Name :- Ashish Verma Soni

Reg no:- RA1911042010102

Section:- CSBS Q2

**Aim:-**

# Developing Best first search and A\* Algorithm for real world problems

# Flow of the program:-

Pseudocode:

Best-First-Search(Graph g, Node start)

1. Create an empty PriorityQueue PriorityQueue **pq**;
2. Insert "start" in pq. pq.insert(start)
3. Until PriorityQueue is empty u = PriorityQueue.DeleteMin If u is the goal

Exit Else

Foreach neighbor v of u If v "Unvisited"

Mark v "Visited" pq.insert(v)

Mark u "Examined" End procedure

**CODE:-**

#include <bits/stdc++.h>

using namespace std;

typedef pair<int, int> pi;

vector<vector<pi> > graph;

void addedge(int x, int y, int cost)

{

graph[x].push\_back(make\_pair(cost, y));

graph[y].push\_back(make\_pair(cost, x));

}

void best\_first\_search(int source, int target, int n)

{

vector<bool> visited(n, false);

priority\_queue<pi, vector<pi>, greater<pi> > pq;

pq.push(make\_pair(0, source));

int s = source;

visited[s] = true;

while (!pq.empty()) {

int x = pq.top().second;

cout << x << " ";

pq.pop();

if (x == target)

break;

for (int i = 0; i < graph[x].size(); i++) {

if (!visited[graph[x][i].second]) {

visited[graph[x][i].second] = true;

pq.push(make\_pair(graph[x][i].first,graph[x][i].second));

}

}

}

}

int main()

{

int v = 14;

graph.resize(v);

addedge(0, 1, 3);

addedge(0, 2, 6);

addedge(0, 3, 5);

addedge(1, 4, 9);

addedge(1, 5, 8);

addedge(2, 6, 12);

addedge(2, 7, 14);

addedge(3, 8, 7);

addedge(8, 9, 5);

addedge(8, 10, 6);

addedge(9, 11, 1);

addedge(9, 12, 10);

addedge(9, 13, 2);

int source = 0;

int target = 9;

best\_first\_search(source, target, v);

return 0;

}

**OUTPUT:-**

**

Developing A\* algorithm

**Code:**

#include <bits/stdc++.h>

using namespace std;

#define ROW 9

#define COL 10

typedef pair<int, int> Pair;

typedef pair<double, pair<int, int> > pPair;

struct cell {

int parent\_i, parent\_j;

double f, g, h;

};

bool isValid(int row, int col)

{

return (row >= 0) && (row < ROW) && (col >= 0)

&& (col < COL);

}

bool isUnBlocked(int grid[][COL], int row, int col)

{

if (grid[row][col] == 1)

return (true);

else

return (false);

}

bool isDestination(int row, int col, Pair dest)

{

if (row == dest.first && col == dest.second)

return (true);

else

return (false);

}

double calculateHValue(int row, int col, Pair dest)

{

return ((double)sqrt(

(row - dest.first) \* (row - dest.first)

+ (col - dest.second) \* (col - dest.second)));

}

void tracePath(cell cellDetails[][COL], Pair dest)

{

printf("\nThe Path is ");

int row = dest.first;

int col = dest.second;

stack<Pair> Path;

while (!(cellDetails[row][col].parent\_i == row

&& cellDetails[row][col].parent\_j == col)) {

Path.push(make\_pair(row, col));

int temp\_row = cellDetails[row][col].parent\_i;

int temp\_col = cellDetails[row][col].parent\_j;

row = temp\_row;

col = temp\_col;

}

Path.push(make\_pair(row, col));

while (!Path.empty()) {

pair<int, int> p = Path.top();

Path.pop();

printf("-> (%d,%d) ", p.first, p.second);

}

return;

}

void aStarSearch(int grid[][COL], Pair src, Pair dest)

{

if (isValid(src.first, src.second) == false) {

printf("Source is invalid\n");

return;

}

if (isValid(dest.first, dest.second) == false) {

printf("Destination is invalid\n");

return;

}

if (isUnBlocked(grid, src.first, src.second) == false

|| isUnBlocked(grid, dest.first, dest.second)

== false) {

printf("Source or the destination is blocked\n");

return;

}

if (isDestination(src.first, src.second, dest)

== true) {

printf("We are already at the destination\n");

return;

}

bool closedList[ROW][COL];

memset(closedList, false, sizeof(closedList));

cell cellDetails[ROW][COL];

int i, j;

for (i = 0; i < ROW; i++) {

for (j = 0; j < COL; j++) {

cellDetails[i][j].f = FLT\_MAX;

cellDetails[i][j].g = FLT\_MAX;

cellDetails[i][j].h = FLT\_MAX;

cellDetails[i][j].parent\_i = -1;

cellDetails[i][j].parent\_j = -1;

}

}

i = src.first, j = src.second;

cellDetails[i][j].f = 0.0;

cellDetails[i][j].g = 0.0;

cellDetails[i][j].h = 0.0;

cellDetails[i][j].parent\_i = i;

cellDetails[i][j].parent\_j = j;

set<pPair> openList;

openList.insert(make\_pair(0.0, make\_pair(i, j)));

bool foundDest = false;

while (!openList.empty()) {

pPair p = \*openList.begin();

