**CAMPUS NAVIGATION SYSTEM**

by

P KHYATHI REDDY 19BEC1118

DUVVADA MEERA 19BEC1264

P PENCHALA MOHAN 19BEC1047

SANTHOSH SUNKARA 19BEC1279

A project report submitted to

**Dr. VETRIVELAN. P**

**SCHOOL OF ELECTRONICS ENGINEERING**

in partial fulfilment of the requirements for the course of

**CSE2003 – DATA STRUCTURES AND ALGORITHMS**

in

**B.Tech. ELECTRONICS AND COMMUNICATION ENGINEERING**

****

**Vandalur – Kelambakkam Road**

**Chennai – 600127**

**APRIL 2022**

**BONAFIDE CERTIFICATE**

Certified that this project report entitled “**CAMPUS NAVIGATION SYSTEM”** is a bonafide work of **P KHYATI REDDY** – 19BEC1118, **DUVVADA MEERA** – 19BEC1264, **P PENCHALA MOHAN** – 19BEC1047, **SANTHOSH SUNKARA** – 19BEC1279 who carried out the Project work under my supervision and guidance for CSE 2003 – **DATA STRUCTURES AND ALGORITHMS.**

**Dr. VETRIVELAN.P**

Associate Professor Senior

School of Electronics Engineering (SENSE),

VIT University, Chennai

Chennai – 600 127.

**ABSTRACT**

There exist many advanced navigation systems but most of them are unable to provide routes precisely as well as information of building within a region such as campus, shopping mall, hospital and etc. Nowadays, as people are getting more and more connected to technology, they lost their human touch. Also, people feel more convenient to search for the problem themselves rather than asking someone for help.

An informative, reliable and precise guidance system is very important in this technological era. It should be able to navigate the user no matter the user is under the indoor or outdoor environment. The guidance system must be user-friendly and able to process data efficiently.

Since the size of any college/university campus can vary from 30 acres to anywhere around 200 acres, students spend majority of their time in travelling between different buildings. New students feel inconvenient to search their way inside the campus. Therefore, a navigation system is required to find the optimal path within the campus from one point to another point.

**ACKNOWLEDGEMENT**

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. Vetrivelan. P,** Associate Professor Senior, School of Electronics Engineering, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to **Dr.Sivasubramanian. A,**Dean of School of Electronics Engineering, VIT Chennai, for extending the facilities of the School towards our project and for her unstinting support.

We express our thanks to our Head of the Department **Dr.Vetrivelan. P**forhis support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

**P KHYATI REDDY DUVVADA MEERA P PENCHALA MOHAN SANTHOSH SUNKARA**

**TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **SL. NO.** |  | **TITLE** | **PAGE NO.** |
|  |  | **ABSTRACT** | 3 |
|  |  | **ACKNOWLEDGEMENT** | 4 |
|  |  |  |  |
| **1** |  | **INTRODUCTION** | 6 |
|  |  |  |  |
|  | 1.1 | OBJECTIVES | 6 |
|  | 1.2 | BENEFITS | 6 |
|  | 1.3  1.4 | FEATURES  METHODOLOGY | 6  6 |
|  |  |  |  |
| **2** |  | **CAMPUS NAVIGATION**  **SYSTEM - DESIGN** | 7-18 |
|  | 2.1 | FLOW CHART OF SYSTEM DESIGN | 7-9 |
|  | 2.2 | SOFTWARE SPECIFICATIONS | 9-16 |
|  |  |  |  |
| **3** | 3.1  3.2 | **CAMPUS NAVIGATION**  **SYSTEM - IMPLEMENTATION AND ANALYSIS**  SYSTEM IMPLEMENTATION  RESULTS AND INFERENCES | 19-22  19  22 |
|  |  |  |  |
| **4** |  | **CONCLUSION AND FUTURE WORK** | 23 |
|  | 4.1 | CONCLUSION | 23 |
|  | 4.2 | FUTURE WORK | 23 |

APPENDIX

**5** REFERENCES 26

BIO-DATA 27

**CHAPTER 1**

**INTRODUCTION**

**1.1 OBJECTIVES**

* Design the CAMPUS NAVIGATION SYSTEM
* Detect the shortest path from source node to destination node accordingly.
* Detect the optimized shortest path using Dijkstra’s algorithm for single source shortest path and show distance.
* Detect the optimized shortest path using Floyd-Warshall’s algorithm in case of negative weights detection or to find the shortest path between all paths and show distance.

**1.2 BENEFITS**

* It helps people to reach their destinations.
* It helps to track parcels, stolen vehicles, missing persons, properties etc.
* Accessible Wayfinding.
* Safety & Security.
* Emergency Navigation.
* On-Campus Digital Directories.
* On-Campus Services and Amenities.
* Optimized Routing.

