SRM INSTITUTE OF SCIENCE & TECHNOLOGY KATTANKULATHUR CHENNAI

Name ➢ GAURAV GUPTA

Subject ➢ ADVANCED PROGRAMMING PRACTICE

Section ➢ W2

Roll No. ➢ RA2211026010284

Title ➢ ASSIGNMENT WEEK 12

ASSIGNMENT

WEEK 12

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

CODE:

```
from automata.fa.nfa import NFA

nfa=NFA(
    states={'q0','q1'},
    input_symbols={'a'},
    transitions={
        'q0':{'a':{'q1'}},
        'q1':{'a':{'q1'}}
    },
    initial_state='q0',
    final_states={'q1'}
)

input_string = input("Enter a string: ")
if nfa.accepts_input(input_string):
    print("Accepted")
else:
    print("Rejected")
```

Output:

```
= RESTART: D:/SRM/SEMESTERS/3rd SEM/Advance Programming/python/tutorial 12.py
Enter a string: a
Accepted
>>>
== RESTART: D:/SRM/SEMESTERS/3rd SEM/Advance Programming/python/tutorial 12.py =
Enter a string: abcd123
Rejected
```

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").

CODE:

```
from automata.fa.nfa import NFA
nfa=NFA(
    states={'q0','q1','q2'},
    input_symbols={'a','b'},
    transitions={
        'q0':{'a':{'q1'},'b':{'q2'}},
        'q1':{'b':{'q2'}},
        'q2':{'a':{'q1'}}
    },
    initial_state='q0',
    final_states={'q1','q2'}
)

input_string = input("Enter a string: ")
if nfa.accepts_input(input_string):
    print("Accepted")
else:
```

```
    print("Rejected")
```

Output:

```
>>>
= RESTART: D:/SRM/SEMESTERS/3rd SEM/Advance Programming/python/tutorial 12.py
Enter a string: ab
The NFA accepts the string 'ab'.
>>>
== RESTART: D:/SRM/SEMESTERS/3rd SEM/Advance Programming/python/tutorial 12.py =
Enter a string: ba
The NFA accepts the string 'ba'.
>>>
== RESTART: D:/SRM/SEMESTERS/3rd SEM/Advance Programming/python/tutorial 12.py =
Enter a string: BABA
The NFA does not accept the string 'BABA'.
```

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

CODE:

```
# Define the NFA transitions
nfa_transitions = {
    ('q0', 'a'): {'q1'},
    ('q1', 'b'): {'q2'},
    ('q2', 'a'): {'q1'},
    ('q0', ''): {'q2'}
}

# Initialize DFA variables
dfa_states = set()
dfa_transitions = {}
dfa_initial = ('q0',)
dfa_final = set()
```

```
# Initialize a queue for processing states
queue = [dfa_initial]

while queue:
    current_states = queue.pop()
    dfa_states.add(current_states)

    for symbol in {'a', 'b'}:
        next_states = set()
        for nfa_state in current_states:
            epsilon_transition = nfa_transitions.get((nfa_state, ''))
            if epsilon_transition:
                next_states.update(epsilon_transition)

            symbol_transition = nfa_transitions.get((nfa_state, symbol))
            if symbol_transition:
                next_states.update(symbol_transition)

        next_states = tuple(sorted(next_states))
        dfa_transitions[(current_states, symbol)] = next_states

        if next_states not in dfa_states:
            queue.append(next_states)

# Identify final states
for dfa_state in dfa_states:
    for nfa_state in dfa_state:
        if nfa_state in {'q1', 'q2'}:
            dfa_final.add(dfa_state)
```

```
# Print the DFA
print("DFA States:", dfa_states)
print("DFA Transitions:", dfa_transitions)
print("DFA Initial State:", dfa_initial)
print("DFA Final States:", dfa_final)
```

Output:

```
>>>
== RESTART: D:/SRM/SEMESTERS/3rd SEM/Advance Programming/python/tutorial 12.py =
DFA States: {('q2',), ('q0',), ('q1', 'q2'), ('q1',), ()}
DFA Transitions: {(('q0',), 'a'): ('q1', 'q2'), (('q0',), 'b'): ('q2',), (('q2',
), 'a'): ('q1',), (('q2',), 'b'): (), ((), 'a'): (), ((), 'b'): (), (('q1',), 'a
'): (), (('q1',), 'b'): ('q2',), (('q1', 'q2'), 'a'): ('q1',), (('q1', 'q2'), 'b
'): ('q2',)}
DFA Initial State: ('q0',)
DFA Final States: {('q2',), ('q1', 'q2'), ('q1',)}
>>>
```