// Remove this vertex from the open list

openList.erase(openList.begin());

// Add this vertex to the closed list

i = p.second.first;

j = p.second.second;

closedList[i][j] = true;

double gNew, hNew, fNew;

if (isValid(i - 1, j) == true) {

if (isDestination(i - 1, j, dest) == true) {

cellDetails[i - 1][j].parent\_i = i;

cellDetails[i - 1][j].parent\_j = j;

printf("The destination cell is found\n");

tracePath(cellDetails, dest);

foundDest = true;

return;

}

else if (closedList[i - 1][j] == false

&& isUnBlocked(grid, i - 1, j)

== true) {

gNew = cellDetails[i][j].g + 1.0;

hNew = calculateHValue(i - 1, j, dest);

fNew = gNew + hNew;

if (cellDetails[i - 1][j].f == FLT\_MAX

|| cellDetails[i - 1][j].f > fNew) {

openList.insert(make\_pair(

fNew, make\_pair(i - 1, j)));

cellDetails[i - 1][j].f = fNew;

cellDetails[i - 1][j].g = gNew;

cellDetails[i - 1][j].h = hNew;

cellDetails[i - 1][j].parent\_i = i;

cellDetails[i - 1][j].parent\_j = j;

}

}

}

if (isValid(i + 1, j) == true) {

if (isDestination(i + 1, j, dest) == true) {

cellDetails[i + 1][j].parent\_i = i;

cellDetails[i + 1][j].parent\_j = j;

printf("The destination cell is found\n");

tracePath(cellDetails, dest);

foundDest = true;

return;

}

else if (closedList[i + 1][j] == false

&& isUnBlocked(grid, i + 1, j)

== true) {

gNew = cellDetails[i][j].g + 1.0;

hNew = calculateHValue(i + 1, j, dest);

fNew = gNew + hNew;

if (cellDetails[i + 1][j].f == FLT\_MAX

|| cellDetails[i + 1][j].f > fNew) {

openList.insert(make\_pair(

fNew, make\_pair(i + 1, j)));

cellDetails[i + 1][j].f = fNew;

cellDetails[i + 1][j].g = gNew;

cellDetails[i + 1][j].h = hNew;

cellDetails[i + 1][j].parent\_i = i;

cellDetails[i + 1][j].parent\_j = j;

}

}

}

if (isValid(i, j + 1) == true) {

if (isDestination(i, j + 1, dest) == true) {

cellDetails[i][j + 1].parent\_i = i;

cellDetails[i][j + 1].parent\_j = j;

printf("The destination cell is found\n");

tracePath(cellDetails, dest);

foundDest = true;

return;

}

else if (closedList[i][j + 1] == false

&& isUnBlocked(grid, i, j + 1)

== true) {

gNew = cellDetails[i][j].g + 1.0;

hNew = calculateHValue(i, j + 1, dest);

fNew = gNew + hNew;

if (cellDetails[i][j + 1].f == FLT\_MAX

|| cellDetails[i][j + 1].f > fNew) {

openList.insert(make\_pair(

fNew, make\_pair(i, j + 1)));

cellDetails[i][j + 1].f = fNew;

cellDetails[i][j + 1].g = gNew;

cellDetails[i][j + 1].h = hNew;

cellDetails[i][j + 1].parent\_i = i;

cellDetails[i][j + 1].parent\_j = j;

}

}

}

if (isValid(i, j - 1) == true) {

if (isDestination(i, j - 1, dest) == true) {

cellDetails[i][j - 1].parent\_i = i;

cellDetails[i][j - 1].parent\_j = j;

printf("The destination cell is found\n");

tracePath(cellDetails, dest);

foundDest = true;

return;

}

else if (closedList[i][j - 1] == false

&& isUnBlocked(grid, i, j - 1)

== true) {

gNew = cellDetails[i][j].g + 1.0;

hNew = calculateHValue(i, j - 1, dest);

fNew = gNew + hNew;

if (cellDetails[i][j - 1].f == FLT\_MAX

|| cellDetails[i][j - 1].f > fNew) {

openList.insert(make\_pair(

fNew, make\_pair(i, j - 1)));

cellDetails[i][j - 1].f = fNew;

cellDetails[i][j - 1].g = gNew;

cellDetails[i][j - 1].h = hNew;

cellDetails[i][j - 1].parent\_i = i;

cellDetails[i][j - 1].parent\_j = j;

}

}

}

if (isValid(i - 1, j + 1) == true) {

if (isDestination(i - 1, j + 1, dest) == true) {

cellDetails[i - 1][j + 1].parent\_i = i;

cellDetails[i - 1][j + 1].parent\_j = j;

printf("The destination cell is found\n");

tracePath(cellDetails, dest);

foundDest = true;

return;

}

else if (closedList[i - 1][j + 1] == false

&& isUnBlocked(grid, i - 1, j + 1)

