Kick Start 2020 - Round E

Analysis: Golden Stone

The given set of junctions and streets can be considered as an undirected graph. The problem statement guarantees that the graph is connected.

We can consider all possible combinations of junctions and stones, and try to figure out the optimal cost of making that stone at that junction. Let's define the cost of gathering stone *c* at junction *j* as $Cost_{j,\ c}$. Then the solution would be the minimum of $Cost_{junction,\ golden}$ over all the junctions.

Test set 1

We can consider a new graph, where each {junction, stone type} combination is a vertex, and for each street {u,v} in the given input, there will be **S** new edges {(u,c), (v,c)}, one for each stone type c. Then we can run a variation of Bellman-Ford algorithm on this graph to fill the two dimensional cost table as mentioned above.

Firstly, for all the stone types that are directly available at some junctions (as given in the input), the $Cost_{junction,\ stone\_type}$ would be 0. All other combinations of junction and stone type will have initial cost set to infinity.

Then we can try to relax the cost as in the classic Bellman-Ford algorithm, with some modification, that, in addition to relaxing edges, we also try to reduce cost by applying recipes on the stones that are available at the same junction at some point.

For each junction *u*, we can relax the cost table as follows:

- For each edge *{v, u}* in the input graph, and stone type *c*, try to reduce the $Cost_{u,c}$ by carrying one type *c* stone from *v* to *u*.
- Try to apply each recipe at the junction *u* to see if we can improve the $Cost_{u,\ c}$ where *c* is the stone produced by the recipe.

We must repeat the relaxation steps mentioned above for each junction as long as there are any improvements made to the cost values. It can be proved that no more than **N** × **S** iterations are required.

For the first type of relaxation, each {edge, stone type} combination is considered only once, and then the whole process is repeated O(**N** × **S**). So, ther overall complexity is O(**N** × **M** × $\mathbf{S}^2$).

For the second type of relaxation, the process takes O(**R**) time per junction, and the whole relaxation process will repeat O(**N** × **S**) times per junction. So the complexity is O($\mathbf{N}^2$ × **S** × **R**).

So, the total complexity of this approach is O(**N** × **S** × ((**M** × **S**) + (**N** × **R**))).

Test set 2

Filling up the cost table can also be achieved with Dijkstra's algorithm. The cost table can be initialized in the same approach used in test set 1. Then the vertices with cost 0 are put into a minimum priority queue with the calculated cost as the key.

At each step of trying to reduce cost for a vertex {*u*, *c*} from the queue, we can do the following:

- For each edge {*u*, *v*}, try to reduce the $\text{Cost}_{v,\ c}$ by carrying one type *c* stone from *u* to *v*.
- Try to reduce the $\text{Cost}_{u,\ \text{stone\_type}}$ by applying recipes where *c* is an ingredient.

In this approach, there are **N** × **S** vertices. For each edge in the input graph there will be corresponding **S** edges, so in total **M** × **S** edges.

The relaxation of the first type follows classic Dijkstra's algorithm's style, so the complexity is O((**N** × **S** + **M** × **S**) × log(**N** × **S**)).

For the relaxation of the second type for each vertex {*u*, *c*}, we need to try only the recipes where *c* is an ingredient. So in total, for each junction, a recipe will be applied once for each of its ingredients. Each recipe has at most 3 ingredients. Which gives us additional O(**R** × **N** × log(**N** × **S**)) complexity. So the total time complexity of this approach is O((**N** × **S** + **M** × **S** + **R** × **N**) × log(**N** × **S**)) which is sufficient for test set 2.