Code Jam 2012 - Round 2

Analysis: Swinging Wild

Translating to a graph problem

To analyse the problem, let us begin with describing the state we can be in at any particular point during traversing the swamp. It is easy to see that we need two items of information to describe this state - which vine are we currently holding on to, and how far away from the root are we holding it.

Once we have such a description, we can frame the problem as a path-finding problem in a graph. We start in the state (0, $\mathbf{d}_0$), and we want to any state (**i**, **p**) with $\mathbf{p} + \mathbf{d_i} \geq \mathbf{D}$ (to simplify, we can also add an artificial **N**+1st vine where our true love stands and demand that we reach the state (**N**, **p**) for any **p**).

The transitions between states (or, in other words, edges in the graph) are allowed swings we can make. If we currently hold vine **i**, at **p** units away from the root, we can swing to any vine **j** rooted between $\mathbf{d_i} - \mathbf{p}$ and $\mathbf{d_i} + \mathbf{p}$, and the new length will be the minimum of $|\mathbf{d_i} - \mathbf{d_j}|$ and $\mathbf{l_j}$. Note that the only use of climbing up a vine is to catch a vine that would be too short to be within our swing path - so we implicitly include it in the transitions described above (and thus do not need extra transitions from (**i**, **p**) to (**i**, **p**-1)).

Limiting the number of nodes

As described, we could have a large number of states (as there are very many possible vine lengths. A crucial fact to notice is that the second part of the state (the length away from root) is uniquely determined by the previous vine we got here from; and it is independent of where did we hold it (assuming we held it far enough to actually reach this state). This means that there are at most $\mathbf{N}^2$ states we will ever consider, as each is determined by a pair of vine indices.

We can now solve the small input. We have a graph with $\mathbf{N}^2$ nodes and at most **N** edges from each node, and we want to verify whether a path between some two nodes exists (we can merge all the target nodes into one node for simplicity). As we have at most $\mathbf{N}^3$ edges in total, any standard graph traversal algorithm (like BFS or DFS) will allow us to solve the problem.

Limiting the number of edges

For the large input, a $O(\mathbf{N}^3)$ solution will be unsatisfactory, and we need a bit more subtlety. There is a number of tricks one can use to beat down the complexity. One is to make use again of the fact that the target node depends only on the vine we start from, and not on the position at which we hold it. This means that there are in fact at most **N** edges from a given vine - if we manage to reach some vine **j** from a given vine **i** when holding it at position **A**, we do not need to check this edge when considering moves from vine **i** held at position **B** (because even if we reach vine **j**, we will arrive in a state that we have already analysed). Thus, we need to make at most $\mathbf{N}^2$ edge traversals to visit all reachable nodes. There are various techniques to actually code this (for instance, for each vine, we could order the other vines by distance from this vine, and each time process this list from the closest vine and remove all traversed edges), we encourage you to explore the options.

An alternative for the large problem

An alternative is to notice another fact - the position part of the state (that is, the distance away from the root that you hold the vine at) never increases. This is because if you swing from vine **i** to vine **j**, it means you were holding **i** at least $|\mathbf{d}_i - \mathbf{d}_j|$ away from the root, while this quantity is at the same time an upper bound on the position you will hold vine **j** at.

This means that we can use Dijkstra's algorithm to find, for each vine, the maximum position we can hold this vine at - we treat the decreasing vine position as increasing time, and in each step we analyse what lengths could we obtain for each vine by moving from the current vine, and then choose the vine with the largest length to analyse. This will give us a $O(\mathbf{N}^2 \log \mathbf{N})$ solution, which should be fast enough.

Going even faster

A fun fact is that this problem can be solved faster than $O(\mathbf{N}^2)$, although you didn't need to notice this to solve our data sets. The key fact here is that if you can pass the swamp at all, you can always do it without going backwards (that is, you always catch a vine that's in front of you, you never swing back). An easy way to use this observation is to modify the Dijkstra's algorithm mentioned above to process the vines from first to last, which will turn $O(\mathbf{N}^2 \log \mathbf{N})$ into $O(\mathbf{N}^2)$.

To go down to $O(\mathbf{N})$, we need one more trick. Notice that if we can reach a vine, we will get the largest (meaning best) position if we swing to it from a vine that is as far away as possible. As we move only forward, this means that as soon as we can reach any particular vine, we should note the position achieved and we never need to check any other way of reaching it. This means we can get an $O(\mathbf{N})$ solution by keeping track of the vine we are currently processing and farthest we have reached so far, and from each vine trying to update only vines that we have not reached as yet. As each vine will be updated at most once, and read at most once, we will do $O(\mathbf{N})$ operations.

We encourage you to flesh out the details and try to prove the "never go back" lemma - it's not trivial!