**1.3 FEATURES**

* Shows the shortest distance between any two points using a famous algorithm called dijkstra’s algorithm.
* If there are any negative weights the dijkstras algorithm cannot give the shortest possible distance.
* So we implemented another algorithm called Floyd-warshall algorithm i.e all pairs shortest problem also it takes negative weights.
* Finally the shortest distance will be printed using the algorithms.
* These are the features of the project.

**1.4 METHODOLOGY**

* First, we use Dijkstra’s algorithm to generate a shortest path tree with given source as root.
* We maintain two sets, one set contains vertices included in shortest path tree, and other set includes vertices not yet included in shortest path tree. At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has a minimum distance from the source.
* And finally it suggests the shortest optimal path, but when we have negative weights, we have to go with the Floyd-Warshall algorithm**.**
* Also, if we want to know whether the graph contains negative cycles or not, the Floyd-Warshall algorithm can help us with that.
* In conclusion, Dijkstra's algorithm never ends if the graph contains at least one negative cycle. By a negative cycle, we mean a cycle that has a negative total weight for its edges.
* So finally if there are any negative edges Floyd-Warshall algorithm comes to the implementation and gives the shortest possible distance between two points.
* In conclusion we will compare the time complexity of both algorithms and gives a best navigation to the user.

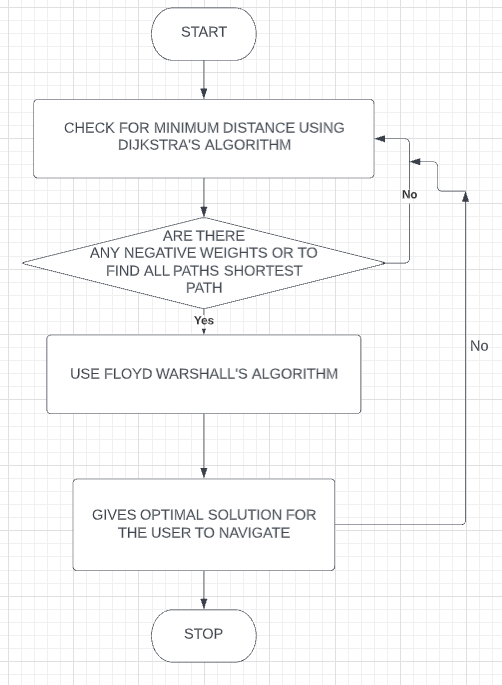
**CHAPTER 2**

**CAMPUS NAVIGATION SYSTEM - DESIGN**

**2.1 FLOW DIAGRAM**

The main features of the below flow diagram (given below) are

* Dijkstra’s Algorithm
* Floyd Warshall's algorithm



**Figure 1: CAMPUS NAVIGATION SYSTEM**

Figure 1 shows the flow diagram of campus navigation system using dijkstra’s and Floyd-Warshall algorithm.

**2.2 SOFTWARE SPECIFICATIONS**

* + C++ Programming
  + Designing 2D campus map using graphics library using C++

**CHAPTER 3**

**SOFTWARE IMPLEMENTATION AND ANALYSIS**

This section describes system implementation and results with inferences.

**3.1 SYSTEM IMPLEMENTATION**

The implementation contains two algorithms:

1. Dijkstra’s Algorithm
2. Floyd-Warshall Algorithm

**DIJKSTRA’S ALGORITHM PSEUDO CODE**

Dijkstra\_Algorithm(source, G):

"""

parameters: source node--> source, graph--> G

return: List of cost from source to all other nodes--> cost

"""

unvisited\_list = [] // List of unvisited verticesvertices

cost = []

cost[source] = 0 // Distance (cost) from source to source will be 0

for each vertex v in G: // Assign cost as INFINITY to all vertices

if v ≠ source

cost[v] = INFINITY

add v to unvisited\_list // All nodes pushed to unvisited\_list initially

while unvisited\_list is not empty: // Main loop

v = vertex in unvisited\_list with min cost[v] // v is the source node for first iteration

