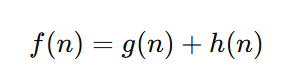
# **INTRODUCTION:**

# **OUTLINE:**

Here, we have decided to create a very interesting project which highlights the main concept of DSA (Data Structure and algorithm). Our project is about pathway tracking of hospital areas map. In order to track shortest path like if you see a real-life example when we have two path and we have to reach on time, so we will take a shortcut as shortest path. Displacement concept is used here. We will use weight as to calculate distance. Here, we will use graph data structure for linking connection between paths. We have collected csv file which includes stakeholder name and their specific room. There are total 600 rooms and 31 stakeholders, who are involved in it. We also used dynamic memory for memory allocation, manager uses this for allocation the memory. Functions for allocation, we made user-defined function which includes update, delete and linking connection. Each stakeholder would have 5 rooms {A, B, C, D, E}. These are considered as nodes and edges are connected with each of them for result. We used Matrices for tracking area path. This is a mini-project; it includes the concept of tracking path.

# **TRACKING PATH ALGORITHMS FORMULA:**

1. **A\* Algorithm.**

****

# **THE IMPLEMENTATION:**

# **ALGORITHM:**

**DESCRIPTION: Loc (N, Pathway, Distance)**

[Here, is the Pathway array is given with N size and Distance are the weights to reach to the nearest destination. This algorithm is to find Pathway of hospital area using graph, there are total 600 rooms and 31 stakeholders and 155 rooms are used and remaining are under construction. Each stakeholder holds 5 rooms (A, B,C,D,E).These all nodes are link with each other from the Matrix, where 1->forward,-1->backward and ->0 for null not link with this nodes.We have used g and h here in order to calculate shortest distance g is the cost start node and h is heuristic estimate and visited and not visited are the position for track whole graph so in order to find a shortest path]

Step 1: [INITIALIZE] Set area: =POS [AREA IS STRING]

[HERE K REPRESENT SIZE OF LOGIC MATRIX]

Step 2: [INITIALIZE LOGIC] Set LOGIC[K][K]: = {

{0, 1, 0, 0, -1},

{0, 0, 1, -1, 0},

{1, 0, 0, 0, 1},

{0, 1, -1, 0, 0},

{-1, 0, 0, 1, 0} };

[INITIALIZE HER

[HERE INITIAL IS CONSIDER AS COUNTER OUTER]

STEP 3: [OUTLOOP REPEAT STEP 4 AND STEP 13] for INITIAL<K, INITIAL++:

[HERE APPROACH IS CONSIDER AS COUNTER INNER]

STEP 4: [INNER LOOP REPEAT STEP 5 TO STEP 11]for APPROACH<K,APPROACH++:

STEP 5: [CONDITION OF AREA TRACK ->1]if LOGIC[K][K]==1:

STEP 6: [SET AREA FOR LINKING] POS:=LOGIC[K][K]

STEP 7: [END OF CONDITION]

STEP 8:[CONDITION OF AREA TRACK ->-1]if LOGIC[K][K]==-1:

STEP 9: [SET AREA FOR LINKING] POS: =LOGIC[K][K]

STEP 10: [END OF CONDITION]

STEP 11: [ELSE IF AREA IS NOT 1 OR -1] [AREA NOT FOUND]

STEP 12: [END OF INNER LOOP]

STEP 13: [END OF OUTERLOOP]

STEP 14: [GRAPH HAS BEEN LINKED TO EACH NODES WITH EDGES A->B->C->D->E]

STEP 16: [NOW TO FIND THE SHORTEST DISTANCE] [HERE A\*ALGORITHM WE USED]

STEP 17: [INITIALIZE] to\_visit, visited, g,f and came\_from

STEP 19:[SET DECLARATION] set g:=start,set h:=estimate value

STEP 20:[SET FORMULA] SET Distance:=g+h

STEP 21:[ITERATION FOR LOOP OCCURS TILL DISTANCE TRACK COMPLETELY]

