KARNATAK LAW SOCIETY’S

GOGTE INSTITUTE OF TECHNOLOGY

UDYAMBAG, BELAGAVI-590008

(An Autonomous Institution under Visvesvaraya Technological University, Belagavi)

**(APPROVED BY AICTE, NEW DELHI)**

Department of Electronics and Communication Engineering



*Course Project Report on*

**Rat in a Maze | Backtracking using stack**

*Submitted in partial fulfillment of the requirement for the award of the degree of*

**Bachelor of Engineering**

**in**

**Data Structures Lab**

*Submitted by*

SHRILAKSHMI DESAI

**Guide**

Prof. Pratijnya Ajawan

**2019 – 2020**

**DECLARATION BY THE STUDENT**

I ***Shrilakshmi Desai,*** hereby declare that the course project report entitled **Rat in a Maze | Backtracking using Stack (Project No.5)**submitted by me to KLS Gogte Institute of Technology, Belagavi, in partial fulfillment of the Degree of **Bachelor of Engineering** in **Electronics and Communication Engineering** is a record of the project carried out in **Data Structures Lab**. This report is for the academic purpose.

I further declare that the course project report has not been submitted and will not be submitted, either in part or full, to any other institution and University for the award of any diploma or degree.

Name of the students: Shrilakshmi Desai

USN : 2GI17EC131

Signature :

Place: K L S GOGTE INSTITUTE OF TECHNOLOGY, BELAGAVI

Date:

**Rat in a Maze | Backtracking using stack**

**Introduction:**

In computer science, a data structure is a data organization, management, and storage format that enables efficient access and modification. More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data.

**Abstract:**

A Maze is given as N\*M binary matrix of blocks and there is a rat initially at (0, 0) ie. maze[0][0] and the rat wants to eat food which is present at some given block in the maze (fx, fy). In a maze matrix, 0 means that the block is a dead end and 1 means that the block can be used in the path from source to destination. The rat can move in any direction (not diagonally) to any block provided the block is not a dead end.

The task is to check if there exists any path so that the rat can reach the food or not. It is not needed to print the path.

**Data structures used in the project:**

In our project we have made use of linked list, stack.

In linked list insertion and deletion operation are easier. Efficient memory utilization i-e no need to pre allocate memory. Faster access time, can be expanded in constant time without memory overhead. Linear datastructures such as stack,queue can be easily implemented using Linked list.

**CODE:**

#include <cstring>

#include <iostream>

#include <stack>

using namespace std;

#define N 4

#define M 5

class node {

public:

    int x, y;

    int dir;

    node(int i, int j)

    {

        x = i;

        y = j;

        // Initially direction

        // set to 0

        dir = 0;

    }

};

// maze of n\*m matrix

int n = N, m = M;

// Coordinates of food

int fx, fy;

bool visited[N][M];

bool isReachable(int maze[N][M])

{

    // Initially starting at (0, 0).

    int i = 0, j = 0;

    stack<node> s;

    node temp(i, j);

    s.push(temp);

    while (!s.empty()) {

        // Pop the top node and move to the

        // left, right, top, down or retract

        // back according the value of node's

        // dir variable.

        temp = s.top();

        int d = temp.dir;

        i = temp.x, j = temp.y;

        // Increment the direction and

        // push the node in the stack again.

        temp.dir++;

        s.pop();

        s.push(temp);

        // If we reach the Food coordinates

        // return true

        if (i == fx and j == fy) {

            return true;

        }

        // Checking the Up direction.

        if (d == 0) {

            if (i - 1 >= 0 and maze[i - 1][j] and

                                    visited[i - 1][j]) {

                node temp1(i - 1, j);

                visited[i - 1][j] = false;

                s.push(temp1);

            }

        }

        // Checking the left direction

        else if (d == 1) {

            if (j - 1 >= 0 and maze[i][j - 1] and

                                    visited[i][j - 1]) {

                node temp1(i, j - 1);

                visited[i][j - 1] = false;

                s.push(temp1);

            }

        }

        // Checking the down direction

        else if (d == 2) {

            if (i + 1 < n and maze[i + 1][j] and

                                    visited[i + 1][j]) {

                node temp1(i + 1, j);

                visited[i + 1][j] = false;

                s.push(temp1);

            }

        }

        // Checking the right direction

        else if (d == 3) {

            if (j + 1 < m and maze[i][j + 1] and

                                    visited[i][j + 1]) {

                node temp1(i, j + 1);

                visited[i][j + 1] = false;

                s.push(temp1);

            }

        }

        // If none of the direction can take

        // the rat to the Food, retract back

        // to the path where the rat came from.

        else {

            visited[temp.x][temp.y] = true;

            s.pop();

        }

    }

    // If the stack is empty and

    // no path is found return false.

    return false;

