1) Deterministic finite Automata (DFA)

Aim:

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

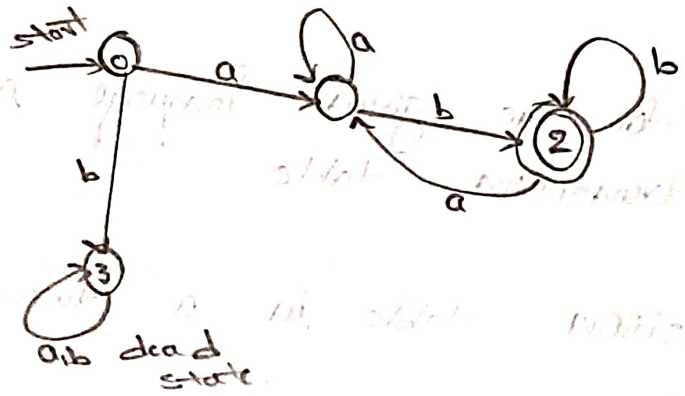
Algorithm

* Draw a DFA for the given language and construct the transition table.
* Store the transition table in a two-dimensional array.
* find the length of the input string.
* Repeat step 8 for every character.
* Refer the transition table for the table for the Entry corresponding to the present state the current input symbol and update the next state.
* when we reach the end input if the final stage is reached the input is accepted otherwise the input is not accepted

Example -

Simulate a DFA for the language accepting strings over $\Sigma = \{a, b\}$ that start with a and end with b

Design the DFA.



State / Input	a	b
→ 0	1	3
1	1	2
(2)	1	2
3	3	3

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 find_state = 2;
  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<j++)
{
    If(input_string(i) == '0')
next_state = trans_table[present_state][0];
  Else
    Invalid = 1;
Present_state = next_state
}
```

```
present_state = next_state.
}
if (invalid == 1)
{
    printf ("Invalid input");
}
Else if (present_state == final_state)
Print + ("Accept \n")
Else
Print + (" Don't Accept \n");
}
```

~~Output~~

Output -

```
Enter a string: abaab
Accept
Process returned 0 (0x0)   Execution time : 7.313 s
Press any key to continue.

enter a string: abbaaba
Don't Accept
Process returned 0 (0x0)   time : 9,710 s
Pres any key to continous.
```

(2) finding ε- closure for NFA with ε- ~~NOTES~~ MOVES

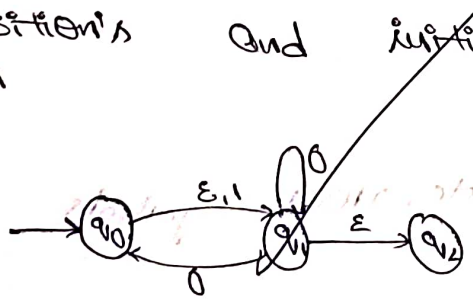
Aim -

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

Algorithm -

* Get the following as input from the user
  i. NO. of states in the NFA
  ii. NO. of Symbols in the input alphabet Including ε
  iii. Input Symbols
  iv. Number of final states and their names.

* Declare a 3-dimensional matrix to store the transition's and initialize all the entries with 1



* Initializing a two-dimensional matrix ε-closure with -1 in all the entries

* ε-closure of a state q_i is defined as the set of all state's that can be reached from state q_i using only ε-transition

* for every state i, find ε-closure as follows
If there is an ε-transitions from state i to state j, add j to the matrix ε-closure.
* for every state, print the ε-closure value

the function find ε-closure

This function finds ε-closure of a state recursively by tracing all the ε-transitions

Program:

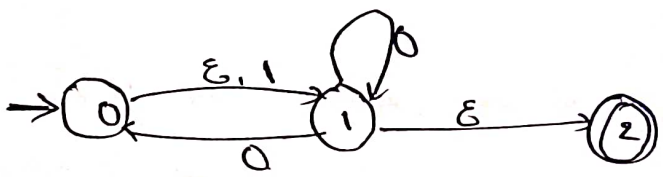
```
#Include <stdio.h>
#Include <string.h>
int trans_table [10][5][3].
Char Symbol [5]. a.
int e_closure [10][10], ptr, state.
void find_e_ closure (int x);
int Main
{
int i, j, k, n nom_state., nom_ symbols
for (i=0; i<10; i++)
{
for (j=0; j<5; j++)
{
trans_table [i][j][k] = -1;
}}}
```

