

APP WEEK 13

Sample programs to Construct NFA and DFA using Python

Q1) Write a Python program to create an NFA that accepts strings containing only the letter 'a'

```
q1.py > ...
1 def is_accepting_state(current_state):
2     return current_state == 1
3
4 def nfa_accepts(input_string):
5     current_state = 0
6     for char in input_string:
7         if current_state == 0 and char == 'a':
8             current_state = 1
9         elif current_state == 1 and char == 'a':
10            current_state = 1
11
12    return is_accepting_state(current_state)
13
14 # Get the input string from the user
15 input_string = input("Enter a string to check if it contains only 'a's: ")
16
17 if nfa_accepts(input_string):
```

Q2) Create a Python function to check if a given string is accepted by an NFA that recognizes the pattern "ab|ba" (either "ab" or "ba").



```

Q2.PY > ...
1 def is_accepting_state(current_state):
2     return current_state in [2, 4]
3
4 def nfa_accepts(input_string):
5     current_state = 0
6
7     for char in input_string:
8         if current_state == 0 and char == 'a':
9             current_state = 1
10        elif current_state == 0 and char == 'b':
11            current_state = 3
12        elif current_state == 1 and char == 'b':
13            current_state = 2
14        elif current_state == 2:
15            break # Accept if "ab"
16        elif current_state == 3 and char == 'a':
17            current_state = 4
18        elif current_state == 4 and char == 'a':
19            current_state = 1
20
21    return is_accepting_state(current_state)

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

Python + ▾ 🗑️ ⋮

```

PS C:\Users\MANI\Desktop\week 13> & C:\Python312\python.exe "c:/Users/MANI/Desktop/week 13/Q2.PY"
Could not find platform independent libraries <prefix>
Enter a string to check if it matches 'ab|ba': bangalore
Accepted: The string matches 'ab|ba'.

```

Q3) Implement a Python script that converts a simple NFA into a DFA with two states

```

def nfa_to_dfa(nfa):
    dfa_states = set()
    dfa_transitions = {}
    dfa_start_state = nfa.start_state
    dfa_accept_states = set()

    # Initialize the DFA with the NFA's start state
    dfa_states.add(frozenset([nfa.start_state]))
    queue = [frozenset([nfa.start_state])]

    while queue:
        current_state = queue.pop(0)

        for symbol in nfa.alphabet:
            next_states = set()
            for nfa_state in current_state:
                next_states.update(nfa.transitions[nfa_state, symbol])

            dfa_next_state = frozenset(next_states)
            dfa_transitions[current_state, symbol] = dfa_next_state

            if dfa_next_state not in dfa_states:
                dfa_states.add(dfa_next_state)
                queue.append(dfa_next_state)

    # Determine accept states of the DFA
    for nfa_state in nfa.states:
        if nfa.is_accepting(nfa_state):
            dfa_accept_states.add(frozenset([nfa_state]))

    return dfa_states, dfa_transitions, dfa_start_state, dfa_accept_states

def main():
    nfa = NFA()
    dfa_states, dfa_transitions, dfa_start_state, dfa_accept_states = nfa_to_dfa(nfa)

    print("DFA States:", dfa_states)
    print("DFA Transitions:", dfa_transitions)
    print("DFA Start State:", dfa_start_state)
    print("DFA Accept States:", dfa_accept_states)

if __name__ == "__main__":
    main()

```

Terminal Output:

```

PS C:\CODE\.vscode> & C:/Users/RISHAW/AppData/Local/Programs/Python/Python310/python.exe c:/CODE/.vscode/q133.py
DFA States: {frozenset(['q0']), frozenset(['q0', 'q1']), frozenset(['q2'])}
DFA Transitions: {(frozenset(['q0']), 'a'): frozenset(['q0', 'q1']), (frozenset(['q0', 'q1']), 'a'): frozenset(['q0', 'q1']), (frozenset(['q0', 'q1']), 'b'): frozenset(['q2']), (frozenset(['q2']), 'a'): frozenset(['q2']), (frozenset(['q2']), 'b'): frozenset(['q2'])}
DFA Start State: q0
DFA Accept States: {frozenset(['q2'])}
PS C:\CODE\.vscode>

```

Q4) Write a Python program to construct a DFA that accepts binary strings ending in '01'

```

def is_accepting_state(current_state):
    return current_state == 2

def dfa_accepts(input_string):
    current_state = 0

    for char in input_string:
        if current_state == 0 and char == '0':
            current_state = 1
        elif current_state == 1 and char == '1':
            current_state = 2
        else:
            current_state = 0

    return is_accepting_state(current_state)

# Get the input binary string from the user

```

Terminal Output:

```

PS C:\Users\MANI\Desktop\week 13> & C:/Python312/python.exe "c:/Users/MANI/Desktop/week 13/q4.py"
Could not find platform independent libraries <prefix>
Enter a binary string: 101
Accepted: The string ends with '01'.
PS C:\Users\MANI\Desktop\week 13>

```

Q5) Develop a Python function that takes an NFA and returns the set of states that can be reached from a given state on a specific input symbol.

