Analysis for the task Artillery

Let's choose some vertex as a root of the tree. Let it be X and its subtrees be S_1, … S_M.

It's useful to think of the task as a dynamic programming problem. Let's assume we know the minimal K required to destroy the pawn in each of the subtrees: minK(S_i). Then we can acquire the following idea: Using minK(S_1)+1 to destroy the pawn if it is in S_1 as each time we hit the root as well to assure that the pawn does not escape from S_1. Then the same with minK(S_2)+1 and so on until minK(S_M)+1.

This way the answer will be max(minK(S_i)+1). And implementing this solution, it would be with time comlexity O(N*N), because we have to try for each choice of X to calculate minK for each node and its subtrees. The unpleasant thing about this solution is that it does not give an optimal solution. For example for a path it will give (N-1)/2 rounded up (when X is the middle vertex), as the optimal answer is 2.

And here we come to the conclusion that it is not necessary to hit the root every time. And the interesting question is when not to shoot the root.

If we do not shoot the root this enables the pawn to enter the root from one of the subtrees in which the pawn can be and then could enter every subtree if it is not stopped by another shot.

There are 3 ways to not shoot the root. One is to hit the vertices exactly bellow the root, but that is worse than just hitting the root. Another option is for us to still have no guarantee on which subtree the pawn cannot be. Then if the pawn changes subtrees this does not affect us. A third option is if there is only one subtree where the pawn can be and to just „push" it down that subtree.

This gives us the following sequence of shots: First shoot a subtree „from bottom to top" (i.e. to force the pawn away from that subtree towards the root), as we do not need to hit the root. Then we hit all subtrees but one as we are hitting the root with each turn (as in the first idea). And finally when only one subtree where the pawn can be is left, to shoot this subtree „from top to bottom" (i.e. to not give the pawn a chance to escape from this subtree via the root) as in this case we can again not shoot at the root.

As of implementation it is actually a DP realization with two states: root of subtree and whether we are shooting the subtree „from bottom to top" (up), „from top to bottom" (down) or normal. And the idea above is implemented with the following recurrent relations:

- minK(X, normal) = max(minK(S_i, up), minK(S_j, down), 1+$\max_{p!=i,j}$(minK(S_p, normal))) for best i and j
- minK(X, up) = max(minK(S_i, up), 1+$\max_{p!=i}$(minK(S_p, normal))) for best i
- minK(X, down) = max(minK(S_i, down), 1+$\max_{p!=i}$(minK(S_p, normal))) for best i

Best i and j can be calculated with one pass through the children. And thus, the calculation of minK with a chosen X has a time complexity O(N) and the whole solution becomes O(N*N).

Now the hard part is over and this solution actually finds the optimal solution. Actually we are almost done with the proof. The statements above, for when we cannot shoot the root, are almost enough. The only missing part is the fact that we can achieve optimal solution by shooting the subtrees one by one. Which is quite obvious, since we can optimize other solutions shooting at more than one subtree to shoot at a single subtree at a time without increasing K.

Now the easy part is to achieve O(N). There are a few ways. The intuitive and standard one (and the harder to implement) is to make the same calculation of minK for a random root X. And then to make a second pass, when we will consider for each node the option to be a root by calculating only its subtree above its current position. All other ones are already calculated by the first pass. Thus, with two passes of time O(N) we find the best root and answer.

The other (easier to code) method is to make an observation that each node which isn't a leaf will give us an optimal answer. I will leave the proof as an exercise for the reader.

This way the solution is to just choose any X such that it is not a leaf and to run the minK algorithm described above. This gives as the solution for time complexity O(N).

Author: Ivo Dilov