T2621 Chained Together

Ethen Yuen {ethening}

2026-05-24

Background

- Problem Idea by `ethening`
- Preparation by `__jk__`, `bedrockfake`
- Special Thanks to `QwertyPi`, `WongChun1234`, `__declspec` for testing!

- Inspired by 3D platformer (ragebait) game: **Chained Together**!

The Problem

Given a tree with N nodes

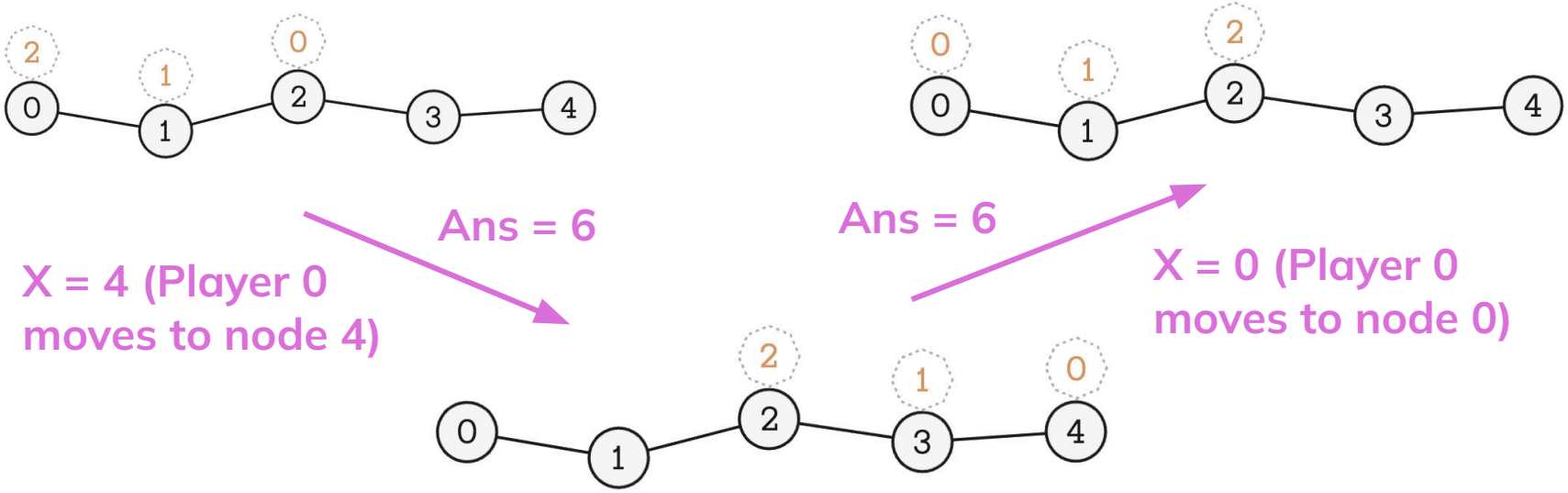
- There are M players initially at node A[0], A[1], ..., A[M - 1]
- Distance between player j and player (j + 1) **must not exceed K at all times**

Support Q queries online:

- Each operation, a player can move to an adjacent node
- Given a target node X, find the minimum number of operations for player 0 to move to X
- Each query is persistent. The player stay in their new position and start the new query.

