

## Day 7

**Name: Rohit**

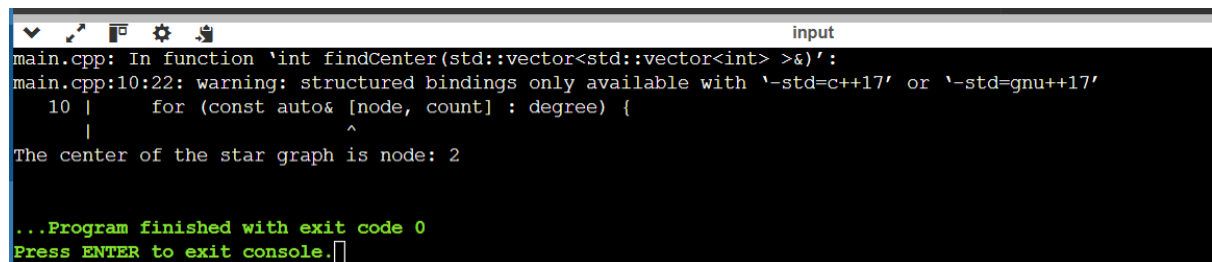
**UID: 22BCS15476**

**Section: 620-B**

### Ques1: Find Center of Star Graph

```
#include <iostream>
#include <vector>
#include <unordered_map>
int findCenter(std::vector<std::vector<int>>& edges) {
    std::unordered_map<int, int> degree;
    for (const auto& edge : edges) {
        degree[edge[0]]++;
        degree[edge[1]]++;
    }
    for (const auto& [node, count] : degree) {
        if (count == edges.size()) {
            return node;
        }
    }
    return -1;
}
int main() {
    std::vector<std::vector<int>> edges = {{1, 2}, {2, 3}, {4, 2}};
    int center = findCenter(edges);
    std::cout << "The center of the star graph is node: " << center << std::endl;
    return 0;
}
```

### Output:

A screenshot of a terminal window showing the execution of a C++ program. The terminal has a dark background with light-colored text. At the top, there's a title bar with icons and the word 'input'. The main text shows the program's output: 'The center of the star graph is node: 2'. Above this, there are some compiler warnings and error messages. At the bottom, it says '...Program finished with exit code 0' and 'Press ENTER to exit console.'.

```
main.cpp: In function 'int findCenter(std::vector<std::vector<int> >&)':
main.cpp:10:22: warning: structured bindings only available with '-std=c++17' or '-std=gnu++17'
    10 |         for (const auto& [node, count] : degree) {
        |             ^
The center of the star graph is node: 2

...Program finished with exit code 0
Press ENTER to exit console.
```

### Ques 2: Find the Town Judge

```
#include <iostream>
```

```

#include <vector>

int findJudge(int n, std::vector<std::vector<int>>& trust) {
    std::vector<int> trustCount(n + 1, 0);
    for (const auto& t : trust) {
        trustCount[t[0]]--;
        trustCount[t[1]]++;
    }
    for (int i = 1; i <= n; ++i) {
        if (trustCount[i] == n - 1) {
            return i;
        }
    }
    return -1;
}

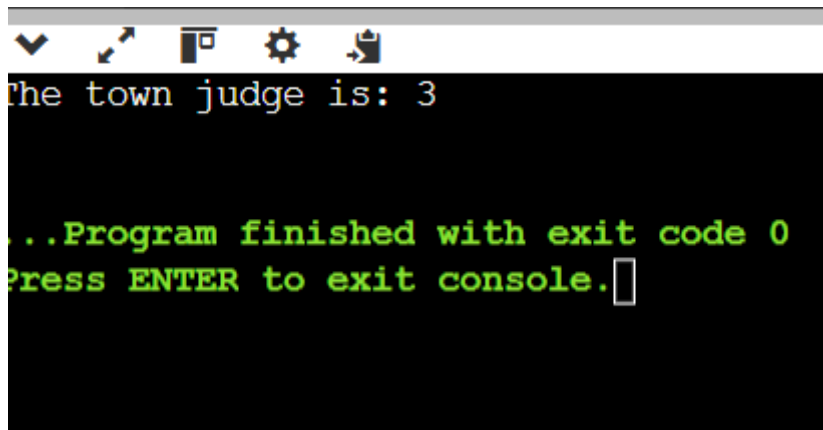
int main() {
    int n = 3;
    std::vector<std::vector<int>> trust = {{1, 3}, {2, 3}};
    int judge = findJudge(n, trust);

    if (judge == -1) {
        std::cout << "No town judge found." << std::endl;
    } else {
        std::cout << "The town judge is: " << judge << std::endl;
    }

    return 0;
}

