Design & Implementation

My Turing Machine simulator acts as an universal TM, because reads a description and checks for its validity, writes starting state name and then goes into a loop by checking current state, checking transition table for that state and updating current state accordingly. For the purpose of clarity, at each state transition I print out a part of the “virtual” tape to display how the TM works. I will explain more about the tape later on in the document. I introduce one minor limitation, to make the Turing Machine more realistic, which is a left most end of the tape, so when the tape head tries to access beyond the last tape cell on the left it is blocked and stays on the same position. However, my tape is still infinite to the right. This makes the TM a bit more realistic because something infinite in both ends is more fictional than something “virtually” infinite in one end. I use the word “virtually” because when the available memory on the PC ends, the tape’s end would come. To make my program more modular I have split it into various classes which I will describe briefly and give out their purpose. I have also included helpful comments in the classes to enhance the readability of the code.

State

My state class, simply represents the notion of a state, with all its necessities. Namely, a boolean whether it is accepting or not, a state name, and a list of transitions.

Transition

The transition class represents a single transition between two states. It consists of two state names for the first and second state, two characters for input read from tape and output written on tape and a move character (R,L or S) for the move direction of the tape head.

Tape

The tape class represents the tape of my TM. It is represented by an array list of characters, because this seemed the most straightforward implementation at first, while I had thoughts of changing it later, it was sufficient for meeting the requirements of the practical. The tape is virtually infinite because it has a particular length as any array, but whenever the tape head needs to access a cell that exceeds the size of the tape, it is extended by 1 cell, so this makes it look as if it is infinite. The class also introduces functions for moving over the tape and writing on it. An index integer is used to keep track of tape position.

TM Description Reader

This class is used to read the TM description while checking for all kinds of errors along the way, to make sure the description strictly matches the protocol specified in the practical. It has some static final integers to avoid magic numbers appearing in the code where possible. A list of states and initial state which are later passed to the Turing machine. In its constructor it calls the check description function, which thoroughly checks for errors in the description file.

Machine

The machine class is used to act as any TM given a correct description file. It consists of a list of states, start state, current state and a tape. The run machine function calls the make step function in a loop until the machine halts. The make step function either accepts if the machine is in an accepting state or tries to find a transition for the current state, given the current input symbol, if it finds one, corresponding output is written in tape cell, tape head advances and current state is changed. Input is rejected if no possible transition is found.

runtm

This class is simply the main method, which prompts the user for input and runs the machine.

Analysis

Time complexity for a TM is the number of transitions made. Space complexity is number of tape cells written to (without rewriting – act like memory). Thus time taken is always at least as big as space taken but can possibly be greater. For making sure my TM descriptions solve the problems that they address correctly I did a lot of testing on different inputs with my "runtm" main method as well as running on examples by hand and going through the TM diagrams in my notebook.
Note: I have two versions of each machine description in .txt files . A working one and a commented one. The commented versions include thorough explanation of states and transitions where not obvious what the machine is doing, thus I am not going to explain my TM descriptions in great detail in the report.

Finding if a binary number contains odd number of ones & Finding out if a binary number is divisible by 3

Brief Description: The first one just goes once over the input string, while changing states depending of the input seen (0,1) in order to determine final number of 1's. The latter again goes only once but its job is a bit more complicated, namely, to keep track of the remainder of a division by 3.The machines recognise any string over the alphabet {0,1}. The language recognised is {(0*1*)*}.

Input size: 7
Transitions made: 8
Space taken: 8

Input size: 14
Transitions made: 15
Space taken: 15

Input size is: 173
Transitions made: 174
Space taken: 174

Brief Analysis: These two machines only need to go through the whole input once, so time and space complexities are f(n + 1), where n is the size of the input, because of the final transition and cell written to at the end of the input string.

Mirror copy Binary string

Brief Description: The copying starts from the rightmost end of the input string, because it is mirrored out in the copy. Each symbol copy/pasted is replaced by l/o, only the initially copied symbol is replaced by L/O, in order to know where the start of the input string is. After the string is copy/pasted in reverse order all the l/o's and the L/O are replaced by corresponding 0/1's. The TM recognises any input string over the alphabet {0,1}. The language recognised is {(0*1*)*}.

Input size is: 7
Transitions made: 114
Space taken: 14

Input size is: 108
Transitions made: 23546
Space taken: 216

Input size is: 201
Transitions made: 81206
Space taken: 402

Brief Analysis: For each input string this machine needs to:
1. Initially, go through the whole input string + 1 for the blank char = n + 1
2. Then for each symbol of the input string, search that character starting from the first blank symbol on the left, going through the already copy/pasted (amended) symbols (n – p, where p is number of not copied symbols) and writing it back on the first blank spot on the left , which gives us 2(n - p) + 1 (1 for the pasting transition).
3. Finally, we have a “finalise” state in which we go through the whole input string and its mirrored copy and a transition to the accepting state, which gives us n * 2 + 1.

To conclude, our time complexity is $3n + \sum_{p=0}^{n} n(2(n-p)+1) + 2 = 3n + 2n^3 - 2n^2p^2 + n + 2$.
Which gives us an upper asymptotic bound for our time complexity of $O(n^3)$.
For the space complexity, it is obvious, as we are making a copy it is always twice the size of the input string : f(n * 2).

