T2623 Peacock Aviary

Isaac Wong {WongChun1234}
2026-05-24

Background

- Problem Idea by `QwertyPi`
- Preparation by `WongChun1234`, `firewater`
- Special Thanks to `QwertyPi`, `yellowtoad`, `firewater` for testing!
- Firewater ACed this task for 16 times already...

Peacocks(孔雀) in an aviary(鳥舍)
(taken during HKOI BBQ)

The Problem

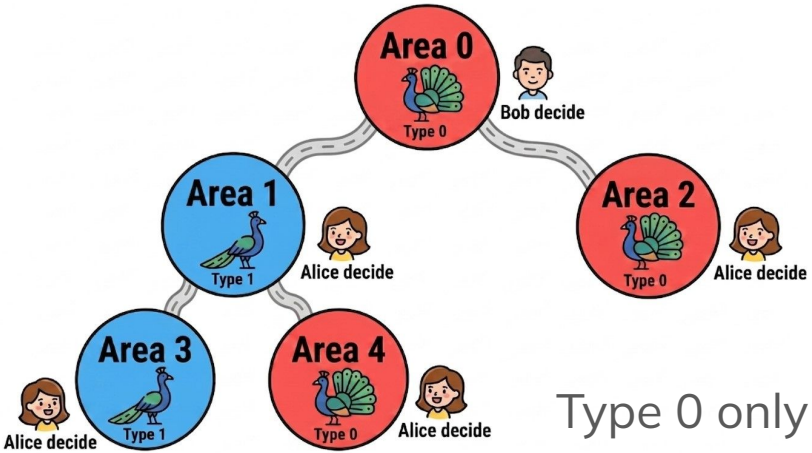
Given a tree with N nodes

- Each node has one type of peacock, and is either controlled by Alice or Bob
- Alice and Bob walks from the root of the tree to some leaf, whoever controls the node decides which child to go to next
- How many types of peacock that if Alice decides at the beginning that she want to see it, she really can see it no matter what path Bob chooses?

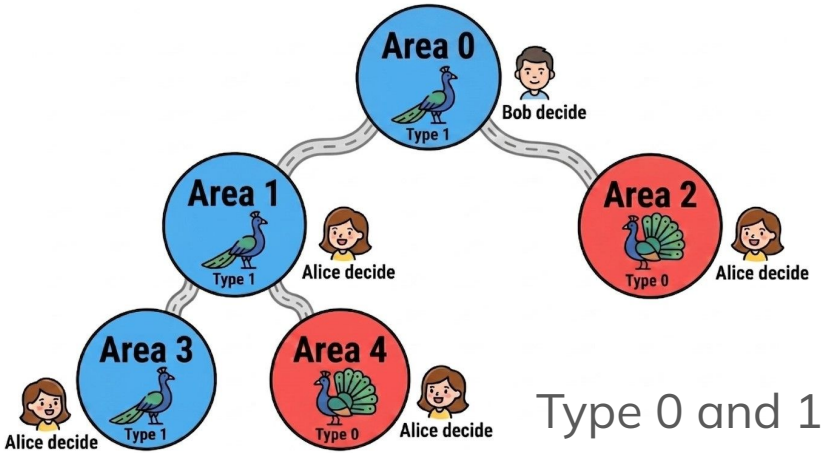
Support Q queries online:

- Change the type of peacock for some node and answer the same question again

Example



Type 0 only



Type 0 and 1

Subtasks

#	Score	N, K	Q	Constraints
1	5	≤ 5000		Star
2	7	≤ 5e4		Star
3	16	≤ 5000	= 0	
4	9	≤ 5000		
5	14	≤ 5e4		Binary
6	23	≤ 5e4	= 0	
7	22	≤ 5e4		
8	4	≤ 2e5		

Statistics

- ~~Easy task :P~~

#	Score	Predict #	Actual #
1	5	60	59
2	7	40	52
3	16	30	44
4	9	15	20
5	14	8	9
6	23	6	14
7	22	4	0
8	4	3	0

Statistics

Average Score: **39.9**

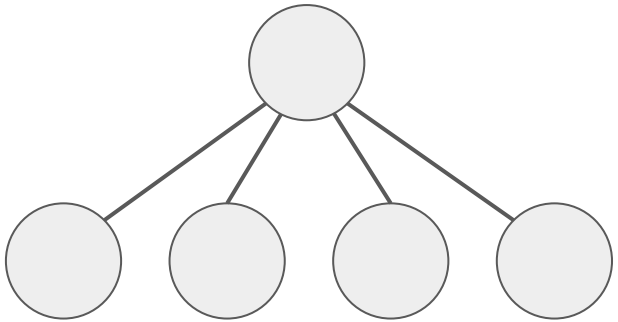
Average Score: **29.5**

Seeing the peacock gives you an advantage in this task. Come to BBQ next year!

