AIM:

To convert an NFA to DFA using python.

ALGORITHM:

- Create a dictionary for the NFA and enter the number of states and the transitions each state can have.
- Enter the state and transition for each state.
- Print the NFA transition table
- Enter the final state.
- Create a new_states_list to hold all the new states created in DFA, dictionary to store the DFA, keys_list to store all the states in NFA and DFA and path_list to store the path.
- Compute the first row of DFA transition table by taking the first row of NFA table.
- Then, compute the other rows. Include the states that have newly been created also. For the new states, take the union for each state and add them to a temporary list. Finally add this to the DFA.
- Finally. Print the DFA transition table.

PYTHON CODE

FILE 1

```
class NFA:
    def __init__(self):
        self.num_states = 0
        self.states = []
        self.symbols = []
        self.num_accepting_states = 0
        self.accepting_states = []
        self.start_state = 0
        self.transition_functions = []

    def init_states(self):
        self.states = list(range(self.num_states))

    def print_nfa(self):
        print(self.num_states)
        print(self.states)
        print(self.symbols)
        print(self.num_accepting_states)
        print(self.accepting_states)
        print(self.start_state)
        print(self.transition_functions)

    def construct_nfa_from_file(self, lines):
        self.num_states = int(lines[0])
        self.init_states()
        self.symbols = list(lines[1].strip())

        accepting_states_line = lines[2].split(" ")
        for index in range(len(accepting_states_line)):
            if index == 0:
                self.num_accepting_states = int(accepting_states_line[index])
            else:
                self.accepting_states.append(int(accepting_states_line[index]))

        self.startState = int(lines[3])

        for index in range(4, len(lines)):
            transition_func_line = lines[index].split(" ")

            starting_state = int(transition_func_line[0])
            transition_symbol = transition_func_line[1]
            ending_state = int(transition_func_line[2])

            transition_function = (starting_state, transition_symbol, ending_state);
            self.transition_functions.append(transition_function)

class DFA:
    def __init__(self):
        self.num_states = 0
        self.symbols = []
        self.num_accepting_states = 0
        self.accepting_states = []
        self.start_state = 0
        self.transition_functions = []
        self.q = []

    def convert_from_nfa(self, nfa):
        self.symbols = nfa.symbols
        self.start_state = nfa.start_state

        nfa_transition_dict = {}
        dfa_transition_dict = {}

        # Combine NFA transitions
        for transition in nfa.transition_functions:
            starting_state = transition[0]
            transition_symbol = transition[1]
            ending_state = transition[2]
            if (starting_state, transition_symbol) in nfa_transition_dict:
                nfa_transition_dict[(starting_state, transition_symbol)].append(ending_state)
            else:
                nfa_transition_dict[(starting_state, transition_symbol)] = [ending_state]

        self.q.append((0,))

        # Convert NFA transitions to DFA transitions
        for dfa_state in self.q:
            for symbol in nfa.symbols:
                if len(dfa_state) == 1 and (dfa_state[0], symbol) in nfa_transition_dict:
                    dfa_transition_dict[(dfa_state, symbol)] = nfa_transition_dict[(dfa_state[0], symbol)]

                    if tuple(dfa_transition_dict[(dfa_state, symbol)]) not in self.q:
                        self.q.append(tuple(dfa_transition_dict[(dfa_state, symbol)]))
                else:
                    destinations = []
                    final_destination = []

                    for nfa_state in dfa_state:
                        if (nfa_state, symbol) in nfa_transition_dict and nfa_transition_dict[(nfa_state, symbol)] not in destinations:
                            destinations.append(nfa_transition_dict[(nfa_state, symbol)])

                    if not destinations:
                        final_destination.append(None)
                    else:
                        for destination in destinations:
                            for value in destination:
                                if value not in final_destination:
                                    final_destination.append(value)

                    dfa_transition_dict[(dfa_state, symbol)] = final_destination

                    if tuple(final_destination) not in self.q:
                        self.q.append(tuple(final_destination))

        # Convert NFA states to DFA states
        for key in dfa_transition_dict:
            self.transition_functions.append((self.q.index(tuple(key[0])), key[1], self.q.index(tuple(dfa_transition_dict[key]))))

        for q_state in self.q:
            for nfa_accepting_state in nfa.accepting_states:
                if nfa_accepting_state in q_state:
                    self.accepting_states.append(self.q.index(q_state))
                    self.num_accepting_states += 1

    def print_dfa(self):
        print(len(self.q))
        print("".join(self.symbols))
        print(str(self.num_accepting_states) + " " + " ".join(str(accepting_state) for accepting_state in self.accepting_states))
        print(self.start_state)
        for transition in sorted(self.transition_functions):
            print(" ".join(str(value) for value in transition))
```