}

// Driver code

int main()

{

    // Initially setting the visited

    // array to true (unvisited)

    memset(visited, true, sizeof(visited));

    // Maze matrix

    int maze[N][M] = {

        { 1, 0, 1, 1, 0 },

        { 1, 1, 1, 0, 1 },

        { 0, 1, 0, 1, 1 },

        { 1, 1, 1, 1, 1 }

    };

    // Food coordinates

    fx = 2;

    fy = 3;

    if (isReachable(maze)) {

        cout << "Path Found!" << '\n';

    }

    else

        cout << "No Path Found!" << '\n';

    return 0;

}

**Applications:**

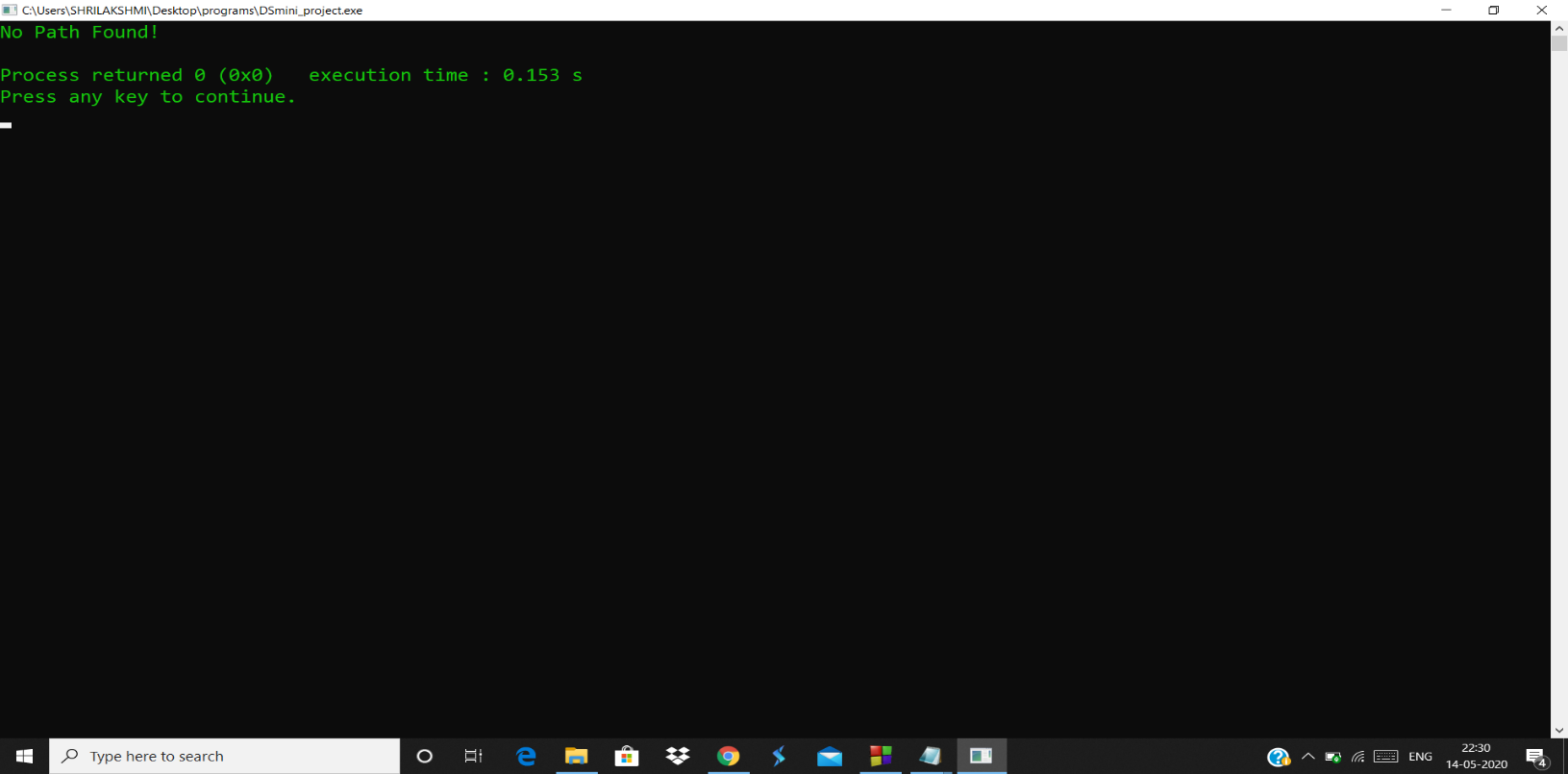
Applications of linked list :

