APIOBike

APIOBike is a newly launched bike-sharing service in APIO City. Several stations have been set up throughout the city, allowing users to borrow or return bikes at any station. Due to its convenience and low cost, it has quickly become the most important mode of transportation in APIO City. However, APIOBike has encountered a common problem faced by all bike-sharing services: an imbalance between borrowing and returning rates across different stations. This results in some stations having very few bikes, leaving nearby residents unable to borrow one, while other stations become overstocked with bikes that local residents do not need.

To address this issue, APIOBike plans to dispatch a rebalancing truck every night to redistribute bikes across the stations, ensuring that each station maintains an appropriate number of bikes. APIO City has a total of N stations, numbered $0, 1, \ldots, N-1$. Through observation, APIOBike has found that every evening, the number of bikes at station i is always a fixed value $A[i]$, and no one borrows or returns bikes at night. The company aims to have exactly $B[i]$ bikes at station i every morning.

Among these N stations, there are $N-1$ dedicated roads for the rebalancing truck. Each road connects two distinct stations, and the truck is restricted to travelling only on these roads. Every road is one unit long. This network of stations is connected, ensuring that the truck can travel between any pair of stations. Every night, APIOBike must dispatch a rebalancing truck to ensure that each station i has exactly $B[i]$ bikes. The truck may start at any station and may end its route at any station.

The truck is empty at the beginning of rebalancing. Whenever the truck is at a station, the driver may load any number of bikes from the station onto the truck, or unload any number of bikes from the truck onto the station. The truck and the stations have unlimited capacities for bikes, but the number of bikes at any location must never drop below zero. The company wants to determine the minimum possible travelling distance to complete the rebalancing and its corresponding strategy.

Implementation Details

You should implement the following procedure.

```
std::pair<std::vector<int>, std::vector<long long>>
find_rebalancing_strategy(int N,
                          std::vector<int> A,
                          std::vector<int> B,
                          std::vector<int> U,
                          std::vector<int> V)
```

- A: an array of length N, where $A[i]$ is the number of bikes at station i each evening.
- B: an array of length N, where $B[i]$ is the target number of bikes at station i each morning.
- U, V: arrays of length $N-1$ representing the road network. For each $0 \le i < N-1$, the i-th road connects station $U[i]$ and station $V[i]$.
- This procedure is called **at most** $150\,000$ **times** for each test case.

The procedure should return a pair of arrays (X, Y) of equal length $k+1$, representing a rebalancing strategy:

- X: the sequence of station indices visited in chronological order.
- Y: the net bike delivery at each visited station. For each $0 \le j \le k$:
  - If $Y[j] \ge 0$, the driver unloads $Y[j]$ bikes at station $X[j]$ from the truck, increasing the bike count of the station by $Y[j]$.
  - If $Y[j] < 0$, the driver loads $-Y[j]$ bikes from station $X[j]$ onto the truck, decreasing the bike count of the station by $-Y[j]$.

A **valid** rebalancing strategy (X, Y) with travelling distance k must satisfy the following conditions:

- For each $0 \le j \le k$, $0 \le X[j] < N$.
- For each $0 \le j < k$, stations $X[j]$ and $X[j+1]$ are directly connected by a road.
- For each $0 \le j \le k$, $\sum_{t=0}^{j} Y[t] \le 0$.
- For each station $0 \le i < N$ and each step $0 \le j \le k$, let $S_{i,j}$ be sum of $Y[t]$ over all steps t such that $0 \le t \le j$ and $X[t] = i$.
  - The number of bikes at station i never drops below zero: $A[i] + S_{i,j} \ge 0$.
  - After the strategy concludes, each station i must contain exactly $B[i]$ bikes: $A[i] + S_{i,k} = B[i]$.

Constraints

- $2 \le N \le 300\,000$
- The sum of N over all calls to `find_rebalancing_strategy` does not exceed $300\,000$ in each test case.
- $0 \le U[i], V[i] < N$ and $U[i] \ne V[i]$ for each i such that $0 \le i < N$.

