1791. Find Center of Star Graph

Easy Graph

Problem Description

The task involves identifying the center of an undirected star graph composed of `n` nodes, which are numbered from `1` to `n`. A star graph has one central node and exactly `n - 1` edges connecting this central node to all other nodes. In simpler terms, a star graph looks like a star, where every point or tip of the star is connected to a central or middle point.

You are given an array `edges`, where each element `edges[i]` represents an edge in the graph and consists of two integers `[u_i, v_i]`. These integers denote an edge connecting the nodes `u_i` and `v_i`. The goal is to figure out which node is the center of the star graph.

This approach works because of two key properties of the star graph:

1. The center node is connected to all other nodes.
2. There is only one center node in a star graph.

Given these properties, the common node between any two edges in the given star graph must be the center.

Solution Approach

The implementation of the solution is quite straightforward and doesn't require complex data structures, patterns, or algorithms. The simplicity of the problem stems from the unique structure of the star graph where the center node is the one common node in all edges.

Here is a walk-through of the implementation based on the provided Python code:

```
class Solution:
    def findCenter(self, edges: List[List[int]]) -> int:
        return edges[0][0] if edges[0][0] in edges[1] else edges[0][1]
```

1. We define a function `findCenter` which takes as an input parameter `edges`, a list of lists, with each sublist representing an edge in the star graph.
2. The function returns either `edges[0][0]` or `edges[0][1]`. These are the two nodes in the first edge. One of these must be the center because every edge in the graph must contain the center node due to the definition of a star graph.
3. We use a simple `if` statement to check if the first node of the first edge `edges[0][0]` is also in the second edge `edges[1]`. If `edges[0][0]` is indeed in the second edge, we return it as it is the center. Otherwise, we return the second node from the first edge `edges[0][1]`.

This implementation essentially leverages the property that the second edge must also contain the center of the star graph. In other words, we are using the fact that the center node will be the only node that repeats in the first two edges.

No additional algorithms or data structures are required, as the star graph's structure ensures that comparing the first two edges is sufficient to identify the center node.

What makes it efficient is that we only need to look at the first two edges. We don't need to check the entire `edges` list, so the time complexity is constant, O(1), since the number of operations does not depend on the number of edges or nodes in the graph.

Example Walkthrough

Let's use a small example to illustrate the solution approach. Suppose we have a star graph with 4 nodes which results in 3 edges and we are given the `edges` array representing these edges as:

```
edges = [[1, 2], [2, 3], [4, 2]]
```

Based on our solution approach, we know that one of the nodes in `edges[0]`, which is `[1, 2]`, must be the center of the star graph. Now we only need to determine which one it is. To do this, we look at `edges[1]`, which is `[2, 3]`.

Step 1: We identify the nodes in the first edge. They are:

- `u1` from `edges[0]` is `1`
- `v1` from `edges[0]` is `2`

Step 2: We examine the second edge, which contains:

- `u2` from `edges[1]` is `2`
- `v2` from `edges[1]` is `3`

Step 3: We determine the common node between the first and second edge:

- `2` is the common node because it is present in both `[1, 2]` and `[2, 3]`.

Since the node `2` is common to both edges, we can deduce that `2` is the center of our star graph. Hence, the solution function will return `2`.

The execution of the code using our example `edges` array would be as follows:

```
solution = Solution()
center = solution.findCenter(edges)
print(center)  # This will print 2, as node 2 is the center of the star graph
```

Our analysis confirms that the first two edges are sufficient to determine the center and that the function `findCenter` behaves as expected. It returns `2`, which is the common node in both edges, and thus the center of the given star graph.

