Code Jam 2017 - Round 3

Analysis: Good News and Bad News

Good News And Bad News: Analysis

The ordered pairs of friends form a directed graph in which each friend is a node and each news path is an edge. Notice that we can work on one connected component at a time, because each can be solved independently of the others. In the following, we assume a connected graph.

One additional observation is that if any edge of the undirected graph is a bridge — that is, an edge not present in any cycle — then the task is impossible. This is a consequence of the news balance on nodes extending to subsets of nodes: if we group nodes on each side of the bridge, the aggregate balance of each subset must be zero, and values in edges internal to the subset don't affect that balance. Since the bridge is the only external edge, this forces its value to be zero. Formally, the news balance property is preserved by vertex identification, and we can identify all vertices on each side of the bridge to reach a contradiction.

Note that since we can pass positive or negative news on each edge, we can forget about the directions of edges in our graph, and consider it as undirected. If the directed version had opposing edges (v, w) and (w, v), the undirected version can just have two edges between v and w. After solving the problem on the undirected graph, we simply adjust the sign of the value we assign to each edge according to its original direction.

Small dataset

The Small dataset can have up to 4 friends, so there can be as many as 12 edges. The bad news is that with up to 2 Ã— 4^2 = 32 possible legal values for each of those edges, there can be 32^{12} possible answers to check, which is far too large to handle with brute force. The good news is that there are only on the order of 2^{12} graphs to consider; we can rule out the ones that have bridges (since those are impossible), and then run whatever method we come up with on all of the others to see if we get solutions.

What if we don't really need the full extent of the [-16, 16] range? We can try restricting our values to a smaller set and see whether we get solutions. As Sample Case #5 demonstrates, we might need negative values, so we can try a set like {-2, -1, 1, 2, 3}, for instance. It turns out that this set alone is enough to solve all possible Small cases that do not contain a bridge, so we can use brute force to find all possible assignments.

Large dataset

Judging by the Small solution, we may believe that all bridgeless graphs do have a solution, and that's indeed the case (as we will see in Solution 2, below).

Solution 1. Start by using depth-first search to build a spanning tree of the underlying undirected graph. Assign all non-tree edges a random nonzero value in [-K, K]. Now, the current balance of each node is some integer. In leaf-to-root order, we can go through all nodes using the single non-assigned edge remaining to make the balance zero. Notice that if we fix all **F**-1 non-root nodes, the root is automatically fixed, because the sum of the balance over all nodes always equals 0 (an edge with value v contributes v to one balance and -v to another). Notice that it is possible that a node's balance is 0 when we need to fix it, leaving us no non-zero choice for the corresponding tree edge. Additionally, if the balance on a node is too high, the value we need for the corresponding edge could be out of range. However, a value for K around **F** yields a

small probability of both, which we can find through experimentation. And, if we happen to run into a problem, we can re-run the algorithm. As long as the probability of success is a not-too-small constant, running it enough times will eventually yield a valid assignment. Since we can implement this in linear time, and the bounds are small, we can afford thousands of runs per test case, or possibly more. So, even with weak experimental data on the probabilities, we can build enough confidence that this method works. Notice that we can avoid explicitly checking for bridges by assuming that failure after 1000 or so retries means the case is impossible.

Solution 2. Again, find a DFS spanning tree of the graph and number the nodes in discovery order. Remember that in a DFS over an undirected graph, every non-tree edge goes from a node to some ancestor in the tree. Direct all edges in root-to-leaf direction (we reverse or split edges after solving, as explained above). We assign edges not in the tree a value of 1, that is, they send positive news from nodes to descendants. On the other hand, we assign negative values to tree edges. We process the nodes in reverse discovery order: as in the previous randomized solution, each node has a single adjacent edge with no assigned value, so we assign it the only value that makes the node balanced.

Let us prove that the values assigned this way are all negative if there is no bridge or contain a zero if there is a bridge: assume a value of 0 for unassigned edges to check the current balance of a node x right before we attempt to balance it. Let A be the set of descendants of x, including x itself. There is exactly one tree edge going into A: the edge that we are trying to assign at this step. Let us call it e. All other edges going into A are non-tree, so they all have a value of 1. The current aggregate balance of A then equals the number of non-tree edges going into A. Since nodes are processed in leaf-to-root order, all nodes in A other than x are balanced, and A's balance equals x's balance. If the number is zero, then there are no edges other than e going into A, which makes e a bridge. If the number is greater than 0 (it is a number of edges, so it can't be negative), the current balance of x is greater than 0 and we can assign its opposite, a negative number, to e. This value is always within the legal [-$\mathbf{F}^2$, $\mathbf{F}^2$] range, because its magnitude is equal to some number of edges in the graph, and the total number of edges in the graph cannot exceed **F** × (**F** - 1), which is less than $\mathbf{F}^2$.

Solution 3. We keep the nodes balanced at all times. Start with 0 on all edges. For each undirected edge, if it has a 0 in the current valuation, find a cycle that contains it and then add -K to all edges of the cycle, where K is an integer different from the current values of all edges in the cycle. This choice of K guarantees that the procedure doesn't change a non-zero edge into zero, and it always changes at least one zero edge into a non-zero (the current edge), so it always finishes. Adding a value to edges in a cycle maintains the balance of each node. If we choose K with minimum possible absolute value, and cycles as short as possible, this won't exceed the range. The easy to prove maximum absolute value this can produce is **F** × **P** / 2, because K is always in the range [-**F**/2, **F**/2] as there can't be more than **F** forbidden values for it. There are also at most **P** steps. However, there are many results showing that graphs with many edges have lots of short cycles, so we believe it's impossible to construct a case in which the range is exceeded. Further, if we randomize the order to fix edges and retry many times, we can work around the worst cases that specifically try to overload a given edge, but possibly produce lots of small cycles that help fix the others cheaply. Notice that the idea of using the DFS tree to fix balances in the previous solutions is actually the same idea from this theoretically simpler solution, but using the fundamental cycles of the graph instead of potentially all of them.