1 APIOBike

1.1 Subtask 1

In this subtask, the road network is a path with stations numbered $0, 1, ..., N-1$ in sequential order. Every station i satisfies $A[i] \neq B[i]$, and there is exactly one station s that initially contains all bikes.

Our goal is to redistribute bikes from station s to all other stations. The rebalancing truck starts at s and loads $A[s] - B[s]$ bikes. The number of bikes on the truck then equals $\sum_{i \neq s} B[i]$. As the truck visits each station $i \neq s$, the driver unloads exactly $B[i]$ bikes. To complete the rebalancing, the truck must visit every station at least once.

To minimize the travelling distance while visiting all stations, we consider two candidate walks:

- $X = [s, s+1, ..., N-2, N-1, N-2, ..., 0]$.
- $X = [s, s-1, ..., 1, 0, 1, ..., N-1]$.

The optimal answer is the shorter of these two sequences.

1.2 Subtask 2

We observe that there always exists a rebalancing strategy with a distance $\leq 2N-2$. In this subtask, the tree is a path and N is small, allowing us to enumerate all possible walks. There are at most $N \times 2^{2N-2}$ such walks.

To check if a station sequence X is valid, we greedily construct Y as follows: For each station i, let its visited steps be $t_1, t_2, ..., t_k$ in chronological order.

- At t_1, set $Y[t_1] = -A[i]$ (load all initial bikes onto the truck).
- At t_k, set $Y[t_k] = B[i]$ (unload the target number of bikes at the station).
- If $t_1 = t_k$, then $Y[t_1] = B[i] - A[i]$.
- For all other steps t_j, set $Y[t_j] = 0$.

This construction ensures that the number of bikes at station i never drops below zero. We then verify if the truck always maintains a non-negative bike count, which is equivalent to checking if every prefix sum of Y is non-positive. It takes $O(N)$ time to check if the sequence Y is valid.

By finding the valid strategy with the minimum travelling distance, the overall time complexity is $O(N^2 2^{2N})$.

1.2.1 Correctness proof

Observation 1.1. If the input tree is a path with stations $0, 1, ..., N-1$ in sequential order, there exists a rebalancing strategy with travelling distance $2N-2$.

Proof. Consider the strategy: The truck starts at station 0, and collects all bikes at $0, 1, ..., N-1$. At this point, all bikes are on the truck. The truck then travels back and unloads bikes at stations $N-1, N-2, ..., 0$ to ensure each station i has exactly $B[i]$ bikes. □

Observation 1.2. If there exists a rebalancing strategy with station sequence X, then there exists a rebalancing strategy (X, Y) where Y is constructed as described above.

Proof. Let Y be the sequence constructed as described above. We aim to show that Y is valid. Suppose (X, Y') is any arbitrary valid strategy.

For each step t, we want to show that $\sum_{t' \le t} Y[t'] \le \sum_{t' \le t} Y'[t'] \le 0$. We consider the contribution to the sum from each station j:

- If station j is not visited again after step t, then

$$\sum_{\substack{t' \le t \\ X[t']=j}} Y[t'] = \sum_{\substack{t' \le t \\ X[t']=j}} Y'[t'] = A[j] - B[j].$$

- If station j has not been visited at all by step j, then

$$\sum_{\substack{t' \le t \\ X[t']=j}} Y[t'] = \sum_{\substack{t' \le t \\ X[t']=j}} Y'[t'] = 0.$$

- If station j has been visited at least once and will be visited again later, then

$$\sum_{\substack{t' \le t \\ X[t']=j}} Y[t'] = -A[j] \le \sum_{\substack{t' \le t \\ X[t']=j}} Y'[t']$$

  because the bike count at the station $A[j] + \sum_{t' \le t} Y'[t']$ must be ≥ 0.

Summing over all stations j:

$$\sum_{t' \le t} Y[t'] = \sum_{j=0}^{N-1} \sum_{\substack{t' \le t \\ X[t']=j}} Y[t'] \le \sum_{j=0}^{N-1} \sum_{\substack{t' \le t \\ X[t']=j}} Y'[t'] = \sum_{t' \le t} Y'[t'] \le 0.$$

□

Thus, to check if a station sequence X can be valid, it suffices to verify the specific construction of Y.

1.3 Subtask 4

Unlike previous subtasks, the road network is no longer a path but a general tree. However, there is still only one station s with $A[s] > 0$. Similar to subtask 1, we must find the shortest walk starting at s that visits all *required* stations.

