Problem 1. LEMPEL-ZIV ENCODING
(a) [10 pts] Encode the following string using Lempel-Ziv encoding:
B B A A B B A B A B A A
Show the contents of the (new) dictionary and the actual encoded string.
(b) [10 pts] Decode the following string using Lempel-Ziv decoding:
66 256 66 65 258 259
Show the contents of the (new) dictionary and the actual decoded string.

Problem 2. SHORTEST PATHS
(a) [12 pts] Run Dijkstra’s algorithm on the graph of Figure 24.2 in the text (page 648) using vertex z (not s!) as the source. Show (1) the order in which nodes are removed from the heap [5 pts], and (2) the updates made to the distances and predecessors of each node [7 pts].

(b) [10 pts] Suppose we want to send goods by truck from a production facility in New York to a warehouse in Chicago. We would like to make as few trips as possible (since we have only one truck), and the critical issue is that each road segment (the piece of a road between two intersections) has a weight limit for the truck. Therefore, if for example, a road segment has weight limit 10 tons, then we either have to put at most 10 tons of goods in the truck, or not use that road segment. Suppose we know the weight limit for each road segment.

Our goal is to find a route which allows the heaviest truck load (i.e. with the highest weight limit), even if it is longer than a route with a lower weight limit. For example, for the graph in Figure 24.2, if the edge weights are the loads, then the best path from s to x is s − y − z − x with a weight limit of 5.

Carefully describe an efficient algorithm to find such a route. You do not need to provide pseudocode.

(c) [5 pts] If there are r road segments (which connect two intersections) and j intersections (where we can change roads), what is the running time of your solution in (b)?

Problem 3. SHORTEST PATHS

Figure 1: Sample graph

Consider the problem of finding a shortest path from a start node s to a destination node t in a directed graph with positive edge weights.
One solution is to simultaneously run Dijkstra’s algorithm from both \( s \) and \( t \). Specifically, we first scan (look at the neighbors of) \( s \) and update their \( \text{dist} \) values. Then we scan the nodes with edges \textit{directed to} \( t \) and update their distance value from \( t \). To avoid confusion, let \( t_{\text{dist}} \) denote the values for distances from \( t \) (note that when considering distances “from” \( t \) we are actually considering the graph with the directions reversed: if there is a node \( x \) with an edge to \( t \) of length 3, then we set \( x.t_{\text{dist}} \) to 3). Thus if we find a node \( y \) such that \( y.\text{dist} = 10 \) and \( y.t_{\text{dist}} = 15 \), we know there is a path of length 25 going through \( y \).

Next we consider the unprocessed vertex with the smallest \( \text{dist} \) value, and update its neighbors’ \( \text{dist} \) values. Then we switch back to our other search, and select the unprocessed vertex with the smallest \( t_{\text{dist}} \) value, and update the \( t_{\text{dist}} \) values of its neighbors (note that by “unprocessed” we mean with respect to the search from \( s \) or \( t \) respectively).

We continue going back and forth between the two searches until we can be sure we have found a shortest path.

(a) [10 pts] Show the progress and result of this algorithm on the example in Figure 1. Specifically, show the updates made to \( \text{dist} \) and \( t_{\text{dist}} \) for each vertex and give the final shortest path and its distance.

(b) [15 pts] When can we be sure we have the shortest path? Formally, describe what conditions need to hold, what the shortest path will be and what the shortest distance will be. Justify your answer.