Laboratory Manual

Theory of Computation
(CS-501)

For

Third Year Students
Department: Computer Science & Engineering

INSTITUTE OF TECHNOLOGY & MANAGEMENT
GWALIOR • MP • INDIA

Department of Computer Science and Engineering

Vision of CSE Department:

The department envisions to nurture students to become technologically proficient, research competent and socially accountable for the welfare of the society.

Mission of the CSE Department:

I. To provide high quality education through effective teaching-learning process emphasizing active participation of students.

II. To build scientifically strong engineers to cater to the needs of industry, higher studies, research and startups.

III. To awaken young minds ingrained with ethical values and professional behaviors for the betterment of the society.

Program Educational Objectives:

Graduates will be able to

I. Our engineers will demonstrate application of comprehensive technical knowledge for innovation and entrepreneurship.

II. Our graduates will employ capabilities of solving complex engineering problems to succeed in research and/or higher studies.

III. Our graduates will exhibit team-work and leadership qualities to meet stakeholder business objectives in their careers.

IV. Our graduates will evolve in ethical and professional practices and enhance socioeconomic contributions to the society.

INSTITUTE OF TECHNOLOGY & MANAGEMENT
GWALIOR • MP • INDIA

Program Outcomes (Pos):

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering Fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

INSTITUTE OF TECHNOLOGY & MANAGEMENT
GWALIOR • MP • INDIA

Course Outcomes

Theory Of Computation(CS-501)

CO1 :	Explain the basic concepts of switching and finite automata theory & languages.
CO2 :	Relate practical problems to languages, automata, computability and complexity.
CO3 :	Construct abstract models of computing and check their power to recognize the languages.
CO4 :	Analyse the grammar, its types, simplification and normal form.
CO5 :	Interpret rigorously formal mathematical methods to prove properties of languages, grammars andautomata.

Course	Course Outcomes	CO Attainment	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	Understand and apply concept of finite state machine to design a deterministic finite automata and non-deterministic finite automatafor a problem		2	2	1	1	1				1				2		1
CO2	Analysis and Apply ardern's theorem to compute regular expression for a given deterministic and non deterministic finite automata.		2	1	1									1		1	
CO3	Analyze whether the given language is regular or not, equivalence of languages accepted by Push Down Automata and languages generated by context free grammars.			0		1	1									1	
CO4	Analysis and comprehension between Deterministic finite automata, non Deterministic finite automata, Push Down Automata, Turing machine on the basis of their power.		1	1		1					1				1		
CO5	Understand and apply conceptof Turing machine to design machine for a given problem.		1	1			1							1	2		

INSTITUTE OF TECHNOLOGY & MANAGEMENT
GWALIOR • MP • INDIA

List of Program

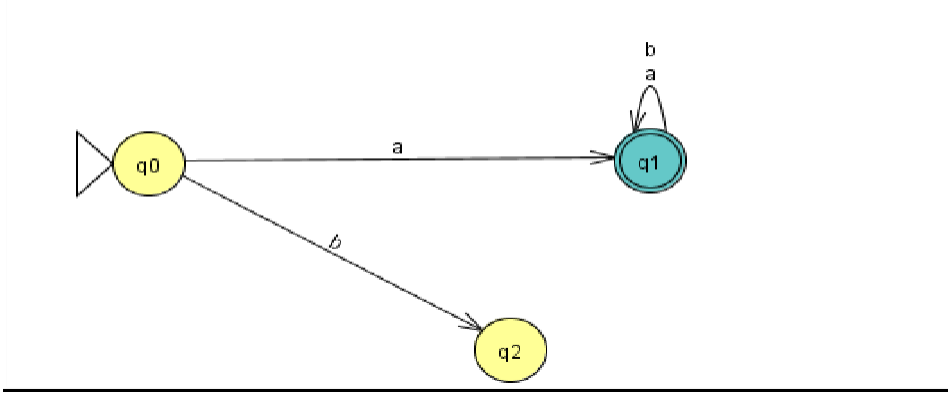
Program-1

Design a DFA over the input set {a,b} accept all the string starting with symbol a.

Explanation: DFA or Deterministic Finite Automata is a finite state machine which accepts a string(under some specific condition) if it reaches a final state, otherwise rejects it. In DFA, there is no concept of memory, therefore we have to check the string characterby character, beginning with the 0th character. The input set of characters for the problem is {a, b}. For a DFA to be valid, there must a transition rule defined for each symbol of the input set at every state to a valid state.

DFA Machine that accepts all strings that all the string starting with symbol a.For the above problem statement, we must first build a DFA machine. DFA machine is similar to aflowchart with various states and transitions. DFA machine corresponding to the above problem is shown below:

Diagram:



Code:

```
  def q0(s,i):
if len(s) == i: print("Rejected") return;
elif (s[i] == 'a'):q1(s,i+1);
else: qd(s,i+1)
  def qd(s,i):
if len(s) == i: print("Rejected") return;
elif (s[i] == 'a'):qd(s,i+1);
  else:
  qd(s,i+1)
```

```
def q1(s,i):
if len(s) == i:
print("Accepted") return;
elif (s[i] == 'a'):q1(s,i+1);
else:
q1(s,i+1)
if _name_ == "_main_":
s = input("Enter a string : ")q0(s,0);
```

