**LAB ASSIGNMENT-1**

Submitted for

**COMPILER CONSTRUCTION (UCS802)**

Submitted by

**Aryan Shanker Saxena**

**102103613**

**4 COE 22**

Submitted to

**Mr. Rajesh**



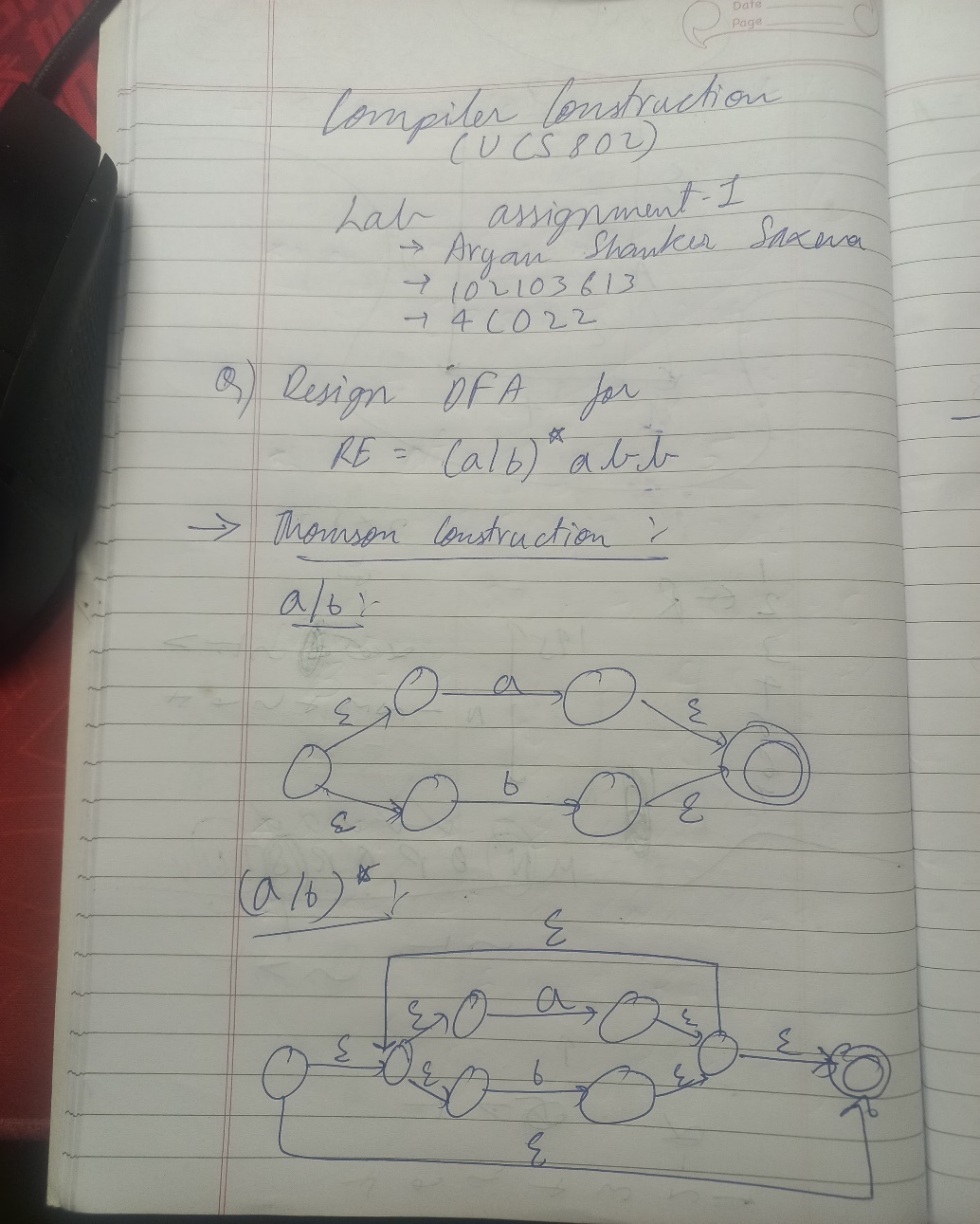
Computer Science and Engineering Department

