

Assignment #5

0. Warm-up exercise: Hand-simulation of Turing machine programs

Simulate the Turing machine programs shown in the puzzle boxes on pages 22-7 in the Turing Machines chapter on the web site. It is important to work these out by hand rather than with the Turing machine simulator so that you get a good sense as to how they work. You do not, however, need to hand these in.

1. Turing machine programming: Remainder by 3

Implement a Turing machine $M_{\%3}$ that computes the remainder of its input when divided by 3. Given, for example, an input tape containing the number 8

...

0	1	1	1	1	1	1	1	1	0
---	---	---	---	---	---	---	---	---	---

 ...

executing $M_{\%3}$ should yield the number 2, because 2 is the remainder of 8 divided by 3.

...

0	1	1	0
---	---	---	---

 ...

Note that your program must leave the tape head at the beginning of the answer.

2. Turing machine programming: Duplicating an input value

Implement machine M_{copy} that copies an input value on the tape, leaving two identical values on the output tape separated by a single 0. Thus, if the input tape is

...

0	1	1	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---

 ...

the final configuration of the tape should look like this:

...

0	1	1	1	0	1	1	1	0	0
---	---	---	---	---	---	---	---	---	---

 ...

For this part of the assignment, it makes sense to use a Turing machine simulator; mine is available as an applet at the following URL:

<http://cs.stanford.edu/~eroberts/applets/TuringMachine/>

3. The Busy Beaver problem

Machines that represent the Busy Beaver solution for 2, 3, and 4 states—along with the best potential solutions for 5 and 6 states—are all over the web, making it impossible to put those problems on a homework assignment (even though they are wonderful puzzles to try to solve on your own). Suppose, however, that we change the problem slightly. Suppose that you are given a Turing machine that recognizes three symbols instead of two. Thus, instead of having columns for just the symbols 0 and 1, programs for this more sophisticated Turing machine have three columns for the symbols 0, 1, and 2. The addition of an extra character in what is called the *alphabet* doesn't change what you can compute with a Turing machine, but it may change how much you can get done with a fixed number of states.

Given this new Turing machine with an alphabet of size 3, you can define a “Bigger Busy Beaver” function as the largest finite number of nonzero symbols you can write on an input tape that starts out with only 0s. If you think about the problem for just a second or two, you can see that $BBB(1)$ must still be 1, just as it is for the two-symbol machine.

The **0** entry in the first state can write either a **1** or a **2**, but after that it has to move. No matter which way it moves, the machine ends up looking at a tape square that also contains a **0**, so the machine would just head off in that direction forever. The best you can do is to write a single nonzero symbol and halt.

Answer the following questions about the Bigger Busy Beaver problem for a three-symbol machine with two states?

- a) How many different three-symbol/two-state Turing machines are there?
- b) Find one of those machines that writes out more than the four **1**s (or, in this case, **1**s and **2**s) that the two-symbol machine was able to write. If you want to try for the maximum here, $BBB(2) = 9$.

4. The Church-Turing thesis

The primary reason that people tend to believe the Church-Turing thesis is that no change to the Turing machine model that anyone has yet proposed changes the set of functions that the machine is able to compute. The extended Turing machine is no powerful than the original. For example, I made the claim in the last question that adding a new symbol to the alphabet didn't change the computational power of the Turing machine, even though it might change the efficiency of that computation. But what does that mean? In your own words, make an argument that any computation you can represent with a three-symbol Turing machine can be transformed into one for a machine that uses only two-symbols. For example, if you are given an input tape that contained a string such as

...

0	1	2	1	0
---	---	---	---	---

 ...

how might you represent that using a presumably longer tape with only two symbols? How would you change a three-symbol Turing machine program into one that would perform the same operations on a two-symbol machine, subject to the strategy you choose for representing the tape value with one less symbol?

If this problem seems too hard after the first lecture on Turing machines, wait until after Thursday's class, when I'll talk about how one proves the equivalence of computational models. Also, instead of trying to prove that having three symbols doesn't help, you might find it even easier to prove that having *four* symbols doesn't help.