Introduction

The problem in this assignment is to create a software that simulates a Turing Machine (TM) that works for any situation. The machine is expected to get an input string and tell if the string is ***accepted***, ***rejected***, or ***looped*** given the transition rules. An example of the input file is given below:

```
1 (number of variables in input alphabet)
0 (the input alphabet)
2 (number of variables in tape alphabet)
0 X (the tape alphabet)
b (blank symbol, it is always a member of tape alphabet, and it does not count in the number of variables in tape alphabet)
7 (number of states)
q1 q2 q3 q4 q5 qA qR (states)
q1 (start state)
qA (accept state)
qR (reject state)
q1 0 b R q2
q1 b b R qR
q1 X X R qR
q2 0 X R q3
q2 X X R q2
q2 b b R qA
q3 X X R q3
q3 0 0 R q4
q3 b b L q5
q4 X X R q4
q4 0 X R q3
q4 b b R qR
q5 0 0 L q5
q5 X X L q5
q5 b b R q2
000 (string to be detected)
```

The input text file includes information about the machine such as the state number, tape alphabet, etc. and most importantly, the transitions themselves. At the last line, it gives the input string to be detected.

Method

A Turing Machine (TM) consists of a tape on which read and write operations can be performed. The tape can be consist of infinite cells on which each cell either contains input symbol or a special symbol called blank. It also includes a head pointer which points to cell currently being read and it can move in both directions (left/right).

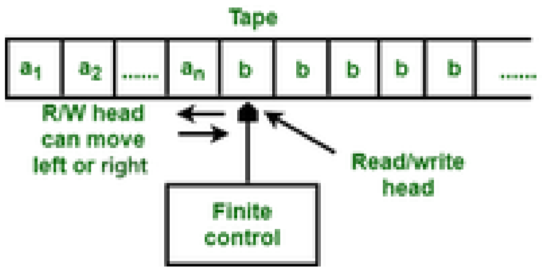

Figure 1: Turing Machine tape structure

A Turing Machine is expressed as a 7-tuple (Q, T, B, ∑, δ, q0, F) where:

- Q is a finite set of states
- T is the tape alphabet (symbols which can be written on Tape)
- B is blank symbol (every cell is filled with B except input alphabet initially)
- ∑ is the input alphabet (symbols which are part of input alphabet)
- δ is a transition function which maps Q × T → Q × T × {L,R}. Depending on its present state and present tape alphabet (pointed by head pointer), it will move to new state, change the tape symbol (may or may not), and move head pointer to either left or right.
- q0 is the initial state
- F is the set of final states. If any state of F is reached, input string is accepted.

The Turing Machine first gets an input string and puts it into the middle of the tape (the tape can have infinite length on both sides, and the cells are filled with the blank symbol by default). After that, it sets its state to the q0, and points the head pointer to the cell of the first letter of the input string. When these steps are completed, the machine gets ready to run. Machine does its operations by following the given transition rules. A transition rule has the structure of a 5-tuple (currentState, read, write, shift, nextState). For instance, the rule (q1, b, X, R, q4) means "When the current state is q1, and the symbol is *b* in the current cell, change the symbol with *X*, point to the cell on the right side, and switch the state to q4". The machine is set to q0 at first, and starting from these configurations, it works until reaching a state of the final states set. After that, the machine stops (halts).

Now, let’s implement this logic to the given example input in the page 1. The Turing Machine’s workflow is as follows:

1. Create a tape with blank symbols in its cells

b	b	b	b	b	b	b	b	b	b	b	b	b	b	b

2. Place the input string in the middle of the tape, and point to the first letter of the string with current state is set to q1 since it is given as the start state

b	b	b	b	b	b	0	0	0	b	b	b	b	b	b

^(q1)

3. Start applying the transition rules until reaching to qA (accept state) or qR (reject state) since they are given as the final states:

    a. q1 0 b R q2

b	b	b	b	b	b	b	0	0	b	b	b	b	b	b

^(q2)

    b. q2 0 X R q3

b	b	b	b	b	b	b	X	0	b	b	b	b	b	b

^(q3)

    c. q3 0 0 R q4

b	b	b	b	b	b	b	X	0	b	b	b	b	b	b

^(q4)

    d. q4 b b R qR

b	b	b	b	b	b	b	X	0	b	b	b	b	b	b

^(qR)

4. If the final state is “qA” (accept state), accept the input string. Otherwise, if the final state is “qR“, reject the input string. In this example, the final state is “qR”. Hence, the input string is rejected by the Turing Machine.

The machine has traversed the states by q1->q2->q3->q4->qR order.

Sometimes, if the machine calculates its next transition same with the previous one. In this case, it can stuck into infinite loop and switch between the same cells and puts the same symbols over and over again. There should also be a mechanism to prevent this.

Implementation

Java is used in the project's implementation part with the object oriented programming (OOP) approach. There there classes in the system;

1. ***Transition:*** The class has 5 attributes that correspond to the 5-tuple structure of the transitions in the input file; currentState, read, write, shift, nextState. Given a transition string to the constructor of the Transition instance, it parses the string and places the elements correctly. It has a single method named "boolean isEqual(Transition t)" that gets another transition instance and compares its values with its own values.
2. ***State:*** The class has only a single ArrayList of the Transition objects. This class' instances are representing the states of the TM (q1, q2, …) and each state object keeps only the transitions that has the represented state as the "currentState" value.
3. ***Machine:*** This class keeps all of the functionalities of the Turing Machine. It has the following methods:
    a. ***buildMachine():*** It uses the scanner and reads the lines of the input file. During the reading process, it puts the values as the machine's attributes. For assigning the *State* and *Transition* objects, it calls the "addState()" method for each state of the machine. To do this, it uses "state count" given in the input file.
    b. ***addState():*** It gets state ID and read the next *m* lines where *m*=tape variable count + 1. The reason of this is that tape alphabet does not contain the blank symbol, and there there are *m* possible symbols a cell can have. Since in the input file, there are lines for each state/symbol combination, this technique has been chosen. The addState() method reads correct number of lines, create *Transition* object for each transition, and at the end, it creates a *State* object with list of the created *Transition* objects.
    c. ***buildTape():*** It creates the tape of the TM. The tape has "$" sign as the first and last letters. The length of the tape depends on the input string because the structure is "$ + 30 blank symbols + input string + 30 blank symbols + $". 30 blank symbol is put on the both sides to have the enough space for shifting the head pointer.
    d. ***startMachine():*** This is the method where the main workflow of the machine happens. It reads the tape's current value, and regarding the current state, it finds the correct transition of the machine. According to the transition rules, it changes the current cell's value, and change the index along with the current state. The transitions continue since one of the final states become the current state. Moreover, there is a loop prevention mechanism. For this mechanism, last 3 transitions are kept in an up to date ArrayList, and whenever a transition repeats itself, it put the currentState's value to "qLoop" and ends the program.

   Another possibility to switch to “qLoop” state is to have the same two states continuously (e.g. q1->q3->q1->q3).

e. ***printResult():*** It reports the result of the machine by printing the traversed states, and whether the input string is accepted, rejected, or looped.

In the main method, a scanner for the input text file, and a *Machine* instance is created. Its methods are called with the following order; buildMachine(), buildTape(), startMachine(), printResult().

Results

The output of the program has the following format:

```
ROUT: q1 q2 q3 q4 qR

RESULT: rejected
```

This is also the output of the program given the example input file in the page 1. The program works only with the input files that has a compatible format with the example input file.

The program has passed all of the tests where different symbols and transition rules are used.

Atahan Çaldır

S018194