```
num_states = 3
num_symbols = 2.
 Symbols[10] = 'c';
n=1;
trans_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_states; i++)
  {
  if (trans_table[i][0][0] == -1)
  continue.
  else
   {
   state = i
   ptr = 1;
   find_e_closure(i);
  }}
  for (i=0; i<num_states; i++)
  {
   if (e_closure[i][j] != -1)
  {
```

```
void find_c_closure (int r)
{
int i, j, y(10), num_trans;
i=0;
while (trans_table [r][0][i] != -1)
{
y(i) = trans = 1.
{
c_closure (state)[pti] = y[j]
pti++;
find_c_closure (y[j]);
}}
```

Example -

find ε-closure for all the state for the NFA with ε-moves given Below.



Transition table -

Input/Output	ε	0	1
→0	1	-	1
1	2	{0,1}	-
2	-	-	-

Output –

How many states in NFA with ε-move : 3

how many symbols in the input alphabet including ε : 3

Enter symbols without space. Give ε first : ε 1

Enter the transitions from state 0 for input 0 : 0

how many transitions from state 0 for input 1 : 1

Enter transitions : 1 from state 1 for the input : 2

how many transactions from state 2 for in input 1 : 0

ε - closure {0} = {0, 1, 2, }

ε - closure {1} = {1, 2, }

ε - closure {2} = {2, }

1

3) Checking whether a string Belogs to a Grammer.

Aim -

To write a c program to check whether a string belongs to the grammer.

$S \rightarrow 0A1$

$A \rightarrow 0A|1A|\varepsilon$

Language defind by the grammer

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

Algorithim -

* Get the input string from the user.
* find the length of the string.
* check whether all the symbols in the input are either 0 or 1 if so, print. "string is valid" and go to step 4.
  print "string not print" and quit the program.
* if the first symbol is 0 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 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) != 'i')
     {
   flag=0
    }

}
if (flag != 1)
     printf ("string is not valid\n")
if (flag == 1)
{
   if (s(0) == 0' && s(-1) == 'i');
  else
     print ("string is not accepted\n")
  }
}
```

Output -

Enter a string to check : 01010111101.
String is accepted.
Process returned 0 (0x0) Execution time = 25.79.

Enter a string to check : 01110101010
String is not accepted.
Process returned 0 (0x0) Execution time : 7.039 s
Press any key to continue.

Enter a string to check : abbababa.
String is not valid.
Process returned 0 (0x0) Execution time : 8.0638 s

3) checking wheather a string Belongs to a grammer.

Aim -

To write a c program to check whether a string belogs to the grammer

$S \rightarrow 0S0 / 1S1 / 0 / 1 / \varepsilon$.

Language defined the grammer -

set of all strings over $\Sigma = \{0,1\}$ that are palindrome.

Algorithm -

* Get the input string from the user.
* find the length of the string Let it be n.
* check wheather all symbols in input are either 0 or 1 if so print " string is valid And go to step 4.
* if the 1st symbol And nth symbol are the 2nd symbol And (n-1)th symbol are the same. And so on. then the given string is palindrome so print " string is palindrome. so print " string accepted". Otherwise, print "string not accepted.

Program -

```
#include <stdio.h>
#Include <string.h>
Void main()
{
  Char s[100];
  Int i,flag, flag1,a,b;
  int l;
  Printf("Enter a string to check:");
  scanf("%s",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)
{
    flag1=1;
    a=0; b=l-1;
  while(a!=(l/2))
{
```

```
}
Else
    Printf(" The string is not a palindrome \n");
    Printf(" string is not accepted \n");
    }
  }
}
```

Output.

enter a string to check : 11010111
The string is a palindrome.
string is accepted