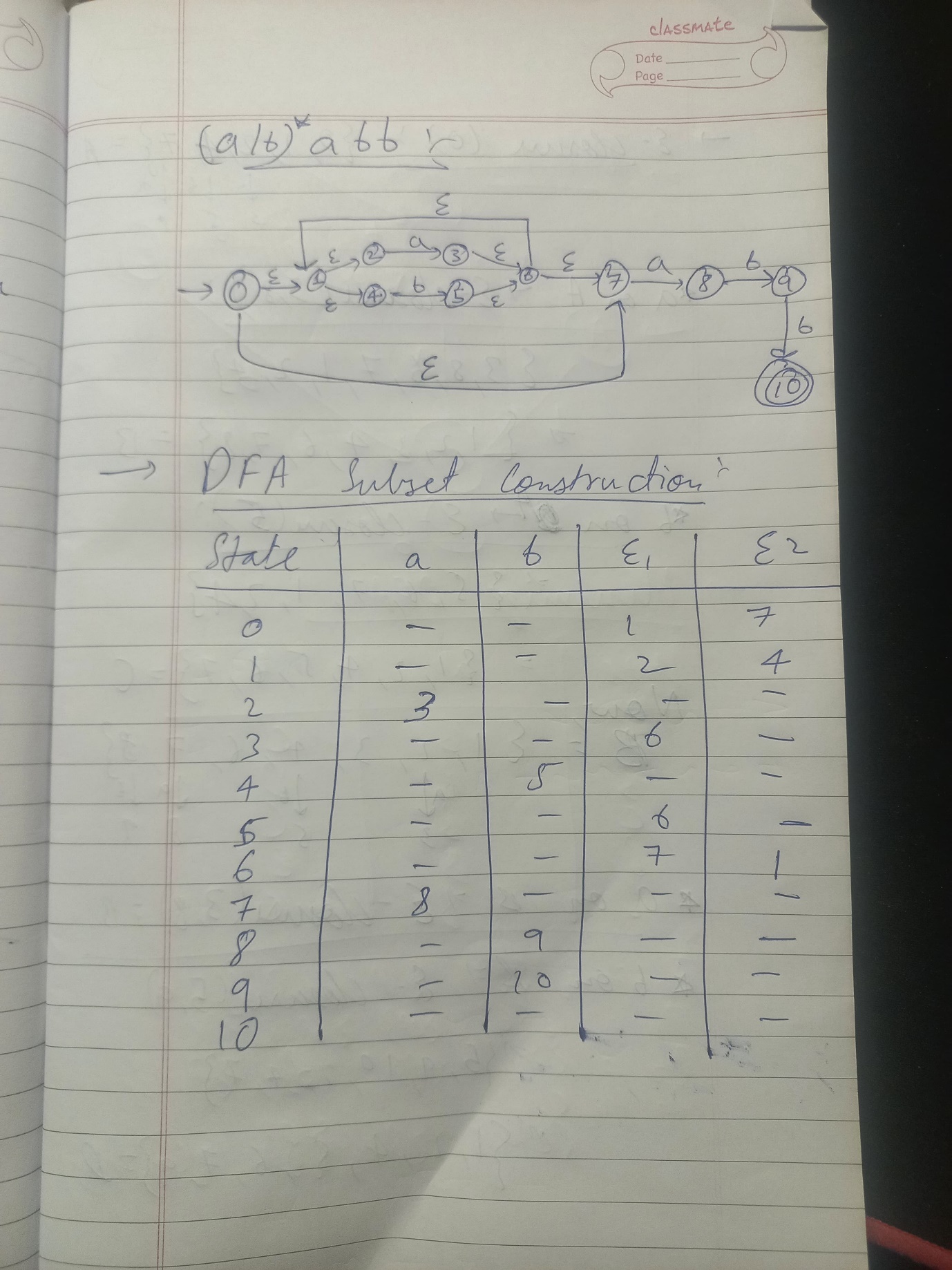
Thapar Institute of Engineering and Technology, Patiala

