B. Tree Tag

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Alice and Bob are playing a fun game of tree tag.

The game is played on a tree of nn vertices numbered from 11 to nn. Recall that a tree on nn vertices is an undirected, connected graph with n−1n−1 edges.

Initially, Alice is located at vertex aa, and Bob at vertex bb. They take turns alternately, and Alice makes the first move. In a move, Alice can jump to a vertex with distance **at most** dada from the current vertex. And in a move, Bob can jump to a vertex with distance **at most** dbdb from the current vertex. The distance between two vertices is defined as the number of edges on the unique simple path between them. In particular, either player is allowed to stay at the same vertex in a move. Note that when performing a move, a player only occupies the starting and ending vertices of their move, not the vertices between them.

If after at most 1010010100 moves, Alice and Bob occupy the same vertex, then Alice is declared the winner. Otherwise, Bob wins.

Determine the winner if both players play optimally.

**Input**

Each test contains multiple test cases. The first line contains the number of test cases tt (1≤t≤1041≤t≤104). Description of the test cases follows.

The first line of each test case contains five integers n,a,b,da,dbn,a,b,da,db (2≤n≤1052≤n≤105, 1≤a,b≤n1≤a,b≤n, a≠ba≠b, 1≤da,db≤n−11≤da,db≤n−1)  — the number of vertices, Alice's vertex, Bob's vertex, Alice's maximum jumping distance, and Bob's maximum jumping distance, respectively.

The following n−1n−1 lines describe the edges of the tree. The ii-th of these lines contains two integers uu, vv (1≤u,v≤n,u≠v1≤u,v≤n,u≠v), denoting an edge between vertices uu and vv. It is guaranteed that these edges form a tree structure.

It is guaranteed that the sum of nn across all test cases does not exceed 105105.

**Output**

For each test case, output a single line containing the winner of the game: "Alice" or "Bob".

**Example**

**input**

**Copy**

4

4 3 2 1 2

1 2

1 3

1 4

6 6 1 2 5

1 2

6 5

2 3

3 4

4 5

9 3 9 2 5

1 2

1 6

1 9

1 3

9 5

7 9

4 8

4 3

11 8 11 3 3

1 2

11 9

4 9

6 5

2 10

3 2

5 9

8 3

7 4

7 10

**output**

**Copy**

Alice

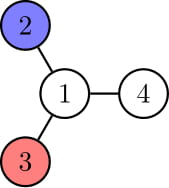
Bob

Alice

Alice

**Note**

In the first test case, Alice can win by moving to vertex 11. Then wherever Bob moves next, Alice will be able to move to the same vertex on the next move.



In the second test case, Bob has the following strategy to win. Wherever Alice moves, Bob will always move to whichever of the two vertices 11 or 66 is farthest from Alice.

https://espresso.codeforces.com/e01029d970f0284101ce72c70949e9dd589efb53.png

1. **#include** <bits/stdc++.h>
3. **using** **namespace** std;
5. **const** **int** N = 1e5 + 5;
6. **int** n, a, b, da, db, depth[N];
7. **vector<int>** adj[N];
8. **int** diam = 0;
10. **int** dfs(**int** x, **int** p) {
11. **int** len = 0;
12. **for**(**int** y : adj[x]) {
13. **if**(y != p) {
14. depth[y] = depth[x] + 1;
15. **int** cur = 1 + dfs(y, x);
16. diam = max(diam, cur + len);
17. len = max(len, cur);
18. }
19. }
20. **return** len;
21. }
23. **int** main() {
24. ios\_base::sync\_with\_stdio(**false**);
25. cin.tie(**nullptr**);
26. **int** te;
27. cin >> te;
28. **while**(te--) {
29. cin >> n >> a >> b >> da >> db;
30. **for**(**int** i = 1; i <= n; i++) adj[i].clear();
31. **for**(**int** i = 0; i < n - 1; i++) {
32. **int** u, v;
33. cin >> u >> v;
34. adj[u].push\_back(v);
35. adj[v].push\_back(u);
36. }
37. diam = 0;
38. depth[a] = 0;
39. dfs(a, -1);
40. cout << (2 \* da >= min(diam, db) || depth[b] <= da ? **"Alice"** : **"Bob"**) << **'\n'**;
41. }
42. }