NOI '20 P5 - Surreal

Time limit: 1.0s **Memory limit:** 512M

Your code will be run through three extra samples. These sample files can be found here.

In this problem, a *tree* is defined recursively: a single node gives rise to a tree, letting a tree to be the left (or right) child (of the root node) gives rise to a tree, and letting two trees to be left and right children (of the root node) gives rise to a tree. All structures generated using the above three rules in finite steps are called *trees*. *In other words, the "tree" here refers to a non-empty, rooted binary tree that distinguishes left children and right children*.

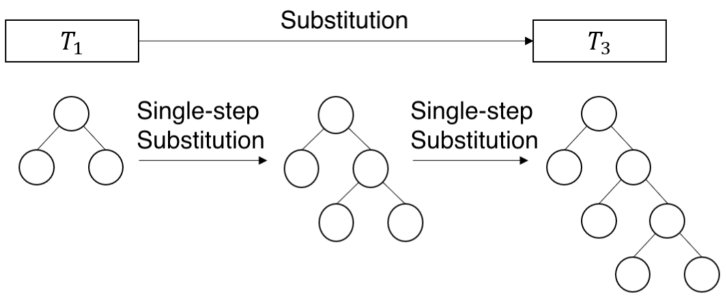
Two trees T, T' are said to be *isomorphic* ($T \equiv T'$) if they meet one of the following four conditions: (1) trees that are formed by one node are isomorphic; (2) if the root nodes of T and T' have only the left child, and their left subtrees are isomorphic, then T and T' are isomorphic; (3) if the root nodes of T and T' have only the right child, and their right subtrees are isomorphic, then T and T' are isomorphic; (4) if the root nodes of T and T' have both the left and right children, their left subtrees are isomorphic, and their right subtrees are also isomorphic, then T and T' are isomorphic. In other words, two trees are isomorphic if and only if they are the same when the nodes are unlabeled but we are distinguishing left children and right children.

It is obvious that the isomorphism of trees forms an equivalence relation over all trees, and we treat isomorphic trees as the same. We say two trees are different if and only if they are not isomorphic.

A *leaf* of a tree is defined in the usual way: a *leaf* is a node without any children.

We say T may be converted to T' using a *single-step substitution* if we may replace a leaf node of T with another tree T'' and the resulting tree is isomorphic to T', and we use $T \to T'$ to denote T may be converted to T' using a single-step substitution. We say T may be converted to T' by *substitution* if there exists a natural number $n \geq 1$ and trees $T_1, T_2, \ldots, T_n$ such that $T \equiv T_1 \to T_2 \to \cdots \to T_n \equiv T'$. We use $T \to^* T'$ to denote T may be converted to T' by substitution.

In other words, a single-step substitution means we are deleting a leaf of the tree and putting a new tree at the corresponding position, just like a larger subtree growing at the original leaf node. If a tree T may be converted to another tree T' by substitution, then it just means we may use zero, one, or multiple rounds of single-step substitutions to convert T into T'. For example, any tree may be converted to itself by substitution, or in other words, for any tree T, we have $T \to^* T$. This figure shall help understand the meaning of substitution and single-step substitution:



In particular, we can convert any tree into infinitely many different trees by substitution, and a tree formed by a single node can be converted to any other tree by substitution. For a tree T, we define $\text{grow}(T)$ to be the set of trees that we may convert T into by substitution, or in other words, $\text{grow}(T) = \{T' \mid T \to^* T'\}$. Moreover, if $\mathscr{T} = \{T_1, T_2, \ldots, T_n\}$ is a finite set of trees, then $\text{grow}(\mathscr{T})$ is defined to be the union of $\text{grow}(T_i)$ where $i = 1, 2, \ldots, n$. So we have $\text{grow}(\mathscr{T}) = \bigcup_{T_i \in \mathscr{T}} \text{grow}(T_i)$.

We may treat $\mathrm{grow}(\mathscr{T})$ as the set of trees the trees in set $\mathscr{T}$ can grow into. In other words, the set of trees that trees in $\mathscr{T}$ can grow into includes all trees that may be converted into from some $T \in \mathscr{T}$ by substitution. We may call a set of trees a *forest*. Not rigorously speaking, the new forest that a given forest can grow into are all the trees in the given forest and all possible trees that a tree in the given forest may grow into. It is obvious that the forest a non-empty forest can grow into is an infinite forest, but the infinite forest, or in other words, $\mathrm{grow}(\mathscr{T})$, does not necessarily contain all the trees. Moreover, it does not have to contain "nearly all" trees.

