Star Trek Solution

Topics

graph, tree, winning-losing states, dynamic programming, dfs

Problem author

Máté Busa

Subtask 2

$N = 2$

The Captain always wins.
A possible winning strategy: she uses only tunnels.
In this way, Gábor is forced to use only portals. After using a portal they will be in a new universe where the Captain can use the tunnel. So Captain can always move after Gábor, but there is a point where Gábor can't move. That's why Gábor can't win.
The answer is the total number of possible placements: 4^D. it can be computed in $O(logD)$ operations via fast exponentiation.

This subtask can be solved in $O(logD)$

Winning-Losing states

Lets play this game in a rooted tree where the first player moves from the root r.
Let's call a node L(osing)-state if the player moving from there can't win. Call that node W(inning)-state otherwise.
The players can only increase the distance from r; that's why every leaf is an L-state.
A node is W-state iff it has an L-state child.
The root's state can be calculated in $O(N)$ operation with dfs if the size of the tree is N.

L : set of nodes that are **L**-states as the root
W : set of nodes that are **W**-states as the root

Subtask 3

$N \leq 100, D = 1$

We test all possible placements. A placement will give us a tree of size $2N$ rooted at P_1^0. For all possibility we check the root's state. There are $O(N^2)$ different placements and it takes $O(N)$ operation to check a single one.

This subtask can be solved in $O(N^3)$

Critical L-state

It's clear that we have $D+1$ trees of the same structure and they are connected into a bigger tree. This big tree is rooted at P_1^0, but all small trees have a root-like node in the big tree

(P_1^0 in the first universe and $P_{B_{i-1}}^i$ in the ith parallel universe).
We will work now with the small tree of size N.

Let's root this tree in an arbitrary node r. Let's call this tree $tree_r$. Let's denote the parent of node c by $P(c)$.
Connecting a new node y with a given state to a node x (of the original tree) may change some states.

It can be proved that the state of x changes iff x and y are both L-states. In this case, x will become W-state. This may cause further changes in the tree:
If $P(x)$ has only one L-state child (x), then its state will also change from W to L.
If $P(P(x))$ is L-state, then its state will also change from L to W.
$P(P(P(x)))$ will act like $P(x)$. This wave of change will stop at some node z, where z will be the uppermost node whose state changed. We call x a critical L-state if $z = r$.

$\mathbf{C_r}$: set of nodes that are critical L-states when r is the root.
$\mathbf{C_r}$ can be computed in $O(N)$ time for a given r using dfs.

Subtask 4

$N \leq 1000$ and $D = 1$

We should connect 2 uniform trees of size N. The first tree is rooted at index 1 (the starting node).
If the root of $tree_1$ is W-state it will only change it's state if we connect a L-state to one of its critical node. That's why; the answer is $N * |W| + (N - |C_1|) * |L|$.
If the root of $tree_1$ is L-state it will only change it's state if we connect a L-state to one of its critical node. That's why; the answer is $|C_1| * |L|$.

Calculating $|C_1|$ requires $O(N)$ time while calculating $|L|$ and $|W|$ requires $O(N^2)$.

This subtask can be solved in $O(N^2)$

Subtask 5

$D = 1$

We must calculate $|L|$ and $|W|$ faster than in subtask 4.
Let's say the original tree is rooted in v and we want to reroot this tree in one of its neighbors, u. We can see that only v's and u's state may change while doing this. The new states can be computed in constant time if we know the number of L-state children for every node (which can be computed by a single dfs).
We can reroot the tree easily in all nodes with one dfs.

This subtask can be solved in $O(N)$

Subtask 6

$N \leq 1000, D \leq 10^5$

For all possible roots r we calculate $|C_r|$. This takes $O(N^2)$ time.
Let us define L_D as the number of ways to

- choose a starting node, and
- install portals,

such that the first player will **lose**, when considering the game with D parallel universes (plus the starting one).

By definition, the **W**/**L** status of the starting node v will change if and only if we add an edge from a node in $\mathbf{C_v}$ leading to an **L** state. Therefore, the number of ways to add all D portals in a way that changes the status of the starting node is

$$|\mathbf{C_v}|\, L_{D-1}.$$

We can now calculate $(L_D)_v$, the number of ways to make v a losing root with respect to the remaining D (plus one) universes: if v is **W**, then we have to add the remaining portals in a way that *changes the status of* v; if v is **L**, then we have to add them in *any other way*. Hence,

$$(L_D)_v = \begin{cases} |\mathbf{C_v}|\, L_{D-1} & \text{if } v \in W \\ N^{2D} - |\mathbf{C_v}|\, L_{D-1} & \text{if } v \in L. \end{cases}$$

Clearly,

$$\begin{aligned} L_D &= \sum_v (L_D)_v \\ &= \sum_{v \in W} |\mathbf{C_v}|\, L_{D-1} + \sum_{v \in L} (N^{2D} - |\mathbf{C_v}|\, L_{D-1}) \\ &= |\mathbf{L}|\, N^{2D} + \left(\sum_{v \in W} |\mathbf{C_v}| - \sum_{v \in L} |\mathbf{C_v}| \right) L_{D-1} \\ &= |\mathbf{L}|\, N^{2D} + E \cdot L_{D-1} \qquad \text{where } E \triangleq \left(\sum_{v \in W} |\mathbf{C_v}| - \sum_{v \in L} |\mathbf{C_v}| \right). \end{aligned}$$

In the last universe, we have $L_0 = |\mathbf{L}|$, by definition.

The answer to the original question is the number of ways to make the starting node v_1 into **W**, which is given by

$$\mathit{Solution} = N^{2D} - (L_D)_{v_1}.$$

We can calculate this value in $O(D)$ time using dynamic programming.

This subtask can be solved in $O(N^2 + D)$

Subtask 7

$D \leq 10^5$

We must calculate $|C_r|$ for all r faster than in subtask 6.
We can use the idea described in subtask 5 (when calculating $|L|$ and $|W|$ fast).

This subtask can be solved in $O(N + D)$

Subtask 8

Original constraints.

Solution 1:

This subtask can be solved like subtask 6 but we calculate L_1, L_2, L_4, L_8 ... (i.e. L_{2^i}), where we can compute L_{2^i} from $L_{2^{i-1}}$. With the bit-representation of $D-1$ we can calculate L_{D-1} in $O(logD)$ operations.

This subtask can be solved in $O(N + logD)$

Solution 2: Closed Form

To solve the last subtask, we need to calculate L_{D-1} in sub-linear time. To do this, we can solve the recurrence relation, which yields the closed form

$$L_{D-1} = |L| \frac{N^{2D} - E^D}{N^2 - E}.$$

This could be calculated via $O(\log D)$ exponentiation and modular inverse, but this is not necessary: we can easily eliminate the division by writing $a \triangleq N^2$ and $b \triangleq E$, and using the well-known identity for $(a^D - b^D)$ to get

$$L_X = |L| \sum_{k=0}^{X} a^k b^{X-k}.$$

This can be calculated in $O(\log D)$ time, by repeatedly halving:

$$\begin{aligned} L_{2X+1} &= (b^{X+1} + a^{X+1}) L_X \\ L_{2X} &= b^{X+1} L_{X-1} + |L| a^X b^X + a^{X+1} L_{X-1}. \end{aligned}$$