In this document, I discuss the solution to a Hacker Rank problem called Grid Walking. It was one of the most involved algorithmic challenges that I have ever solved and there are quite a few complex parts.

1 Problem Statement

You are situated in an N dimensional grid at position $(x_1, x_2, \cdots, x_N)$. The dimensions of the grid are $(D_1, D_2, \cdots, D_N)$. In one step, you can walk one step ahead or behind in any one of the N dimensions. (So there are always $2N$ possible different moves). In how many ways can you take M steps such that you do not leave the grid at any point? You leave the grid if at any point x_i, either $x_i \geq 0$ or $x_i > D_i$.

2 Solution

Independence between dimensions The problem essentially decomposes into a two-part dynamic programming solution. The first key insight is to note that the dimensions are independent. That is, movement in dimension D_i does not in any way constrain the moves you can execute in dimension D_j. For example, if you took k steps in dimension D_1, you have not moved at all in dimension D_2.

Compute Indivitiual Dimensions For reasons that will become clear later, we begin considering each dimension separately. For each dimension D_i, compute the number of total possible moves for steps $m = 1, \cdots, M$. In total, that is $n \times m$ number of computations, where n is the number of dimensions and m the number of steps. We maintain n tables (1 for each dimension) to store these computations. Denote $T[i][p_i][m]$ to be total possible moves in dimension D_i for fixed m steps and starting position p_i.

Notice that this is also a dynamic programming problem. To see this, fix the dimension to be D_i and starting position p_i. It follows then that (not including edge conditions)

$$T[i][p_i][m] = T[i][p_i + 1][m - 1] + T[i][p_i + 1][m - 1]$$

$$= \text{paths stepping left} + \text{paths stepping right}$$

This is because we can decide to step left or step right. The prior to this we have decided The tables are constructed in the function **build Dimensions()**.

Combining Dimensions Our next step is to combine the tables in order to solve the target problem. To do this, we first demonstrate a simple example. Suppose we only had 2 dimension to worry about, each with starting position p_1, p_2 respectively. Further, suppose we needed to solve the problem for M

steps. If we took 0 steps in D_1 then we would take M steps in D_2. If we took 1 step in D_1 then we would take $M-1$ steps in D_2. Essentially, we are computing a convolution. The idea is that we want to figure out how we can interleave valid paths of different lengths, so that the new path is exactly length M. The formula then becomes

$$\sum_{m=0}^{M} (T[1][p_1][m] \times T[1][p_1][M-m])$$

Upon closer examination, however, the above sum only computes the total number of paths if we always travel in dimension D_1 first and then D_2. But sometimes we also travel in dimension D_2 and then D_1. Other times, we may take a few steps in D_1 and then another few steps in D_2 and flip flop. To account for this, we need to consider the combinations of paths.

Think of this as the number of ways to order a sequence consisting of m p_2 tokens and $M-m$ p_1 tokens.

$$p_1 p_1 \cdots p_1 p_2 \cdots p_2$$

To create new paths, we simply wish to determine how we can interleave two FIXED existing paths. Note that the order of the p_1's cannot change relative to each other, otherwise we would be considering a different path in dimension 1. This is why we only consider combinations rather than permutations.

$$= \sum_{m=0}^{M} \binom{M}{m} (\text{paths in } D_1 \text{ length m}) \times (\text{paths in } D_2 \text{ length M-m})$$

For simpler notation, let $T[i][p_i][m] = p_m^{(i)}$. It follows that we can continue on this logic and compute the paths recursively. Let $p_m^{(1,\cdots,i)}$ denote total number of paths that are length m for dimensions $1, \cdots, i$. We then have the recursive formula

$$p_M^{(1,\cdots,i+1)} = \sum_{m=0}^{M} \binom{M}{m} p_m^{(i+1)} p_{M-m}^{(1,\cdots,i)}$$

So, our final solution is given as

$$\text{solution} = \sum_{i=1}^{n} \sum_{m=0}^{M} \binom{M}{m} p_m^{(i+1)} p_{M-m}^{(1,\cdots,i)}$$

3 Optimization

Overflow Our path counts can get very very large, which is why the solution on hacker rank asks us to mod out. Firstly, use long long types to handle large

numbers. Further, we need to perform our operations in modulo p space. Note a few helpful identies of modular arithmetic.

$$(x + y)\%p = (x\%p + y\%p)\%p$$

$$(xy)\%p = (x\%p)(y\%p)\%p$$

We use these operations in our program.

Fast computation of binomial coefficients We use many binomial coefficients throughout our program. To speed up computation, we precompute these using the following formula.

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$