LAB REPORT

CSE 114 : Data Structure and Algorithms Sessional

PREPARED BY

Mehrin Farzana
ID: 2101013
Session: 2021-2022
Date: 20/10/2023

SUPERVISED BY

Suman Saha
Lecturer
Department of IRE, BDU

Bangabandhu Sheikh Mujibur Rahman Digital University

BANGABANDHU SHEIKH MUJIBUR RAHMAN DIGITAL UNIVERSITY

(BDU)

List of Problems

1. You are given a 2D grid representing a maze. The maze consists of open cells and walls. Your task is to implement a Breadth-First Search (BFS) algorithm to find the shortest path from a given start cell to a target cell, if one exists. A path is a sequence of open cells from the start to the target, and you can move horizontally or vertically but not diagonally. Implement BFS to find the shortest path, and return the length of the shortest path. If no path exists, return -1.

2. Write a program to find and count the connected components in an undirected graph using DFS. The graph is represented as a list of edges, where each edge is a tuple of two nodes. Implement DFS to traverse the graph and identify connected components.

Problem No.: 01

Problem Statement:

You are given a 2D grid representing a maze. The maze consists of open cells and walls. Your task is to implement a Breadth-First Search (BFS) algorithm to find the shortest path from a given start cell to a target cell, if one exists. A path is a sequence of open cells from the start to the target, and you can move horizontally or vertically but not diagonally. Implement BFS to find the shortest path, and return the length of the shortest path. If no path exists, return -1.

.

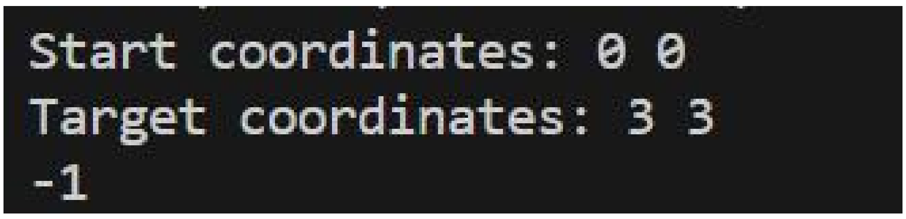
Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#define ROW 4
#define COL 4

int dRow[] = {-1, 0, 1, 0};
int dCol[] = {0, 1, 0, -1};

bool isValid(int row, int col, int maze[][COL], bool visited[][COL])
{
   return (row >= 0) && (row < ROW) && (col >= 0) && (col < COL)
         && maze[row][col] && !visited[row][col];
}

void BFS(int maze[][COL], bool visited[][COL], int startRow, int startCol,
        int targetRow, int targetCol)
{
   if (!maze[startRow][startCol] || !maze[targetRow][targetCol]) {
      printf("-1");
      return;
   }

   bool queue[ROW * COL];
   int front = -1;
```

```
    int rear = -1;

    visited[startRow][startCol] = true;
    queue[++rear] = startRow;
    queue[++rear] = startCol;

    while (front != rear) {
        int currRow = queue[++front];
        int currCol = queue[++front];

        if (currRow == targetRow && currCol == targetCol) {
            printf("%d", visited[currRow][currCol] - 1);
            return;
        }

        for (int i = 0; i < 4; i++) {
            int adjRow = currRow + dRow[i];
            int adjCol = currCol + dCol[i];

            if (isValid(adjRow, adjCol, maze, visited)) {
                visited[adjRow][adjCol] = visited[currRow][currCol] + 1;
                queue[++rear] = adjRow;
                queue[++rear] = adjCol;
            }
        }
    }

    printf("-1");
}

int main()
{
    int maze[ROW][COL] =
            {
                    {1, 0, 0, 0},
                    {1, 1, 0, 1},
                    {0, 1, 1, 0},
                    {1, 1, 1, 1}
            };
    int startRow, startCol, targetRow, targetCol;
    printf("Start coordinates: ");
    scanf("%d%d", &startRow, &startCol);
    printf("Target coordinates: ");
    scanf("%d%d", &targetRow, &targetCol);
    bool visited[ROW][COL];
    memset(visited, false, sizeof(visited));

    BFS(maze, visited, startRow, startCol, targetRow, targetCol);

