Faculty of Arts and Sciences
Department of Computer Science
CMPS 200 – Introduction to Programming
Assignment 11 – Due Nov 23, 2018

Announcements:

- This assignment is exceptionally due this week on Friday Nov 23 at noon.
- A reminder that the third exam will take place on Friday Nov 23, 5:00 pm–6:30 pm. Students who have pre-approval for additional time should be at the lobby of Bliss Hall by 4:15pm.
- You may use your own laptop during the exam, but please make sure it has enough battery power to last you for the whole exam period (up to 90 minutes).
- The exam will be in Nicely 500 for those using their own laptops, and in Bliss 133/134 for those using the lab computers.
- All wifi and bluetooth connections must be completely turned off during the exam.
- The exam is closed-book, but you may bring a one-sided A4 sized crib sheet.

Reading Material:

- Section 13.1 from textbook
- https://tinyurl.com/y8r5sqsy. This is section 4.11 from an online textbook "Problem Solving with Algorithms and Data Structures using Python". It covers maze traversal material.

1. Fibonacci numbers.
The n-th Fibonacci number is defined for positive numbers n through the following recursion:

$$\text{fib}(n) = \begin{cases} 1, & n = 0, 1 \\ \text{fib}(n-1) + \text{fib}(n-2), & n > 1 \end{cases}$$

- Write a recursive function `fib(n)` that directly encodes the definition above, i.e., by calling itself twice at every recursive invocation.
- The version above is hopelessly inefficient because it needlessly repeats the same computations over and over again. One way to improve its efficiency is to record the value returned by the first call and then look it up rather than compute it each time it is needed. A dictionary may be used to record these values as described in section 13.1 of your textbook.
  Write a function `fastfib(n)` that computes the n-th Fibonacci using this technique.
- What is the computational complexity of `fastfib`? How much storage is needed for it?

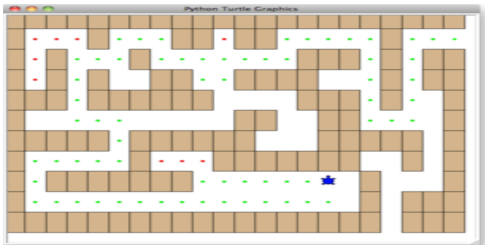
2. Weighted Line Segment.
In this program, you will specialize the line segment class you built in the previous homework to handle *weighted* segments. A weighted segment is one that defined, not only by the pair of coordinates of its left and right boundaries x_0 and x_1 as before, but also by two end weights: one at its left boundary (w_0) and one at its right boundary (w_1). The centroid of a weighted line segment if defined as $(w_0x_0 + w_1x_1)/(w_0 + w_1)$. Note that the centroid become simply the midpoint if $w_0 = w_1$.

Implement a class `Wsegment` that inherits from the base class `Segment` and includes the methods:

- `__init__()`: constructor that takes two or four arguments to build the weighted segment. If only two values are given, they are assumed to be the coordinate and weight of the right end; the left coordinate end defaults to zero and its weight defaults to 1. The constructor should raise an exception if the weights are not positive numbers.
- `__str__()`: returns an appropriate string representation of the segment
- `centroid()`: returns coordinate of the centroid of the segment
- `__lt__()`: a "less than" method (<) that compares the centroids of the segments. This method overrides the "<" method of the base class.

You should include a function that tests the methods of the `Wsegment` class and make sure it inherits properly from the base class. You will need to think about and write an appropriate set of tests.

3. Maze Traversal.



In this program, you will write a program that solves maze traversal problems such as the one shown above. We will assume that our maze is divided up into "squares". Each square of the maze is either open or occupied by a section of wall, and we can only pass through the open squares of the maze. Starting from an initial position on an open square, we want to write a program that can find a path out of the maze.

- Following the outline at the URL in the reading material, implement a class `Maze`. The class should have an appropriate constructor, printing method, setter methods for updating the internal representation of the maze (list of lists), and utilities for checking whether a position is an exit, or a wall, etc.
  Note. Your printing method need not be a fancy window-based turtle graphic. It could simply be a set of characters in the terminal as below, where every line is a row and every character represents a square in the maze.

```
++++++++++++++++++++++
+   +   ++ ++     +
+ +   +       +++ + ++
+ + +  ++  ++++   + ++
+++ ++++++    +++ +  +
+          ++  ++    +
+++++ ++++++   +++++ +
+     +   +++++++  + +
+ +++++++      S +   +
+                + +++
++++++++++++++++++ +++
```

- Write a recursive function that takes in a maze object, a starting row, and starting column and returns true if a path out of the maze is found, and false otherwise. The maze object argument will be mutated to indicate the squares that lie on the path if found.
  `search(maze, row, col)`
  The algorithm should try going up one square and then recursively try the procedure from there. If not successful by starting up, it should go down one square and then try the procedure recursively. If down does not work, it should try going to the right first and then recursively applying the procedure. If right does not work either, it should try going left and then recursively applying the procedure. If none of these directions works then there is no way to get out of the maze and the algorithm returns False.
- Modify the method that displays the maze so it shows the path found out of the maze. Again, you only need to display the path using characters on the terminal. Indicate the squares of the path using the character `o`.
- Does the algorithm you wrote return the shortest path out of a maze? If not, how can you characterize the path that is found?

Zip your files in a single archive file `asst11_`***`netid`*** where ***`netid`*** is your AUBnet user name. Your submission to Moodle must be received by noon of the due date. Late submissions will get no grade.