Subtask 1-2: Basic Understanding

When all P[i] = 0, the tree is a star

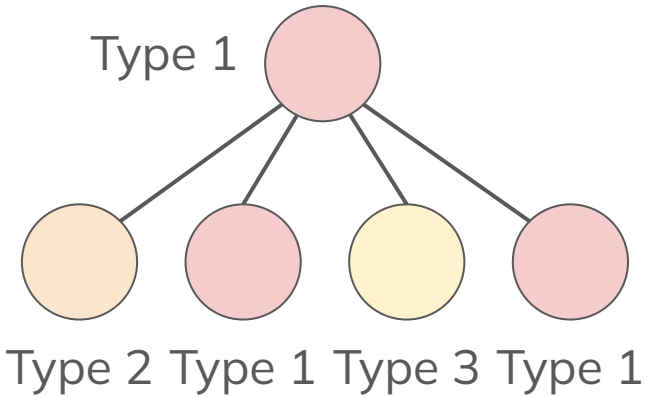
Only 1 decision need to be made



Subtask 1-2: Basic Understanding

Case 1: Alice controls node 0

What types can Alice guarantee seeing?

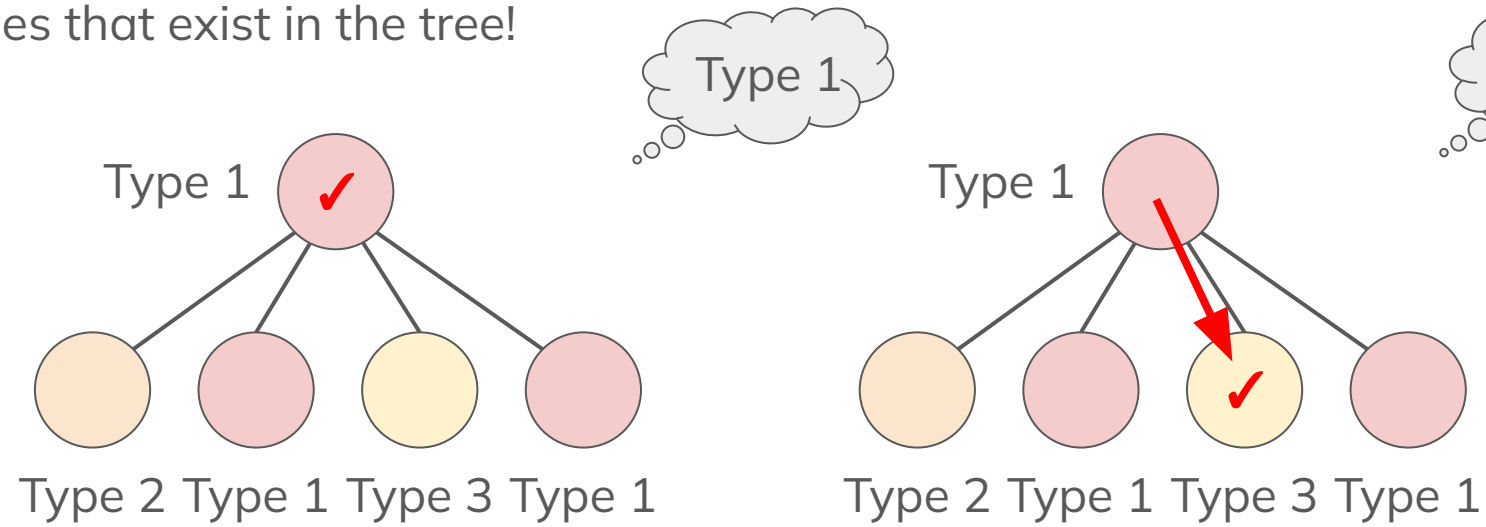


Subtask 1-2: Basic Understanding

Case 1: Alice controls node 0

What types can Alice guarantee seeing?