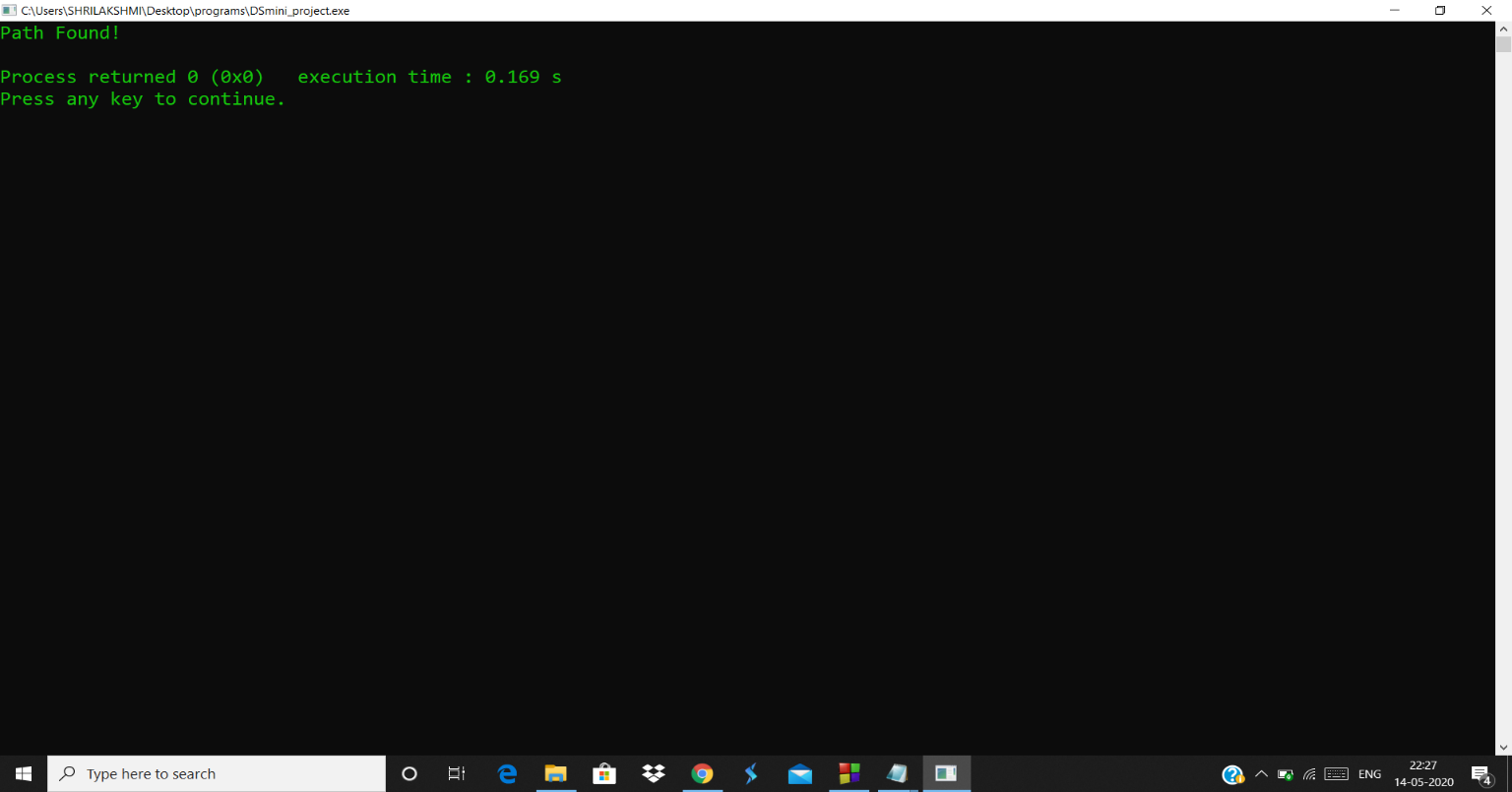
* Implementation of stacks and queues
* Implementation of graphs : Adjacency list representation of graphs is most popular which is uses linked list to store adjacent vertices.
* Dynamic memory allocation : We use linked list of free blocks.
* Performing arithmetic operations on long integers

Applications of linked list in real world-

* Image viewer – Previous and next images are linked, hence can be accessed by next and previous button.
* Previous and next page in web browser – We can access previous and next url searched in web browser by pressing back and next button since, they are linked as linked list.
* Music Player – Songs in music player are linked to previous and next song. you can play songs either from starting or ending of the list

**OUTPUT:**

****

****

**Analysis(limitations):**

* They use more memory than arrays because of the storage used by their pointers.
* Difficulties arise in linked lists when it comes to reverse traversing. For instance, singly linked lists are cumbersome to navigate backwards and while doubly linked lists are somewhat easier to read, memory is wasted in allocating space for a back-pointer.
* **Stack** memory is very limited.
* Creating too many objects on the **stack** can increase the risk of **stack** overflow.
* Random access is not possible.