



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Day-6

Name:Rajan Mishra
BRANCH: BE-CSE

UID:22BCS13621
SEC-615-IOT

1. Find Center of Star Graph

There is an undirected star graph consisting of n nodes labeled from 1 to n . A star graph is a

graph where there is one center node and exactly $n - 1$ edges that connect the center node with every other node.

You are given a 2D integer array `edges` where each `edges[i] = [ui, vi]` indicates that there is an edge between the nodes `ui` and `vi`. Return the center of the given star graph.

Example 1:

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

Output: 2

Explanation: As shown in the figure above, node 2 is connected to every other node, so 2 is the

Center

Answer:

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

int findCenter(vector<vector<int>>& edges) {
    unordered_map<int, int> count; for (auto&
    edge : edges) { count[edge[0]]++;
    count[edge[1]]++;
    }
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
for (auto& entry : count) { if
    (entry.second == edges.size()) {
        return entry.first;
    }
} return -
1;
}
```

2. Find if Path Exists in Graph

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] = [ui, vi]` denotes a bi-directional edge between vertex `ui` and vertex `vi`.

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:

Answer:

```
#include <vector>
#include <unordered_map>
#include <stack> using
namespace std;
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
bool dfs(int node, int destination, unordered_map<int, vector<int>>& adjList,
vector<bool>& visited) { if (node == destination) return true; visited[node] =
true;
    for (int neighbor : adjList[node]) { if (!visited[neighbor] &&
        dfs(neighbor, destination, adjList, visited)) { return true;
    }
}
return false;
}
```

```
bool validPath(int n, vector<vector<int>>& edges, int source, int destination)
{ unordered_map<int, vector<int>> adjList; for (auto& edge : edges) {
    adjList[edge[0]].push_back(edge[1]); adjList[edge[1]].push_back(edge[0]);
}
vector<bool> visited(n, false);
return dfs(source, destination, adjList, visited);
}
```

3.01 Matrix

Given an m x n binary matrix mat, return the distance of the nearest 0 for each cell.
The distance between two adjacent cells is 1.

Example 1:

Input: mat = [[0,0,0],[0,1,0],[0,0,0]]

Output: [[0,0,0],[0,1,0],[0,0,0]]

Answer:

```
#include <vector>
#include <queue>
using namespace std;
vector<vector<int>> updateMatrix(vector<vector<int>>& mat) {
```



```
int m = mat.size(), n = mat[0].size();
vector<vector<int>> dist(m, vector<int>(n, -1));
queue<pair<int, int>> q; for (int i = 0; i < m; i++)
{ for (int j = 0; j < n; j++) {
    if (mat[i][j] == 0) {
        q.push({i, j}); dist[i][j]
        = 0;
    }
}
}
vector<pair<int, int>> dirs = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}}; while
(!q.empty()) {
    auto [x, y] = q.front();
    q.pop(); for (auto& dir
    : dirs) {
        int newX = x + dir.first, newY = y + dir.second;
        if (newX >= 0 && newX < m && newY >= 0 && newY < n &&
dist[newX][newY] == -1) {
            dist[newX][newY] = dist[x][y] + 1;
            q.push({newX, newY});
        }
    }
}
return dist;
}
```

4. Word Search

Given an m x n grid of characters board and a string word, return true if word exists in the grid.



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

The word can be constructed from letters of sequentially adjacent cells, where adjacent cells are horizontally or vertically neighboring. The same letter cell may not be used more than once.

Example 1:

Input: board = [["A","B","C","E"], ["S","F","C","S"], ["A","D","E","E"]], word = "ABCCED"

Output: true **Answer:**

