CS 30 Lab 8 — UTMs and Busy Beavers

In this lab we will improve our Python Turing machine simulator and use it to investigate certain types of machines called Busy Beavers.

---

1. In your home directory, execute the following Linux commands (type them exactly as shown). The first command copies the entire contents of the directory /common/cs/cs30/lab8 to your home directory. The second command moves you to that directory, where you will find the UTM.py program we developed in class this week, along with a number of Turing machine description files.

   ```
   cp -r /common/cs/cs30/lab8 .
   cd lab8
   ls
   ```

2. Open the UTM program in IDLE and test it out on the machines **inverter.tm**, **eraser.tm**, and **looper.tm**. For example, you can run the Inverter TM on the input string "000111" as follows (we'll add some underscore symbols representing blanks after the string to give the machine room to work):

   ```
   >>> inverter = loadTM('inverter.tm')
   >>> utm(inverter, '000111____')
   ```

3. [From *The New Turing Omnibus*, by A. K. Dewdney] Turing machines, runing back and forth along their tapes, reading a symbol here and writing a symbol there, are a little like beavers who busily ply the waters between forest and dam, carrying sticks and branches to and fro. Just how busy can a Turing machine be? Some Turing machines are infinitely busy, in the sense that they never halt. Moreover, many of those that do halt may be made busy for arbitrarily long periods by increasing the size of their inputs. Thus it seems sensible to frame this question in the context of an initially empty tape, for all machines that halt with such a tape as input.

   The *Busy Beaver problem* asks what is the maximum number of **1**'s that any *n*-state Turing machine can print on an initially blank tape before halting? You might think that Turing machines with only a few states aren't complicated enough to print out very many **1**'s before halting. Let's investigate this problem by simulating busy beaver machines with our UTM program...

4. Try running your UTM program on the **copier.tm** machine with the initial tape contents `'11000q'`, containing no blank symbols, and observe what happens. As soon as the machine runs out of tape, the program crashes because the `headPosition` index is no longer within a valid range. However, we can work around this problem for **copier** by just including extra blanks on the initial tape, so that there is enough room to hold a copy of the input string. Try running **copier** on `'11000q_____'` as an example.

5. To simulate busy beavers, we need to fix this problem, because we don't know in advance how much tape a machine will end up using before it halts. We need a way of automatically adding more blanks to a tape if the machine tries to move off the end of it. Write a function called `expandRight(tape)` that takes a tape (that is, a list of symbols) as input and returns a new tape twice as long containing the symbols of the original tape followed by blank symbols (underscore characters). For example:

   `expandRight(['a', 'b', 'c'])` should return the list `['a', 'b', 'c', '_', '_', '_']`

   Next, add the lines below to the UTM in the appropriate place. This checks to see if the tape head is about to move off the right end of the tape, and if so replaces the tape with an expanded version.

   ```
   if headPosition == len(tape) - 1:
      tape = expandRight(tape)
   ```

   Test the UTM to make sure it works by running it on **copier.tm** with the input tape `'11000q'` as before. This time, you should see the tape expand when the read/write head moves past the `q` symbol.

6. We need to do this for the left side of the tape as well. Write a function called `expandLeft(tape)` that works like `expandRight`, except that it adds blanks to the tape on the *left* side instead of the right side:

   `expandLeft(['a', 'b', 'c'])` should return the list `['_', '_', '_', 'a', 'b', 'c']`

7. Modify the UTM so that attempting to move off the left end of the tape automatically doubles the size of the tape first. This is slightly trickier than the right-side case, because you can't compute the new head position by just subtracting one (which would give a negative value). Instead, the head should point to the *middle* of the newly expanded tape.

   Test your code with the machine called **busy3.tm**, which is a 3-state busy beaver. When started on an initial tape of five blanks `'_____'`, it moves one cell to the right, then one cell to the left, then one more cell to the left. At this point, you should see the tape expand to the left. The machine should halt after another 10 steps. Make sure your UTM program works correctly on this machine before going on.

   You should also try out **christmasTree.tm** and **zigzag.tm** at this point. These machines go into infinite loops, and thus never halt, although they do produce interesting patterns as their tape head weaves its way back and forth across the tape. To interrupt execution, just type Control-C.

8. How many **1**'s does the **busy3** machine leave behind on its tape after halting? As it turns out, **busy3** prints out the maximum number of **1**'s possible for a three-state machine. The corresponding values for a few other machine sizes are given below:

# of states	max possible # of **1**'s printable
1	1
2	4
3	?
4	13
5	?

9. From the table, it seems like small numbers of states only allow a few **1**'s to be printed out. However, machines with only *five* states can exhibit much more complicated behavior. Try out the machine **busy5a.tm**, which is a five-state busy beaver. This machine runs for a very long time before halting! To find out just how long, add a counter to the UTM to count the number of steps and have it print out this value upon halting. Also, it will help to temporarily comment out the `printTape(tape, headPosition)` line to speed up execution. How many steps does **busy5a** take?

10. Next, let's find out how many **1**'s are left behind on the tape. There are too many to count by hand, so write a function `countOnes(tape)` to do the work for you and have the UTM report the final number of **1**'s on the tape when it halts. For example, `countOnes` should work as shown below:

    `countOnes(['0', '1', '1', '0', '1', '0', '0', '_', '_'])` should return 3

    How many **1**'s does **busy5a** leave behind? (Compare this to the other values in the table above.)

11. The machine **busy5b.tm** is also a five-state busy beaver. This one takes more than *twice* as many steps as **busy5a** to halt, although it doesn't leave behind quite as many **1**'s. If you have the patience, find out how many steps it does take, and how many **1**'s it leaves behind.

12. Incidentally, consider the mathematical functions *ones(n)* and *steps(n)*, which tell you the maximum possible number of **1**'s printable by any Turing machine with *n* states, and the corresponding number of steps required. The table from part 8 above showed the first few values of *ones(n)*. These are examples of *provably uncomputable* functions, in the sense that no computer program could ever be written, even in principle, to correctly compute the values of *ones(n)* and *steps(n)* for all possible *n*.