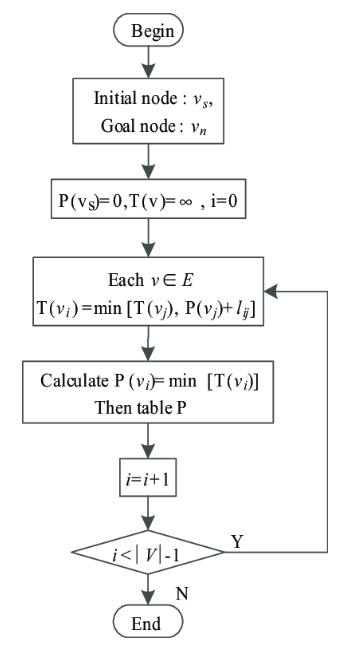
remove v from unvisited\_list // Marking node as visited

for each neighbor u of v: // Assign shorter path cost to neigbour u

cost\_value = Min( cost[u], cost[v] + edge\_cost(v, u)]

cost[u] = cost\_value // Update cost of vertex u

return cost

****

**Dijkstra’s algorithm flow diagram**

**FLOYD-WARSHALL ALGORITHM PSEUDO CODE**

Floyd-Warshall(W)

n = W.rows

D^(0) = W

for k = 1 to n

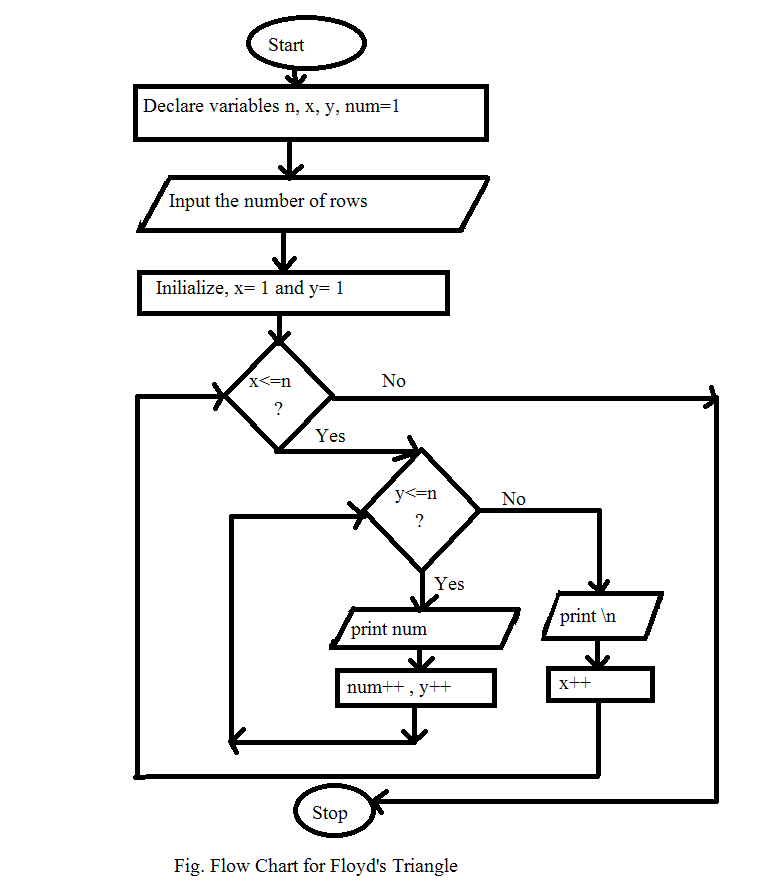
let D(k) = (d^(k)ij ) be a new n × n matrix

