1. **ASYMMETRIC TSP.**

   **[20 Points]**

   You are given the complete directed graph $G = (V, E)$ over $n$ vertices. There is an associated weight function $w : E \to \mathbb{R}^+$ on the edges, that complies with the directed version of the triangle inequality. That is $w(x \to y) + w(y \to z) \geq w(x \to z)$, for any $x, y, z \in V$. Notice, however, it is quite possible that for some $x, y \in V$ we have that $w(x \to y) \neq w(y \to x)$ (i.e., the weight function is asymmetric).

   Providing an approximation to the TSP in this case is quite harder than the undirected case, and we will tackle it in this question.

   (A) **[5 Points]** Show how to compute in polynomial time a set of vertex disjoint cycles of total minimum cost that covers all the vertices in the graph. (This is easy, but you want to be careful here.) Note, that every cycle would have at least two edges.

   (B) **[5 Points]** Let $X \subseteq V$ be a subset of the vertices of $G$. Prove that there exists a cycle that visits only the vertices of $X$ of cost at most $\text{Cost}_{\text{opt}}$, where $\text{Cost}_{\text{opt}}$ is the cost of the optimal TSP visiting all the vertices of $G$.

   (C) **[5 Points]** Consider the algorithm that computes, using (A) a “cheap” cover by cycles of the vertices of the graph, it then selects a vertex from each cycle, and let $Z$ be this set of vertices. Next, the algorithm recursively compute a cheap TSP for the vertices of $Z$, and it somehow generates a TSP tour for the whole graph.

   Describe precisely how the algorithm computes the TSP.

   (D) **[5 Points]** What is the bound on the quality of approximation provided by the TSP computed by the above algorithm? Prove your answer.

2. **SEQUENCES AND CONSEQUENCES.**

   **[20 Points]**

   Let $\mathcal{X} = \langle X_1, X_2, \ldots, X_n \rangle$ be a sequence of $n$ numbers generated by picking each $X_i$ independently and uniformly from the range $\{1, \ldots, n\}$.

   (A) **[5 Points]** What is the entropy of $\mathcal{X}$?

   (B) **[5 Points]** Consider the sequence $\mathcal{Y} = \langle Y_1, \ldots, Y_n \rangle$ that results from sorting the sequence $\mathcal{X}$ in increasing order. For example, if $\mathcal{X} = \langle 4, 1, 4, 1 \rangle$ then $\mathcal{Y} = \langle 1, 1, 4, 4 \rangle$.

   Describe an encoding scheme that takes the sequence $\mathcal{Y}$ and encodes it as a sequence of $2n$ binary bits (you will lose points if your scheme uses more bits). Given this encoded sequence of bits, how do you recover the sequence $\mathcal{Y}$? (Hint: Consider the differences sequence $Y_1, Y_2 - Y_1, Y_3 - Y_2, \ldots, Y_n - Y_{n-1}$. And do not use Huffman’s encoding.)

   Demonstrate how your encoding scheme works for the sequence $\mathcal{Y} = \langle 1, 1, 4, 6, 6, 6 \rangle$.

   (C) **[5 Points]** Consider the set $U$ of all sequences $\mathcal{Y}$ that can be generated by the above process (i.e., it is the set of all monotonically non-decreasing sequences of length $n$ using
integer numbers in the range 1 to $n$). Provide (and prove) an upper bound on the number of elements in $U$. Your bound should be as small as possible. (Hint: Use (B).)
(Note, that we are not asking for the exact bound on the size of $U$, which is doable but harder.)

(D) [5 Points] Prove an upper bound (as low as possible) on the entropy of $\mathcal{Y}$. (Proving a lower bound here seems quite hard and you do not have to do it.)

3. **In the search for worthy ancestors.**

[20 Points]
You are given a tree $T$ with $n$ nodes, and root $r$. In the following, you can assume that one can compute the depth of a node of $T$ in constant time, and that given any node $v$ and any $k > 0$, one can compute $\text{ancestor}(v, k) = \text{parent}^{(k)}(v)$ of $v$ in constant time.

(A) [5 Points] Show how to preprocess $T$, such that given a query which is made out of two nodes $u, v$ of $T$, one can decide, in $O(1)$ time, if $u$ is a descendant of $v$. (We denote this new operation by $\text{isDescendant}(u, v)$.)

(B) [5 Points] Using (A), describe an algorithm, as fast as possible, such that given a query made out of two nodes $u, v$ of $T$, it computes the LCA (least common ancestor) of $u$ and $v$ in $T$. How fast is your algorithm as a function of $n$? How fast is it in term of $h$ ($h$ is the height of $T$)?

