input 0:2
Enter the transitions 1 from state 1 for input 0:0
Enter the transitions 2 from State 1 for input 0:1
How many transitions from state 1 for input 1:0
How many transitions from state 2 for input e:0
How many transitions from State 2 for input 0:0
How many transitions from State 2 for input 1:0
e-closure(0) = {0, 1, 2,}
e-closure(1) = {1, 2,}
e-closure(2) = {2, 1}

CHECKING WHETHER A STRING BELONGS TO A GRAMMAR.

Exp No:3

AIM:

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

S → 0 A 1

A → 0A | 1A | ε

Language defined by the Grammar:

Set of all strings over $\Sigma = \{0, 1\}$ that start with 0 and end with 1.

Algorithm:

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. If so, print "String is Valid" and go to step 4. Otherwise print "String not valid" and quit the program.

4.) If the first symbol is 0 and the last Symbol is 1, print "String accepted". Otherwise, print "String not accepted".

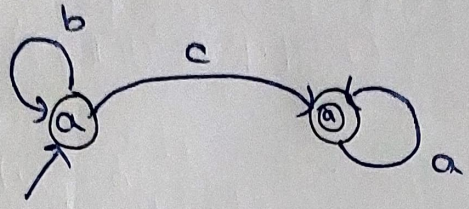
Program:

```
#include<stdio.h>
#include<string.h>
int main(){
char s[100];
int i, flag;
int l;
Print("enter a string to check:");
```

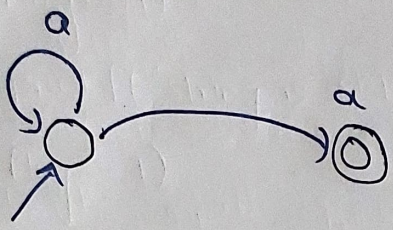
```
scanf("%s",s);
l=strlen(s);
flag=1;
for(i=0;i<l;i=i+1)
{
  if(s[i]=='0'&&s[i]!='1')
  {
   flag=0;
  }
}
 if(flag!=1)
    printf("String is Not Valid\n");
 if(flag==1)
{
  if(s[0]=='0'&&s[l-1]=='1')
      printf("String is accepted\n");
   else
      printf("String is Not accepted\n");
  }
}
```

Output:

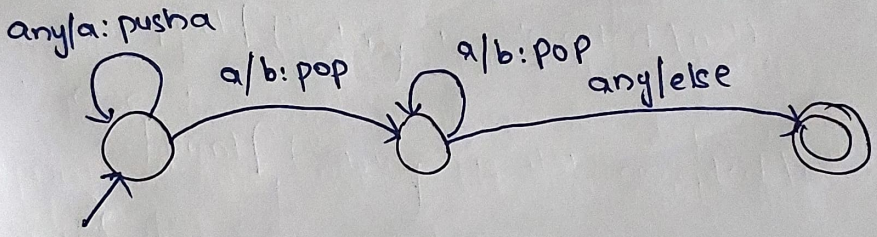
1) Design DFA to accept bcaaaaaaaaaaaaaa, bc. and c.



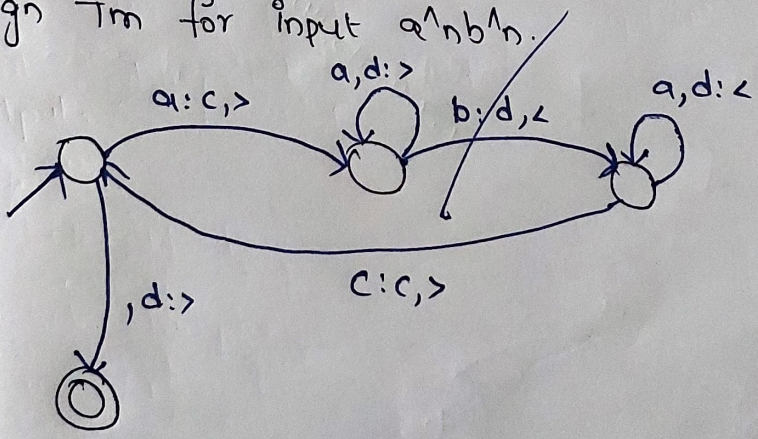
2) Design NFA to accept aaaaaa



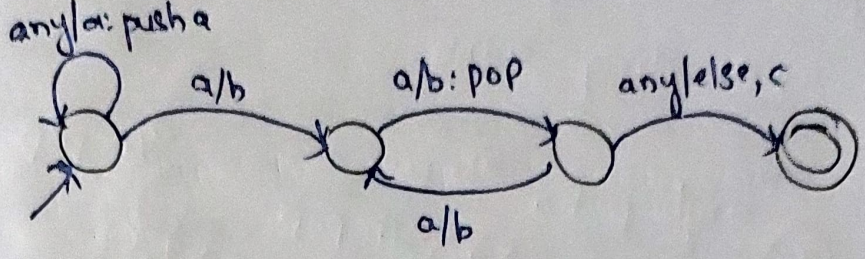
3) Design PDA for the input a^nb^n.