for i = 1 to n

for j = 1 to n

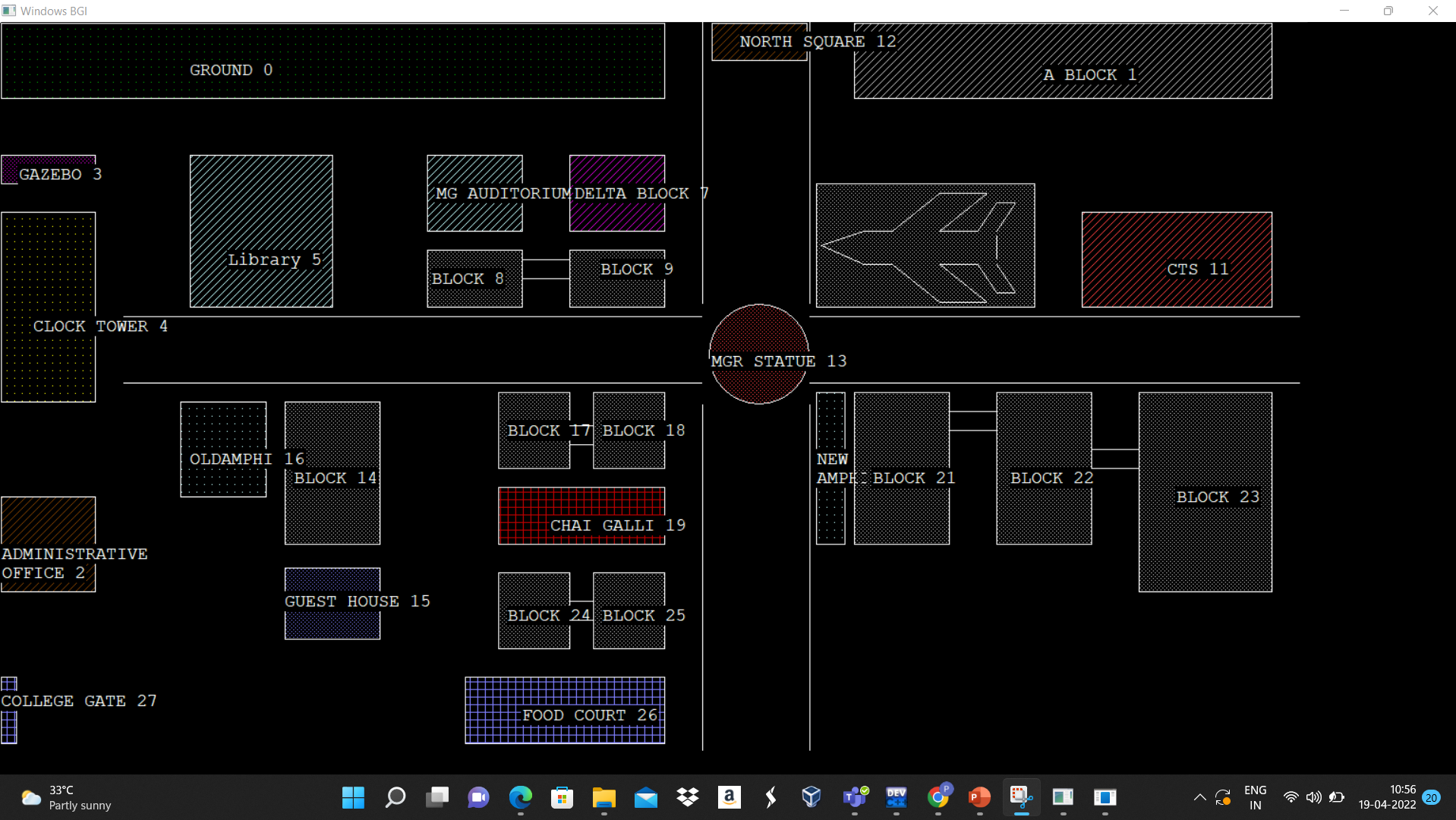
d^(k)ij = min(d^(k−1)ij ,d^(k−1)ik +d^(k−1)kj )

return D^(n)