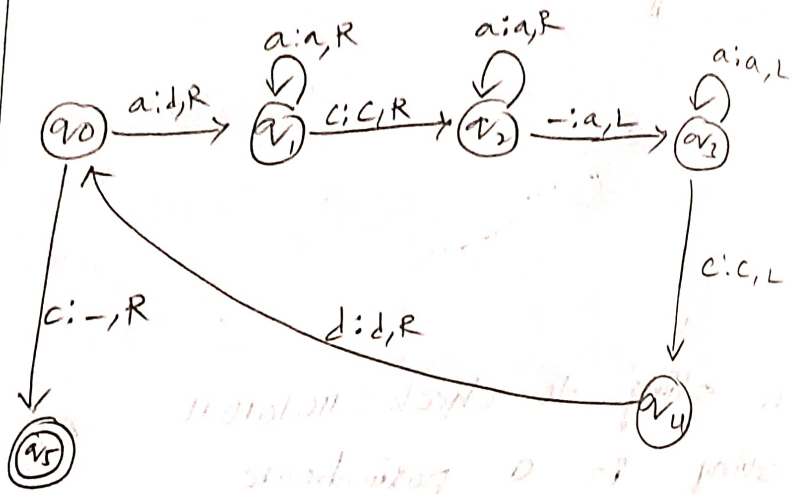
Process returned 0 (0x0) Execution time : 14.737s
press any key to continue.

Enter a string to check : 10000010ab.
string is not valid.
process returned 0 (0x0). Execution time : 6.766s.
press any key to continue.

Enter a string to check : 10000010ab.
string. is not valid.
process. returned. 0 (0x0) Execution time : 6.766s.
press any key to continue.

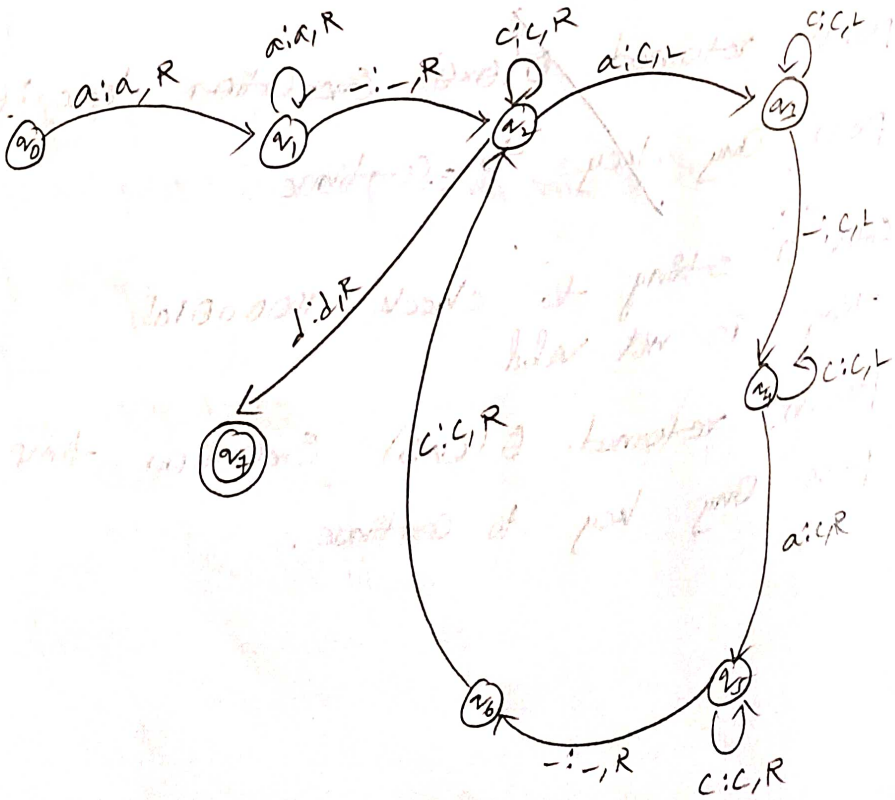
10

Input → w = abba bcb aac aaa



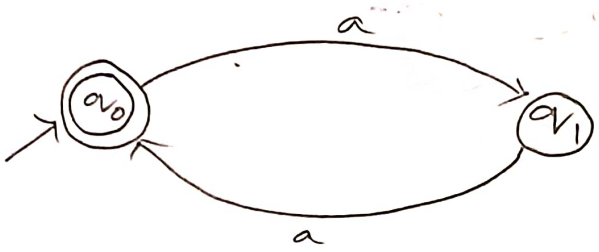
11

Input → aaa-aa



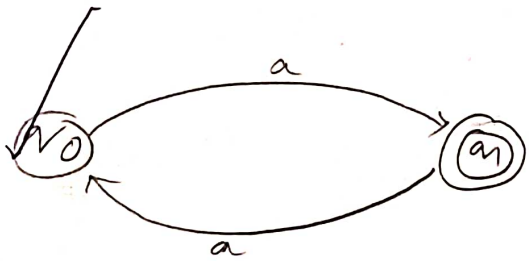
(12)