- $0 \leq A[i], B[i] \leq 10^9$ for each i such that $0 \leq i < N$.
- It is possible to travel between any pair of stations.
- $\sum_{i=0}^{N-1} A[i] = \sum_{i=0}^{N-1} B[i]$.
- There exists at least one i such that $0 \leq i < N$ and $A[i] \neq B[i]$.

Subtasks and Scoring

Here, T is the number of calls to `find_rebalancing_strategy`, and $\sum N$ is the sum of all N over all calls to `find_rebalancing_strategy`.

Subtask	Score	Additional Constraints
1	4	The stations and roads satisfy property P (see below). There exists exactly one i such that $0 \leq i < N$ and $A[i] > 0$.
2	11	$T \leq 10$. $N \leq 7$. The stations and roads satisfy property P.
3	16	The stations and roads satisfy property P.
4	9	There exists exactly one i such that $0 \leq i < N$ and $A[i] > 0$.
5	24	$\sum N \leq 500$.
6	15	$\sum N \leq 5000$.
7	21	No additional constraints.

Property P: For each $0 \leq i < N - 1$, $U[i] = i$ and $V[i] = i + 1$. For each $0 \leq i < N$, $A[i] \neq B[i]$.

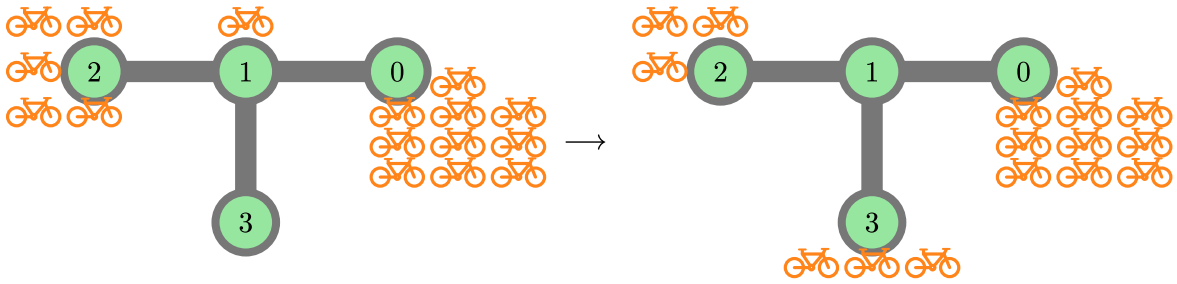
If the return arrays X and Y have length exactly $k^* + 1$, where k^* is the minimum possible travelling distance, but (X, Y) is not a valid strategy, you will receive 50% of the score.

Example

Example 1

Consider the following call:

```
find_rebalancing_strategy(4,
                          [10, 1, 5, 0],
                          [10, 0, 3, 3],
                          [0, 1, 1],
                          [1, 2, 3])
```



There are $N = 4$ stations in APIO City. Initially, the stations have $A = [10, 1, 5, 0]$ bikes, and the goal is to have $B = [10, 0, 3, 3]$ bikes at every station.

Suppose the rebalancing truck starts at station 2, and follows these steps:

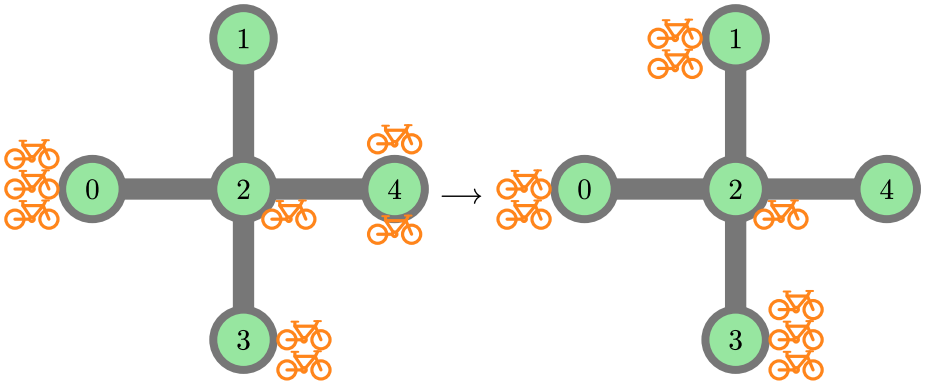
- At station 2, load 2 bikes onto the truck. Station 2 now has 3 bikes, and the truck has 2 bikes.
- Move to station 1 and load 1 bike onto the truck. Station 1 now has 0 bikes, and the truck has 3 bikes.
- Move to station 3, and unload 3 bikes. Station 3 now has 3 bikes, and the truck has 0 bikes.