```

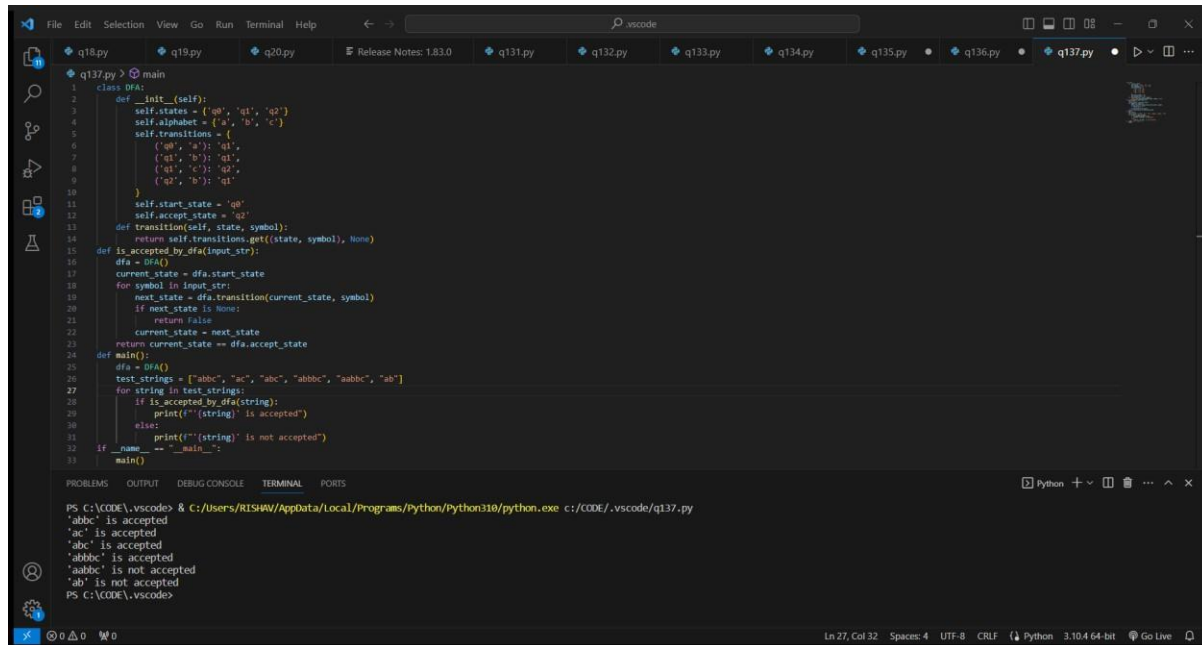
1 def nfa_transition(nfa, current_states, symbol):
2     next_states = set()
3
4     for state in current_states:
5         if state in nfa and symbol in nfa[state]:
6             next_states |= nfa[state][symbol]
7
8     return next_states
9
10 # Example NFA
11 nfa = {
12     0: {'a': {0, 1}},
13     1: {'b': {2}},
14     2: {'c': {3}},
15     3: {'d': {4}},
16     4: {'e': {5}},
17 }

```

Q6) Create a Python script to minimize a simple DFA with three states by merging equivalent states.

[illegible]

Q7) Implement a Python function that checks if a given string is accepted by a DFA that recognizes the pattern "ab*c"

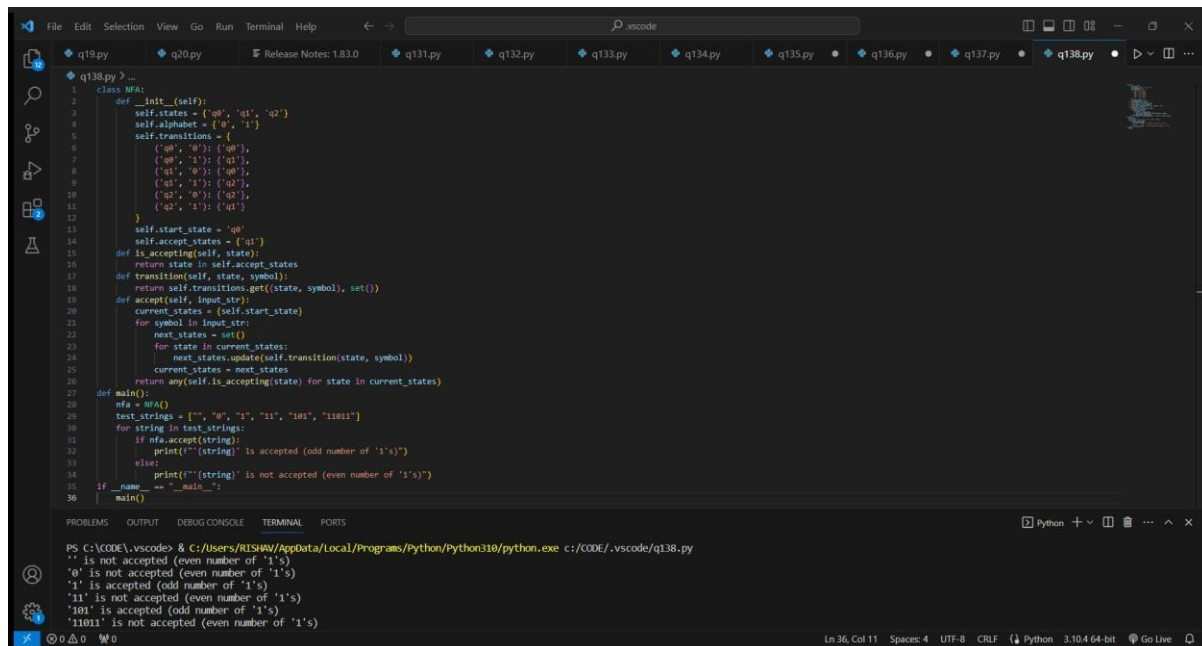


