Homework 6 Assigned: November 28, 2018

95-771 DSA Due: 11:59:59 PM, Wednesday, Dec. 12, 2018

Standard note: In order to achieve a good score on the Final Exam, be sure to work through the details of this project on your own.

Task 1. Write a Java program that simulates a Turing Machine.

The Turing machine that we will simulate can be formally defined as M = $(Q,\Sigma,\Gamma,\partial,q_0,B)$: where

Q, a finite set of states. For this program Q = {0,1,2,…n-1} and is selected by the client programmer.
Γ = { 0,1,B } is the finite set of allowable *tape symbols*
B, a symbol of Γ, is the *blank*
Σ = { 0,1}, a subset of Γ not including B, is the set of input symbols
∂: Q x Γ $\rightarrow$ Q x Γ x {L, R} (∂ may, however, be undefined for some arguments)
q_0 = 0 is the initial state

Our tape will be bounded on the left but infinite on the right. To simulate this, define an array of size 100 and set the initial read/write head to 0 (the leftmost position on the Turing machine tape). Our simulation will only work for machines that stay within this range. You need not test boundary cases.

Suppose that we wanted our program to simulate the machine with the following values for delta:

∂(q0,0) = (q0,1,R)
∂(q0,1) = (q0,0,R)
∂(q0,B) = (q1,B,R)

This machine reads the tape from left to right and replaces any 1's with 0's and any 0's with 1's. It stops, by entering the halt state, when it encounters a B in the input. We will adopt the convention that an n state machine will always use state n-1 as the halting state. So, in the machine above, we have a two state machine and state 1 will be our halt state.

Your task is to write a Java program (TuringFlipper.java) that simulates this machine. The main routine of your solution will look exactly like the following:

```
public static void main( String args[]) {
    Turing machine1 = new Turing(2);    // A two state machine

    State s0 = new State(0);              // Only s0 has transitions

    s0.addTransition(new Transition('0','1',Transition.RIGHT,0));
    s0.addTransition(new Transition('1','0',Transition.RIGHT,0));
    s0.addTransition(new Transition('B','B',Transition.RIGHT,1));
```

```
            machine1.addState(s0);               // Add the state to the machine

            String inTape = "0101010101010";     // Define some input

            System.out.println(inTape);

            String outTape = machine1.execute(inTape);  // Execute the machine

            System.out.println(outTape);  // Show the machine's output
   }
```

And the output of this program is shown below:

```
C:\McCarthy\www\95-771\TuringMachine>java TuringFlipper
0101010101010
1010101010101BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
```

Submit a zipped directory containing all of the Java code that you used to simulate the machine. For grading, we must be able to edit the main java file TuringFlipper.java so that we can test your program against various input tapes. The main routine will be identical to the main routine above.

Task 2. Write a Java program (TuringSubtractor.java) so that it simulates the following machine:

Proper subtraction m – n is defined to be m – n for m >= n, and zero for m < n. The TM

M = ({q0,q1,...,q6}, {0,1}, {0,1,B}, ∂, q0, B, {})

defined below, if started with 0^m10^n on its tape, halts with 0^{m-n} on its tape. M repeatedly replaces its leading 0 by blank, then searches right for a 1 followed by a 0 and changes the 0 to a 1. Next, M moves left until it encounters a blank and then repeats the cycle. The repetition ends if

i) Searching right for a 0, M encounters a blank. Then, the n 0's in 0^m10^n have all been changed to 1's, and n+1 of the m 0's have been changed to B. M replaces the n+1 1's by a 0 and n B's, leaving m-n 0's on its tape.
ii) Beginning the cycle, M cannot find a 0 to change to a blank, because the first m 0's already have been changed. Then n >= m, so m – n = 0. M replaces all remaing 1's and 0's by B.

The function ∂ is described below.

∂(q0,0) = (q1,B,R) Begin. Replace the leading 0 by B.

∂(q1,0) = (q1,0,R) Search right looking for the first 1.

∂(q1,1) = (q2,1,R)

∂(q2,1) = (q2,1,R) Search right past 1's until encountering a 0. Change that 0 to 1.
∂(q2,0) = (q3,1,L)

∂(q3,0) = (q3,0,L) Move left to a blank. Enter state q0 to repeat the cycle.
∂(q3,1) = (q3,1,L)
∂(q3,B) = (q0,B,R)

If in state q2 a B is encountered before a 0, we have situation i described above. Enter state q4 and move left, changing all 1's to B's until encountering a B. This B is changed back to a 0, state q6 is entered and M halts.

∂(q2,B) = (q4,B,L)
∂(q4,1) = (q4,B,L)
∂(q4,0) = (q4,0,L)
∂(q4,B) = (q6,0,R)

If in state q0 a 1 is encountered instead of a 0, the first block of 0's has been exhausted, as in situation (ii) above. M enters state q5 to erase the rest of the tape, then enters q6 and halts.

∂(q0,1) = (q5,B,R)
∂(q5,0) = (q5,B,R)
∂(q5,1) = (q5,B,R)
∂(q5,B) = (q6,B,R)

Note that This machine is not being used to accept or reject strings and so its set of accepting states is empty. Instead, this machine is being used to perform proper subtraction. The B represents a blank that may appear on the tape.

Submit a zipped directory containing all of the Java code that you used to simulate the machine. For grading, we must be able to edit the main java file TuringSubtractor.java so that we can test your program against various input tapes. The main routine should look almost identical to the main routine above. The only difference should be the size of the machine and the transitions assigned to states.

Task 3. Design a Turing machine that reads a series of zeroes and ones and decides the language L = {0^n1^n, n >= 1}. The sequence of zeroes and ones will be termintated by a blank. Your Turing machine will read the input and decide the string. It will leave the result (a 1 = 'yes' or a 0 = 'no') on the output tape. Here are some example runs:

```
C:\McCarthy\www\95-771\TuringMachine>java TuringDecider
0000011
0BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBB

C:\McCarthy\www\95-771\TuringMachine>java TuringDecider
01
1BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBB

C:\McCarthy\www\95-771\TuringMachine>java TuringDecider
10
0BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBB

C:\McCarthy\www\95-771\TuringMachine>java TuringDecider
0000000001111111111
1BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBB
```

Write a Java program (TuringDecider.java) so that it simulates your new machine. Of course, you are required to adopt the same approach as in Task 1 and Task 2.

Submit a zipped directory containing all of the Java code that you used to simulate the machine. The main routine should look almost identical to the main routines above (except that this program reads the sting from the keyboard). The only other difference should be the size of the machine and the transitions assigned to states. Hint: And, perhaps, an extended tape alphabet.