**Code Jam 2019 - World Finals**

# Analysis: Go To Considered Helpful

There are two kinds of programs: those that terminate, and those that don't!

For any set of instructions for Marlin that does not loop indefinitely (because it reaches the end of the program), there is an equivalent program that has no goto instructions, and this is the shortest way to write that program.

Similarly, if a program loops indefinitely, there is an equivalent program that consists only of move instructions with a single goto at the end of the program, and this is the shortest way to write that program.

So we only need to consider those two cases: programs that consist only of move instructions, and programs with only a single goto, as the last instruction. Then we take the shortest of all of these.

We can easily find a minimal program of the first type with a breadth-first search (BFS) which starts at `M` and stops when it reaches `N`, avoiding `#`s. If the search terminates without reaching `N`, then we can output that the case is impossible.

Finding a minimal program of the second type is harder. The program can be split into two sections which compute two separate paths: an initial path P produced by the instructions which do not repeat, and then a path Q which repeats until `N` is reached.

Let B be the point where P ends. We can use a BFS, as in the previous case, to find a minimal P for all possible points B.

Optimizing Q is slightly more involved. The first time through Q, Marlin walks through some set of grid cells. The second time through, Marlin walks through grid cells of the same pattern, but offset by some displacement vector D. This continues for some number of iterations K until Marlin reaches the `N` cell. So the first iteration of Q takes Marlin from B to B+D, but the path must avoid not only cells with a `#`, but also cells which have an equivalent cell in a later iteration which is a `#`. Additionally, the first iteration of Q must include a cell which, in a later iteration, will be `N`.

It is not easy to satisfy these constraints unless we already know the displacement D, and the number of iterations, K. So, we loop over all possibilities for these.

Inside this loop, we determine the shortest program for all possibilities for B, for the given values of D and K. We split Q into two parts, $Q_1$ and $Q_2$. $Q_1$ contains the instructions which repeat K times, and reach cell `N` on the final iteration. $Q_2$ contains the instructions which only repeat K-1 times. (If we reach `N` at the end of an iteration, $Q_2$ is empty.)

Let N be the location of the cell containing `N`. $Q_1$ is a path from B to N-(K-1)×D, and $Q_2$ is a path from N-(K-1)×D to B+D.

The path for $Q_1$ can only touch cells X such that X+i×D is empty for 0<i<K. We do a BFS from N-(K-1)×D, using these cells, to find an optimal $Q_1$ for all possibilities for B.

The path for $Q_2$ can only touch cells X such that X+i×D is empty for 0<i<K-1. We do a BFS from N-(K-1)×D, using these cells, to find an optimal $Q_2$ for all possibilties for B.

Finally, we loop over all possible cells for B, and sum up the lengths of the shortest P, $Q_1$ and $Q_2$, plus one for the goto statement.

**Running time**

Let max(**R**,**C**)=N. For the outer loop that iterates over D and K, there are $O(N^2)$ choices for D and O(N) choices for K. However, not all combinations are possible. If the number of iterations, K, is large, then the displacement D for each iteration must be small; otherwise Marlin's path would have to leave the grid to complete K iterations. The total number of valid combinations of D and K is $O(N^2)$ — we leave the proof as an exercise for the reader.

Inside the loop, setting up the grids for the BFS, running the two BFSs, and trying all values for B all take $O(N^2)$ time. So the loop takes $O(N^4)$ time.

The BFS outside the loop that computes all optimal paths P takes $O(N^2)$ time, and so does the search for the optimal solution with no goto statements. So the overall solution is $O(N^4)$, and runs fast enough to solve both test sets.

Being less careful can result in an $O(N^5)$ solution — for example, by doing O(K) work to check whether a cell is valid for the two BFSs inside the loop. This is sufficient to solve test set 1.

An exponential-time solution that naively tries all possibilities for Q is unlikely to work even for test set 1, since the maximum length of Q is too large.