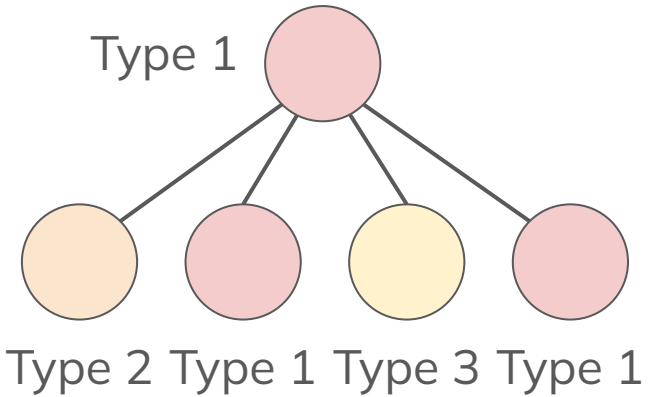
All types that exist in the tree!



Subtask 1-2: Basic Understanding

Case 2: Bob controls node 0

What types can Alice guarantee seeing?

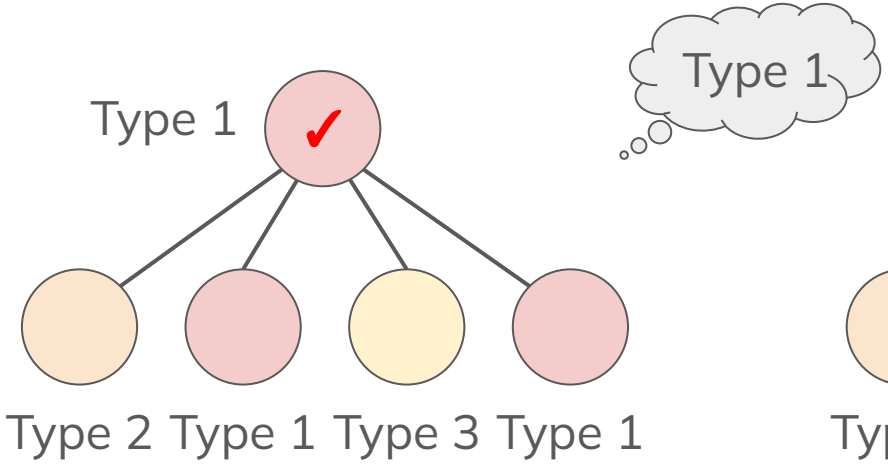


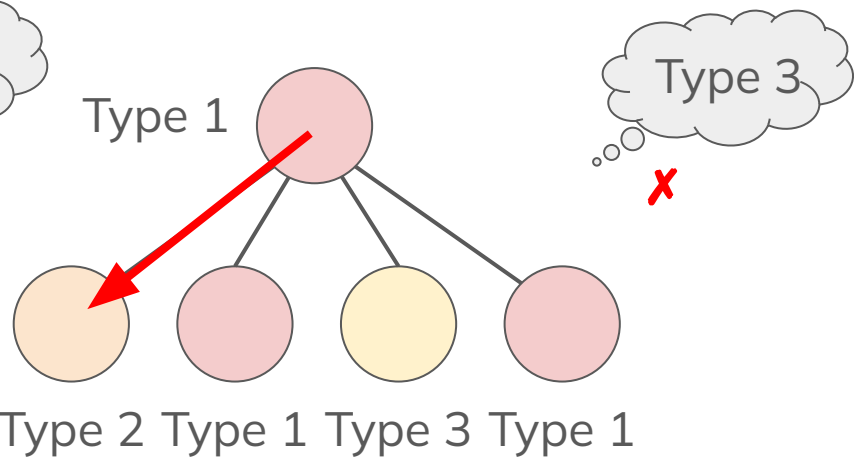
Subtask 1-2: Basic Understanding

Case 2: Bob controls node 0

What types can Alice guarantee seeing?

Type of the root and type of the leaf only if all leaf are the same type





Subtask 1 (5%): N, K, Q ≤ 5000, Star

- If Alice controls node 0:
  - Maintain a frequency array of types
  - Output number of types with >0 occurrences
  - Time complexity: O(QK)
- If Bob controls node 0:
  - Output 2 if type of leaf ≠ type of node 0 and all leaf has same color
  - Output 1 otherwise
  - Time complexity: O(QN)

Subtask 2 (7%): N, K, Q ≤ 5×10^4, Star

Useful Trick

Instead of “Changing type of node X from type Y to type Z”, we can view the updates as “Remove type Y from node X” and “Add type Z to node X”

