Five questions with total 130 points.

For this homework, electronic submission is allowed. Please email your answers to address: cai@cs.uga.edu with subject title: 4470/6470 Homework #4 by Midnight 11:59pm Tuesday April 2, 2019. No late submission will be accepted.

1. (30 points) In the class, we formulated a recursive solution for Huffman Code problem. In particular, we defined the cost function $B(C, f_C)$ to be the minimum cost of a Huffman code (tree) on character set $C$ and frequency function $f_C$ for the characters in $C$. The definition made it possible to obtain a recursive solution:

$$B(C, f_C) = \min_{x,y \in C} \{ f(x) + f(y) + B(C', f_{C'}) \}$$

where $C' = C - \{x,y\} \cup \{[x/y]\}$ by replacing characters $x$ and $y$ with a pseudo-character $[x/y]$ such that frequency function $f_{C'}$ retains the frequency function $f_C$ by excluding frequencies for $x$ and $y$ but including $f_{C'}([x/y]) = f_C(x) + f_C(y)$ for pseudo-character $[x/y]$.

(1) What should the base case(s) for $B(C, f_C)$ be?

(2) How would a computation result for function $B(C, f_C)$ give the information about the corresponding Huffman code (tree)?

(3) We showed that the time complexity $T(n)$ has the recurrence $T(n) = T(n-1) \times cn^2$, which would yield $\Omega((n!)^2)$ lower bound if a direct, top-down recursive algorithm were to be implemented.

Instead, we can carry out a bottom-up, dynamic programming implementation for the function. What would the dimension(s) of the dynamic programming table be? What would the corresponding time complexity upper bound be?
2. **(30 points)** Consider the following **Street lights Problem**: lights need to be installed in a long city street (stretching \( n \) feet long, for some large \( n \)). Lights are only bright enough to shine the area of radius \( r \) feet. The problem is to compute the smallest number of light poles that are needed for this street. Consider foot to be the smallest unit for the length measurement.

   (1) Formulate a recursive solution for this problem. If it helps you, you may refer to the Activity Selection problem where a recursive solution had been formulated before a greedy algorithm was introduced.

   (2) Prove that this problem has a greedy-choice property.

3. **(30 points)** Consider the **Minimum Spanning Tree** problem.

   (1) Formulate a recursive solution for finding a minimum spanning tree from a given undirected graph based on the following divide-and-conquer idea. It first selects some edge to include in the minimum spanning tree and then recursively solves the problem for two subgraphs whose minimum spanning trees will be connected to the two end vertices of the first selected edge, respectively. Make sure that the obtained tree does not contain a cycle. It is okay if the formulated solution may incur exponential time complexity when a top-down, recursive implementation or a bottom-up, iterative implementation were to be carried out.

   (2) Explain that there is a way to make all but one subproblem disappear in your formulated recursive solution, leading to a greedy algorithm.

   (3) Explain which of the greedy algorithms, Prim’s or Kruskal’s, coincides with your greedy strategy.

4. **(20 points)** Give an edge-weighted non-directed graph example that contains at least 5 edges. All edge weights are positive and distinct. Run a minimum spanning tree algorithm (either Prim’s or Kruskal’s) on the graph and show the details about why the produced minimum spanning tree does not always contain all the top 3 smallest weight edges.

5. **(20 points)** Someone designed a recursive algorithm to solve the **Singe-source Shortest Path** Problem. Given an edge-weighted, directed graph \( G = (V, E) \), the algorithm first picks up some directed edge \((s, v)\) that connects the source and vertex \(v\), then it recursively solves the Singe-source Shortest Path problem for graph \( G_1 = (V_1, E_1) \) with source \(s\), and for graph \( G_2 = (V_2, E_2) \) with the source \(v\), respectively. The graphs \(G_1\) and \(G_2\) are modified from \(G\), respectively, as follows.
\[ V_1 = V - \{v\}, \quad E_1 = E - \{(v, x) : x \in V, (v, x) \in E\} - \{(x, v) : x \in V, (x, v) \in E\} \]
\[ V_2 = V - \{s\}, \quad E_2 = E - \{(s, x) : x \in V, (s, x) \in E\} - \{(x, s) : x \in V, (x, s) \in E\} \]

The algorithm post-processes the result from the two recursive calls as follows. For every vertex \( u, u \neq s \) and \( u \neq v \), it compares the length of the path \( s \rightsquigarrow_1 u \) found by the first recursive call with the length of the path \( s \rightarrow v \rightsquigarrow_2 u \) found by the first step and the second recursive call. It only keeps the shorter path. The algorithm optimizes the solution by considering all options for \( v \) in the first step.

Is the algorithm valid (i.e, correct to find shortest paths from the source \( s \) to all other vertices)? Explain in detail why you think it is valid (or not valid). To ease discussion, you may assume all weights are non-negative.

The answers must be word-processed or typed. You may substitute formulae and figures with hand-writings. Your submitted algorithms should be in the pseudo-code, not in any specific programming language. Answers deviating from these requirements will be returned without grading.

The answers must be the student’s own work. Idea sharing and referencing to others’ work (including those online) are not allowed. Plagiarism and other forms of academic dishonesty will be handled within the guidelines of the Student Handbook and reported to the University.