1761. Minimum Degree of a Connected Trio in a Graph

Hard Graph Leetcode Link

Problem Description

The problem requires finding the minimum degree of a connected trio in an undirected graph. A connected trio is a set of three nodes where there is an edge connecting every pair of nodes within the set. The degree of the connected trio is the total number of edges that are connected to the trio's nodes where the other endpoint of these edges is not within the trio itself.

You are given:

- An integer `n` indicating the total number of nodes in the graph.
- An array `edges` where each element is a pair `[u, v]`, representing an undirected edge between nodes `u` and `v`.

The task is to calculate the minimum degree among all possible connected trios in the graph. If the graph contains no connected trios, the function should return `-1`.

Intuition

To solve this problem, we need to find all possible connected trios and determine their degrees. We then find the minimum degree among them.

The intuition behind the solution is to:

1. Create a graph representation that allows us to quickly check if an edge exists between any two nodes.
2. Count the degree of each node, which is the number of edges connected to it.
3. Iterate through all possible combinations of three nodes and check if they form a connected trio.
4. If a connected trio is found, compute its degree by adding up the degrees of the three nodes, then subtracting the six edges that form the trio from the calculation because these internal edges should not count towards the degree of the trio.
5. Keep track of the minimum degree found.

The algorithm uses a 2D list `g` that acts as an adjacency matrix with a boolean value indicating whether an edge exists between any two nodes (indexed by `u-1` and `v-1` to convert to zero-based indices). Another list `deg` holds the degree count for each node.

The solution iterates over three nested loops to consider every possible trio of nodes (`i`, `j`, `k`), checking if `g[i][j]`, `g[i][k]`, and `g[j][k]` are all True, which would indicate a connected trio. Upon finding a trio, it calculates the degree of the trio and updates the minimum answer (`ans`) accordingly, ensuring to subtract six to exclude the internal edges of the trio.

The answer (`ans`) is initially set to infinity (a value representing an unreachable high number) to allow it to be updated to any lower valid degree value found during the iterations. If after all the iterations, the `ans` value remains infinity, it means that no trios were found, and thus `-1` is returned. Otherwise, the minimum degree found is returned.

Solution Approach

The solution approach consists of several key steps implemented through the use of effective data structures and algorithms:

Graph Representation:

- The solution uses an adjacency matrix `g` to represent the graph, where `g[i][j]` holds a boolean value indicating the presence of an edge between nodes `i` and `j`.
- The adjacency matrix is chosen for its ease of checking if a pair of nodes is directly connected.
- This matrix is populated in the first loop where we iterate over the `edges` and mark `True` for the corresponding positions in the matrix `g[u][v]` and `g[v][u]`, as well as increment the degree `deg[u]` and `deg[v]` for the ends of each edge.

Degree Calculation:

- An array `deg` of size `n` is used to keep track of the degree of each node (i.e., the number of edges connected to each node).
- While setting up the adjacency matrix, the degree of each node involved in an edge is incremented, thus populating the `deg` array.

Finding Connected Trios and Calculating Degrees:

- Three nested loops over the nodes are used to test each possible trio (a combination of three nodes).
- For each potential trio, the solution checks if all three possible edges between them exist. This is done using the adjacency matrix. If `g[i][j]`, `g[i][k]`, and `g[j][k]` are all `True`, a trio is found.
- For each connected trio, its degree is calculated by summing up `deg[i]`, `deg[j]`, and `deg[k]`, and then subtracting `6` to exclude the edges within the trio itself.
- The intuition behind subtracting `6` is that each edge within the trio would have been counted twice - once for each node it connects.

Finding the Minimum Degree:

- The variable `ans` is used to keep the minimum degree found. It is initialized to `inf` (infinity) to ensure that the first valid degree calculated will be less than this initial value.
- Once a connected trio is found and its degree is calculated, `ans` is updated if the current trio's degree is lower than the previously stored value.
- After checking all possible trios, if `ans` is still `inf`, it means no connected trios exist, and therefore `-1` is returned.
- If at least one connected trio is found, `ans` will hold the minimum degree of a connected trio, which is returned.

Through the use of an adjacency matrix and the precomputation of node degrees, the implementation efficiently finds the connected trios and computes their degrees. This approach is effective for the problem as it minimizes the number of operations needed to determine the existence of edges within possible trios.

Example Walkthrough

Let's walk through the solution approach with a small example:

Suppose we are given 4 nodes `(n = 4)` and the following edges in a graph: `edges = [[1, 2], [2, 3], [3, 1], [2, 4], [3, 4]]`. This graph contains multiple connected trios. Now, let's apply the steps of our solution approach to this example:

1. **Graph Representation**: We first create an adjacency matrix `g` of size `n x n` and initialize it with `False` values. We also create a `deg` array to keep track of the degrees of each node. Since `n = 4`, we will have a 4×4 matrix and a degree array with 4 elements.

