Petrol Stations Editorial

Subtask 1

First, we precompute how many vertices lie on each side of every edge. Then for each vertex, we can simulate all outgoing cars using DFS. During DFS, we store the current state of the fuel tank and how many cars are going in the particular direction.

This trick (storing how many outgoing cars there are for a given vertex) will be useful in the majority of the following solutions.

Subtask 2

As we have a path, we will initially renumber the vertices $0 \dots N-1$ from the left to the right. Then, the number of cars stopping to refuel in a vertex i is:

$$\left\lfloor \frac{i}{K} \right\rfloor (n-i-1) + \left\lfloor \frac{n-i-1}{K} \right\rfloor i$$

Subtask 3

We will count cars going left and cars going right separately. Let's consider only those from the left to the right (the other case is equivalent).

We will store current cars in a queue, grouped by the current amount of fuel. For each group, we will remember the number of cars in it and the maximum distance from the vertex 0 they can go without refueling.

We will go through the cities from the left to the right. In each city, we remove all groups that cannot reach the next city. We will merge them with cars starting here and create a group from all of them. They can reach the distance of the current city plus K.

This solution will run in $\mathcal{O}(N)$ time.

Subtask 4

Initially, we root the tree.

We can observe that each car will first move upwards and then downwards. (One of the parts can be empty.)

First, we will consider the upwards part of each car's path. Using DFS, we compute for every vertex the number of cars that will reach it going upwards. Cars will be stored in an array indexed by remaining fuel. Therefore, the array for each vertex can be obtained by adding arrays from its children.

Using another DFS, we compute the downward parts. In each vertex we will have some cars from our parent. Then we have to add cars from our siblings — they will turn around in our parent. By adding all of them, we have the array for the current vertex.

Subtask 5

As adding cars from all siblings is the only slow part for this subtask, we only modify that. We can compute the sum of all other siblings as a sum of all sibling minus us. The sum of all siblings can be precomputed once for each parent in $\mathcal{O}(N)$ time.

Full solution (using heavy-light decomposition)

In our solution, we will use the following data structure: A persistent binary search tree, with keys being remaining fuel in cars and values corresponding count of cars. For each vertex we also store the number of cars in its subtree.

In addition, we will require the tree to be able to split itself by remaining fuel. Finally, we require adding given amount of fuel to all cars. This can be achieved by an offset for the entire tree. Suitable implementation of this data structure can be a treap.

Using this data structure, we can efficiently implement almost entire solution of Subtask 5. When going up, we merge smaller tree into bigger one. (You can look at this as merging into heavy edges of heavy-light.)

A problem arises when going down using a light edge, where we need to join trees from the parent and a heavy edge, but we don't have time for going through any of them. Therefore during the path from the root, we will propagate a group of trees. For every light edge, we will create a new one, so there will be at most $\log n$ trees.

Total time complexity will be $\mathcal{O}(N \log^2 N)$ because we do $\mathcal{O}(N \log N)$ transfers between trees and each takes $\mathcal{O}(\log N)$.

Full solution (using centroid decomposition)

We will repeatedly find a centroid and count all cars going through it. Then repeat for each subtree of this centroid.

The path of each car is divided to parts before and after the centroid. The part before it is simple — for each vertex, we can use binary search to compute the remaining fuel of a car starting in this vertex at the arrival in the centroid.

In the centroid, we will build a binary search tree of all cars passing through. The key will be how far can the car go from the centroid without refueling (initially remaining fuel). Additionally, we will want to answer queries for the number of cars in a given interval.

The part from the centroid will be computed by DFS where we modify the tree. When returning to a vertex, we will reverse all operations, so to have its original state. Before entering a child, we count the cars having to refuel and replace them with ones with a full tank. When returning, we remove them. It can seem that we have to remove the refueling cars. However, if we don't ask for the interval in which can be a car that we should have removed, it won't matter.

Instead of balanced search trees, we can also use a sparse segment tree on the universe 0 to $N \cdot K$.