Output:

```
Enter a string : abbaabb
Accepted
```

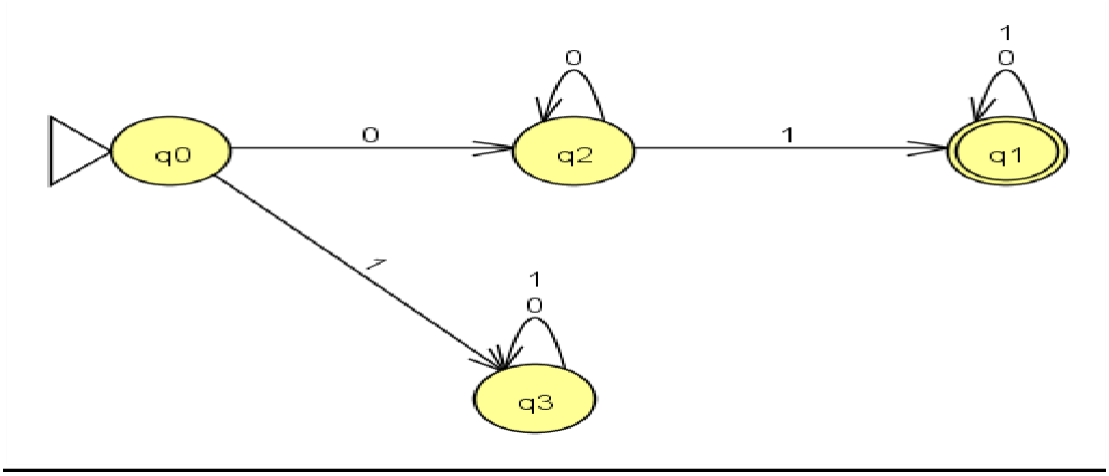
Program-2

Design a DFA over the input set {0,1}. accept all the Strings Starting with 0 and ending with 1.

Theory:

- **Step 1** − q0 is the initial state on input '0' it goes to q2, and input 1 at q2 goes to q1 which is the final state, and '01' string is accepted.
- **Step 2** − q0 on '1' goes to q3 which is dead state because for q3 there is no path to reach to the final state.
- **Step 3** – q2 on input '0' will remain in q2.

Diagram:



Code:

```
  def q0(s,i):
if len(s) == i: print("Rejected") return;
elif (s[i] == '1'):qd(s,i+1);
else: q1(s,i+1)
  def q1(s,i):
  if len(s) == i:
  print("Rejected") return;
```

```
elif (s[i] == '1'):q2(s,i+1);


else: q1(s,i+1)
  def qd(s,i):
  if len(s) == i:
  print("Rejected") return;
elif (s[i] == '1'):qd(s,i+1);
  else:
  qd(s,i+1)

  def q2(s,i):
if len(s) == i: print("Accepted") return;
elif (s[i] == '1'):q2(s,i+1);
  else:
  q1(s,i+1)

if__name___== "__main___": s = input("Enter a string : ")q0(s,0);
```

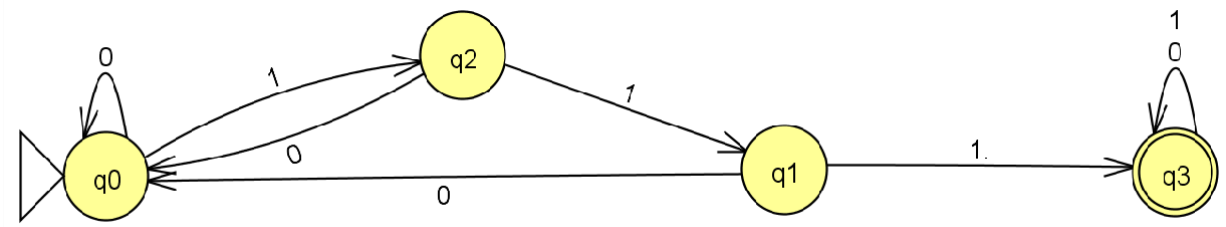
Output:

```
Enter a string : 011
Accepted
```

Program-3

Aim: Design a Program for creating machine that accepts three consecutiveones.

Diagram:



Code:

```
 def q0(s,i):
if len(s) == i: print("Rejected")return;
elif (s[i] == '1'):q1(s,i+1);
else: q0(s,i+1)

 def q1(s,i):
if len(s) == i: print("Rejected")return;
elif (s[i] == '1'):q2(s,i+1);
else: q0(s,i+1)

 def q2(s,i):
if len(s) == i: print("Rejected")return;
elif (s[i] == '1'):q3(s,i+1);
else: q0(s,i+1)


 def q3(s,i):
if len(s) == i: print("Accepted")return;
elif (s[i] == '1'):q3(s,i+1);
 else:
 q3(s,i+1)

if__name__== "__main__":s = input("Enter a string : ")q0(s,0);
```

