Due 2/12

Write a program to simulate the execution of a restricted Turing machine. Your program can be written in Java, C, or C++, and needs to be able to be compiled and executed on bingsuns. If you do not know Java, C, or C++, you may submit a version in Python (with my permission).

Your program will read the definition of the machine from a file (the first command line argument), with the second command line argument being the string to simulate the machine running on (the input to the automata). The output of your program will be written to standard output. The output will consist of the list of states the automata transitions through while reading the input string (separated by "->") followed by either "accept" or "reject", depending on wether the automata accepts or rejects the input string.

For this program, the states will be numbered between 0 and 1,000 (not necessarily contiguous or in any particular order). There are three special states that you need to know about – the start state, the accept state, and the reject state. The first three lines of the machine definition will include this information, in the following format:

```
state   7       start
state 12        accept
state   952     reject
```

There is no guarantee on the order of these three lines, and the start and accept states or start and reject states can be the same. The accept and reject states will not be the same. The input file will be tab delimited (should be easily parsed by Java, C, or C++).

The remainder of the file defines the transitions. For this machine, the transition format is "q,a->r,b,x" where q is the current state that the machine is in, the a is the symbol that the machine reads, the r is the state that the machine transitions to, the b is the value that the machine writes on top of the a, and the x is either an L or an R, which tells you to either move back one symbol (L) or to the next symbol (R).

The format of the transitions in the file will be:

```
transition      q       a       r       b       x
```

Since q and r are states, they will be numbers in the range of 0 – 1,000. For this program you can assume that a and b will be digits {0, 1, …, 9}, lower case letters {a, b, …, z} or special characters {$, #, _, %}. And finally x will be in {L, R}.

The input will be a string, consisting of digits, lower case letters, and special characters. Initially the machine will be looking at the left most symbol of the input string. The transitions will tell you what symbol to process next.

If the machine ever transitions to the accept state, then it should immediately stop and output the sequence of states that is transitioned though, followed by a blank space and "accept".

If the machine ever transitions to the reject state, then it should immediately stop and output the sequence of states that is transitioned though, followed by a blank space and "reject".

If the machine hasn't entered the accept state or reject state after 5,000 transitions, then is should stop and output the sequence of states that it transitioned though, followed by a blank space and "quit".

Due 2/12

If the machine ever tries to moved past the left or right end of the input, then is should stop and output the sequence of states that is transitioned though, followed by a blank space and “bounds”.

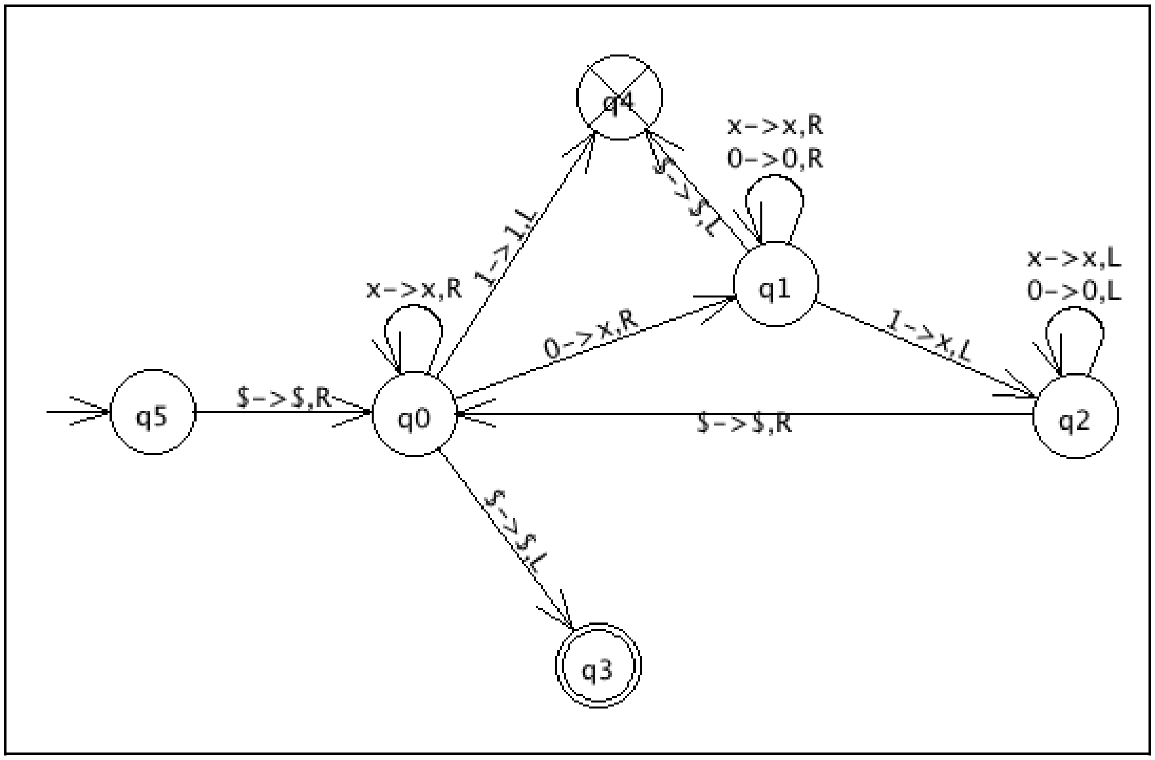
For java, standard input is System.in, standard output is System.out, and standard error is System.err.
For C, standard input is stdin, standard output is stdout, and standard error is stderr.
For C++, standard input is cin, standard output is cout, and standard error is cerr.

