

1) Deterministic finite Automata(DFA)

Aim:- To write a C program to simulate a

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

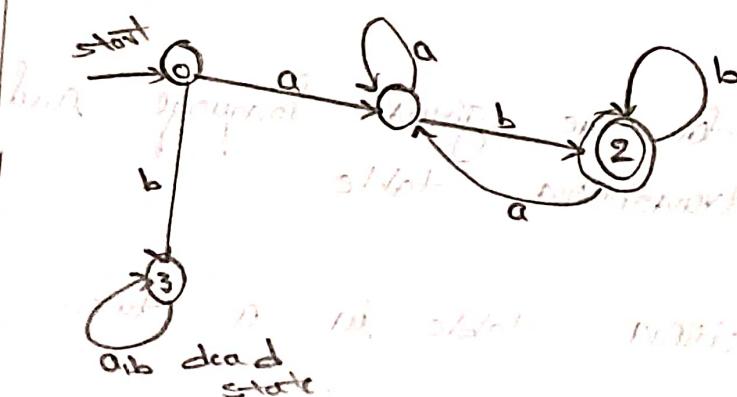
Algorithm

- * 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 strings over $\Sigma = \{a, b\}$ that start with a and end with b .

Design the DFA.



State / Input	a	b
$\rightarrow 0$	1	3
0/a	1	2
0/b	2	2
1/a	0	1
1/b	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 i = strlen(input_string);
```

```
for (i=0; i<j++;)
```

```
{
```

```
If (input_string[i] == 'a')
```

```
next_state = trans_table[present_state][0];
```

```
Else
```

```
invalid = 1; // mark first bad character
```

```
present_state = next_state;
```

```
}
```

```

Present-state = next-state
{
    if (invalid == 1)
    {
        Print ("Invalid input.");
    }
    Else If (Present-state == final-state) and
        Print ("Accept \n")
    Else
        Print ("Don't Accept \n");
}

```

Output -

Enter a string: abab

Accept

Process returned 0 (0x0) time : 7.313s

Press any key to continue.

Enter a string: abbaabaa

Don't Accept

Process returned 0 (0x0) time : 9.710s

Press any key to continue.

- ② finding ϵ -closure for NFA with ϵ -transitions

Aim - To implement ϵ -closure algorithm.

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

Algorithm -

* Get the set of states of NFA.

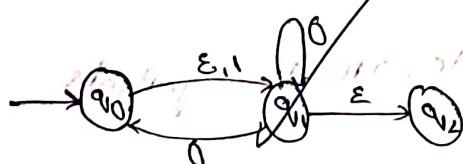
i. NO. of States in the NFA

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 transitions and initializing all the entries with 1.



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

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

* for every state i , find ϵ -closure

If there is an ϵ -transitions from i ,

j. add j to the matrix closure

* for every state, print the ϵ -closure

The function find ϵ -closure

This function finds ϵ -closure

State recursively by tracing all the ϵ -transitions

Program:

```
#include <stdio.h>
#include <iostream.h>
#include <string.h>

int trans-table[10][5][3];
char symbol[5].a;
int num_state, num_symbol;
int e-closure[10][10], pth, state;

void find_e-closure (int x);
int main()
{
    int i, j, k, m, num-state, num-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(0) = 'c';

n=1;

trans-table (0)(0)(0) = 1;

for (i=0; i<10; i++)

{

for (j=0; j<10; j++)

{

c-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-closure (i);

}

for (i=0; i<num-states; i++)

{

if (c-closure (i)(j) == -1)

{

Void find- ϵ -closure (List r)

{

int i, j, y[10], num_trans;

i=0;

while Trans-table [x][0][i] != -1

{

y[i] = trans = 1,

{

ϵ -closure (state) [p*i*] = y[i]

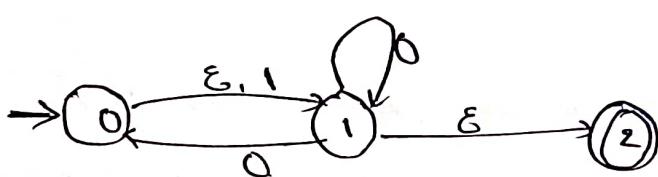
p*i* ++;

find- ϵ -closure (y[j]);

}

Example -

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



Transition table -

<u>Input/Output</u>	<u>e</u>	<u>0</u>	<u>1</u>
$\rightarrow 0$	1 -	1	
1	2 (0,1)	-	
2	- -	-	

Output -

How many states in NFA with C-Move: 3

How many symbols in the input alphabet including: 3

Enter symbols without space. Given C-first : C1.