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

CODE:

```
# Get the input binary string from the user
input_string = input("Enter a binary string: ")

# Initialize current state
current_state = 0

# Iterate through characters in the input string
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

# Check if the last two characters are '01'
if input_string[-2:] == '01':
    print("Accepted: The string ends with '01'.")
else:
    print("Rejected: The string does not end with '01'.")
```

Output:

```
== RESTART: D:/SRM/SEMESTERS/3rd SEM/Advance Programming/python/tutorial 12.py =
Enter a binary string: 1001
Accepted: The string ends with '01'.
>>>
== RESTART: D:/SRM/SEMESTERS/3rd SEM/Advance Programming/python/tutorial 12.py =
Enter a binary string: 1010
Rejected: The string does not end with '01'.
>>>
```

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.

CODE:

```
# Define the NFA properties
nfa = {
    'states': {'q0', 'q1', 'q2'},
    'alphabet': {'a', 'b'},
    'transitions': {
        'q0': {'a': {'q1'}, 'b': {'q2'}},
        'q1': {'a': {'q1'}, 'b': {'q2'}},
        'q2': {'a': {'q2'}, 'b': {'q2'}}
    },
```

```
    'initial_state': 'q0',
    'final_states': {'q2'}
}

# Get input state and symbol from user
state = input("Enter a state: ")
symbol = input("Enter an input symbol: ")

# Find reachable states
reachable_states = set()

for transition_state, transitions in nfa['transitions'].items():
    if transition_state == state and symbol in transitions:
        reachable_states.update(transitions[symbol])

# Print reachable states
print(f"States reachable from '{state}' on input '{symbol}':",
reachable_states)
```

Output:

```
== RESTART: D:/SRM/SEMESTERS/3rd SEM/Advance Programming/python/tutorial 12.py =
Enter a state: q0
Enter an input symbol: a
States reachable from 'q0' on input 'a': {'q1'}
>>>
```

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

CODE:

```
# Define the DFA properties
dfa = {
```

```
    'states': ['q0', 'q1', 'q2'],
    'alphabet': {'0', '1'},
    'transitions': {
        'q0': {'0': 'q0', '1': 'q1'},
        'q1': {'0': 'q2', '1': 'q1'},
        'q2': {'0': 'q0', '1': 'q1'},
    },
    'initial_state': 'q0',
    'final_states': ['q2']
}

# Step 1: Split states into two sets - final states and non-final states
final_states = set(dfa['final_states'])
non_final_states = set(dfa['states']) - final_states

# Step 2: Initialize partition with the two sets
partition = [final_states, non_final_states]

# Step 3: Refine the partition using the transitions
while True:
    new_partition = []

    for group in partition:
        new_groups = []

        for symbol in dfa['alphabet']:
            target_states = set()
            for state in group:
                target_state = dfa['transitions'][state][symbol]
                target_states.add(target_state)
```

```
            for subgroup in new_groups:
                if target_states == subgroup:
                    subgroup.update(group)
                    break
            else:
                new_groups.append(target_states)

        if len(new_groups) > 1:
            break
    else:
        new_groups = [group]

    new_partition.extend(new_groups)

    if new_partition == partition:
        break

    partition = new_partition

# Step 4: Build the new DFA
new_states = []
new_transitions = {}

for group in partition:
    new_state = ','.join(sorted(group))
    new_states.append(new_state)

    for state in group:
        if state in final_states:
```

```
            final_state = new_state
            break
    else:
        final_state = None

    new_transitions[new_state] = {}

    for symbol in dfa['alphabet']:
        target_states = set()
        for state in group:
            target_state = dfa['transitions'][state][symbol]
            target_states.add(target_state)

        for subgroup in partition:
            if target_states == subgroup:
                new_transitions[new_state][symbol] = ','.join(sorted(subgroup))
                break

minimized_dfa = {
    'states': new_states,
    'alphabet': dfa['alphabet'],
    'transitions': new_transitions,
    'initial_state': ','.join(sorted(partition[0])),
    'final_states': [final_state] if final_state is not None else [],
}

# Print the minimized DFA
print(minimized_dfa)
```