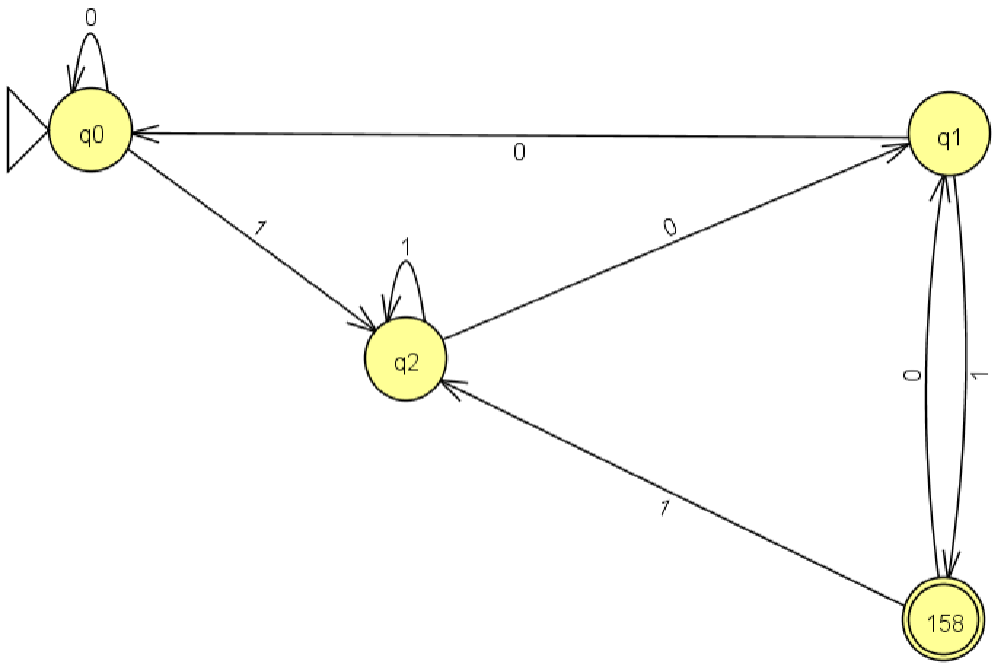
Output:

```
Enter a string : 01110
Accepted
```

Program-4

Design a program for creating a machine that accepts the string always ending with 101.

Diagram:



Code:

```
 def q0(s,i):
if len(s) == i: print("Rejected")return;
elif (s[i] == '1'):q1(s,i+1);
else: q0(s,i+1)

 def q1(s,i):
if len(s) == i: print("Rejected")return;
elif (s[i] == '1'):q1(s,i+1);
else: q2(s,i+1)

 def q2(s,i):
if len(s) == i: print("Rejected")return;
```

```
elif (s[i] == '1'):q3(s,i+1);


  else:
  q0(s,i+1)

  def q3(s,i):
if len(s) == i: print("Accepted")return;
elif (s[i] == '1'):q1(s,i+1);
  else:
  q2(s,i+1)

if__name__== "__main__":s = input("Enter a string : ")q0(s,0);
```

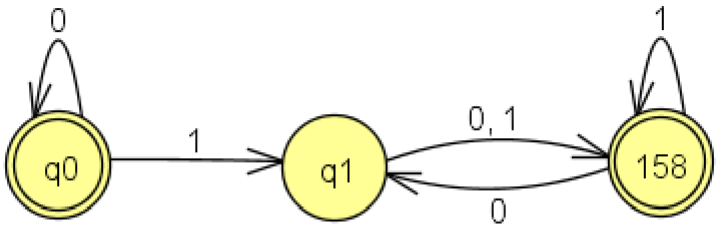
Output:

```
    q0(s,0);

Enter a string : 11101
Accepted
```

Program- 5

Design a Program for Mode 3 Machine.

Diagram:



Code:

```
 def q0(s,i):

if len(s) == i: print("Accepted")return;
elif (s[i] == '1'):q1(s,i+1);
else: q0(s,i+1)

  def q1(s,i):
if len(s) == i: print("Rejected")return;
elif (s[i] == '1'):q0(s,i+1);
else: q2(s,i+1)

  def q2(s,i):
if len(s) == i: print("Rejected")return;
elif (s[i] == '1'):q2(s,i+1);
  else:
  q1(s,i+1)

if__name__== "__main__":s = input("Enter a string : ")q0(s,0);
```

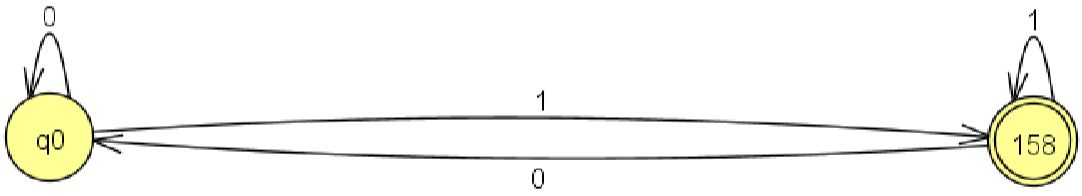
Output:

```
Enter a string : 110
Accepted
```

Program- 6

Design a program for accepting binary numbers divisible by 2.

Diagram:



Code:

```
  def q0(s,i):
if len(s)==i: print("Accepted")return;
  if (s[i]=='1'):
    q1(s,i+1)else:
  q0(s,i+1)

  def q1(s,i):
if len(s)==i: print("rejected")return;
  if (s[i]=='1'):
    q1(s,i+1)else:
  q0(s,i+1)

  if _name_ == "_main_":
  s = input("Enter a string")q0(s,0);
```