Enter the transitions from state 0 for input 0:0.

How many transitions from state 0 for input 1:1.

Other transitions. 1 from state 1 for the input: 2.

How many transitions, from state 2 for input 1:0.

$$C\text{-closure } \check{V} = \{0, 1, 2, \cdot\}$$

$$C\text{-closure } (1) = \{1, 2, \cdot\}$$

$$C\text{-closure } (2) = \{2, \cdot\}$$

3) Checking whether a string belongs to a grammar.

Aim -

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

$S \rightarrow 0A1$

$A \rightarrow 0A1 \cup \epsilon$

Language defined by the grammar

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

Algorithm -

- * 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.
- * if "String not valid" then print "String not accepted" 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 <cs50.h>
#include <string.h>

int main()
{
    char s[100];
    int i, flag = 1;

    printf("Enter a string to check.");
    scanf("%s", s);

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

Output -

Enter a string to check : 01010111101.
String is accepted.

Process returned 0 (0x0) Execution time = 25.71s

Enter a string to check : 01110101010
String is not accepted.

Process returned 0 (0x0) Execution time = 7.039s
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 whether a string belongs to L^0 .
grammer.

Aim -

To write a C program to check whether
a string belongs to the grammar
 $S \rightarrow 0=0 \mid 1=1 \mid 011011 \in \Sigma$.