```

**Output:**



```
The town judge is: 3

...Program finished with exit code 0
Press ENTER to exit console.
```

### Ques 3: Flood Fill - link

```
#include <iostream>
#include <vector>
using namespace std;
void floodFillHelper(vector<vector<int>>& image, int sr, int sc, int color, int originalColor) {
    if (sr < 0 || sr >= image.size() || sc < 0 || sc >= image[0].size() || image[sr][sc] !=
originalColor) {
        return;
    }
    image[sr][sc] = color;
    floodFillHelper(image, sr - 1, sc, color, originalColor);
    floodFillHelper(image, sr + 1, sc, color, originalColor);
    floodFillHelper(image, sr, sc - 1, color, originalColor);
    floodFillHelper(image, sr, sc + 1, color, originalColor);
}
vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc, int color) {
    int originalColor = image[sr][sc];
    if (originalColor != color) {
        floodFillHelper(image, sr, sc, color, originalColor);
    }
    return image;
}
int main() {
    vector<vector<int>> image = {{1,1,1},{1,1,0},{1,0,1}};
    int sr = 1, sc = 1, color = 2;
    vector<vector<int>> result = floodFill(image, sr, sc, color);
    cout << "Modified image after flood fill:" << endl;
    for (const auto& row : result) {
```

```

        for (const auto& pixel : row) {
            cout << pixel << " ";
        }
        cout << endl;
    }
    return 0;
}

```

#### Output:

```

Modified image after flood fill:
2 2 2
2 2 0
2 0 1

...Program finished with exit code 0
Press ENTER to exit console.

```

#### Ques 4: Find if Path Exists in Graph

```

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

bool validPath(int n, vector<vector<int>>& edges, int source, int destination) {
    vector<vector<int>> graph(n);
    for (const auto& edge : edges) {
        graph[edge[0]].push_back(edge[1]);
        graph[edge[1]].push_back(edge[0]);
    }
    vector<bool> visited(n, false);
    queue<int> q;
    q.push(source);
    visited[source] = true;
    while (!q.empty()) {
        int node = q.front();
        q.pop();
        if (node == destination) {
            return true;
        }
    }
}

```

```

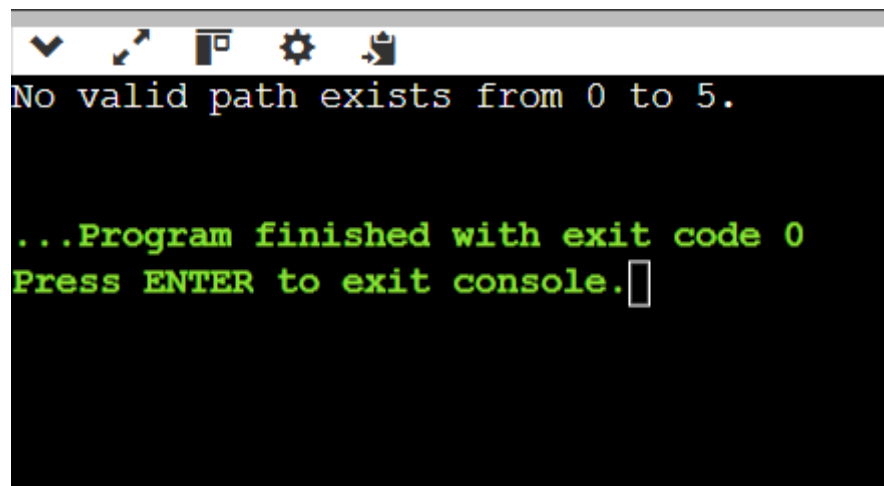
        for (const auto& neighbor : graph[node]) {
            if (!visited[neighbor]) {
                visited[neighbor] = true;
                q.push(neighbor);
            }
        }
    }
    return false;
}

int main() {
    int n = 6;
    vector<vector<int>> edges = {{0, 1}, {0, 2}, {3, 5}, {5, 4}, {4, 3}};
    int source = 0, destination = 5;

    if (validPath(n, edges, source, destination)) {
        cout << "A valid path exists from " << source << " to " << destination << "." << endl;
    } else {
        cout << "No valid path exists from " << source << " to " << destination << "." << endl;
    }
    return 0;
}

