CS41 Lab 5: Greedy Graph Algorithms

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. **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[...].

   (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.

2. **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.

3. **A Simpler Wedding Planner Problem.** This problem is a formalized and simplified version of the Wedding Planner Problem you saw in Lab 1.

   There are n boolean variables $x_1, \ldots, x_n$. A *literal* is either a variable x_i or its negation $\bar{x}_i$. A 2-*constraint* consists of the OR of two literals $f \vee g$. Think of each variable as a person, a literal as a decision (invite Bob or don't invite Bob?) and a constraint as a description of what the bride and groom want in terms of invitations. (e.g. $x_i \vee \bar{x_j}$ means invite i or don't invite j).

   An assignment sets truth values for each variable. For example, if $n = 3$, one such assignment is $\{x_1 = \textsc{True}, x_2 = \textsc{False}, x_3 = \textsc{True}\}$. An assignment A satisfies a constraint $f \vee g$ if at least one of the literals f, g are satisfied. For example, the constraint $x_i \vee \not{x}_j$ is satisfied if either $x_i = \textsc{True}$ or $x_j = \textsc{False}$ (or both).

   Give a linear-time algorithm that takes a list of n variables and m 2-constraints and produces a satisfying assignment or returns that no such assignment exists. **Hint:** convert the input for this problem into a graph, then develop a greedy algorithm to find the satisfying assignment.

4. You're helping a group of ethnographers analyze some oral history data they've collected by interviewing members of a village to learn about the lives of people who have lived there over the past two hundred years.

   From these interviews, they've learned about a set of n people (all now deceased), whom we'll denote $P_1, P_2, \ldots, P_n$. They've also collected facts about when these people lived relative to one another. Each fact has one of the following two forms:

   - For some i and j, person P_i died before person P_j was born; or
   - for some i and j, the life spans of P_i and P_j overlapped at least partially.

   Naturally, they're not sure that all these facts are correct; memories are not very good, and a lot of this was passed down by word of mouth. So what they'd like you to determine is whether the data they've collected is at least *internally consistent*, in the sense that there could have existed a set of people for which all the facts they've learned simultaneously hold.

Give an efficient algorithm to do this: either it should produce proposed dates of birth and death for each of the n people so that all the facts hold true, or it should report (correctly) that no such dates can exist—that is, the facts collected by the ethnographers are not internally consistent.