Code Jam 2020 - Round 1A

Analysis: Pascal Walk

Test set 1

The answer for **N** ≤ 500 can always be constructed by walking along the leftmost positions of the rows of the triangle: (1, 1), (2, 1), ..., (**N**-1, 1), (**N**, 1).

We can handle the case **N** = 501 by making a detour to pick up the 2 in the third row before returning to the left side. That is, our walk takes us through the positions (1, 1), (2, 2), (3, 3), (3, 2), (3, 1), (4, 1), ..., (498, 1).

Test set 2

With only 1000 possible test cases to consider, we can find an answer to each one before submitting. One way to do this is to walk over the triangle in e.g. a breadth-first or random way, taking care never to reuse a position or visit more than 500 positions. We can check our cumulative sum at each position, and whenever we encounter a sum that we have never seen before, we record the sequence of positions that we used to obtain it. When our sum exceeds 1000, we backtrack in our search or start over with a new random path, and so on, until we have an answer for every possible **N**. It's difficult to prove a priori that this will work, but we can be optimistic given that the top few rows of the triangle have many small values to work with. Indeed, this method finds a full set of solutions very quickly.

An alternate method is to observe that the positions immediately to the right of the leftmost positions — named (x, 2) for each x ≥ 2 — are 1, 2, 3, 4, 5, and so on. We can move from the top position in the triangle down to the 1 at position (2, 1), then follow that line to the 2 at (3, 2), the 3 at (4, 3), etc., until our next move would cause the cumulative sum to exceed our target. Then, we can instead make a move to the left to reach a 1 on the left edge of the triangle, and then proceed downward along that edge, taking as many extra 1s as we need. Since the sum of the first 45 natural numbers is over 1000, we can be sure that we will never need to take more than 45 of these extra 1s, or visit more than 45 of the positions along this line of natural numbers. This ensures we never use more than 90 positions in total.

Test set 3

There are various ways to solve the third test set, but one of our favorites takes advantage of a special property of Pascal's triangle: the entries in the r-th row (counting starting from 1) sum to 2^{r-1}. This is to be expected from the way in which each row is constructed from the previous one: each element in a row contributes to two elements in the next row, so the sum doubles with each row.

This observation suggests a strategy for constructing any **N** that we want: write **N** in binary, and then look through that representation, starting from the least significant bit. If the r-th least significant bit (counting starting from 1) is a 1, our path should consume all of the elements in the r-th row. If the r-th least significant bit is a 0, we should skip that row somehow. We only need the first 30 rows of the triangle, since the 31st row's sum is 2^{30}, which is larger than the maximum possible **N** of 10^9. Even if we used every element in these first 30 rows (which we would never need to), that would only use 465 positions, which is within the limit of 500.

This doesn't quite work as written, though, because our path must be continuous, and so there is no way to skip a row. But we can get close via the following variant: proceed down one side of

the triangle (that is, the diagonal line of 1s) as long as we keep encountering 0 bits. Whenever we encounter a 1 bit, we send our path horizontally across the corresponding row, sending us to the other side of the triangle. This almost gets us the number we want, but we probably overshoot because we have consumed some extra 1s from the rows corresponding to 0 bits. We can be sure that there are fewer than 30 of those, though, since we visit at most 30 rows.

So, we can use that variant, but instead of constructing **N**, we construct **N**-30 instead. Once we finish, we can tack on additional 1s from our current edge of the triangle until we reach **N**. (We can handle cases with **N** ≤ 30 specially, and just go down one edge of the triangle.)

There are other less elaborate ways to solve this test set, though! One is to walk down the centers of the rows (e.g., going from (1, 1) to (2, 1) to (3, 2) to (4, 2) to (5, 3)...). This is where the largest numbers live, so we will grow our sum as rapidly as possible. At some point, though, when moving downward would make our sum grow too much, we can instead move to a one step away from the center and continue zigzagging downward, and so on. Eventually we will reach the edge full of 1s, and we can take as many more as we need. We must be careful not to increase the sum so fast that we will not be able to "escape" to the edge of 1s. Nor should we reach the edge too early, since we can only take so many 1s. However, it's not too difficult to get this method to work.