We say a forest is *almost complete* (or in other words, contains almost all trees) if there are only finitely many trees are not in the forest. For a finite forest $\mathscr{T}$, $\mathrm{grow}(\mathscr{T})$ may contain all trees, contain almost all trees, or there are infinitely many trees not in the forest. For a given finite set of trees $\mathscr{T}$, there exists an efficient algorithm to decide whether $\mathrm{grow}(\mathscr{T})$ is almost complete, i.e., there are only finitely many trees that trees in $\mathscr{T}$ cannot grow into.

The problem asks given a finite set of trees $\mathscr{T}$, whether there exists only finitely many trees T satisfying $T \notin \mathrm{grow}(\mathscr{T})$. $T \notin \mathrm{grow}(\mathscr{T})$ simply means there does not exist a $T' \in \mathscr{T}$ such that $T' \to^* T$.

Input Specification

Each test case contains multiple instances. The first line contains a positive integer T. There are T instances following, and each instance is specified in the following format: the first line is an integer m denoting the number of trees in the set. We will specify the m trees using the following format: the first line is an integer n denoting the number of nodes in the tree. The nodes are numbered $1, 2, \ldots, n$. The following n lines contain two non-negative integers each, and the i-th line contains l_i, r_i separated by a space denoting the left child and the right child of node i. If the left or the right child does not exist, then l_i or r_i is equal to 0. Of course, the leaf nodes satisfy $l_i = r_i = 0$. The input guarantees that it will be a tree with node 1 being the root. Please note that the labels of the nodes are for convenience only, and isomorphic trees are considered to be the same.

There may be isomorphic trees in the m trees of an instance. If we remove the duplicate trees (i.e. we only keep one tree for each isomorphism class), they shall form a set $\mathscr{T}$. You need to decide whether $\mathrm{grow}(\mathscr{T})$ is almost complete.

Output Specification

The output contains T lines specifying the answers to the T instances. The i-th line contains a string: if in the i-th instance, the m trees in the input grow into an almost complete forest (or in other words, there are only finitely many trees the trees specified in the instance cannot grow into), output `Almost Complete`. Otherwise, output `No`. Please pay attention to spelling and capitalization.

Sample Input 1

```
1
1
1
0 0
```

Sample Output 1

```
Almost Complete
```

Sample Input 2

```
1
3
3
2 3
0 0
0 0
2
2 0
0 0
2
0 2
0 0
```

Sample Output 2

```
Almost Complete
```

Sample Input 3

```
1
2
3
2 3
0 0
0 0
2
2 0
0 0
```

Sample Output 3

```
No
```

Explanation for Sample Output 3

It is obvious for all $n \geq 2$ chain-looking trees such that every non-leaf node has only the right child are not in $\mathrm{grow}(\mathscr{T})$, and so there are infinitely many trees not in $\mathrm{grow}(\mathscr{T})$.

Constraints

For all test cases, $\sum n \leq 2 \times 10^6$, $\sum m \leq 2 \times 10^6$, $\max h \leq 2 \times 10^6$, $T \leq 10^2$. Here, $\sum n$ denotes the sum of numbers of nodes of the trees in the instances occurring in a test case, $\sum m$ denotes the sum of number of trees occurring in the instances in a test case, $\max h$ denotes the maximum height of trees occurring in the particular test case (a tree with only one node has height 1).

Test case	T	$\sum n$	$\sum m$	$\max h$	Additional Constraints
1	100	≤ 1000		≤ 1	None.
2				≤ 2	For each instance, $m \leq 4$, or in other words, there are at most 4 trees in the set of trees occurring in the instance.
3					
4		$\leq 1\,000\,000$		≤ 4	None.
5				≤ 5	For each instance, the trees in the set have the same height.
6				≤ 8	None.
7				≤ 9	For each instance, the trees in the set have the same height.
8				≤ 10	None.
9				$\leq 1\,000\,000$	For each instance, the trees are chains (i.e. any non-leaf node has only one child).
10	20	≤ 1000	≤ 100	≤ 1000	For each instance, there are only trees satisfying one of the following two conditions (1) any non-leaf node has only one child (2) there are exactly two leaf nodes, and they have the same parent. All other nodes have exactly one child.
11		≤ 2000			
12		$\leq 100\,000$			
13		$\leq 200\,000$			

14		≤ 800	≤ 200	≤ 800	None.
15		≤ 1000	≤ 100	≤ 1000	
16		≤ 2000			
17	40	$\leq 300\,000$			
18		$\leq 600\,000$			
19		$\leq 900\,000$			
20		$\leq 1\,200\,000$			
21		$\leq 1\,500\,000$			
22		$\leq 2\,000\,000$			
23					
24					
25					