Day 3 editorial

Ivan Smirnov
ifsmirnov@yandex.ru

September 28, 2018

A. Garland Checking

Solution 1 Use link/cut tree, ~~Luke~~ Anakin!

Solution 2 Maintain the structure directly, storing a set of neighbours for each vertex. Also store the color (some identifier of a connected component) of each vertex explicitly. To join two components you should recolor the smaller one (in terms of number of vertices) to the color of larger one. To split two components you should run a BFS/DFS through a smaller component and recolor it to some new color.

Each vertex will be visited $O(\log n)$ times. Reasoning is roughly the same as in the HLD. For join, each time the vertex is visited the size of its connected component doubles. Reasoning for split is the same.

How to run a BFS only over smaller component? Run BFS simultaneously in both parts, making equal number of steps on both sides, and terminate when one part is visited.

B. Desert Game

Heavy-light decomposition with treaps/splay trees.

To reverse a path, find all related elementary subpaths (parts of preferred paths), merge them into a single treap, reverse it and split it back into the trees of corresponding sizes.

Bonus: solve this problem with a link/cut tree. It may be complicated because we strongly rely on static structure of the HLD in the solution above.

C. Galilei

- The vertex is *special* if we have ever checked out at it, or commited it, or it is the lower vertex of the rebased path.
- All vertices with degree > 1 are special. We call them *branching* vertices.
- For all special vertices store their closest branching ancestor and the distance to it.

- Store the depth of each special vertex.
- Store an euler tour tree over special vertices in a treap. That allows taking LCA via range minimum query (LCA of u and v is the vertex with minimum depth between u and v).
- Now for $rebase(u, v)$ we know the LCA of u and v and the distance from v to the LCA.

D. For Fun and Lulz

Solution 1 Use link/cut tree! Store current MST in a link/cut with weighted edges. To update the graph with some edge (u, v), find the largest edge on path (u, v). If it is larger than the new edge, cut it and link u and v with the new edge.

Solution 2 Random graphs have small diameters (that can be checked experimentally), so there are not that many edges on the path between (u, v). Store the pointer to parent in each vertex. To find a path between u and v, go from u to the root and from v to the root, effectively finding their LCA in $O(diameter)$.

E. Link Cut Digraph

First, note that there is no "cut" in the problem. Only "link".

For each directed edge (u, v) we want to know when it is contracted (that is, the moment of time when vertices u and v start being in the same strongly connected component). Divide-and-conquer technique helps. It is very similar to the "dynamic connectivity offline" problem. Make a function $go(l, r, E)$ which processes all edges from set E knowing that they are contracted at some moment between l and r. To do a transition, set $m = (l + r)/2$ and take edges from E that appear before m. Make a graph condensation on these edges. If the edge is contracted, put it into the set E_1, else into E_2, and run $go(l, m, E_1)$ and $go(m, r, E_2)$.

When we know the contraction time for each edge, we may treat the graph as an undirected one. Now process edges one by one, store connected components and their sizes in a disjoint set union.