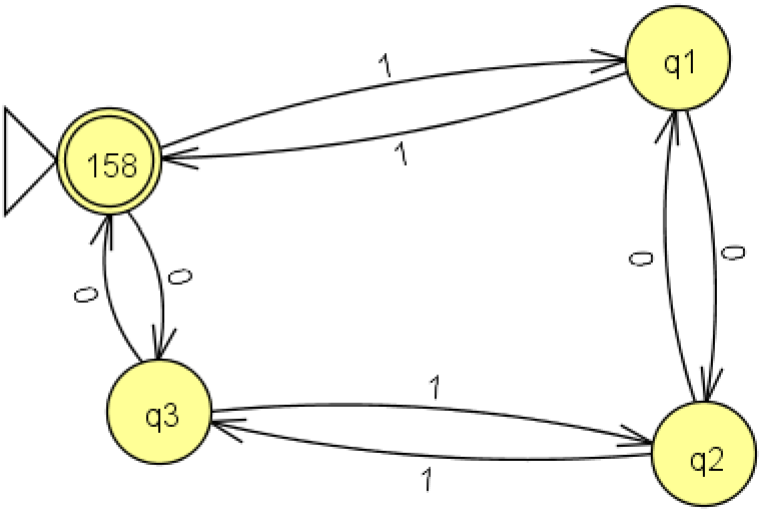
Output:

```
Enter a string : 110
Accepted
```

Program- 7

Design a program for creating a machine which accepts string having even number of 1's and 0's.

Diagram:



Code:

```
 def q0(s,i):
if len(s) == i: print("Accepted")return;
elif (s[i] == '1'):q2(s,i+1);
else: q1(s,i+1)

 def q1(s,i):
if len(s) == i: print("Rejected")return;
elif (s[i] == '1'):q3(s,i+1);
else: q0(s,i+1)

 def q2(s,i):
if len(s) == i: print("Rejected")
 return;
elif (s[i] == '1'):q0(s,i+1);
 else:
 q3(s,i+1)
```

```
  def q3(s,i):
if len(s) == i: print("Rejected")return;
elif (s[i] == '1'):q1(s,i+1);
  else:
  q2(s,i+1)

  if _name_ == "_main_":
  s = input("Enter a string : ")q0(s,0);
```

Output:

```
Enter a string : 11001
Rejected
```

Program- 8

Design a program for creating a machine which counts number of 1's and 0's in a given string

Code:

```
def q0(s,i,c0,c1):
if len(s) == i:
print("Number of 0's = ",c0)print("Number of 1's = ",c1)return;
elif (s[i] == '1'):
c1 = c1 + 1
  q1(s,i+1,c0,c1);else:
c0 = c0 + 1
q0(s,i+1,c0,c1)

def q1(s,i,c0,c1):
if len(s) == i:
print("Number of 0's = ",c0)print("Number of 1's = ",c1)return;
elif (s[i] == '1'):
c1 = c1 + 1
  q1(s,i+1,c0,c1);else:
c0 = c0 + 1
q0(s,i+1,c0,c1)

if _name_ == "_main_":
s = input("Enter a string : ")count_0 = 0
count_1 = 0 q0(s,0,count_0,count_1);
```

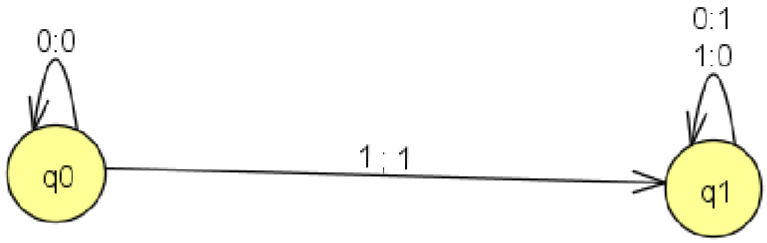
Output:

```
Enter a string : 11010101
Number of 0's =  3
Number of 1's =  5
```

Program- 9

Design a program to find 2's complement of a given binary number.

Diagram:



Code:

```
    def q0(s,i):
if len(s) == 0 or len(s) == i:return
elif s[i] == '1': q1(s,i+1) print('1',end="")
    else:
    q0(s,i+1) print('0',end="")

    def q1(s,i):
if len(s) == 0 or len(s) == i:return
elif s[i] == '1': q1(s,i+1) print('0',end="")
    else:
    q1(s,i+1) print('1',end="")

    def my_function(x):
    return x[::-1]

    if _name_ == "_main_":
    mystr = input("Enter a string : ")
    s = my_function(mystr)q0(s,0)
```

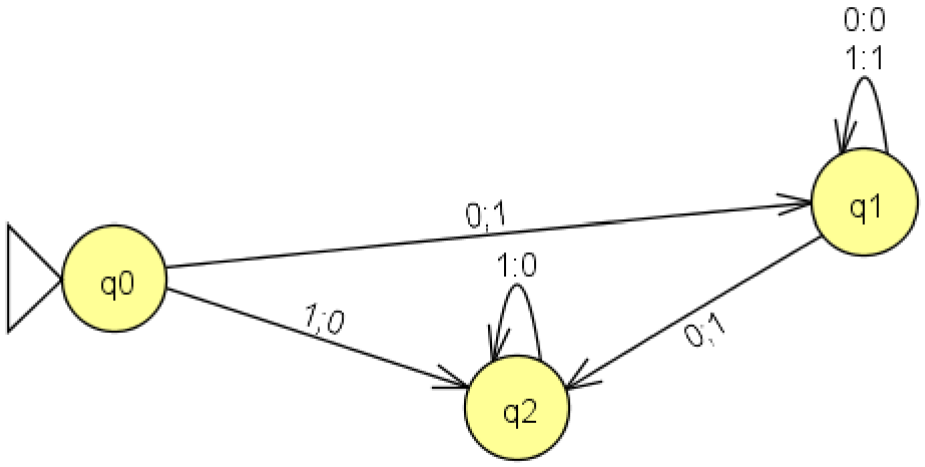
Output:

```
Enter a string : 10101
01011
```

Program- 10

Design a program which will increment the given binary number by 1.