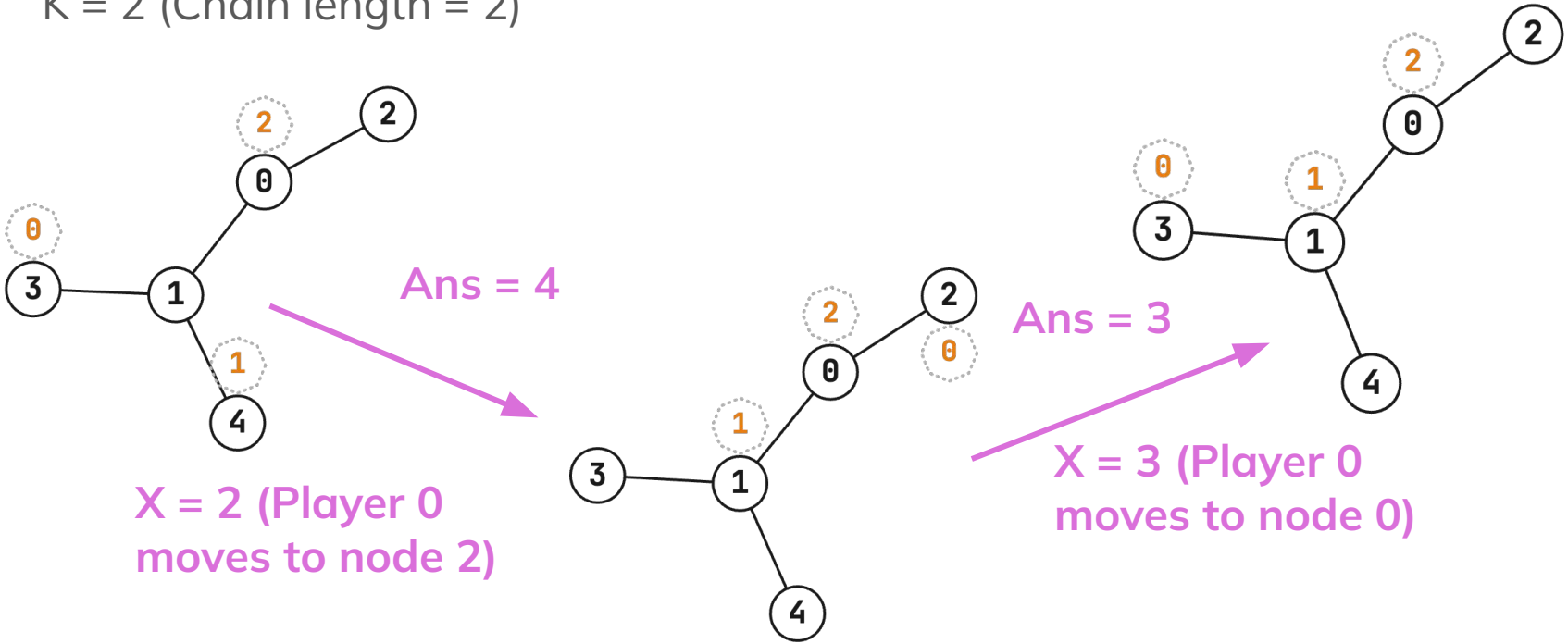
Examples

- K = 1 (Chain length = 1)



Examples

- K = 2 (Chain length = 2)



Subtasks

#	Score	N, M, Q	Constraints
1	6	≤ 100	Chain
2	7	≤ 500	
3	10	≤ 2500	
4	18	≤ 1e5	Chain
5	21	≤ 1e5	K = 1, Node X is adj.
6	15	≤ 1e5	Node X is adj.
7	16	≤ 1e5	
8	7	≤ 2e5	

Statistics

- We hope there would be at least 1 AC...
- First solved by:
- **Unfortunately, no one!**
- Choy Tsz Chai Berton (s20146) got **93 (nearly solved!)** in 3 hr 3 mins 23 secs.

#	Score	# Contestants
1	6	53
2	7	34
3	10	21
4	18	2
5	21	4
6	15	3
7	16	1
8	7	0

Observation 1: Physical Intuition

Before we look at the subtasks, let's understand the physics of the chain.

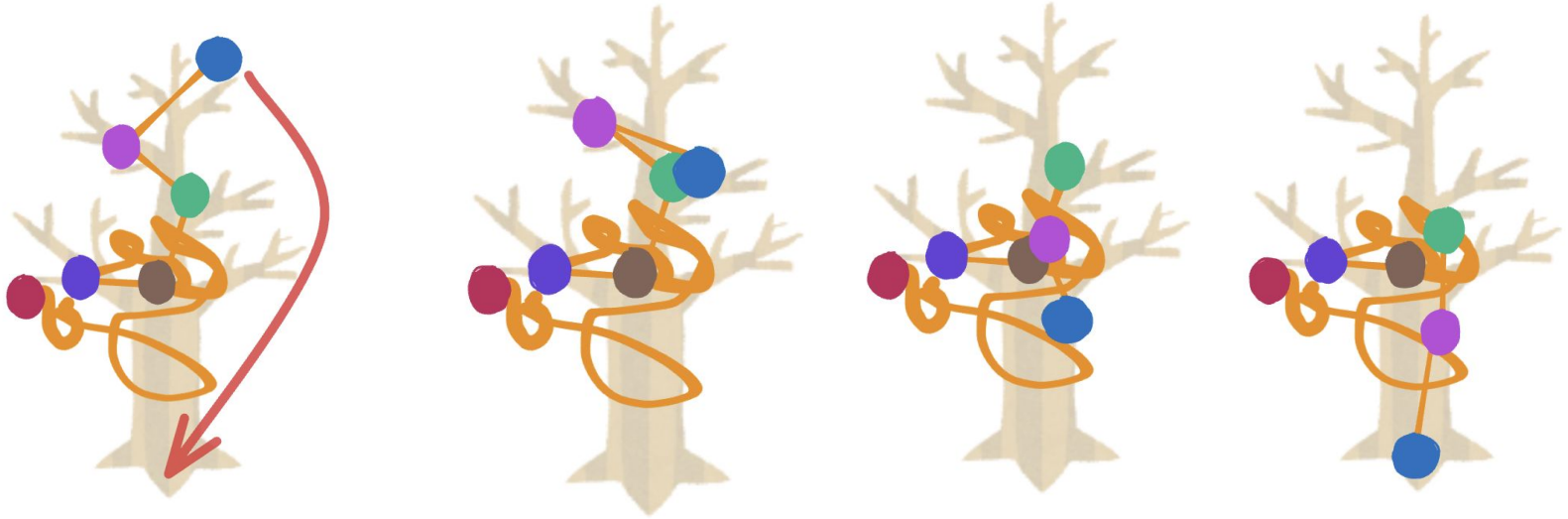
- Imagine the players are tied together by a physical chain of length K

- In the beginning, some chain are **tense** (pulled to its maximum extend (i.e. length K))
- Some chain are **loose** (the position of the player is closer than K)

Observation 1: Physical Intuition

Before we look at the subtasks, let's understand the physics of the chain.

