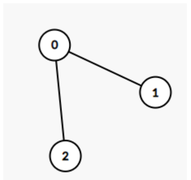
1245. Tree Diameter

Medium 529 11 Add to List Share

Given an undirected tree, return its diameter: the number of **edges** in a longest path in that tree.

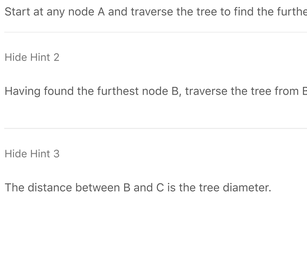
The tree is given as an array of `edges` where `edges[i] = [u, v]` is a bidirectional edge between nodes `u` and `v`. Each node has labels in the set `{0, 1, ..., edges.length}`.

Example 1:




```
Input: edges = [[0,1],[0,2]]
Output: 2
Explanation:
A longest path of the tree is the path 1 - 0 - 2.
```

Example 2:




```
Input: edges = [[0,1],[1,2],[2,3],[1,4],[4,5]]
Output: 4
Explanation:
A longest path of the tree is the path 3 - 2 - 1 - 4 - 5.
```

Constraints:

- `0 <= edges.length < 10^4`
- `edges[i][0] != edges[i][1]`
- `0 <= edges[i][j] <= edges.length`
- The given edges form an undirected tree.

Accepted 21,623 | Submissions 34,901

Seen this question in a real interview before? Yes No

Companies

0 ~ 6 months 6 months ~ 1 year 1 year ~ 2 years

Facebook 6 Google 2 Amazon 2 Microsoft 2

Related Topics

Tree Depth-First Search Breadth-First Search

Similar Questions

Count Subtrees With Max Distance Between Cities Hard

Hide Hint 1

Start at any node A and traverse the tree to find the furthest node from it, let's call it B.

Hide Hint 2

Having found the furthest node B, traverse the tree from B to find the furthest node from it, lets call it C.

Hide Hint 3

The distance between B and C is the tree diameter.

Java Autocomplete

```java
class Solution {
    private List<List<Integer>> graph;
    private Integer diameter = 0;

    public int treeDiameter(int[][] edges) {

        // build the adjacency list representation of the graph.
        this.graph = new ArrayList<List<Integer>>();
        boolean[] visited = new boolean[edges.length + 1];
        for (int i = 0; i < edges.length + 1; ++i) {
            this.graph.add(new ArrayList<Integer>());
            visited[i] = false;
        }
        for (int[] edge : edges) {
            Integer u = edge[0], v = edge[1];
            this.graph.get(u).add(v);
            this.graph.get(v).add(u);
        }

        dfs(0, visited);

        return this.diameter;
    }

    /**
     * return the max distance
     *   starting from the 'curr' node to its the leaf nodes
     */
    private int dfs(int curr, boolean[] visited) {
        // the top 2 distance starting from this node
        Integer topDistance1 = 0, topDistance2 = 0;

        visited[curr] = true;
        for (Integer neighbor : graph.get(curr)) {
            int distance = 0;
            if (!visited[neighbor])
                distance = 1 + this.dfs(neighbor, visited);

            if (distance > topDistance1) {
                topDistance2 = topDistance1;
                topDistance1 = distance;
            } else if (distance > topDistance2) {
                topDistance2 = distance;
            }
        }

        // with the top 2 distance, we can update the current diameter
        this.diameter = Math.max(this.diameter, topDistance1 + topDistance2);

        return topDistance1;
    }
}
```