LeetCode 1971 - Find if Path Exists in Graph - BFS

Report Issue

Video Solution

In this video solution, we'll use a BFS approach to solve the problem of finding whether a path exists from the source vertex to the target vertex on an undirected graph.

Implementation

C++ Java Python3 Copy

```cpp
class Solution {
public:
    bool validPath(int n, vector<vector<int>>& edges, int start, int end) {
        vector<vector<int>> adjacency_list(n);
        for (vector<int> edge : edges) {
            adjacency_list[edge[0]].push_back(edge[1]);
            adjacency_list[edge[1]].push_back(edge[0]);
        }

        queue<int> q;
        q.push(start);
        vector<bool> seen(n);
        seen[start] = true;

        while (!q.empty()) {
            // Get the current node.
            int node = q.front();
            q.pop();

            // Check if we have reached the target node.
            if (node == end) {
                return true;
            }

            // Add all neighbors to the stack.
            for (int neighbor : adjacency_list[node]) {
                // Check if neighbor has been added to the queue before
```

Complexity Analysis

- Time Complexity: $O(V + E)$. Here, V represents the number of vertices and E represents the number of edges.
    - To create the adjacency list, we must iterate over each of the E edges.
    - In the while loop, at most we will visit vertex once.
    - The for loop inside the while loop will have a cumulative sum of at most E iterations since it will iterate over all of the node's neighbors for each node.

- Space Complexity: $O(V + E)$.
    - The adjacency list, will contain $O(V + E)$ elements.
    - The queue will also contain $O(V)$ elements.
    - The `seen` set will use $O(V)$ space to store the visited nodes.