E-mail your program (source file(s) and makefile if required attached to the e-mail, I don't want a tar, zip, rar, jar, or any other package file) to me (david.garrison@binghamton.edu) by 11:59:59pm on the date due. Your main filename must be your last name (lower case) followed by “_p1” (for example, my filename would be “garrison_p1.java”) and the subject of your e-mail is “CS 373 program 1”.

The grading will be based on the percentage of correct results your program gets for my collection of test finite automata and test strings and following the directions.

In particular, 10% of the grade will be following the instructions (filename being your last name in lower case, subject, attaching files to e-mail) and 90% will be correct results (check my out put format). If you are using java, I should be able to execute “javac “your last name in lower case_p1”.java” to compile your program. For C or C++ your should include a makefile so that I can simply execute “make “your last name in lower case_p1”” to compile and ./”your last name in lower case_p1” to execute.

Here's is a sample state transition diagram of a restricted Turning machine.



Here q4 is the reject state (has an “x” through it). Since I said the states were numbers, the text file associated with this diagram will have the “q” removed from the state names.

Here is the text file associated with the diagram.

Due 2/12

```
state   3       accept
state   4       reject
state   5       start
transition      0       x       0       x       R
transition      0       1       4       1       L
transition      0       0       1       x       R
transition      1       0       1       0       R
transition      1       x       1       x       R
transition      1       1       2       x       L
transition      2       0       2       0       L
transition      2       x       2       x       L
transition      2       $       0       $       R
transition      0       $       3       $       L
transition      5       $       0       $       R
transition      1       $       4       $       L
```

The language accepted by this machine is { $\$0^n1^n\$ \mid n \geq 0$ }.

For my C++ version of the program, I only have a single file, garrison_p1.cpp. So, I can simply execute “make garrison_p1” to compile and create the executable “garrison_p1”.

Below are three samples runs that ideally are correct.

```
./garrison_p1 prog1_sample.txt '$01$'                 ← command line
5->0->1->2->2->0->0->0->3 accept                      ← output written to standard output
```

The “$” has special meaning in the unix shell(s), so the single quotes around the input string, '01' are so that the unix shell treats the “$” symbol as a normal symbol. We could also enter the string as \$01\$.

Below are examples for accept, reject, and bounds. For quit, an input string of somewhere around 250 '0's followed by 250 '1's will results in a quit.

Accept example:

```
./garrison_p1 prog1_sample.txt '$000111$'
5->0->1->1->1->2->2->2->2->0->0->1->1->1->2->2->2->2->2->0->0->0->1->1->1->2->2->2->2->2->2-
>0->0->0->0->0->0->0->3 accept
```

Reject example:

```
./garrison_p1 prog1_sample.txt '$00011$'
5->0->1->1->1->2->2->2->2->0->0->1->1->1->2->2->2->2->2->0->0->0->1->1->1->4 reject
```

Bounds example:

```
./garrison_p1 prog1_sample.txt '$000111'
5->0->1->1->1->2->2->2->2->0->0->1->1->1->2->2->2->2->2->0->0->0->1->1->1->2->2->2->2->2->2-
>0->0->0->0->0->0->0 bounds
```