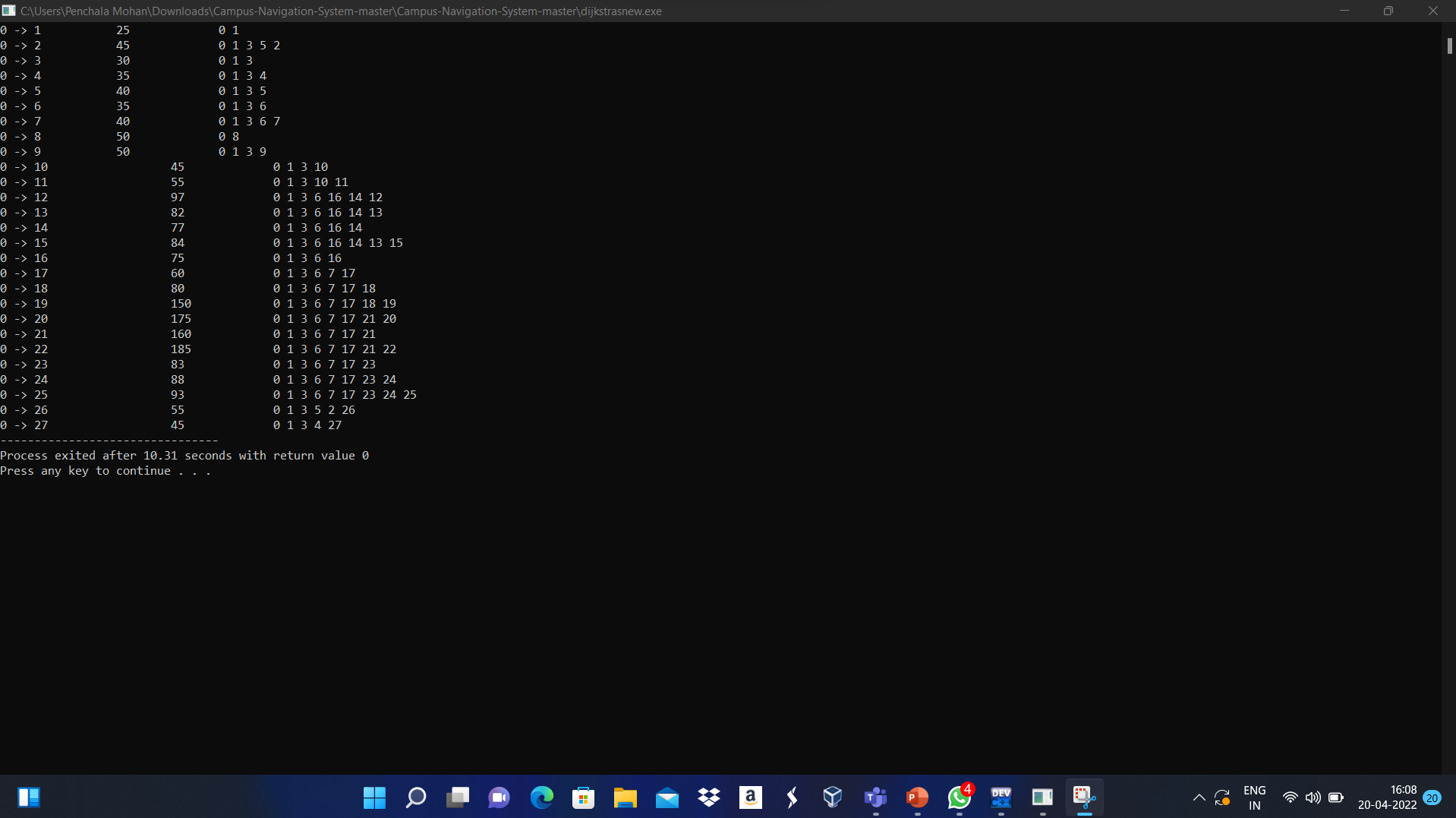


**3.1.1 IMPLEMENTATION OUTPUTS**

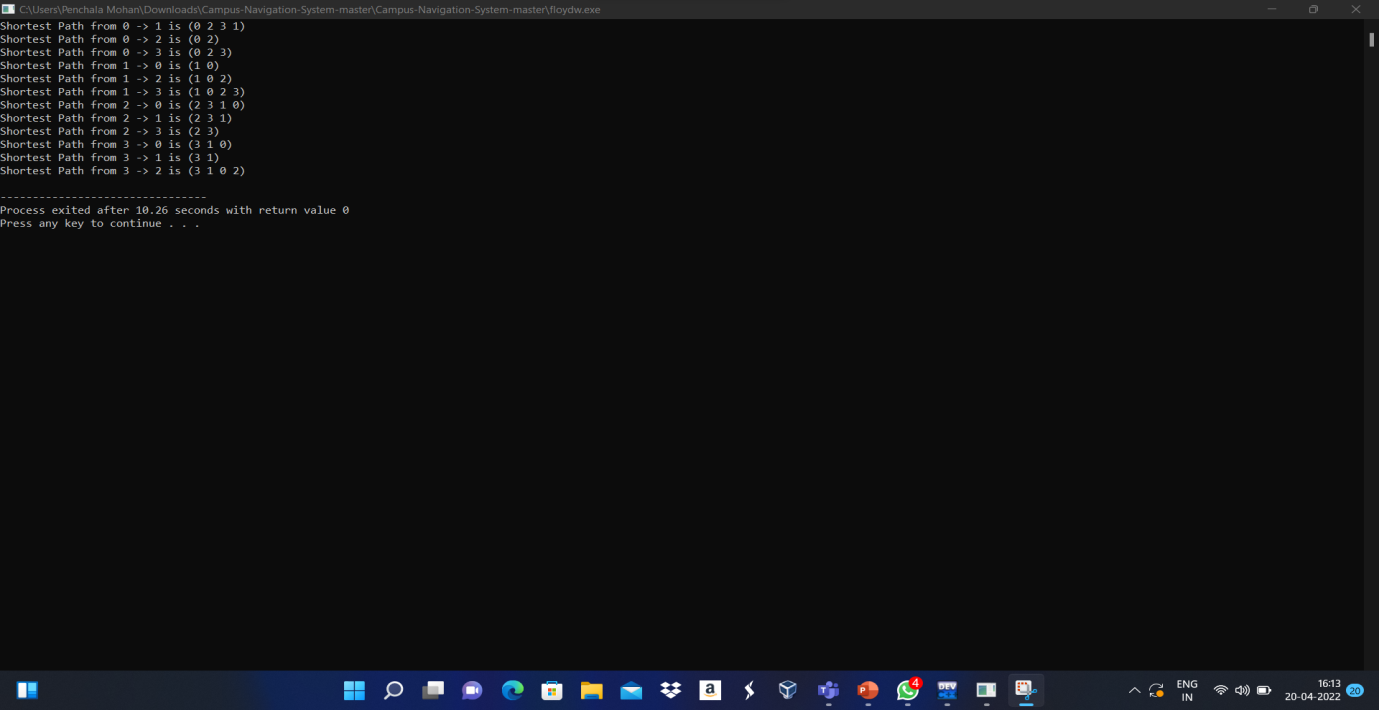
2D-Campus map using graphics library in C++.