- When Player 0 is dragged to X, it may stretched the chain and make the chain **tense** and dragged more players along the way.



Observation 1: Physical Intuition

Greedy Strategy: Everyone only moves as much as they are forced to!

Let **B_i** be the optimal final position of player i

- **B_0 = X**
- **B_i** = closest node to A_i that is within distance K of B_i-1

Observation 1: Physical Intuition

Greedy Strategy: Everyone only moves as much as they are forced to!

- **B_i** = closest node to A_i that is within distance K of B_i-1

This greedy strategy, turns out, is **optimal, unique and always achievable** while distance between adjacent players not exceed K at all times.

- This conclusion could be guessed during contest
  - The proof we have is not that straightforward
- Intuitively speaking, because players are pulled sequentially from one end, they will never deadlock!

Subtask 1 & 2 (13%): N, M, Q <= 500

- We can directly apply Observation 1 to simulate the movement!
- For each player i from 1 to M - 1, we need to naively find the closest valid node to B_i-1 from their current position A_i

Subtask 1 & 2 (13%): N, M, Q <= 500

- Since N, M, Q are small, we can just run a DFS from B_i-1 to search for Ai
  - Maintain the current path in a stack
  - When we reach A_i, we can get the optimal solution from the stack
    - If length <= K, B_i = A_i
    - Otherwise, B_i = stack[K]

Time Complexity: O(NMQ)

Expected Score: 13

Subtask 3 (10%): N, M, Q <= 2500

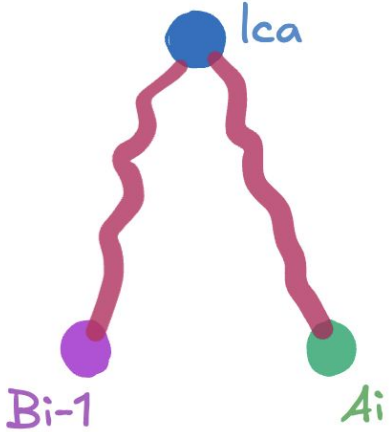
- In previous solution, we are frequently recomputing the path for every single player every single query.
- How can we find the **K-th node on the path quickly from B_i-1 to Ai** without searching the whole tree?

Subtask 3 (10%): N, M, Q <= 2500

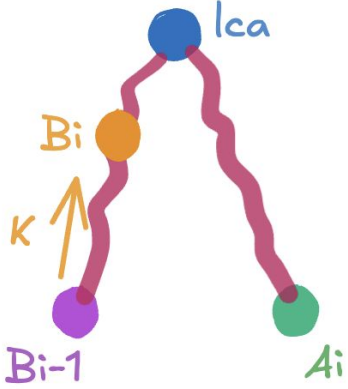
- We know that we can query distance on tree if we precompute binary lifting table to get LCA.
  - dist(u, v) = depth(u) + depth(v) - 2 * depth(LCA)
- If dist(B_i-1, A_i) <= K, no need to move!
- Otherwise, we can simply get the K-th node from path B_i-1 to A_i using the binary lifting table we already computed!

Subtask 3 (10%): N, M, Q <= 2500

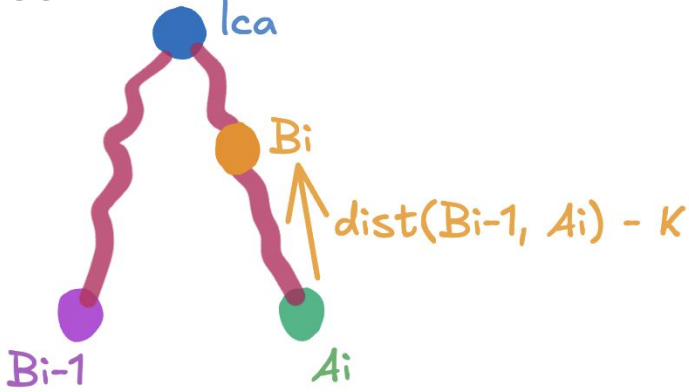
- We can simply get the K-th node from path B_i-1 to A_i using the binary lifting table we already computed! We just split into two cases and turned into a finding ancestor problem.



Case 1:



Case 2:



Subtask 3 (10%): N, M, Q <= 2500

- If dist(B_i-1, A_i) <= K, no need to move!
- Otherwise, B_i = K-th node from path B_i-1 to A_i, computed with binary lifting + LCA

Time Complexity: Precompute O(N log N), Query O(QM log N)

Expected Score: 23

Observation 2: Who Actually Moves?

You may have noticed the **chain reaction:**

- If player i does not move, then player i+1 feels absolutely no pull, and will never move!

Observation 2: Who Actually Moves?

This means in every query, the players who move will always be a **prefix** (0..p)

- The remaining players (p + 1 ... M - 1) do not move at all.
- We can also see that after the prefix player move, it always left the chains of the prefix **completely tense**.
  - Completely tense means that their position is in some kind of structure
  - When there are structure, we may be able to exploit that structure!

