Due date: November 25, 2019

Problem 1: Consider the following modification of Karger’s algorithm for finding a minimum \( s-t \) cut in an unweighted, undirected graph. In a given iteration, let \( s \) and \( t \) denote the possibly contracted vertices that contain the original vertices \( s \) and \( t \), respectively. Select an edge that does not connect \( s \) and \( t \) uniformly at random, and contract it. Continue until there are only two vertices remaining, these must be \( s \) and \( t \). Output the corresponding \( s-t \) cut. Give an example to show that the probability this algorithm finds a minimum \( s-t \) cut can be exponentially small.

Problem 2: Let \( S = \{ p_1, \ldots, p_n \} \subset [\Delta]^2 \) be a set of \( n \) points that lie on the 2-dimensional grid where \( \Delta = O(n) \). Pick uniformly at random \( b \in [\Delta]^2 \). Consider the randomly shifted quadtree \( T \) for \( S \) having \( R_{\text{root}} = b + [-\Delta, \Delta]^2 \) as the root cell. Let \( R_v \) be the square associated with node \( v \), and let \( l_v \) be the side length of \( R_v \). Let the weight of each edge be the side length of the lower square, i.e. \( w(v, \text{parent}(v)) = l_v \). For a pair \( p, q \in S \), let \( \pi(p, q) \) be the path in \( T \) between the leaves containing \( p \) and \( q \). The distance of \( p, q \) in the quadtree is \( d_T(p, q) = \sum_{e \in \pi(p, q)} w_e \). Show that \( E[d_T(p, q)] = O(\log n) \left\| p - q \right\| \).

(Hint: What is the probability that \( \text{lca}(p, q) \) is at depth \( i \)?)

Problem 3:
(i) Let \( T = (V, E) \) be a rooted tree with edge-weight function \( w : E \to \mathbb{R}^+ \). Let \( R \) and \( B \) be two point sets stored at the vertices of \( T \), where \( |R| = |B| = n \). That is, \( R \) (resp. \( B \)) can be regarded as a function \( R : [1, \ldots, n] \to V \) (resp. \( B : [1, \ldots, n] \to V \)). The distance \( d(r, b) \) is the weight of the path in \( T \) between the vertices storing \( r \) and \( b \). Show that a minimum-weight perfect matching between \( R \) and \( B \) can be computed in \( O(n) \) time.

(ii) Let \( R, B \subset [\Delta]^2 \) be two point sets such that \( |R| = |B| = n \) and \( \Delta = O(n) \). Describe an \( O(n \log n) \) time algorithm to compute a perfect matching between \( R \) and \( B \) whose expected cost is \( O(\log n) \) times that of the Euclidean minimum-weight perfect matching, i.e., the minimum-weight perfect matching in the complete graph \( G = (R \cup B, R \times B) \) with \( w(r, b) = \| r - b \| \).

(Hint: Use Problem 2 and part (i).)

Problem 4: Let \( S = \{ p_1, \ldots, p_n \} \subset \mathbb{R}^2 \) and \( \varepsilon > 0 \). A \( \varepsilon \)-spanner of \( S \) is a weighted graph \( G \) whose vertices are the points of \( S \), and for any \( p, q \in S \) we have:

\[
\| p - q \| \leq d_G(p, q) \leq (1 + \varepsilon) \| p - q \| ,
\]

where \( d_G(p, q) \) denotes the shortest-path distance between \( p \) and \( q \) in \( G \). Consider the following construction: first build a quadtree \( T \) for \( S \). For every node \( v \), choose an arbitrary point in the square \( R_v \) and denote it \( \text{rep}_v \). Also let \( l_v \) be the side length of \( R_v \). Let \( \delta = \varepsilon / c \) for some constant \( c \geq 16 \). For each node \( v \in T \), let \( w \) be a node in \( T \) at the same level as \( v \) such that the distances of their centers is at most \( l_v / \delta \). We add \( (\text{rep}_v, \text{rep}_w) \) as an edge to \( G \) with weight \( \| \text{rep}_v - \text{rep}_w \| \). Prove that \( G \) is an \( \varepsilon \)-spanner of \( S \).

(Hint: Use induction on the \( \binom{n}{2} \) distances of \( S \)).

Problem 5: Suppose we are getting a stream of real values. Let \( S \) be the set of items seen so far, and let \( \varepsilon > 0 \) be a parameter. Describe a sketch of size \( O(\frac{1}{\varepsilon} \log n) \) so that for a query \( x \in \mathbb{R} \), its rank in \( S \) can be estimated with additive error at most \( \varepsilon n \). You can assume that you know the maximum value of \( n \). (Hint: Store a subset of values.)