Code Jam 2018 - Round 3

Analysis: Fence Construction

For both solutions discussed below, we model the situation as a planar graph G with endpoints as nodes and fences as edges. Getting a representation of this graph as an adjacency list takes quasilinear time on the input (because we need to identify endpoints, which may add a logarithmic factor or an expected constant factor depending on whether we use trees, hashing or sorting to do it).

Test set 1

Let us assume for a brief moment that there is no prescribed partial ordering (for instance, when **K** = 1). There are many greedy algorithms that would retrieve a correct ordering of the fences. You can try working the fences from the outside in or from the inside out, cleverly using convex-hull-type algorithms. Each way has its own pros and cons, but many of them also have the property that the reverse of the ordering they produce is also a valid ordering. This is enough to deal with **K** up to 2: retrieve a greedy order. If the partial order is obeyed, return that; otherwise, return the reverse. One of the two is guaranteed to work. Another way that works for some greedy alternatives is that they may allow us to fix a single fence as the first one to be built and then work from there. If you fix fence 1 to be the first one built, the partial order is guaranteed to be respected.

We will describe a different option than the one discussed above, because it is simpler to describe, prove and implement. Build any spanning tree T of G that contains fence 1. Build the fences in T first in any order that starts with fence 1. This guarantees that we respect the partial order. Since we are building fences that close no cycle, there are no restrictions to moving the printer so far: any point not on a fence is reachable.

For each the remaining fences f, consider the cycle that is formed when f is added to T. That cycle represents a simple polygon. If we simply build these remaining fences in increasing order of area of that polygon (breaking ties arbitrarily) we obtain a valid order. The reason is that we can always keep the printer on the "outside" of all closed cycles. Since the next cycle to be closed at any time has area no less than any already closed cycles, it cannot be contained in any of them, which means the fence that is closing it is also on the "outside", and so the printer can get arbitrarily close to the fence's position and find a point from where the fence can be built.

Notice that this algorithm can be implemented in quadratic time. Building a spanning tree takes linear time. For each fence outside of the spanning tree, we take linear time to get the area of the polygon, which leads to quadratic overall. Then, we only need to sort the remaining fences, which can be done in faster-than-quadratic time.

Test set 2

For Test set 2, a bit of theoretical knowledge is required. Build the dual graph of G and call it H. Moreover, let G(S) and H(S) be the graph corresponding to a given set of fences S and its dual, respectively. As usual, when we name faces or nodes of H, we include the face that is on the outside (i.e., the only one with infinite area).

When we finish our work, the printer will be left in one of the faces of G, that is, one of the nodes of H, and the last fence we built will be one of the edges that are adjacent to that face. If we step through the build order backwards, right before building the last fence f, the printer is on a face of G(F - {f}), where F is the set of all the fences. For any set S and any fence f, there is a strong

relationship between G(S) and G(S - {f}), and correspondingly, H(S) and H(S - {f}). G(S - {f}) is the result of removing f, and possibly its endpoints if they become isolated, from G(S). Correspondingly, H(S - {f}) is either equal to H(S) or the result of merging two nodes of H(S) into one: f may have a single adjacent face in G(S) or two, which leads to nothing changing or two nodes merging in H(S), respectively.

Focusing back on G(F - {f}), we know that the next-to-last fence g to be built has to be one of the fences that is adjacent to a face adjacent to f. The one right before that is adjacent to a face adjacent to either f or g, and so on. That means any ordering of the fences is a possible search on the graph X of fences as nodes where two fences are adjacent in X if they are adjacent to the same face of G. A search on a graph is an ordering of the nodes such that any prefix of the ordering is connected. Breadth-first-searches and depth-first-searches are examples of searches, but not the only ones.

Now that we have identified valid orders with searches on a graph, we can greedily build one that respects the partial order. From this point on, we will build the reverse of the output. Let us fix the first fence of our order f. From here, we can keep track of the reached fences. Any reached fence that is not in the partial order can be added immediately to our order, as it only adds more reachable fences and it cannot violate the partial order. Fences that are in the partial order, however, need to be added respecting that, which we can also do greedily. If at some point all the reachable fences are in the partial order but neither is the next one, we have failed and we need to choose a different f.

The explanations above lead to a quadratic algorithm: building G and then H takes quasilinear time. Building X from H takes quadratic time (since it may have a quadratic number of edges). The greedy algorithm takes constant time for each fence it adds to the order, and we may need to try a linear number of starting fences f, so that time is quadratic overall.

The algorithm can actually be improved to quasilinear overall: there is no need to build X explicitly. We can use X implicitly from H and only inspect adjacencies to fences that haven't been reached. That removes the quadratic cost of explicitly building X. As for trying every f, we can prove that fixing f to be the first fence in the partial order (that is, fence **K**, since we are building a reverse order) is always optimal. For a valid search of X that respects the partial order, reversing the prefix that ends on fence **K** is also a valid search that respects the partial order, and has fence **K** first.