Homework 8

Due Friday, July 23 by 11:59pm

Fun Sea Creature Fact: Jellyfish have survived 5 major extinctions and are older than dinosaurs!

Installing and Handing In

Accept the GitHub assignment **here**. Homework is handed in through Gradescope. For written portion, submit a PDF and match the questions accordingly. For the coding portion, submit ALL of the .py files.

1 Written Problems

Problem 8.1

Fast Graduation

Purpose: Understanding and writing Topsort
Resources: Topsort Lecture (July 13)

Suppose Katie's curriculum consists of n courses, all mandatory, all of them lasting one semester, and all of them offered every semester (i.e. the limit does not exist). The prerequisite structure of the curriculum can be organized in a graph, where each node is a course, and there's a directed edge from node A to node B if and only if A is a prerequisite for B. (Note that if A is a prereq for B and B for C, then that implicitly makes A a prereq for C. The arrow from A to C may or may not be included in the graph.) A course may have any number of prerequisites.

Write pseudocode for an algorithm that computes the minimum number of semesters needed for Katie to complete the curriculum. (She may take any number of courses in each semester). The run time of your algorithm should be $O(|V| + |E|)$.

2 Python Problems

Problem 8.2

Multiple shortest paths

Purpose: Practice working with graphs and traversing graphs (will be helpful for graph project!)
Resources: Intro to Graphs Lecture (June 22) and Shortest Paths Lecture (June 24)

Let $G = (V, E)$ be an undirected graph with *unit* edge lengths (i.e., the length of each edge is 1). Note that there might be multiple shortest paths between a pair of nodes in G. Write python code for a linear-time algorithm that finds the number of distinct shortest paths between nodes u and v. (Hint: $O(VE)$ is not linear, but $O(V + E)$ is) You may assume that there are no parallel edges or self loops in G, and you may assume that there is at least one path between u and v, if u and v are both in the graph (which is not guaranteed). You may also assume that u and v are distinct nodes. Keep in mind that your code need only return the **number** of distinct shortest paths, not the paths themselves. However, in order to find the shortest path(s) you will need to calculate the distance (in this case equal to number of edges) between the given pair of nodes in G.

For this assignment, the only data structure you are allowed to use is python's list. If you plan to use the list as a queue, you'll need to figure out how to pop off only its first element. You are allowed to manipulate and decorate the graph. A stencil for this code, named **numShortestPaths.py**, has been provided for you. When testing, **DO NOT** write your tests within the example test functions we provide! Our scripts will skip the test functions we provide, so write your own functions to test your code thoroughly.

Problem 8.3

Merge, Insertion, and Selection Sort

Purpose: Implementing sorting
Resources: Sorting Lecture (June 17) and Section 4 slides

Implement in Python the following sorting algorithms: merge sort, insertion sort, and selection sort.

Requirements

Your job is to implement the sorts listed above by filling in the methods defined in the stencil code (in `sort.py`), so that, given an array of integers, each will return an array of the same integers sorted in **descending order**. If you sort in ascending order or sort in ascending order and then reverse the list, you will receive a 0. Also please note that you **may not** change the signature of any

stencil method (doing so will result in no credit). You may also, of course, not call Python's built-in **sort** procedure.

- Each sorting algorithm must conform to the following run time requirements:

  Merge sort must run in worst case $O(n \log n)$ time.

  Insertion sort must run in worst case $O(n^2)$ time.

  Selection sort must run in worst case $O(n^2)$ time.

- **Note:** Make sure you throw an `InvalidInputException` if the input list is None. Empty lists are fine, though. You may consider them an already-sorted list.

How To Test Your Code

To test your code, add more assert statements to `sort_test.py`. **DO NOT** write your tests within the example test functions we provide! Our scripts will skip the test functions we provide, so write your own functions to test your code thoroughly. Be sure to test all significant cases, as well as testing that your algorithms handle invalid inputs properly

Sort Profiling

Now that you have your sorting algorithms, why not time them to observe their relative performances? We have provided among the install files two lists of 10,000 numbers (one per line). numlist1.txt has a truly random assortment of integers and numlist2.txt is partially sorted. We have also provided a profiler `sort_profiler.py`. This profiler runs all your sorts on each file and outputs the time it takes to run each.

Run the sort profiler and write a brief readme (in `profiler_readme.txt`) to discuss the differences between your relative sorting times, including the differences in timing for the two text files.