Subtask 4 (18%): Chain

- How do we find the prefix p efficiently on a chain?
- Assume we move right (X > A_0). Player i will be pulled if they end up too far from X:
  - `X - A_i > i * K`
  - `A_i + i * K < X` (We let C_i = `A_i + i * K`)
- Because original chain is valid (A_i - A_i+1 <= K), we can easily prove that **C_i is always non-decreasing**.
  - Since C_i is sorted, finding the prefix p where C_p < X is just a **binary search**.
  - We can also define `D_i = A_i - i * k`. With similar proof, we can show that D_i is always non-increasing, and for moving left, we just need to do binary search on D_p > X

Subtask 4 (18%): Chain

Summary: The followings are monotonic invariants. Given the original A_i is valid, these are always true:

- C_i = A_i + i * K is always **non-decreasing**
- D_i = A_i - i * K is always **non-increasing**

Each query, we need to:

- Find the prefix p (which player moves) by binary searching on C_i or D_i
- Calculating the answer to query with sum(A[0] .. A[p]) and p
- Updating C_i and D_i and A_i for [0..p]

How to do this?

Subtask 4 (18%): Chain

How to do this?

- Lazy Segment Tree w/ range assign arithmetic progression & range sum

Time Complexity: O(Q log M) [note: 2 logs if you do binary search on seg tree naively]

Expected Score: 24 (Cumulative: 41)

Subtask 5 (21%): K = 1, Node X is adjacent

Back to general trees. The prefix property (observation 2) still hold. Since in this subtask K = 1, each query answer = p + 1.

Let's define a **"tense block"** as consecutive range of players where the chain between them is completely tense (at maximum range K) and lie on a simple path.

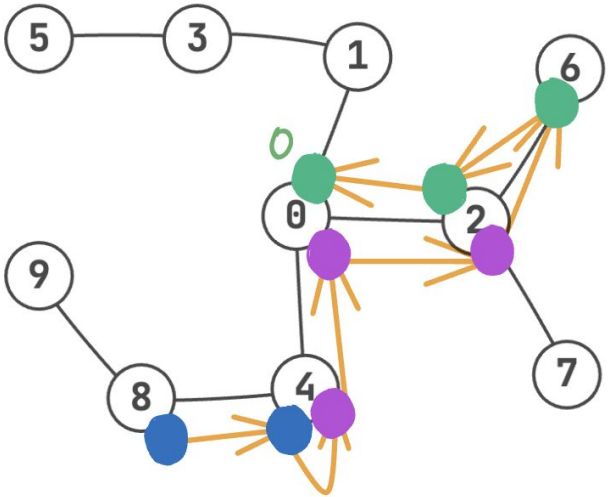
When player 0 moves, depending on its move direction, either

1. Only player 0 moves.
2. The whole *maximum* tense block containing player 0 moves.

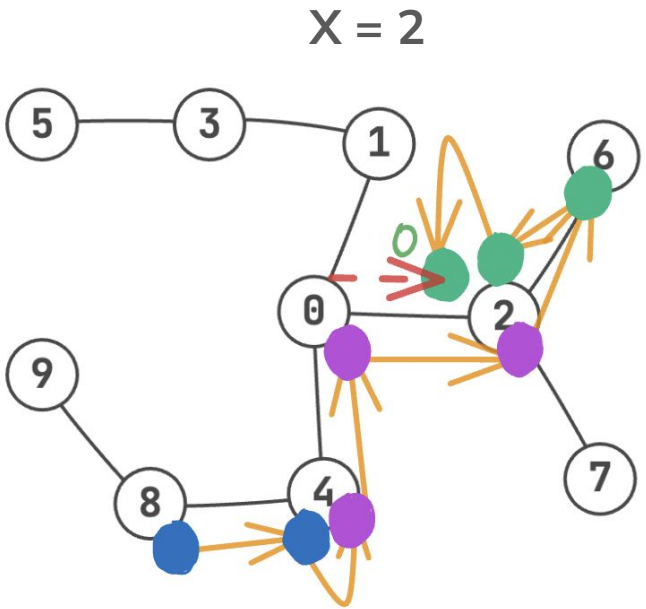
Subtask 5 (21%): K = 1, Node X is adjacent