2. **Populate the Matrix and Degree Array**:

   - We iterate over the given `edges` to fill in the adjacency matrix and update the degrees.
   - After iterating through all edges, our `g` matrix looks like this:

	1	2	3	4
1	F	T	T	F
2	T	F	T	T
3	T	T	F	T
4	F	T	T	F

   - Our `deg` array is `[2, 3, 3, 2]`, because node `1` has 2 edges, node `2` has 3, and so on.
3. **Finding Connected Trios and Calculating Degrees**:

   - Now, we look for all possible connected trios with 3 nested loops.
   - We check the possible connected trio (1,2,3). Since `g[1][2]`, `g[2][3]`, and `g[3][1]` are all `True`, we have found a connected trio.
   - The degree of this trio is the sum of degrees of nodes `1`, `2`, and `3`, which is `2+3+3 = 8`, then we subtract `6` (the internal edges of the trio), resulting in `2`.
   - Next, we consider the trio (1,2,4), (1,3,4), and (2,3,4). Of these, only (2,3,4) is a connected trio.
   - For the connected trio (2,3,4), we sum up their degrees `3+3+2 = 8` and subtract `6`, which gives us a degree of `2`.
4. **Finding the Minimum Degree**:

   - We initialize `ans` to infinity (or an appropriately large number).
   - As we find connected trios, we update `ans` with the minimum degree found. With the connected trios we found, `ans` becomes `2`.
   - Since we found at least one connected trio, we don't return `-1`. The smallest degree among all connected trios is `2`, which is our final answer.

By following these steps with our example graph, we determined that the minimum degree of a connected trio is `2`.

Python Solution

```python
from typing import List
from math import inf

class Solution:
    def min_trio_degree(self, n: int, edges: List[List[int]]) -> int:
        # Initialize adjacency matrix and degrees list
        graph = [[False] * n for _ in range(n)]
        degrees = [0] * n

        # Populate the adjacency matrix and calculate the degrees
        for u, v in edges:
            u, v = u - 1, v - 1
            graph[u][v] = graph[v][u] = True
            degrees[u] += 1
            degrees[v] += 1

        # Initialize the minimum trio degree to infinity
        min_degree = inf

        # Iterate through possible trios to find the one with the minimum degree
        for i in range(n):
            for j in range(i + 1, n):
                if graph[i][j]:
                    for k in range(j + 1, n):
                        if graph[i][k] and graph[j][k]:
                            # Calculate the trio degree by summing degrees of the nodes
                            # and subtracting 6 (because we are double counting the edges in the trio)
                            current_degree = degrees[i] + degrees[j] + degrees[k] - 6
                            # Update the minimum trio degree if necessary
                            min_degree = min(min_degree, current_degree)

        # If no trio is found, return -1, otherwise return the minimum trio degree
        return -1 if min_degree == inf else min_degree

```

Java Solution

```java
public class Solution {
    public int minTrioDegree(int n, int[][] edges) {
        // Create an adjacency matrix to represent the graph
        boolean[][] graph = new boolean[n][n];
        // An array to store the degree of each vertex
        int[] degrees = new int[n];
        // Fill the adjacency matrix and compute the degrees
        for (int[] edge : edges) {
            int vertexU = edge[0] - 1;
            int vertexV = edge[1] - 1;
            graph[vertexU][vertexV] = true;
            graph[vertexV][vertexU] = true;
            // Increment the degree for each vertex of the edge
            degrees[vertexU]++;
            degrees[vertexV]++;
        }

        // Set an initial high value as the minimum trio degree
        int minTrioDegree = Integer.MAX_VALUE;
        // Iterate over all possible triples of vertices to check for trios
        for (int i = 0; i < n; ++i) {
            for (int j = i + 1; j < n; ++j) {
                if (graph[i][j]) { // Check if there is an edge between vertex i and j
                    for (int k = j + 1; k < n; ++k) {
                        // Check if there is a trio (cycle of three vertices)
                        if (graph[i][k] && graph[j][k]) {
                            // The degree of a trio is the sum of degrees of the vertices minus 6 (each edge in the trio is counted t
                            int trioDegree = degrees[i] + degrees[j] + degrees[k] - 6;
                            // Update the minimum trio degree
                            minTrioDegree = Math.min(minTrioDegree, trioDegree);
                        }
                    }
                }
            }
        }
        // If no trio is found, return -1; otherwise, return the minimum trio degree
        return minTrioDegree == Integer.MAX_VALUE ? -1 : minTrioDegree;
    }
}