Input :- aa



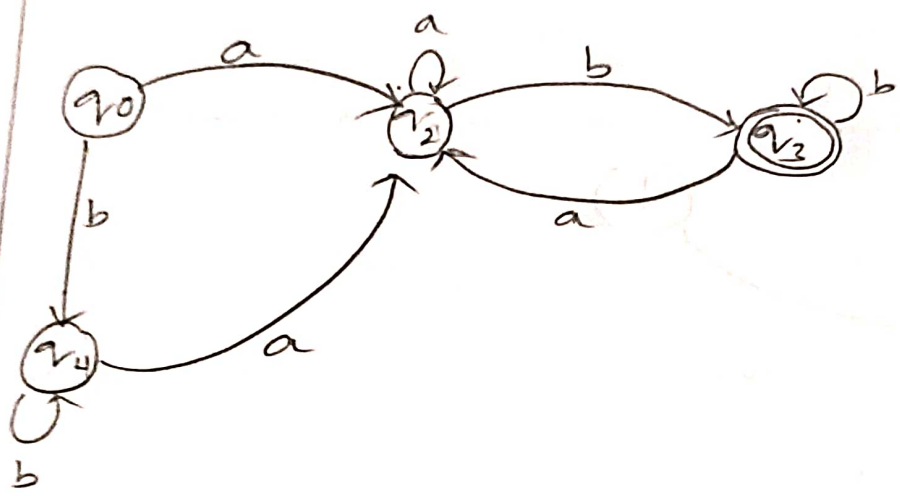
(12)

Input :- aaa



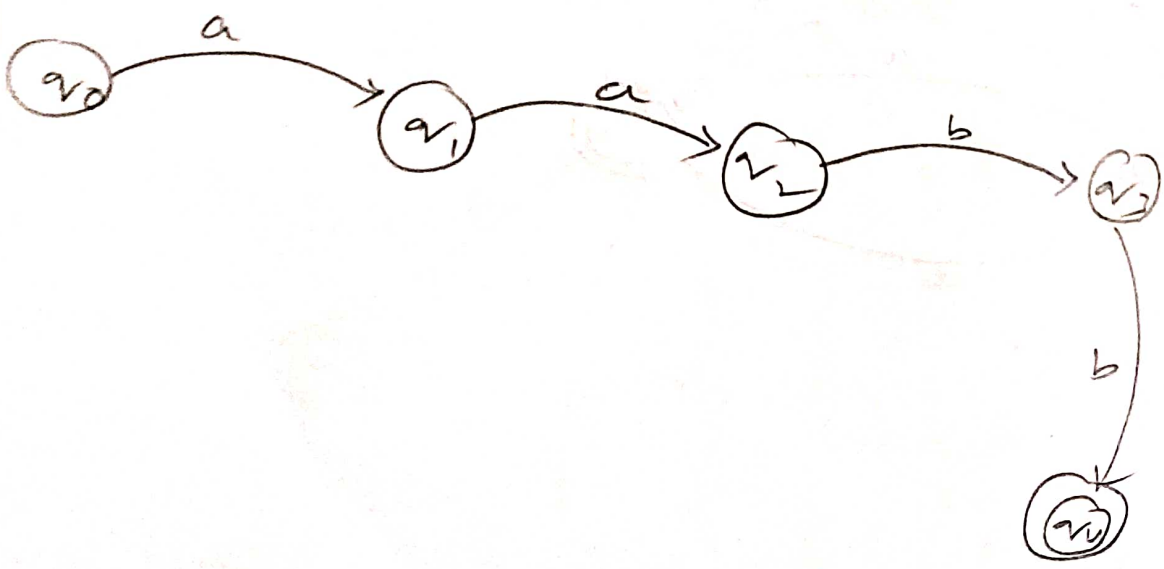
(14)

