Maze7x10.txt

|  | **Memory** | **Nodes Explored** | **Path Length** | **Execution Time (ticks)** | **Cost** |
| --- | --- | --- | --- | --- | --- |
| **Breadth First** | 4 | 29 | 13 | 21582 | 37 |
| **Depth First** | 9 | 20 | 19 | 14532 | 45 |
| **Uniform Cost** | 4 | 29 | 15 | 17706 | 39 |
| **Iterative Deepening** | 9 | 20 | 19 | 30184 | 45 |

namespace Assignment1

{

class Program

{

static ArrayList maze = new ArrayList();

static ArrayList paths;

static void Main(string[] args) {

Console.WriteLine("Please enter the file name:");

string fileName = Console.ReadLine();

readFromFile(fileName);

Console.WriteLine("Select an algorithm: (enter the number)");

Console.WriteLine("1) Breadth First Search");

Console.WriteLine("2) Depth First Search");

Console.WriteLine("3) Uniform-Cost Search");

int ans = int.Parse(Console.ReadLine());

switch (ans) {

case 1: BFS();

break;

case 2: DFS();

break;

case 3: UCS();

break;

}

foreach (ArrayList list in maze) {

for (int i = 0; i < list.Count; i++)

if (list[i] != null)

Console.Write(list[i]);

else

Console.Write(" ");

Console.WriteLine();

}

Console.ReadLine();

}

static void readFromFile(string fileName) {

try {

StreamReader reader = new StreamReader(fileName);

while (!reader.EndOfStream) {

string line = reader.ReadLine();

ArrayList list = new ArrayList();

foreach (char x in line)

list.Add(int.Parse(x.ToString()));

maze.Add(list);

}

} catch (Exception e) {

Console.WriteLine("Exception: " + e.Message);

}

}

static void BFS() {

Queue<Position>frontier = new Queue<Position>();

HashSet<Position> explored = new HashSet<Position>();

paths = new ArrayList();

// Add initial State to frontier

foreach (int x in (ArrayList)maze[0])

if (x > 0) {

int pos = ((ArrayList)maze[0]).IndexOf(x);

frontier.Enqueue(new Position(0, pos));

break;

}

// Find Goal

Position goal = new Position(0, 0);

foreach (int x in (ArrayList)maze[maze.Count - 1])

if (x > 0) {

int pos = ((ArrayList)maze[maze.Count - 1]).IndexOf(x);

goal = new Position(maze.Count - 1, pos);

break;

}

while (!queueContains(goal, frontier) && frontier.Count > 0) {

// explore a new position

Position curPos = frontier.Dequeue();

explored.Add(curPos);

addToPath(curPos);

// Find new states for the frontier

ArrayList tempFront = new ArrayList();

if (curPos.y - 1 < ((ArrayList)maze[0]).Count)

tempFront.Add(new Position(curPos.x, curPos.y - 1));

if (curPos.x + 1 < maze.Count)

tempFront.Add(new Position(curPos.x + 1, curPos.y));

if (curPos.y + 1 >= 0)

tempFront.Add(new Position(curPos.x, curPos.y + 1));

if (curPos.x - 1 >= 0)

tempFront.Add(new Position(curPos.x - 1, curPos.y));

// Add new states to the frontier

foreach (Position pos in tempFront)

if ((int)((ArrayList)maze[pos.x])[pos.y] > 0 && !queueContains(pos, frontier) && !hashContains(pos, explored))

frontier.Enqueue(pos);

}

// Find the final path

ArrayList path = new ArrayList();

foreach (ArrayList list in paths)

if (canReach(goal, (Position)list[list.Count - 1])) {

path = list;

break;

}

// Display path

for (int x = 0; x < maze.Count-1; x++)

for (int y = 0; y < ((ArrayList)maze[x]).Count; y++)

if ((int)((ArrayList)maze[x])[y] != 0 && !arrayListContains(new Position(x, y), path))

((ArrayList)maze[x])[y] = null;

}

static void DFS() {

Stack<Position> frontier = new Stack<Position>();

HashSet<Position> explored = new HashSet<Position>();

paths = new ArrayList();

// Add initial State to frontier

foreach (int x in (ArrayList)maze[0])

if (x > 0) {

int pos = ((ArrayList)maze[0]).IndexOf(x);

frontier.Push(new Position(0, pos));

break;

}

// Find Goal

Position goal = new Position(0, 0);

foreach (int x in (ArrayList)maze[maze.Count - 1])

if (x > 0) {

int pos = ((ArrayList)maze[maze.Count - 1]).IndexOf(x);

goal = new Position(maze.Count - 1, pos);

break;

}

while (!stackContains(goal, frontier) && frontier.Count > 0) {

// explore a new position

Position curPos = frontier.Pop();

explored.Add(curPos);

addToPath(curPos);

// Find new states for the frontier

ArrayList tempFront = new ArrayList();

if (curPos.x - 1 >= 0)

tempFront.Add(new Position(curPos.x - 1, curPos.y));

if (curPos.y + 1 >= 0)

tempFront.Add(new Position(curPos.x, curPos.y + 1));

if (curPos.x + 1 < maze.Count)

tempFront.Add(new Position(curPos.x + 1, curPos.y));

if (curPos.y - 1 < ((ArrayList)maze[0]).Count)

tempFront.Add(new Position(curPos.x, curPos.y - 1));

// Add new states to the frontier

foreach (Position pos in tempFront)

if ((int)((ArrayList)maze[pos.x])[pos.y] > 0 && !stackContains(pos, frontier) && !hashContains(pos, explored))

frontier.Push(pos);

}

// Find the final path

ArrayList path = new ArrayList();

foreach (ArrayList list in paths)

if (canReach(goal, (Position)list[list.Count - 1])) {

path = list;

break;

