CS41 Lab 6: Greedy Graph Algorithms

October 12 2020

In typical labs this semester, you'll be working on a number of problems in groups of 3-4 students. You will not be handing in solutions; the primary purpose of these labs is to have a low-pressure space to discuss algorithm design. However, it will be common to have some overlap between lab exercises and homework sets.

1. In CS35, you likely saw the *shortest path on weighted graphs* problem. In this problem, each edge e has an edge length ℓ_e, and the length of a path $s \rightsquigarrow t$ is the sum of the edge lengths along the path. Your goal is to find, for a specific start vertex, the length of the shortest $s \rightsquigarrow v$ path for all vertices v.

   Problem: Shortest Paths

   Inputs:

   - A graph $G = (V, E)$
   - For each edge e a positive *edge length* ℓ_e
   - A start vertex $s \in V$

   Output: a List of distances d, where $d[v]$ is the length of the shortest $s \rightsquigarrow v$ path.

   Below is pseudocode for Dijkstra's Algorithm, which finds the shortest path in a graph G between a start vertex s and any other vertex.

   DIJKSTRA(G, s, ℓ)

   ```
   1  S = {s}.
   2  d[s] = 0.
   3  while S ≠ V
   4        pick v ∈ V \ S to minimize min_{e=(u,v):u∈S} d[u] + ℓ_e.
   5        add v to S.
   6        d[v] = d[u] + ℓ_e
   7  Return d[...].
   ```

   (a) Show that Dijkstra's Algorithm solves the shortest path problem. (Hint: use the stays ahead method)

   (b) What is the asymptotic running time of Dijkstra's 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.

   Note: the pseudocode given above is *high-level* pseudocode. One reason why this is high-level is because it doesn't specify how to compute the edge $e = (u, v)$ such that $u \in S$ that minimized $d[u] + \ell_e$. You'll need to understand how to compute this edge efficiently.

2. **Minimum Spanning Trees: implementation.** Two common greedy MST algorithms are:

- Prim's algorithm: 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 $V \setminus S$.
- Kruskal'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.

We saw high-level pseudocode for both these algorithms in class. Implementation details **matter a lot** in considering which of these algorithms to use.

(a) What is the asymptotic running time of Prim's 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.

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

3. **Longest Path** In CS35, you saw that BFS can be used to find the *shortest path* in an unweighted graph, where "shortest path" here means the path with the fewest number of edges.

(a) Give an efficient algorithm which takes a directed acyclic graph $G = (V, E)$ as input and returns the length of the *longest* path in G.

(b) Give an efficient algorithm which takes a directed graph $G = (V, E)$ and returns the length of the longest path in G.