```

**Output:**



```

No valid path exists from 0 to 5.

...Program finished with exit code 0
Press ENTER to exit console.

```

**Ques 5: BFS of graph link**

```

#include <iostream>
#include <vector>
#include <queue>

```

```

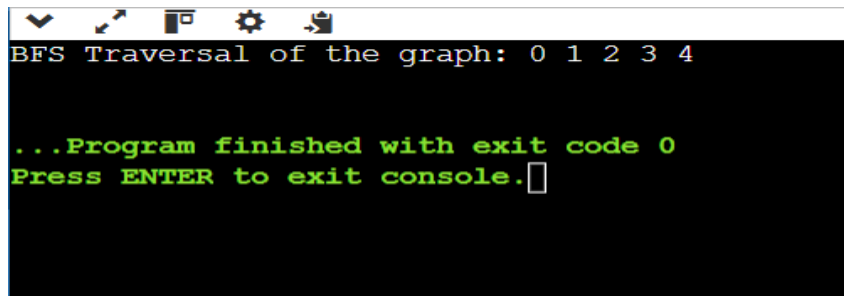
using namespace std;
vector<int> bfsOfGraph(int V, vector<vector<int>>& adj) {
    vector<int> bfsResult;
    vector<bool> visited(V, false);
    queue<int> q;
    q.push(0);
    visited[0] = true;
    while (!q.empty()) {
        int node = q.front();
        q.pop();
        bfsResult.push_back(node);
        for (int neighbor : adj[node]) {
            if (!visited[neighbor]) {
                visited[neighbor] = true;
                q.push(neighbor);
            }
        }
    }
    return bfsResult;
}

int main() {
    int V = 5;
    vector<vector<int>> adj = {
        {1, 2, 3},
        {0, 4},
        {0, 4},
        {0},
        {1, 2}
    };
    vector<int> bfsResult = bfsOfGraph(V, adj);
    cout << "BFS Traversal of the graph: ";
    for (int vertex : bfsResult) {
        cout << vertex << " ";
    }
    cout << endl;
    return 0;
}

```

```
}
```

### Output:

A screenshot of a terminal window with a black background and green text. The text displays the output of a BFS traversal: "BFS Traversal of the graph: 0 1 2 3 4". Below this, it says "...Program finished with exit code 0" and "Press ENTER to exit console." followed by a cursor icon.

```
BFS Traversal of the graph: 0 1 2 3 4

...Program finished with exit code 0
Press ENTER to exit console.
```