Diagram:

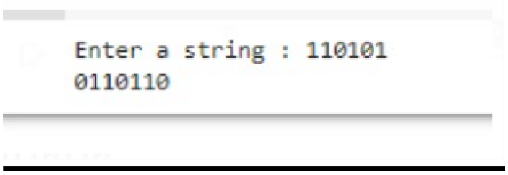


Code:

```
    q0(s,i):
if len(s) == 0 or len(s) == i:return
elif s[i] == '1': q2(s,i+1) print('0',end="")
    else:
    q1(s,i+1) print('1',end="")

    def q1(s,i):
if len(s) == 0 or len(s) == i:return
elif s[i] == '1': q1(s,i+1) print('1',end="")
    else:
    q1(s,i+1) print('0',end="")

    def q2(s,i):
if len(s) == 0 or len(s) == i:return
elif s[i] == '1': q2(s,i+1) print('0',end="")
    else:
    q1(s,i+1) print('1',end="")

    def my_function(x):
    return x[::-1]
```

```
if _name_ == "_main_":
mystr = input("Enter a string : ")s = my_function(mystr)
q0(s + '0',0)
```

Output:



Program- 11

Design a Program to convert NDFA to DFA.Theory:-

Conversion from NFA to DFA

In NFA, when a specific input is given to the current state, the machine goes to multiple states. It can have zero, one or more than one move on a given input symbol. On the other hand, in DFA, when a specific input is given to the current state, the machine goes to only one state. DFA has only one move on a given input symbol.

Let, M = (Q, ∑, δ, q0, F) is an NFA which accepts the language L(M). There should be equivalent DFA
denoted by M' = (Q', ∑', q0', δ', F') such that L(M) = L(M').

Steps for converting NFA to DFA:

Step 1: Initially Q' = ϕ

Step 2: Add q0 of NFA to Q'. Then find the transitions from this start state.

Step 3: In Q', find the possible set of states for each input symbol. If this set of states is not in Q', then addit to Q'.

Step 4: In DFA, the final state will be all the states which contain F(final states of NFA).

CODE:-

```
import pandas as pd
# Taking NFA input from
User nfa = {}
n = int(input("No. of states : "))            #Enter total no. of states
t = int(input("No. of transitions : "))       #Enter total no. of transitions/paths eg:
a,b so input 2 for a,b,c input 3
for i in range(n):
state = input("state name : ") #Enter state name eg: A, B, C, q1, q2 ..etc
nfa[state] = {}                          #Creating a nested dictionary
for j in range(t):
path = input("path : ")        #Enter path eg : a or b in {a,b} 0 or 1 in
      {0,1} print("Enter end state from state {} travelling through path {} :
".format(state,path))
          reaching_state = [x for x in input().split()] #Enter all the end states that
      nfa[state][path] = reaching_state        #Assigning the end states to the paths
in dictionary
```

```
print("\nNFA :- \n")
print(nfa)                          #Printing NFA
print("\nPrinting  NFA table  :-")
                nfa_table           =
pd.DataFrame(nfa)
print(nfa_table.transpose())

print("Enter final state of NFA : ")
nfa_final_state = [x for x in input().split()]      # Enter final state/states of NFA
#########################################################

new_states_list = []                    #holds all the new states created in dfa
dfa = {}                            #dfa dictionary/table or the output structure we
needed
keys_list = list(list(nfa.keys())[0])               #conatins all the states in nfa plus
the states created in dfa are also appended further
path_list = list(nfa[keys_list[0]].keys()) #list of all the paths eg: [a,b] or [0,1]



#########################################################
# Computing first row of DFA transition table
dfa[keys_list[0]] = {}                  #creating a nested dictionary in dfa
for y in range(t):
   var = "".join(nfa[keys_list[0]][path_list[y]]) #creating a single string from all
the elements of the list which is a new state
dfa[keys_list[0]][path_list[y]] = var     #assigning   the   state   in   DFA
table if var not in keys_list:              #if the state is newly created
      new_states_list.append(var)               #then append it to the
new_states_list
      keys_list.append(var)                 #as well as to the keys_list which
contains all the states
#########################################################
# Computing the other rows of DFA transition table
while len(new_states_list) != 0:                #consition is true only if the
new_states_list is not empty
   dfa[new_states_list[0]] = {}                 #taking the first element of the
new_states_list and examining it
for _ in range(len(new_states_list[0])):for i
in range(len(path_list)):
temp = []                               #creating a temporay list
for j in range(len(new_states_list[0])):
            temp += nfa[new_states_list[0][j]][path_list[i]] #taking the union of
```

```
    the states
    s = ""
            s = s.join(temp)                    #creating a single string(new state) from all
    the elements of the list
  if s not in keys_list:                        #if the state is newly created
    new_states_list.append(s)                       #then append it to the
    new_states_list
              keys_list.append(s)               #as well as to the keys_list which
    contains all the states
            dfa[new_states_list[0]][path_list[i]] = s #assigning the new state in the
    DFA table
       new_states_list.remove(new_states_list[0])          #Removing the first element
    in the new_states_list
    print("\nDFA :- \n")
    print(dfa)                                  #Printing the DFA
    created print("\nPrinting DFA table :- ")
    dfa_table = pd.DataFrame(dfa)
    print(dfa_table.transpose())
    dfa_states_list = list(dfa.keys())
    dfa_final_states = []
  for x in dfa_states_list:for
    i in x:
  if i in nfa_final_state:
    dfa_final_states.append(x)


    break



    print("\nFinal states of the DFA are : ",dfa_final_states)          #Printing
    Final states of DFA
```

