1971. Find if Path Exists in Graph

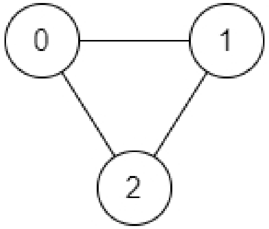
Easy　Topics　Companies

There is a **bi-directional** graph with `n` vertices, where each vertex is labeled from `0` to `n - 1` (**inclusive**). The edges in the graph are represented as a 2D integer array `edges`, where each `edges[i] = [u_i, v_i]` denotes a bi-directional edge between vertex `u_i` and vertex `v_i`. Every vertex pair is connected by **at most one** edge, and no vertex has an edge to itself.

You want to determine if there is a **valid path** that exists from vertex `source` to vertex `destination`.

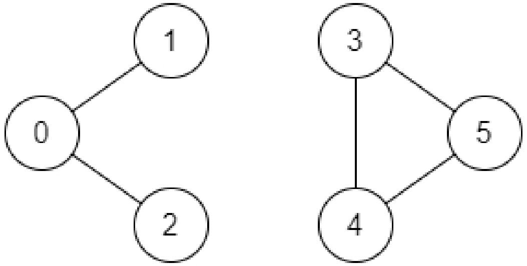
Given `edges` and the integers `n`, `source`, and `destination`, return `true` *if there is a **valid path** from* `source` *to* `destination`*, or* `false` *otherwise.*

Example 1:

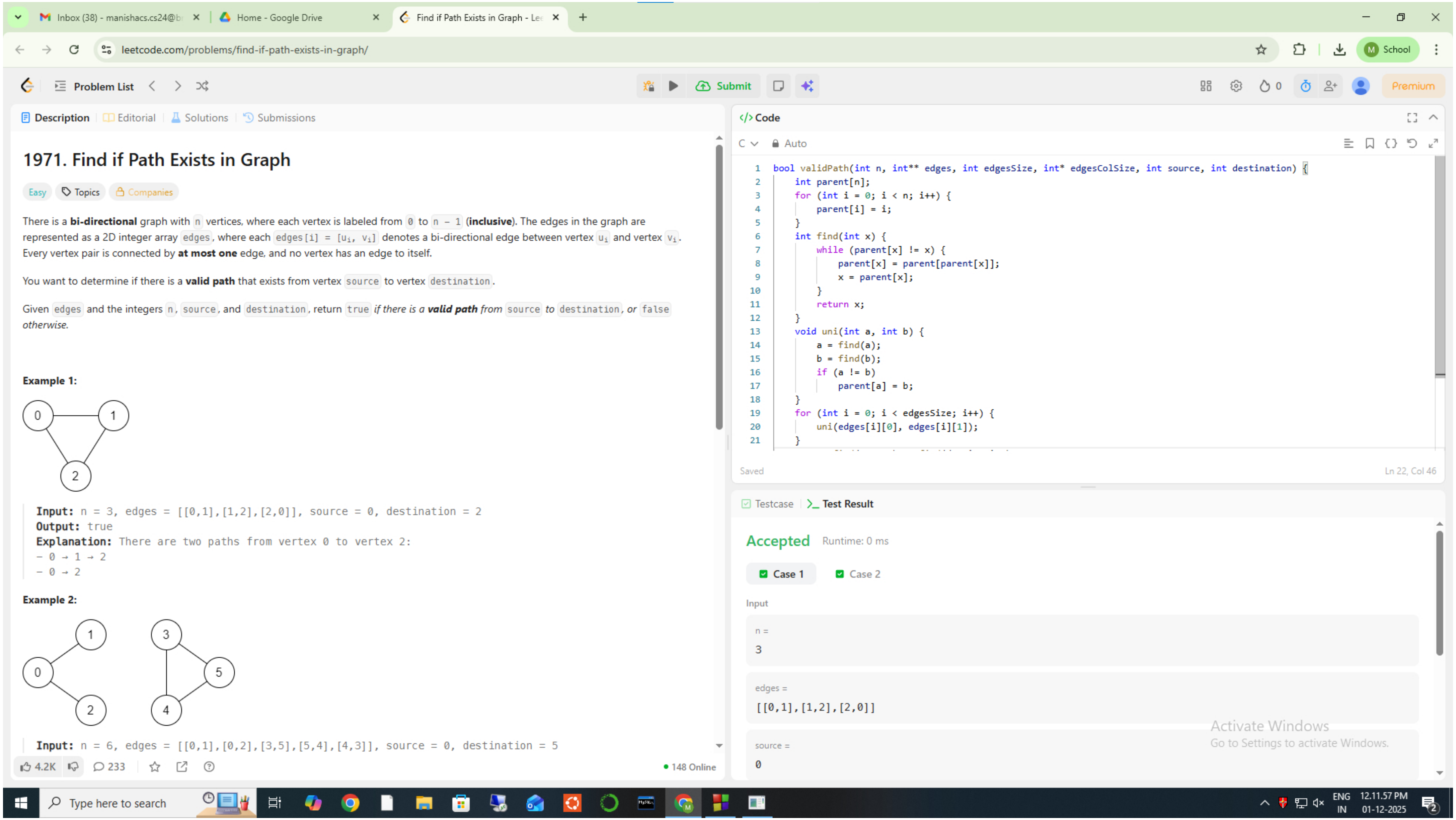


```
Input: n = 3, edges = [[0,1],[1,2],[2,0]], source = 0, destination = 2
Output: true
Explanation: There are two paths from vertex 0 to vertex 2:
- 0 → 1 → 2
- 0 → 2
```

