CSC 410: Theoretical Foundations
Project 1

1 Purpose

In this project, you will learn more about how Finite Automata work.

2 Description & Coding Requirements

In this project you will implement a Deterministic Finite Automata (DFA) simulator. The input is a file containing a description of a DFA (described below). After constructing the DFA, your program should prompt the user for a string to evaluate. Your program's output is a computation and an indication of acceptance or rejection.

You may implement your code in either Java or Python, but you must implement to functionality of the DFA yourself. Do not use any packages or classes that you may find in the APIs.

3 Input

The DFA will be input from a file named DFA.txt. The file will be in the following format:

> **Line 1:** The first line of your file will be a single integer that indicates how many states the DFA contains. *A note about states*: States are represented by consecutive integers starting from zero. State zero is always the start state. All states of the DFA must be represented; this includes the dead state.
>
> **Line 2:** The second line of the file contains a space-separated list of integers that represent accepting states.
>
> **Line 3:** Alphabet: a space-separated list of characters in the DFA's alphabet.
>
> **Lines 4 – end:** Transitions: These will be represented in a table indexed by state number and character. The input file will have one line for each row of the table. Each row consists of integers that represent the states the machine will transition to on each of the alphabet characters. If the DFA has seven states and three characters, there will be seven lines containing three integers each. The first row corresponding to the transitions from state zero, the second from state one, etc. Note that all transitions must be represented; do not assume a missing transition for a character is to the dead state.

4 Submission

Source code is to be submitted on D2L. Your program should be well-documented and well-structured. There should be a header at the top of your program in the following format:

```
\*************************************************************
*
* <Program name> *
* <Your name> *
* *
* <Program Description> *
* *
*************************************************************/
```

The program description should provide an overview of your program.

Functions/methods should have similar headers, with the function name in place of the program name, and a description of the purpose of the function:

```
\*************************************************************
* <Function name> *
*************************************************************/
\\Purpose:
=============================================================
```

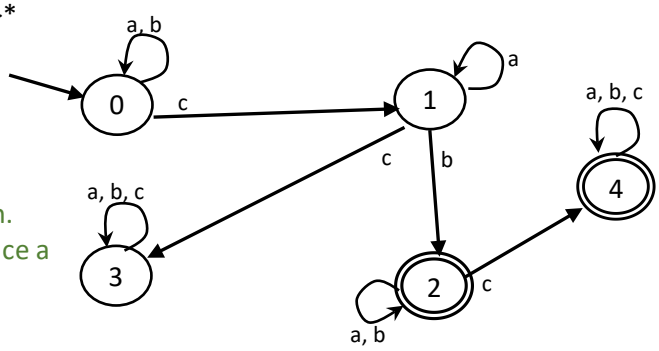
Sample input file

```
5
2 4
a b c
0 0 1
1 2 3
2 2 4
3 3 3
4 4 4
```

Regular Expression

{a, b}*ca*b{a, b}*c{a, b, c}*

Graph Representation



This is just for clarification. You do not need to produce a regular expression or graph representation of the input file.

Sample Output

```
>>>Loading DFA.txt…
>>>Please enter a string to evaluate:
abbbcab
>>>Computation…
0,abbbcab -> 0,bbbcab
0,bbbcab -> 0,bbcab
0,bbcab -> 0,bcab
0,cab -> 1,ab
1,ab -> 1,b
1,b -> 2,{e}
ACCEPTED
>>>Please enter a string to evaluate:
d
>>>Computation…
0,d -> INVALID INPUT
REJECTED
>>> Please enter a string to evaluate:
ccc
>>>Computation…
0,ccc -> 1,cc
1,cc -> 3,c
3,c -> 3,{e}
REJECTED
>>>Please enter a string to evaluate:
Quit
>>>Goodbye!
```

Symbol for the empty string

Rubric

Code compiles: _____/5

Code reads a file in the local directory called “DFA.txt”: _____/5

Code prompts user for an input string: _____/5

Correctly produces the computation for input strings: _____/15

Correctly accepts a valid string: _____/15

Correctly rejects invalid strings: _____/15

Handles invalid input characters by

indicating the character and rejecting the string: _____/5

Total: _____ / 65