CPSC 320: Intermediate Algorithm Design and Analysis
Assignment #3, due Thursday, February 9th, 2012 at 11:00

[9] 1. In class, we discussed an algorithm that constructs a minimum prefix tree for a string of length n with m distinct characters. Show how to construct the minimum prefix tree in $O(m)$ time in the case where the input to the algorithm is a list of pairs

$$(c_1, f_1), (c_2, f_2), \ldots, (c_m, f_m)$$

where f_i is the frequency of the character c_i, and the f_i's are sorted in increasing order (that is, $f_1 \leq f_2 \leq \cdots \leq f_m$). Hint: what can you say about the sequence of frequencies assigned to the nodes created when you merge two already existing trees?

Solution: The algorithm relies on the following observation: because we always merge the two nodes with lowest frequencies, the nodes we create are sorted by increasing frequency. The revised algorithm thus maintains two sorted lists:

- The original input $(c_1, f_1), (c_2, f_2), \ldots, (c_m, f_m)$.
- The nodes created after merging other nodes, which it will store in a queue (whose elements will thus be sorted, by the observation).

In order to select the two nodes with lowest frequencies, the algorithm will consider the first remaining element of the original input, and the node at the front of the queue, and use that with the lowest frequency. It will then repeat the operation to obtain the node with the second lowest frequency. The node created by the merge will then be enqueued.

[13] 2. Explain what adjustments, if any, need to be made to Dijkstra's single-source shortest paths algorithm and/or to the given input graph to solve the following problems.

a. Find a shortest path between two given vertices of a weighted graph or a directed graph (this variation is called the *single-pair shortest-path problem*).

Solution: We use Dijkstra's algorithm as normal, starting at one of the two given vertices, and stop as soon as the second vertex has been extracted from the heap. As proved in class, by the time a vertex is extracted from the heap, the shortest path from the course to that vertex has already been found. Hence our algorithm is correct.

b. Find the shortest path to a given vertex from each other vertex of a weighted directed graph (this variation is called the *single-destination shortest-paths problem*).

Solution: This can be accomplished by reversing the direction of every edge of the graph, running Dijkstra's algorithm on the resulting graph, and finally reversing the direction of every edge in the resulting tree. The correctness of this algorithm is clear.

c. Solve the single-source shortest-paths problem in a graph with non-negative numbers assigned to its vertices (and the length of the path defined as the sum of the vertex numbers on the path).

Solution: The question did not specify whether the graph was directed or undirected, so we need a solution that works for both. If the graph is undirected, we start by making it a directed graph by replacing every undirected edge $\{v, w\}$ by the pair of directed edges (v, w) and (w, v). We will henceforth assume the graph is directed.

We modify the graph in three steps:

i. First, we assign a weight of 0 to every edge of the graph.

ii. Next, we replace every vertex v by a pair of vertices v_{in}, v_{out}. All edges coming into v now come into v_{in}, and all edges going out of v now go out of v_{in}.

iii. Finally, for each pair v_{in}, v_{out}, we add an edge from v_{in} to v_{out} whose weight is the non-negative value originally assigned to vertex v.

We then run Dijkstra's algorithm on the modified graph, starting at vertex s_{in}. Every path from a vertex v to a vertex w in the original graph has the same cost as the corresponding path from v_{in} to w_{out} in the modified graph. This means that the cost associated with a vertex v_{out} at the end of the algorithm will be the length of the shortest path from s to v in the original graph.