language defined the grammar

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 whether 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 n^{th} symbol,
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;
    Point f("Enter a string to check:");
    Scanf ("%s", s);
    flag = 1;
    for (i=0; i<l; i++)
    {
        If (s[i] == '0' && s[i+1] == '1') flag = 0;
        If (s[i] == '1' && s[i+1] == '0') flag = 1;
    }
    If (flag == 1) Point f("String is valid");
    Else Point f("String is not valid\n");
}
Int a, b;
While (a != (b/2))
{
    a = 0; b = 1 - 1;
}
```

{

Else

Print ("The string is not a palindrome\n");

Print ("String is not accepted\n");

}

}

}

Output.

Enter a string to check : 110101011

The string is a palindrome.

String is accepted

Process returned 0 (0x0) Execution time : 14.737 s

Press Any key to continue.

Enter a string to check : 1000001ab.

String is not valid.

Process returned 0 (0x0) Execution time : 6.766 s

Press Any key to continue.

Enter a string to check : 1000001ab.

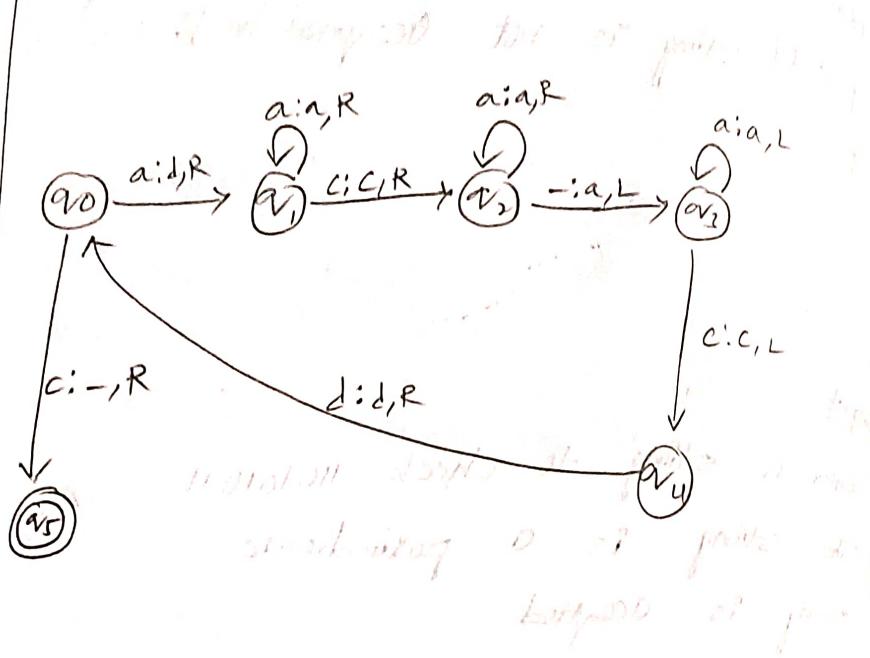
String is not valid.

Process returned 0 (0x0) Execution time : 6.766 s

Press Any key to continue.

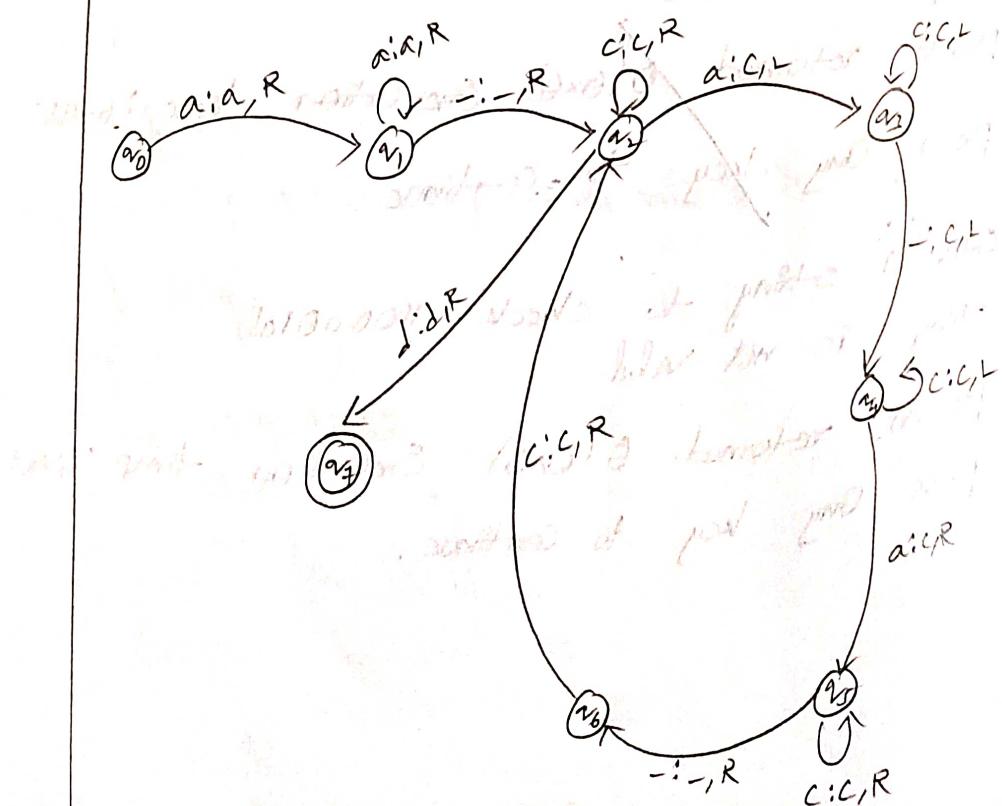
(10)

Input $\Rightarrow w = abaa \text{ lex } - aacaa \text{ aam } \rightarrow$



(11)

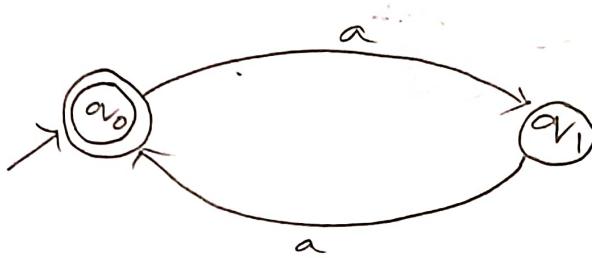
Input $\Rightarrow aaa - aa$



(12)

Input: aa

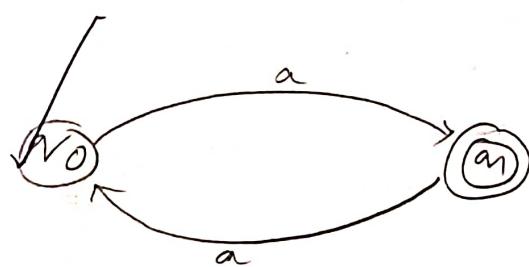