== true) {

gNew = cellDetails[i][j].g + 1.414;

hNew = calculateHValue(i - 1, j + 1, dest);

fNew = gNew + hNew;

if (cellDetails[i - 1][j + 1].f == FLT\_MAX

|| cellDetails[i - 1][j + 1].f > fNew) {

openList.insert(make\_pair(

fNew, make\_pair(i - 1, j + 1)));

cellDetails[i - 1][j + 1].f = fNew;

cellDetails[i - 1][j + 1].g = gNew;

cellDetails[i - 1][j + 1].h = hNew;

cellDetails[i - 1][j + 1].parent\_i = i;

cellDetails[i - 1][j + 1].parent\_j = j;

}

}

}

if (isValid(i - 1, j - 1) == true) {

if (isDestination(i - 1, j - 1, dest) == true) {

cellDetails[i - 1][j - 1].parent\_i = i;

cellDetails[i - 1][j - 1].parent\_j = j;

printf("The destination cell is found\n");

tracePath(cellDetails, dest);

foundDest = true;

return;

}

else if (closedList[i - 1][j - 1] == false

&& isUnBlocked(grid, i - 1, j - 1)

== true) {

gNew = cellDetails[i][j].g + 1.414;

hNew = calculateHValue(i - 1, j - 1, dest);

fNew = gNew + hNew;

if (cellDetails[i - 1][j - 1].f == FLT\_MAX

|| cellDetails[i - 1][j - 1].f > fNew) {

openList.insert(make\_pair(

fNew, make\_pair(i - 1, j - 1)));

cellDetails[i - 1][j - 1].f = fNew;

cellDetails[i - 1][j - 1].g = gNew;

cellDetails[i - 1][j - 1].h = hNew;

cellDetails[i - 1][j - 1].parent\_i = i;

cellDetails[i - 1][j - 1].parent\_j = j;

}

}

}

if (isValid(i + 1, j + 1) == true) {

if (isDestination(i + 1, j + 1, dest) == true) {

cellDetails[i + 1][j + 1].parent\_i = i;

cellDetails[i + 1][j + 1].parent\_j = j;

printf("The destination cell is found\n");

tracePath(cellDetails, dest);

foundDest = true;

return;

}

else if (closedList[i + 1][j + 1] == false

&& isUnBlocked(grid, i + 1, j + 1)

== true) {

gNew = cellDetails[i][j].g + 1.414;

hNew = calculateHValue(i + 1, j + 1, dest);

fNew = gNew + hNew;

if (cellDetails[i + 1][j + 1].f == FLT\_MAX

|| cellDetails[i + 1][j + 1].f > fNew) {

openList.insert(make\_pair(

fNew, make\_pair(i + 1, j + 1)));

cellDetails[i + 1][j + 1].f = fNew;

cellDetails[i + 1][j + 1].g = gNew;

cellDetails[i + 1][j + 1].h = hNew;

cellDetails[i + 1][j + 1].parent\_i = i;

cellDetails[i + 1][j + 1].parent\_j = j;

}

}

}

if (isValid(i + 1, j - 1) == true) {

if (isDestination(i + 1, j - 1, dest) == true) {

cellDetails[i + 1][j - 1].parent\_i = i;

cellDetails[i + 1][j - 1].parent\_j = j;

printf("The destination cell is found\n");

tracePath(cellDetails, dest);

foundDest = true;

return;

}

else if (closedList[i + 1][j - 1] == false

&& isUnBlocked(grid, i + 1, j - 1)

== true) {

gNew = cellDetails[i][j].g + 1.414;

hNew = calculateHValue(i + 1, j - 1, dest);

fNew = gNew + hNew;

if (cellDetails[i + 1][j - 1].f == FLT\_MAX

|| cellDetails[i + 1][j - 1].f > fNew) {

openList.insert(make\_pair(

fNew, make\_pair(i + 1, j - 1)));

cellDetails[i + 1][j - 1].f = fNew;

cellDetails[i + 1][j - 1].g = gNew;

cellDetails[i + 1][j - 1].h = hNew;

cellDetails[i + 1][j - 1].parent\_i = i;

cellDetails[i + 1][j - 1].parent\_j = j;

}

}

}

}

if (foundDest == false)

printf("Failed to find the Destination Cell\n");

return;

}

int main()

{

int grid[ROW][COL]

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

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

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

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

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

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

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

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

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

Pair src = make\_pair(8, 0);

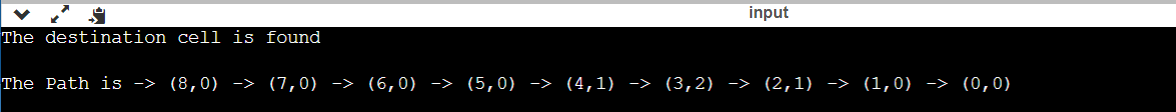
Pair dest = make\_pair(0, 0);

aStarSearch(grid, src, dest);

return (0);

}

# *OUTPUT :*

**