When player 0 moves, depending on its move direction, either

1. **Only player 0 moves**



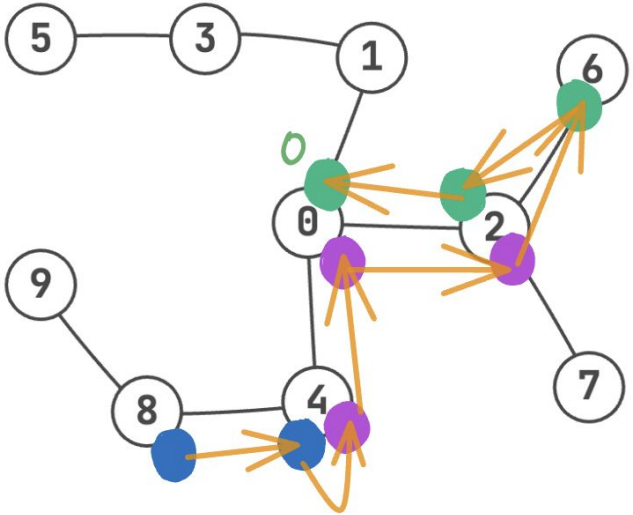
X = 2

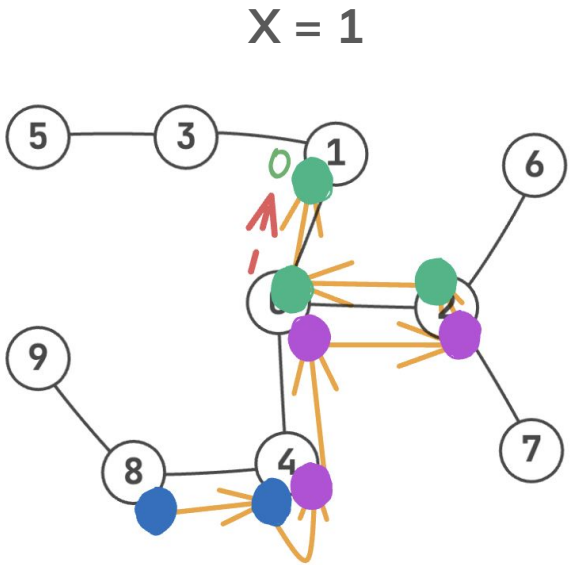


Subtask 5 (21%): K = 1, Node X is adjacent

When player 0 moves, depending on its move direction, either

2. **The whole maximum tense block containing player 0 moves.**





Subtask 5 (21%): K = 1, Node X is adjacent

When player 0 moves, depending on its move direction, either

1. Only player 0 moves **(when player 0 moves towards 2nd node of its block)**
2. The whole maximum tense block containing player 0 moves **(otherwise)**

Subtask 5 (21%): K = 1, Node X is adjacent

We can maintain all **maximum tense block** with a stack (e.g. std::vector)

- Each block contains the start and end node of the path, and node count

Each query, we only need to update O(1) block. For each of the two cases:

1. Update first block to exclude 0 + add new block with only 0
2. Update first block start and end.
   * If at this point the second block's first node can extend first block's path, need to extract first node from second block and merge to first block. (To preserve the maximality)

(K-th node from path u to v helper function from subtask 3 is helpful)

Subtask 5 (21%): K = 1, Node X is adjacent

- Maintain stack of maximal tense block
- Update O(1) block per query
  - Use K-th node from path u to v helper -> using LCA

Time Complexity: O(Q log N)

Expected Score: 21 (Cumulative: 62)

Subtask 6 (15%): Node X is adjacent

K = 1 constraint is dropped

What is same as the previous solution?

- because X moves by one edge, each moved player moves at most one edge
- each query only affects the first block and maybe the first node of the second block

What part about the previous solution doesn't work anymore?

- second block's first node can extend first block's path
- + second block first node **has a tense chain with first block last node**

Subtask 6 (15%): Node X is adjacent

What part about the previous solution doesn't work anymore?

- second block's first node can extend first block's path
- + second block first node **has a tense chain with first block last node**

Actually, if you think about it, whether or not to extract second block's first node (let's call it player i) to first block is actually asking:

- Is player i being pulled when player 0 move to node X?
- dist(X, A_i) <= i * K: No
- dist(X, A_i) > i * K: Yes

Subtask 6 (15%): Node X is adjacent

- Maintain stack of maximal tense block
- Update O(1) block per query
  - Use K-th node from path u to v helper -> using LCA
  - When combining node from second block, check whether it will get pulled by player 0

Time Complexity: O(Q log N)

Expected Score: 36 (Cumulative: 77)

Subtask 7 (16%): N, M, Q <= 1e5

What part about the previous solution doesn't work anymore?

- Player 0 can move multiple steps per query, so prefix p = answer is not true anymore.
- When a prefix of player is pulled, the part of the chain still become completely tense.
- However, we cannot determine the prefix easily by looking at O(1) block anymore! How can we easily determine which part is being pulled?

Subtask 7 (16%): N, M, Q <= 1e5

From subtask 6, when we determine if player i of the second block being pulled when player 0 move to node X?

- dist(X, A_i) <= i * K: No
- dist(X, A_i) > i * K: Yes

Does this seems like something occurred in subtask 4 (chain)?

Subtask 7 (16%): N, M, Q <= 1e5

Does this seems like something occurred in subtask 4 (chain)?

Viewing from node X (the target):

- Player i is pulled if dist(X, A_i) > i * K
- Let us define V_i = dist(X, A_i) - i * K, so if V_i > 0, player i is pulled

Is V_i non-increasing like the chain subtask?

Subtask 7 (16%): N, M, Q <= 1e5

Is V_i non-increasing like the chain subtask?