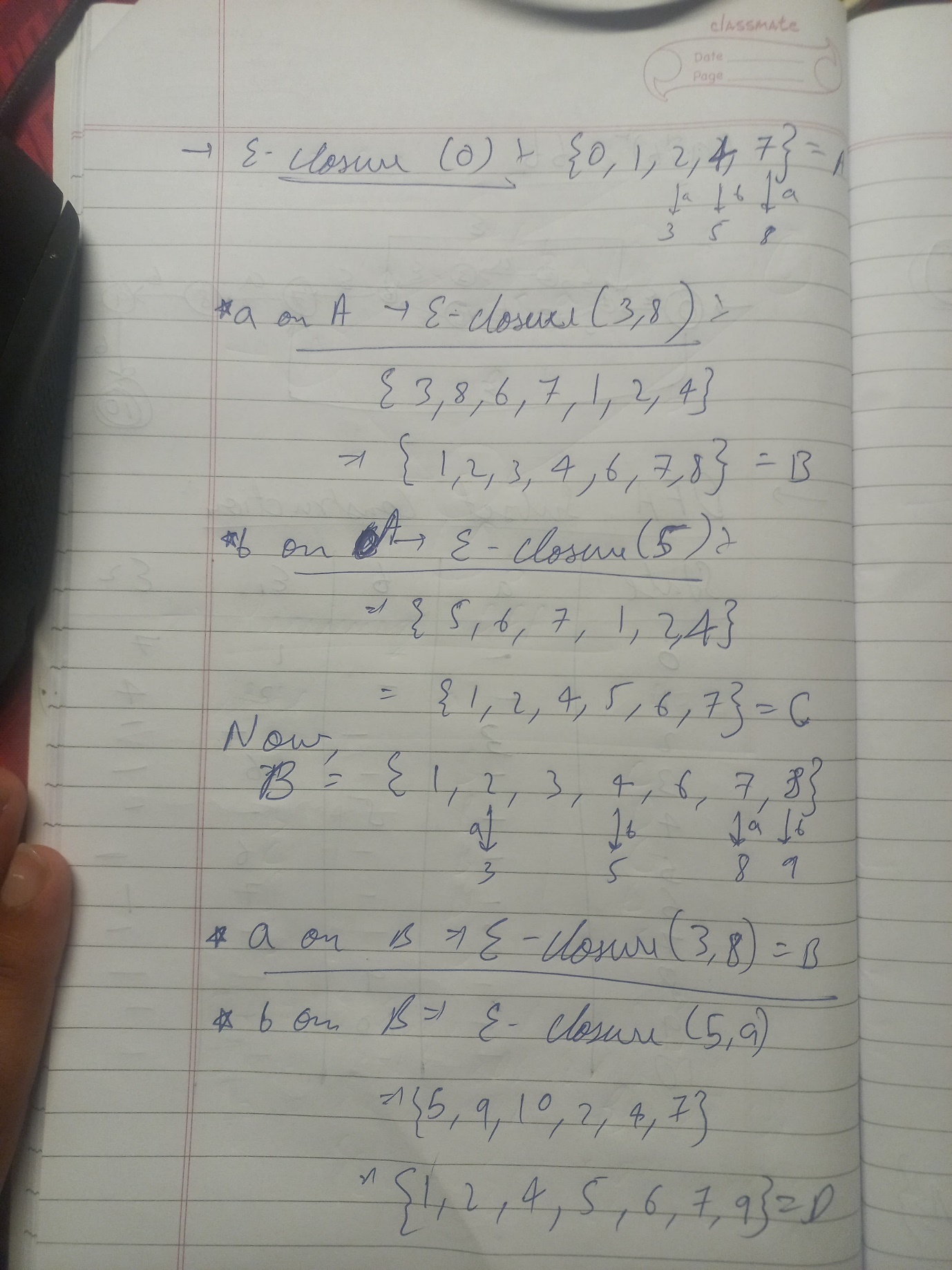
**QUESTION:**

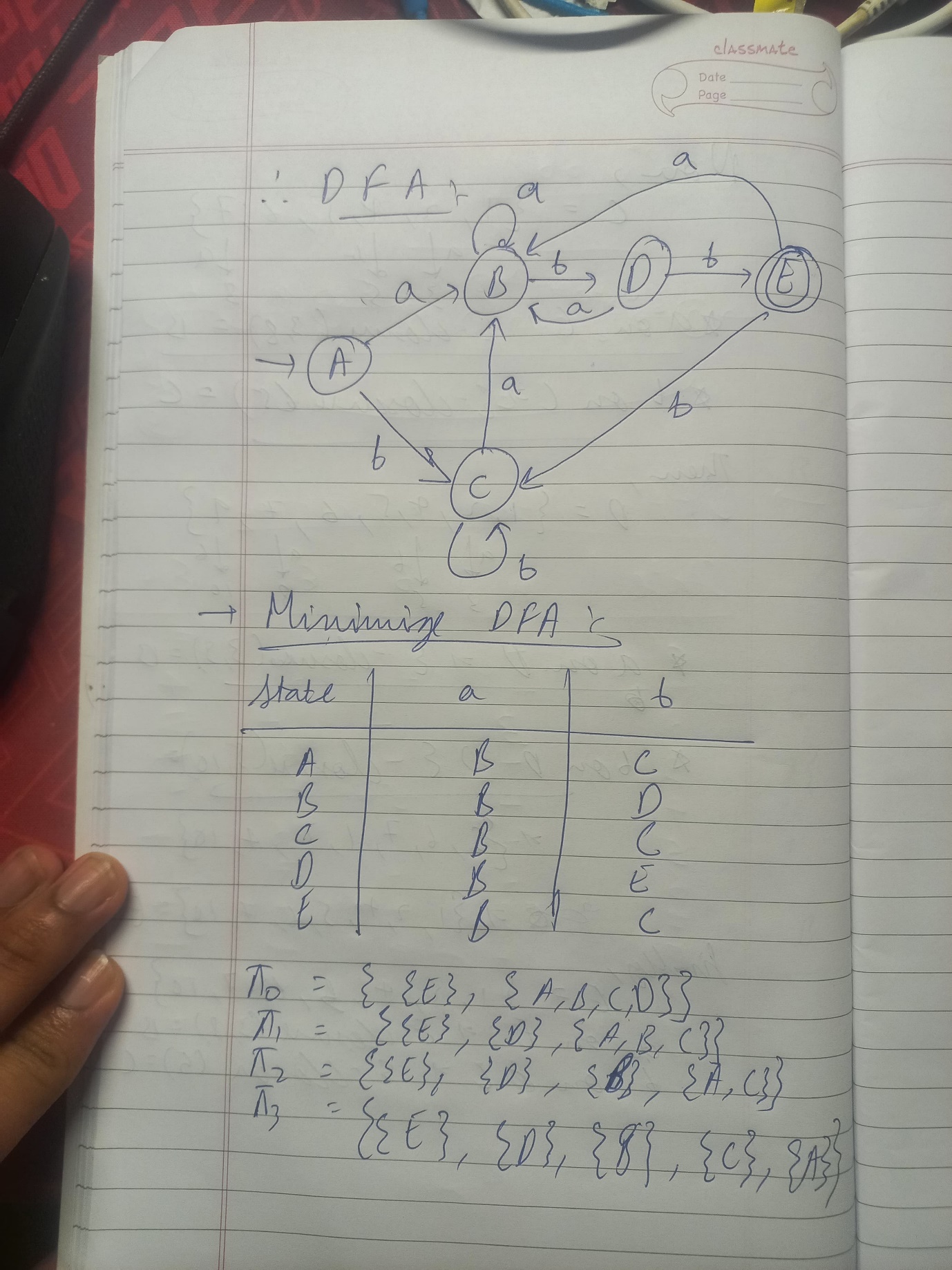