(C) [1 Points] Assume that you are given an operation $\text{firstOnBit}(x)$, which returns the location of the first bit of $x$ ($x$ is a non-negative integer number) which is non-zero (starting from the least significant bit). For example $\text{firstOnBit}(6) = \text{firstOnBit}(110_2) = 1$ and $\text{firstOnBit}(5) = \text{firstOnBit}(101_2) = 0$, and $\text{firstOnBit}(16) = \text{firstOnBit}(10000_2) = 4$.

Describe the implementation of a function, denoted by $g(x, y)$, such that given two positive integers $x$ and $y$ it computes the first bit in which they differ. You can assume that you are allowed to use all standard bitwise operations, like AND, OR, XOR, and NOT.

(D) [8 Points] Assume that every node in $T$ has at most two children. Show how to preprocess $T$, so that given two nodes $u, v$ of $T$, one can compute their LCA in $O(1)$ time. How much time does the preprocessing takes?

(E) [1 Points] Why is the solution for (D) is unacceptable in practice?

4. **Strong duality.**

[20 Points]
Consider a directed graph $G$ with source vertex $s$ and target vertex $t$ and associated costs $\text{cost}(\cdot) \geq 0$ on the edges. Let $\mathcal{P}$ denote the set of all the directed (simple) paths from $s$ to $t$ in $G$.

Consider the following (very large) integer program:

\[
\begin{align*}
\text{minimize} & \quad \sum_{e \in E(G)} \text{cost}(e) x_e \\
\text{subject to} & \quad x_e \in \{0, 1\} \quad \forall e \in E(G) \\
& \quad \sum_{e \in \pi} x_e \geq 1 \quad \forall \pi \in \mathcal{P}.
\end{align*}
\]

2
(A) [5 Points] What does this IP computes?

(B) [5 Points] Write down the relaxation of this IP into a linear program.

(C) [5 Points] Write down the dual of the LP from (B). What is the interpretation of this new LP? What is it computing for the graph G (prove your answer)?

(D) [5 Points] The strong duality theorem states the following.

**Theorem 0.1** If the primal LP problem has an optimal solution \( x^* = (x_1^*, \ldots, x_n^*) \) then the dual also has an optimal solution, \( y^* = (y_1^*, \ldots, y_m^*) \), such that

\[
\sum_j c_j x_j^* = \sum_i b_i y_i^*.
\]

In the context of (A)–(C) what result is implied by this theorem if we apply it to the primal LP and its dual above? (For this, you can assume that the optimal solution to the LP of (B) is integral.)

5. A RANDOM QUESTION.

[20 Points]

(A) [2 Points] Let \( G = (V, E) \) be a graph with \( n \) vertices and \( m \) edges, with a minimum cut of size \( k \).

Pick every edge of \( G \) into a set \( R \) with probability \( 1/k \) (this is done independently for each edge). What is the expected size of \( R \)?

(B) [2 Points] Consider a set \( X \) of \( k \) edges in \( G \) that realizes the minimum cut. What is the **exact** probability that no edge of \( X \) is in \( R \)?

(C) [2 Points] Let \( H \) be the graph resulting from contracting all the edges of \( R \) in the graph \( G \). Give a lower bound on the probability that the minimum cut of \( G \) survives in the graph \( H \).

(D) [4 Points] Let \( f \) and \( g \) be two given functions defined over the numbers \( 1, \ldots, n \). There are two possibilities: (i) \( f(i) = g(i) \) for all \( i \), or (ii) there are at most \( k \) values \( i_1 < i_2 < \ldots < i_k \) such that \( f \) and \( g \) are equal for these values and they are different for all other values.

Assume that evaluating \( f \) and \( g \) is expensive. Describe a deterministic algorithm that decides if \( f \) and \( g \) are equal on all values. How many evaluations of \( f \) and \( g \) your algorithm performs (the fewer the better, naturally).

(E) [8 Points] Describe a randomized algorithm that receives \( f \) and \( g \), and a parameter integer \( \beta \) (think about \( \beta \) as being 2 or 3). Describe a randomized algorithm that performs exactly \( \beta \) evaluations of \( f \) and \( \beta \) evaluations of \( g \), and the algorithm outputs whether it thinks \( f = g \) or \( f \neq g \).

What is the probability of your algorithm to return a correct answer for the case \( f = g \)?

What is the probability of your algorithm to return a correct answer for the case \( f \neq g \)?

(Naturally, the higher these probabilities, the better your algorithm is.)