FINAL EXAM 2014

DAY TWO

Algorithms description

Let's construct a new graph where the nodes are the roads from the task. When adding a new city, two new roads are created. In our new graph, let's connect each of these roads with the old road with which they form a triangle. We can notice that the resulting graph is a tree.

We will display the shortest path between two nodes with four real numbers, each one marking the distance between each pair of ending city roads that are represented by the initial nodes. When searching for a shortest path between two cities, we need to take either one of two new roads created when adding the first city and each road created when adding the second city and then find the shortest road between the corresponding nodes in the new graph.

In order to find the shortest path between two arbitrary nodes in our tree, we will calculate and memorize the shortest path between each node and its 2^ith ancestor. Now, in order to find the shortest path between two nodes, we need to find their lowest common ancestor (LCA) – we are certain that it is contained in the shortest path.

By using previously calculated shortest paths to ancestors, we can calculated the shortest path from each node to LCA in $O(\log N)$ steps and connect these paths. The time and memory complexity of the solution is $O(N \log N)$.

For implementation details, consult the source code.

For additional information about this type of graph:

- `http://en.wikipedia.org/wiki/Partial_k-tree`
- `http://en.wikipedia.org/wiki/Tree_decomposition`

In order to solve this task, we need to be able to efficiently determine which rectangle from the set is a certain point located in and efficiently add and delete rectangles from the set.

We can notice right away that, because of the special position of the rectangle, these operations can be reduced to two simpler ones: a query about the first length below (to the left of) some point and adding a new horizontal (vertical) length to the set.

These queries can be reduces to two operations on an array of V sets (where V is the largest possible coordinate value):

- add the number P to all sets from set A to set B
- answer what the largest number smaller than Q is in the set, located on the ith position of the array

This problem can be efficiently solved using a tournament tree that remembers in each node a balanced binary tree (STL set) of all the numbers that belong in the interval which it covers.

As for the operation of adding a number, we will add a number in $O(\log V)$ sets. The complexity of this operation is $O(\log V)$, which means that the complexity of adding a length is $O(\log^2 V)$. Also, when making a query about the largest number smaller than Q, we will make a *lower_ bound* query on $O(\log V)$ sets so the query complexity is, again, $O(\log^2 V)$.

In total, we have $O(N)$ queries, so the complexity of this solution is $O(N \log^2 V)$.

For implementation details, consult the source code.