Code Jam 2009 - Round 2

Analysis: A Digging Problem

The setup of this problem, where you can go left, right and down but never up hints that some dynamic programming solution is the way to go, but the details are a bit involved.

When we're on a row we need to know which are the rock cells that were dug previously so that we know how much we can move left or right. This means that we could have a state in our algorithm be (i, j, air_holes) where i is the current row, j is the current column and air_holes is the set of cells on row i that were dug previously or were empty. Filling the values for these states in the whole matrix would take exponential time as air_holes can take on as many as 2^C values. This would be enough to solve the small input, but for the large we need to improve our algorithm a bit.

First let's observe that it only makes sense to dig out cells that will form a connected empty zone. After we fall one row, we'll be able to use just the current zone of empty boxes. Now our state is (i, j, start, end) where start is the starting column index of the current empty zone and end is the index of the zone's end column. This idea yields a polynomial solution, as we have O(R * C^3) possible states and there are at most C^2 different states that we can create on the next row.

But we can improve on this solution. It doesn't make sense to change directions after we started to dig; if we're moving right, it doesn't matter how many empty cells we have on the left. This changes the state to (i, j, dir, count) where dir is the current direction (left or right), and count is the number of empty cells in that direction. This reduces the state space to O(R * C^2).

The implementation details are somehow tricky, and you have to make sure you don't fall more than F steps -- a detail we've skipped here. There are many possible ways to implement this problem, some of which result in much simpler code than others. We encourage you to download and study various correct implementations from the scoreboard.

Before the contest began, we evaluated this problem as the second easiest in the contest; but the many details needed to solve it resulted in this problem having the second-smallest number of successful solutions.

Something related

If you are among our older contestants, this problem may bring back sweet memories of the classic game Lode Runner, and perhaps memories of many happy days associated with it.