Problem 1. [Greedy/Union-Find]

Use Dijkstra’s algorithm to compute the cost of the shortest (i.e., minimum weight) path from vertex a to the other vertices. Indicate the D value and the vertices in the cloud C after each iteration of the main loop in the table below.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>{a}</td>
<td>0</td>
<td>+\infty</td>
<td>+\infty</td>
<td>+\infty</td>
<td>+\infty</td>
<td>+\infty</td>
<td>+\infty</td>
<td>+\infty</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{a, h, b, g, c, d, f, e}</td>
<td>0</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>11</td>
<td>10</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

Problem 2. [Greedy/Union-Find]

Prove the following statement. Let G = (V, E) be a weighted undirected graph. If all the edge weights in G are distinct, the minimum spanning tree is unique.

Answer: Suppose for contradiction that there are two distinct spanning trees T and T' for G, which means that differ by at least one edge. Among those edges that are in only one of the two trees, let e be one of minimum cost. Assume without loss of generality that e ∈ T (the other case, e ∈ T', is symmetric). Adding e to T' creates a simple cycle C. Since T is acyclic, C must have an edge e' that is not in T. Since e' is in T − T', by the choice of e, the weight of e is at most the weight of e'. Since e' and e must have different weights, the weight of e is strictly less than the weight of e'. Let T'' = T' ∪ \{e\} − \{e'\}. Then T'' is a spanning tree of weight less than the weight of T', which contradicts the assumption that T' is a minimum spanning tree.

Problem 3. [Dynamic Programming]

Complete the Theorem we covered in class about the optimal substructure for the Longest Common Subsequence problem. No need to prove the theorem.

Answer: See slides.

Problem 4. [Dynamic Programming]

Given an array A = \{a_1, a_2, \ldots, a_n\} of integers, we say that a subsequence \{a_{i_1}, a_{i_2}, \ldots, a_{i_k}\} is (monotonically) increasing if for every i_s < i_t, we have a_{i_s} < a_{i_t}. Given an array A of size n, we want to compute the length of the longest increasing subsequence (LIS) in A. For instance, if A = \{9, 5, 2, 8, 7, 3, 1, 6, 4\} the length of the LIS is 3, because (2, 3, 4) (or (2, 3, 6)) are LIS of A. Give
a $O(n^2)$ dynamic programming algorithm for this problem. Analyze the time- and space-complexity of your solution.

**Answer:** Define $L(i)$ be the length of the LIS for a prefix $\{a_1, \ldots, a_i\}$ of $A$ such that $a_i$ is the last element in LIS; then $L(i)$ can be recursively written as:

$$L(i) = \begin{cases} 
1 & \text{if } i = 1 \\
1 + \max_{1 \leq j < i} \{L(j) : a_j < a_i\} & \text{otherwise}
\end{cases}$$

where we assume that the max of an empty set would return zero.

Time complexity is $O(n^2)$ because it takes linear time to fill each entry of the array (it is possible to decrease the total complexity to $O(n \log n)$, but it is a little more complicated). Space complexity is $O(n)$. 
