Practical 1:

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

Theory: -

NFA stands for *non-deterministic finite automata.*

To understand NFA we first look at what is an Automata.

The term "*Automata*" is derived from the Greek word "*αὐτόματα*" which means "*self-acting*". An automaton (Automata in plural) is an abstract self-propelled computing device which follows a predetermined sequence of operations automatically.

An automaton with a finite number of states is called a **Finite Automaton** (FA) or **Finite State Machine** (FSM).

Formal definition of a Finite Automaton

An automaton can be represented by a 5-tuple (Q, ∑, δ, q_0, F), where −

- **Q** is a finite set of states.
- **∑** is a finite set of symbols, called the **alphabet** of the automaton.
- **δ** is the transition function.
- **q_0** is the initial state from where any input is processed ($q_0 \in Q$).
- **F** is a set of final state/states of Q ($F \subseteq Q$).

NFA:-

The finite automata are called NFA when there exist many paths for specific input from the current state to the next state. Every NFA is not DFA, but each NFA can be translated into DFA. NFA is defined in the same way as DFA but with the following two exceptions, it contains multiple next states, and it contains ε transition.

Example:-

In the following image, we can see that from state q0 for input a, there are two next states q1 and q2, similarly, from q0 for input b, the next states are q0 and q1. Thus it is not fixed or determined that with a particular input where to go next. Hence this FA is called non-deterministic finite automata.

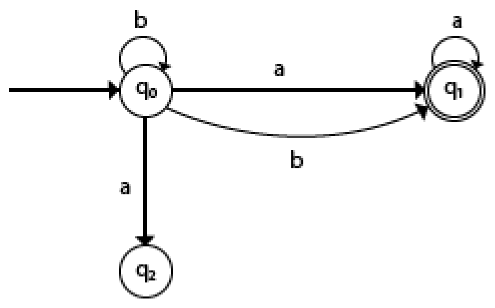


Fig:- NDFA

Formal definition of NFA:

NFA also has five states same as DFA, but with different transition function, as shown follows:

$\delta: Q \times \Sigma \rightarrow 2^{Q}$

where,

1. Q: finite set of states
2. ∑: finite set of the input symbol
3. q0: initial state
4. F: final state
5. δ: Transition function

Graphical Representation of an NFA

An NFA can be represented by digraphs called state diagram. In which:

1. The state is represented by vertices.
2. The arc labeled with an input character show the transitions.
3. The initial state is marked with an arrow.
4. The final state is denoted by the double circle.

Example 1:-

1. Q = {q0, q1, q2}
2. ∑ = {0, 1}
3. q0 = {q0}
4. F = {q2}

Solution:

Transition diagram:

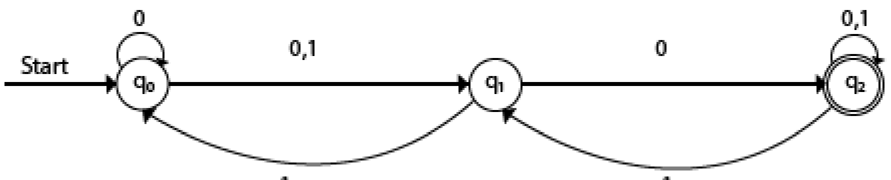


Fig: NFA

Transition Table:

Present State	Next state for Input 0	Next State of Input 1
→q0	q0, q1	q1
q1	q2	q0
*q2	q2	q1, q2

In the above diagram, we can see that when the current state is q0, on input 0, the next state will be q0 or q1, and on 1 input the next state will be q1. When the current state is q1, on input 0 the next state will be q2 and on 1 input, the next state will be q0. When the current state is q2, on 0 input the next state is q2, and on 1 input the next state will be q1 or q2.

Example 2:

NFA with ∑ = {0, 1} accepts all strings with 01.

