CS41 Lab 4: Topsort and Greedy Graph Algorithms

Many of the lab problems this week center around greedy algorithms for some graph problems, and on data structures for implementing these algorithms. Most of you have seen one or more of these problems before. The purpose of this lab is to (1) practice/develop skills to show that greedy algorithms work optimally, and (2) to explore the connection between algorithms and data structures.

1. Design an efficient algorithm to find the *longest* path in a directed acyclic graph.

2. **Shortest Paths in Weighted Graphs.**

   - a directed graph $G = (V, E)$.
   - a start vertex $s \in V$.
   - each edge $e \in E$ has an *edge length* $\ell_e > 0$.

   Your goal is to output, for each $v \in V$, the length of the shortest $s \rightsquigarrow v$ path.

   If you took CS35, you saw a solution to this problem called Dijkstra's Algorithm. Dijkstra's Algorithm is a greedy algorithm which works by iteratively and greedily building a set of "visited nodes" S. The nodes are selected in increasing path length.

   DIJKSTRA(G, s, ℓ)
   1 $S = \{s\}$.
   2 $d[s] = 0$.
   3 **while** $S \neq V$
   4 pick $v \in V \setminus S$ to minimize $\min_{e=(u,v):u \in S} d[u] + \ell_e$.
   5 add v to S.
   6 $d[v] = d[u] + \ell_e$
   7 Return $d[\ldots]$.

   (a) Show that Dijkstra's algorithm correctly returns the minimum length of paths from s to other nodes.

   (b) What is the running time of Dijkstra's algorithm? For this answer, you may assume any data structure that you've seen from CS35 or CS41, but the running time should be as low as possible.

3. **Minimum Spanning Tree.** A *tree* is an undirected graph that is connected and acyclic. Given a graph $G = (V, E)$, a *spanning tree* for G is a graph $G' = (V, T)$ such that $T \subseteq E$ and G' is a spanning tree. (It is also common to refer to T as the spanning tree).

   Suppose your graph has edge weights: $\{w_e : e \in E\}$. The cost of a spanning tree T equals $\sum_{e \in T} w_e$. A *minimum spanning tree* is a spanning tree of minimal cost.

   In the Minimum Spanning Tree (MST) problem, you are given a connected (undirected) graph $G = (V, E)$ with edge weights $\{w_e : e \in E\}$, and you must compute and output a minimum

spanning tree T for G. MST is another graph problem amenable to greedy algorithms. Two common greedy MST algorithms are:

- Maintain a set of connected nodes S. Each iteration, choose the cheapest edge (u, v) that has one endpoint in S and one endpoint in $V \setminus S$. This is known as *Prim's* algorithm.
- Start with an empty set of edges T. Each iteration, add the cheapest edge from E that would not create a cycle in T. This is known as *Kruskal's* algorithm.

For this problem, take one of the greedy algorithms suggested above, or attempt to create your own greedy algorithm for the minimum spanning tree problem.

(a) Write out your algorithm in pseudocode.

(b) Try to prove that your algorithm correctly returns a MST.

(c) What is the asymptotic running time of your algorithm? If you were to implement it (in, say, C++) what data structures would you need? Would you need any additional data structures beyond structures you've seen from CS35? If so, try to design an implementation for them.

4. You are helping some security analysts monitor a collection of networked computers, tracking the spread of a virus. There are n computers in the system, call them C_1, C_2, ..., C_n. You are given a trace indicating the times at which pairs of computers communicated. A trace consists of m triples (C_i, C_j, k) that indicate that C_i communicated with C_j at time t_k. At that time, the virus could have spread between C_i and C_j.

   We assume that the trace holds the triples in sorted order by time. For simplicity, assume that each pair of computers communicates at most once over the time of the trace. Also, it is possible to have pairs (C_s, C_j, k) and (C_t, C_j, k): this would indicate that computer C_j opened connections to both C_s and C_t at time t_k, allowing the virus to spread any way among the three machines.

   Design and analyze an efficient algorithm that, given as input a collection of time-sorted trace data and a virus query, answers the question "if the virus was introduced to C_i at time x, could it spread to C_j at time y?" That is, is there a sequence of communications that could have lead to the virus moving from C_i to C_j?