- With this, every part of update only involves 1 type instead of 2

Subtask 2 (7%): N, K, Q ≤ 5×10^4, Star

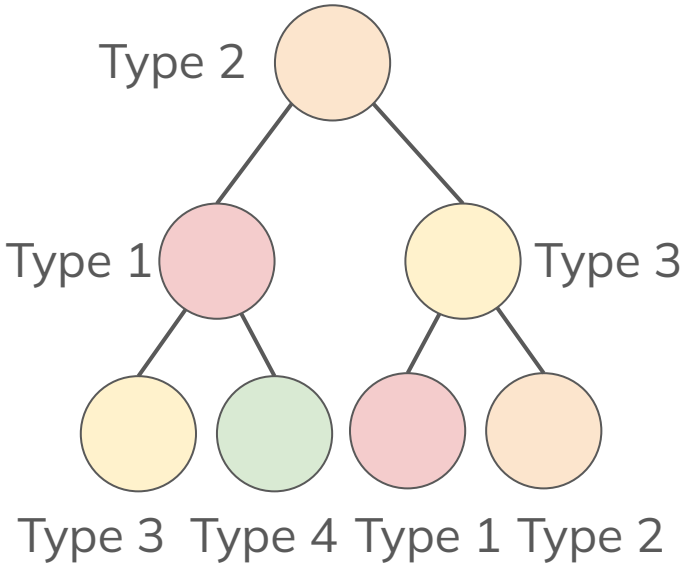
- Calculate initial answer by frequency array in O(N + K)
- If Alice controls node 0:
  - Add type Y: +1 to answer if originally no type Y, +1 to frequency array
  - Remove type Z: -1 to answer if originally exactly one type Z, -1 to frequency array
- If Bob controls node 0:
  - Add type Y: +1 to answer if originally N-2 type Y in leaf nodes, +1 to frequency array
  - Remove type Z: -1 to answer if originally N-1 type Z in leaf nodes, -1 to frequency array
  - Special handle type in root node
- Time complexity: O(N + K + Q)

Subtask 3 (16%): N, K ≤ 5000, Q = 0

- The maximum depth for the tree is more than 1
- We can shrink the depth of the tree step by step

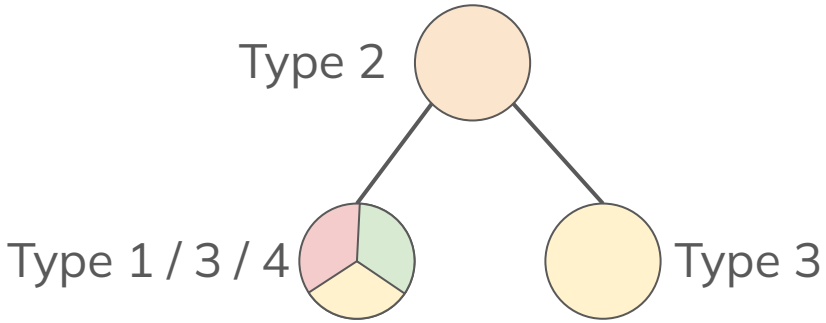
Subtask 3 (16%): N, K ≤ 5000, Q = 0

- The maximum depth for the tree is more than 1
- We can shrink the depth of the tree step by step



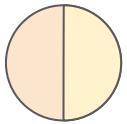
Subtask 3 (16%): N, K ≤ 5000, Q = 0

- The maximum depth for the tree is more than 1
- We can shrink the depth of the tree step by step



Subtask 3 (16%): N, K ≤ 5000, Q = 0

- The maximum depth for the tree is more than 1
- We can shrink the depth of the tree step by step

Type 2 / 3 

Subtask 3 (16%): N, K ≤ 5000, Q = 0

- We need to make sure we can visit type X no matter which child we go, so the original problem becomes subproblems of all children
- Maintain a boolean array DP[u][0 .. K - 1] of size K for each node u
- For nodes u controlled by Alice, DP[u][x] is true iff DP[v][x] is true for any child v
- For nodes u controlled by Bob, DP[u][x] is true iff DP[v][x] is true for all child v
- If you use std::bitset, you can simply do &= and |= on the bitsets
- Set DP[u][T[i]] to true
- Output number of truth values in DP[0][0 .. K - 1]
- Time complexity: O(NK)

Subtask 4 (9%): N, K, Q ≤ 5000

- When a node changes type, we only need to calculate the DP of relevant types again
- ~~Copy and paste the same code again and remove the for loop for types~~
- Use functions!
- E.g. calculate()
- Run calculate(i) for i in [0 .. K - 1] at first
- Run calculate(X) and calculate(Y) when changing a node from type X to type Y
- Time complexity: O(NK + NQ)

