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.

Answer:

<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>+∞</td>
<td>+∞</td>
<td>+∞</td>
<td>+∞</td>
<td>+∞</td>
<td>+∞</td>
<td>+∞</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 \( e \in T \) (the other case, \( e \in T' \), 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 \( T'' = T' \cup \{e\} - \{e'\} \). 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]

Give the pseudocode for the algorithm we explained in class that computes the longest common subsequence (LCS) between two strings \( x \) and \( y \) using linear space. You can assume that the routine \textsc{LenLCS}(x, y) that returns the length of the LCS between string \( x \) and \( y \) in linear space is available. For simplicity, you can assume the length of one of two strings to be a power of two.

Answer:
Algorithm \textsc{LinearLCS}(x, y : string)
1 \hspace{0.5em} n, m \leftarrow |x|, |y|
2 \hspace{0.5em} \textbf{if} n = 1 \textbf{then return} x[1]
3 \hspace{0.5em} \text{pre} \leftarrow \text{LenLCS}(x[1 : n/2], y)
4 \hspace{0.5em} \text{suf} \leftarrow \text{LenLCS}(x^R[1 : n/2], y^R)
5 \hspace{0.5em} \text{len} \leftarrow \text{pre} + \text{suf}
6 \hspace{0.5em} \text{mid} \leftarrow \arg\max_{1 \leq i \leq m} \text{len}[i]
7 \hspace{0.5em} \textbf{return} \text{LinearLCS}(x[1 : n/2], y[1 : \text{mid}]), (x[n/2], y[\text{mid}]), \text{LinearLCS}(x[n/2 + 1 : n], y[\text{mid} + 1 : m])

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 \( O(n) \).