Trains

Consider a rail network covering N cities. There are exactly $N-1$ direct routes between some pairs of the N cities such that these routes form the structure of a rooted tree with the root being city 1. For every city i except 1 you are given F_i, the immediate parent of node i.

Every second, starting with second 0, a train leaves city 1 on the shortest path to some other city. All trains have the same speed and all the direct routes are such that it takes exactly 1 second to go between two cities that share a direct link.

For each city you know the latest time T_i when the first train needs to arrive in the city, as demanded by the customers.

You have to help the train dispatcher and determine if there is a way to direct each train such that at least one of them arrives in or passes through city i before or at T_i (the final destinations of the trains are irrelevant).

You should implement a function named `solution` that takes as arguments:

- `int N` : the number of cities;
- `int *F` : a `0` indexed array with $N+1$ elements (`F[0]` and `F[1]` are undefined) representing the map;
- `int *T` : a `0` indexed array with $N+1$ elements (`T[0]` is undefined) representing the customers' demands.

and returns `int` : `1` if all customers can be pleased or `0` otherwise.

Standard input

The template code reads the input data in binary format. You should not (be able to) edit this part of the code.

Standard output

The template code calls your function multiple times within the same test case, each time with possibly a different rail network. Each time the function should return the correct answer.

Testing

This is an interactive problem. You don't need to read or write anything from standard input or to standard output.

The code we provide reads from the standard input in the following format:

- The first line contains `C` , the number of times the function `solution` is called.
- `C` similar blocks of data, describing the parameters of the function follow. Each have the format:

```
N
F[2] F[3] F[4] ... F[N]
T[1] T[2] T[3] ... T[N]

```

- Note that `F[0]` , `F[1]` and `T[0]` don't appear in the input as they are undefined and you should not use their values when solving the problem.
- Two equivalent examples are provided: one of them is in text format and follows the structure described above, the other is in binary format and contains the same information encoded differently.

Constraints and notes

- The function will be called at most 50 times.

- $1 \leq N \leq 1.5 \cdot 10^5$
- $1 \leq S \leq 5 \cdot 10^6$, $S =$ sum of all N within a testcase
- $0 \leq T_i \leq 10^9$ for every $1 \leq i \leq N$

Subtasks

Test cases will be scored ***individually***.

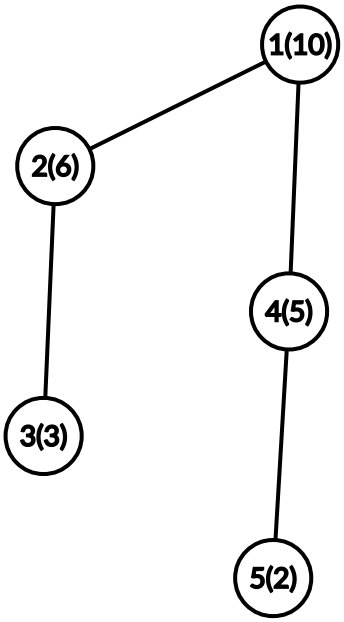
Subtask	Percentage of test cases	Additional input constraints
1	10%	$N \leq 50$
2	10%	$50 < N \leq 10^3$
3	20%	$10^3 < N \leq 10^4$
4	30%	$10^4 < N \leq 5 \cdot 10^4$
5	30%	none

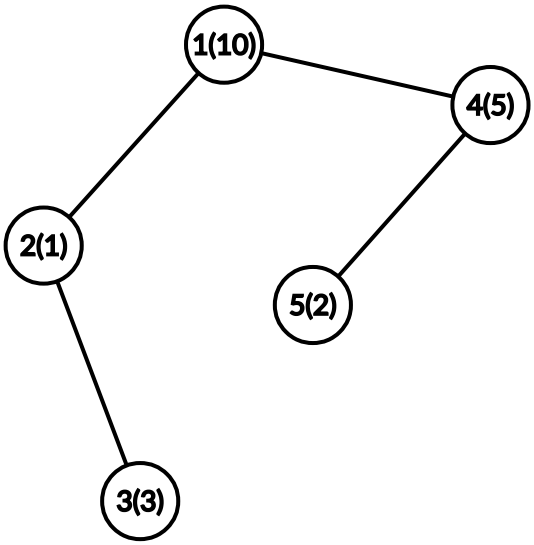
Examples

Parameters	Return value

`N = 5`
`F = - - 1 2 1 4`
`T = - 10 6 3 5 2`
	`1`

`N = 5`
`F = - - 1 2 1 4`
`T = - 10 1 3 5 2`
	`0`


1(10)
2(6)
4(5)
3(3)
5(2)


1(10)
4(5)
2(1)
5(2)
3(3)