- By Triangle Inequality:
  dist(X, A_i) <= dist(X, A_i-1) + dist(A_i-1, A_i)
- Since chain is valid:
  dist(A_i-1, A_i) <= K
- **dist(X, A_i)** <= dist(X, A_i-1) + dist(A_i-1, A_i) <= **dist(X, A_i-1) + K**
- **dist(X, A_i) - i *K <= dist(X, A_i-1) - (i - 1) * K**
- **V_i <= V_i-1**

The moved players are exactly the prefix where V_i > 0.

Subtask 7 (16%): N, M, Q <= 1e5

Let each tense block (note, no need maximal here) be represented by (L, R, U, V)

- “Player L to R are spaced exactly K apart on the simple path from node U to V”
- We can again, maintain the blocks in a stack.
- When Player 0 moves to X: We loop through each block
  - If the whole block moves (V_R > 0), pop it
  - Otherwise, binary search on the block, split it, and pop the moving half
  - Calculate the contributions to the answer for each pop block
  - Finally, push one new block for players 0..p starting from X
- We only add 1 new block per query, so the amortized number of pop is O(1)!

Subtask 7 (16%): N, M, Q <= 1e5

"Calculate the contributions to the answer for each pop block"
How?

Suppose we are popping a block (L, R, U, V), we need to compute
sum(i=L..R) dist(X, A_i)

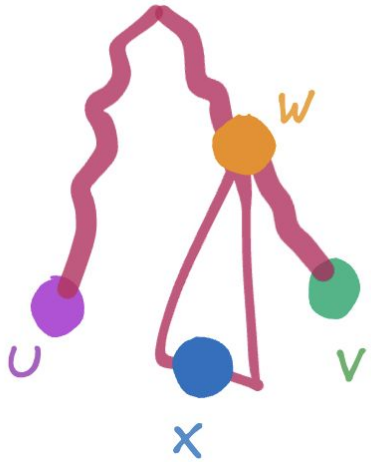
Why not directly find **dist(B_i, A_i), the actual answer**?

- We know that dist(X, B_i) + dist(B_i, A_i) = dist(X, A_i)
- We also know that for popped block, dist(X, B_i) = i * K
- So knowing sum(i=L..R) dist(X, A_i) is enough for calculating the actual answer.

Subtask 7 (16%): N, M, Q <= 1e5

Suppose we are popping a block (L, R, U, V), we need to compute **sum(i=L..R) dist(X, A_i)**

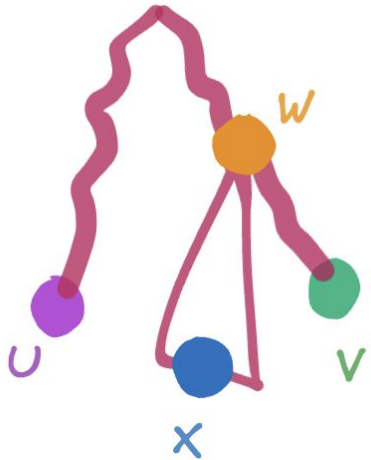
- We want to first find the **projection** (let's call it W) of X on path U, V, where W is the closest node to X and lie on U-V.
- There is a quite common trick here:
  LCA(U, V), LCA(U, X), LCA(V, X)
- The one with the maximum depth is the projection



Subtask 7 (16%): N, M, Q <= 1e5

Suppose we are popping a block (L, R, U, V), we need to compute **sum(i=L..R) dist(X, A_i)**

- Once we find W, the distances to the players of the block split perfectly into two Arithmetic Progressions. We can calculate the sum using O(1) formulas.



Subtask 7 (16%): N, M, Q <= 1e5

- Maintain stack of tense block (L, R, U, V)
- Determine if player x is being pulled by checking V_i = dist(X, A_i) - i * K > 0
- Pop full blocks if player R is pulled
- Otherwise, binary search and split the block
- Calculate the contribution of each popped block by projection + sum of AS
- Add a new block, U = X and V = Kth_Node(X, A_p, p * K)

Subtask 7 (16%): N, M, Q <= 1e5

- Maintain stack of tense block (L, R, U, V)
- Determine if player x is being pulled by checking dist(X, A_i) - i * K > 0
  - O(log N) from dist()
- Pop full blocks if player R is pulled
  - O(1)
- Otherwise, binary search and split the block
  - O(log N) from binary search
    - O(log N) to get the actual node from Kth_Node(U, V, mid * K) & calculating dist(X, A_i)
- Calculate the contribution of each popped block by projection + sum of AS
  - O(log N) for projection
- Add a new block, U = X and V = Kth_Node(X, A_p, p * K)
  - O(log N) from Kth_Node

Subtask 7 (16%): N, M, Q <= 1e5

- Binary search and split the block
  - O(log N) from binary search
    - O(log N) to get the actual node from Kth_Node(U, V, mid * K) & calculating dist(X, A_i)

- Time Complexity: O(Q log^2 N)
- Expected Score: 93

Subtask 8 (7%): No additional constraints

- Binary search and split the block
  - O(log N) from binary search
    - O(log N) to get the actual node from Kth_Node(U, V, mid * K) & calculating dist(X, A_i)

- This is the bottleneck of the whole algorithm. Let's try to optimize it.

