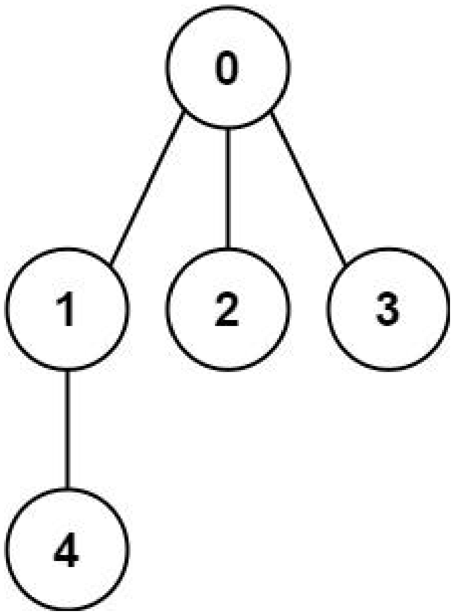
261. Graph Valid Tree

You have a graph of n nodes labeled from 0 to n - 1. You are given an integer n and a list of edges where edges[i] = [a_i, b_i] indicates that there is an undirected edge between nodes $a_i$and b_i in the graph.

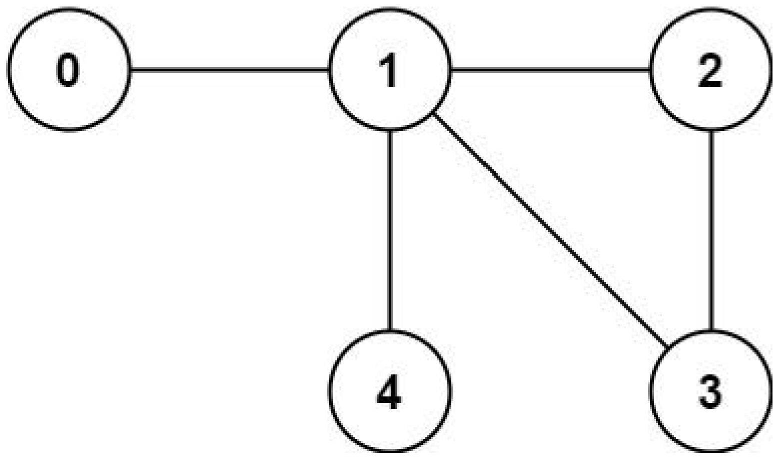
Return true *if the edges of the given graph make up a valid tree, and* false *otherwise*.

Example 1:



```
Input: n = 5, edges = [[0,1],[0,2],[0,3],[1,4]]
Output: true
```

Example 2:



```
Input: n = 5, edges = [[0,1],[1,2],[2,3],[1,3],[1,4]]
Output: false
```

Constraints:

- 1 <= n <= 2000
- 0 <= edges.length <= 5000
- edges[i].length == 2
- 0 <= a_i, b_i < n
- a_i != b_i
- There are no self-loops or repeated edges.

Code:

```
class Solution:
    def validTree(self, n: int, edges: List[List[int]]) -> bool:

        rank = [0]*n
        parent = [i for i in range(n)]

        def find(x):
            if parent[x]==x:
                return x
            parent[x]=find(parent[x])
            return parent[x]
```

```
def union(x,y):
    a = find(x)
    b = find(y)
    if (a==b):
        return False
    elif (rank[a]>rank[b]):
        parent[b]=a
    elif (rank[b]>rank[a]):
        parent[a]=b
    else:
        parent[a]=b
        rank[b]+=1
    return True

if len(edges) != n-1: return False

#for n-1 edges to form by n nodes, n-1 merges should take place. i.e even a single merge
not taking place implies presence of cycle.
for edge in edges:
    if not union(*edge):
        return False

return True
```