Solution Implementation

Python

```
# Import the List type from typing module for type annotations
from typing import List

class Solution:
    def findCenter(self, edges: List[List[int]]) -> int:
        # The center of a star graph is connected to all other nodes.
        # The center must be one of the two vertices in the first edge.
        # Check if the first vertex of the first edge is in the second edge:
        if edges[0][0] in edges[1]:
            # If the first vertex is in the second edge,
            # it's the center and return it.
            return edges[0][0]
        else:
            # Otherwise, the second vertex of the first edge is the center, return it.
            return edges[0][1]
```

Java

```
class Solution {
    public int findCenter(int[][] edges) {
        // Extract the first edge's vertices
        int firstVertexOfFirstEdge = edges[0][0];
        int secondVertexOfFirstEdge = edges[0][1];

        // Extract the second edge's vertices
        int firstVertexOfSecondEdge = edges[1][0];
        int secondVertexOfSecondEdge = edges[1][1];

        // The center of a star graph is connected to all other vertices.
        // Therefore, the center must be one of the vertices in the first edge.
        // We check if the first vertex of the first edge is the same as any vertex in the second edge.
        if (firstVertexOfFirstEdge == firstVertexOfSecondEdge || firstVertexOfFirstEdge == secondVertexOfSecondEdge) {
            // If true, then the first vertex of the first edge is the center.
            return firstVertexOfFirstEdge;
        } else {
            // If not, then the second vertex of the first edge must be the center.
            return secondVertexOfFirstEdge;
        }
    }
}
```

C++

```
#include <vector>

class Solution {
public:
    // Finds the center of a star graph.
    // The star graph has one node in the center connected to all other nodes.
    // edges are provided as pairs indicating two nodes that are connected.
    int findCenter(vector<vector<int>>& edges) {
        // Extract node values from the first edge
        int firstNodeOfFirstEdge = edges[0][0];
        int secondNodeOfFirstEdge = edges[0][1];

        // Extract node values from the second edge
        int firstNodeOfSecondEdge = edges[1][0];
        int secondNodeOfSecondEdge = edges[1][1];

        // The center node is connected to every other node, so it will appear in both edges.
        // If the first node of the first edge is the center, it should be equal to either node of the second edge.
        // Otherwise, the center is the other node of the first edge.
        if (firstNodeOfFirstEdge == firstNodeOfSecondEdge || firstNodeOfFirstEdge == secondNodeOfSecondEdge) {
            // The first node of the first edge is the center.
            return firstNodeOfFirstEdge;
        } else {
            // The second node of the first edge is the center.
            return secondNodeOfFirstEdge;
        }
    }
};
```

TypeScript

```
// Finds the center of a star graph
// @param {number[][]} edges - A 2D array containing pairs of connected nodes
// @returns {number} The value of the central node
function findCenter(edges: number[][]): number {
    // Loop through each node in the first pair of connected nodes
    for (let node of edges[0]) {
        // Check if the node is also a part of the second pair of connected nodes
        if (edges[1].includes(node)) {
            // If it is, then it is the center node, return this node
            return node;
        }
    }
    // If not found (though theoretically for a star graph it should always return in the loop), return 0
    // As per LeetCode's constraints, there will always be a center, so the function will always return within the loop
    return 0;
}
```

```
# Import the List type from typing module for type annotations
from typing import List

class Solution:
    def findCenter(self, edges: List[List[int]]) -> int:
        # The center of a star graph is connected to all other nodes.
        # The center must be one of the two vertices in the first edge.
        # Check if the first vertex of the first edge is in the second edge:
        if edges[0][0] in edges[1]:
            # If the first vertex is in the second edge,
            # it's the center and return it.
            return edges[0][0]
        else:
            # Otherwise, the second vertex of the first edge is the center, return it.
            return edges[0][1]
```

Time and Space Complexity

The time complexity of the given code is `O(1)`. This is because the code only checks for the presence of `edges[0][0]` in `edges[1]`, which is a constant-time operation since it involves two fixed index accesses and one `in` operation on a list with only two elements.

The space complexity of the code is also `O(1)`. The code does not use any additional space that grows with the input size. It only uses a fixed amount of space for the variables in the function, regardless of the input size.