MATH 8160, Spring 2022
Network Algorithms and Data Structures
Project 1: Linked List Module

Due: 22 February 2022

From Goodrich, Tamassia, and Goldwasser, *Data Structures and Algorithms in Python*, Wiley, 2013.

> One way to construct a maze starts with an $n \times n$ grid such that each grid cell is bounded by four unit-length walls. We then remove two boundary unit-length walls, to represent the start and finish. For each remaining unit-length wall not on the boundary, we assign a random value and create a graph G, called the dual, such that each grid cell is a vertex in G and there is an edge joining the vertices for two cells if and only if the cells share a common wall. The weight of each edge is the weight of the corresponding wall. We construct the maze by finding a minimum spanning tree T for G and removing all the walls corresponding to edges in T . Write a program that uses this algorithm to generate mazes and then solves them. Minimally, your program should draw the maze and, ideally, it should visualize the solution as well.

You'll divide into three teams of two programmers each, and each team will implement a different algorithmic strategy for the spanning tree part of the assignment. There is a technique from the "agile" software development methodology called "pair programming," where two programmers sit together at a workstation (or in a Zoom session) and develop code as a team. From the Wikipedia entry:

> Pair programming is an agile software development technique in which two programmers work together at one workstation. One, the driver, writes code while the other, the observer or navigator,[1] reviews each line of code as it is typed in. The two programmers switch roles frequently.
>
> While reviewing, the observer also considers the "strategic" direction of the work, coming up with ideas for improvements and likely future problems to address. This is intended to free the driver to focus all of their attention on the "tactical" aspects of completing the current task, using the observer as a safety net and guide.

Feel free to try it out or develop your own work style, but do make sure you share the work.

Specializing

Team Prim-Jarník

Implement the Prim-Jarník algorithm for your spanning trees. Implement an adjustable priority queue for tracking candidate edges for extending your tree. Your priority queue should implement the ADT for adjustable priority queue as described in the slides. The internal data structure should be a binary heap.

Team Kruskal

Implement Kruskal's algorithm for your spanning trees. Implement a union-find data structure for merging connected components. You should implement your data structure as a tree with path compression.

Team Barůvka-Sollin

Implement the Barůvka-Sollin algorithm for your spanning trees.

Common Parts

Feel free to collaborate across teams on the other pieces of the project, including graph construction, maze solving (a graph search on the tree) and output of the maze and the solution.

The input should be a parameter on the command line indicating the number of rows and columns and an option for producing a printout of the maze or the maze with the solution path marked with asterisks in the cells on the solution path. The output should appear on stdout (the terminal) using underscores and vertical bars to form cells:

```
  _________
| |_|_| |_|
|__  __  _|
|_|__ |_|_|
|_|_|____ |
```

Notes

- Follow the style guidelines at the end of Chapter 4 of the Python tutorial at `https://docs.python.org/2.7/tutorial/controlflow.html#intermezzo-coding-style`
- The idea of an abstract data structure is that the user should be unaware of the internal representation of his data. It should be possible to substitute any team's MST code in the rest of the program with no other changes and have it function correctly.
- In case of an error conditions, either from the specification or because an operation makes no sense in some context, you should throw an exception.
- You should test your module thoroughly. It is good practice to code your tests before you code your module. (See the Python **pass** statement for a placeholder while developing tests before you code.) This strategy is referred to as *test-driven development* or TDD. There are various tools available online to help structure the TDD process. See, for example, `https://docs.python.org/2/library/unittest.html`. You are not required to use PyUnit or any other testing framework,

but you should consider adopting the idea of TDD in your development process.