directed graph



(12)

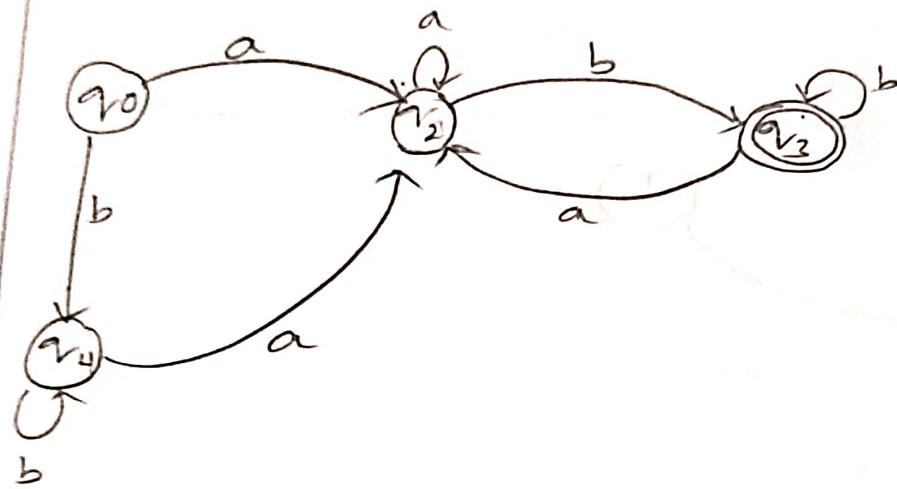
Input: aaa

directed graph



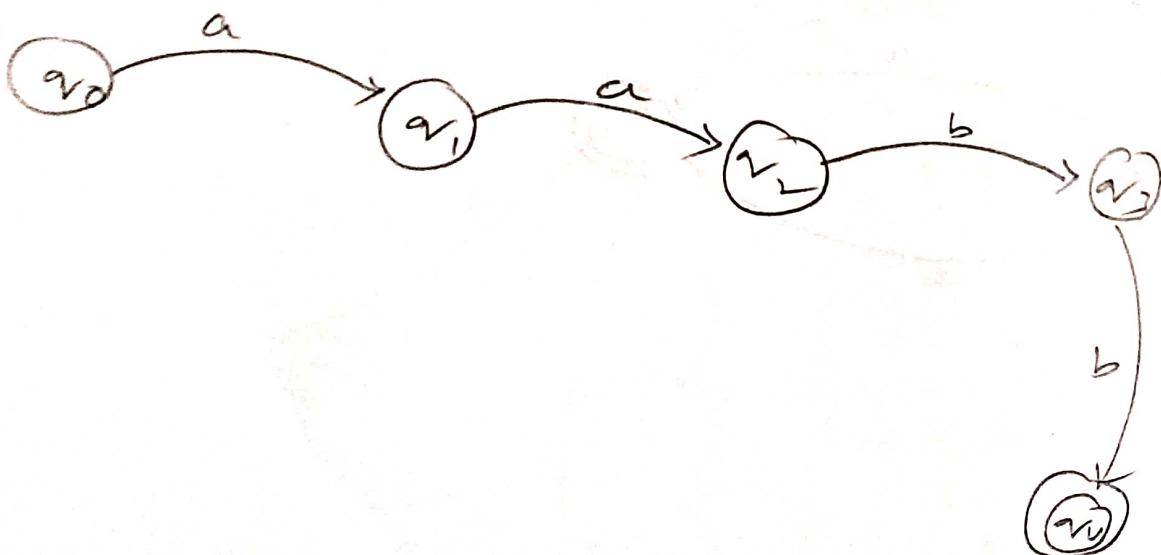
(14)

Input : $a^m b^n b$

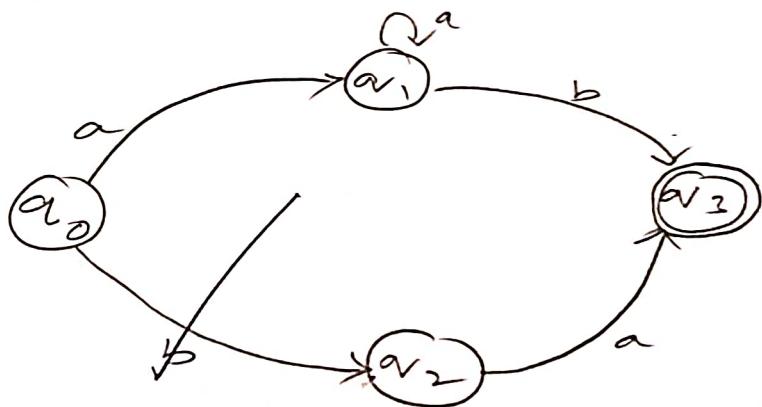


(15)

Input : $a^m b^n$



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

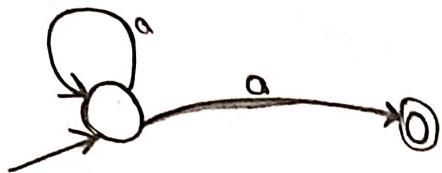


5) Design c DNA to Acetyl bcooooooo, bc, and

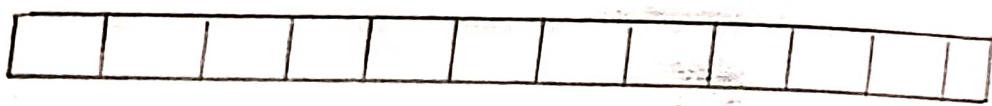
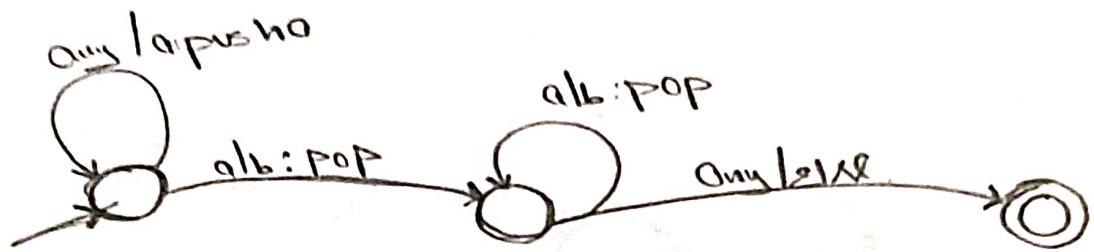


b c a a a a a a a a