Subtask 5 (14%): N, K, Q ≤ 5×10^4, Complete binary tree

- When a node changes type, we only need to calculate the DP of relevant types again
- In fact, for a specific type, we only need to calculate the DP values from all nodes of the type to the root and all other DP values must be false
- When updating, only update DP of relevant types on the path from root to the modified node
- To speed up the “and” and “or” processes, use the frequency array from subtask 2 again
- Use the map to save memory on irrelevant types for nodes
- Time complexity: O(NH + QH) where H is the height of the tree

Subtask 6 (23%): $N, K \leq 5 \times 10^4, Q = 0$

- This subtask is unfortunately not that relevant to the full solution :(

Not so useful observation

Relevant types of each node is bounded by the subtree size of the node

Subtask 6 (23%): $N, K \leq 5 \times 10^4, Q = 0$

- By using small-to-large merging, we can ensure it runs in O(N log N) as long as we don't do anything with complexity O(largest child subtree size)
- For nodes with one child only, we only add the type of current node to the set of types of the only child
- For nodes controlled by Alice, insert the type of all other childs and the type of the current node to the set of types of largest child
- For nodes controlled by Bob, iterate through types in the smallest child subtree, check if it exists in all other childs **and break immediately if it does not exist in some child**
- Time complexity: $O(N \log^2 N)$

Subtask 6 (23%): $N, K \leq 5 \times 10^4, Q = 0$

Alternative solution (magic):

- Set all nodes with one child to be controlled by Alice
- In increasing i, if C[P[i]] = C[P[P[i]] = 0, set P[i] to P[P[i]]
- Run the O(NH) DP
- AC??
- Proof is left as exercise :)

Subtask 6 (23%): $N, K \leq 5 \times 10^4, Q = 0$

Alternative solution: bitset

- Time complexity: $O(NK / \omega)$

Subtask 7 - 8: Optimization

- To further simplify the solution, we will perform everything offline and calculate for each type one by one
- There are in total O(N + Q) important timestamps for all colors
  - Initial tree creates N timestamps (Add) in total
  - Every update creates 2 timestamps (Remove, add)
  - Finally, create N timestamps to remove all types
- In the following slides, we will only consider one single type and their add / remove operations
- Use difference array afterwards to obtain the answer for all Q + 1 days

Subtask 7 - 8: Optimization

Example with N = 5, K = 4, Q = 4, T = [0, 2, 3, 1, 2], X = [0, 3, 2, 4] and Y = [1, 2, 3, 0]

Type \ Day	0	1	2	3	4	5
0	+ 0	- 0			+ 4	- 4
1	+ 3	+ 0	- 3			- 0
2	+ 1, 4		+ 3		- 4	- 1, 3
3	+ 2			No change		- 2

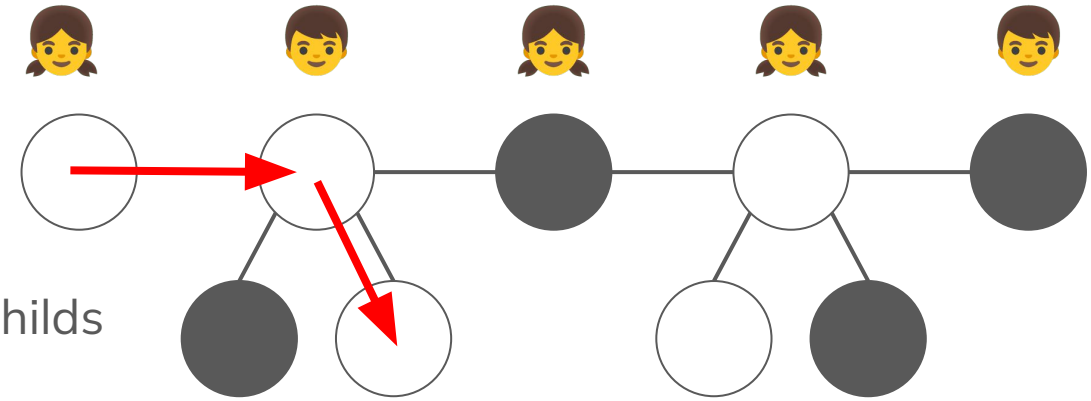
Subtask 7 - 8: Optimization

- Furthermore, we will use heavy-light decomposition
  - Every update only update DP values from the modified node to the root
  - Only O(log N) chains need to be updated
- Now, we can only consider the case of chain and further simplifying the problem
- For simplicity, in the following slides, we only consider updating one chain
- In practice, update the chain containing the parent of the head of the current chain after updating the current chain