In this subtask, there may be stations where $A[i] = B[i]$. If a leaf station i satisfies $A[i] = B[i]$, it does not need to be visited. We can repeatedly remove such leaves until all remaining leaves have $A[i] \neq B[i]$. After this pruning process, all remaining stations in the tree must be visited at least once.

Suppose t is the final station in the walk.

- Each edge on the simple path from s to t must be traversed at least once.
- All other edges in the tree must be traversed at least twice.

Thus, a walk from s to t that visits all stations has a length of at least $2(N-1) - \text{dist}(s,t)$. ($N$ is the number of required stations here.) This walk can be generated via DFS:

- Perform a DFS starting from s.
- For each station, treverse subtrees that do not contain the final station t first.
- Traverse the subtree containing t last.

The walk concludes immediately upon reaching t for the last time, achieving the lower bound distance. To minimize the total distance, we simply choose t as the station farthest from s in the pruned tree.

The time complexity is $O(N)$.

1.4 Subtask 3

1.4.1 Incorrect Approach

Following the ideas from subtask 1 and subtask 4, an intuitive approach is to find a starting station s and a final station t such that the strategy traverses edges on the s-t simple path exactly once and all other edges exactly twice. This would result in a travelling distance $2(N-1) - \text{dist}(s,t)$. (Note: This is incorrect, but we will explain this idea first.)

The question is which (s,t) pairs can support such a strategy. To identify these, we orient every edge according to the required flow of bikes across the edge. For every edge (u,v), let S_u and S_v be the net balance (the sum of $A[i] - B[i]$) of the two components separated by the edge, respectively. Since the total number of bikes is conserved, $S_u = -S_v$. We can orient the edge based on this balance:

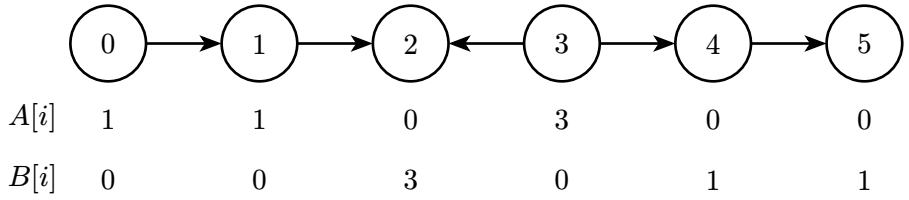
- $S_u > 0$: Bikes must move from u's side to v's side, so the orientation is $u \to v$.
- $S_u < 0$: Bikes must move from v's side to u's side, so the orientation is $u \leftarrow v$.
- $S_u = 0$: There is no net transfer required across the edge. The orientation can be in either direction; we assume it is oriented away from s.

Obviously, if an edge $u \leftarrow v$ has an orientation toward s, a valid strategy cannot traverse it only once; otherwise, we would be unable to move bikes from v's side to u's side. Therefore, to traverse edges on the s-t simple path only once, all edges on that path must be oriented toward t. We would then seek the longest directed path and let its endpoints be the starting station s and the final station t. Since this is the longest directed path, s has no incoming edges and t has no outgoing edges among all incident edges.

We can then construct a strategy. Without loss of generality, assume that $s < t$.

- The strategy starts at s. First, it collects all bikes from stations $s, s-1, ..., 0$, and unloads bikes at $0, 1, ..., s-1$ to reach their target $B[i]$. This is valid because $\sum_{i=0}^{s}(A[i] - B[i]) \geq 0$ due to the orientation $s \to s+1$.
- Then, for each $i = s, s+1, ..., t-1$, it loads or unloads bikes to achieve $B[i]$. This is valid because $\sum_{j=0}^{i}(A[j] - B[j]) \geq 0$ due to the orientation $i \to i+1$.
- Finally, it collects all bikes at stations $t, t+1, ..., N-1$ and unloads bikes at $N-1, N-2, ..., t$. This is valid because $\sum_{i=0}^{N-1}(A[i] - B[i]) = 0$.

Although this strategy is valid, **it is not the optimal solution**. Consider the following counterexample:



The approach above might identify $(s,t)=(0,2)$ or $(s,t)=(3,5)$, both of which have a travelling distance of 8. However, there is a better solution:

$$X=[0,1,2,3,2,3,4,5],\quad Y=[-1,-1,0,-3,3,0,1,1].$$

This strategy has a travelling distance of 7.

