Kick Start 2018 - Round D

Analysis: Paragliding

Paragliding: Analysis

Small dataset

First, we need a way to determine if a balloon is able to be collected. Note that Edge glides at a 45 degree angle, which means that for each unit he glides horizontally, he also descends one unit vertically. Since Edge can glide down from any position on a tower, Edge can collect any balloon if and only if there is a tower such that the height of the tower is greater than or equal to the horizontal distance between the balloon and the tower. For each balloon i, we can execute a linear search of the towers for a tower j which satisfies **h**[j] ≥ **y**[i] + abs(**p**[j] - **x**[i]). The time complexity for this approach is O(**N** Ã— **K**).

Large dataset

We will need to optimize our previous solution in order to have a fast enough solution for the Large dataset. Since we definitely need to iterate through every balloon, we can conclude that we should be optimizing the search for a possible tower for Edge to jump from.

A key observation is that some towers may be irrelevant. For two towers A and B, if the height of A is less than or equal to the height of B minus the distance between A and B, then any balloon which is reachable by A is also reachable by B. A is considered an irrelevant tower. All towers which are not irrelevant can be considered relevant. To eliminate irrelevant towers efficiently, we want to create a sorted array of all towers based on their positions on the x-axis. Then, we want another array to keep track of the relevant towers and their positions. We iterate through the sorted array of all towers and do the following for each tower i: compare tower i to the tower at the end of the array of relevant towers. If tower i is irrelevant with respect to that last tower, continue on to the next tower in the sorted array of all towers; otherwise, continue removing towers from the end of the relevant tower array until both tower i and the last tower are relevant or until the relevant tower array is empty. Then, add tower i to the end of the relevant tower array and continue to the next tower in the sorted array.

The end result is that we have an array of all relevant towers which are also sorted by the towers' positions on the x-axis. Now, for each balloon we can check to see if it is further left or further right than all of the relevant towers. If so, we can check the closest tower to see if it allows Edge to collect the balloon; otherwise if the balloon is not further left or further right, we can use binary search to find the two closest towers to the balloon (one on each side) on the x-axis because we removed all of the irrelevant towers, meaning the only towers which can get to a certain balloon are the closest towers. If Edge can reach the balloon from either of these towers, then the balloon is collectable.

So what is the time complexity of our new solution? Sorting the original list of towers will take O(**N** log **N**) time. During the process of determining relevant towers, each tower cannot be added or removed more than once, so that step is O(**N**). It is possible that all towers are relevant, so doing a binary search at each balloon will take O(**K** log **N**) time; therefore, our time complexity is O((**N** + **K**) log **N**).