Subtask 8 (7%): No additional constraints

- First, dist(X, A_i) time complexity is based on LCA time complexity
- **Why is LCA log(N)?** Binary lifting

Subtask 8 (7%): No additional constraints

- First, dist(X, A_i) time complexity is based on LCA time complexity
- **Why is LCA log(N)?** Binary lifting

- There exists **O(1) LCA**
- **Graph (III):**

Problem 2 Solution:

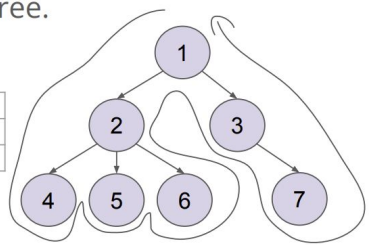
Store the value of nodes in an array using the Euler Tour of the tree.
Then values of nodes in a subtree is contiguous in the array.

This reduces (flattens) the problem to range update, point query problem, which can be solved with data structures like segment tree.

For example, the subtree of node 2 is shown below:

Index	0	**1**	2	3	4	5	6	7	**8**	9	10	11	12	13
Order	1	**2**	4	4	5	5	6	6	**2**	3	7	7	3	1
Value	0	**0**	0	0	0	0	0	0	**0**	0	0	0	0	0

width = subtree size * 2



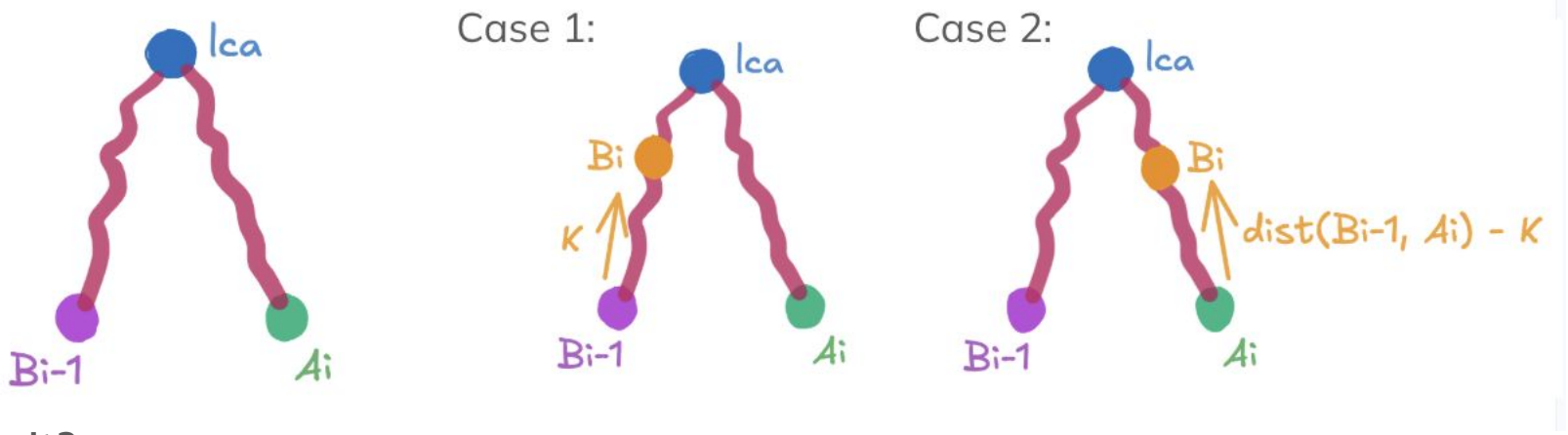
Subtask 8 (7%): No additional constraints

- First, dist(X, A_i) time complexity is based on LCA time complexity
- **Why is LCA log(N)?** Binary lifting

- There exists **O(1) LCA**
- We can store the euler tour of the whole tree. Each entry & exit mark down the depth of the node.
- Then LCA becomes min(depth[euler[U] .. euler[V])), which can be preprocessed with a sparse table. Then, we can do O(1) min with sparse table.

Subtask 8 (7%): No additional constraints

- Next, Kth_Node(U, V, mid * K) time complexity is based on binary lifting from U or V to find the exact ancestor.
- The lifting part always take O(log N)



- Or is it?

Subtask 8 (7%): No additional constraints

- Next, Kth_Node(U, V, mid * K) time complexity is based on binary lifting from U or V to find the exact ancestor.
- The lifting part always take O(log N)? **There exists O(1) Kth_Ancestor**

Subtask 8 (7%): No additional constraints

- Next, Kth_Node(U, V, mid * K) time complexity is based on binary lifting from U or V to find the exact ancestor.
- The lifting part always take O(log N)? **There exists O(1) Kth_Ancestor**
- Long Chain Decomposition (known as 長鏈剖分 in China)
  - Similar to Heavy-light decomposition but instead of defining heavy child as the child with largest subtree size, it define heavy child as the child with longest chain