Palindromes recognition {0,1,2}

Brief Description: The machine simply starts by replacing a 0,1 or 2 with blank and goes right until it reaches a blank and then tries to remove a corresponding 0,1 or 2 in the left cell from there. If it does not find the corresponding char it rejects, otherwise it replaces this char with blank, the next char with blank and goes right until it reaches a blank and repeats the mirrored explanation from above and loops this until finished. The language recognised is {p1p2, where p1 is (0*1*2*)* and p2 is the same sequence of symbols as p1 but in reversed order}.

Input Accepted
Input size is: 12
Transitions made: 55
Space taken: 13

Input size is: 108
Transitions made: 3079
Space taken: 109

Input size is: 324
Transitions made: 26731
Space taken: 325

Brief Analysis: For each letter for half of the input string this machine needs to go to the first found blank letter to the left or right, move in opposite direction and remove if letters match. This is:

$\sum_{p=0}^{n/2} n-p+1$,where p is number of removed letters and 1 is from reaching the blank and going back one step to the symbol to remove. This gives us $n^2/2 – np/2 + n/2$. So we have an upper asymptotic bound of $O(n^2)$ for our time complexity.
Our space complexity is f(n + 1) because of writing on one blank cell when, initially, starting the machine and going to the end of the input.

Binary Adder and Equality Check

Brief Description: The machine tries to take a digit from both numbers (sum them), while replacing them with blank and compares them with a digit from the third number. If one of the first two numbers has no more digits just the digit from the other is taken. When the third number has no more digits, the other two are checked to see if they have any remaining digits, which if true will result in a reject state. In the case of a carry overflow a part of the tape is rewritten to address it appropriately.

Input size is: 30
Transitions made: 434
Space taken: 31

Input size is: 14
Transitions made: 92
Space taken: 15

Brief Analysis: For summing two digits of the first two numbers, comparing them to a digit from the third and resetting to the start of the tape for new digits to compare, it takes i1 + i2 + 2 transitions to go through the first two numbers and two # , i3 + 1 transitions to compare with the third number's digit, while i3 is the number of digits already compared, where i1(first number length), i2 (second number length) and 1 for the decide transition. For the number of transitions we get this number => $\sum_{i3=0}^{[i3]} 2(i1+i2+3+i3)$. We have a multiply by 2 because we go back through all the input to reset at beginning of tape for new digits. i3 is roughly n/3 – 2 as it is one of the three numbers on the input string and -2 for the two #. So when n grows towards infinity we can replace i3 by n and we get a quadratic upper asymptotic bound of $O(n^2)$ for our time complexity.
Our space complexity is f(n + 1) because of writing on one blank cell when, initially, starting the machine and going to the end of the input.

Testing

For the purpose of testing I have looked at:
1. For the description I have made all sorts of variants of a possibly malicious description, which can lead to corruption of the TM. These variants with errors in the protocol are not accepted by my program due to the thorough description file validation.
2. For the description file I have also looked at normal cases, which should be accepted by my program, namely, the machines that I have devised.
3. For the purpose of input files I have looked at various malicious cases which comprise of symbols, not present in the alphabet of the corresponding TM. As well as symbols from the alphabet but not present in the language recognised by the TM. The tests also include corner cases like including blank spaces at end of input file etc.. Also I have looked at normal cases with inputs that belong to the languages recognised by the Turing machines and they all work as intended.

A couple of corner cases where input is rejected:

For Example, “0#1#1#” for the summation machine (not part of the language):
Enter TM description file name: biAddEqCheck.txt
Enter TM input file name: in.txt
Current state is : s0
_#1#1#
Current state is : s0r
_#1#1#
Current state is : #0r
_#_#1#
Current state is : g1r
_#_#1#
Current state is : comp1r
_#_#_#
Current state is : decide

Input Rejected

input is “05#1#1#” for the summation machine (5 not part of the alphabet):
Enter TM description file name: biAddEqCheck.txt
Enter TM input file name: in.txt
Current state is : s0
_5#1#06
Current state is : s0r

Input Rejected

input is “012210 ”, notice the space in the end :
Enter TM description file name: palindromes{0,1,2}.txt
Enter TM input file name: in.txt
Current state is : s0
_12210
Current state is : s0R
_12210
Current state is : s0R
_12210
Current state is : s0R
_12210
Current state is : s0R
_12210
Current state is : s0R
_12210
Current state is : s0R

Input Rejected

Input symbols that are included in the tape alphabet but not part of the recognised language:
Enter TM description file name: mirrorCopyBiInput.txt
Enter TM input file name: in.txt
Current state is : s-1

LLloO10
Current state is : s0

Input Rejected

An input accepting case for each machine:

I have included test text files in my submission with the display of my Turing Machines when they reach an accepting state with more lengthy inputs.

Evaluation & Conclusion

I think that I have made a reasonable submission for this practical, which successfully completes Tasks 1, 2 and 3 and also provides more than two additional Turing Machines for the second part. I have also thoroughly tested my submission as well as including a small amount of the tests, either in the report or as separate text files. However, in stead of using an array list for my tape I could have used two stacks (left and right) to represent an infinite tape in both sides, but I decided to focus on straightforwardness and stick to my initial idea. I have also provided substantial theoretical and little experimental analysis of time and space complexities of my Turing machines (experiments only used to confirm my theoretical analysis).

References

https://turingmachinesimulator.com/ - I have used the ideas for my 3 additional Turing machines from this link.