Turing Machine Simulator and non-computable functions, aka Busy Beaver machine

Golam Mortuza

CS 561

Fall 2019

Turing Machine:

The "Turing" machine was invented in 1936 by Alan Turing. It's a hypothetical device that manipulates symbols on a strip of tape according to a table of rules. Turing Machine is capable of enumerating some arbitrary subset of valid strings of an alphabet; these strings are parts of a recursively enumerable set. A Turing machine has a tape of infinite length on which it can perform read and write operations.

Busy Beaver:

In computability theory, a busy beaver is a Turing machine that attains the maximum number of steps performed, or maximum number of nonblank symbols produced finally on the tape, among all Turing machine.

Working principle of Turing Machine Simulator:

At the starting state of the simulator the current state of the simulator will be set to 0. Initially tape will contain the tape value that is read from the input file. If the tape is empty there will be a list with one zero only. However, During execution of the simulator. The Simulator will act like this single element list is a bi-infinite list of 0. If the tape needs to go right or left. And there is no symbol on that or the list throws index error. The simulator will add another list element on that direction (left/right). And continue the execution. Tape head index will always start from zero.

At the very beginning of staring the simulator. It will create a dictionary of transition for each individual state and for each individual symbols on that state. For each state and each symbol it will have three values *next_state, write_to, move.* The method propagate machine will start an infinite loop which will only exit if the current state of the simulator is the final state. Initially the current state and tape head index will be passed to this method. The initial state is 0 and initial tape head index is also 0. The current state and tape head index will automatically update based on the input symbol and the transition functions. As the tape is a list, current tape index will indicate the index whose value need to read from the tape.

If the current tape head index is 0. And the simulator wants to go left. It will insert an new list element with the value zero at the begging of the list. And if the tape head index is on the right most cell on the list. And it's transition is right. It will append a new 0 to the tape and proceed. Thus, it will simulate the infiniteness of the tape.