****

Shortest distance using Dijkstra’s algorithm.



Shortest distance using Floyd-Warshall Algorithm.



**3.1.2 COMPARISON OF DIJKSTRA’S AND FLOYD-WARSHALL ALGORITHMS**

## Dijkstra’s Algorithm

**It solves the Single Source Shortest Path (SSSP) problem.**

## Floyd-Warshall Algorithm

**It solves the All-Pairs Shortest Paths (APSP) problem.**

### **Algorithmic Paradigm**

**Dijkstra’s algorithm follows the greedy paradigm.** In other words, it makes locally optimal choices at each step leading to a globally optimal solution on termination. The defining characteristics of a problem which a greedy algorithm can solve are optimal substructure and the greedy choice property. Such algorithms execute in a top-down fashion.

By optimal substructure, we mean that an optimal solution to a problem consists of optimal solutions to its subproblems. For example, the SSSP problem possesses optimal substructure but the [Unweighted Longest Simple Path](https://en.wikipedia.org/wiki/Longest_path_problem) problem does not.

The greedy choice property states that if the algorithm makes a greedy choice at the first step, then there exists an optimal solution that is compatible with it. In particular, making a greedy choice restricts the subproblems that we have to solve.

In contrast, Floyd-Warshall’s algorithm follows the dynamic programming (DP) paradigm. Such algorithms either execute top-down with memoization applied or construct solutions bottom-up. Problems solvable by DP possess optimal substructure and overlapping subproblems.

A problem is said to have overlapping subproblems if the same subproblems are solved at multiple steps of the algorithm. Thus, its subproblem space is small enough that no new subproblems are generated.

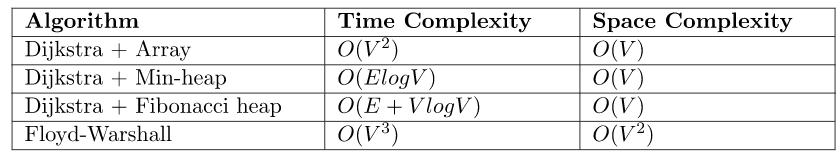
### Asymptotic Analysis

All in all, there are three ways to implement Dijkstra’s algorithm.

* **arrays:** insert and decrease key operations cost while min-extraction costs, leading to overall.
* **min-heaps:** decrease key and min-extraction operations cost and are done and times respectively.
* **Fibonacci heaps:** decrease key costs and min-extraction costs. This gives overall.

Additionally, its space complexity is , assuming adjacency list-based implementation.

In contrast, Floyd-Warshall operates on the adjacency matrix, which is implemented as a 2D array. Its time and space complexity is and respectively:



**3.2 RESULTS AND INFERENCES**

We have implemented all our work and have attached the results in the implementation section and compared both the algorithms analysed it’s time complexity and space complexity.

We can infer that Floyd Warshall works for negative edge but no negative cycle, whereas Dijkstra’s algorithm don’t work for negative edges and also both the algorithms are famous in navigation systems has many advantages when compared with other algorithms.

**CODE ANALYSIS**

* **Number of Lines: 254**
* **Number of Functions/Routines: 15**

**CHAPTER 4**

**CONCLUSION AND FUTURE WORK**

**4.1 CONCLUSION**

* The CAMPUS NAVIGATION SYSTEM designed and implemented successfully.
* The system is targeted at colleges (which is covered with land area) to help the students/parents/teachers to find the shortest path from their source to their destination at colleges.
* The prototype developed can also be used in shopping malls.
* The preliminary test results are promising.

**4.2 FUTURE WORK**

* We have implemented both Dijkstra’s algorithm and Floyd-Warshall algorithms for the shortest path between two points and it worked well.
* In future we try to explore some more algorithms which are working effectively in navigations and we implement that by using some advanced software.
* Also, there were some disadvantages for the existing navigation systems like it cannot give the optimal solution due to complexities in the algorithms.
* We try to find that and solve the problems in the navigation systems.
* This is the idea of our future work.