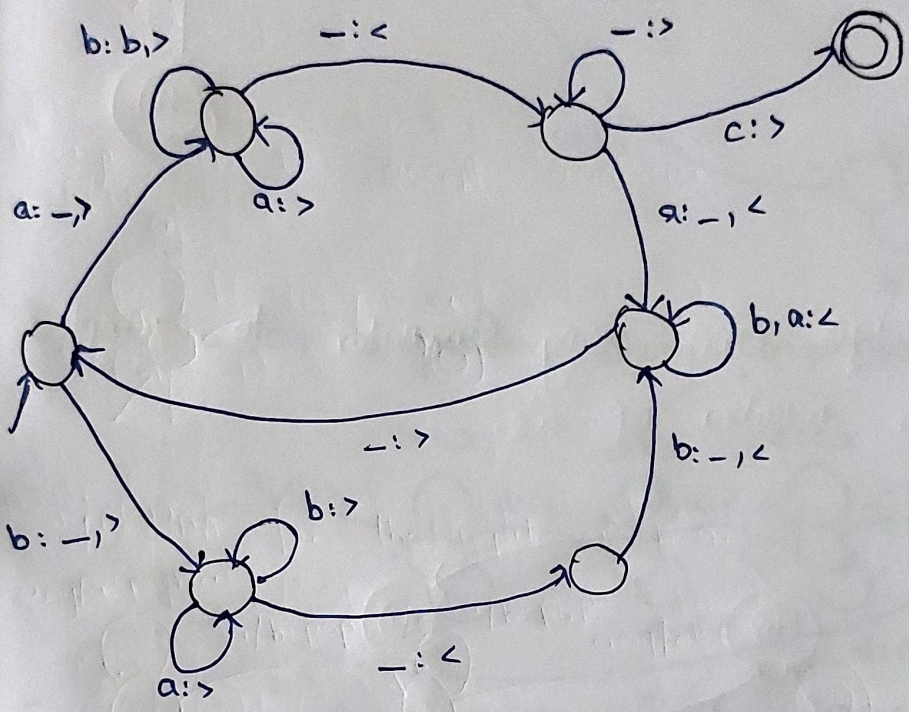
4) Design TM for input a^nb^n.



6) Design PDA for input aabbbbc ($L = a^n b^{2n}$)



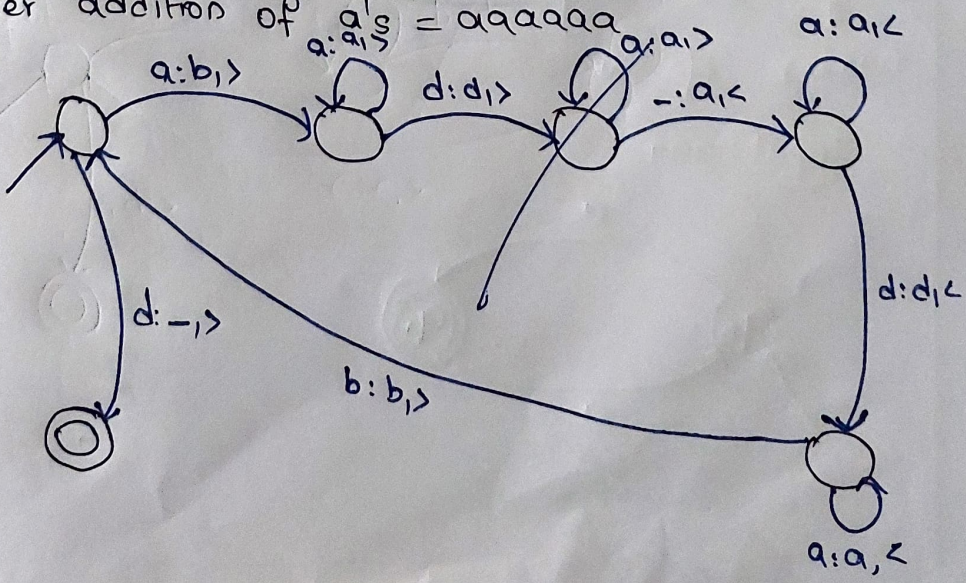
6) TM Simulation for Palindrome W = ababa C