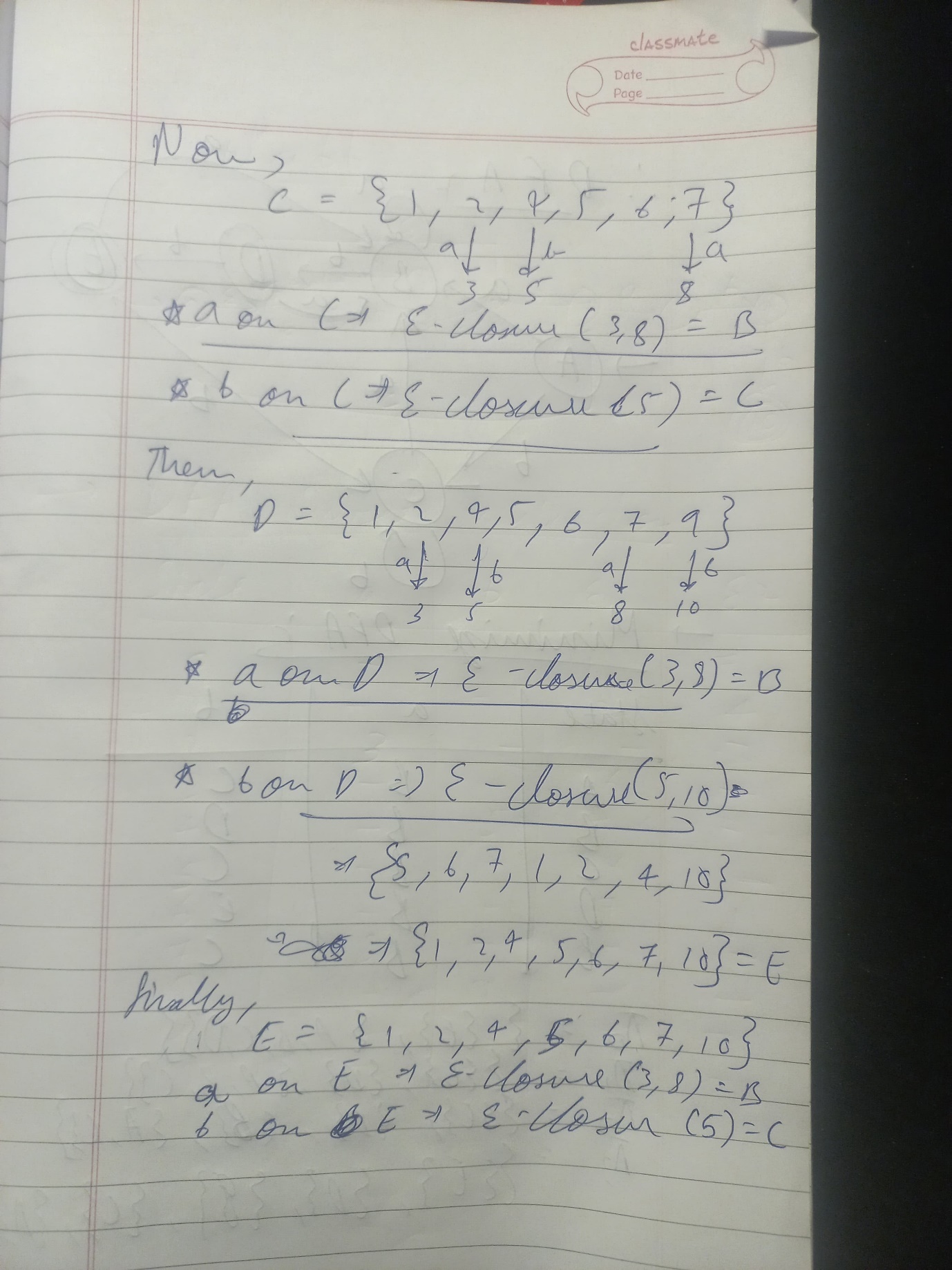
Design a Minimized DFA for the Regular Expression (a/b)\*abb i.e. All strings ending with abb.  
  