Solution:

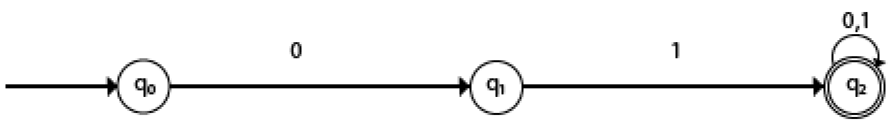


Fig: NFA

Transition Table:

Present State	Next state for Input 0	Next State of Input 1
→q0	q1	ε
q1	ε	q2
*q2	q2	q2

Example 3:

NFA with ∑ = {0, 1} and accept all string of length atleast 2.

Solution:

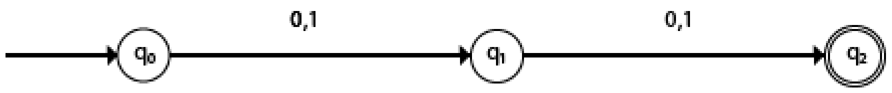


Fig: NFA

Transition Table:

Present State	Next state for Input 0	Next State of Input 1
→q0	q1	q1
q1	q2	q2
*q2	ε	ε

Implement NFA algorithm (based on Python)

Ideas for solving problems

Design of data structure

Referring to the definition of NFA, NFA is a 5-tuple: M = (Q, ∑, Δ, s, F)

Where: Q is the finite set of states

Σ is a finite alphabet

s is the start state

F contained in Q, end state set

Δ Set of state transition functions

Solution algorithm:

NFA identification string: there are many possible paths for a string to pass through the NFA. As long as there is one that can finally reach the end state, the NFA can identify the string. Therefore, just get the set of states that the string finally reaches through the NFA and check whether there is an end state. If it exists, NFA can recognize this string, otherwise it cannot.

We should also consider the problem of empty character e, the set of states that a state can reach through empty character e, that is, the problem of empty closure of a state needs to be considered. If I is a subset of a set of States, I will have an empty closure set, which is recorded as ε- closure(I). The empty closure set is also a state set, and its elements conform to the following points: all elements of I are elements of the empty closure set; For each element in I, start from that element and pass through any bar ε The states that an arc can reach are elements of an empty closure set.

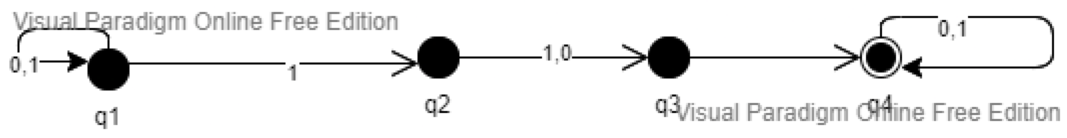
- In class NFA, four functions are designed, which are read_string()，e_closure()， next_state()， match_final_state().
- Function read_string(input_str): read the input string and judge whether it can be accepted by NFA
- Function e_closure(state): find the empty closure of a state (the idea of BFS)

- Function next_state(current_states, c): the next state set reached by the current state set through the character C
- Function match_final_state(final_current_states): the final state matches the accepted state