Example 2:



```
Input: n = 6, edges = [[0,1],[0,2],[3,5],[5,4],[4,3]], source = 0, destination = 5
```



1971. Find if Path Exists in Graph

Easy Topics Companies

There is a **bi-directional** graph with `n` vertices, where each vertex is labeled from `0` to `n - 1` (**inclusive**). The edges in the graph are represented as a 2D integer array `edges`, where each `edges[i]` = [u_i, v_i] denotes a bi-directional edge between vertex u_i and vertex v_i. Every vertex pair is connected by **at most one** edge, and no vertex has an edge to itself.

You want to determine if there is a **valid path** that exists from vertex `source` to vertex `destination`.

Given `edges` and the integers `n`, `source`, and `destination`, return `true` *if there is a **valid path** from* `source` *to* `destination`*, or* `false` *otherwise.*

Example 1:

```
Input: n = 3, edges = [[0,1],[1,2],[2,0]], source = 0, destination = 2
Output: true
Explanation: There are two paths from vertex 0 to vertex 2:
- 0 → 1 → 2
- 0 → 2
```

Example 2:

```
Input: n = 6, edges = [[0,1],[0,2],[3,5],[5,4],[4,3]], source = 0, destination = 5
```

```c
bool validPath(int n, int** edges, int edgesSize, int* edgesColSize, int source, int destination) {
    int parent[n];
    for (int i = 0; i < n; i++) {
        parent[i] = i;
    }
    int find(int x) {
        while (parent[x] != x) {
            parent[x] = parent[parent[x]];
            x = parent[x];
        }
        return x;
    }
    void uni(int a, int b) {
        a = find(a);
        b = find(b);
        if (a != b)
            parent[a] = b;
    }
    for (int i = 0; i < edgesSize; i++) {
        uni(edges[i][0], edges[i][1]);
    }
```