Note: ***THE CODE IS TYPE WRITTEN IT IS JUST THE BACKGROUND OF VSCode THAT MIGHT MAKE IT LOOK LIKE AN IMAGE BUT IT IS PURE TEXT***

**SOLVED ON PAPER:**

****

****

****

****

**PYTHON CODE:**

class NFA:

    def \_\_init\_\_(self):

        self.states = {i: {'a': None, 'b': None, 'ε1': None, 'ε2': None} *for* i *in* range(11)}

    def add\_transition(self, state, symbol, next\_state):

*if* symbol in self.states[state]:

            self.states[state][symbol] = next\_state

    def print\_nfa(self):

        print("State | a  | b  | ε1 | ε2")

        print("-------------------------")

*for* state, transitions *in* self.states.items():

            print(f"  {state}  | {transitions['a']} | {transitions['b']} | {transitions['ε1']} | {transitions['ε2']}")

*# Initializing the NFA with correct transitions*

nfa = NFA()

nfa.add\_transition(0, 'ε1', 1)

nfa.add\_transition(0, 'ε2', 7)

nfa.add\_transition(1, 'ε1', 2)

nfa.add\_transition(1, 'ε2', 4)

nfa.add\_transition(2, 'a', 3)

nfa.add\_transition(3, 'ε1', 6)

nfa.add\_transition(4, 'b', 5)

nfa.add\_transition(5, 'ε1', 6)

nfa.add\_transition(6, 'ε1', 7)

nfa.add\_transition(6, 'ε2', 1)

nfa.add\_transition(7, 'a', 8)

nfa.add\_transition(8, 'b', 9)

nfa.add\_transition(9, 'b', 10)

print("Regular Expression: (a/b)\*abb")

print("")

print("Submitted by: Aryan Shanker Saxena (102103613)")

print("")

nfa.print\_nfa()

print("")

*# epsilon\_closure function to use the nfa.states dictionary*

def epsilon\_closure(nfa\_states, state):

    stack = [state]

    closure = set(stack)

*while* stack:

        current = stack.pop()

*for* next\_state *in* [nfa\_states[current]['ε1'], nfa\_states[current]['ε2']]:

*if* next\_state and next\_state not in closure:

                closure.add(next\_state)

                stack.append(next\_state)

*return* closure

epsilon\_closure\_set = epsilon\_closure(nfa.states, 0)

print("ε-closure(0):", epsilon\_closure\_set)

epsilon\_closure\_set = epsilon\_closure(nfa.states, 1)

print("ε-closure(1):", epsilon\_closure\_set)

epsilon\_closure\_set = epsilon\_closure(nfa.states, 2)

print("ε-closure(2):", epsilon\_closure\_set)

epsilon\_closure\_set = epsilon\_closure(nfa.states, 3)

print("ε-closure(3):", epsilon\_closure\_set)

epsilon\_closure\_set = epsilon\_closure(nfa.states, 4)

print("ε-closure(4):", epsilon\_closure\_set)

epsilon\_closure\_set = epsilon\_closure(nfa.states, 5)

print("ε-closure(5):", epsilon\_closure\_set)

epsilon\_closure\_set = epsilon\_closure(nfa.states, 6)

print("ε-closure(6):", epsilon\_closure\_set)

epsilon\_closure\_set = epsilon\_closure(nfa.states, 7)

print("ε-closure(7):", epsilon\_closure\_set)

epsilon\_closure\_set = epsilon\_closure(nfa.states, 8)

print("ε-closure(8):", epsilon\_closure\_set)

epsilon\_closure\_set = epsilon\_closure(nfa.states, 9)

print("ε-closure(9):", epsilon\_closure\_set)

epsilon\_closure\_set = epsilon\_closure(nfa.states, 10)

print("ε-closure(10):", epsilon\_closure\_set)

print("")

*# The subset construction to pass the correct argument*

def subset\_construction(nfa):

    dfa\_states = {}

    unmarked\_states = []

    initial\_state = epsilon\_closure(nfa.states, 0)

    dfa\_states[frozenset(initial\_state)] = {}

    unmarked\_states.append(initial\_state)

*while* unmarked\_states:

        current\_set = unmarked\_states.pop(0)

*for* symbol *in* ['a', 'b']:

            new\_set = set()

*for* state *in* current\_set:

*if* nfa.states[state][symbol] is not None:

                    new\_set.update(epsilon\_closure(nfa.states, nfa.states[state][symbol]))

*if* new\_set and frozenset(new\_set) not in dfa\_states:

                unmarked\_states.append(new\_set)

                dfa\_states[frozenset(new\_set)] = {}

            dfa\_states[frozenset(current\_set)][symbol] = frozenset(new\_set)

*return* dfa\_states

*# Construct DFA from NFA*

dfa = subset\_construction(nfa)

accepting\_states = set()

*for* state *in* dfa:

*if* 10 in state:

        accepting\_states.add(state)

*# Function to check if string is accepted*

def check\_string(dfa, string):

    current\_state = frozenset(epsilon\_closure(nfa.states, 0))

*for* char *in* string:

*if* char in dfa[current\_state]:

            current\_state = dfa[current\_state][char]

*else*:

*return* "Not Accepted"

*if* current\_state in accepting\_states:

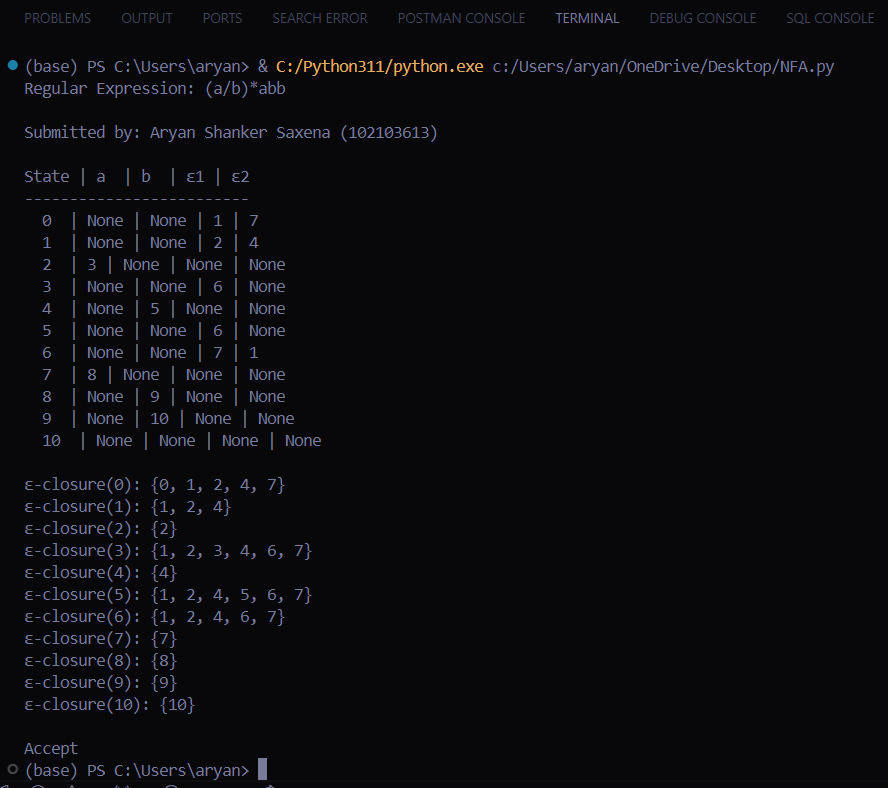
*return* "Accept"

*return* "Not Accepted"

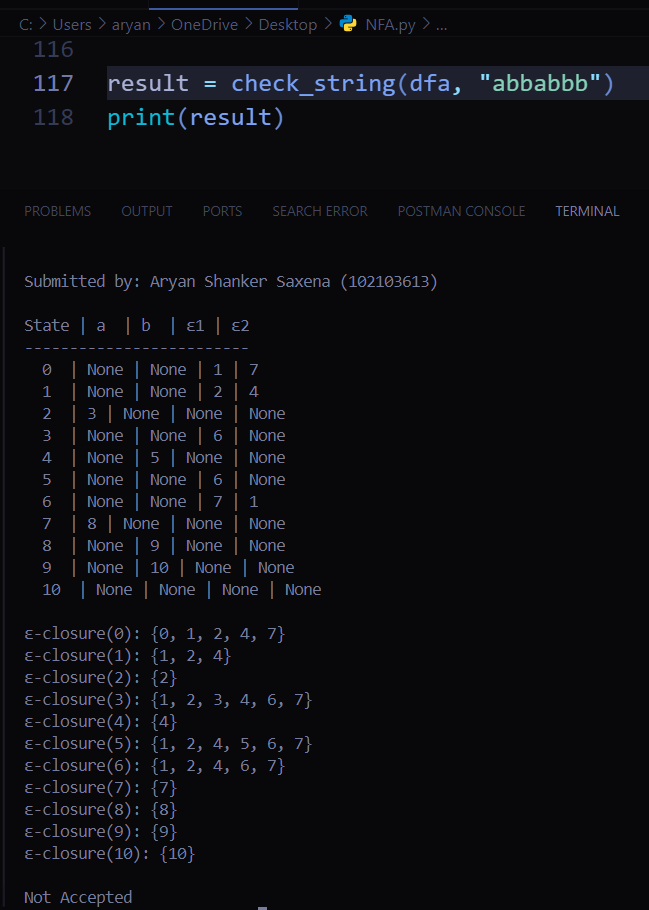
result = check\_string(dfa, "abbabb")

print(result)

**OUTPUT:**

****

**Output when the string does not end in abb:**

****