    return 0;
```

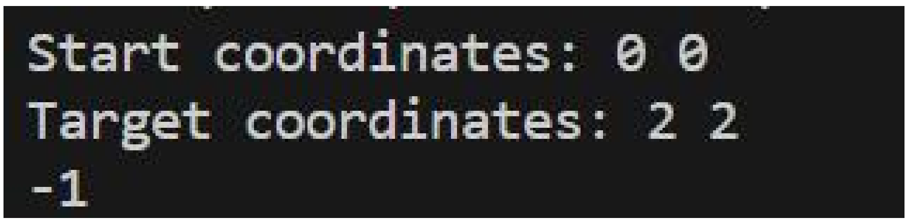
}

Output:



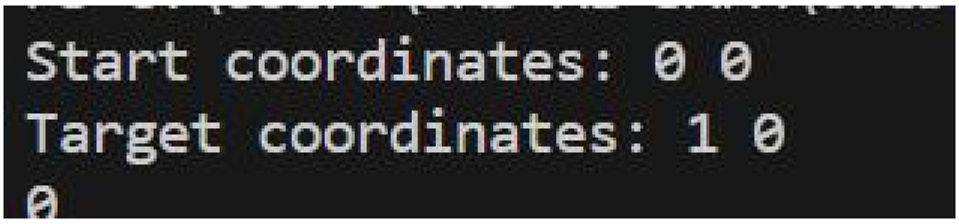
```
Start coordinates: 0 0
Target coordinates: 3 3
-1
```

Fig 1.1: Output on console for case 1.



```
Start coordinates: 0 0
Target coordinates: 2 2
-1
```

Fig 1.2: Output on console for case 2.



```
Start coordinates: 0 0
Target coordinates: 1 0
0
```

Fig 1.3: Output on console for case 3.

Problem No.: 02

Problem Statement:

Write a program to find and count the connected components in an undirected graph using DFS. The graph is represented as a list of edges, where each edge is a tuple of two nodes. Implement DFS to traverse the graph and identify connected components.

.

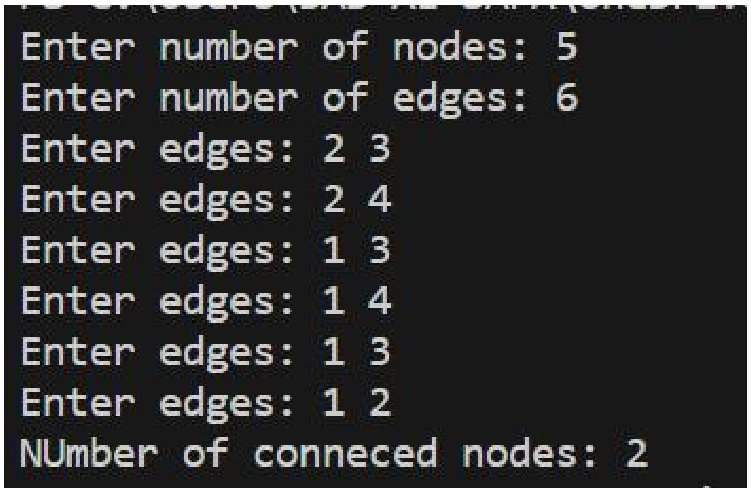
Code:

```
#include <stdio.h>
#include <stdlib.h>

// Stores the parent of each vertex
int parent[1000000];

// Function to find the topmost
// parent of vertex a
int root(int a)
{
   // If current vertex is
   // the topmost vertex
   if (a == parent[a]) {
      return a;
   }

   // Otherwise, set topmost vertex of
   // its parent as its topmost vertex
   return parent[a] = root(parent[a]);
}

// Function to connect the component
// having vertex a with the component
// having vertex b
void connect(int a, int b)
{
   // Connect edges
   a = root(a);
   b = root(b);

   if (a != b) {
      parent[b] = a;
```

```
    }
}

// Function to find unique top most parents
void connectedComponents(int n)
{
    int s[n];
    int count = 0;

    // Traverse all vertices
    for (int i = 0; i < n; i++) {

        // Insert all topmost vertices obtained
        s[i] = root(parent[i]);
        int j;
        for(j=0;j<i;j++){
            if(s[j]==s[i]){
                break;
            }
        }
        if(j==i){
            count++;
        }
    }

    // Print count of connected components
    printf("NUmber of conneced nodes: %d\n",count);
}

// Function to print answer
void printAnswer(int N,
                int edges[][2])
{

    // Setting parent to itself
    for (int i = 0; i <= N; i++) {
        parent[i] = i;
    }

    // Traverse all edges
    for (int i = 0; i < 5; i++) {
        connect(edges[i][0], edges[i][1]);
    }

    // Print answer
    connectedComponents(N);
}

// Driver Code
int main()
```

```
{
   // Given N
   int N , e;
   printf("Enter number of nodes: ");
   scanf("%d", &N);
   printf("Enter number of edges: ");
   scanf("%d", &e);
   int edges[e][2];
   // Given edges
   for(int i=0; i<e; i++){
      printf("Enter edges: ");
      scanf("%d%d", &edges[i][0], &edges[i][1]);
   }

   // Function call
   printAnswer(N, edges);

   return 0;
}
```

Output:



```
Enter number of nodes: 5
Enter number of edges: 6
Enter edges: 2 3
Enter edges: 2 4
Enter edges: 1 3
Enter edges: 1 4
Enter edges: 1 3
Enter edges: 1 2
NUmber of conneced nodes: 2
```

Fig 1.1: Output on console for case 1.