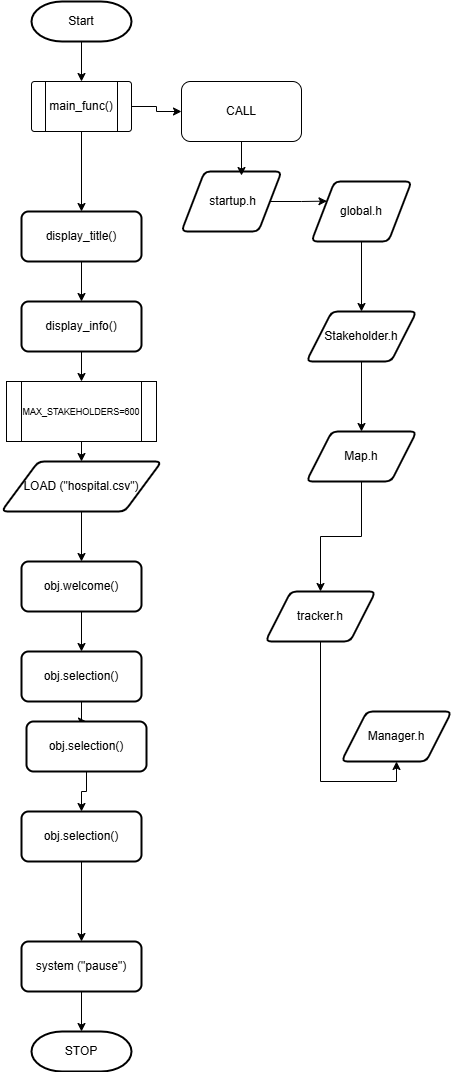
STEP 22: [END ITERATION ]

STEP 23:WRITE:[Shortest Distance]Distance[FOUND SUCCED]

STEP 24:[SUGGESTED DISTANCE OF USER DESIRE]

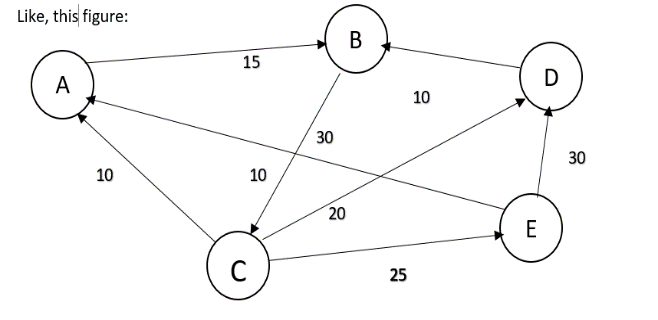
STEP 25: EXIT

# **FLOWCHART:**

****

# **DESCRIPTION OF DATA STRUCTURE USED:**

In this project, we have used a directed graph with specific weights. A directed graph consists of edges that have a defined direction, representing the flow or relationship between vertices. Each edge in the graph is also assigned a weight, which can represent metrics such as cost, distance, or any other relevant value. This structure is particularly useful for solving problems that involve finding the shortest path, optimizing routes, or analyzing connections with specific constraints and values.We used 5-location for each stackholders ,as remaining is in under-construction.we have made 5x5 Matrix, 0 represent that location is not connected and 1 for location connected to other pathway (nodes).

* **1** indicating the vertex is part of the edge (and typically for an outgoing edge),
* **-1** indicating the vertex is part of the edge (and typically for an incoming edge),
* **0** if the vertex is not part of the edge. 

int pathway[5][5] = { {0, 1, 0, 0, -1}, {0, 0, 1, -1, 0 {1, 0, 0, 0, 1}, {0, 1, -1, 0, 0}, {-1, 0, 0, 1, 0}};

int distance[5] = {10, 15, 20, 25, 30};

int heuristic\_value[5] = {20, 40, 60, 70, 80};

//SAMPLE CODE FOR TRACKING

# **SOURCE CODE OF THE PROJECT:**

# **CODE:**

#include <iostream>

#include <string>

#include <climits>

using namespace std;

struct Data {

string name;

string loc;

int initial;

int final;

int weight;

int estimated\_value;

};

class SampleCode {

public:

void All() {

cout << "AREA TRACKER" << endl;

cout << "Loading..." << endl;

string\* people = new string[4];

people[0] = "Manager";

people[1] = "Patient";

people[2] = "Doctor";

people[3] = "Nurse";

cout << "Enter the stakeholder you want from the list (0-3):" << endl;

for (int i = 0; i < 4; i++) {

cout << i << ". " << people[i] << endl;}

cout << "-->: " << endl;

int stakeholder;

cin >> stakeholder;

if (stakeholder >= 0 && stakeholder <= 3) {

cout << "You selected: " << people[stakeholder] << endl;

if (stakeholder == 0) {

manageRooms();

} else {

trackArea();}

} else {

cout << "Invalid selection!" << endl;

}

delete[] people;}

void manageRooms() {

int numRooms = 5;

string\* areas = new string[numRooms];

areas[0] = "Area\_1";

areas[1] = "Area\_2";

areas[2] = "Area\_3";

areas[3] = "Area\_4";

areas[4] = "Area\_5";

int roomChoice, roomToDelete, linkRoom1, linkRoom2;

while (true) {

cout << "\nManager Menu:" << endl;

cout << "1. Create Room" << endl;

cout << "2. Delete Room" << endl;

cout << "3. Link Rooms" << endl;

cout << "4. Exit Manager Mode" << endl;

cout << "Enter your choice (1-4): ";

cin >> roomChoice;

switch (roomChoice) {

case 1: {

string\* tempAreas = new string[numRooms + 1];

