Central European Olympiad in Informatics 2017

July 2017, Ljubljana, Slovenia

Problem Solutions - Day 1

One-Way Streets

Tomaž Hočevar

The problem is asking whether individual edges in a directed version of the given graph must have a fixed orientation to maintain reachability between the given pairs of nodes.

Cycles. Consider a cycle in the graph. We can direct all the edges to form a directed cycle in one or the other direction. This way all nodes are reachable from each other within a cycle. Compress the cycle into a single node and repeat the process until there are no more cycles left and we're left with a forest. We could start with any cycle in any direction, therefore the answer for all compressed edges is B. Because there is a unique path between a pair of nodes (if they are part of the same component) these edges will have a fixed orientation. Just direct the edges along the path from the start to destination. This solves the problem but not efficiently enough.

Bridges. After compressing the cycles we're left with a tree. The edges in this tree are the bridges of the original graph, which we can find in linear time with Tarjan's algorithm. This solves the second subproblem.

Lowest Common Ancestor. The only source of inefficiency stems from directing the edges in a tree. Going over every edge in every path takes too much time and we might direct the same edge several times. Can we avoid this? One approach is to process the paths in a helpful order. Let's root the tree at some node and process it in $O(n \log n)$ for answering lowest common ancestor queries in $O(\log n)$. We can split every path between two nodes in two parts — one going up the tree and the other one going down — and then solve the problem for paths going from a node towards the root. We will process the paths by their lowest common ancestors starting with those nearest the root. When directing the edges on a path towards the root we might at some point reach an already directed edge and all edges from there on will already be directed in the same direction. Because we are guaranteed a solution exists this direction will also be the correct one otherwise we would have a contradiction.

Sure Bet

Tomaž Hočevar

The problem is asking us to maximize the value $\min(\sum a_i - n_a - n_b, \sum b_j - n_a - n_b)$, where a_i represent the odds for placed bets on the first outcome and n_a the number of them, while b_j and n_b stand for the corresponding values for the second outcome.

Brute force. The straight-forward way to solve the problem is to simply try all possible subsets of bets. There is 2^{2n} of them, which is small enough for $n \leq 10$ and solves the first subproblem.

Greedy. Let's simplify the formula we're trying to maximize. If we subtract 1 from every quota, the formula simplifies to $\min(\sum a_i - n_b, \sum b_j - n_a)$. Note that the two outcomes are independent for a fixed n_a and n_b — the exact set of odds we choose for the first outcome does not affect the optimal choice of bets for the second outcome. Therefore, it makes sense to sort the values a_i and b_i. We can choose the n_a largest a_i and n_b largest values b_i for every pair of n_a and n_b. This gives us a $O(n^2)$ solution for the second subproblem.

Ternary search. Consider what is the score for a fixed value of n_a and different values of n_b. As we increase n_b the score grows until the term $\sum b_j - n_a$ becomes greater than $\sum a_i - n_b$ at which point it starts to decrease. Therefore, we can use ternary search to find its maximum or a binary search over the list of differences in score for consecutive values of n_b. Precompute prefix sums to compute the score for a given pair n_a and n_b in $O(1)$. Time complexity of this solution is $O(n \log n)$.

Linear solution. There is in fact a linear solution assuming the values are already sorted. We can try to visualize what is going on as we try different values of n_a and n_b. For every a_i that we bet on, we increase the first value in the minimum by a_i and decrease the other one by 1, while trying to maintain the lower one as large as possible. This would suggest that if n_b is the optimal choice for n_a then as we increase n_a to $n_a + 1$ we should also increase n_b or keep it at the same value but not decrease it.

Let $A = \sum_{n_a} a_i - n_b$ and $B = \sum_{n_b} b_j - n_a$ where n_b is the optimal choice for n_a. As we increase n_a by 1, we get $A' = A + a$ and $B' = B - 1$. We will handle two cases:

1. $A' \geq B'$. Because B' is already the smaller of the two values decreasing n_b would only make the minimum smaller.

2. $A' < B'$. Suppose that decreasing n_b would give us a better solution. The new score in this case $\min(A' + 1, B' - b)$ would have to be larger than the previous one $\min(A', B') = A'$. This would imply $A' < B' - b$. Rewriting it we get $A + 1 < B - b - a$. This means that n_b is not the optimal solution for n_a as we assumed because we could decrease n_b and obtain a better solution for n_a; we have a contradiction.

We don't have to restart the search for n_b from scratch for every value of n_a in increasing order. Instead, we can simply increase n_b until we find the maximum for a given n_a. For $n_a + 1$ we start the search for n_b from where we finished previously, overall making a single pass over all values of n_b.