7) Design TM to perform addition of following

W = aa + aaaa

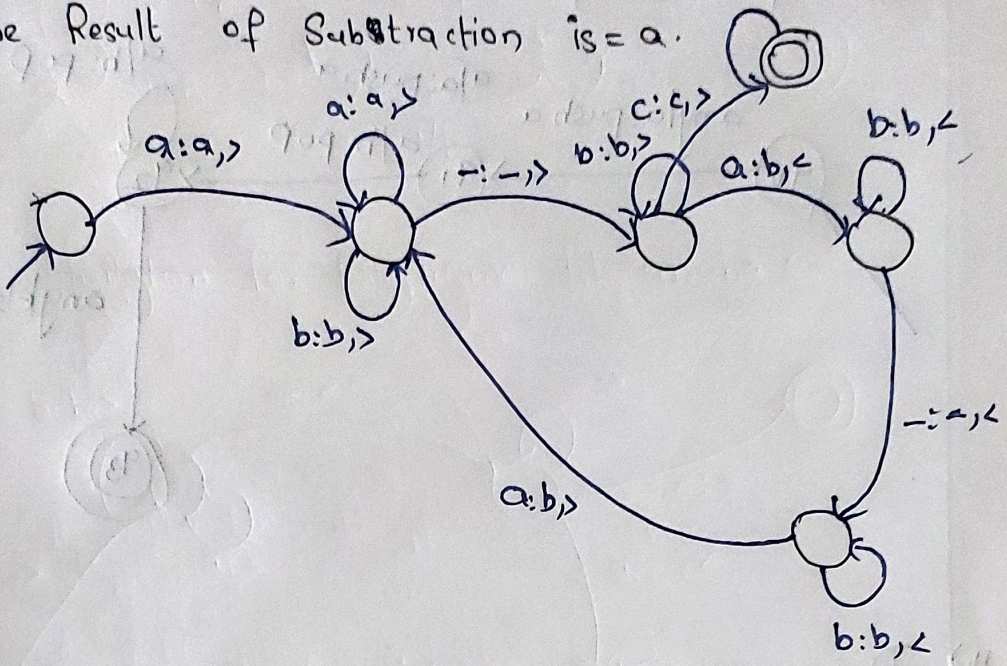
After addition of a's = aaaaaa



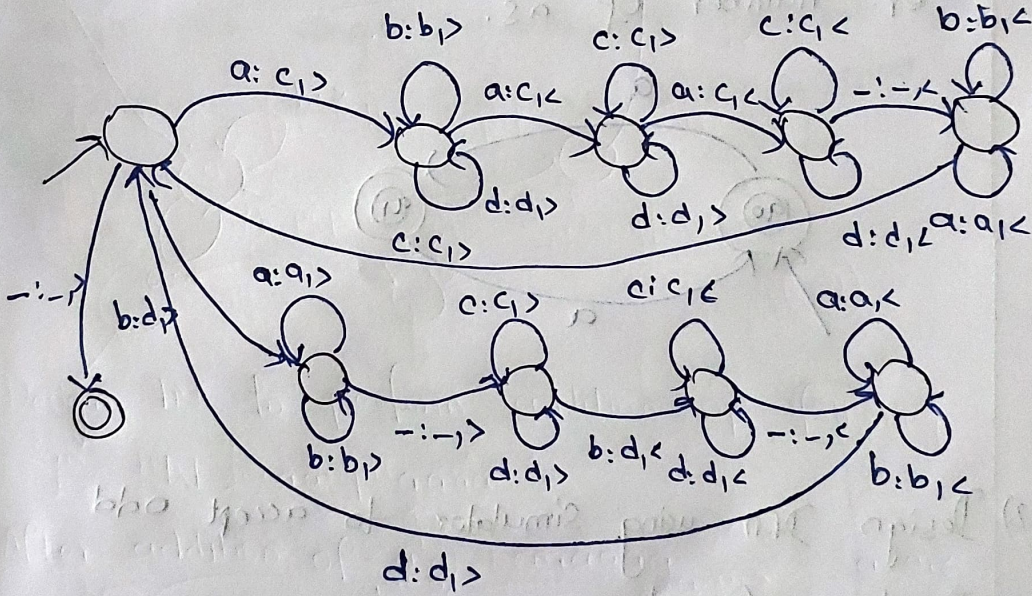
8.) Design TM to perform Subtraction

W = aaa-aa