### Ques 6: DFS of Graph

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

void dfsHelper(int node, vector<vector<int>>& adj, vector<bool>& visited, vector<int>&
dfsResult) {
    visited[node] = true;
    dfsResult.push_back(node);
    for (int neighbor : adj[node]) {
        if (!visited[neighbor]) {
            dfsHelper(neighbor, adj, visited, dfsResult);
        }
    }
}

vector<int> dfsOfGraph(int V, vector<vector<int>>& adj) {
    vector<int> dfsResult;
    vector<bool> visited(V, false);

    dfsHelper(0, adj, visited, dfsResult);
    return dfsResult;
}

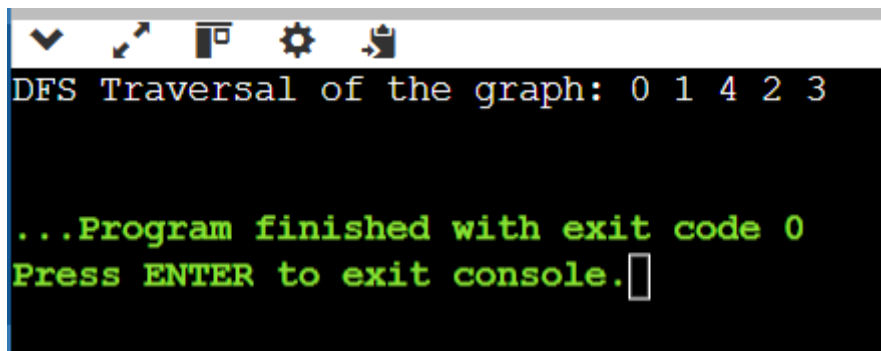
int main() {
    int V = 5;
    vector<vector<int>> adj = {
        {1, 2, 3},
        {0, 4},
        {0, 4},
        {0},
```

```

        {1, 2}
    };
    vector<int> dfsResult = dfsOfGraph(V, adj);
    cout << "DFS Traversal of the graph: ";
    for (int vertex : dfsResult) {
        cout << vertex << " ";
    }
    cout << endl;
    return 0;
}

```

**Output:**



The screenshot shows a console window with a dark background. At the top, there is a toolbar with icons for a dropdown menu, a cursor, a window, a gear, and a clipboard. Below the toolbar, the text "DFS Traversal of the graph: 0 1 4 2 3" is displayed in a monospaced font. Below this, the text "...Program finished with exit code 0" is shown in green. At the bottom, the text "Press ENTER to exit console." is also in green, followed by a small white square icon representing the ENTER key.

**Ques 7: 01 Matrix**

```

#include <iostream>
#include <queue>
#include <climits>
using namespace std;
const int MAX = 1000;
void updateMatrix(int mat[MAX][MAX], int m, int n) {
    int dist[MAX][MAX];
    queue<pair<int, int>> q;
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            if (mat[i][j] == 0) {
                dist[i][j] = 0;
                q.push({i, j});
            } else {
                dist[i][j] = INT_MAX;
            }
        }
    }
}

```



```

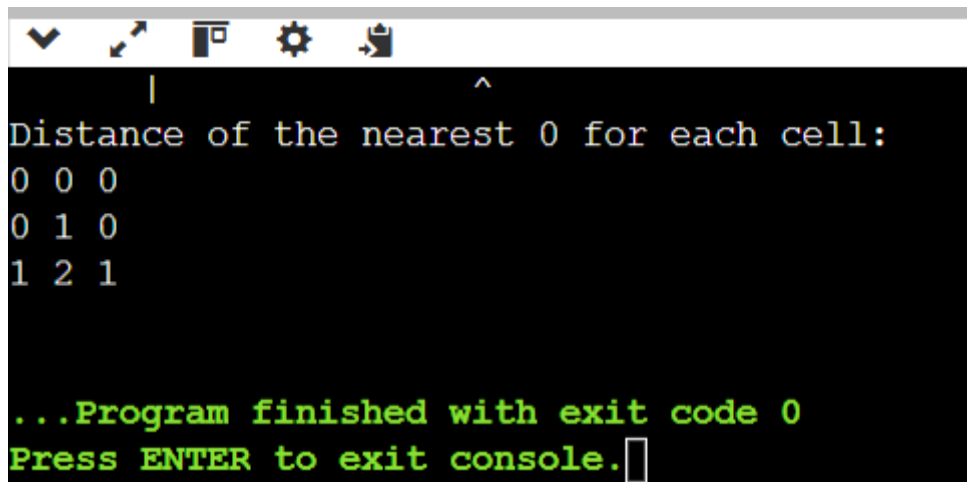
    }
    int directions[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
    while (!q.empty()) {
        auto [x, y] = q.front();
        q.pop();
        for (auto dir : directions) {
            int newX = x + dir[0];
            int newY = y + dir[1];
            if (newX >= 0 && newX < m && newY >= 0 && newY < n && dist[newX][newY] >
dist[x][y] + 1) {
                dist[newX][newY] = dist[x][y] + 1;
                q.push({newX, newY});
            }
        }
    }
}