**(OPTIONAL)**

**APPENDIX**

**SOURCE CODES**

#include <bits/stdc++.h>

using namespace std;

// Number of vertices in the graph

#define V 28

// A utility function to find the vertex with minimum

// distance value, from the set of vertices not yet included

// in shortest path tree

int minDistance(int dist[], bool sptSet[])

{

// Initialize min value

int min = INT\_MAX, min\_index;

for (int i = 0; i < V; i++)

if (sptSet[i] == false && dist[i] <= min)

min = dist[i], min\_index = i;

return min\_index;

}

// Function to print shortest path from source to j using

// parent array

void printPath(int parent[], int j)

{

// Base Case : If j is source

if (parent[j] == -1)

return;

printPath(parent, parent[j]);

cout << j << " ";

}

// A utility function to print the constructed distance

// array

int printSolution(int dist[], int n, int parent[])

{

int src = 0;

cout << "Vertex\t Distance\tPath";

for (int i = 1; i < V; i++) {

printf("\n%d -> %d \t\t %d\t\t%d ", src, i, dist[i],

src);

printPath(parent, i);

}

}

// Function that implements Dijkstra's single source

// shortest path algorithm for a graph represented using

// adjacency matrix representation

void dijkstra(int graph[V][V], int src)

{

// The output array. dist[i] will hold the shortest

// distance from src to i

int dist[V];

// sptSet[i] will true if vertex i is included / in

// shortest path tree or shortest distance from src to i

// is finalized

bool sptSet[V] = { false };

// Parent array to store shortest path tree

int parent[V] = { -1 };

// Initialize all distances as INFINITE

for (int i = 0; i < V; i++)

dist[i] = INT\_MAX;

// Distance of source vertex from itself is always 0

dist[src] = 0;

// Find shortest path for all vertices

for (int count = 0; count < V - 1; count++) {

// Pick the minimum distance vertex from the set of

// vertices not yet processed. u is always equal to

// src in first iteration.

int u = minDistance(dist, sptSet);

// Mark the picked vertex as processed

sptSet[u] = true;

// Update dist value of the adjacent vertices of the

// picked vertex.

for (int v = 0; v < V; v++)

// Update dist[v] only if is not in sptSet,

// there is an edge from u to v, and total

// weight of path from src to v through u is

// smaller than current value of dist[v]

if (!sptSet[v] && graph[u][v]

&& dist[u] + graph[u][v] < dist[v]) {

parent[v] = u;

dist[v] = dist[u] + graph[u][v];

}

}

// print the constructed distance array

printSolution(dist, V, parent);

}

// Driver Code

int main()

{

// Let us create the example graph discussed above

int graph[V][V] = { {00,25,00,00,00,00,00,00,50,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00},

{25,00,30,05,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00},

{00,30,00,10,10,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,10,00},

{05,00,00,00,05,10,05,00,00,20,15,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00},

{00,10,00,05,00,05,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,10},

{00,00,05,10,00,00,05,05,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,05},

{00,00,00,05,00,05,00,05,00,00,00,00,00,00,00,00,40,00,00,00,00,00,00,00,00,00,00,00},

{00,00,00,00,00,05,05,00,00,00,23,00,00,00,00,00,00,20,00,00,00,00,00,00,00,00,00,00},

{05,00,00,00,05,10,05,00,00,20,15,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00},

{50,00,00,20,00,00,00,00,20,00,30,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00},

{00,00,00,15,00,00,00,00,15,30,00,10,70,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00},

{00,00,00,00,00,00,00,00,00,00,10,00,00,35,00,00,00,00,00,00,00,00,00,00,00,00,00,00},

{00,00,00,50,00,00,00,00,50,00,70,35,00,15,20,00,00,00,00,00,00,00,00,00,00,00,00,00},

{00,00,00,00,00,00,00,00,00,00,00,00,15,00,05,02,00,00,00,00,00,00,00,00,00,00,00,00},

{00,00,00,00,00,00,40,00,00,00,00,00,20,05,00,00,02,00,00,00,00,00,00,00,00,00,00,00},

{00,00,00,00,00,00,00,00,00,00,00,00,00,02,00,00,15,80,00,00,00,00,00,00,00,00,00,00},

{00,00,00,00,00,00,00,00,00,00,00,00,00,00,02,15,00,20,00,00,00,00,00,00,00,00,00,00},

{00,00,00,00,00,00,00,20,00,00,00,00,00,00,00,80,20,00,20,00,00,100,00,23,00,00,00,00},

{00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,20,00,70,00,00,00,00,00,00,00,00},

{00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,70,00,00,00,00,00,00,00,00,00},

{00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,15,00,00,00,00,00,00},

{00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,15,00,25,00,00,00,00,00},

{00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,25,00,00,00,00,00,00},

{00,00,00,00,00,00,00,10,00,00,00,00,00,00,00,00,23,00,00,00,00,00,00,00,05,00,00,00},

{00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,05,00,05,00,00},

{00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,05,00,00,00},

{00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00},

{00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00},

};

dijkstra(graph, 0);

return 0;

}

#include <iostream>

#include <vector>

#include <climits>

#include <iomanip>

using namespace std;

// Recursive function to print path of given vertex `u` from source vertex `v`

void printPath(vector<vector<int> > const &path, int v, int u)