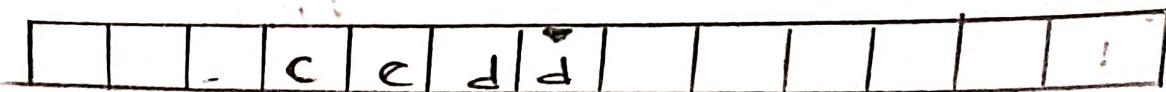
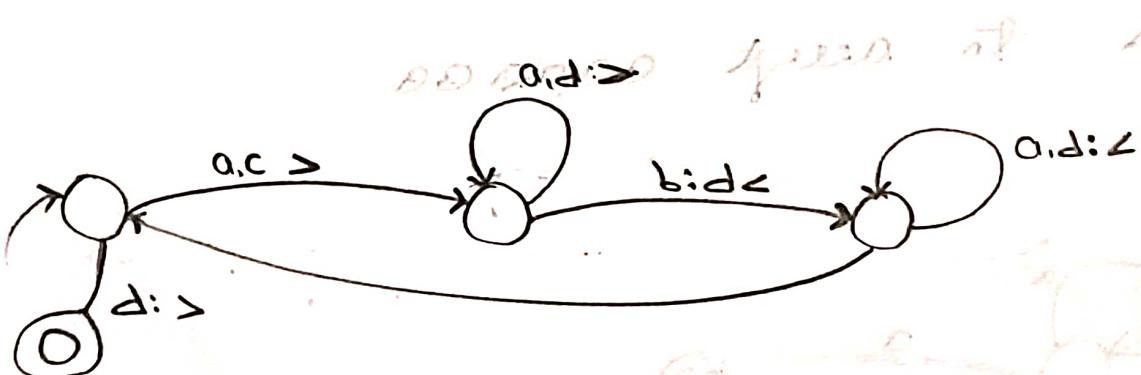
2) Design NFA to accept 000000



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

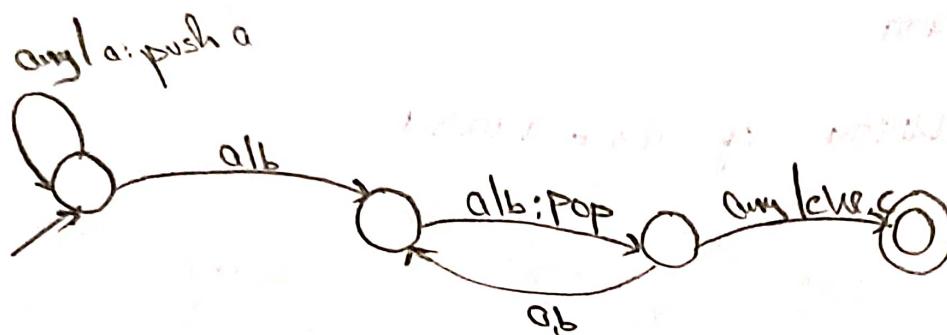


4) Design -Tm for input Qⁿab^m

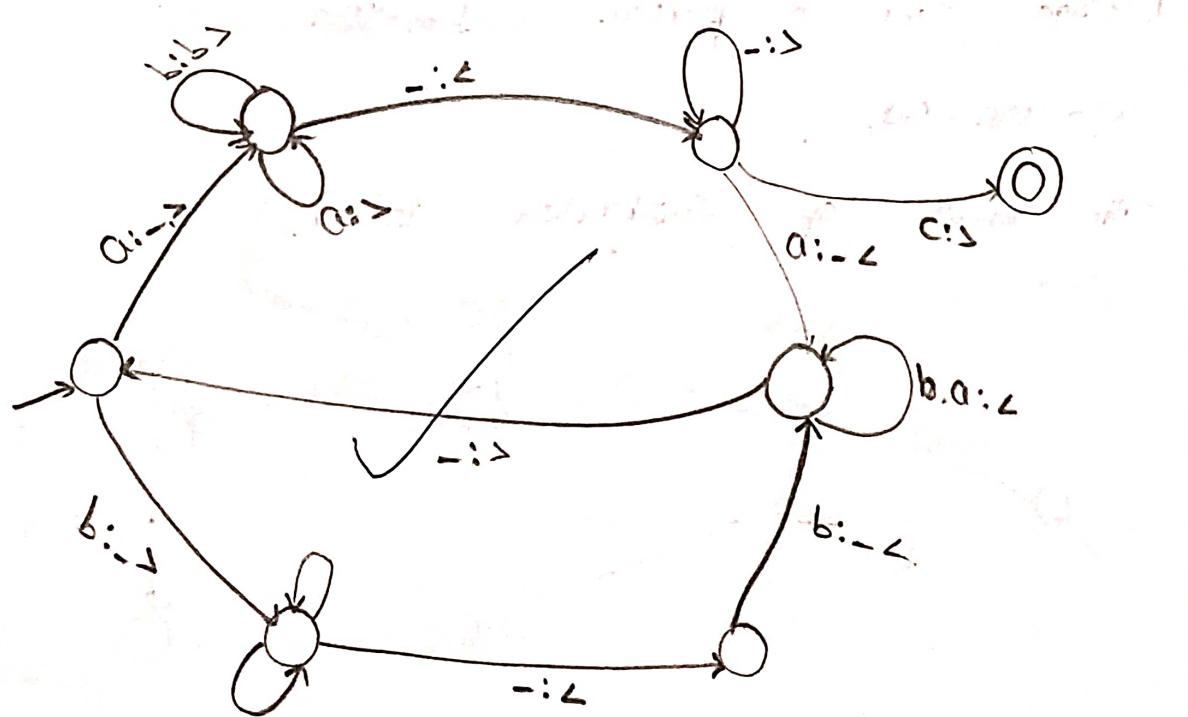


Full Stack Developer

5) Design PDA for input $aabbcc$ ($L = a^n b^n c^n$)

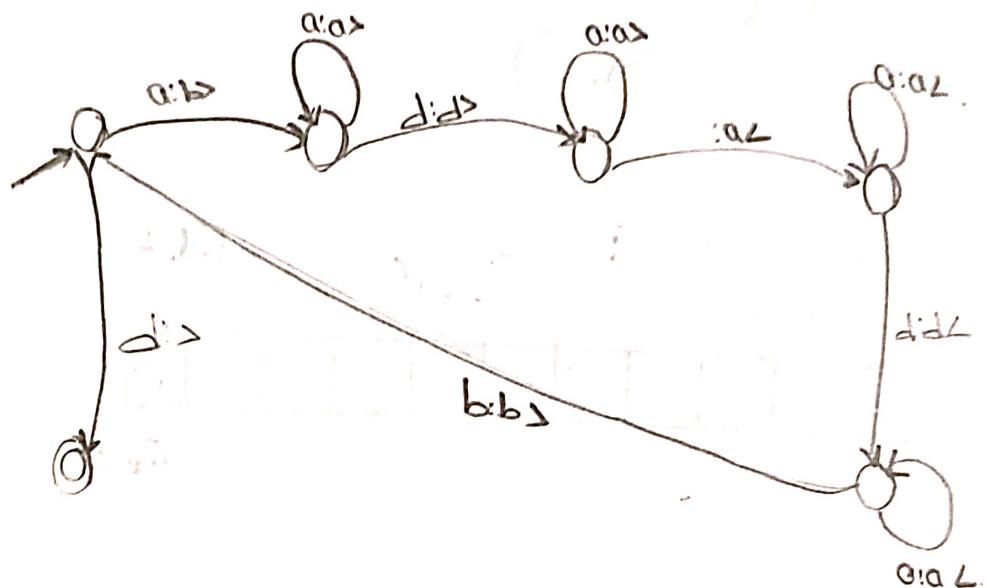


6) TM simulation for Palindrome $w = ababa c$



