Experiment No - 1 **Date of Experiment : 15 Jan 2022**

Program : - Write a program to accept a string and validate using NFA.

Theory:- **NFA** (Non-deterministic Finite automata) finite state machine that can move to any combination of states for an input symbol i.e. there is no exact state to which the machine will move.

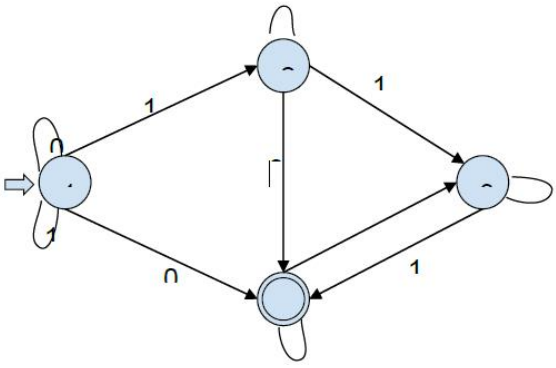
NFA / NDFA (Non-deterministic Finite automata) can be represented by 5-tuple (Q, Σ, δ, q0, F) where –

- Q is a finite set of states.
- Σ is a finite set of symbols called the alphabets.
- δ is the transition function where d: Q × Σ → 2Q (Here the power set of Q (2Q) has been taken because in case of NDFA, from a state, transition can occur to any combination of Q states)
- q0 is the initial state from where any input is processed (q0 ∈ Q).
- F is a set of final state/states of Q (F ⊆ Q).

In programming, NFA is created using a directed graph. Each vertex of the graph denotes the states of NDA. The edges of the graph can have one of the two values 0 or 1. Edge labeled as 0 represents non-accepting transition whereas Edge labeled as 1 represents accepting transition.

There is an entry point to the graph generally vertex 1 from where it takes input string which is a binary array of finite length.

Let's see an NFA graphical form and then solve a grammar using it.



Starting state - vertex 1

Accepting states - Vertices with double circles(label 1) // Vertex 4

Non accepting states - single circles (label 0). // Vertices 1, 2 and 3.

For Input : 1010

In-state 1, we have two possibilities, either follow the self-loop and stay in state 1 or follow the edge labeled 1 and go to state 3.

{1} 1010 --> {1, 3} 010

In-state 3, there is no edge labeled 0, so the computation will die out.

In-state 1, we have two possibilities, either follow the self-loop and stay in state 1, or follow the edge labeled 0 to state 2.

{1, 3} 010 --> {1, 2} 10

Now there is no edge labeled 1 from state 2. The computation will die out. We have two possibilities: either follow the self-loop to state 1 or follow the edge labeled 1 to state 3.

{1, 2} 10 --> {1, 3} 0

In-state 3, there is no edge labeled 0. So the computation will die out. In-state 1, we have two possibilities: either follow the self-loop to state 1 or the edge labeled 0 to state 2.

{1, 3} 0 --> {1, 2}

Now the NFA has consumed the input. It can be either be in states 1 or 2, both of which are non accepting. So the NFA has rejected the input 1010.

For Input: 1100

{1} 1100 --> {1, 3} 100 {1, 3} 100 --> {1, 3, 4} 00 {1, 3, 4}

00--> {1, 2, 4} 0 {1, 2, 4} 0--> {1, 2, 4}

Now the NFA has consumed the input. It can either be in states 1, 2, or 4. State 4 is an accepting state. So, the NFA accepts the string 1100.

We can easily verify that the given NFA accepts all binary strings with "00" and/or "11" as a substring.

