CS 361: Project 3 (Turing Machine simulator)

1 Project Overview

In this project, you will write code that simulates a bi-infinite TM. You will have to parse the encoding of a TM together with the input string, and then simulate how this instance of a TM processes the input. The focus is on both the correctness and the efficiency of your implementation.

From wikipedia.org

> A **Turing machine** is a hypothetical device that manipulates symbols on a strip of tape according to a table of rules. Despite its simplicity, a Turing machine can be adapted to simulate the logic of any computer algorithm, and is particularly useful in explaining the functions of a CPU inside a computer.
> The "Turing" machine was invented in 1936 by Alan Turing who called it an "a-machine" (automatic machine). The Turing machine is not intended as practical computing technology, but rather as a hypothetical device representing a computing machine. Turing machines help computer scientists understand the limits of mechanical computation.

2 Turing Machine Simulator

2.1 Design Requirements

The objective is to write **an efficient** Turing Machine simulator, i.e., the faster, the better. The simulator should simulate a variant of Turning Machine with a **bi-infinite tape**, i.e., an infinite in both directions tape. You can assume that all Turing Machines are deterministic, and has transitions for each tape symbol, and eventually will halt. The machine only has one halting (accepting) state. There is no reject state. Your simulator stops when it reaches the halting state of the machine. At the end of the simulation, the content of **visited** tape cells should be printed out to the stdout.

We assume that the usual blank symbol, i.e., $\sqcup$ is now the 0 symbol in the tape alphabet. Thus, 0 is not a part of the input string alphabet Σ, but always present in the tape's alphabet Γ, which is in our case $\Gamma = \{0\} \cup \Sigma$. The symbols of the input string alphabet are represented by integer numbers starting at number 1.

States of the TM Q are labeled with integer numbers. We always assume the state with label 0 to be the start state and the halting state is the state with the largest label, i.e., $|Q| - 1$.

Your simulator should accept a single command line argument, which is the name of the input file with the encoding of a Turing machine and the input string.

2.2 Input Specifications

The first line of an input file displays the total number of states in a TM. If there are n states, then state 0 is the start state and state $n-1$ is the accepting halting state (TM encodings won't have "rejecting" state since they would just modify the tape's content). The second line shows the number of symbols in Σ. If there are m symbols then $\Sigma = \{1, 2, \ldots, m\}$. We set m to be at most 9.

The next $|\Gamma| \times |Q| - 1$ lines define a transition function, where the current state (from state) and the input's symbol (symbol under the tape head) are defined by the line number/index of the transition. For example, for a machine with 2 states and 1 symbol in Σ the first transition line describes the tradition from state 0 on symbol 0, i.e., blank. The second transition line describes the transition from state 0 on symbol 1. The third line the transition from state 1 (halting in this case) on symbol 0, and the fourth line the transition from state 1 on symbol 1. Since the halting state has no outgoing transitions it is not in the transition function but only $|Q| - 1$ states

Each transition line has the following comma delimited format:

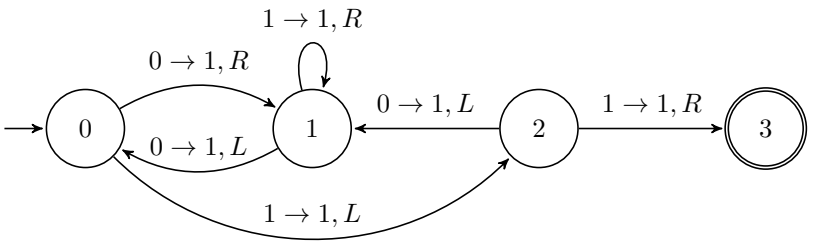
$$next\_state, write\_symbol, move$$

where

- *next_state* – the state that the machine transitions to.
- *write_symbol* – the symbol the machine will write to the cell.
- *move* – the direction, either L or R, to move the tape head to.

The last line of the file contains the input string for the machine, followed by a carriage return. If this line is blank, then there is no empty symbols, i.e, ε, and the tape is initialized to blanks (all 0s).