Mousetrap

Vid Kocijan

Exhaustive search. A minimax algorithm with full exhaustive search is a slow solution with no simple implementation. With the following observation, an exhaustive search solution becomes much cleaner and easier to implement. It doesn't make sense to block a passage after cleaning some other passage. Cleaning the passage increases the mouse's movement options, therefore Dumbo would be better off blocking the other passage before cleaning one.

If Dumbo leaves the mouse running, it will eventually get stuck. Before the mouse gets stuck Dumbo will only block passages. Which ones should be blocked is a matter of the exhaustive search. Once the mouse is stuck, Dumbo will block all the side passages leading away from the path to the trap and then clean that path. The mouse will have no choice but to run directly into the trap.

Mouse starts the game next to the trap. Root the tree so that the root corresponds to the trap. What happens if Dumbo just leaves the mouse running until it gets stuck? The mouse will get stuck in one of the leaves, Dumbo will block all the side edges on the path leading from this leaf to the root and then clean the dirty edges in this path.

We can calculate the exact number of moves Dumbo needs to win the game if the mouse is caught in a certain leaf. Let's call this the *weight* of the leaf and denote the weight of a leaf l with w_l. Note that if Dumbo leaves the mouse running around, it will move to the leaf with the largest weight. On the other hand, Dumbo will try to block the mouse from reaching leaves with large weights.

Suppose the mouse is in node v. Because the mouse starts at depth 1, we can safely assume it only moves down the tree towards the leaves. Node v has i children $u_1 \dots u_i$. Without loss of generality we assume that u_1 is the optimal choice for the mouse and u_2 the second best. Dumbo will want to block the passage to the node u_1. Does this effect weights of the leaves in subtrees $u_2 \dots u_i$? Because edge (v, u_1) is a side edge from a path from any leaf in subtrees $u_2 \dots u_i$ to the root, it's already part of their weights and the edge (v, u_1) has to be blocked anyway. So blocking the passage next to the mouse doesn't change the weights of the leaves in the subtrees. Hence, Dumbo will block the best possible child of the node and mouse will traverse the second best.

From here on we can generalize the definition of weights for any node. Weight of a node v is the number of moves Dumbo needs to win if it's his turn and the mouse is currently in node v. Dumbo

will block the edge to the heaviest child and the mouse will traverse the edge to the second heaviest one. Hence, weight of a node v is the weight of second heaviest child or 1 if v only has one child. Weights of nodes can easily be calculated with one tree traversal which takes linear time. The number of moves in the game is equal to the weight of the mouse's starting node.

General case How does the game change in the general case? The mouse doesn't start the game at depth 1, which means it might make a few moves up the tree before going down. Dumbo cannot block upgoing edge, because he would block mouse from reaching the trap. As soon as the mouse makes a move down the tree, the game continues as described in the previous section.

When selecting the next edge to block, Dumbo must therefore not limit himself to outgoing edges of the mouse's current location. Possible candidates are also the side passage on the path from the mouse to the trap.

In the starting position of the game, a path P leads from the mouse to the trap. Dumbo may not block edges in P, so weights of nodes in P are undefined. There are multiple subtrees attached to P. Each of these subtrees (actually their roots) has its weight as defined in the previous section.

Suppose the mouse arrives into a subtree S_i. Any blockade Dumbo made on side edges of P below the depth of S_i do not count towards its weight w_i. Let's say there were B_i such moves, therefore Dumbo has to make $B_i + w_i$ moves in total. What is the minimal $B_i + w_i$ if Dumbo plays optimally?

We can find this by a binary search over the total number of Dumbo's moves. Whether Dumbo can win in at most X moves can be verified by simulating a mouse's run up the path P and testing if all the side edges i with $w_i + B_i > X$ have been blocked.

We can present each subtree S_i with pair $(d(v, S_i), w_i)$, where $d(v, S_i)$ is distance of subtree S_i from node v. Dumbo can only block path to this subtree in first $d(v, S_i)$ rounds, then the mouse reaches it. We sort pairs by first component. Let L be this sorted list of pairs. We will use variable B to count the number of blocked side paths. Initially $B := 0$.

For each pair L_i in L: If $L_i(2) + B > X$ the passage has to be blocked and $B := B + 1$. If at any point $L_i(2) + B > X$ and $B > L_i(1)$ mouse reached a subtree before Dumbo could block the passage to it, which means Dumbo can't win in X moves. Otherwise, we found a way for Dumbo to win in X moves.

Each bisection iteration takes linear time (the list L doesn't change, it can be reused multiple times). Hence, total running time is $O(n \log n)$.