Output:

```
>>>
== RESTART: D:/SRM/SEMESTERS/3rd SEM/Advance Programming/python/tutorial 12.py =
{'states': ['q1', 'q2'], 'alphabet': {'1', '0'}, 'transitions': {'q1': {'1': 'q1
', '0': 'q2'}, 'q2': {'1': 'q1'}}, 'initial_state': 'q1', 'final_states': ['q2']
}
>>>
```

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

CODE:

```
# Define the DFA properties
dfa = {
    'states': {'q0', 'q1', 'q2', 'q3'},
    'alphabet': {'a', 'b', 'c'},
    'transitions': {
        'q0': {'a': 'q1'},
        'q1': {'b': 'q2'},
        'q2': {'b': 'q2', 'c': 'q3'},
        'q3': {}
    },
    'initial_state': 'q0',
    'final_states': {'q3'}
}

# Get input string from user
input_string = input("Enter a string: ")

# Initialize current state
current_state = dfa['initial_state']
```

```
# Iterate through characters in the input string
for symbol in input_string:
    if symbol not in dfa['alphabet']:
        print(f"Invalid symbol '{symbol}' in the input.")
        break
    current_state = dfa['transitions'][current_state].get(symbol)
    if current_state is None:
        break

# Check if the current state is a final state
if current_state in dfa['final_states']:
    print("Accepted")
else:
    print("Rejected")
```

Output:

```
>>>
    == RESTART: D:/SRM/SEMESTERS/3rd SEM/Advance Programming/python/tutorial 12.py =
    Enter a string: abbbc
    Accepted
>>>
    == RESTART: D:/SRM/SEMESTERS/3rd SEM/Advance Programming/python/tutorial 12.py =
    Enter a string: abccc
    Rejected
```

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

CODE:

```
# Define the NFA properties
nfa = {
    'states': {'q0', 'q1'},
    'input_symbols': {'0', '1'},
```

```
    'transitions': {
        'q0': {'0': {'q0'}, '1': {'q0', 'q1'}},
        'q1': {'0': {'q1'}, '1': {'q1'}}
    },
    'initial_state': 'q0',
    'final_states': {'q1'}
}

# Get input string from user
input_string = input("Enter a string: ")

# Initialize current states as a set with the initial state
current_states = {'q0'}

# Iterate through characters in the input string
for symbol in input_string:
    if symbol not in nfa['input_symbols']:
        print(f"Invalid symbol '{symbol}' in the input.")
        break
    next_states = set()
    for state in current_states:
        next_states.update(nfa['transitions'][state].get(symbol, set()))
    current_states = next_states

# Check if the final state is in the set of current states and the input string has an odd number of '1's
if 'q1' in current_states and input_string.count('1') % 2 != 0:
    print("Accepted")
else:
    print("Rejected")
```

Output:

```
>>>
== RESTART: D:/SRM/SEMESTERS/3rd SEM/Advance Programming/python/tutorial 12.py =
Enter a string: 1111
Rejected
>>>
== RESTART: D:/SRM/SEMESTERS/3rd SEM/Advance Programming/python/tutorial 12.py =
Enter a string: 1011
Accepted
```

Q9) Develop a Python function that converts a simple regular expression like "a(b|c)*" into an equivalent NFA.

CODE:

```
# Define the NFA properties
nfa = {
    'states': {'q0', 'q1', 'q2', 'q3'},
    'input_symbols': {'a', 'b', 'c'},
    'transitions': {
        'q0': {'a': {'q1'}},
        'q1': {'b': {'q2', 'q1'}, 'c': {'q2', 'q1'}},
        'q2': {'b': {'q2'}, 'c': {'q2'}},
        'q3': {}
    },
    'initial_state': 'q0',
    'final_states': {'q2'}
}

# Get input string from user
input_string = input("Enter a string: ")

# Initialize current states as a set with the initial state
current_states = {'q0'}
```

```
# Iterate through characters in the input string
for symbol in input_string:
    if symbol not in nfa['input_symbols']:
        print(f"Invalid symbol '{symbol}' in the input.")
        break
    next_states = set()
    for state in current_states:
        next_states.update(nfa['transitions'][state].get(symbol, set()))
    current_states = next_states

# Check if the final state is in the set of current states
if 'q2' in current_states:
    print("Accepted")
else:
    print("Rejected")
```

Output: