Analysis of task Connect

Subtask 1. For each query, we will first check if there is a path in both directions with *DFS*. If not, we simply answer with 0. Otherwise, we try to remove each of sthe edges and run *DFS* again in both directions. If none of the edges disconnects the two routes, we answer with M+1.

Complexity – **O(Q × M × (N+M))**.

Subtask 2. Here we have the classic problem of determining whether there is a path in both directions between two given vertices, which translates to whether two vertices are in the same strongly connected component (SCC for short). Using the linear algorithm, we can easily find in which SCC each vertex falls and answer queries in constant time.

Complexity – **O(N + M + Q)**.

Subtask 3. From now on, we will call a strong bridge an edge, which, if removed, increases the number of strongly connected components. Obviously, if an edge is not a strong bridge, removing it will have no effect on the routes between any pair of vertices (the SCCs will not change). To answer a query, it is enough to know two things:

- Whether the two vertices are in different SCCs in the begining - answer 0.
- Whether there is a strong bridge, which disconnects the two vertices in two different SCC - answer the corresponding number of the edge.
- None of the above - answer **M**+1.

Let's take a look at how we could answer the second question without checking it every time for each call. We don't know which edges might be strong bridges, so we'll try to remove each one. After each removal, we determine the SCCs and which of them each vertex belongs to. Thus, we get a table with **N** rows and **M** columns, where each row shows the SCCs a given vertex falls in when removing each edge. Now, to answer the second question, it is enough to find the first column in which two rows of this table differ.

Complexity – **O(M × (N+M) + Q × N)**.

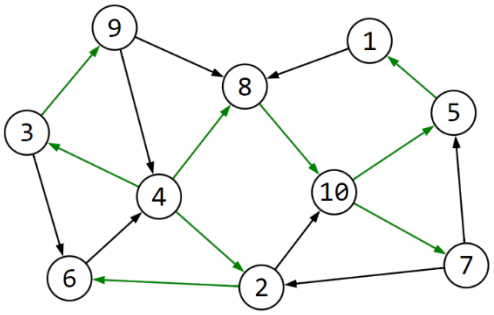
Subtask 4. Due to the larger number of edges, we need to somehow speed up the construction of the table. Note that the edges p → p+1 and p+1 → p for each $1 \leq p \leq N-1$ alone make the entire graph strongly connected. However, if they are sufficient to bring all of the vertices into one and the same SCC, could any of the remaining edges be a strong bridge? The answer is no, because no matter which one we remove, the guaranteed $2 \times (N-1)$ edges in this subtask keep the entire graph strongly connected. Thus, only the edges between vertices with consecutive numbers remain candidates for a strong bridge. The only exception are the multiedges, which obviously cannot be strong bridges.

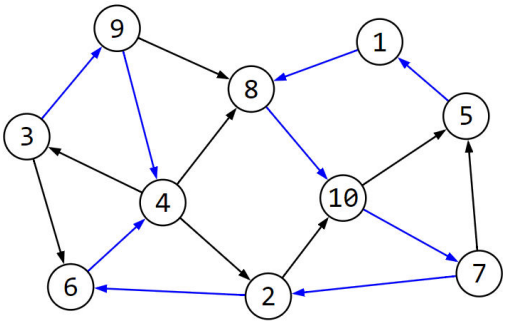
Complexity – **O(N × (N+M) + Q × N)**.

Subtask 5. The solution of the previous subtask leads us to the question of whether really a lot of the edges can be strong bridges. We don't have any edges here to guarantee the existence of any SCC, but nothing stops us from trying to find some. Let us have the SCCs of the original graph. The edges outside of them, by definition, cannot be strong bridges.

One of the main properties of a SCC is that every vertex can reach all the others in it. Therefore, we can construct a spanning tree starting at an arbitrary vertex where all the edges point from the root to the leaves.

In the illustration to the right, the green edges form an example of such a spanning tree with root vertex 4. It can be found by a simple DFS traversal, coloring in green the edges leading to an unvisited vertex.





The other main and analogous property of a SCC is that every vertex is reachable from all the others. Therefore, we can construct a spanning tree from an arbitrary vertex in which all edges point from the leaves to the root.

In the illustration on the left, the blue edges form an example of such a spanning tree, rooted again at vertex 4. It can be found by *DFS* on the opposite edges of the graph, coloring in blue the edges leading to an unvisited vertex.

Consider the union of these two spanning trees. It turns out that they form a "spanning strongly connected graph". Why is this so? Let's take two arbitrary vertices, u and v. From u, we can get to the root using the second spanning tree, and from there to v using the first. Similarly, there is a path in the opposite direction from v to u. Therefore, the two vertices are in one SCC.

Similar to the previous subtask, all the edges that do not participate in these two trees cannot be strong bridges - the SCC stays connected. In the worst case, the two trees have no edges in common, which also gives the maximum number of candidates for strong bridges: $2 \times N - 2$.

Complexity – **O(N × (N+M) + Q × N)**.

Subtask 6. The only thing that remains is to optimize the algorithm for finding a column in which two given rows of the table differ. With precompute, we can calculate the answers for every pair of rows, but that would be $\mathbf{N}^3$ steps, which is too slow. At such times, the standard thing is to sort, and in this case we will sort rows lexicographically. If we denote with $differ[i][j]$ the first column in which rows i and j differ, the following inequality holds: $differ[i-1][j] \leq differ[i][j] \geq differ[i][j+1]$. If we assume the opposite - $differ[i][j] < differ[i][j+1]$, it will turn out that rows i and $j+1$ have a larger common prefix than rows i and j, which contradicts the lexicographical order. So with two pointers we can precompute the answers much faster.

Complexity – **O(M × (N+M) + N × N × log(N) + Q)**.

Subtask 7. We combine the solutions of the fifth and sixth sutasks.

Author: Alexander Gatev