Overview

The purpose of this assignment is to broaden your understanding of *deterministic finite automata* and their *acceptance relation*. Recall that a *deterministic finite automaton* is defined as a 5-tuple of the form $(Q, \Sigma, F, q_0, \delta)$ where:

- Q is a finite set of states;
- Σ is an input alphabet;
- $F \subseteq Q$ is a set of final states;
- $q_0 \in Q$ is the initial (or start) state; and
- $\delta : Q \times \Sigma \rightarrow Q$ is a transition function

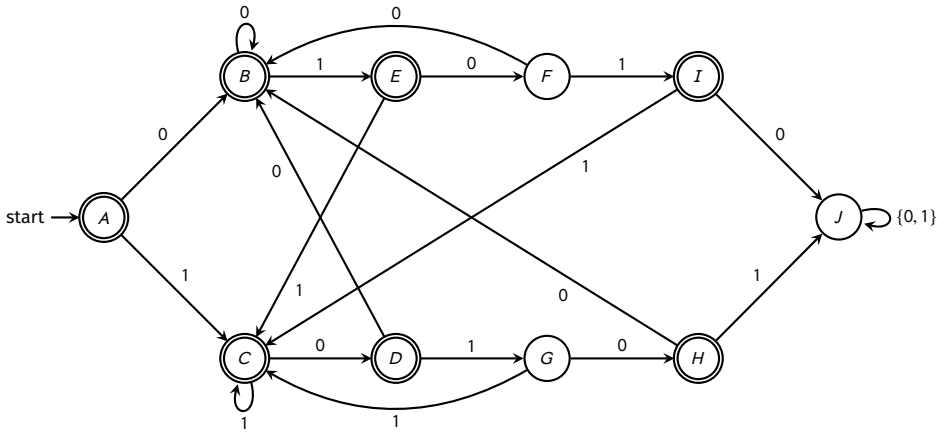
and that a string ω is accepted by a DFA M, if and only if there exists a path from the start state of M to some final state in M labeled by the constituent characters of ω. In this assignment, you will implement a program in Java which will accept as its input a pair of file names, the first of which contains a textual representation of a DFA, and the second of which contains a collection of strings, and determines whether or not the given strings are accepted by the specified automaton.

Specification

Define a program called `DFACheck` which will accept as its input a pair of file names, the first of which contains a textual representation of a DFA, and the second of which contains a collection of strings, and determines whether or not the given strings are accepted by the specified automaton.

DFA Input File Specification

Consider the DFA, M_0 shown in the figure below:



M_0 will be represented in plain text form as follows:

```
1  % Q
2  A
3  B
4  C
5  D
6  E
7  F
8  G
9  H
10 I
11 J
12 % Sigma
13 0
14 1
15 % F
16 A
17 B
18 C
19 D
20 E
21 H
22 I
23 % Qo
24 A
25 % Delta
26 A 0 B
27 A 1 C
28 B 0 B
29 B 1 E
30 C 0 D
31 C 1 C
32 D 0 B
33 D 1 G
34 E 0 F
35 E 1 C
36 F 0 B
37 F 1 I
38 G 0 H
39 G 1 C
40 H 0 B
41 H 1 J
42 I 0 J
43 I 1 C
44 J 0 J
45 J 1 J
```

Note that in this representation, those lines preceded by a % sign (i.e., lines 1, 12, 15, 25, and 27) are to be treated as single line comments in Java - any content on the line after the % sign is to be ignored and is entirely optional. Also note, that for this assignment you are guaranteed the following about the DFA input file:

- the DFA within it is correctly specified (i.e., is both a a DFA, and in the correct textual form), and

- that the order of the DFA's constituent parts are as in the example above (which by design corresponds with how the definition of a DFA is formulated).

String Input File Specification

For this assignment you may assume that the string input file is formatted with exactly one string per line and that the strings are guaranteed to be formed from characters of the input alphabet (which in the context of this example is $\{0, 1\}$). The following is one possible input file[1] to be used with the deterministic finite automaton M_0 from the previous section:

```
1  100100
2  111
3  00100100
4  10101
5  01010101
```

Usage

The program `DFACheck` is intended to be run via the following command-line invocation:

```
1  $ java DFACheck dfa.txt input.txt
```

If no input file is specified or if the input file is not found, an appropriate error message must be displayed. For example:

```
1  $ java DFACheck
2  DFACheck: no input files specified
3
4  $ java DFACheck foo.txt
5  DFACheck: invalid usage - the program must be given two files as input
6
7  $ java DFACheck no-such-file.txt input.txt
8  DFACheck: the file 'no-such-file.txt' could not be opened
9
10 $ java DFACheck dfa.txt no-such-file.txt
11 DFACheck: the file 'no-such-file.txt' could not be opened
```

If valid files are specified as input, then for every string of the second file you should output both the string and the word `accepted` if the given DFA accepts it and `rejected` otherwise, with only one string's results given per line. Using the files `dfa.txt` and `input.txt` given as examples above, your program should behave as follows[2]:

```
1  $ java DFACheck dfa.txt input.txt
2  100100 accepted
3  111 accepted
4  00100100 accepted
5  10101 rejected
6  01010101 rejected
```

[1] Line numbers are included for reference only.
[2] Note that in the example output there is exactly one space between the string and its results. As before, line numbers are included for reference only.

Compilation Notes

Your program must compile using the following command-line invocation:

```
$ javac *.java
```

Do not submit any IDE specific files and in the interest of simplicity do not use packages for this assignment. If your program does not compile using the command mentioned above you will receive no credit.