AC-IO 2020 Contest 1 - Editorial

February 11, 2020

1 Jogging

Rephrasing the problem in graph theory terms with roads as edges and intersections as nodes, let $d(u,v)$ denote length of shortest path from u to v. Then the length of the shortest route involving node i is $d(1,i) + d(i,1)$.

- For subtask 1 We may use Floyd-Warshall to compute d(u,v)
- For subtask 2 Note that $d(1,i) = d(i,1)$ so we run Dijkstra from node 1.
- For subtasks 3 and 4 We compute $d(1,i)$ as in subtask 1, reverse all the edges and run Dijkstra again from node 1 on the new graph to get $d(i,1)$.

Hence the solution runs two dijkstras and hence has complexity $O(M \log N)$.

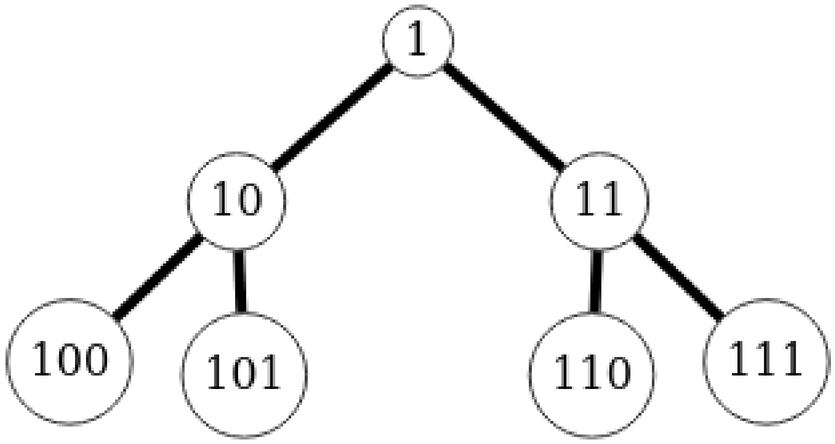
2 Endgame Spoilers

Call an edge special if it has non-unit weight. Also, we root the tree at node 1 so the parent of node i, denoted $p(i)$, is equal to $\lfloor \frac{i}{2} \rfloor$.

Observe, that the answer is at least $2H - 2$. From now on, we only consider paths which pass at least 1 special edge. The critical observation is that such paths must all pass an ancestor of an node incident to a special edge. Further, there are only $O(NH)$ such nodes. Hence in the spirit of the tree dp used in subtask 2 we must determine for all such nodes the maximum distance to a leaf via its left and right subtrees.

One way to think about this is to simulate the tree dp of Subtask 1 and 2 but lazily stopping when the current subtree contain no more special edges and returning the depth of the subtree instead. However, there initially seems to be no way of doing this without a set or map.

Consider this way of modelling such lazy create perfect binary trees - as a prefix tree modelled on the integers in binary. Consider the following diagram:



As you can see, the ancestors of each node in the tree are all prefixes of the node, and so you can in fact start at the top of the tree and walk down it to reach the nodes you want.

The complexity of this solution is $O(NH)$. Inelegant implementations in $O(N^2H)$ were still sufficient to pass subtask 3, and $O(NH \log N)$/unordered map solutions were sometimes accepted if optimisation was applied. The official solution takes a tiny 0.062 seconds on CMS.

- Subtasks 1 and 2 were semi-brute force subtasks. The standard tree diameter solution in $O(2^H)$ suffices to pass them. However the idea of the tree diameter solution offer insight to the full solution.
- Subtask 3 is quite hard and needed all observations needed for the full solution, but could be more slack in implementation.

3 Superheroes

Two villians cannot be defeated on the same day if and only if there exists a hero that is needed to defeat them both. Hence lets construct a graph where the nodes are villians and two villians share an edge if and only if they cannot both be defeated on the same day. This can be done is $O(N^2M)$. Now we have reduced the problem to finding the minimum number of colours to colour a graph such that no two adjacent nodes have the same colour, and the number of ways to do so.

As bounds on N are small, we may use subset dynamic programming. Let $dp[S]$ denote the number colours needed colour the subgraph induced by the nodes in S with a minimal amount of colours. Clearly $dp[\varnothing] = 0$ and $ways[\varnothing] = 1$. Now for a particular subset S, we may pick an arbitary subset of nodes $I \subseteq S$ such that between the nodes in I there is no edges and colour it with a new colour. Lets call such a set I

as independent. Hence the dp equation is:

$$dp[S] = \min_{I \subseteq S \text{ and I independent}} \{dp[S \backslash I] + 1\} \tag{1}$$

We proceed similarly with number of ways. We can find all independent sets of a graph in $O(2^N N^2)$ by enumerating all subsets, and at each dp subproblem enumerate all subsets of a subproblem and take each independent subset enumerated into account. To analyse the complexity of the dynamic program, there are $\binom{N}{k}$ subsets of the N nodes with K elements. Each K element subset takes $O(2^K)$ to enumerate by precalculating subsets or bitwise operations, hence the complexity is

$$\sum_{k=1}^{N} \binom{N}{k} 2^k = (1+2)^N = 3^N \tag{2}$$

Overall, the complexity is $O(MN^2 + 2^N N^2 + 3^N)$. For subtasks:

- Subtask 1 was a brute force subtask. $O(N^N)$ sufficed.
- Subtask 2,3 were intended to demonstrate the idea of independence.
- Subtask 4,5 were intended to allow inferior dp solutions to pass.
- Subtask 6 was intended to only allow better solutions.