{

if (path[v][u] == v) {

return;

}

printPath(path, v, path[v][u]);

cout << path[v][u] << ", ";

}

// Function to print the shortest cost with path information between

// all pairs of vertices

void printSolution(vector<vector<int> > const &cost, vector<vector<int> > const &path)

{

int n = cost.size();

for (int v = 0; v < n; v++)

{

for (int u = 0; u < n; u++)

{

if (u != v && path[v][u] != -1)

{

cout << "The shortest path from " << v << " —> " << u << " is ["

<< v << ", ";

printPath(path, v, u);

cout << u << "]" << endl;

}

}

}

}

// Function to run the Floyd–Warshall algorithm

void floydWarshall(vector<vector<int> > const &adjMatrix)

{

// total number of vertices in the `adjMatrix`

int n = adjMatrix.size();

// base case

if (n == 0) {

return;

}

// cost[] and path[] stores shortest path

// (shortest cost/shortest route) information

vector<vector<int> > cost(n, vector<int>(n));

vector<vector<int> > path(n, vector<int>(n));

// initialize cost[] and path[]

for (int v = 0; v < n; v++)

{

for (int u = 0; u < n; u++)

{

// initially, cost would be the same as the weight of the edge

cost[v][u] = adjMatrix[v][u];

if (v == u) {

path[v][u] = 0;

}

else if (cost[v][u] != INT\_MAX) {

path[v][u] = v;

}

else {

path[v][u] = -1;

}

}

}

// run Floyd–Warshall

for (int k = 0; k < n; k++)

{

for (int v = 0; v < n; v++)

{

for (int u = 0; u < n; u++)

{

// If vertex `k` is on the shortest path from `v` to `u`,

// then update the value of cost[v][u] and path[v][u]

if (cost[v][k] != INT\_MAX && cost[k][u] != INT\_MAX

&& cost[v][k] + cost[k][u] < cost[v][u])

{

cost[v][u] = cost[v][k] + cost[k][u];

path[v][u] = path[k][u];

}

}

// if diagonal elements become negative, the

// graph contains a negative-weight cycle

if (cost[v][v] < 0)

{

cout << "Negative-weight cycle found!!";

return;

}

}

}

// Print the shortest path between all pairs of vertices

printSolution(cost, path);

}

int main()

{

// define infinity

int I = INT\_MAX;

// given adjacency representation of the matrix

vector<vector<int> > adjMatrix =

(

{ 0, I, -2, I },

{ 4, 0, 3, I },

{ I, I, 0, 2 },

{ I, -1, I, 0 }

);

// Run Floyd–Warshall algorithm

floydWarshall(adjMatrix);

return 0;

}

**CHAPTER 5**

**REFERENCES**

* <https://en.wikipedia.org/wiki/Navigation_system>
* <https://en.wikipedia.org/wiki/Shortest_path_problem#:~:text=The%20most%20important%20algorithms%20for,edge%20weights%20may%20be%20negative>.
* <https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm>
* <https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm>
* <https://www.geeksforgeeks.org/time-complexity-and-space-complexity/>
* <https://link.springer.com/chapter/10.1007/978-3-642-41674-3_113>
* <https://ieeexplore.ieee.org/document/8394464>

|  |  |
| --- | --- |
|  | Name : P PENCHALA MOHAN  Mobile Number : 6281443848  E-mail : [penchala.mohan2019@vitstudent.ac.in](file:///C:\Users\P%20K%20Reddy\Downloads\penchala.mohan2019@vitstudent.ac.in)  Permanent Address: Nellore, Andhra Pradesh |
| **WhatsApp Image 2021-12-03 at 20.47.13.jpeg** | Name : PATUR KHYATHI REDDY  Mobile Number : 8688153518  E-mail : [paturkhyathi.reddy2019@vitstudent.ac.in](mailto:paturkhyathi.reddy2019@vitstudent.ac.in)  Permanent Address: Chittoor, Andhra Pradesh |
| **WhatsApp Image 2021-12-03 at 20.57.56.jpeg** | Name : MEERA DUVVADA  Mobile Number : 9963088916  E-mail : [meera.duvvada2019@vitstudent.ac.in](mailto:meera.duvvada2019@vitstudent.ac.in)  Permanent Address: Hyderabad, Telangana |
| **WhatsApp Image 2021-12-04 at 5.54.39 PM.jpeg** | Name : SANTHOSH SUNKARA  Mobile Number : 8019999269  E-mail : [hemanth.santhosh2019@vitstudent.ac.in](file:///C:\Users\P%20K%20Reddy\Downloads\hemanth.santhosh2019@vitstudent.ac.in)  Permanent Address: Rajahmundry, Andhra Pradesh |