CS 109: Introduction to Computing
Goodney
Fall 2017

Homework Assignment 2

Assigned: 9/18/2017 via Blackboard
Due: 23:59 (i.e. 11:59:00 pm) on 10/2/2017 via Blackboard

Notes:

a. This is the second homework assignment of CS 109. It is worth 7.5% of your overall grade for this class. You must solve all problems correctly to obtain full credit.

b. You are free to discuss the problems on this assignment with your classmates. However, to receive credit, you must write up and submit solutions completely on your own. You are responsible for understanding your answers. The purpose of discussing the questions with your classmates is to deepen your understanding of the material – not simply to obtain answers from them. To get the most out of the class, you are strongly encouraged to make a serious effort to understand and solve the problems on your own before discussing them with others.

c. On the work you turn in, you must list the names of everyone with whom you discussed the assignment.

d. All answers must be typed, not handwritten. The homework must be turned in as a single **PDF** document on Blackboard. Other formats will not be graded.

Problem 1: Graphs [1 point]

The degree of a vertex in a Graph is defined as the number of other vertices to which it is connected.

Consider an undirected Graph with N vertices. Out of these N vertices, you are told that m vertices have degree 3, and the remaining vertices have degree 2. How many edges does this graph have? Your answer should be a function of N and m (in other words, a formula containing N and m). Give a short (1-2 sentence) explanation in support of the formula you found.

Problem 2: Trees [0.5 points]

A Forest is a collection of one or more disconnected Trees. You can convert a Forest into a single Tree by adding certain number of edges between the disconnected Trees (hence connecting them). Given a Forest consisting of k Trees with $n_1, n_2, \ldots, n_k$ vertices, respectively, how many edges must be added to convert the Forest into a single Tree?

Problem 3: Minimum Spanning Trees [1.5 points]

In class you learned two methods for finding the minimum spanning tree (MST) of a graph. You explain both methods to your roommate who has not taken CS 109. Your roommate claims that she's got a third algorithm to find the MST that takes advantage of your algorithms. She claims it's particularly good for large graphs.

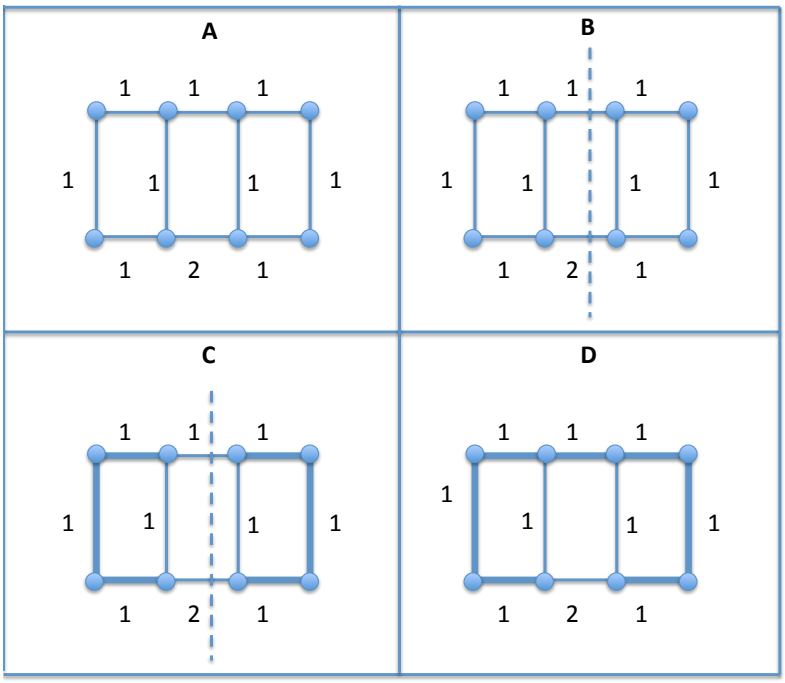
Your roommate's algorithm goes as follows:

1. Partition the original graph into two graphs, and find their individual MSTs (using either of the two methods you learned in class).

2. Among the edges connecting the two partitions, choose the one with the smallest edge weight to connect the two MSTs.

To convince you that her algorithm is correct your roommate shows you the following example.

Figure A: Consider the original graph.
Figure B: Partition it into two graphs as shown.
Figure C: Find the MST of each partition.
Figure D: Look at the edges connecting the two partitions (there are two edges, one with weight 1 and the other with weight 2), and keep only the edge with the least weight. Voila! The resulting tree is an MST of the original graph.



Having taken CS 109, you immediately realize that your roommate's algorithm is not always guaranteed to find the MST of a graph. To show her the error of her ways, you give a counterexample. Write out a counterexample where you pick an original graph, and follow your roommate's algorithm and show that doing so does not lead to the MST of your graph.

Problem 4: Insertion Sort [2 points]

Insertion Sort is an algorithm that adds the first element to a "sorted" sublist and sorts one element at a time by comparing it with previously sorted elements. The algorithm can be described as follows:

- *Proceed one element at a time from left to right (skip the first element)*
- *Take the current element and find its insertion position in the sublist on the left by comparing it to the elements on the left sublist (which is already sorted)*

The algorithm always maintains sorted elements on the left. By the end of execution the whole list is sorted.

For instance, given the list 12, 7, 9, 2, 14:

12	7	9	2	14

The initial array.
The first element is skipped, so we begin with 7.
Compare 7 to 12; 7 < 12, so the two values are swapped.
Number of comparisons: 1

7	**12**	9	2	14

Compare 9 to 12; 9 < 12, so the two values are swapped.
Compare 9 to 7; 7 < 9, so there is no swap.
Number of comparisons: 2

7	**9**	**12**	2	14

Compare 2 to 12; 2 < 12, so the two values are swapped.
Compare 2 to 9; 2 < 9, so the two values are swapped.
Compare 2 to 7; 2 < 7, so the two values are swapped.
Number of comparisons: 3

2	**7**	**9**	**12**	14

Compare 14 to 12; 12 < 14, so there is no swap.
Number of comparisons: 1

2	**7**	**9**	**12**	**14**

This is our sorted array, which took 7 comparisons.

Consider the following list: 13, 9, 6, 3, 15, 2, 21, 4, 16, 8, 11

a. How many comparisons will insertion sort require? Show your work by performing insertion sort on this list and counting the comparisons.

b. How many comparisons will selection sort require? Show your work by performing selection sort on this list and counting the comparisons.

c. If the list were almost ordered (for instance: 2, 3, 4, 8, 6, 9, 11, 13, 16, 15, 21) which algorithm would be the better choice? Give a short (2-3) sentence explanation for your answer.

Problem 5: Complexity [1.5 points]

Consider the three functions below and determine the expression (in Big-O notation) that best represents the simplified asymptotic runtime for each. Compare the three expressions. Which expression grows the fastest? Give a short (2-3) sentence explanation for your answer.

$f(n) = 7n^2 + 3n \log(n) + 5n + 1000$
$g(n) = 7n^4 + 3^n + 1000000$
$h(n) = 7n(n^2 + \log(n))$

Problem 6: Complexity [1 point]

Consider the four lists below. Determine which list is correctly ordered from slowest growth (left) to fastest growth (right). Give a short (2-3) sentence explanation for your answer.

1, log (n), n*log (n), n, n^2, n^3, 2^n, n!
n!, log(n), n*log(n), n, 2^n, 1
1, log (n), n, n*log (n), n^2, n^3, 2^n, n!
1, log (n), n, n*log (n), n^2, n^3, n!, 2^n