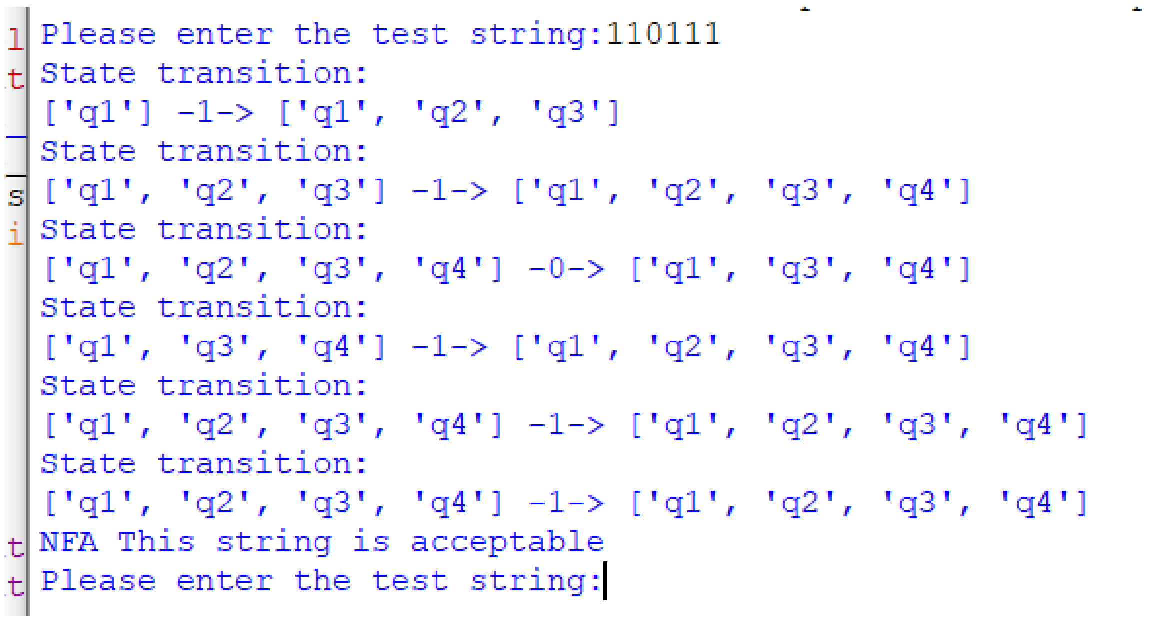
Code:-

```
"""Design to recognize strings NFA"""
import string
from collections import deque

class NFA:
    def __init__(self, states, symbols, t_state, start_state, final_state):     # initialization
        self.states = states                    # State set
        self.symbols = symbols                  # Input symbol set / alphabet
        self.t_state = t_state                  # State transition function
        self.start_state = start_state          # Start state
        self.final_state = final_state          # End state

    # The collection of the next state that the
    # current state can reach through a character, including empty closures
    def next_state(self, current_states, c):
        # The next state set, including the set of empty closures of the next state set
        next_states = []
        for state in current_states:
            if state in self.t_state.keys():
                if c in self.t_state[state].keys():
                    n_states = self.t_state[state][c]
                    for s in n_states:          # Gets the empty closure of the next state set
                        s_closure = self.e_closure(s)
                         # Add the set of empty closures to the next state set
                        for s_c in s_closure:
                            if s_c not in next_states:
                                next_states.append(s_c)
        print('State transition:')
        print(current_states, end=' ')
        print('-' + c + '->', end=' ')
        print(next_states)
# Returns the set of the next state obtained / the state set through the character c
        return next_states


    def e_closure(self, state):                 # Find the empty closure of a state
        d_state = deque()                       # Dual ended queue, storing intermediate status
        empty_bag = []                          # Empty closure of a state
        d_state.append(state)
        empty_bag.append(state)
        while (len(d_state)):
            current_state = d_state[0]
            if current_state in self.t_state.keys():  # If the current state exists
                if 'e' in self.t_state[current_state].keys(): #If a null character transfer
                    for s in self.t_state[current_state]['e']:
                        if s not in empty_bag:  # If the state is not in an empty closure
                            # Queue and join the state arrived via e into the empty closure
                           d_state.append(s)
                            empty_bag.append(s)
            d_state.popleft()                   # Throw the current state
        return empty_bag
```