1.4.2 Correct Approach

The key insight is that edges on the s-t simple path **can** be oriented toward s, but such edges must be traversed three times. We assign weights to the edges: if an edge is oriented away from s, its weight is 1; if it is oriented toward s, its weight is -1. Let $w(s,t)$ be the total weight of edges on the s-t simple path.

- Every edge not on the s-t simple path must be traversed at least twice.
- Every edge on the s-t simple path oriented away from s must be traversed at least once.
- Every edge on the s-t simple path oriented toward s must be traversed at least three times.

Thus, a strategy from s to t has a travelling distance of at least $2(N-1)-w(s,t)$. Our goal is to find the (s,t) pair that maximizes $w(s,t)$. By the maximality, s will have no incoming edges and t will have no outgoing edges among all incident edges. We can then construct a walk that achieves the lower bound. Without loss of generality, assume $s<t$.

We partition the stations on the s-t simple path (including s and t) into groups $P_1,P_2,...,P_k$ in order from s to t. Each group P_i contains $s_i,s_i+1,...,t_i$ (where $s_i\le t_i$), defined such that:

- Within every P_i, all edges are oriented toward s.
- For every $1\le i<k$, the edge (t_i,s_{i+1}) is oriented away from s.

That is, continuous segments of stations connected by edges oriented toward s form groups. We complete the rebalancing for each group sequentially from $i=1$ to k. First, the truck collects all bikes at stations $s,s-1,...,0$ and unloads them at $0,1,...,s-1$. This is valid because $\sum_{i=0}^{s}(A[i]-B[i])\ge 0$ due to the orientation $s\to s+1$. The truck then enters the first group.

For the i-th group, the truck first moves forward to collect all bikes at stations $s_i,s_i+1,...,t_i$, then moves back to s_i without loading or unloading bikes, and finally moves to t_i and unloads bikes to stations $s_i,s_i+1,...,t_i$. This is valid because after collecting bikes at t_i, the truck carries $\sum_{j=0}^{t_i}A[j]-\sum_{j=0}^{s_i-1}B[j]$ bikes. By our grouping method, $\sum_{j=0}^{t_i}(A[j]-B[j])\ge 0$ ensures the truck bike count never drops below zero.

After the last group is processed, the truck collects all bikes at $t+1,t+2,...,N-1$ and unloads bikes at $N-1,N-2,...,t+1$.

In conclusion, (X,Y) is a valid strategy with a travelling distance of $2(N-1)-\max w(s,t)$. For this subtask, $\max w(s,t)$ can be found by assigning weights to edges (there are two cases: orienting flexible edges either right (if $s<t$) or left (if $t<s$)) and computing the longest weighted path. This can be implemented in $O(N)$ time.

1.5 Subtask 5, 6

The main idea is similar to subtask 3. First, we remove all leaves where $A[i]=B[i]$ as described in subtask 4, and assume N represents the number of remaining stations. We then enumerate the

starting station s and assign edge weights based on the directions of edges. (Recall that the directions of edges with flexible orientations are determined by s.) We choose the station t that maximizes the weighted path sum $w(s,t)$ as the final station. The minimum travelling distance is thus $2(N-1)-w(s,t)$.

We construct a strategy to achieve the distance bound. Similar to subtask 3, we partition the stations on the s-t simple path into groups $P_1, P_2, ..., P_k$ using the same logic. However, because the road network is now a general tree, we must handle subtrees branching off from the s-t path.

Consider the tree rooted at s. For each station i, let S_i be the sum of $A[j]-B[j]$ over stations j in the subtree rooted at i. The edge connecting station i and its parent is oriented upward if $S_i > 0$ and downward otherwise.

Observation 1.3. If the truck enters a subtree rooted at v carrying at least $\max(-S_v, 0)$ bikes, there exists a strategy to complete the rebalancing of all stations in that subtree with a travelling distance of $2(N_v-1)$, starting and ending at v, where N_v is the size of the subtree.

Proof. We prove this by induction. Suppose the truck is at station v and carries exactly $\max(-S_v, 0)$ bikes. First, the truck collects $A[v]$ bikes at station v. For every child c of v such that $S_c \geq 0$, since the truck carries at least $0 \geq -S_c$ bikes, we can complete rebalancing of the subtree rooted at c in $2(N_c-1)+2 = 2N_c$ steps and return to v. After rebalancing these subtrees in arbitrary order, the truck carries

$$\max(-S_v,0) + A[v] + \sum_{\substack{\text{child } c \\ S_c \geq 0}} S_c = \max(-S_v,0) + S_v + B[v] - \sum_{\substack{\text{child } c \\ S_c < 0}} S_c \geq B[v] + \sum_{\substack{\text{child } c \\ S_c < 0}} |S_c|$$

bikes. For every child c with $S_c < 0$, the truck now carries at least $-S_c$ bikes, allowing us to complete the rebalancing of those subtrees and return to v. Finally, we unload $B[v]$ bikes at station v. The total distance is $\sum_{\text{child } c} 2N_c = 2(N_v-1)$. □

In short, we rebalance subtrees rooted at children c with non-negative S_c first, followed by those with negative S_c.