Consider the following TM:



Its encoding is:

```
[esherman@onyx cs361]$ cat BB3111.txt
4
1
1,1,R
```

```
2,1,L
0,1,L
1,1,R
1,1,L
3,1,R
111
```

Here the first line 4 since the total number of states is 4: $\{0, 1, 2, 3\}$. The second line contains 0, which means $\Sigma = \{1\}$, i.e., all initial inputs on the type are sequences of 1's. The next 6 lines are 6 transitions for three non-halting states $\{0, 1, 2\}$ on symbols $\{0, 1\}$, i.e., the alphabets' 1 and the tape's blank 0. Per definition the halting state 3 cannot have outgoing transitions since a TM stops in that state.

Thus, the first `1,1,R` means that from state 0 (because it the first state) on symbols 0 (because its the first tape's symbol) it goes to sate 1, writes 1 on the tape and moves the head to the right. The second `2,1,L` means that from state 0 (because it the first state) on symbols 1 (because it's the second tape's symbol) it goes to sate 2, writes 1 on the tape and moves the head left.

The next two transitions are for the state 1: for symbol 0 and for symbol 1. The last two transitions are for sate 3 for the same two symbols: first 0, and then 1. The last line is the input string 111, or a blank line for ε input.

2.3 Output Specifications

At the end of the computation, your Turing Machine simulator should print out the content of the visited tape squares followed by a new line. The main class is `TMSimulator.java` in `tm` package. Then

```
[esherman@onyx cs561]$java tm.TMSimulator BB3111.txt
1111111
```

In order to indicate that the machine starts with the empty tape, we leave that last line blank, i.e.,

```
[esherman@onyx cs361]$ cat BB3.txt
4
1
1,1,R
2,1,L
0,1,L
1,1,R
1,1,L
3,1,R

EOF
```

For this file the output is

`[esherman@onyx cs561]$java tm.TMSimulator BB3.txt`

`111111`

We also provide you with two additional files with large simulations, so you can check how well your simulator scales.

file2.txt

output:

03123333333333333333333333333333333333333333333333333333333333333333333333333333333333333
33333333333333333333333333333333333333333333333333333

output length: 140

sum of symbols: 414

file5.txt

output: very large

output length: 20430

sum of symbols: 47189

2.4 Program Specifications

The code should be implemented in Java and run on onyx. You are welcome to use Java collections, but you cannot use any Turing Machine libraries in any language. The main class should be named **TMSimulator.java**. The correctness is evaluated on 7 files with 5 minute timeout for each run. If your program does not terminate within 5 minutes, then it fails that correctness test.

3 Grading Rubrics

1. 5 points – properly commented (Javadocs and inline comments) code and Javadocs generated without errors.
2. 5 points – properly formatted and detailed README.
3. 5 points – program submitted correctly, compiles and runs on `onyx`.
4. 10 points – code quality, i.e., easy to read, proper data structures used and proper variable naming.
5. 5 points – OO principles, i.e., TMState class, TM class and so on
6. 70 points for the program running correctly (within 5 minutes timeout): 10 points for each file.
7. **Extra 15, 10 and 5 points**: your implementation is among 1st, 2nd and 3rd, respectively fastest on all 7 test cases among your classmates.

4 Submitting Project 3

If you haven't done it already, add a **Javadoc comment** to your program. It should be located immediately before the class header and before each method that was not inherited

- Have a class javadoc comment before the class.
- Your class comment must include the *@author* tag at the end of the comment. This will list you as the author of your software when you create your documentation.
- Use *@param* and *@return* tags. Use inline comments to describe how you've implemented methods and to describe all your instance variables.

Include a plain-text file called **README** that describes the program and how to compile and run it. Remember to include also in the README who are the authors of the code. Expected formatting and content are described in README_TEMPLATE. An example is available in README_EXAMPLE.

You will follow the same process for submitting each project.

1. Open a console and navigate to the project directory containing your source files,
2. Remove all the `.class` files using the command:

   ```
   rm *.class
   ```
3. In the same directory, execute the submit command :

   Section 1:
   ```
   submit cs361 cs361 p3_1
   ```
   Section 2:
   ```
   submit cs361 cs361 p3_2
   ```

4. Look for the success message and timestamp. If you don't see a success message and timestamp, make sure the submit command you used is EXACTLY as shown.

Required Files:

- `TMSimulator.java` and other source files for the Turing Machine simulator in `tm` package.
- A README file describes all submitted files and how to compile them, and your approach to the implementation.