The total moving distance is 2. The corresponding return arrays for this strategy are $X = [2, 1, 3]$ and $Y = [-2, -1, 3]$. It can be proven that this strategy achieves the minimum distance. Thus, the procedure should return `([2, 1, 3], [-2, -1, 3])`.

Example 2

Consider the following call:

```
find_rebalancing_strategy(5,
                          [3, 0, 1, 2, 2],
                          [2, 2, 1, 3, 0],
                          [2, 2, 2, 2],
                          [0, 4, 3, 1])
```



There are $N = 5$ stations in APIO City. Initially, the stations have $A = [3, 0, 1, 2, 2]$ bikes, and the goal is to have $B = [2, 2, 1, 3, 0]$ bikes at every station.

Suppose the rebalancing truck starts at station 0, and follows these steps:

- At station 0, load 1 bike.
- Move to station 2 and load 1 bike.
- Move to station 1 and unload 2 bikes.
- Move to station 2.
- Move to station 4 and load 2 bikes.
- Move to station 2 and unload 1 bike.
- Move to station 3 and unload 1 bike.

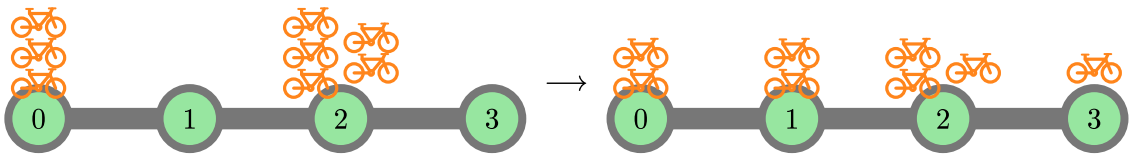
The total moving distance is 6. The corresponding return arrays for this strategy are $X = [0, 2, 1, 2, 4, 2, 3]$ and $Y = [-1, -1, 2, 0, -2, 1, 1]$. It can be proven that this strategy achieves the minimum distance. Thus, the procedure should return `([0, 2, 1, 2, 4, 2, 3], [-1, -1, 2, 0, -2, 1, 1])`.

Other valid strategies with distance 6, such as $X = [0, 2, 3, 2, 4, 2, 1]$ and $Y = [-1, 0, 1, 0, -2, 0, 2]$, are also considered correct. Returning arrays X, Y with length $7 = 6 + 1$ but not a valid strategy, such as $X = Y = [0, 0, 0, 0, 0, 0, 0]$, will receive 50% of the score.

Example 3

Consider the following call:

```
find_rebalancing_strategy(4,
                          [3, 0, 5, 0],
                          [2, 2, 3, 1],
                          [0, 1, 2],
                          [1, 2, 3])
```



An optimal solution is $X = [2, 1, 0, 1, 2, 3]$ and $Y = [-1, 1, -1, 1, -1, 1]$. The procedure should return `([2, 1, 0, 1, 2, 3], [-1, 1, -1, 1, -1, 1])`. Other valid strategies, such as $X = [2, 1, 0, 1, 2, 3]$ and $Y = [-2, 2, -1, 0, 0, 1]$, are also considered correct.

This example satisfies the constraints of subtasks 2 and 3.

Sample Grader

The first line of the input should contain a single integer T, the number of scenarios. A description of T scenarios should follow, each in the format specified below.

Input format:

```
N
A[0] A[1] ... A[N-1]
B[0] B[1] ... B[N-1]
U[0] V[0]
U[1] V[1]
...
U[N-2] V[N-2]
```

Output format:

```
k
X[0] X[1] ... X[k]
Y[0] Y[1] ... Y[k]
```