Problem C. Bounded Spanning Tree

Authors: Ihor Barenblat, Matvii Aslandukov
Developer: Ihor Barenblat
Editorialist: Ihor Barenblat

Subtask 1. For each edge, its weight is known. There is need to check that edges can have different weights from the range $[1, m]$ and that edges with indices $1, 2, \ldots, n-1$ form a minimum spanning tree of the given graph. The last can be done using any suitable algorithm. Time complexity: $O(m \cdot log(m))$ or $O(m)$.

Subtask 2. Just try all possible assignments for the weights of the edges and for each one check compliance with the rules described in the statement. Time complexity: $O(m! \cdot m)$.

Subtask 3. Use dynamic programming on subsets of the edges. Lets denote $dp[S]$ as a boolean variable indicating that there is assignment of the weights $1, 2, \ldots, |S|$ to the edges from S, such that there is no rules were broken. Here by "breaking mst rule" we mean assigning to a *non-spanning-tree* edge such value that this edge will be taken into minimum spanning tree when considering it in the process of execution of Kruskal's algorithm. Time complexity: $O(2^m \cdot m)$.

Subtask 4. The same idea as in the `Subtask 5`, but with worse algorithm complexity. Also possible to solve using "matching in bipartite graph" approach. Time complexity: $O(m^3)$ or $O(m^2)$.

Subtask 5. We see that for this subtask there is no "mst rule". So we just need to find an assignment of the edge weights using integers from the range $[1, m]$. It is possible to do this using greedy approach: set 1 to the edge that have $l_i = 1$ with the minimum possible r_i. This is true, since any valid assignment can be changed without losing validity to be compatible with the mentioned greedy assignment. So the overall greedy algorithm is the following: for each i from 1 to m set i as edge weight for the edge with $l_j \le i$ with the minimum possible r_j, between all edges with unset weight. Time complexity: $O(m \cdot log(m))$.

Subtask 6. Lets call the *non-spanning-tree* edge as "special". After the assigning a value for the special edge we need to be sure that there exist possibility of assigning for *spanning-tree* edges weights from the range $[1, m]$ (except the already assigned one). Lets find all suitable possible values that satisfy this condition, choose the greatest one from the range of the special edge and update r_j for the edges on the cycle to be sure that they are smaller that the value we assigned to the special edge. For finding all suitable possible values that satisfy mentioned condition we can use Hall's marriage theorem on the bipartite graph (first part consists of vertices corresponding to the edges and second part consists of vertices corresponding to possible edge weights): value k is suitable if and only if there is no L, R such that $L \le k \le R$ and $R - L + 1 \le (\#\ i$ such that $1 \le i \le (n-1)$ and $L \le l_i \le r_i \le R)$. Time complexity: $O(m \cdot log(m))$.

Subtask 7. Lets call pair of edges (e_{tree}, e_{non_tree}) `interesting` (here e_{tree} is a *spanning-tree* edge and e_{non_tree} is a *non-spanning-tree* edge) if e_{tree} lies on the simple path in the expected minimum spanning tree between vertices that connects e_{non_tree}. It is easy to see that for each interesting pair of edges we can do the following update: $e_{tree}.r = min(e_{tree}.r, e_{non\_tree}.r - 1)$, $e_{non\_tree}.l = max(e_{non\_tree}.l, e_{tree}.l + 1)$. With such modifications, any assignment found by the greedy algorithm from the `Subtask 5` will satisfy "mst rule". Time complexity: $O(m \cdot n)$.

Subtask 8. The same idea as in the `Subtask 7`, but with better algorithm complexity. You can use segment tree to do mentioned updates. Time complexity: $O(m \cdot log(n))$.

Subtask 9. The same idea as in the `Subtask 7`, but with better algorithm complexity. You can use heavy-light decomposition to do mentioned updates. Time complexity: $O(m \cdot log^2(n))$.

Subtask 10. The same idea as in the `Subtask 7`, but with better algorithm complexity. You can use binary lifting approach or non-trivial heavy-light decomposition approach to do mentioned updates. Time complexity: $O(m \cdot log(n))$.