Subtask 7 - 8: Optimization

Solid nodes: nodes of current type / light childs with the DP value true

<- root head tail leaf ->



DP of light childs

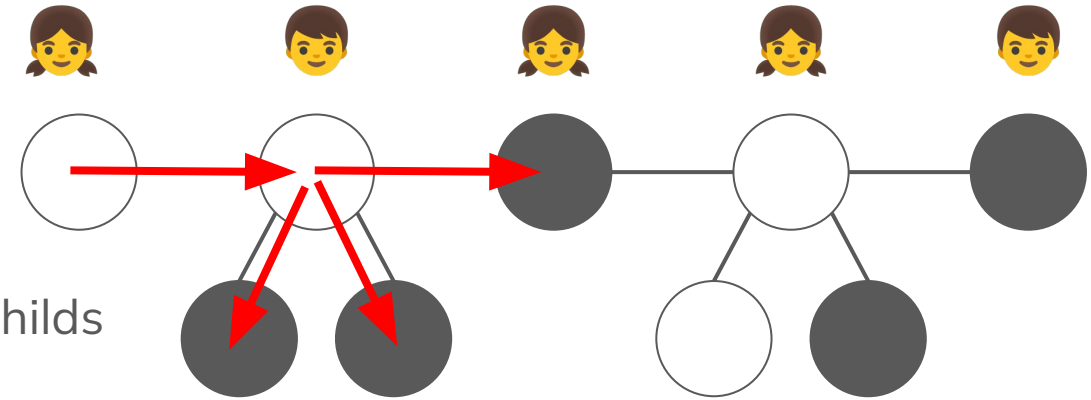
Is DP[head] true or false?

False

Subtask 7 - 8: Optimization

Solid nodes: nodes of current type / light childs with the DP value true

<- root head tail leaf ->



DP of light childs

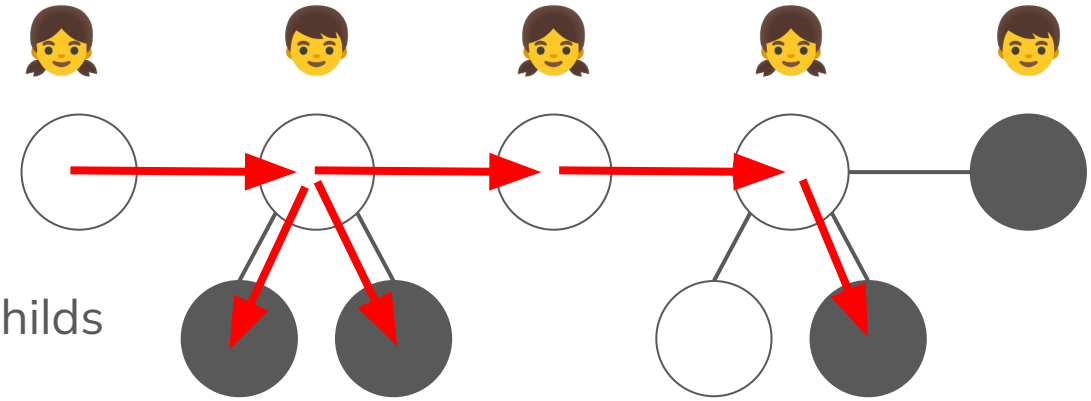
Is DP[head] true or false?

True

Subtask 7 - 8: Optimization

Solid nodes: nodes of current type / light childs with the DP value true

<- root head tail leaf ->



DP of light childs

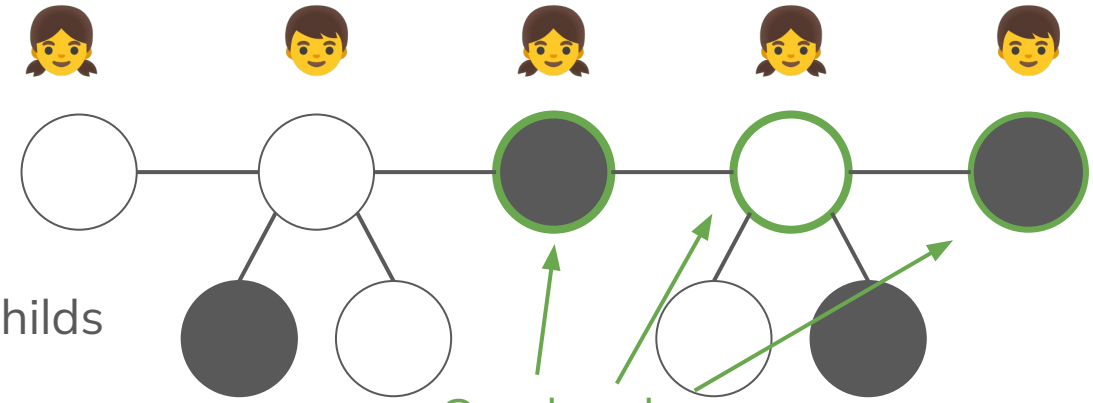
Is DP[head] true or false?

