Problem 1. [Greedy] (25 points)
Prove (or disprove) the following statement on the optimal substructure of any shortest path on a weighted graph \( G = (V, E) \). \textbf{Fact:} Let \( p = \{e_1, e_2, \ldots, e_{k-1}\} \) be the shortest path from \( v_1 \) to \( v_k \) in \( G \), composed of the following \( k-1 \) edges \( e_1 = (v_1, v_2), e_2 = (v_2, v_3), \ldots, e_{k-1} = (v_{k-1}, v_k) \). Then, \( \{e_i, \ldots, e_j\} \) must be the shortest path from \( v_i \) to \( v_{j+1} \) for all choices of \( 1 \leq i < j < k \).

\textbf{Answer:} Let us decompose shortest path \( p \) into three sub-paths, namely \( \{e_1, e_2, \ldots, e_{i-1}\}, \{e_i, e_{i+1}, \ldots, e_j\} \) and \( \{e_{j+1}, e_{j+1}, \ldots, e_{k-1}\} \) where \( 1 \leq i < j < k \). If \( i = 1 \) the first subpath would be empty. If \( j = k-1 \) the third subpath would be empty.

The total weight of \( p \) is the sum of the weights of the three sub-paths. If there was a better path from \( v_i \) to \( v_{j+1} \), then we could replace \( \{e_i, e_{i+1}, \ldots, e_j\} \) in \( p \) with a cheaper path, which would imply that \( p \) was not optimal. This is a contradiction, thus \( \{e_i, e_{i+1}, \ldots, e_j\} \) is the shortest path from \( v_i \) to \( v_{j+1} \).

Problem 2. [Greedy] (25 points)
You are given two unsorted arrays \( A = \{a_0, a_1, a_2, \ldots, a_n\} \) and \( B = \{b_0, b_1, b_2, \ldots, b_n\} \) composed of distinct positive integers. Give a \( O(n \log n) \)-time greedy algorithm that determines an ordering of the elements of \( A \) and \( B \) such that \( W = \prod_{i=1}^{n} a_i b_i \) is maximized. Explain why your algorithm runs in \( O(n \log n) \)-time, and prove the greedy choice property for your algorithm. No need to prove the optimal substructure.

\textbf{Answer:} Here is the algorithm:

- **Algorithm** \textsc{Greedy}(\( A : \text{array}, B : \text{array} \))
  - \textbf{sort} \( A \) and \( B \) in decreasing order
  - \textbf{return} \( (A, B) \)

The algorithm is \( O(n \log n) \) because of the sorting step. Next we prove that \textsc{Greedy} has the greedy choice property for this problem.

\textbf{Proof:} The usual exchange argument. Consider any indices \( i \) and \( j \) such that \( i < j \), and consider the terms \( a_i^{b_i} \) and \( a_j^{b_j} \). We want to show that the objective function \( W \) will not get worse by taking \( a_i^{b_i} \) and \( a_j^{b_j} \) instead. In other words, we need to show that \( a_i^{b_i} a_j^{b_j} \geq a_i^{b_j} a_j^{b_i} \). Since \( A \) and \( B \) are sorted in decreasing order and \( i < j \) we have \( a_i \geq a_j \) and \( b_i \geq b_j \). Since \( a_i \) and \( a_j \) are positive and \( b_i - b_j \) is nonnegative, we have \( a_i^{b_i-b_j} \geq a_j^{b_i-b_j} \). Multiplying both sides by \( a_i^{b_j} a_j^{b_j} \) yields \( a_i a_j^{b_j} \geq a_i a_j^{b_j} \).

Problem 3. [Dynamic Programming][Design] (25 points)
You are given a directed graph \( G = (V, E) \), two vertices \( s \) and \( t \), and an integer \( k \). We want to compute the number of paths in \( G \) from \( s \) to \( t \) that have exactly \( k \) edges. The path does not have to be \textit{simple}, i.e., vertices can be used more than once. Give a dynamic-programming algorithm that runs in time \( O((n + m)k) \). Analyze the time- and space-complexity of your algorithm.

\textbf{Answer:} Define \( M[v, i] \) be the number of paths from \( s \) to \( v \in V \) that have exactly \( i \) edges, \( 0 \leq i \leq k \). The recurrence relation is

\[ M[v, i] = \begin{cases} 1 & \text{if } v = t \text{ and } i = 0 \\ 0 & \text{if } v \neq t \text{ and } i = 0 \\ \sum_{w: (w, v) \in E} M[w, i - 1] & \text{if } i > 0 \end{cases} \]

The algorithm initializes \( M[v, 0] \) for all \( v \) as above, then, for \( i = 1, 2, \ldots, k \), computes \( M[v, i] \) using the recurrence above. There are \( k \) iterations. Each iteration does constant work for each edge and vertex of \( G \), so the total time per iteration is \( O(n + m) \). The space-complexity is \( O(nk) \). The time-complexity is \( O((n + m)k) \).
Problem 4. [Dynamic Programming][Knowledge] (25 points)

We want to extend the LCS dynamic programming algorithm we covered in class to find the longest common subsequence between three strings $X$, $Y$ and $Z$.

Let $X_i$ be a prefix of string $X$ of length $i$, $Y_j$ be a prefix of string $Y$ of length $j$, and $Z_k$ be a prefix of string $Z$ of length $k$. If we define $C[i, j, k]$ to store the length of the longest common subsequence between $X_i$, $Y_j$, and $Z_k$, then

$$C[i, j, k] = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \text{ or } k = 0 \\
C[i - 1, j - 1, k - 1] + 1 & \text{if } i > 0, j > 0, k > 0 \text{ and } X[i] = Y[j] = Z[k] \\
\max\{C[i - 1, j, k], C[i, j - 1, k], C[i, j, k - 1]\} & \text{otherwise}
\end{cases}$$

The time complexity of this algorithm is $O(lmn)$. The space complexity of this algorithm is $O(lmn)$.