Input : ^aabab^b

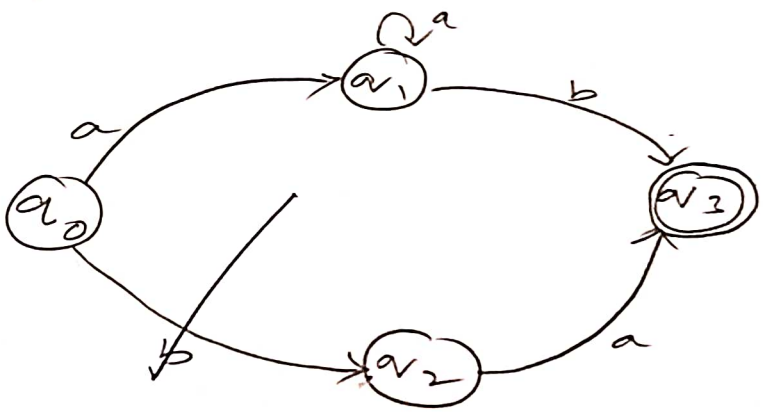


(15)

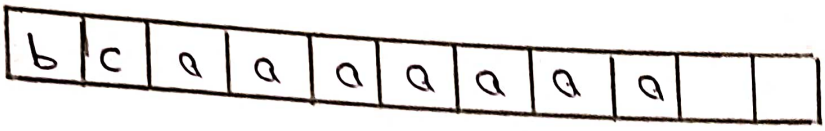
Input : aabb



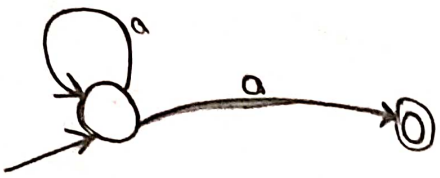
Input: a or b $\{a, b\}$

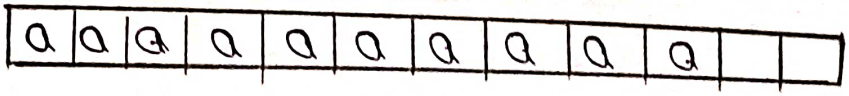


1) Design DFA to Accept bcaaaaaaaa, bc and c

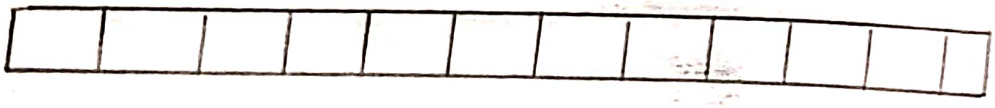


2) Design NFA to accept aaaaaa.

