Editorial for CCC '23 S4 - Minimum Cost Roads

Remember to use this editorial **only** when stuck, and **not to copy-paste code from it**. Please be respectful to the problem author and editorialist.
Submitting an official solution before solving the problem yourself is a bannable offence.

Author: **Plasmatic**

Subtask 1

This subtask can be solved by running an MST (Minimum Spanning Tree) algorithm that only considers the cost of each edge. Note that we may have multiple components, so our final answer will be the sum of costs of the MST of each component. Luckily, an algorithm like Kruskal's handles this problem without any additional complexity.

Subtask 2

The constraints of this subtask allow us to focus on an invariant that is very useful in the long run. Consider a single edge $e = (a, b, l, c)$. We get the following cases:

- If e is the unique shortest path from a to b, we definitely want to take e;
- If there exists a path from a to b with length $l' < l$, we definitely don't want to take e;
- If there exists another path from a to b with length $l' = l$ (but no paths that are shorter), we still don't want to take e. However, this time the proof is much more complex:

  Suppose p is a path from a to b consisting of edges $\{p_1, p_2, \ldots, p_k\}$ such that

  $$\sum_{i=1}^{k-1} \text{length}(p_i) = l$$

  Now, suppose that $\forall i$, p_i is the unique shortest path between its endpoints $a(p_i)$ and $b(p_i)$. If this is not the case for some i, then we can find another path from $a(p_i)$ to $b(p_i)$ with length exactly $\text{length}(p_i)$ and replace p_i with that path (if that path had length $< \text{length}(p_i)$ then we would have a path from a to b with length $< l$, which is a contradiction). Observe that this process can only be repeated a finite number of times.

  In the end, we have a path from a to b that has length l, does not take the edge e, and every edge in that path is the unique shortest path between its endpoints. We'll call the edges on this path $q_1, \ldots, q_m$.

  Now consider some optimal answer that contains the edge e. Clearly, $q_1, \ldots, q_m$ must be in the answer. However, this means that we can remove e as any time we need to traverse from a to b, we can take the edges $q_1, \ldots, q_m$ instead. This violates the optimality of the answer and is thus a contradiction.

This results in a very simple solution for this subtask: we loop over every edge $e = (a, b, l, c)$ and check whether it's the unique shortest path between a and b. One way to do this is to first run Dijkstra's algorithm n times to find the shortest path between every pair of nodes (and store it in a 2-D array $dist$). Then, for each edge we check if

Remark: Notice that we don't even care about costs at all! In fact, the original problem didn't have costs ($l_i = c_i$), but we thought it was an interesting extension :)

Subtask 3

Unfortunately, we lose the guarantees that allow our solution from subtask 2 to work. Fortunately, we can restore these guarantees with some clever reductions.

First, let's look at the graph formed by all of the 0-length edges. If any pair of nodes (a, b) are connected in this graph, then our answer must contain a path of length 0 between a and b. Thus, we must choose a subset of the 0-length edges that has the same connectivity properties as the original graph (i.e. if (a, b) are connected by 0-length edges in the original graph, then they must be connected by 0-length edges in our answer). Trying to do so while also minimizing cost is analogous to finding the minimum spanning forest of the graph, which is the same as our solution in subtask 1.

Unfortunately, the 0-length edges are still part of our graph. In order to remove them for good, we must make the following observation: we can treat each component in our spanning forest as a single node as we're able to travel between any two nodes in that component with a path of length 0. This motivates us to compress our initial graph by these components to remove all 0-length edges.

Note: Be careful, the compressed graph may also contain self-edges so make sure to ignore them.

Next, to remove multiedges notice that we only ever want at most one edge between any pair of nodes. Thus, we take the one with lowest length, breaking ties by cost.

Finally, we run our solution from subtask 2 and sum the answer we obtained from that solution with the one from computing the minimum spanning forest.

Alternative Solution

There is also an alternative solution much closer to Kruskal's algorithm for computing MST: we loop over the edges in order of length, breaking ties by cost and adding it to the graph if it improves the shortest path between its endpoints. This solution also has the added benefit of not worrying about 0-length edges or multi-edges separately.

Comments

There are no comments at the moment.