for (int i = 0; i < numRooms; i++) {

tempAreas[i] = areas[i]; }

cout << "Enter new room name (e.g., Area\_6): ";

cin >> tempAreas[numRooms];

delete[] areas;

areas = tempAreas;

numRooms++;

cout << "Room " << areas[numRooms - 1] << " created!" << endl;

break;}

case 2: {

cout << "Enter the room number to delete (1-" << numRooms << "): ";

cin >> roomToDelete;

if (roomToDelete >= 1 && roomToDelete <= numRooms) {

cout << "Room " << areas[roomToDelete - 1] << " deleted!" << endl;

for (int i = roomToDelete - 1; i < numRooms - 1; i++) {

areas[i] = areas[i + 1];

}

numRooms--;

} else {

cout << "Invalid room number!" << endl;}

break;}

case 3: {

cout << "Enter two room numbers to link (1-" << numRooms << "): ";

cin >> linkRoom1 >> linkRoom2;

if (linkRoom1 >= 1 && linkRoom1 <= numRooms && linkRoom2 >= 1 && linkRoom2 <= numRooms) {

cout << "Rooms " << areas[linkRoom1 - 1] << " and " << areas[linkRoom2 - 1] << " are now linked!" << endl;

} else {

cout << "Invalid room numbers!" << endl; }

break;}

case 4:

cout << "Exiting Manager Mode..." << endl;

delete[] areas;

return;

default:

cout << "Invalid choice! Please try again." << endl;

break; }}}

void trackArea() {

string\* areas = new string[5];

areas[0] = "Area\_1";

areas[1] = "Area\_2";

areas[2] = "Area\_3";

areas[3] = "Area\_4";

areas[4] = "Area\_5";

int initial, final;

int pathway[5][5] = {

{0, 1, 0, 0, -1},

{0, 0, 1, -1, 0},

{1, 0, 0, 0, 1},

{0, 1, -1, 0, 0},

{-1, 0, 0, 1, 0}};

int distance[5] = {10, 15, 20, 25, 30};

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; j++) {

if (pathway[i][j] == 1) {

cout << areas[i] << " --> " << areas[j] << endl; }

else if (pathway[i][j] == -1) {

cout << areas[i] << " <--- " << areas[j] << endl; } } }

cout << "Enter initial location (1-5): ";

cin >> initial;

cout << "Enter final location (1-5): ";

cin >> final;

if (initial >= 1 && initial <= 5 && final >= 1 && final <= 5) {

cout << "Tracking from " << areas[initial - 1] << " to " << areas[final - 1] << endl;

int actual\_distance = calculateActualDistance(initial - 1, final - 1, pathway, distance);

cout << "Actual Distance: " << actual\_distance << " units." << endl;

int suggested\_start = -1, suggested\_end = -1;

int min\_distance = INT\_MAX;

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; j++) {

if (i != j) {

int temp\_distance = calculateActualDistance(i, j, pathway, distance);

if (temp\_distance < min\_distance) {

min\_distance = temp\_distance;

suggested\_start = i;

suggested\_end = j; }} }}

cout << "Suggested alternative path: " << areas[suggested\_start] << " to " << areas[suggested\_end] << " with a shorter distance of " << min\_distance << " units." << endl;

logic(initial - 1, final - 1, areas);

} else {

cout << "Invalid locations!" << endl;

}

delete[] areas;

}

void logic(int start, int end, string\* areas) {

int pathway[5][5] = {

{0, 1, 0, 0, -1},

{0, 0, 1, -1, 0},

{1, 0, 0, 0, 1},

{0, 1, -1, 0, 0},

{-1, 0, 0, 1, 0}

};

int distance[5] = {10, 15, 20, 25, 30};

int heuristic\_value[5] = {20, 40, 60, 70, 80};

int\* to\_visit = new int[5];

int\* visited = new int[5];

int\* g = new int[5];

int\* f = new int[5];

int\* came\_from = new int[5];

for (int i = 0; i < 5; i++) {

to\_visit[i] = -1;

visited[i] = -1;

g[i] = INT\_MAX;

f[i] = INT\_MAX;

came\_from[i] = -1;