7) Design TM to perform addition
 $w = ab + cd$

After Addition of $ab = aaaaa$



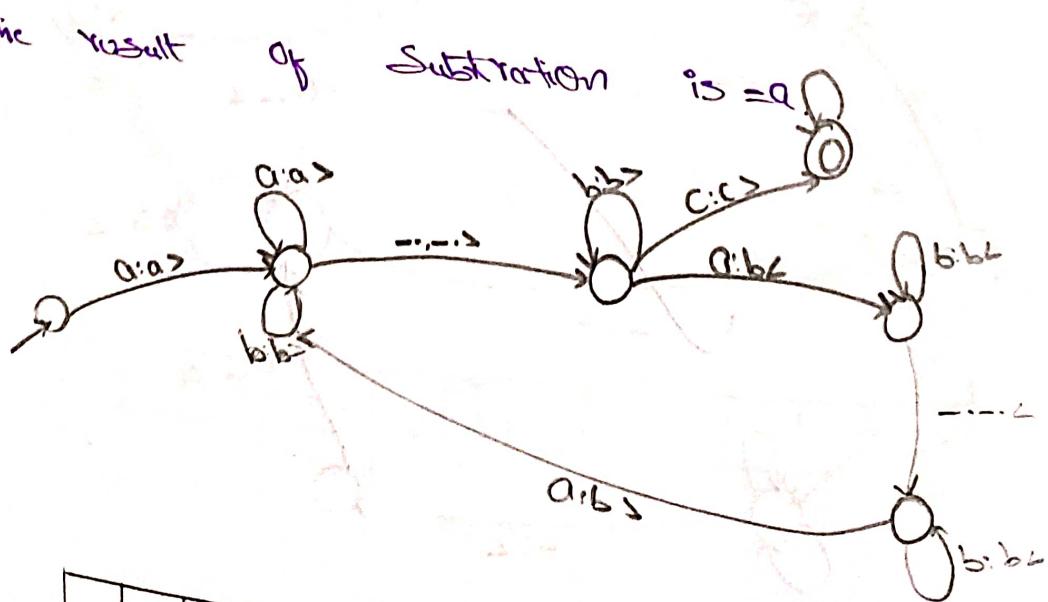
	b	b	a	a	a	a	a	a	
--	---	---	---	---	---	---	---	---	--

2) Design TM to perform subtraction $w = ab - cd$

Design TM to perform Subtraction

$$w = ab - cd.$$

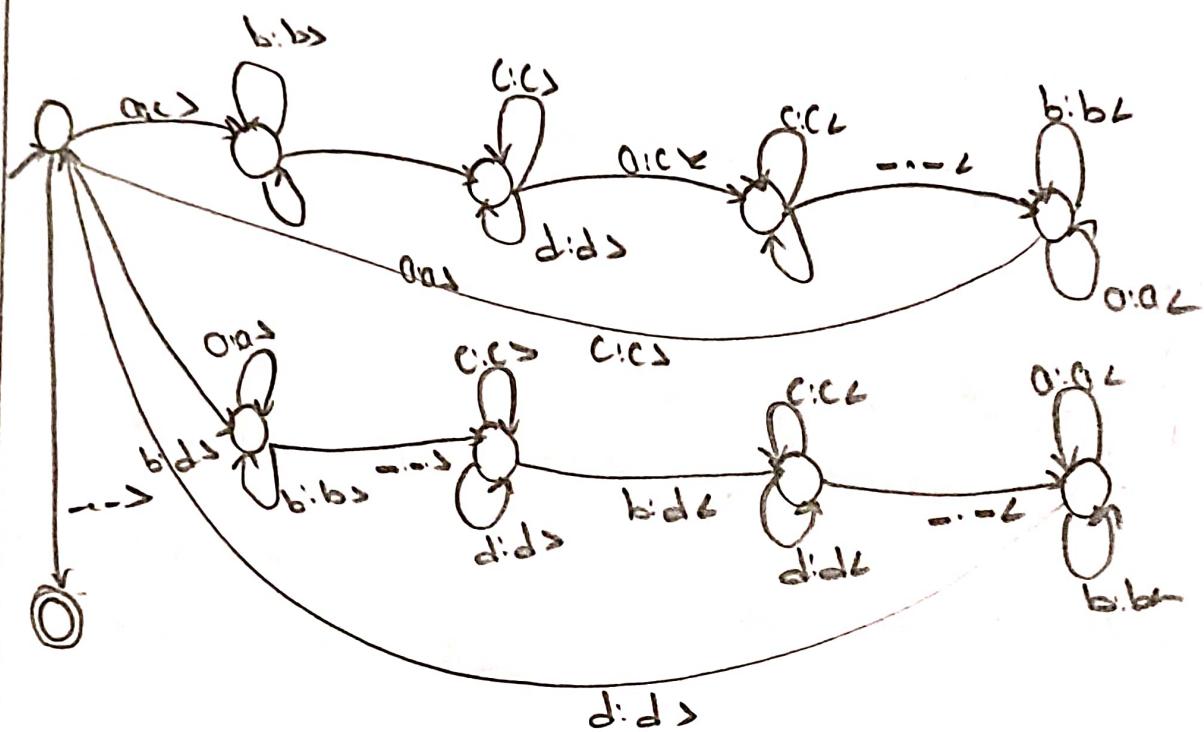
The result of Subtraction is $= ab$



a	b	b							
---	---	---	--	--	--	--	--	--	--

a) Design Tm to Perform string comparison.

$w = abc \ abab$



	c	d	c		d	c				

