Code Jam 2013 - Qualification Round

Analysis: Treasure

Lexicographically Smallest?

This problem asks you to open the chests in the lexicographically smallest order. As if it's not hard enough to find just any old way to open the chests, we make you find a particular way! If you have not tried similar problems before, this part might seem especially daunting.

However, it's a red herring. Suppose you can answer the following simpler question: "Is it possible at all to open all the chests?". Then you can use that as a black box to find the lexicographically smallest way of opening them. Check whether you have the key to open chest #1 first, and whether the black box says it is possible to open all the remaining chests after doing so. If the answer to both questions is yes, then you should definitely start by opening chest #1. Otherwise, you have no choice but to try a different chest. Repeating this logic for every chest at every time, you can get the lexicographically smallest solution.

This is not very fast, but there aren't too many chests to worry about here, so that's okay. And that means, from now on, we can focus on the slightly simpler question: "Is it possible to open all the chests?".

Eulerian Path

Unfortunately, even this question is still pretty hard!

When dealing with a really tricky problem, it can sometimes be helpful to look at special cases to build up your intuition. Towards that end, let's suppose that you begin with exactly one key, that one chest is empty, and that all remaining chests also contain exactly one key. Why look at this case in particular? You'll see!

In this version of the problem, you will always have exactly one key at any given time. When you open a chest of type A containing a key of type B, you are switching from a key of type A to a key of type B. This suggests that we can represent the problem as a graph. We will have one vertex for each chest/key type, and for every chest, we will add a directed edge from the chest type to the contained key type.

You begin with a single key, and then you must choose a chest of the matching type, giving you a key of perhaps a different type. From there, you must choose a chest of the new type, and so on, eventually choosing every chest exactly once. In the graph formulation, you can think of this as beginning at the vertex corresponding to your starting key, and then repeatedly following edges, using each edge exactly once. In other words, you are looking for an Eulerian Path on the graph!

The good news here is that Eulerian Path is a famous problem and you can look up on the internet how to tell if one exists:

- At most one vertex (namely the start vertex) has OutDegree - InDegree = 1.
- At most one vertex has InDegree - OutDegree = 1.
- All other vertices have InDegree = OutDegree.
- There is a path from the start vertex to every other vertex in the graph.

The bad news is that if this special case is already as hard as Eulerian Path, the full problem is going to be even harder!

Generalizing the Eulerian Path

If you come up with the previous observation, the first thing you might try is to reduce the full problem directly to Eulerian paths. Unfortunately, this is probably doomed to fail. Once you have multiple keys in a single chest, there is no graph structure to use. (Nothing we have found anyway!)

The better plan is to generalize only the conditions required for an Eulerian path to exist. In fact, those conditions can be described very naturally in terms of chests and keys:

- For each type, there must be at least as many keys of that type as there are chests of that type.
- It must be possible to get at least one key of any single type.

Let's say a chest/key configuration is *connected* if it satisfies the second condition, and *good* if it satisfies both conditions. It turns out that it is possible to open all the chests if and only if the configuration is good!

And it is not too hard to check if a configuration is good - the first condition is just a count, and the second can be checked with a Breadth First Search or a Depth First Search. So all that remains for a complete solution is convincing ourselves that checking goodness is indeed equivalent to the original problem:

Claim: It is possible to open all the chests if and only if the configuration is good.

Proof: One direction is easy. If the configuration is not good, then we will never be able to open enough chests of one type, either because there are not enough keys, or because we can never reach even one of those keys.

For the other direction, let's suppose we have a good configuration. Nothing we do from this point on will change whether there are enough keys of each type. We will show there is always at least one chest we can open that also maintains the connectivity property. The resulting configuration would then also be good, so there would also be a chest we could open there that would maintain connectivity, and so on. Repeating, we can keep opening chests until there is nothing left.

So all we need to do is prove that there is at least one chest that can be opened without breaking connectivity. We know the configuration is connected initially. For each type T, there is a sequence of chests we can open to get a key of type T. To be precise, here exists a sequence of types T_1, T_2, ..., T_n, with the following properties:

- You already have at least one key of type T_1.
- T_n = T.
- For each i, there is a chest of type T_i which you would be able to open to get a key of type T_{i+1}.

Suppose you have a key of some arbitrary type A. If you already have *all* keys of type A, then you can open all the chests of type A, and doing so will certainly not break connectivity. So that case is easy.

Otherwise, there must exist some chest of type B containing a key of type A. Let T_1, T_2, ..., T_{n-1}=B, T_n=A denote a sequence of key types that you can go through to get a key of type B and then use that to get another key of type A. As mentioned above, we know such a sequence must exist. We can also assume that T_i != A for 1 < i < n. Otherwise, we could just chop off part of the sequence to get a faster method! Now we consider two cases:

- Suppose that T_1 != A. Then you can use your key of type A to open anything you want, and we claim the resulting configuration will still be connected.

  To prove this, pick an arbitrary key type C (possibly equal to A). Before opening any chests, we know there is some sequence of key types S_1, S_2, ..., S_m=C that will give you a key of type C. If A is not part of this list, then opening a chest of type A does not interfere with getting a key of type C. Otherwise, the S list and T list intersect somewhere, so we can let j be the largest integer such that S_j equals some T_i. Then after using our A key, we can still get a key of type C in the following way: T_1, T_2, ..., T_i=S_j, S_{j+1}, ... S_m=C.

  Since this is true for every C, we know the configuration is still connected!

- Suppose that T_1 = A. In this case, you should use your key to open a chest of type T_2. You can now still obtain a key of type B, and so the rest of the argument follows exactly as before.

Therefore, no matter what happens, you can always open a chest without breaking connectivity, and the claim is proven!