January 2020 Plat Review

Ray Bai and Helena Liu

February 2020

1 Problem 1: Cave Painting

1.1 Problem Statement

Bessie has become an artist and is creating paintings of caves! Her current work in progress is a grid of height N such that each row of the grid contains exactly M squares ($1 \leq N, M \leq 1000$). Each square is either empty, filled with rock, or filled with water. Bessie has already painted the squares containing rock, including the entire border of the painting. She now wants to fill some empty squares with water such that if the painting were real, there would be no net motion of water. Define the height of a square in the $i-th$ row from the top to be $N+1-i$. Bessie wants her painting to satisfy the following constraint:

Suppose that square a is filled with water. Then if there exists a path from a to square b using only empty or water squares that are not higher than a such that every two adjacent squares on the path share a side, then b is also filled with water.

Find the number of different paintings Bessie can make modulo $10^9 + 7$. Bessie may fill any number of empty squares with water, including none or all.

1.2 Solution

- Construct connected components bottom up
- For each row, use either DP or combinatorics to find answer

1.3 Union Find/Combinations

We will run the following procedure on every separate component, which can be found using simple flood fill in $O(NM)$ time. Let's begin from the bottom row. On top of maintaining each component, we will also store the depth of each component. The depth of a component is the index of the lowest cell contained in the component minus the highest cell. After each phase of merging/creating new components, we must find a way to contribute the number of possible paintings without over-counting. While it is possible to create an explicit DP array, this is also possible with combo. We know that the length of each component is at most M, so we can store the heights of each component in a list. Then, we iterate through this list of integers and maintain the total number of different paintings minus the total number of paintings with no components' depth filled completely with water. By subtracting these two values, we will obtain the contribution of this state without double counting. Repeat for the remaining components, and simply multiply by each components' value. Be careful of modding/overflow issues!

1.4 Time Complexity Analysis

Note that we run this algorithm for every "component" in this painting. If the input consisted of one large component, the run-time for it would be $O(NM * \alpha(N)$, where $\alpha(N)$ describes the inverse Ackermann function from union-find operations.

2 Problem 3: Falling Portals

2.1 Problem Statement

There are N ($2 \leq N \leq 2*10^5$) worlds. Each world i (for $1 \leq i \leq N$) starts at x coordinate i and y coordinate A_i ($1 \leq A_i \leq 10^9$). All A_i values are distinct. There also is a cow on each world. At time 0, the worlds fall in the negative y-direction, world i falling at a speed of i units per second. Whenever two worlds are at the same y coordinate (possibly a fractional time), a cow on one world can choose to teleport instantaneously to the other world. For each world i, the cow on world i wants to travel to world Q_i ($Q_i \neq i$).

Determine how long the journey will take for each cow, if they travel optimally. The output for each query should be a fraction a/b where a and b are positive and relatively prime integers, or -1 if the journey is impossible.

2.2 Solution

- Consider each world to be a line
- Run Convex hull algorithm
- Binary search to find intersections

Consider each world to be a line with slope -i and y-intersection A_i. Call a point (T, Y) on this graph "attainable" if it is possible for the cow on world i to be at y-coordinate Y at time T.

Suppose that $A[Q_i] < A_i$. By the definition of convex hull, there must be no attainable points below the lower convex hull of all lines representing worlds j such that $A_j \geq A_i$. In addition, all points on this hull must be attainable. Therefore, the task is to find the t-coordinate of the intersection of the line $y = -Q_i * t + A[Q_i]$, which represents the path of the destination world, with this lower hull.

We can compute the hulls for all worlds by sorting the lines by decreasing order of A_i and adding them to the convex hull one by one. After computing the hull for world i, we can binary search through the lines making up the convex hull to find the intersection of the line $y = -Q_i * t + A[Q_i]$ with the hull. The same logic applies in reverse for queries in which $A[Q_i] > A_i$. Greatest common denominator can be used to simplify the answer into the desired fraction.

2.3 Time Complexity

Computing a convex hull takes $O(N \log N)$ time and binary searching on each convex hull will take $O(\log N)$ time per search. Therefore, the solution will run in $O(N(\log N)^2)$ time.

Note that USACO gives a certain number of lower-bound test cases, so an $O(N^3)$ brute force or optimized $O(N^2)$ brute force can also be used to receive partial credit on this problem.