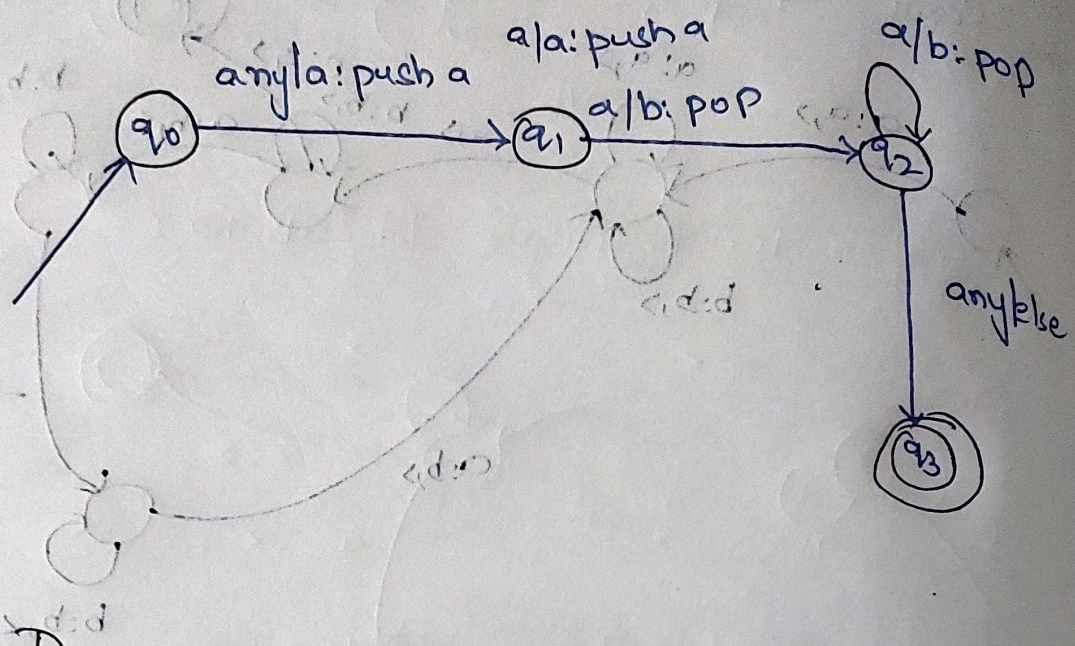
The Result of Subtraction is = a.



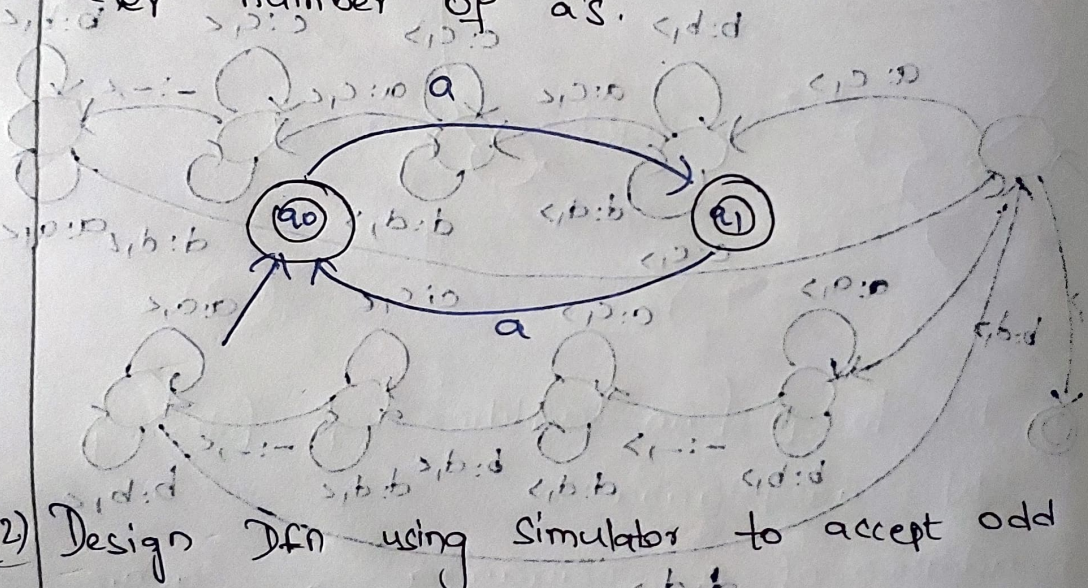
9.) Design a TM to perform String comparison.



10.) Design PDA using simulator to accept the input string aabb.

