EECS 281 – Fall 2020
Lab 9 Assignment (13 points)

Due Friday, December 4, 2020 at 11:59 PM

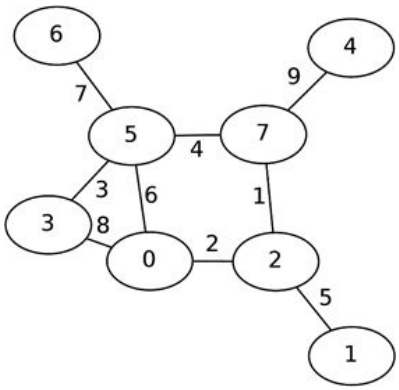
Note: This lab assignment contains a multiple choice quiz and a coding portion. Submit your quiz answers to the lab 9 Canvas quiz and your code to the autograder. You will have **three** attempts on the quiz.

You MUST include the following assignment identifier at the top of every file you submit to the autograder as a comment. This includes all source files, header files, and your Makefile (if there is one). If there is no autograder assignment, you may ignore this.

Assignment Identifier: 472D3C8289DE4915774A47683EC45FFBA373B980

Part A: Graphs and Searching Algorithms (3 points)

For questions 1-2, use the graph below:



9.1 - Running Prim's Algorithm (0.5 points)

What is the sequence of vertices added to the MST for the graph above when running Prim's algorithm, if you start with vertex 7?

- **A.** 7, 2, 0, 5, 1, 3, 4, 6
- **B.** 7, 2, 0, 5, 3, 1, 6, 4
- **C.** 7, 2, 0, 3, 5, 1, 6, 4
- **D.** 7, 2, 5, 4, 0, 1, 3, 6
- **E.** 7, 2, 5, 4, 0, 3, 1, 6

9.2 - Running Kruskal's Algorithm (0.5 points)

What is the sequence of edges added to the MST for the graph above when running Kruskal's algorithm? Use the edges' weights as their label for this question.

- **A.** 1, 4, 9, 2, 3, 5, 7
- **B.** 1, 2, 4, 5, 3, 9, 7
- **C.** 1, 2, 4, 3, 5, 7, 9
- **D.** 1, 2, 3, 4, 5, 7, 9
- **E.** 1, 2, 3, 4, 5, 6, 7, 8, 9

9.3 - Finding Your Flight (0.5 points)

You currently have a graph that represents several airports (vertices) and the flights that connect these airports (edges), weighted by the distance of each flight. You decide to implement this graph using an adjacency list. Let V represent the number of vertices in this graph, and let E represent the number of edges. If you are given an airport X, what is the worst-case time complexity of determining if *any* flights depart from airport X?

- **A.** $\Theta(1)$
- **B.** $\Theta(E)$
- **C.** $\Theta(V)$
- **D.** $\Theta(1 + E/V)$
- **E.** $\Theta(V^2)$

9.4 - Stacks and Queues (0.75 points)

Which of the following statements is/are **true**? *Select all that apply.*

- **A.** a stack is a good data structure to use for a depth-first search (DFS)
- **B.** a stack is a good data structure to use for a breadth-first search (BFS)
- **C.** a stack is a good data structure for conducting a level-order traversal of a binary tree
- **D.** a queue is a good data structure to use for a depth-first search (DFS)
- **E.** a queue is a good data structure to use for a breadth-first search (BFS)
- **F.** a queue is a good data structure for conducting a level-order traversal of a binary tree

9.5 - Memory Trees (0.75 points)

Suppose you have a binary tree of height 281, where leaf nodes have height 1, and you want to search for an element *k*. You know that *k* exists as a distinct element in this tree, and that it is a leaf node. Which of the following statements is/are **true**? Select all that apply.

- **A.** if you conduct a depth-first search for *k*, you'll never have to store more than 281 nodes in memory
- **B.** if you conduct a breadth-first search for *k*, you'll never have to store more than 281 nodes in memory
- **C.** if you are concerned about the memory efficiency of the search, you should use a queue to conduct this search instead of a stack
- **D.** the path from the root to the element *k* returned by a breadth-first search will always be shorter than the path returned by a depth-first search
- **E.** while a breadth-first search will always find the element *k*, it is possible that a depth-first search may fail to find *k* since depth-first searches do not always visit every element

---

Part B: Coding Assignment (10 points)

9.6 - Number of Islands (10 points)

In this problem, you will be implementing the `number_of_islands()` function, as shown below:

```
int number_of_islands(std::vector<std::vector<char>>& grid);
```

This function takes in a 2-D grid map filled with land (represented using the character `'o'`) and water (represented using the character `'.'`) and **returns the number of islands that exist in the map.** An island is formed by connecting adjacent land characters either horizontally or vertically. In other words, if two land characters are adjacent to each other horizontally or vertically, then they are a part of the same island.

Example: Given the following 2-D map:

```
o...oo.ooo.ooooo.
.ooo.oo..oo..oooo
...o.o.ooo.....oo
oo.ooo..oooo..o..
o....o....ooooo.o
ooo....oooo....oo
....ooooo...ooo.o
ooo.oo.oooo.o.ooo
.ooo....oo......o
...oooooo...oo.oo
```

the `number_of_islands()` function should return 7, since there are 7 islands in this map:

```
o...oo.ooo.ooooo.
.ooo.oo..oo..oooo
...o.o.ooo.....oo
oo.ooo..oooo..o..
o....o....ooooo.o
ooo....oooo....oo
....ooooo...ooo.o
ooo.oo.oooo.o.ooo
.ooo....oo......o
...oooooo...oo.oo
```

Implement your solution in the `islands.h` starter file provided. You can download the file on either Canvas or GitLab. To submit to the autograder, create a `.tar.gz` file containing just `islands.h` by running the following command:

`tar -czvf lab9.tar.gz islands.h` *(you can also run "make fullsubmit" using the Makefile we provide)*

If you are working with a partner, ***both partners must submit to the autograder***. Only students who submit code to the autograder will receive points. It's fine if both of you submit the same code for this assignment. If you do end up working with a partner, make sure to put both of your names at the top of the file you are submitting.

Hints *(try to work the problem on your own before you look at this)*

The purpose of this exercise is to give you practice with graph searching algorithms (BFS and DFS) before the final exam. There are two primary methods that you can use to solve this problem: either by using a *breadth-first search* (BFS) or a *depth-first search* (DFS). A summary of the algorithm is as follows:

1. Initialize a counter to keep track of the number of islands.
2. Iterate over every cell of the 2-D vector (a double `for` loop is okay).
3. If the cell you are currently visiting is a land character (`'o'`):
    - Increment your counter by one.
    - Mark the cell as visited. (You do not need to create a separate container to keep track of whether a cell has been visited or not! Hint: you can directly modify the values in the map.)
    - Complete a BFS or DFS of the entire island you are on. To do this, push any adjacent land locations into a queue or stack (make sure that you aren't accessing out of bounds). In other words, you would push into a queue or stack the coordinates directly to your north, east, south, and west, as long as they are also land locations. Mark any coordinates as visited, if necessary. If you are doing a DFS, you can either explicitly initialize a `std::stack` or use recursion, which utilizes the program stack.
    - After the BFS or DFS is complete, continue iterating the 2-D vector. Keep on repeating steps 2 and 3 until the entire 2-D vector is processed.

You do not need to solve this problem using both BFS and DFS, but you are encouraged to practice both algorithms for the exam (since you may get a problem that can only be solved using one method and not the other).