True

Subtask 7 - 8: Optimization

Solid nodes: nodes of current type / light childs with the DP value true

<- root head tail leaf ->

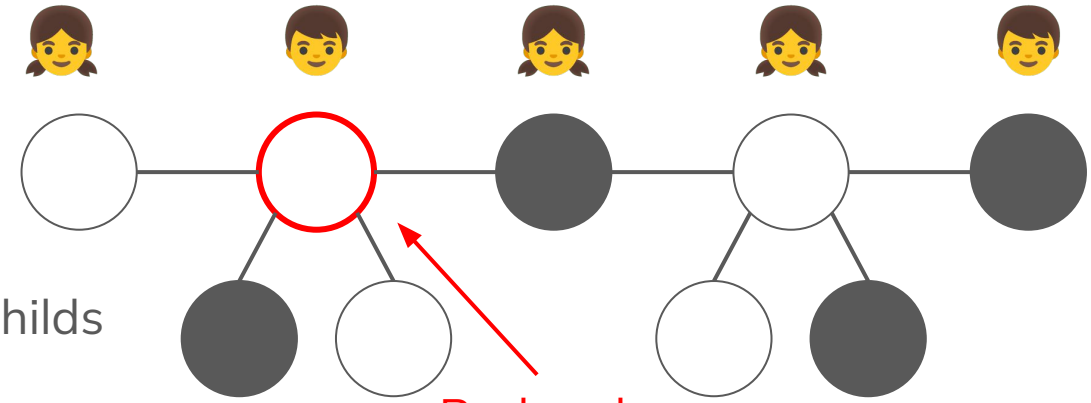


We define good nodes as nodes of current type **or** nodes controlled by Alice with at least one light child with the DP value true

Subtask 7 - 8: Optimization

Solid nodes: nodes of current type / light childs with the DP value true

<- root head tail leaf ->



We define bad nodes as nodes not of the current type **and** controlled by Bob with at least one light child with the DP value false and