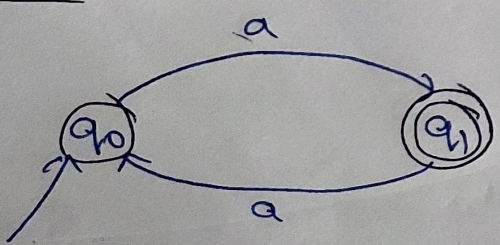


11.) Design DFA using simulator to accept even number of a's.



12) Design DFA using simulator to accept odd number of a's

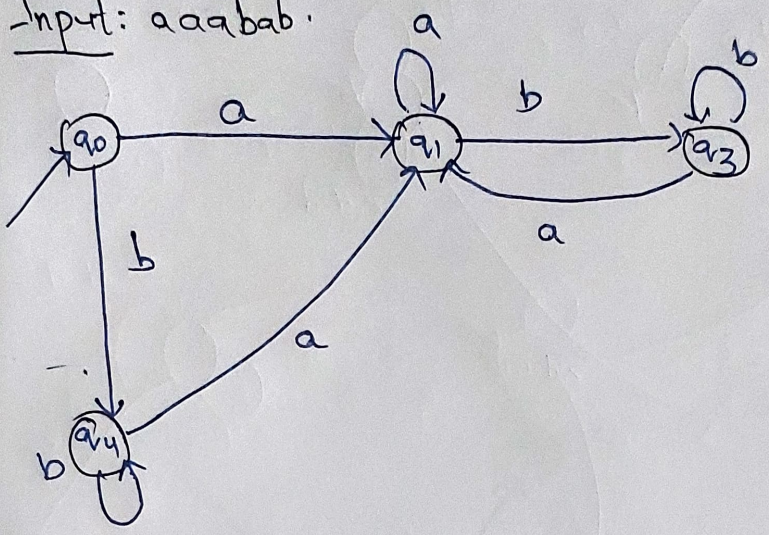
Input: aaa



13) Design DFA using simulator to accept the string the end with ab over set {a, b}

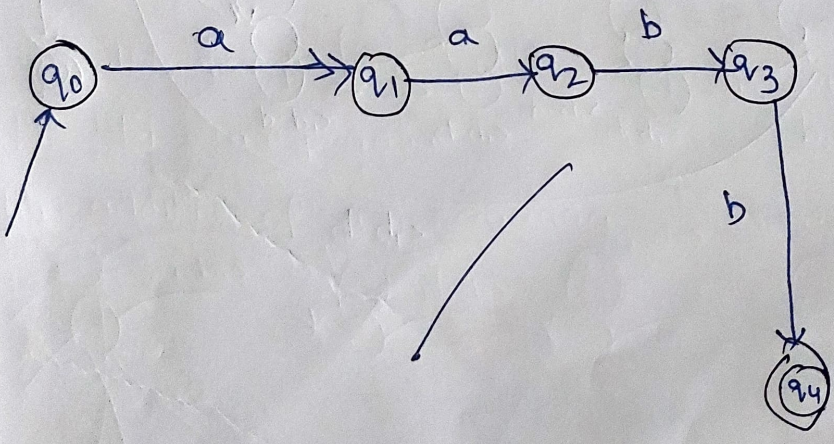
W = aaabab.

Input: aaabab.



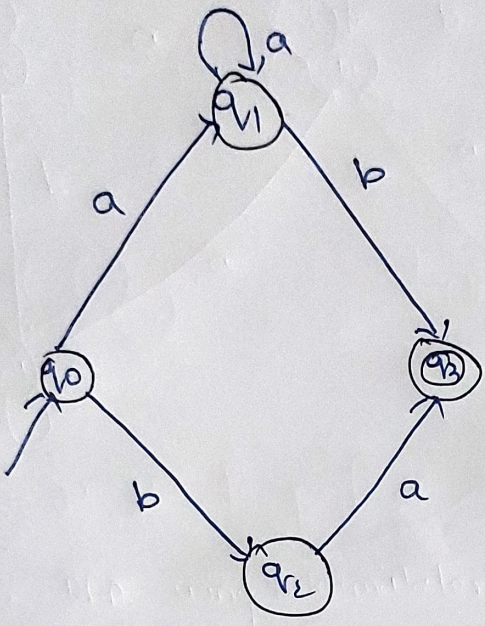
14) Design DFA using simulator to accept the string having "ab" as substring over the set {a, b}

Input: aabb



15.) Design DFA using Simulator -to accept the string Strat with a or b over Set {a, b}

Input: a or b {a, b}



16) Design TM using Simulator to accept the input string $A^n B^n$.