OUTPUT:-

```
No. of states : 4
No. of transitions : 2
state name : A
path : a
Enter end state from state A travelling through path a :
A B
path : b
Enter end state from state A travelling through path b :
A
state name : B
path : a
Enter end state from state B travelling through path a :
C
path : b
Enter end state from state B travelling through path b :
C
state name : C
path : a
Enter end state from state C travelling through path a :
D
path : b
Enter end state from state C travelling through path b :
D
state name : D
path : a
Enter end state from state D travelling through path a :

path : b
Enter end state from state D travelling through path b :


NFA :-

{'A': {'a': ['A', 'B'], 'b': ['A']}, 'B': {'a': ['C'], 'b': ['C']}, 'C': {'a': ['D'], 'b': ['D']}, 'D': {'a': [], 'b': []}}
```

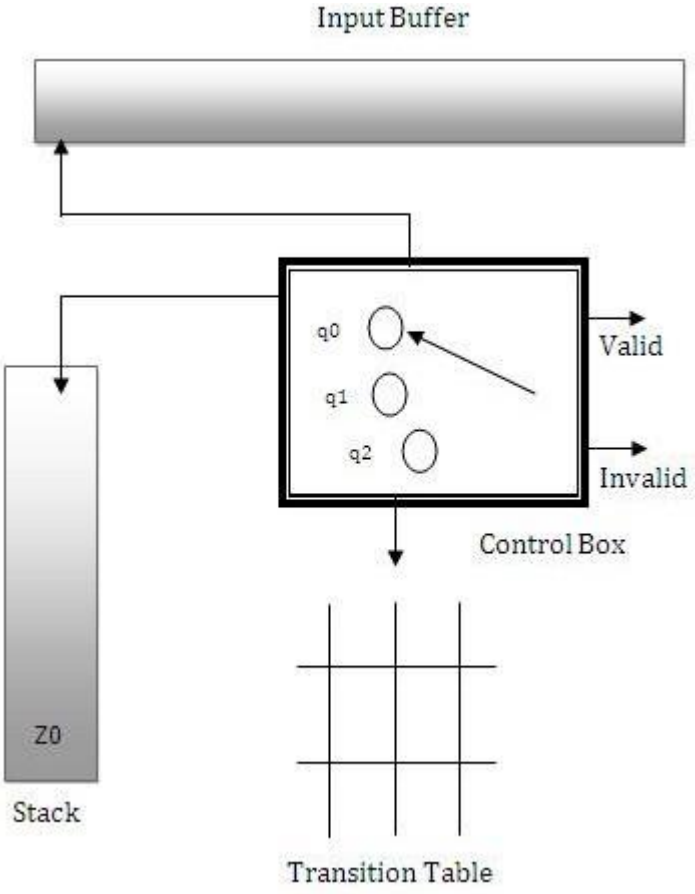
✓ 2m 3s completed at 1:06 PM

```
Printing NFA table :-
      a   b
A  [A, B]  [A]
B     [C]  [C]
C     [D]  [D]
D      []   []
Enter final state of NFA :
D

DFA :-

{'A': {'a': 'AB', 'b': 'A'}, 'AB': {'a': 'ABC', 'b': 'AC'}, 'ABC': {'a': 'ABCD', 'b': 'ACD'}, 'AC': {'a': 'ABD', 'b': 'AD'}, 'ABCD': {'a': 'ABCD', 'b': 'ACD'}, 'ACD': {'a': 'ABD', 'b': 'AD'}, 'ABD': {'a': 'ABC', 'b': 'AC'}, 'AD': {'a': 'AB', 'b': 'A'}}

Printing DFA table :-
      a   b
A    AB   A
AB   ABC  AC
ABC  ABCD ACD
AC   ABD  AD
ABCD ABCD ACD
ACD  ABD  AD
ABD  ABC  AC
AD   AB   A
```

Program- 12

Design a Program to create PDA machine that accept the well-formed parenthesis.

Theory: As per the AIM, set of valid strings that can be generated by given language is represented in set A:
A = {(),(()), ()(), (()), ((()(()))) ...}
means all the parenthesis that are open must closed or combination of all legal parenthesis formation. Here, opening par is '(' and closing parenthesis is ')'. Block diagram of push downautomata is shown in Figure 1.



Input string can be valid or invalid, valid if the input string follow set A (define above). PDA has todetermine whether the input string is according to the language or not.

Let M be the PDA machine for above AIM, hence it can be define as M(Q, Σ, Γ, δ, q0, Z0, F)
whereQ: set of states: {q0, q1}
Σ: set of input symbols: {(,)} Γ: Set of stack symbols: {(, Z} **q0:** initial state (q0)

Z0: initial stack symbol (Z)

F: set of Final states: { } [Note: Here, set of final states is null as decision of validity of string is based on stack whether it is empty or not. If empty means valid else invalid.]

δ: Transition Function:

$$\delta(q0, (, Z) \rightarrow (q0, (Z)$$
$$\delta(q0, (, () \rightarrow (q0, (()$$
$$\delta(q0,), () \rightarrow (q0, \varepsilon)$$
$$\delta(q0, \varepsilon, Z) \rightarrow (q1, \varepsilon)$$

Rules for implementing PDA for a given language

Initial Setup: Load the input string in input buffer, push Z as an initial stack symbol and consider the machine in at initial state q0.

