February Contest Review

Mihir Patel

2 March 2018

1 Gold: Snow Boots

1.1 Problem

It's winter on the farm, and that means snow! There are N tiles on the path from the farmhouse to the barn, conveniently numbered $1 \dots N$, and tile i is covered in f_i feet of snow. In his farmhouse cellar, Farmer John has B pairs of boots, numbered $1 \dots B$. Some pairs are more heavy-duty than others, and some pairs are more agile than others. In particular, pair i lets FJ step in snow at most s_i feet deep, and lets FJ move at most d_i forward in each step.

Farmer John starts off on tile 1 and must reach tile N to wake up the cows. Tile 1 is sheltered by the farmhouse roof, and tile N is sheltered by the barn roof, so neither of these tiles has any snow. Help Farmer John determine which pairs of snow boots will allow him to make the trek.

1.2 Solution

We create a list of all tiles within the path that is sorted from maximum snow to minimum snow. Each tile also points to its predecessor and its successor. We then create a list of all the boot strengths sorted with maximum first. We create an integer defining the maximum distance between tiles, initially set to one.

We loop through the list of boot strengths. As we hit a new boot, we go through the list removing all tiles that have too much snow. When a tile is removed, its predecessor and successor are updated to contain the new connection. If the difference between the predecessor and successor is a new maximum distance, the maximum distance is updated. A boot is only feasible if it has a greater step size than the maximum distance that exists.

2 Gold: Directory Traversal

2.1 Problem

Bessie the cow is surprisingly computer savvy. On her computer in the barn, she stores all of her precious files in a collection of directories; for example: bessie/ folder1/ file1 folder2/ file2 folder3/ file3 file4 There is a single "top level" directory, called bessie.

Bessie can navigate to be inside any directory she wants. From a given directory, any file can be referenced by a "relative path". In a relative path, the symbol ".." refers to the parent directory. If Bessie were in folder2, she could refer to the four files as follows:

../file1 file2 ../../folder3/file3 ../../file4

2.2 Solution

We create a tree where each node represents a folder and mimics the file system. Each node has an associated value that indicates the number of files inside it. We also store the number of files F. Whenever we descend into a directory, we shave off the length of the directory name times the number of files within it but add 3 (because of "../") for F minus the number of files within a directory.

To calculate the minimum value, we first determine the value starting from the root. We then traverse the entire tree. At each node, we take its parent's value, subtract number of subfiles times length of directory name, and add 3 times F. Throughout this traversal process, we maintain a global minimum value.

3 Plat: New Barns

3.1 Problem

Farmer John notices that his cows tend to get into arguments if they are packed too closely together, so he wants to open a series of new barns to help spread them out.

Whenever FJ constructs a new barn, he connects it with at most one bidirectional pathway to an existing barn. In order to make sure his cows are spread sufficiently far apart, he sometimes wants to determine the distance from a certain barn to the farthest possible barn reachable from it (the distance between two barns is the number of paths one must traverse to go from one barn to the other).

FJ will give a total of Q ($1M = Q <= 10^5$) queries, each either of the form "build" or "distance". For a build query, FJ builds a barn and links it with at most one previously built barn. For a distance query, FJ asks you the distance from a certain barn to the farthest barn reachable from it via a series of pathways. It is guaranteed that the queried barn has already been built. Please help FJ answer all of these queries.

3.2 Solution

We first introduce centroid decomposition. Centroid decomposition is the process of finding the "middle" of a tree where each subtree is $n/2$ or less in size. To do this, we take a random node and see if it matches this criteria through a depth first search. If it does not match this criteria we move from the current node to the node in the subtree that is of a size greater than $n/2$. This is done recursively till a centroid is found.

We construct a forest with all nodes including barns that aren't built yet. We indicate unbuilt barns as inactive. Now, to determine the longest path for a given barn, we utilize centroid decomposition. We identify the centroid and set this as the root. We then find the depth of the two tallest subtrees and store this. The depth of these tallest subtrees only consider active nodes. This is done through DFS. Then, for each subtree of the centroid, we apply centroid decomposition recursively.

This yields an effective querying method. Given the root centroid, we can say if the longest path goes through the centroid, then the path length is the depth of the query plus the tallest subtree. If the tallest subtree contains the query, we use the second tallest subtree (this is why we calculated top two). For the case in which the longest path doesn't go through the centroid, we have to check the subtree containing the query. For this, we go to the centroid within that subtree and repeat this process recursively. The global maximum is tracked. This yields $O(logN)$ time for a query.

We must also handle activating specific barns. To activate a barn, we iterate through all centroids it is a part of and update the depth values. For this update, we go upstream from the activated leaf to the root and compare this to the previously calculated depth values. The number of centroid trees that exist for an addition is, by definition, $logN$, so we get $O(logN)$ time.

With this approach, each step is $logN$ so our total efficiency is $O(NlogN)$. Note that a common approach is to instead use an active graph and simply append each new barn to this active graph that has precomputed maximum distances and respective paths. This approach requires comparing a potentially new path created by the additional node to the current maximum path and discovering this new path requires a BFS. By contrast, with trees, we can always go upstream to the parent node so the path is already known.