Practical Implementation of Insertion Sort :-

Code:-

```
# Design to recognize strings NFA


nfa = 1
# This checks for invalid input.
flag = 0
# Function for the state Q2
def state1(c):
    global nfa,flag

    # State transitions
    # 'a' takes to Q4, and
    # 'b' and 'c' remain at Q2
    if (c == 'a'):
        nfa = 2
    elif (c == 'b' or c == 'c'):
        nfa = 1
    else:
        flag = 1

# Function for the state Q3
def state2(c):
    global nfa,flag

    # State transitions
    # 'a' takes to Q3, and
    # 'b' and 'c' remain at Q4
    if (c == 'a'):
        nfa = 3
    elif (c == 'b' or c == 'c'):
        nfa = 2
    else:
        flag = 1

# Function for the state Q4
def state3(c):
    global nfa,flag

    # State transitions
    # 'a' takes to Q2, and
    # 'b' and 'c' remain at Q3
    if (c == 'a'):
        nfa = 1
    elif (c == 'b' or c == 'c'):
        nfa = 3
    else:
```

```
        flag = 1

# Function for the state Q5
def state4(c):
    global nfa,flag

    # State transitions
    # 'b' takes to Q6, and
    # 'a' and 'c' remain at Q5
    if (c == 'b'):
        nfa = 5
    elif (c == 'a' or c == 'c'):
        nfa = 4
    else:
        flag = 1

# Function for the state Q6
def state5(c):
    global nfa, flag

    # State transitions
    # 'b' takes to Q7, and
    # 'a' and 'c' remain at Q7
    if (c == 'b'):
        nfa = 6
    elif (c == 'a' or c == 'c'):
        nfa = 5
    else:
        flag = 1

# Function for the state Q7
def state6(c):
    global nfa,flag

    # State transitions
    # 'b' takes to Q5, and
    # 'a' and 'c' remain at Q7
    if (c == 'b'):
        nfa = 4
    elif (c == 'a' or c == 'c'):
        nfa = 6
    else:
        flag = 1

# Function for the state Q8
def state7(c):
    global nfa,flag
```

```
    # State transitions
    # 'c' takes to Q9, and
    # 'a' and 'b' remain at Q8
    if (c == 'c'):
        nfa = 8
    elif (c == 'b' or c == 'a'):
        nfa = 7
    else:
        flag = 1

# Function for the state Q9
def state8(c):
    global nfa,flag

    # State transitions
    # 'c' takes to Q10, and
    # 'a' and 'b' remain at Q9
    if (c == 'c'):
        nfa = 9
    elif (c == 'b' or c == 'a'):
        nfa = 8
    else:
        flag = 1

# Function for the state Q10
def state9(c):
    global nfa,flag

    # State transitions
    # 'c' takes to Q8, and
    # 'a' and 'b' remain at Q10
    if (c == 'c'):
        nfa = 7
    elif (c == 'b' or c == 'a'):
        nfa = 9
    else:
        flag = 1

# Function to check for 3 a's
def checkA(s, x):
    global nfa,flag
    for i in range(x):
        if (nfa == 1):
            state1(s[i])
        elif (nfa == 2):
            state2(s[i])
        elif (nfa == 3):
            state3(s[i])
```

```
    if (nfa == 1):
        return True

    else:
        nfa = 4

# Function to check for 3 b's
def checkB(s, x):
    global nfa,flag
    for i in range(x):
        if (nfa == 4):
            state4(s[i])
        elif (nfa == 5):
            state5(s[i])
        elif (nfa == 6):
            state6(s[i])

    if (nfa == 4):
        return True
    else:
        nfa = 7

# Function to check for 3 c's
def checkC(s, x):
    global nfa, flag
    for i in range(x):
        if (nfa == 7):
            state7(s[i])
        elif (nfa == 8):
            state8(s[i])
        elif (nfa == 9):
            state9(s[i])

    if (nfa == 7):
        return True

# Driver Code

s = "bbbca"
x = 5

# If any of the states is True, that is, if either
# the number of a's or number of b's or number of c's
# is a multiple of three, then the is accepted
if (checkA(s, x) or checkB(s, x) or checkC(s, x)):
    print("ACCEPTED")
```

```
else:
    if (flag == 0):
        print("NOT ACCEPTED")

    else:
        print("INPUT OUT OF DICTIONARY.")
```

Output:

ACCEPTED

Conclusion: The entered string were identified as NFA or not based of the provided state diagram .