1. **Another MST Algorithm**

   In this question you will design a new, greedy, algorithm for computing the minimum spanning tree (MST) T of a connected, weighted, undirected graph $G = (V, E)$, with edge weights $w_{(u,v)}$ for edges $(u, v) \in E$.

   (a) Consider the following Lemma:

   > **Lemma.** Consider any cycle in G, and let (u, v) be the edge in that cycle with maximum weight. There exists an MST of G that does *not* include edge (u, v).

   Assuming that this lemma is true, design an algorithm that computes an MST of G by greedily/iteratively deleting edges.

   (b) Prove the lemma from part (a).

   SOLUTION

   (a)
```
Initialize T=G
while there exists a cycle in T
   find such a cycle, update T by removing the max-weight edge of the cycle
```

   To test whether a cycle exists and find a cycle if one does exist, one could consider the graph defined by T, but with all edges having weight -1, and then run the BellmanFord shortest-path algorithm that will find any (negative-weight) cycle if one exists. Any cycle in T will have negative weight in this augmented graph, since all edges have negative weight.

   (b) Fix a cycle C and a maximum-weight edge $e = (u, v)$ in the cycle with weight w. If no MSTs contain edge e, then the lemma holds. Otherwise, consider an MST T that contains edge e. Removing edge e from T will disconnect T, leaving node u in one connected component and node v in the other. There exists an edge $e' \neq e$ of cycle C that connects these these two connected components. [To see this, consider traversing the edges of C to get from u to v the 'long way around', and letting e' be the first edge that would cross from a node in the connected component of u to the connected component of v.] By assumption, $weight(e') \leq weight(e)$, hence the tree T' that is identical to T but uses edge e' instead of e will be a spanning tree with weight at most the weight of T.

2. **Points in Space**

   Imagine a set of n points $X = \{x_1, \ldots, x_n\}$ in some unknown space. The only thing we know about them is the distance $d(x, y)$ between each pair of points x and y.

   We would like to compute a partition of X into disjoint sets of points $C_1, \ldots, C_k$ so as to maximize the minimum distance between any two points in different clusters. Each cluster must contain at least one point, and their union is the set X. More formally, we want a partition $\{C_1, \ldots, C_k\}$ of X that achieves

$$\max_{C_1,\ldots,C_k} \min_{\substack{x \in C_i \\ y \in C_j \\ i \neq j}} d(x, y)$$

   Give an efficient algorithm for this task.

   Hint: Try using an MST algorithm as a sub-routine!

SOLUTION

Algorithm:

The following algorithm finds such a set of points:

- Assemble a graph $G = (X, E)$ with the points as nodes. This is a *complete* graph, meaning that G contains an edge between every pair of nodes. Each $(x, y) \in E$ has $d(x, y)$ as its weight.
- Compute an MST T of G.
- Delete the $k1$ heaviest edges in T to form a forest .
- Using DFS on F, find the connected components (each of which is a tree) $C_1, ..., C_k$, of F.
- Designate the nodes of each component C_i as belonging to the same cluster. Return these clusters.

Correctness:

First observe that our algorithm returns a partition of X into exactly k clusters, since we start with 1 component and delete $k - 1$ edges; each deletion splits up a component.

It remains to show that the returned partition maximizes the minimum inter-cluster distance. The intuition is that an MST algorithm must pick light edges and leaves only heavy edges between components, so that components will have large distance between them. We now show this rigorously.

Lemma 1. *Suppose the heaviest edge e in the forest F has weight $w(e) = \delta$. Then all distances between different components/clusters are at least δ.*

Proof. Suppose towards contradiction that there are two nodes x and y in different components with distance $d(x, y) < \delta$. We reason as follows:

- Could (x, y) have been in the MST T? No, because if it was, then since it is lighter than e, we would have deleted e (or another edge with weight δ) rather than (x, y).
- So, in T, there was a path P from x to y that did not include (x, y). All edges in must have had weight w so that $w \leq d(x, y) < \delta$, for otherwise the MST should include (x, y) rather than that edge. Recall that x and y are in different components in F, implying that at least one edge $e \in P$ was deleted by our algorithm. But our algorithm deletes the heaviest edges first, so it could not have deleted e yet left in e. Contradiction. Hence there are no x, y in different components such that $d(x, y) < \delta$.

Now we use the Lemma to show that no other clustering can have better minimum separation (between clusters) than ours. To that end, consider another clustering $D_1, ..., D_k$. Since this clustering is different, there exist two points x, y that we place in the same cluster C, but the alternative clustering places them in different clusters: say $x \in D_i$ and $y \in D_j$. Now imagine the path between x and y in F:

$$x \cdots x'y' \cdots y$$

where $x' \in D_i$ and y' is in some other cluster D_l (not necessarily D_j). But all of the above nodes are in C, so (x', y') is in F and $d(x', y') \leq \delta$. Therefore in the clustering $D_1, ..., D_k$, the minimum distance between any two points in different clusters is at most δ, as witnessed by x and y. This shows that no alternative clustering is better than ours, and hence our algorithm is correct.

Running Time

This can be implemented by calling Kruskals algorithm to build T, which includes sorting E by weight in $O(|E| \log |E|)$ time, and then deleting heavy edges in $O(k) = O(|E|)$ time, and running DFS on F in $O(|X|)$ time, for a total of $O(|E| \log |E|) = (|E| \log |X|)$. Since is a complete graph, $|E| = (|X|^2)$, so in terms of the input, our running time is $O(|X|^2 \log |X|)$.