The truck first enters the first group P_1. For each group P_i with members $v_1, v_2, ..., v_\ell$ (where $v_1 = s_i$ and $v_\ell = t_i$), s_i is an ancestor of t_i. Before entering P_i, the truck has completed rebalancing for all stations outside the subtree of s_i and carries exactly $-S_{s_i}$ bikes. ($-S_{s_i} \geq 0$ by the orientation $t_{i-1} \to s_i$ for all $s_i \neq s$, and $S_{s_1} = 0$.)

- The truck moves from s_i to t_i. For simplicity, we do not load bikes at intermediate stations $v_1, ..., v_\ell$ yet.
- The truck moves from t_i back to s_i. At each station v_j along the way:
  - The truck now carries $-S_{s_i} + S_{v_{j+1}} - S_{s_{i+1}}$ bikes. $S_u = 0$ if u does not exist. There are $-S_{s_i}$ bikes from the previous groups, and $S_{v_{j+1}} - S_{s_{i+1}}$ from previously processed stations in this group.
  - Load $A[v_j]$ bikes.
  - Rebalance all branching subtrees rooted at children c such that c is not on the s-t simple path and $S_c \geq 0$. After this, the truck carries

$$\underbrace{-S_{s_i}}_{\geq -S_{v_j}} + S_{v_{j+1}} - \underbrace{S_{s_{i+1}}}_{\leq 0} + A[v_j] + \sum_{\substack{\text{child } c \notin P \\ S_c \geq 0}} S_c \geq -S_{v_j} + S_{v_j} + B[v_j] - \sum_{\substack{\text{child } c \notin P \\ S_c < 0}} S_c \geq B[v_j] + \sum_{\substack{\text{child } c \notin P \\ S_c < 0}} -S_c$$

bikes, where P is the s-t simple path, and $S_{v_{j+1}} = S_{s_{i+1}}$ if v_{j+1} does not exist; $S_{s_{i+1}} = 0$ if s_{i+1} does not exist.

- Rebalance all branching subtrees rooted at children c such that c is not on the s-t simple path and $S_c < 0$.
- Finally, unload the required $B[v_j]$ bikes at station v_j.

- Move to station t_i.

The travelling distance is

$$2(N-1) - \text{dist}(s,t) + \sum_{i=1}^{k}(|P_i| - 1) = 2(N-1) - w(s,t).$$

We can find the maximum $w(s,t)$ by enumerating each possible s and computing the weighted path sums for every t. This results in an $O(N^2)$ implementation, which is sufficient for Subtask 6. Subtask 5 allows for a less efficient $O(N^3)$ approach.

1.6 Subtask 7

We can improve the time complexity to $O(N)$ using dynamic programming. First, root the tree at an arbitrary station r. For every station $v \neq r$, we compute $dp_{\uparrow}[v]$ and $dp_{\downarrow}[v]$. Here, $dp_{\uparrow}[v]$ represents the maximum path weight from ***v*'s parent** to a descendant of v, assuming all edges with flexible directions are oriented upward. $dp_{\downarrow}[v]$ is defined similarly, but edges with flexible directions are oriented downward.

These values can be computed efficiently using bottom-up transitions. The maximum weighted path $w(s,t)$ can be found by considering each station v as the lowest common ancestor of s and t. $w(s,t)$ is the maximum value among

$$\max_{v} \max_{\substack{\text{child } c_1, c_2 \text{ of } v \\ c_1 \neq c_2}} \{dp_{\uparrow}[c_1] + dp_{\downarrow}[c_2]\}, \quad \max_{v \neq r}\{dp_{\uparrow}[v]\}, \quad \max_{v \neq r}\{dp_{\downarrow}[v]\}.$$

By backtracking through the DP states or storing the endpoints during the transitions, we can identify the specific stations s and t that yield the maximum weight. Finally, we construct the rebalancing strategy using the method described in the previous subtask. The total time complexity is $O(N)$.