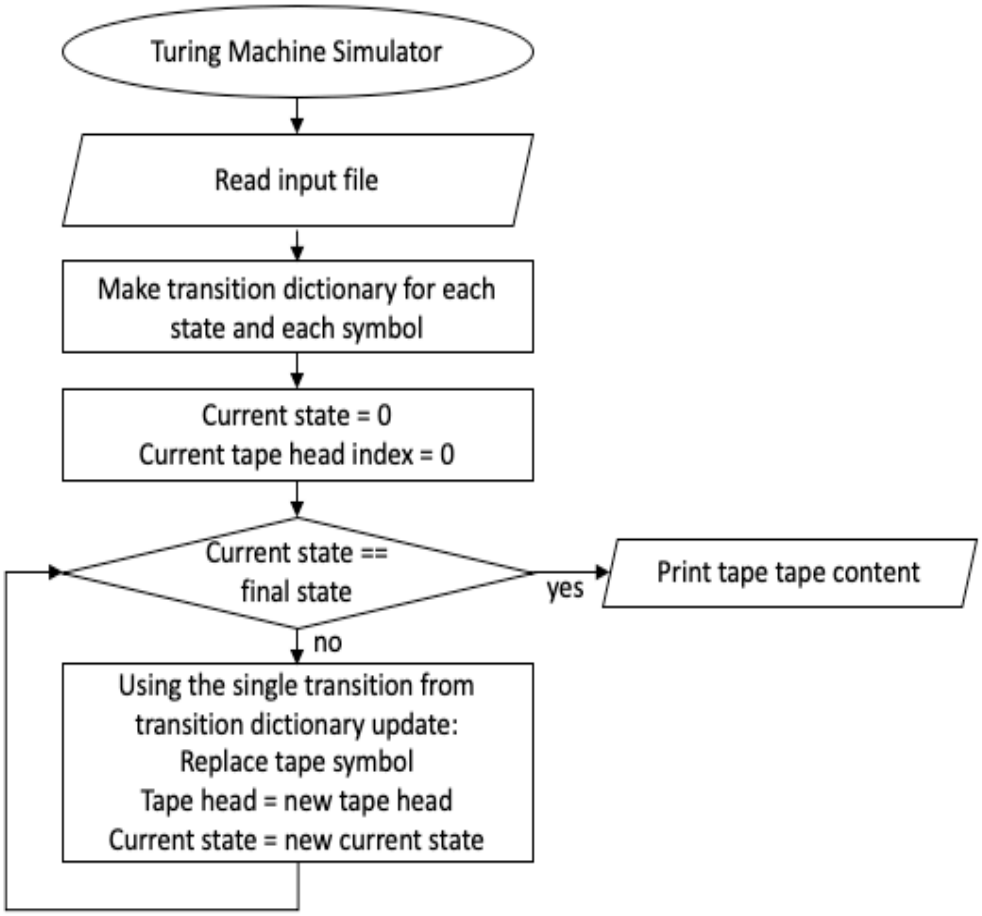

Figure: Flowchart of simulation

Finding Busy Beaver:

Though the solver of a busy beaver problem is a non-computable function. Initially, I didn't realize that this solver might to faster this much. So I wrote a program that perform a linear search to look for best busy beaver. It worked fine for a single state busy beaver. But for multiple state it created so many possible TM that it was impossible to check the solution for each machine. Moreover, some of the machine might go into infinite loop. Or Some of the machine might have few trillion iteration. So it wasn't feasible to check all the Turing machine. So I think it would be more feasible to find the busy beaver by hand using some intuition.

The procedure that I tried to follow to find busy beaver is:

1. The first priority is to find the busy beaver that halt. Doesn't matter how many symbol or which symbol it writes in the tape symbol. It has to halt.
2. Write as much tape symbol as possible in the tape.
3. The value(0,1,2,3) of the tape symbol was the last priority for my busy beaver finding.

Finding Busy Beaver with 1 state and 3 symbols:
According to the formula:

$$(symbols * directions * (states + 1))^{(symbols * states)}$$

This busy beaver has 65536 possible Turing Machine.
As this Turing Machine has only one state. Every time I was trying to write more symbol on the tape. By staying in the same state, it was going in the infinite loop. As the machine is deterministic. For symbol 0 in state 0 if it goes left or right and stay in the same state. It will keep doing it. Cause no matter it goes left or right it will always read symbol 0.
So the best way is to maximize the tape sum is. Whatever we read put the maximum value in the tape that is 3 and go to the final state.

Finding Busy Beaver with 2 state and 3 symbols:
The busy beaver has $1.1x10^{11}$ possible Turing Machine. Checking each of the Turing machine is almost impossible. So, I had to do it using hand.
The main idea is loop back and forth between state 0 and state 1. In any 2 symbol out four. And stay on the same state on other two symbol. And during staying in the same state time it will try to increase the tape symbol value.
So the way it was designed. It gave the tape symbol "1330". However, I got a better machine by tweak my idea. I was just randomly changing and checking the tape value. And this is best tape symbol I get "13333333333333333333333333333333333333333333333333333333333333333333333333333333333333333
33333310"

Finding Busy Beaver with 3 state and 3 symbols:
This busy beaver has $1.1x10^{18}$ possible Turing machine.
The Idea is the same as finding two state busy beaver. Now instead looping back between state 0 and state 2. It will go from state 0 to state 1 to state 2. Then again come back to state 3. And It will stay on state 0 and will try to increase the tape number staying in the same state. Once it get 0 it will go to the state 1.
The summation of the tape symbol is: 52

Finding Busy Beaver with 4 state and 3 symbols:
This busy beaver has $1.7x10^{19}$.
I was following the same idea as before. But no matter what I was doing It was going in a infinite loop. Then I start debugging the tape. I monitor the tape in each transition using the debugger tools of PyCharm. I realized the reason of it's for going into the infinite loop. Then I started modifiying the transition rules accordingly. By checking if it's goes into a loop or not.
However, It would be possible to get the summation of the tape more. But As my first priority was to avoiding loop. I did go forward to get more summation of the tape because of the loop.