cout << "Distance of the nearest 0 for each cell: " << endl;
for (int i = 0; i < m; ++i) {
    for (int j = 0; j < n; ++j) {
        cout << dist[i][j] << " ";
    }
    cout << endl;
}
}

int main() {
    int mat[MAX][MAX] = {
        {0, 0, 0},
        {0, 1, 0},
        {1, 1, 1}
    };
    int m = 3, n = 3;
    updateMatrix(mat, m, n);
    return 0;
}

```

**Output:**



```
Distance of the nearest 0 for each cell:
0 0 0
0 1 0
1 2 1

...Program finished with exit code 0
Press ENTER to exit console.
```

### Ques 8: Course Schedule II

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
vector<int> findOrder(int numCourses, vector<vector<int>>& prerequisites) {
    vector<int> inDegree(numCourses, 0);
    vector<vector<int>> adj(numCourses);
    for (const auto& pre : prerequisites) {
        adj[pre[1]].push_back(pre[0]);
        inDegree[pre[0]]++;
    }
    queue<int> q;
    for (int i = 0; i < numCourses; ++i) {
        if (inDegree[i] == 0) {
            q.push(i);
        }
    }
    vector<int> courseOrder;
    while (!q.empty()) {
        int course = q.front();
        q.pop();
        courseOrder.push_back(course);
        for (int nextCourse : adj[course]) {
            inDegree[nextCourse]--;
            if (inDegree[nextCourse] == 0) {
```

```

        q.push(nextCourse);
    }
}
}
if (courseOrder.size() == numCourses) {
    return courseOrder;
} else {
    return {};
}
}
int main() {
    int numCourses = 4;
    vector<vector<int>> prerequisites = {{1, 0}, {2, 0}, {3, 1}, {3, 2}};
    vector<int> order = findOrder(numCourses, prerequisites);

    if (order.empty()) {
        cout << "It is impossible to finish all courses." << endl;
    } else {
        cout << "The order of courses to finish all courses is: ";
        for (int course : order) {
            cout << course << " ";
        }
        cout << endl;
    }
    return 0;
}

```

**Output:**

```

The order of courses to finish all courses is: 0 1 2 3

...Program finished with exit code 0
Press ENTER to exit console.

```

### Ques 9: Word Search

```

#include <iostream>
#include <vector>

```

```

#include <string>
using namespace std;
bool dfs(vector<vector<char>>& board, int i, int j, string& word, int index) {
    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(board, i+1, j, word, index+1) ||
        dfs(board, i-1, j, word, index+1) ||
        dfs(board, i, j+1, word, index+1) ||
        dfs(board, i, j-1, word, index+1);
    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 (board[i][j] == word[0] && dfs(board, i, j, word, 0)) {
                return true;
            }
        }
    }
    return false;
}
int main() {
    vector<vector<char>> board = {
        {'A', 'B', 'C', 'E'},
        {'S', 'F', 'C', 'S'},
        {'A', 'D', 'E', 'E'}
    };
    string word = "ABCCED";
    if (exist(board, word)) {
        cout << "The word exists in the grid." << endl;
    } else {
        cout << "The word does not exist in the grid." << endl;
    }
}

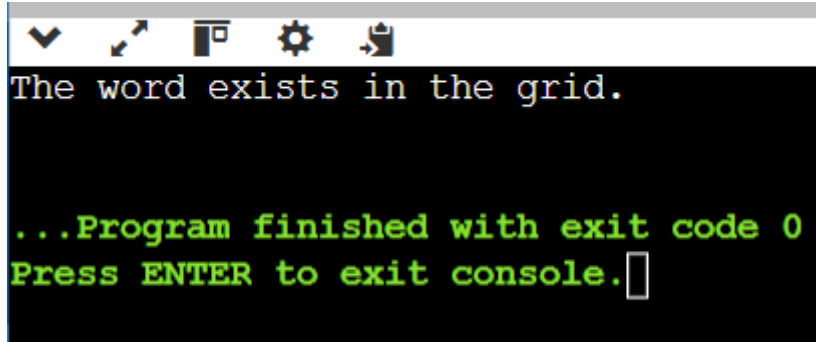
```

```

    }
    return 0;
}