```
1 class DFA:
2     def __init__(self):
3         self.states = {'q0', 'q1', 'q2'}
4         self.alphabet = {'a', 'b', 'c'}
5         self.transitions = {
6             ('q0', 'a'): 'q1',
7             ('q1', 'b'): 'q1',
8             ('q1', 'c'): 'q2',
9             ('q2', 'b'): 'q1'
10        }
11        self.start_state = 'q0'
12        self.accept_state = 'q2'
13        def transition(self, state, symbol):
14            return self.transitions.get((state, symbol), None)
15        def is_accepted_by_dfa(input_str):
16            dfa = DFA()
17            current_state = dfa.start_state
18            for symbol in input_str:
19                next_state = dfa.transition(current_state, symbol)
20                if next_state is None:
21                    return False
22                current_state = next_state
23            return current_state == dfa.accept_state
24        def main():
25            dfa = DFA()
26            test_strings = ["abbc", "ac", "abc", "abbbc", "aabbc", "ab"]
27            for string in test_strings:
28                if is_accepted_by_dfa(string):
29                    print(f'{string} is accepted')
30                else:
31                    print(f'{string} is not accepted')
32        if __name__ == "__main__":
33            main()
```

PS C:\CODE\.vscode> & C:/Users/RISHW/AppData/Local/Programs/Python/Python310/python.exe c:/CODE/.vscode/q137.py

'abbc' is accepted
'ac' is accepted
'abc' is accepted
'abbbc' is accepted
'aabbc' is not accepted
'ab' is not accepted
PS C:\CODE\.vscode>

Q8) Write a Python program to create an NFA that accepts strings with an odd number of '1's.



```
1 class NFA:
2     def __init__(self):
3         self.states = {'q0', 'q1', 'q2'}
4         self.alphabet = {'0', '1'}
5         self.transitions = {
6             ('q0', '0'): 'q0',
7             ('q0', '1'): 'q1',
8             ('q1', '0'): 'q0',
9             ('q1', '1'): 'q2',
10            ('q2', '0'): 'q2',
11            ('q2', '1'): 'q1'
12        }
13        self.start_state = 'q0'
14        self.accept_states = {'q1', 'q2'}
15        def is_accepting(self, state):
16            return state in self.accept_states
17        def transition(self, state, symbol):
18            return self.transitions.get((state, symbol), set())
19        def accept(self, input_str):
20            current_states = {self.start_state}
21            for symbol in input_str:
22                next_states = set()
23                for state in current_states:
24                    next_states.update(self.transition(state, symbol))
25            current_states = next_states
26            return any(self.is_accepting(state) for state in current_states)
27        def main():
28            nfa = NFA()
29            test_strings = ["", "0", "1", "11", "101", "11011"]
30            for string in test_strings:
31                if nfa.accept(string):
32                    print(f'{string} is accepted (odd number of '1's)')
33                else:
34                    print(f'{string} is not accepted (even number of '1's)')
35        if __name__ == "__main__":
36            main()
```

PS C:\CODE\.vscode> & C:/Users/RISHW/AppData/Local/Programs/Python/Python310/python.exe c:/CODE/.vscode/q138.py

'' is not accepted (even number of '1's)
'0' is not accepted (even number of '1's)
'1' is accepted (odd number of '1's)
'11' is not accepted (even number of '1's)
'101' is accepted (odd number of '1's)
'11011' is not accepted (even number of '1's)

Q9) Write a Python program to create an NFA that accepts strings with an odd number of '1's.

[illegible]