}

g[start] = 0;

f[start] = heuristic\_value[start];

to\_visit[0] = start;

int to\_visit\_size = 1;

int visited\_size = 0;

while (to\_visit\_size > 0) {

int current = to\_visit[0];

for (int i = 1; i < to\_visit\_size; i++) {

if (f[to\_visit[i]] < f[current]) {

current = to\_visit[i]; }}

if (current == end) {

string path = areas[start];

int total\_cost = g[end];

while (came\_from[current] != -1) {

path = areas[came\_from[current]] + " ==> " + path;

current = came\_from[current];}

cout << "Tracked Path: " << path << endl;

cout << "Shortest Distance (Estimated): " << total\_cost << " units." << endl;

delete[] to\_visit;

delete[] visited;

delete[] g;

delete[] f;

delete[] came\_from;

return;}

to\_visit\_size--;

visited[visited\_size++] = current;

for (int i = 0; i < 5; i++) {

if (pathway[current][i] == 1 || pathway[current][i] == -1) {

bool is\_visited = false;

for (int j = 0; j < visited\_size; j++) {

if (visited[j] == i) {

is\_visited = true;

break; }}

if (is\_visited) continue;

int tentative\_g = g[current] + distance[i];

bool in\_to\_visit = false;

for (int j = 0; j < to\_visit\_size; j++) {

if (to\_visit[j] == i) {

in\_to\_visit = true;

break;} }

if (!in\_to\_visit) {

to\_visit[to\_visit\_size++] = i; }

if (tentative\_g < g[i]) {

g[i] = tentative\_g;

f[i] = g[i] + heuristic\_value[i];

came\_from[i] = current; }} }

cout << "Path not found!" << endl;

delete[] to\_visit;

delete[] visited;

delete[] g;

delete[] f;

delete[] came\_from;}

int calculateActualDistance(int start, int end, int pathway[5][5], int distance[5]) {

int total\_distance = 0;

for (int i = 0; i < 5; i++) {

if (pathway[start][i] == 1 || pathway[start][i] == -1) {

total\_distance += distance[i];}

}

return total\_distance;}

};

int main() {

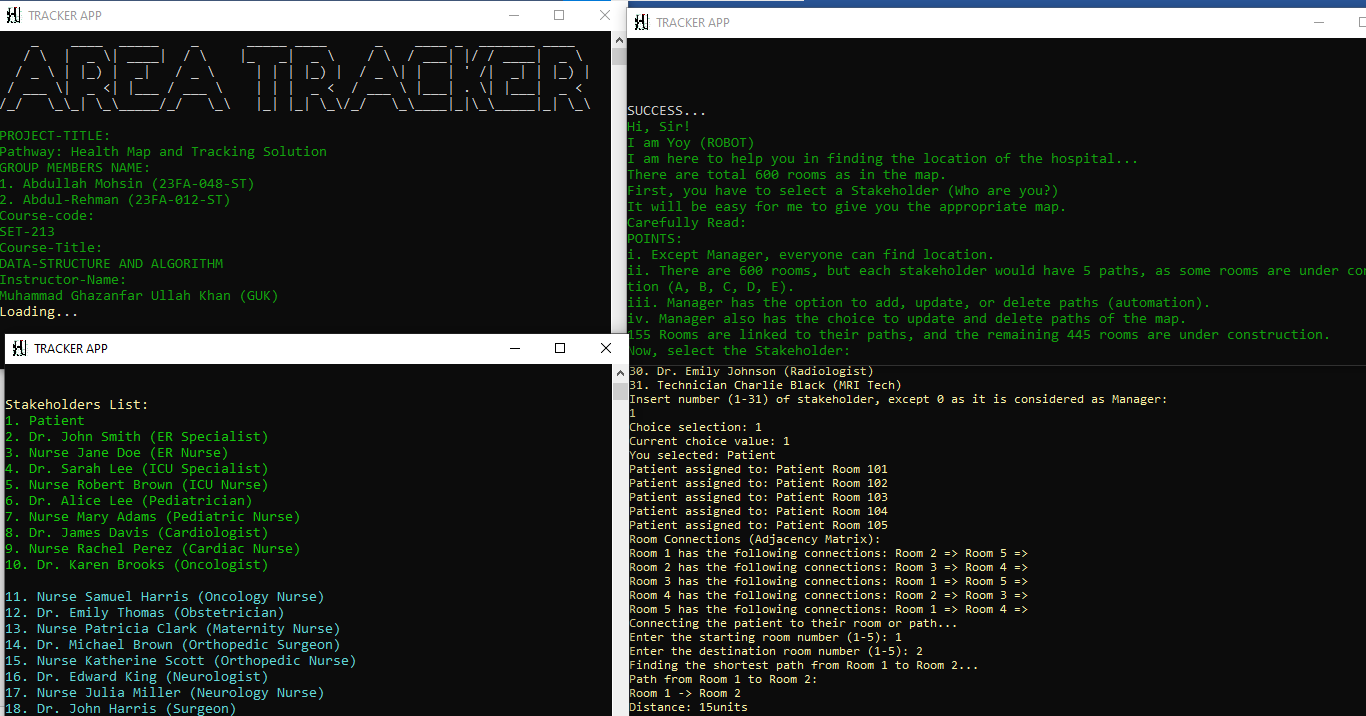
SampleCode tracker;

tracker.All();

return 0;

}

**SAMPLE OUTPUT SCREENSHOTS:**

****