Rules:

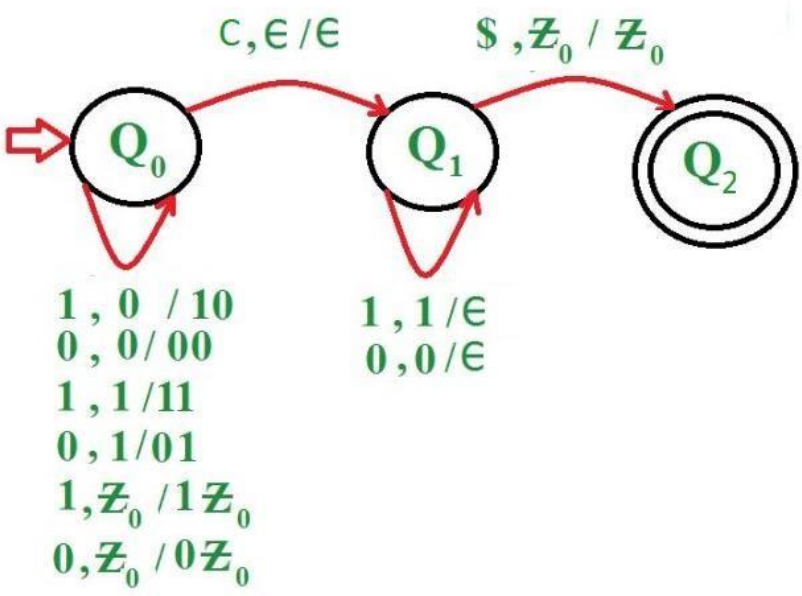
1. It is must that the first symbol should be '('.
2. If the input symbol of string is '(' and stack top is Z then push symbol '(' into stack and readthe next character in input string.
3. If next character is again '(', then push '(' again in stack and repeat the same process for all'(' in input string.
4. If character is ')' and stack top is '(', then pop '(' from stack.
5. If all the characters of input string are parsed and stack top is Z, it means string is valid,pop Z from stack and change the state from q0 to q1.

Program- 13

Design a PDA to accept WCW^R where w is any string and W^R is reverse of that string and C is a Special symbol.

Theory: Approach used in this PDA –
Keep on pushing 0's and 1's no matter whatever is on the top of stack until reach the middle element. When middle element 'c' is scanned then process it without making any changes in stack. Now if scanned symbol is '1' and top of stack also contain '1' then pop the element from top of stack or if scanned symbol is '0' and top of stackalso contain '0' then pop the element from top of stack. If string becomes empty or scanned symbol is '$' and stack becomes empty, then reach to final state else move to dead state.

- Step 1: On receiving 0 or 1, keep on pushing it on top of stack without going to next state.
- Step 2: On receiving an element 'c', move to next state without making any change in stack.
- Step 3: On receiving an element, check if symbol scanned is '1' and top of stack also contain '1' or if symbol scanned is '0' and top of stack also contain '0' then pop the element from top of stack else move to dead state.Keep on repeating step 3 until string becomes empty.
- Step 4: Check if symbol scanned is '$' and stack does not contain any element then move to final state elsemove to dead state.



Examples:
Input : 1 0 1 0 1 0 1 0 1
Output :ACCEPTED

Input : 1 0 1 0 1 1 1 1 0
Output :NOT ACCEPTED

Program- 14

Design a Turing machine that's accepts the following language $a^n b^n c^n$ where n>0.Steps:

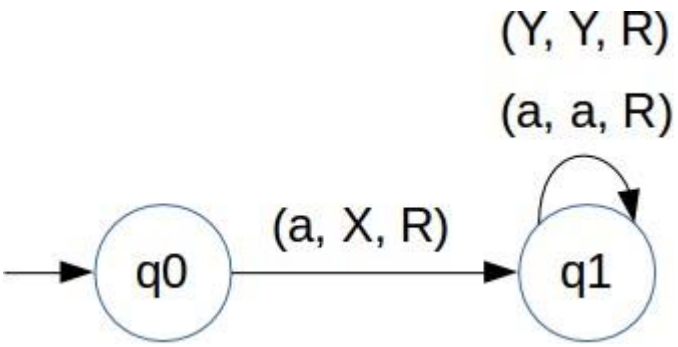
1. Mark 'a' then move right.
2. Mark 'b' then move right
3. Mark 'c' then move left
4. Come to far left till we get 'X'
5. Repeat above steps till all 'a', 'b' and 'c' are marked
6. At last if everything is marked that means string is accepted.

State Transition Diagram:

We have designed state transition diagram for anbncn | n ≥ 1 as

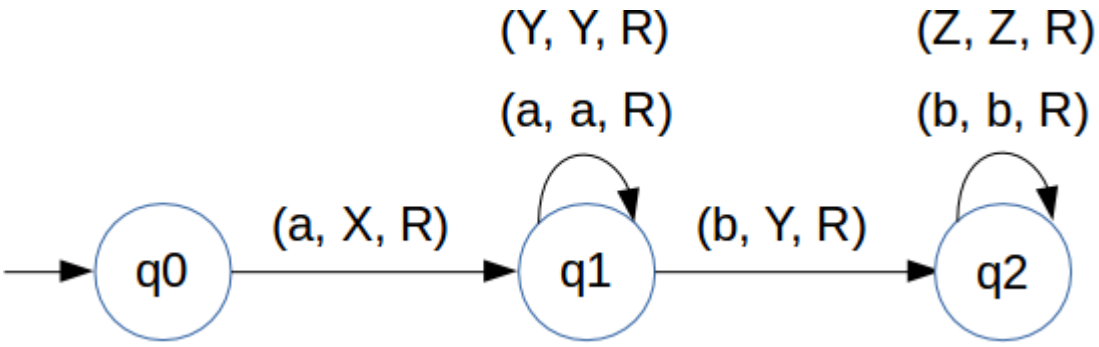
follows:Following Steps:

a. Mark 'a' with 'X' and move towards unmarked 'b'
b. Move towards unmarked 'b' by passing all 'a's
c. To move towards unmarked 'b' also pass all 'Y's if exist



