Code Jam 2017 - World Finals

Analysis: Dice Straight

Dice Straight: Analysis

Small dataset

One brute force approach to the problem is to examine all possible subsets of dice in all possible orders, and look for the longest straights. Since there may be as many as 100 dice in the Small dataset, we need a better strategy.

First, let's create the set of all integers that are present on at least one die, and sort that set in increasing order. Then we will create an interval that initially contains only the first number in that sorted list. We will expand and contract that interval according to the following strategy. As an invariant, the entire interval will always be a straight (a sequence of one or more consecutive numbers).

Check whether it is possible to build the straight using the available dice (we'll get to how to do that in a moment).

- If it is possible: Expand the interval to include the next value to the right.
    - If that value is not one greater than the previous rightmost value, then we no longer have a straight; contract the interval to remove all but that new rightmost value. Then we have a straight again.
- If it is not possible: Contract the interval by removing its leftmost value.

To check whether a possible straight can be built, we can find a bipartite matching from the required integers to the dice by using a flow algorithm such as Ford-Fulkerson. Since each die has exactly 6 faces, the number of edges in the graph is 6**N**, and the running time of one iteration of Ford-Fulkerson is O(**N**^{2}). We run it up to O(**N**) times as we adjust our interval, so this solution is O(**N**^{3}). Other polynomial solutions are possible.

Large dataset

To make the flow algorithm fast enough to solve the Large dataset, we need the additional insight that we do not need to start the algorithm from scratch every time. When expanding by adding a new number, we need to add all O(**N**) edges into that number (either by adding them outright or changing their flow capacity from 0 to 1), and then find and add a single augmenting path to the existing flow, which also takes O(**N**) time (since the total number of edges in the graph is at most 6**N**). So an expansion takes O(**N**) time overall. When contracting, we need to remove edges and flow from the path associated with that number; this takes O(**N**) time. Since we have O(**N**) expansions and contractions, they take O(**N**^{2}) time in total. Adding this to the O(**N**^{2}) from completely solving the flow problem the first time, the overall complexity is still O(**N**^{2}). This is fast enough for the Large dataset.