Love Running Every Day

Problem Description

C thought running was fun, so he decided to make a game called *Love Running Every Day*. "*Love Running Every Day*" is an education simulation game, the player needs to be online on time every day, and complete the punch-in task.

The map of this game can be seen as a tree containing n nodes and n-1 edges, each edge connecting two nodes, and any two nodes have a path reachable to each other. The nodes on the tree are numbered as consecutive positive integers from 1 to n.

There are now m players, and the i^{th} player starts at s_i and ends at t_i. At the beginning of each day, all players start from their starting point at the same time in the 0^{th} second and run at the speed of one edge per second without interruption along the shortest path toward their endpoint. After running to the endpoint, the player is considered to have completed the task of punching in. (Since the map is a tree, each player's path is unique)

C wants to know how active the game is, so he places a spotter on each node. The observer at node j will choose to observe players at second w_j, and a player can be observed by this observer if and only if the player arrives at node j exactly at second w_j. C wants to know how many people each observer will observe.

Note: We assume that a player will end his game when he reaches his end point, he can't wait a while before being observed by the observer. That is, for a player with node j as his endpoint: if he reaches the end point before second w_j, the observer at node j can not observe the player; If he reaches the end at second w_j, the observer at node j can observe the player.

Input

The first line has two integers n and m. Where n is the number of nodes in the tree, which is also the number of observers, and m is the number of players.

For the next n-1 lines, there are two integers u and v in each line, which means there is an edge from node u to node v.

This is followed by a line of n integers, where the j^{th} integer is w_j, which represents the time at which the observer appears at node j.

This is followed by m lines with two integers s_i and t_i per line, representing the starting and endpoints of a player.

For all data, it is ensured that $1 \leq s_i, t_i \leq n$, and $0 \leq w_j \leq n$.

Output

Output 1 line of n integers, with the j^{th} integer indicating how many people can be observed

by the observer at node j.

Sample Input 1

6 3
2 3
1 2
1 4
4 5
4 6
0 2 5 1 2 3
1 5
1 3
2 6

Sample Output 1

2 0 0 1 1 1

Sample Input 2

5 3
1 2
2 3
2 4
1 5
0 1 0 3 0
3 1
1 4
5 5

Sample Output 2

1 2 1 0 1

Hint

[Explanation of Sample 1]

For point 1, w_i=0, so only players starting at point 1 will be observed, so players 1 and 2 are observed, making a total of 2 people observed.

For point 2, no player is at this node at second 2, and a total of 0 players are observed.

For point 3, no player was at this node at second 5, and a total of 0 players were observed.

For point 4, player 1 is observed and a total of 1 player is observed.

For point 5, player 1 is observed and a total of 1 person is observed.

For point 6, player 3 is observed and a total of 1 person is observed.

[Subtask]
The data size and characteristics of each test point are shown in the following table.

Hint: The numbers in the ones place of the data range can help you determine which data type it is.

Test Point	n	m	Conventions
1	= 991	= 991	Everyone's starting point is their own ending
2			ie: Si = Ti
3	= 992	= 992	Wj = 0
4			
5	= 993	= 993	None
6	= 99994	= 99994	The tree degenerated into a chain, where 1
7			and 2 have an edge, 2 and 3 have an edge, ...,
8			n-1 and n have an edge
9	= 99995	= 99995	All si = 1
10			
11			
12			
13	99996	= 99996	All Ti = 1
14			
15			
16			
17	= 99997	= 99997	None
18			
19			
20	= 299998	= 299998	

[Hint]
If your program requires large stack space (which usually means deep recursion), be sure to read the text document running/stack.txt in the player directory carefully to understand the stack space limit at the time of the final evaluation and how to adjust the stack space limit in your current working environment.

There will be no separate limit on the size of the stack at the time of the final evaluation, but in our environment, there will be a default limit of 1MiB. This can cause a stack overflow crash when the number of function call layers is large.

There are ways to modify the size limit of the call stack. For example, enter the following command into the terminal: ulimit -s 1048576

The meaning of this command is to change the size limit of the call stack to 1GiB.

For example, create the following sample.cpp or sample.pas in the contestant directory

sample.cpp

```
void dfs(int a){
if(a == 0)
return;
int t = a;
dfs(a - 1);
}
int main(){
dfs(1000000);
return 0;
}
```

sample.pas

```
procedure dfs(a:
longint);
var t: longint;
begin
if a = 0 then
exit;
t := a;
dfs(a - 1);
end;
begin
dfs(1000000);
end.
```

After compiling the above source code into an executable file sample, you can run the program in the terminal by running the following command

./sample

If you run this program without the command “ulimit -s 1048576”, the program will crash due to a stack overflow; if you run the program after using the above command, the program will not crash.

In particular, when you have multiple terminals open, they will not share the command, so you will need to run the command separately for them.

Note that the space occupied by the call stack is counted into the total space footprint and is bounded by memory along with the rest of the program.