FindIfPathExistsInGraph

Joel Boynton

November 7, 2025

1 Problem Description

For this problem we are given a bidirectional graph with an 'n' number of vertices. Each pair of vertices should have no more than one edge connecting the two. There should be no vertices with an edge to itself. In this problem we will be presented with a source and destination vetex which we will need to determine whether there is a valid path between the two. If the path between the two is valid (following the aforementioned rules) then our program should return true, and false otherwise.

2 Solution Description

For the solution I decided on a Breadth-First Search (BFS) approach. In this approach we will start with our *source* vertex. Which will serve as our entrance in to the graph. We move through the graph by first checking the immediate neighbors to the source. After this we then go through each of *their* neighbors navigating away from the source layer by layer. If, during this exploration, we come across the destination we are looking for, then we can determine that a path exists. If we go through each connected vertex, never encountering the desired destination, then we can deduce that no path exists.

3 Initial Attempt to Code the Solution

For this problem we will use a few different data structures: *Adjaceny lists, a queue, and a visited set.* I started by getting the trivial case out of the way (source == destination) and then moved on to building my logic. I start with building my adjaceny list (vector<vector<int» adj) which will be of size *n*. This will then iterate through the edges list for each edge *[u, v]*. For each adj[u], I push back into v (adding v to u's list) and vice versa. After this I initialized my *BFS* by creatin a queue *q* and a boolean vector. I add the source to the queue, and the mark it as visited. While the queue is not empty I loop through each vertex, iterating through each neighbor until I find the destination. If not then *destination* returns false.