Subtask 8 (7%): No additional constraints

- Next, Kth_Node(U, V, mid * K) time complexity is based on binary lifting from U or V to find the exact ancestor.
- The lifting part always take O(log N)? **There exists O(1) Kth_Ancestor**
- ~~Long Chain Decomposition (known as 長鏈剖分 in China)~~
  - ~~Similar to Heavy-light decomposition but instead of defining heavy child as the child with largest subtree size, it define heavy child as the child with longest chain~~

- **But actually it is not required.**
  - The O(log N) is to get the actual node from Kth_Node(U, V, mid * K) & calculating dist(X, A_i)
  - We don't have to **actually know the node** in order to binary search

Subtask 8 (7%): No additional constraints

- The O(log N) is to get the actual node from Kth_Node(U, V, mid * K) & calculating dist(X, A_i)
- We don't have to **actually know the node** in order to binary search

- dist(X, A_i) = dist(X, W) + abs(dist(U, W) - dist(U, A_i)), where W is projection
  - dist(X, W) -> distance from X to projection, O(1) LCA
  - dist(U, W) -> distance from block start to projection, O(1) LCA
  - dist(U, A_i) -> distance from block start to some middle node. We can just get this from the definition of the block: **mid * K**

Subtask 8 (7%): No additional constraints

- Maintain stack of tense block (L, R, U, V)
- Determine if player x is being pulled by checking dist(X, A_i) - i * K > 0
- Pop full blocks if player R is pulled
- Otherwise, binary search and split the block
  - **Project X on the block first**
  - **Then, calculate dist(X, A_i) - i * K by calculating the offset to U**
- Calculate the contribution of each popped block by projection + sum of AS
- Add a new block, U = X and V = Kth_Node(X, A_p, p * K)

- Time Complexity: O(Q log N)
- Expected Score: 100

Further Improvements (Optional)

- Otherwise, binary search and split the block
  - **Project X on the block first**
  - **Then, calculate dist(X, A_i) - i * K by calculating the offset to U**

- Actually, binary search is not needed, there actually exist a closed formed formula to find the split index (let say the Yth node in the block)
  - We know dist(X, A_i) = dist(X, W) + abs(dist(U, W) - dist(U, A_i))
  - dist(X, A_i) = dist(X, W) + abs(dist(U, W) - Y * K)
  - V_i = dist(X, W) + abs(dist(U, W) - Y * K) - i * K > 0 (last i that is pulled)
  - Suppose Y * K <= dist(U, W) [Y is on the left side of the projection]
    - dist(X, W) + dist(U, W) - Y * K - i * K > 0
    - dist(X, W) + dist(U, W) - Y * K - (prev + Y) * K > 0
    - dist(X, W) + dist(U, W) - prev * K > 2 * Y * K

Further Improvements (Optional)

- Actually, binary search is not needed, there actually exist a closed formed formula to find the split index (let say the Yth node in the block)
  - Suppose Y * K <= dist(U, W) [Y is on the left side of the projection]
    - dist(X, W) + dist(U, W) - Y * K - i * K > 0
    - dist(X, W) + dist(U, W) - Y * K - (prev + Y) * K > 0
    - dist(X, W) + dist(U, W) - prev * K > 2 * Y * K
    - Y < (dist(X, W) + dist(U, W) - prev * K) / (2 * K)
    - Y = ceil((dist(X, W) + dist(U, W) - prev * K) / (2 * K)) - 1
- What about if Y is on the right side of the projection? (i.e. Y * K > dist(U, W))
  - dist(X, W) - dist(U, W) + Y * K - i * K > 0
  - dist(X, W) - dist(U, W) + Y * K - (prev + Y) * K > 0
  - dist(X, W) - dist(U, W) - prev * K > 0
  - The condition is independent of Y! The means either every index on the right side can move, or all of them stay! But we know that R is staying (else we will pop the whole block), so this case is not happening!

Further Improvements (Optional)

- Hence, we can obtain the offset with this closed form.
  - Y = ceil((dist(X, W) + dist(U, W) - prev * K) / (2 * K)) - 1

The procedure then becomes:

- Maintain stack of tense block (L, R, U, V)
- Determine if player x is being pulled by checking dist(X, A_i) - i * K > 0
- Pop full blocks if player R is pulled
- **Otherwise, split the block with projection W**
  - **+ closed form formula ceil((dist(X, W) + dist(U, W) - prev * K) / (2 * K)) - 1**
- Calculate the contribution of each popped block by projection + sum of AS
- Add a new block, U = X and V = Kth_Node(X, A_p, p * K)

Further Improvements (Optional)

- The O(log N) in binary search is gone.
- Adding in Long Chain Decomposition to optimize Kth_Node() to O(1)
- We can optimize the whole program down to O(Q)

- Time Complexity: O(Q)
- Expected Score: 100

Takeaways

- Always play around with example (beyond the given ones) when the rules of the problems seems to be complex, you may deduce some easier constraints from it.
- Exploit structuality
  - In this task, the most important observation is that players with tense chain to the front move together and this structure is kept after movement.
  - This should immediately hint you to think of grouping players together
- You do not have to prove things rigidly during contest
  - Develop a good intuition to know “I can probably prove this is true” without actually spending time to prove it.
  - Although you may want to generate some random cases to verify your belief before wasting time on a wrong assumption.