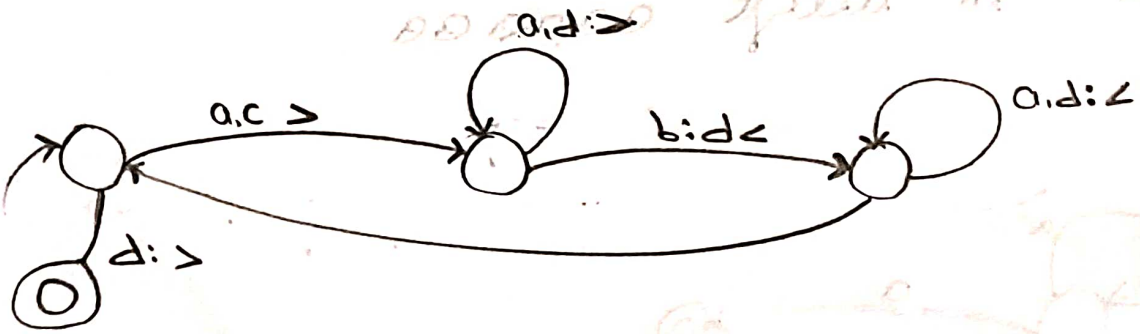


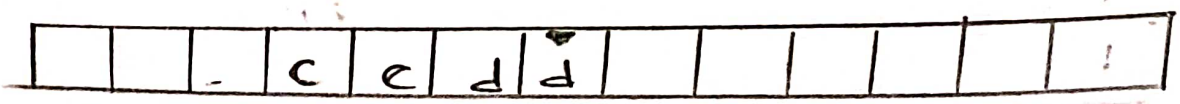


3) Design PDA for the input $a^n b^n$

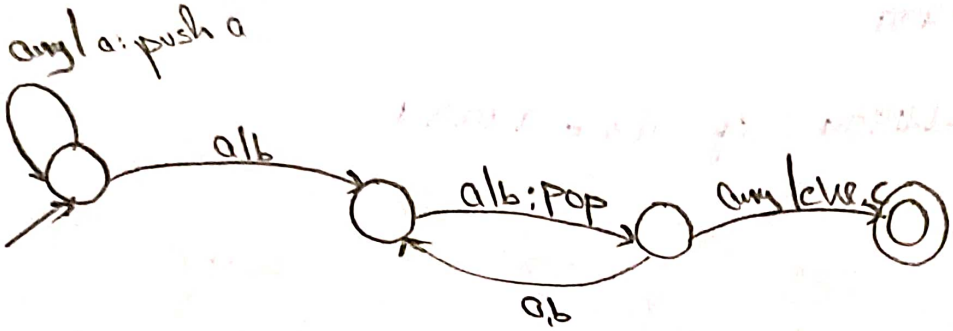


4) Design TM for input $a^n b^n$

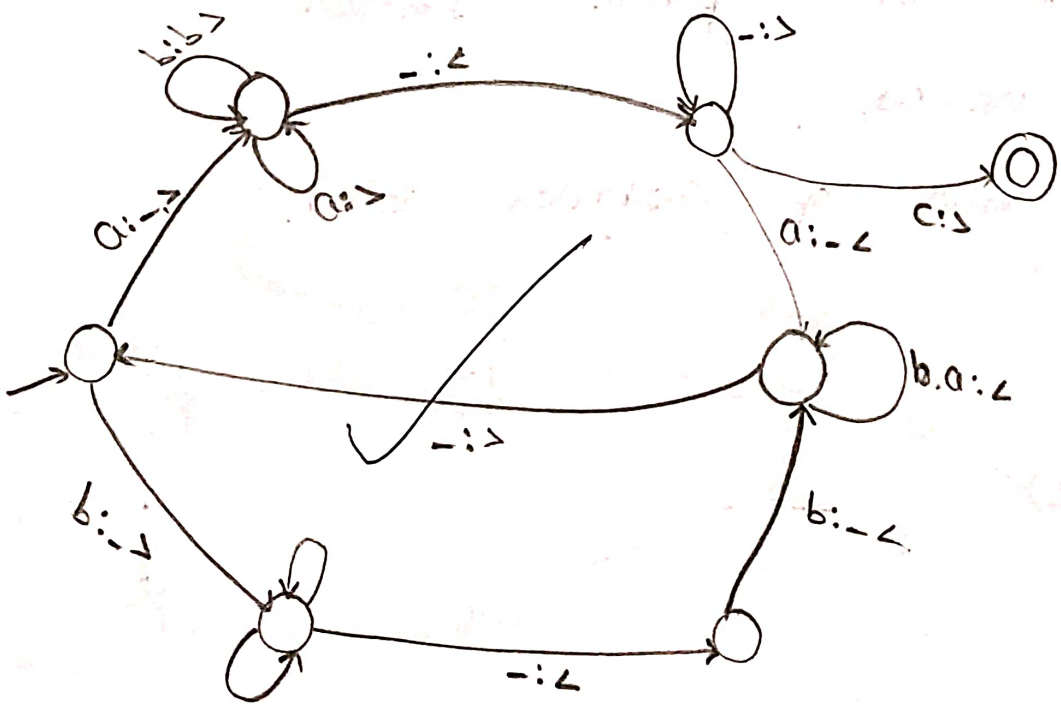




5) Design PDA for input aabbbbc ($L = a^n b^{2n}$)