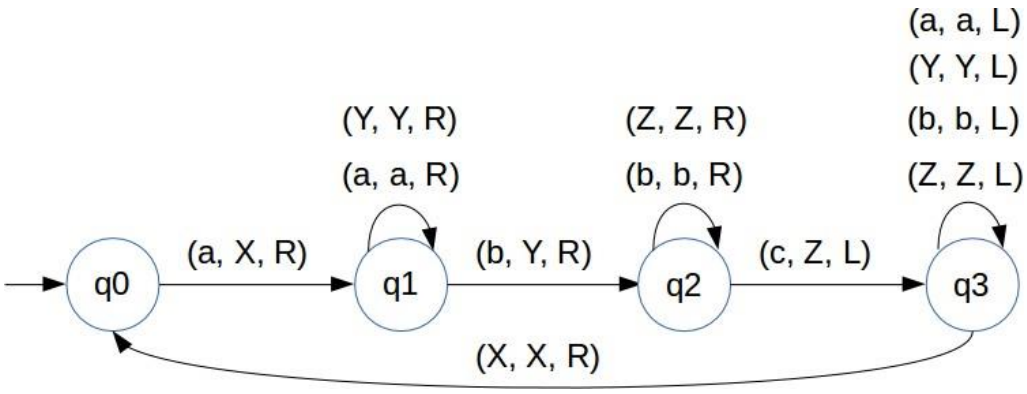
Following Steps:

a. Mark 'b' with 'Y' and move towards unmarked 'c'
b. Move towards unmarked 'c' by passing all 'b's
c. To move towards unmarked 'c' also pass all 'Z's if exist

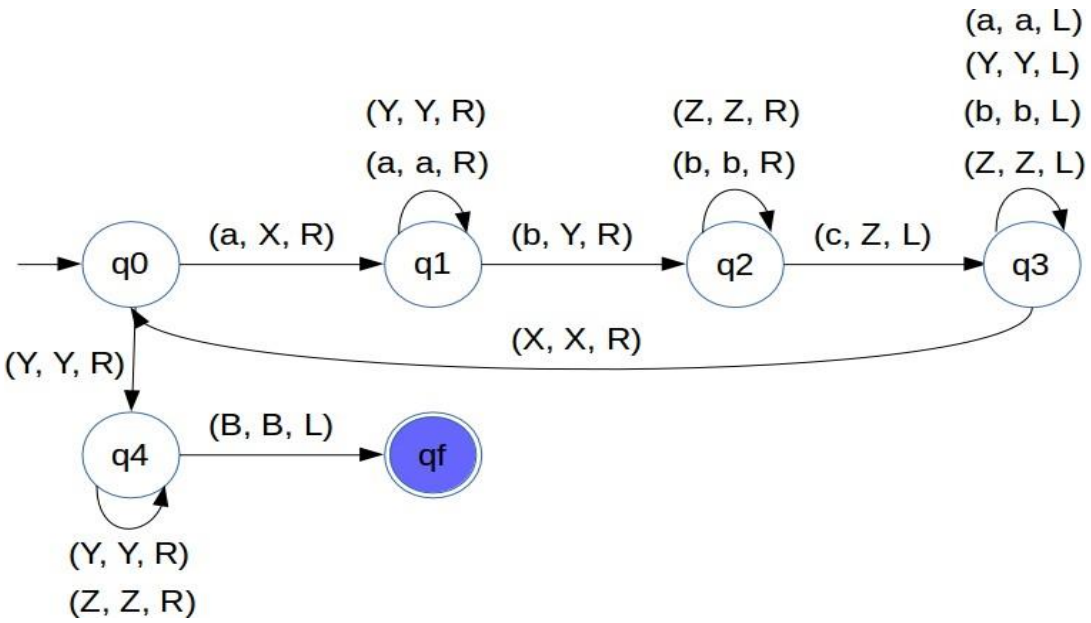


Following Steps:

a. Mark 'c' with 'Z' and move towards first 'X' (in left)
b. Move towards first 'X' by passing all 'Z's, 'b's, 'Y's and 'a's
c. When 'X' is reached just move one step right by doing nothing.



To check all the 'a's, 'b's and 'c's are over add loops for checking 'Y' and 'Z' after "we get 'X' followed by'Y'"
To reach final state(qf) just replace BLANK with BLANK and move either direction


(a, a, L)
(Y, Y, L)
(b, b, L)
(Z, Z, L)
(Y, Y, R)
(a, a, R)
(Z, Z, R)
(b, b, R)
(a, X, R)
(b, Y, R)
(c, Z, L)
q0
q1
q2
q3
(X, X, R)
(Y, Y, R)
(B, B, L)
q4
qf
(Y, Y, R)
(Z, Z, R)