```
#include <vector>
#include <string>
using namespace std;
bool dfs(int i, int j,
int index, const
vector<vector<char
>>& board, const
string& word)
{ if (index == word.size()) return true;
if (i < 0 || i >= board.size() || j < 0 || j >= board[0].size() || board[i][j] != word[index]) {
return false;
}
char temp = board[i][j]; board[i][j]
= '#';
bool found = dfs(i + 1, j, index + 1, board, word) ||
dfs(i - 1, j, index + 1, board, word) ||
dfs(i, j + 1, index + 1, board, word) || dfs(i, j
- 1, index + 1, board, word); board[i][j] =
temp;
```



```
    return found;
}
bool exist(vector<vector<char>>& board, string word) {
    for (int i = 0; i < board.size(); i++) { for (int j = 0; j <
        board[0].size(); j++) { if (dfs(i, j, 0, board, word)) {
            return true;
        }
    }
}
return false;
}
```

5. Accounts Merge

Given a list of accounts where each element `accounts[i]` is a list of strings, where the first element `accounts[i][0]` is a name, and the rest of the elements are emails representing emails of the account.

Now, we would like to merge these accounts. Two accounts definitely belong to the same person

if there is some common email to both accounts. Note that even if two accounts have the same

name, they may belong to different people as people could have the same name. A person can

have any number of accounts initially, but all of their accounts definitely have the same name.

After merging the accounts, return the accounts in the following format: the first element of each

account is the name, and the rest of the elements are emails in sorted order. The accounts themselves can be returned in any order.



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Example 1: Input:

accounts =

```
[["John", "johnsmith@mail.com", "john_newyork@mail.com"], ["John", "johnsmith@mail.com", "john00@mail.com"], ["Mary", "mary@mail.com"], ["John", "johnnybravo@mail.com"]]
```

Output:

```
[["John", "john00@mail.com", "john_newyork@mail.com", "johnsmith@mail.com"], ["Mary", "mary@mail.com"], ["John", "johnnybravo@mail.com"]]
```

Explanation:

The first and second John's are the same person as they have the common email "johnsmith@mail.com".

The third John and Mary are different people as none of their email addresses are used by other accounts.

We could return these lists in any order, for example the answer [['Mary',

'mary@mail.com'], ['John', 'johnnybravo@mail.com'],

['John', 'john00@mail.com', 'john_newyork@mail.com', 'johnsmith@mail.com']] would still be accepted. **Answer:**

```
#include <vector>
```

```
#include <string>
```

```
#include <unordered_map>
```

```
#include <algorithm> using
```

```
namespace std; class
```

```
UnionFind { public:
```

```
    vector<int> parent;
```

```
    UnionFind(int n) { parent.resize(n);
```

```
        for (int i = 0; i < n; i++) parent[i] = i;
```

```
    }
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
int find(int x) { if (parent[x] != x) { parent[x] =  
    find(parent[x]); // Path compression  
}  
    return parent[x];  
}
```

```
void unionSets(int x, int y) { parent[find(x)] = find(y); // Union by  
rank can be added to optimize }  
};
```

```
vector<vector<string>> accountsMerge(vector<vector<string>>& accounts) {  
    unordered_map<string, int> emailToId; int id = 0;  
    UnionFind uf(accounts.size()); for (int i = 0;  
    i < accounts.size(); i++) { for (int j = 1; j <  
    accounts[i].size(); j++) { string email =  
    accounts[i][j];  
        if (emailToId.find(email) == emailToId.end()) { emailToId[email]  
            = id++;  
        }  
        uf.unionSets(emailToId[accounts[i][1]], emailToId[email]);  
    }  
}  
    unordered_map<int, vector<string>> groupedEmails;  
    for (auto& entry : emailToId) { int  
        root = uf.find(entry.second);  
        groupedEmails[root].push_back(entry.first);  
    }  
}
```




DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
vector<vector<string>> result; for
(auto& group : groupedEmails) {
vector<string> emails =
group.second; sort(emails.begin(),
emails.end());
    emails.insert(emails.begin(), accounts[group.first][0]); // Add name
    result.push_back(emails);
}
return result;
}
```