Subtask 7 - 8: Optimization

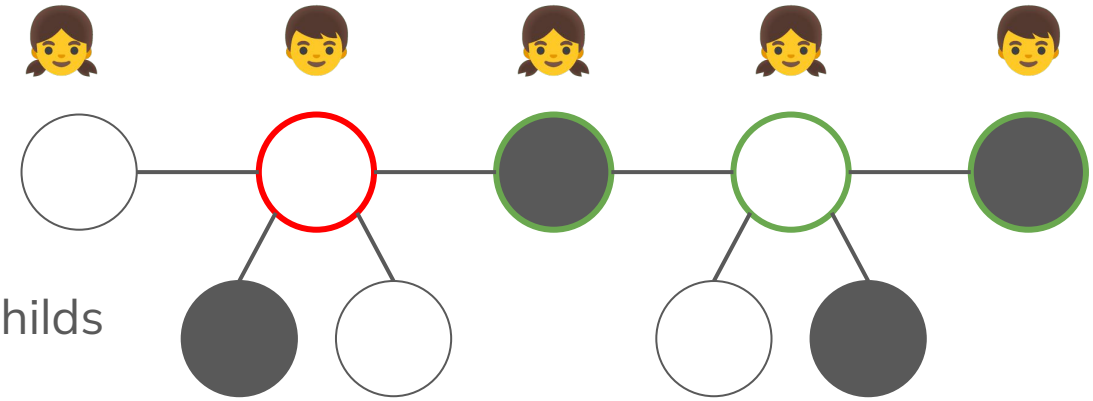
Key observation

DP[head] is true iff the first good node appears before the first bad node

Subtask 7 - 8: Optimization

Solid nodes: nodes of current type / light childs with the DP value true

<- root head tail leaf ->



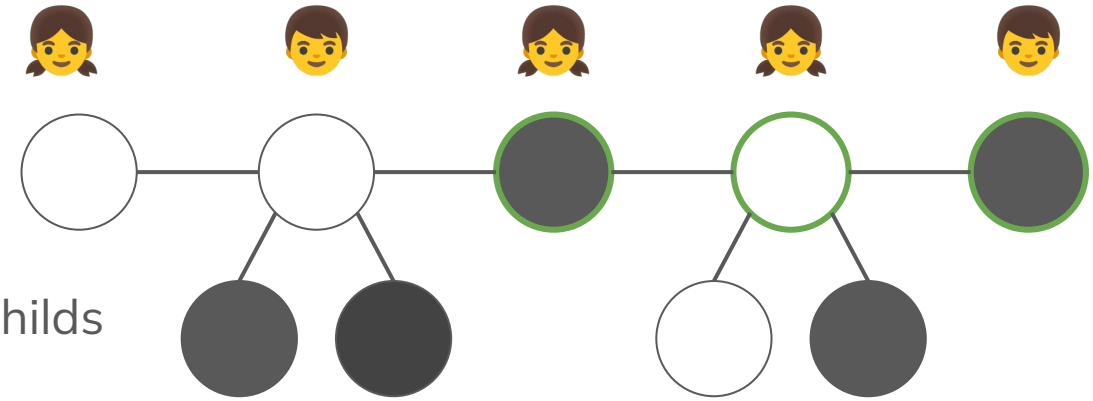
DP of light childs

False

Subtask 7 - 8: Optimization

Solid nodes: nodes of current type / light childs with the DP value true

<- root head tail leaf ->



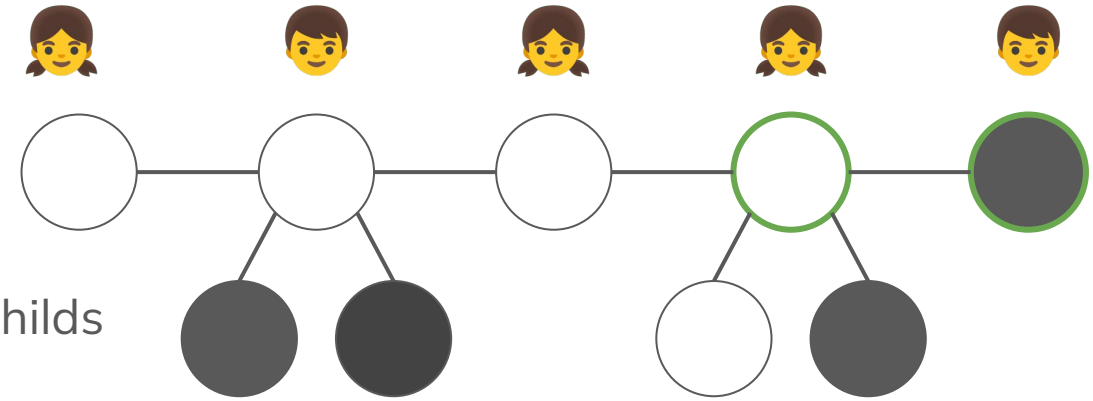
DP of light childs

True

Subtask 7 - 8: Optimization

Solid nodes: nodes of current type / light childs with the DP value true

<- root head tail leaf ->



DP of light childs

True

Subtask 7 - 8: Optimization

- What we need to maintain now:
- Trivial informations
  - Who controls the node
  - Is the node a node of current type
- Number of light childs with DP value true — Frequency array
- Position of first good node — Set
- Position of first bad node — Set

Time complexity: $O((N + Q) \log^2 N)$

Subtask 8 (4%): No additional constraints

If you need to optimize your solution further:

- Use set instead of segment tree
- Use the magic from Subtask 6 alternate solution
- Do not update the chain containing the parent of the head of the current chain if DP[head] is unchanged
- Use one set for each chain instead of one set for the whole tree
- With all optimizations, the solution actually runs in O((N + Q) log N)

Alternative Solution by firewater

- Offline the types as usual
- Segment tree on time
- DSU with rollback by maintaining the “dependencies”
  - DP value of a node controlled by Bob is same as the DP value of the remaining child if all other childs have DP value of 1 already
- ?????
- AC

Takeaways

- Think about easy case first
  - Can you solve the problem for star and chain? (or in this task: near-chain)
- Reduce the problem to simpler subproblems
  - Handling one type only is easier than handling all K types
- Do not ignore alternative solutions!
  - Did you understand the HLD solution for M2633 - Marathon?
- Be good at using std data structures
  - Using std::bitset or std::set can save you time (and runtime)