Accepted Runtime: 0 ms

☑ Case 1 ☑ Case 2

Input

n =

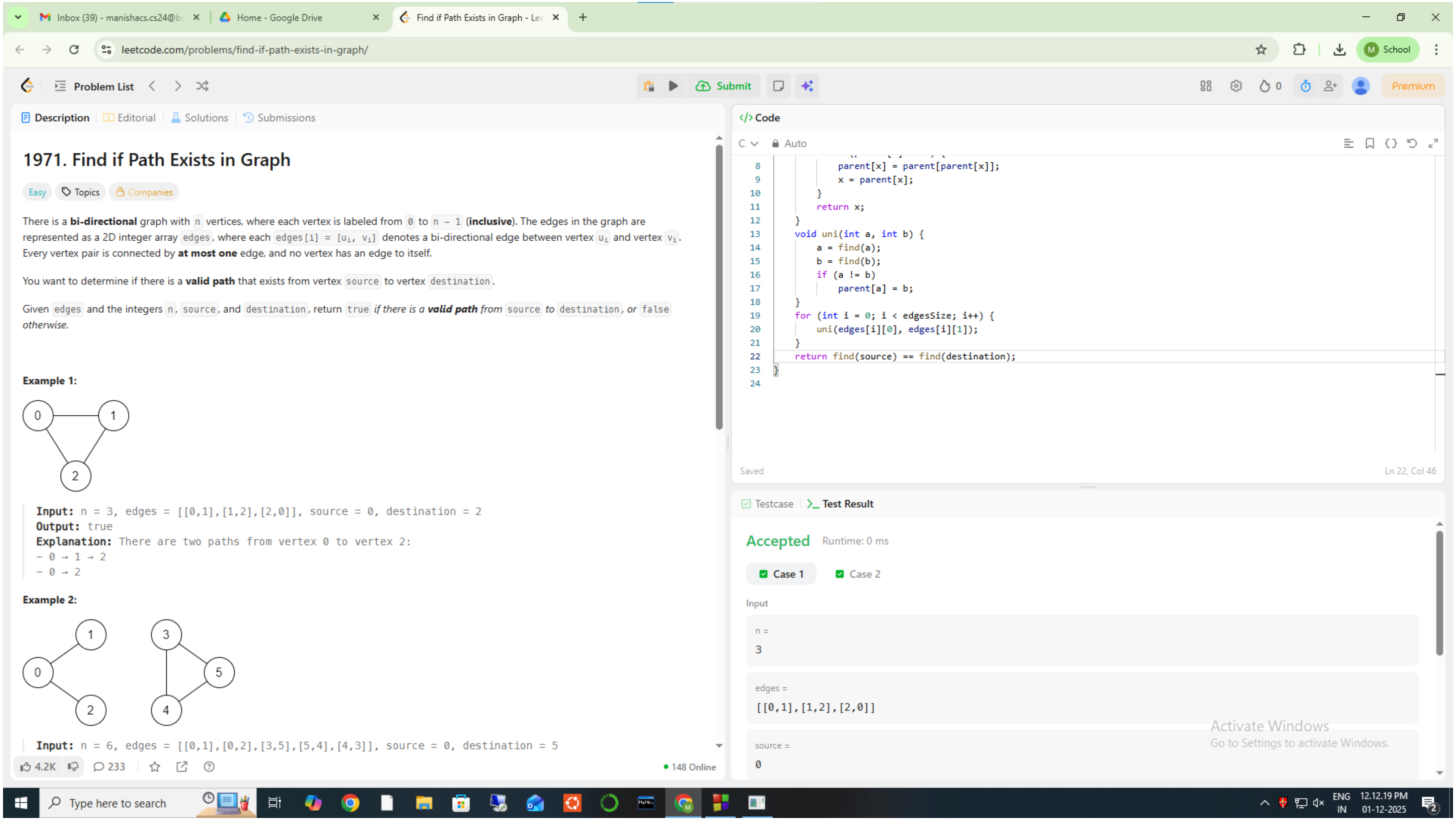
3

edges =

[[0,1],[1,2],[2,0]]

source =

0



1971. Find if Path Exists in Graph

Easy Topics Companies

There is a **bi-directional** graph with `n` vertices, where each vertex is labeled from `0` to `n - 1` (**inclusive**). The edges in the graph are represented as a 2D integer array `edges`, where each `edges[i] = [u_i, v_i]` denotes a bi-directional edge between vertex `u_i` and vertex `v_i`. Every vertex pair is connected by **at most one** edge, and no vertex has an edge to itself.

You want to determine if there is a **valid path** that exists from vertex `source` to vertex `destination`.

Given `edges` and the integers `n`, `source`, and `destination`, return `true` *if there is a* ***valid path*** *from* `source` *to* `destination`*, or* `false` *otherwise.*

Example 1:

```
Input: n = 3, edges = [[0,1],[1,2],[2,0]], source = 0, destination = 2
Output: true
Explanation: There are two paths from vertex 0 to vertex 2:
- 0 → 1 → 2
- 0 → 2
```

Example 2:

```
Input: n = 6, edges = [[0,1],[0,2],[3,5],[5,4],[4,3]], source = 0, destination = 5
```

4.2K 233 ● 148 Online

C Auto

```c
            parent[x] = parent[parent[x]];
            x = parent[x];
        }
        return x;
    }
    void uni(int a, int b) {
        a = find(a);
        b = find(b);
        if (a != b)
            parent[a] = b;
    }
    for (int i = 0; i < edgesSize; i++) {
        uni(edges[i][0], edges[i][1]);
    }
    return find(source) == find(destination);
}
```

Saved Ln 22, Col 46

Accepted Runtime: 0 ms

Case 1 Case 2

Input

n =
3

edges =
[[0,1],[1,2],[2,0]]

source =
0