```

C++ Solution

```cpp
#include <vector>
#include <cstring>
#include <climits>
using namespace std;

class Solution {
public:
    // Function to calculate the minimum trio degree in the graph
    int minTrioDegree(int n, vector<vector<int>>& edges) {
        // Initialize a graph represented as an adjacency matrix
        bool graph[n][n];
        memset(graph, 0, sizeof graph); // Set all values in graph to 0 (false)

        // Initialize degrees array where each element represents the degree of the corresponding node
        int degree[n];
        memset(degree, 0, sizeof degree); // Set all values in degree to 0

        // Iterate over all edges to fill the graph and degree data
        for (auto& edge : edges) {
            int u = edge[0] - 1; // Adjust index to be zero-based
            int v = edge[1] - 1; // Adjust index to be zero-based
            graph[u][v] = graph[v][u] = true; // Mark the edge in the adjacency matrix (undirected)
            degree[u]++, degree[v]++; // Increment the degree of each node involved in the edge
        }

        int ans = INT_MAX; // Initialize minimum trio degree as maximum possible integer value

        // Find all trios (triplet of nodes that form a triangle), and calculate their degrees
        for (int i = 0; i < n; ++i) {
            for (int j = i + 1; j < n; ++j) {
                if (graph[i][j]) { // Check if edge exists between nodes i and j
                    for (int k = j + 1; k < n; ++k) {
                        // Check if the trio forms a triangle
                        if (graph[j][k] && graph[i][k]) {
                            // Calculate trio degree (sum of degrees of nodes) and subtract 6 (for internal edges of trio)
                            ans = min(ans, degree[i] + degree[j] + degree[k] - 6);
                        }
                    }
                }
            }
        }

        // Return the minimum trio degree found, or -1 if no trio exists
        return ans == INT_MAX ? -1 : ans;
    }
};

```

Typescript Solution

```typescript
/**
 * Function to find the minimum trio degree in a graph.
 * The trio degree is defined as the number of edges connected to the trio nodes in addition to the trio itself.
 * A trio is a set of three nodes where every pair is connected by an edge.
 *
 * @param n - The number of nodes in the graph.
 * @param edges - The list of edges in the graph.
 * @returns The minimum trio degree, or -1 if no trio exists.
 */
function minTrioDegree(n: number, edges: number[][]): number {
    // Initialize a 2D array to represent the graph adjacency matrix
    const adjacencyMatrix = Array.from({ length: n }, () => Array(n).fill(false));
    // Initialize an array to track the degree of each node
    const degree: number[] = Array(n).fill(0);

    // Fill the adjacency matrix and update the degree of each node
    for (const [u, v] of edges) {
        const node1 = u - 1;
        const node2 = v - 1;
        adjacencyMatrix[node1][node2] = adjacencyMatrix[node2][node1] = true;
        ++degree[node1];
        ++degree[node2];
    }

    // Initialize the minimum trio degree to Infinity for comparison
    let minimumTrioDegree = Infinity;

    // Iterate over all possible trios in the graph
    for (let i = 0; i < n; ++i) {
        for (let j = i + 1; j < n; ++j) {
            if (adjacencyMatrix[i][j]) { // Check if nodes i and j are connected
                for (let k = j + 1; k < n; ++k) {
                    // If a trio is found, calculate its degree
                    if (adjacencyMatrix[i][k] && adjacencyMatrix[j][k]) {
                        minimumTrioDegree = Math.min(minimumTrioDegree, degree[i] + degree[j] + degree[k] - 6);
                    }
                }
            }
        }
    }

    // If a trio was found, return its minimum degree; otherwise, return -1
    return minimumTrioDegree === Infinity ? -1 : minimumTrioDegree;
}

```

Time and Space Complexity

Time Complexity

The given Python code consists of a triple nested loop. Let's break it down:

- Building the adjacency matrix `g` and the degree array `deg` takes `O(E)` time, where `E` is the number of edges, because it iterates over all edges once (`for u, v in edges`) and sets the corresponding elements in `g` and increments the degrees in `deg`.

- The outermost loop (`for i in range(n)`) runs `n` times, where `n` is the number of nodes.

- The second loop (`for j in range(i + 1, n)`) runs at most `(n - 1)` times, decrementing by `1` with each iteration of the outer loop.

- The innermost loop (`for k in range(j + 1, n)`) runs at most `(n - 2)` times, decrementing by `1` for each iteration of the second loop.

- Inside the innermost loop, the code checks if three nodes form a trio (a triangle in the graph) with `if g[i][k] and g[j][k]:`, which is a constant time operation.

These nested loops lead to a time complexity of `O(n^3)` in the worst case, as each node is checked with every other pair of nodes to determine if they form a trio.

Space Complexity

The space complexity of the code can also be analyzed:

- The adjacency matrix `g` takes `O(n^2)` space as it is a 2D matrix with dimensions `n` by `n`.

- The degree array `deg` takes `O(n)` space since it stores the degree for each node.

No other significant storage is being used. Therefore, the overall space complexity is `O(n^2)` due to the adjacency matrix `g`.

In conclusion, the time complexity of this code is `O(n^3)` and the space complexity is `O(n^2)`.