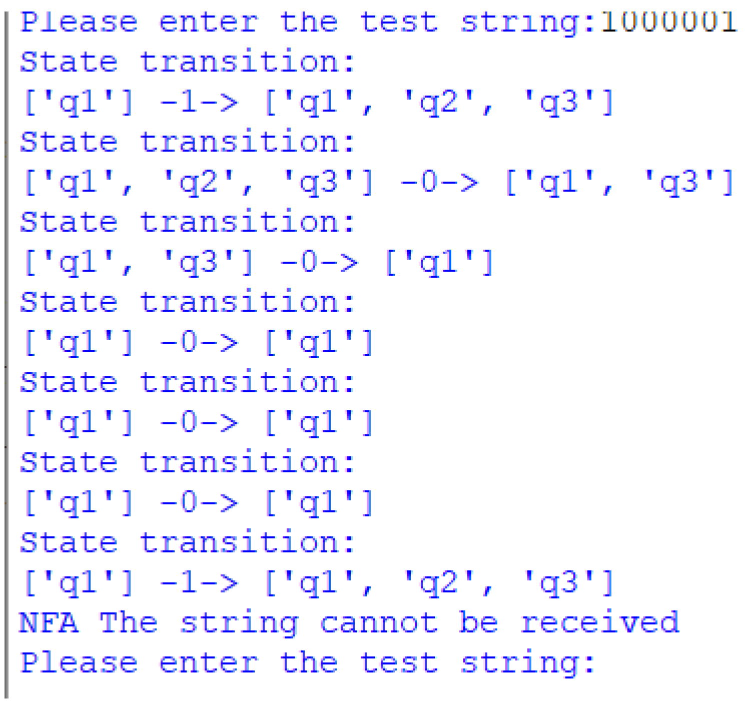
```
 # The final state matches the end state
    def match_final_state(self, final_current_states: list) ->bool :
        for state in final_current_states:
            if state in self.final_state:
                return True
        return False


    def read_string(self, input_str):           # Read string to enter NFA authentication
        for c in input_str:                     # Judge whether the string meets the alphabet
            if c not in self.symbols:
                print('Illegal character in string')
                return

        next_states = [self.start_state]
        for i in str:
            next_states = self.next_state(next_states, i)    #call
            #print(next_states)

        result = self.match_final_state(next_states)        #call
        if result == True:
            print('NFA This string is acceptable')
        else:
            print('NFA The string cannot be received')


if __name__ == '__main__':
    states = ['q1', 'q2', 'q3', 'q4']   # NFA state set
    symbols = ['0', '1']                # NFA alphabet
    t_state = {
        'q1': {
            '0': ['q1'],
            '1': ['q1', 'q2']
        },
        'q2': {
            '1': ['q3'],
            '0': ['q3']
        },
        'q3': {
            '1': ['q4']
        },
        'q4': {
            '0': ['q4'],
            '1': ['q4']
        }
    }                      # State transition function
    start_state = 'q1'                  # Start state
    final_state = ['q4']                # End state set

    while True:
        str = input("Please enter the test string:")
        if str == 'exit':
            break
        nfa = NFA(states, symbols, t_state, start_state, final_state)
        nfa.read_string(str)
```


Visual Paradigm Online Free Edition
0,1
q1
1
q2
1,0
q3
0,1
q4
Visual Paradigm Online Free Edition

Output:-



```
Please enter the test string:110111
State transition:
['q1'] -1-> ['q1', 'q2', 'q3']
State transition:
['q1', 'q2', 'q3'] -1-> ['q1', 'q2', 'q3', 'q4']
State transition:
['q1', 'q2', 'q3', 'q4'] -0-> ['q1', 'q3', 'q4']
State transition:
['q1', 'q3', 'q4'] -1-> ['q1', 'q2', 'q3', 'q4']
State transition:
['q1', 'q2', 'q3', 'q4'] -1-> ['q1', 'q2', 'q3', 'q4']
State transition:
['q1', 'q2', 'q3', 'q4'] -1-> ['q1', 'q2', 'q3', 'q4']
NFA This string is acceptable
Please enter the test string:
```



```
Please enter the test string:1000001
State transition:
['q1'] -1-> ['q1', 'q2', 'q3']
State transition:
['q1', 'q2', 'q3'] -0-> ['q1', 'q3']
State transition:
['q1', 'q3'] -0-> ['q1']
State transition:
['q1'] -0-> ['q1']
State transition:
['q1'] -0-> ['q1']
State transition:
['q1'] -0-> ['q1']
State transition:
['q1'] -1-> ['q1', 'q2', 'q3']
NFA The string cannot be received
Please enter the test string:
```

Conclusion:-

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