MAIN FILE

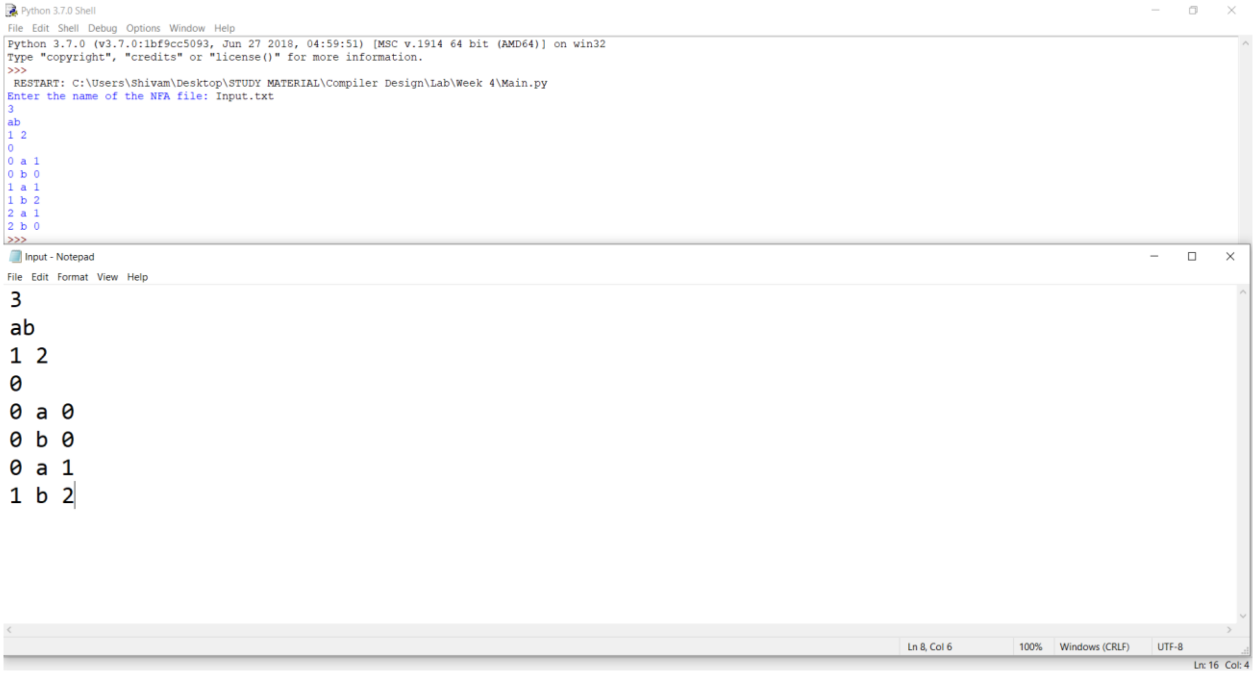
```
import sys
from finite_automata import NFA, DFA

if sys.version_info >= (3, 0):
    filename = input('Enter the name of the NFA file: ')
elif sys.version_info >= (2, 0):
    filename = raw_input('Enter the name of the NFA file: ')
else:
    print("Please update python to version 2.0 or newer")
    quit()

file = open(filename, 'r')
lines = file.readlines()
file.close()

nfa = NFA()
dfa = DFA()

nfa.construct_nfa_from_file(lines)
dfa.convert_from_nfa(nfa)

dfa.print_dfa()
```

EXAMPLE FILES BEING READ FOR TESTING

FILE_1 – Input.txt

```
3
    ab
    1 2
    0
    0 a 0
    0 b 0
    0 a 1
    1 b 2
```

IMPLEMENTATION



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
 RESTART: C:\Users\Shivam\Desktop\STUDY MATERIAL\Compiler Design\Lab\Week 4\Main.py
Enter the name of the NFA file: Input.txt
3
ab
1 2
0
0 a 1
0 b 0
1 a 1
1 b 2
2 a 1
2 b 0
>>> 
```

```
3
ab
1 2
0
0 a 0
0 b 0
0 a 1
1 b 2
```

RESULT

Code was successfully implemented and the output was verified.