}

// Display path

for (int x = 0; x < maze.Count - 1; x++)

for (int y = 0; y < ((ArrayList)maze[x]).Count; y++)

if ((int)((ArrayList)maze[x])[y] != 0 && !arrayListContains(new Position(x, y), path))

((ArrayList)maze[x])[y] = null;

}

static void UCS() {

PriorityQueue frontier = new PriorityQueue();

HashSet<Position> explored = new HashSet<Position>();

paths = new ArrayList();

// Add initial State to frontier

foreach (int x in (ArrayList)maze[0])

if (x > 0) {

int pos = ((ArrayList)maze[0]).IndexOf(x);

frontier.Enqueue(new Position(0, pos, (int)((ArrayList)maze[0])[pos]));

break;

}

// Find Goal

Position goal = new Position(0, 0);

foreach (int x in (ArrayList)maze[maze.Count - 1])

if (x > 0) {

int pos = ((ArrayList)maze[maze.Count - 1]).IndexOf(x);

goal = new Position(maze.Count - 1, pos, (int)((ArrayList)maze[maze.Count-1])[pos]);

break;

}

while (frontier.Count > 0) {

// explore a new position

Position curPos = frontier.Dequeue();

// Have we found the goal?

if (curPos.x == goal.x && curPos.y == goal.y)

break;

explored.Add(curPos);

addToPath(curPos);

// Find new states for the frontier

ArrayList tempFront = new ArrayList();

if (curPos.y - 1 < ((ArrayList)maze[0]).Count)

tempFront.Add(new Position(curPos.x, curPos.y - 1, curPos.cost + (int)((ArrayList)maze[curPos.x])[curPos.y-1]));

if (curPos.x + 1 < maze.Count)

tempFront.Add(new Position(curPos.x + 1, curPos.y, curPos.cost + (int)((ArrayList)maze[curPos.x + 1])[curPos.y]));

if (curPos.y + 1 >= 0)

tempFront.Add(new Position(curPos.x, curPos.y + 1, curPos.cost + (int)((ArrayList)maze[curPos.x])[curPos.y + 1]));

if (curPos.x - 1 >= 0)

tempFront.Add(new Position(curPos.x - 1, curPos.y, curPos.cost + (int)((ArrayList)maze[curPos.x - 1])[curPos.y]));

// Add new states to the frontier

foreach (Position pos in tempFront)

if ((int)((ArrayList)maze[pos.x])[pos.y] > 0 && !hashContains(pos, explored)) {

if (!priorityQueueContains(pos, frontier))

frontier.Enqueue(pos);

else {

foreach (Position pos2 in frontier)

if (pos.x == pos2.x && pos.y == pos2.y)

if (pos.cost < pos2.cost)

pos2.cost = pos.cost;

}

}

}

// Find the final path

ArrayList path = new ArrayList();

foreach (ArrayList list in paths)

if (canReach(goal, (Position)list[list.Count - 1])) {

path = list;

break;

}

// Display path

for (int x = 0; x < maze.Count - 1; x++)

for (int y = 0; y < ((ArrayList)maze[x]).Count; y++)

if ((int)((ArrayList)maze[x])[y] != 0 && !arrayListContains(new Position(x, y), path))

((ArrayList)maze[x])[y] = null;

}

static void addToPath(Position newPos) {

bool placed = false;

if (paths.Count == 0) {

ArrayList newList = new ArrayList();

newList.Add(newPos);

paths.Add(newList);

return;

}

// Check the paths and add

int i = 0;

while (!placed) {

ArrayList list = (ArrayList)paths[i];

if (canReach(newPos, (Position)list[list.Count - 1])) {

paths.Add((ArrayList)list.Clone());

list.Add(newPos);

placed = true;

}

i++;

}

}

static bool canReach(Position startPos, Position endPos) {

if (startPos.x == endPos.x) {

if (startPos.y - 1 == endPos.y || startPos.y + 1 == endPos.y)

return true;

} else if (startPos.y == endPos.y) {

if (startPos.x - 1 == endPos.x || startPos.x + 1 == endPos.x)

return true;

}

return false;

}

static bool arrayListContains(Position position, ArrayList list) {

foreach (Position pos in list)

if (pos.x == position.x && pos.y == position.y)

return true;

return false;

}

static bool queueContains(Position position, Queue<Position> queue) {

foreach (Position pos in queue)

if (pos.x == position.x && pos.y == position.y)

return true;

return false;

}

static bool priorityQueueContains(Position position, PriorityQueue queue) {

foreach (Position pos in queue)

if (pos.x == position.x && pos.y == position.y)

return true;

return false;

}

static bool stackContains(Position position, Stack<Position> stack) {

foreach (Position pos in stack)

if (pos.x == position.x && pos.y == position.y)

return true;

return false;

}

static bool hashContains(Position position, HashSet<Position> set) {

foreach (Position pos in set)

if (pos.x == position.x && pos.y == position.y)

return true;

return false;

}

}

class Position

{

public int x;

public int y;

public int cost;

public Position(int x, int y) {

this.x = x;

this.y = y;

this.cost = 1;

}

public Position(int x, int y, int cost) {

this.x = x;

this.y = y;

this.cost = cost;

}

}

class PriorityQueue : Queue

{

public PriorityQueue() {

}

public Position Dequeue() {

Position min = (Position)base.Dequeue();

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

Position pos = (Position)base.Dequeue();

if (pos.cost - min.cost < 0) {

Enqueue(min);

min = pos;

} else

Enqueue(pos);

}

return min;

}

public Position Peek() {

Position min = (Position)base.Dequeue();

foreach (Position pos in this)

if (pos.cost - min.cost < 0)

min = pos;

return min;

}

}

}