```

**Output:**



```

The word exists in the grid.

...Program finished with exit code 0
Press ENTER to exit console.

```

### **Ques 10: Minimum Height Trees**

```

#include <iostream>
#include <vector>
#include <queue>
using namespace std;
vector<int> findMinHeightTrees(int n, vector<vector<int>>& edges) {
    if (n == 1) return {0};
    vector<vector<int>> adj(n);
    vector<int> degree(n, 0);
    for (const auto& edge : edges) {
        adj[edge[0]].push_back(edge[1]);
        adj[edge[1]].push_back(edge[0]);
        degree[edge[0]]++;
        degree[edge[1]]++;
    }
    queue<int> q;
    for (int i = 0; i < n; ++i) {
        if (degree[i] == 1) {
            q.push(i);
        }
    }
    while (n > 2) {
        int size = q.size();
        n -= size;
        for (int i = 0; i < size; ++i) {

```

```

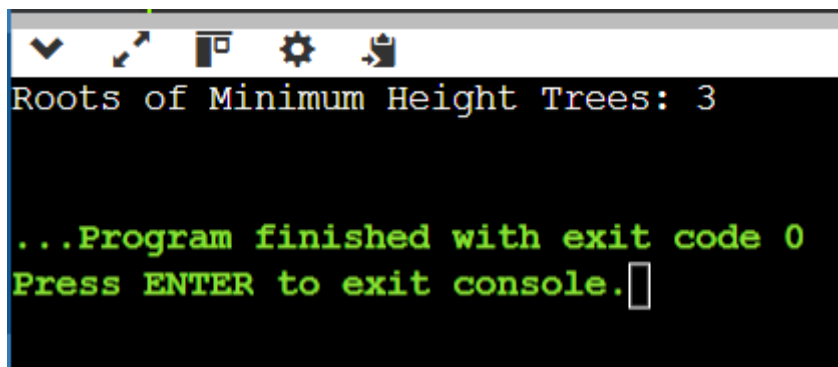
    int node = q.front();
    q.pop();

    for (int neighbor : adj[node]) {
        degree[neighbor]--;
        if (degree[neighbor] == 1) {
            q.push(neighbor);
        }
    }
}
}
}
vector<int> mhts;
while (!q.empty()) {
    mhts.push_back(q.front());
    q.pop();
}
return mhts;
}

int main() {
    int n = 6;
    vector<vector<int>> edges = {{0, 1}, {0, 2}, {0, 3}, {3, 4}, {4, 5}};
    vector<int> mhts = findMinHeightTrees(n, edges);
    cout << "Roots of Minimum Height Trees: ";
    for (int root : mhts) {
        cout << root << " ";
    }
    cout << endl;
    return 0;
}

```

**Output:**

A screenshot of a terminal window. The title bar is light gray and contains five icons: a downward arrow, a magnifying glass, a square, a gear, and a document. The terminal background is black. The text is displayed in a monospaced font. The first line is 'Roots of Minimum Height Trees: 3' in a light blue color. The second line is '...Program finished with exit code 0' in a light green color. The third line is 'Press ENTER to exit console.' in the same light green color, followed by a white cursor box.

```
Roots of Minimum Height Trees: 3

...Program finished with exit code 0
Press ENTER to exit console.
```