Code Jam to I/O for Women 2020 - Code Jam to I/O for Women

Analysis: Imbalance Obviation

Test Set 1

For Test Set 1, we can enumerate all 2^N possible strings of length **N** composed only of `L`s and/or `R`s, and check whether each one would break the scale, by simulating the process explained in the problem statement. We can then output any one string that works; the statement guarantees that at least one exists.

This process has a time complexity of $O(\mathbf{N}\times2^{\mathbf{N}})$ for each test case, which is fast enough for this test set.

Test Set 2

Suppose we've placed our first marble on one of the pans. Then the second marble must go on the other pan. Now we are effectively back in the same situation — the scale is balanced and we have our choice of pan again, but then that choice forces the next choice, and so on.

Similarly, if the number of marbles is even (as is always the case in this test set), the scale is balanced before we start to remove marbles. This means we can remove the first marble from either pan, but the second marble we remove must come from the other one. The third marble can be removed from either one again, but the fourth must be removed from the other one, and so on.

We can formalize the first observation by saying that marbles 2i and 2i-1 must go on different pans, for every i ≥ 1. We can formalize the second observation by saying that marbles $\mathbf{A}_{2i-1}$ and $\mathbf{A}_{2i}$ must go on different pans, for every i ≥ 1.

The rules above help to reduce this problem to graph coloring . In our graph, each node represents a marble, and each pair of marbles that are required to be on different pans is connected by an edge. The resulting graph has **N** nodes and at most **N** edges, and each of the nodes is connected to at most 2 other nodes. If we can 2-color the graph, we can use one of the colors (it doesn't matter which one) to represent the left pan and the other to represent the right pan, and then we will have our desired assignment of marbles to pans. Better yet, 2-coloring takes at most linear time!

How can be sure that a 2-coloring is possible, apart from the assurances in the problem statement that a solution always exists? Well, this is possible if and only if there are no cycles of odd length — can we show that?

The constraints from the marble-adding phase pair up nodes 1 and 2, 3 and 4, etc. Call these the "original pairs". Any additional constraint from the marble-removing phase (that is, from the input permutation **A**) can do one of the following: match an original pair's edge, in which case that pair cannot have other connections to the rest of the graph and is not a cycle, or add an edge that connects two of the original pairs. Then, any path that does not repeat edges alternates between edges connecting original pairs and edges from the marble-removing phase that do not connect original pairs. Therefore, any cycle formed is of even length, or else it would have two edges of the same type one after the other.

So, the resulting graph is bipartite and our strategy will always find a solution.

Test Set 3

Let's reconsider the above strategy and see if we can make it work for odd **N**:

- The first rule ("marbles 2i and 2i-1 must go on different pans, for every i ≥ 1") is still fine — the pans will be balanced right before we place marble **N**, so it does not matter where we put that marble.
- After we remove the marble $\mathbf{A_1}$ (more on that in a moment), the pans will be balanced again, so we need our second and third marbles to be removed from opposite sides, and then our fourth and fifth, and so on. We can replace the second rule above ("marbles $\mathbf{A}_{2i-1}$ and $\mathbf{A}_{2i}$ must go on different pans, for every i ≥ 1") with: marbles $\mathbf{A}_{2i}$ and $\mathbf{A}_{2i+1}$ must go on different pans, for every i ≥ 1. (If we write this as "marbles $\mathbf{A}_{\mathbf{N}-2i+2}$ and $\mathbf{A}_{\mathbf{N}-2i+1}$ must go on different pans, for every i ≥ 1", then the condition is the same as in the even-**N** case.)
- Finally, the first marble removed, $\mathbf{A_1}$, must come from the heaviest pan. We know that marble **N** is in the heaviest pan, so one way to represent this constraint is to replace both $\mathbf{A_1}$ and **N** in the graph with a new "node" that represents them both. We can do this because we know that marble **N** has no constraints arising from the marble-adding phase, and marble $\mathbf{A_1}$ has no constraints arising from the marble-removing phase. So the argument from earlier about no odd cycles still holds; this new "node" can only be entered via one type of constraint and exited via another.

A less complex version of the above idea is: we can introduce an imaginary extra marble that we add last and remove first, making the total number of marbles even again. We can add **N** + 1 to the end of the identity permutation, and to the beginning of the input permutation (since it was the last marble added, it's always safe to remove it first, restoring a previous valid state). Then we can solve the problem as in the even version above, and finally chop off the last character of the sequence. This gives us the same O(**N**) time complexity as in Test Set 2.

A note on 2-SAT

A more abstract view of the same solution is to represent each marble as a Boolean variable (where true corresponds to assignment to one pan and false to assignment to the other pan), and then encode the requirements explained above as two implications of literals, which are in turn disjunctions of literals. The conjunction of all those requirements is a valid input to 2-satisfiability (2-SAT). Notice that the graph created in the typical 2-SAT algorithm is exactly 2 copies of the graph mentioned for the solution above, so the algorithm ends up being the same in both cases.