Go 2 Editorial

This task originally appeared as a practice task in the finals of the Czech olympiad in programming in 2021. The CEOI version comes with better subtasks.

General idea

Let us consider a graph whose vertices are all grid points and edges correspond to the matches placed.

When a match is placed, we check if its edge closes a cycle in the graph. If it does, we remove the match and calculate the area enclosed by the cycle. Otherwise we keep the edge.

In subtasks where all coordinates are small numbers, we can represent the graph as a matrix. Otherwise, we create vertices on the fly as they are referenced by the matches and we keep a hash table mapping coordinates to vertex IDs.

Detecting cycles

In small subtasks, we can check if the endpoints of a new edge are already connected by running DFS or BFS. This requires time $\mathcal{O}(N)$ per match.

A better choice is to use the Union-Find data structure (this is hardly a surprise as we are maintaining a spanning forest of a dynamic graph). With a simple implementation with Union by rank, we reach $\mathcal{O}(\log N)$ time per match. We could employ path compression to speed up Union and Find even further, but it does not help much since the average number of digits used to describe a match in the input is roughly $\log N$ anyway.

Computing areas

In subtasks with a small grid, we can try using flood fill (BFS) to calculate the area one grid square at a time.

There is a faster algorithm for calculating the area enclosed by a rectilinear polygon by walking around its boundary. (It is a simplification of a well-known algorithm for computing the area of a non-convex polygon.)

Suppose for a moment that the whole shape lies above the x axis. We ignore vertical edges. For each horizontal edge e on the boundary, we consider a rectangle whose top side is e and the bottom lies on the x axis. We compute the *signed* area of this rectangle: if e was traversed left-to-right, it is positive, otherwise it is negative. When we add all these signed areas, we get the area enclosed by the polygon (possibly with a negative sign).

This is easy to prove: Assume that we walk around the polygon in a direction such that all topmost edges are oriented left-to-right (otherwise, we flip all signs). Consider a unit square. It is contained in rectangles corresponding to all edges *above* this square. Looking from this square up, the directions of these edges alternate and so do the signs of the rectangle areas. If the number of edges is even, the square is outside the polygon and its area was added exactly as many times as it was subtracted. If the number of edges is odd, the area was added once more than it was subtracted (the topmost rectangle is added).

It remains to show that the assumption that the shape lies above the x axis was needed only for clarity: if we shift all y coordinates by the same amount, the area computed by the algorithm does not change.

This algorithm calculates the area in $\mathcal{O}(N)$ time. In the same time, we can find the boundary by constructing the path joining the endpoints of the newly added match.

Precomputation

Linear time per match is still too much for solving the full task. But all we need to add is a little precomputation.

First we observe that when we run the are computation algorithm on a boundary which goes around a tree, the result is obviously zero: Every edge of the tree contributes once positively a once negatively. Similarly, when we connect a walk around a tree to any boundary, the enclosed area stays unchanged.

Now we let the Union-Find part of the algorithm process all matches. We record all matches closing an area and we end up with a forest formed by the remaining matches. We walk around every tree in this forest and compute prefix sums of the signed areas determined by the edges visited. Also, for each vertex we store the position of one of its occurrences in the walk. This takes $\mathcal{O}(N)$ time on the top of $\mathcal{O}(N \log N)$ time needed by Union-Find.

Then we process the closing matches one by one. For each of them, we consider a sub-walk of the walk we preprocessed between the endpoints of the current match. Together with the current match, it forms the boundary of the enclosed area with some extra trees walked around (which we already know not to affect the area). We can therefore compute the area in constant time as a difference of the prefix sums plus the signed area of the rectangle corresponding to the current match. This is $\mathcal{O}(N)$ in total.

Overall, the whole solution runs in $\mathcal{O}(N \log N)$ time.

Notes

There are several possible variants of this task. For example, one might remove all matches on the cycle instead of just the last one. The techniques we developed can be generalized to these variants. Instead of Union-Find, we have to maintain dynamic spanning trees using either Link-Cut trees by Sleator and Tarjan or Eulerian Tour Trees. Both can be used to hold precomputed areas similarly to our static prefix sums.