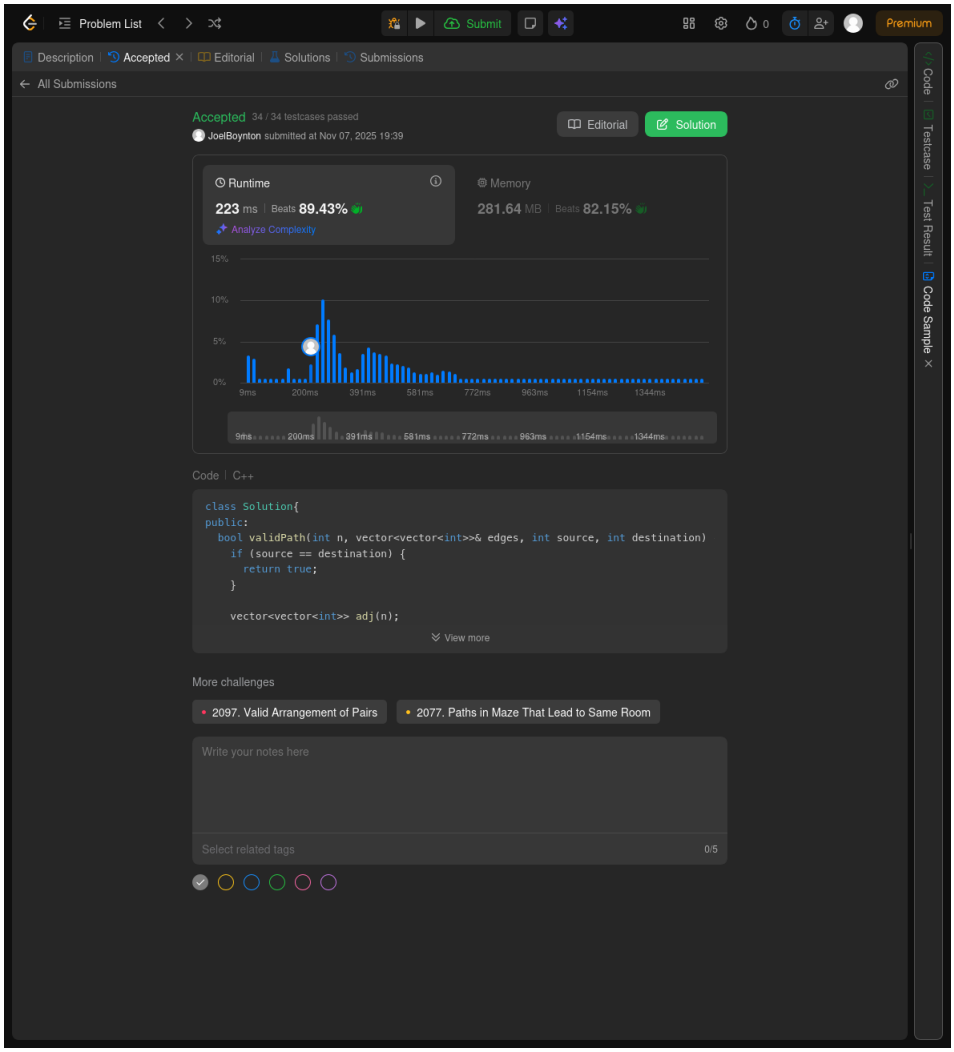
4 Testing Description

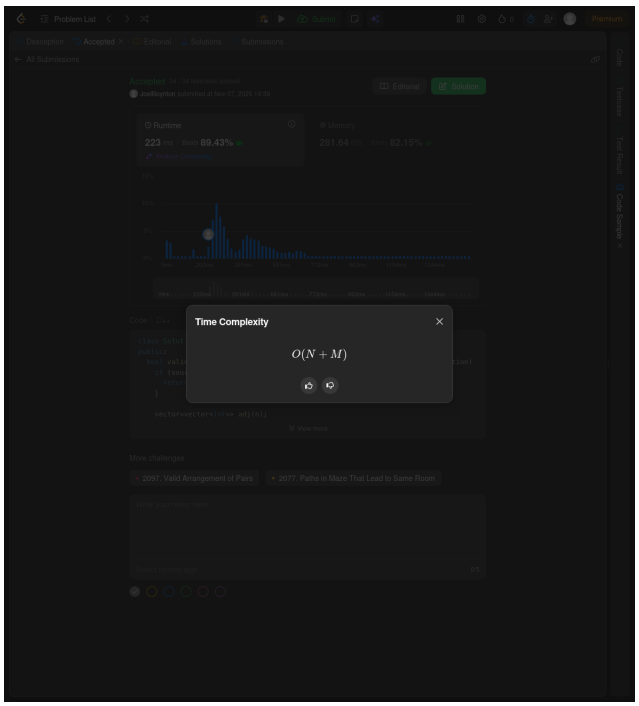
Testing done in LeetCode

5 Code & Testing

```
#include <vector>
#include <queue>
using namespace std;


class Solution{
public:
  bool validPath(int n, vector<vector<int>>& edges, int source, int destination) {
    if (source == destination) {
      return true;
    }

    vector<vector<int>> adj(n);
    for(const auto& edge : edges) {
      int u = edge[0];
      int v = edge[1];
      adj[u].push_back(v);
      adj[v].push_back(u);
    }

    queue<int> q;
    vector<bool> visited(n, false);

    q.push(source);
    visited[source] = true;

    while(!q.empty()) {
      int u = q.front();
      q.pop();

      for(int v : adj[u]) {
        if (v == destination){
          return true;
        }

        if(!visited[v]){
          visited[v] = true;
          q.push(v);
```

```
        }
      }
    }

    return false;
  }
};
```


Problem List
Submit
Premium
Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 34 / 34 testcases passed
JoelBoynton submitted at Nov 07, 2025 19:39
Editorial
Solution
Runtime
223 ms | Beats 89.43%
Analyze Complexity
Memory
281.64 MB | Beats 82.15%
15%
10%
5%
0%
9ms
200ms
391ms
581ms
772ms
963ms
1154ms
1344ms
Code | C++
class Solution{
public:
bool validPath(int n, vector<vector<int>>& edges, int source, int destination)
if (source == destination) {
return true;
}
vector<vector<int>> adj(n);
View more
More challenges
2097. Valid Arrangement of Pairs
2077. Paths in Maze That Lead to Same Room
Write your notes here
Select related tags
0/5
Code
Testcase
Test Result
Code Sample


Time Complexity
$O(N + M)$