**Федеральное агентство связи**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра Математической кибернетики и информационных технологий

**Отчет по лабораторной работе**

по предмету «СИАОД»

на тему:

«Сетевые алгоритмы. Эвристические алгоритмы поиска путей»

Выполнил: студент группы

Алмамма Башар

Руководитель:

Кутейников Иван Александрович

Москва 2020

*Цель работы:* Реализовать алгоритм поиска кратчайшего расстояния между двумя клетками в соответствии с вариантом и алгоритм A\*. Сравнить результаты работы данного алгоритма с алгоритмом A\*.

*Вариант:* 1. Волновой алгоритм

*Код программы:*

**Волновой алгоритм**

import java.util.\*;  
  
public class Lee {  
 private Point2D startPoint, endPoint;  
 private int width, height;  
  
 Lee(int width, int height, Point2D startPoint, Point2D endPoint){  
 this.startPoint = startPoint;  
 this.endPoint = endPoint;  
 this.width = width;  
 this.height = height;  
 }  
 public void findPath(MapCell[][] map){  
 HashMap<MapCell,Integer> Map = new HashMap<MapCell,Integer>();  
 LinkedList<MapCell> list = new LinkedList<>();  
 map[startPoint.y][startPoint.x].setD(0);  
 list.add(map[startPoint.y][startPoint.x]);  
 while (!list.isEmpty()){  
 MapCell current = list.poll();  
 if(current == map[endPoint.y][endPoint.x])break;  
 for (MapCell cell:getNeighbors(map,current.getxCoord(),current.getyCoord(),width,height)) {  
 if(!cell.isPassable())continue;  
 if(cell.getD() < 0){  
 cell.setD(current.getD() + 1);  
 list.add(cell);  
 }  
 }  
 }  
 MapCell current = map[endPoint.y][endPoint.x];  
 if(current.getD() == -1)return;  
 while (current != map[startPoint.y][startPoint.x]){  
 for (MapCell cell:getNeighbors(map,current.getxCoord(),current.getyCoord(),width,height)) {  
 if(cell.getD() + 1 == current.getD()){  
 current = cell;  
 cell.setPath(true);  
 }  
 }  
 }  
 }  
 public ArrayList<MapCell> getNeighbors(MapCell[][] mapCells, int x, int y, int width, int height){  
 ArrayList<MapCell> list = new ArrayList<MapCell>();  
 if(!(x + 1 >= width))list.add(mapCells[y][x+1]);  
 if(!(y + 1 >= height))list.add(mapCells[y+1][x]);  
 if(!(x - 1 < 0))list.add(mapCells[y][x-1]);  
 if(!(y - 1 < 0))list.add(mapCells[y-1][x]);  
 if(!(x - 1 < 0) && !(y + 1 >= height))list.add(mapCells[y+1][x-1]);  
 if(!(x + 1 >= width) && !(y - 1 < 0))list.add(mapCells[y-1][x+1]);  
 if(!(x + 1 >= width) && !(y + 1 >= height))list.add(mapCells[y+1][x+1]);  
 if(!(x - 1 < 0) && !(y - 1 < 0))list.add(mapCells[y-1][x-1]);  
 return list;  
 }  
}

**A\***

import java.util.\*;  
  
public class AStar {  
 private Point2D startPoint, endPoint;  
 private int width, height;  
 private HashMap<MapCell,Integer> costMap = new HashMap<MapCell,Integer>();  
  
 AStar(int width, int height, Point2D startPoint, Point2D endPoint){  
 this.startPoint = startPoint;  
 this.endPoint = endPoint;  
 this.width = width;  
 this.height = height;  
 }  
  
 public void findPath(MapCell[][] map){  
 Comparator<MapCell> comparator = new MyComparator();  
 Queue<MapCell> front = new PriorityQueue<MapCell>(comparator);  
 HashMap<MapCell,MapCell> path = new HashMap<MapCell,MapCell>();  
 path.put(map[startPoint.y][startPoint.x],null);  
 costMap.put(map[startPoint.y][startPoint.x],0);  
 front.add(map[startPoint.y][startPoint.x]);  
 while (!front.isEmpty()){  
 MapCell current = front.poll();  
 if(current == map[endPoint.y][endPoint.x])break;  
 for (MapCell next:getNeighbors(map,current.getxCoord(),current.getyCoord(),width,height)) {  
 int new\_cost = costMap.get(current) + next.getCost();  
 if(!next.isPassable())continue;  
 if(!costMap.containsKey(next) || new\_cost < costMap.get(next)){  
 costMap.put(next,new\_cost);  
 front.add(next);  
 path.put(next, current);  
 }  
 }  
 }  
  
 MapCell current = map[endPoint.y][endPoint.x];  
 while (current != map[startPoint.y][startPoint.x]){  
 current = path.get(current);  
 current.setPath(true);  
 }  
 }  
  
  
 public ArrayList<MapCell> getNeighbors(MapCell[][] mapCells, int x, int y, int width, int height){  
 ArrayList<MapCell> list = new ArrayList<MapCell>();  
 if(!(x + 1 >= width))list.add(mapCells[y][x+1]);  
 if(!(y + 1 >= height))list.add(mapCells[y+1][x]);  
 if(!(x - 1 < 0))list.add(mapCells[y][x-1]);  
 if(!(y - 1 < 0))list.add(mapCells[y-1][x]);  
 if(!(x - 1 < 0) && !(y + 1 >= height))list.add(mapCells[y+1][x-1]);  
 if(!(x + 1 >= width) && !(y - 1 < 0))list.add(mapCells[y-1][x+1]);  
 if(!(x + 1 >= width) && !(y + 1 >= height))list.add(mapCells[y+1][x+1]);  
 if(!(x - 1 < 0) && !(y - 1 < 0))list.add(mapCells[y-1][x-1]);  
 return list;  
 }  
  
 private class MyComparator implements Comparator<MapCell> {  
  
 private double distanceToEnd(int x, int y){  
 return Math.*abs*(x - endPoint.x) + Math.*abs*(y - endPoint.y);  
 }  
  
 @Override  
 public int compare(MapCell o1, MapCell o2) {  
  
 if (distanceToEnd(o1.getxCoord(),o1.getyCoord()) + costMap.get(o1) < distanceToEnd(o2.getxCoord(),o2.getyCoord()) + costMap.get(o2)) {  
 return -1;  
 }  
 if (distanceToEnd(o1.getxCoord(),o1.getyCoord()) + costMap.get(o1) > distanceToEnd(o2.getxCoord(),o2.getyCoord()) + costMap.get(o2)) {  
 return 1;  
 }  
 return 0;  
 }  
 }  
}

|  |  |  |
| --- | --- | --- |
|  | **Волновой алгоритм** | **A\*** |
| Рисунок |  |  |
| Время в мс | **1** | **2** |
| Кол-во клеток | **20х20** | **20х20** |

|  |  |  |
| --- | --- | --- |
|  | **Волновой алгоритм** | **A\*** |
| Рисунок |  |  |
| Время в мс | **2** | **3** |
| Кол-во клеток | **30х30** | **30х30** |

Вывод: построил регулярную сеть в виде клеток с заданной стоимостью прохождения. Реализовал алгоритм поиска кратчайшего расстояния между двумя клетками в соответствии с вариантом и алгоритм A\*. При сравнении скорости работы алгоритмов, можно сделать вывод, что алгоритм A\* лучше справляется со своей задачей, но затрачивает больше времени, а волновой алгоритм быстрее, но не учитывает стоимость.