CS 325: Analysis of Algorithms
Group Assignment 3

Busra Dedeoglu
Hyun Taek Oh
Yu-Hao Shih

February 29, 2024

1 Description of Algorithm

In this problem, we are required to find a subset of edges that along with already selected edges form a spanning tree. We desire the total weights to be minimum among all possible spanning trees. To solve this problem, we propose the use of Prim's algorithm (best-first search with Manhattan distances being weights in priority queue). We make a small modification by assigning 0 to all distances associated with selected edges. This ensures that selected edges are surely chosen by the spanning tree algorithm. Below, we provide a more detailed description of the algorithm.

We store the given graph input in an adjacency matrix $\mathbf{A}$. Every entry in $\mathbf{A}$, A_{ij}, is computed using Manhattan distance, i.e., $A_{ij} = |x_i - x_j| + |y_i - y_j|$. Then, we construct and initialize a priority queue with $(\emptyset, 0), 0$. Subsequently, we follow Prim's algorithm: while the priority queue is not empty, we take highest priority (lowest weight) edge from the priority queue, check if the source vertex was visited before or not, and mark it if it was not visited before. Then, if the destination vertex was not visited before, and the current weight (distance) to that vertex is less than any previous edges, we assign current source as parent of this vertex, push the edge to priority queue, and update weight to reach destination vertex.

It is important to note that priority queue will pop higher priority (lower distance) edges first, ensuring minimum spanning tree. Once priority queue is emptied, we simply sum all weights corresponding to each vertex, i.e., weights of edges to reach the corresponding destination vertices. Since our algorithm finds a minimum spanning tree, every edge is guaranteed to have the minimum weights (distances). Also note that we assigned 0 to the weights that correspond to the selected edges. Therefore, final weights to reach those vertices are guaranteed to add up to 0. So, the final sum will be the total weight of all new edges E^* other than pre-selected edges E'.

2 Running Time Analysis

The running time of minimum spanning tree algorithm depends on the data structure used. Since we used adjacency matrix, we need to access $V = |\mathcal{V}|$ times to read all edges for a single source. To access all edges, we need to perform V^2 operations.

We are using priority queue as a bag of edges. Priority queue has logarithmic insertion complexity. We insert $2E$ many edges (because this is an undirected graph) at most and read them. Also, the input graph is a complete graph. Therefore $E = V^2$. Therefore, priority queue will cause $E\log(E) = V^2\log(V^2)$ running time complexity, where $E = |\mathcal{E}|$ and $\mathcal{E}$ is the set of all edges in the graph. Please note that accessing priority queue contents cannot exceed inserting into the priority queue, therefore reading does not add a multiplicative factor. The final running time complexity becomes $E\log(E) = V^2\log(V^2)$, which would be same as a minimum spanning tree algorithm. This analysis follows the analysis demonstrated at the end of section 5.5 in the book.

3 Proof of Correctness

The proof presented here follows the proof of Lemma 5.1 presented in the book. Similarly, we first show that every vertex is reachable from any vertex $s \in \mathcal{V}$. Let us show any $v \in \mathcal{V}$ can be reached from s. Since $\mathcal{V}$ is connected, $\exists u \in \mathcal{V}$ such that there is a path from s to u and from u to v. Then, by attaching both paths and since $\mathcal{G}$ is a undirected graph, there is a path from s to v, and hence v is reachable. The proposed algorithm puts every edge into the bag and then uses them. Therefore, there exists at least one such path that the algorithm could pick. Let us assume the algorithm does not pick any edge $(*, v)$, i.e., the parent of v is not assigned. We know there exists at least one u that has an edge (u, v) due to above argument. However, the algorithm may decide not to assign u as parent of v if the weight of the edge w_{uv} is larger than the previous assigned weight. This can only happen 2 ways: 1. $w_{uv} = \infty$, which concludes there is not a path to v and hence $\mathcal{G}$ is not a connected graph. This leads to a contradiction! 2. v was visited before with a lower weight edge. In this case there is another $\mu \in \mathcal{V}$ that connects s to v and algorithm finds it. This concludes the fact that the output of the algorithm is a tree.

To show the final spanning tree is actually a minimum spanning tree, we need to show that all weights are the minimum. Let us assume there exists m possible parents to a destination node v. Algorithm is proven to check at least one of these edges (u, v) by the above argument. Note that the priority queue enforces that the edges are sorted in the following order: $(u_1, v), (u_2, v), ..., (u_m, v)$ such that $w_{u_1v} < w_{u_2v} < ... < w_{u_mv}$.Therefore, (u_1, v) is guaranteed to be visited if v was marked as visited. Therefore, any node v is ensured to have the parent that minimizes the weight to get connected to the spanning tree, and hence the resulting spanning tree is a minimum spanning tree.

Finally, we show that the total weights of the final spanning tree is actually the total weights of E^*. Note that the algorithm assigns 0 values to all the edges in E'. Also, we here notice that any other edge $s \notin E'$ will have weights $w_{sv} > 0$. Therefore, any edge $(u, v) \in E'$ will have smaller weight than any other edge $(s, v) \notin E'$. By above argument, $(u, v) \in E'$ will be selected and u will be the parent of v with a cost of 0. Second implication is that all edges that are used in spanning graph will contribute 0 total weight, because all weight values are assigned to be 0. Therefore, $(V, E' \bigcup E^*)$ will have the same total weights as (V, E^*). Since proposed algorithm finds the total weights in $(V, E' \bigcup E^*)$, it also computes the total weights in (V, E^*). This concludes the proof.