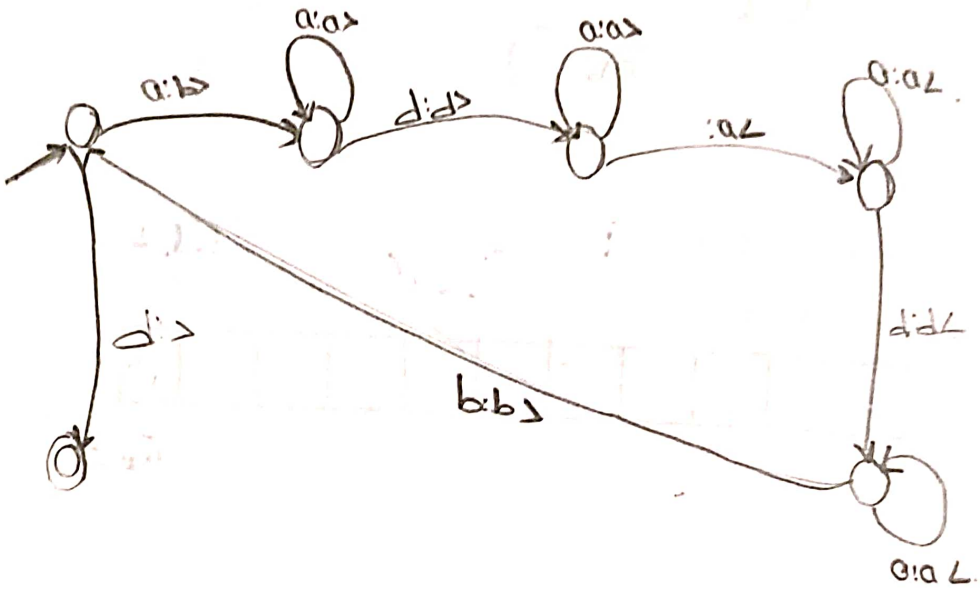
6) TM Simulation for palindrome w = ababa c

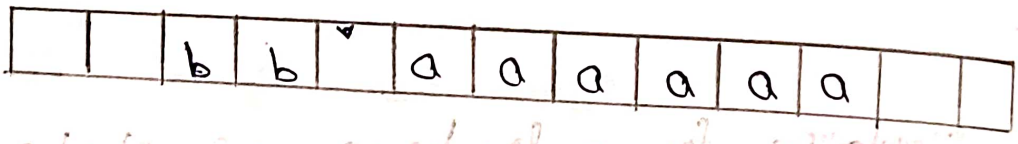


7) Design TM to perform addition of following

$w = aa + aaa$

After Addition of $\Delta s = aaaaa$

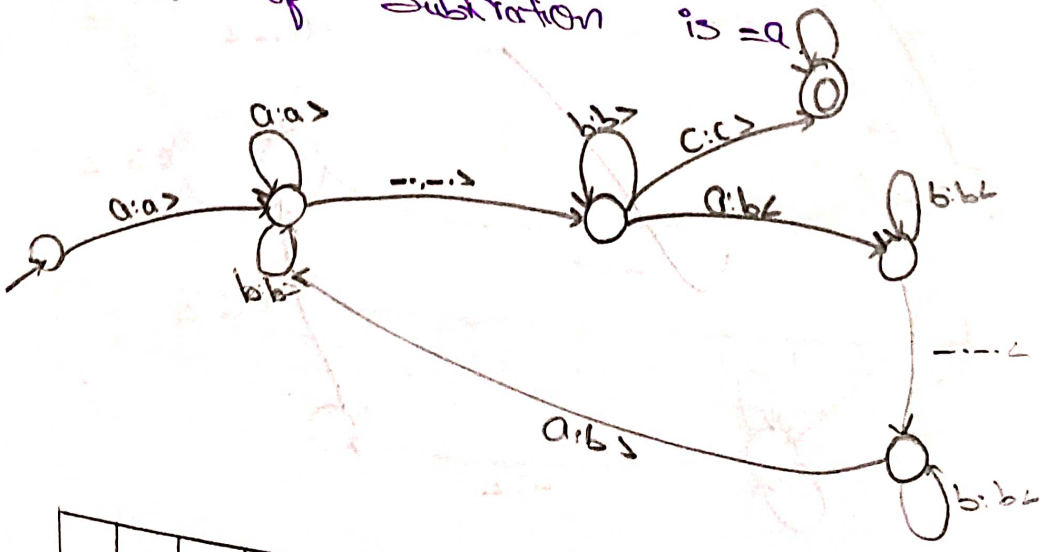




8) Design TM to perform Subtraction

$w = aaa - aa$.

The result of Subtraction is $= a$



9) Design TM to Perform string comparision.

w = abc aba