**Федеральное агентство связи**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

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

Кафедра Информатики

****

**Отчет по лабораторной работе № 7**

по дисциплине «СиАОД»

на тему:

«**Сетевые алгоритмы. Эвристические алгоритмы поиска путей**»

Выполнила: студентка группы БВТ1802

Лаврухина Елена Павловна

Руководитель:

Кутейников Иван Алексеевич

Москва 2020

# **Цель работы**

Построить регулярную сеть в виде клеток с заданной стоимостью прохождения. Реализовать алгоритм поиска кратчайшего расстояния между двумя клетками в соответствии с вариантом и алгоритм A\*. Сравнить результаты работы данного алгоритма с алгоритмом A\*. Предусмотреть задание поля в виде матрицы со стоимостью посещения клетки, читаемой из файла, либо графически с помощью пользовательского интерфейса. Разработать графический интерфейс пользователя с визуализацией поля и отображением кратчайшего расстояния между задаваемыми пользователем клетками. По результатам работы проанализировать временную сложность работы заданного алгоритма в зависимости от числа клеток.

Вариант № 19. Волновой алгоритм (алгоритм Ли)

Выполнение

Код программы

1. LeeAlgorithm

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.MouseEvent;  
import java.awt.event.MouseListener;  
  
public class LeeAlgorithm {  
 private JMapCell[][] mapCells;  
 private final int width;  
 private final int height;  
 private final Location startLoc;  
 private final Location finishLoc;  
 private static class MapCellHandler implements MouseListener {  
 private boolean modifying;  
 private boolean makePassable;  
 public void mousePressed(MouseEvent e) {  
 modifying = true;  
 JMapCell cell = (JMapCell) e.getSource();  
 makePassable = !cell.isPassable();  
 cell.setPassable(makePassable);  
 }  
 public void mouseReleased(MouseEvent e) {  
 modifying = false;  
 }  
 public void mouseEntered(MouseEvent e) {  
 if (modifying) {  
 JMapCell cell = (JMapCell) e.getSource();  
 cell.setPassable(makePassable);  
 }  
 }  
 public void mouseExited(MouseEvent e) {  
 }  
 public void mouseClicked(MouseEvent e) {  
 }  
 }  
 public LeeAlgorithm(int w, int h) {  
 if (w <= 0)  
 throw new IllegalArgumentException("Ширина должна быть > 0");  
 if (h <= 0)  
 throw new IllegalArgumentException("Высота должна быть > 0");  
 width = w;  
 height = h;  
 startLoc = new Location(2, h / 2);  
 finishLoc = new Location(w - 3, h / 2);  
 initGUI();  
 }  
 private void initGUI() {  
 JFrame frame = new JFrame("Ли");  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 Container contentPane = frame.getContentPane();  
 contentPane.setLayout(new BorderLayout());  
 GridBagLayout gbLayout = new GridBagLayout();  
 GridBagConstraints gbConstraints = new GridBagConstraints();  
 gbConstraints.fill = GridBagConstraints.*BOTH*;  
 gbConstraints.weightx = 1;  
 gbConstraints.weighty = 1;  
 gbConstraints.insets.set(0, 0, 1, 1);  
 JPanel mapPanel = new JPanel(gbLayout);  
 mapPanel.setBackground(Color.*GRAY*);  
 mapCells = new JMapCell[width][height];  
 MapCellHandler cellHandler = new MapCellHandler();  
 for (int y = 0; y < height; y++) {  
 for (int x = 0; x < width; x++) {  
 mapCells[x][y] = new JMapCell();  
 gbConstraints.gridx = x;  
 gbConstraints.gridy = y;  
 gbLayout.setConstraints(mapCells[x][y], gbConstraints);  
 mapPanel.add(mapCells[x][y]);  
 mapCells[x][y].addMouseListener(cellHandler);  
 }  
 }  
 contentPane.add(mapPanel, BorderLayout.*CENTER*);  
 JButton findPathButton = new JButton("Найти путь");  
 findPathButton.addActionListener(e -> findAndShowPath());  
 contentPane.add(findPathButton, BorderLayout.*SOUTH*);  
 frame.pack();  
 frame.setVisible(true);  
 mapCells[startLoc.xCoord][startLoc.yCoord].setEndpoint(true);  
 mapCells[finishLoc.xCoord][finishLoc.yCoord].setEndpoint(true);  
 }  
 public double dis(Location L) {  
 return Math.*sqrt*(Math.*pow*(L.xCoord-startLoc.xCoord,2)+Math.*pow*(L.yCoord-startLoc.yCoord,2));  
 }  
 private void findAndShowPath() {  
 *time*();  
 boolean activ = true;  
 System.*out*.println(mapCells[0][0].isPassable());  
 int[][] M = new int[width][height];  
 for (int i = 0; i < width; i++)  
 for (int j = 0; j < height; j++)  
 M[i][j]=-9;  
 M[startLoc.xCoord][startLoc.yCoord] = 0;  
 for (int i = 0; i < width; i++) {  
 for (int j = 0; j < height; j++) {  
 if (M[i][j] != 0) {  
 if (mapCells[i][j].isPassable()) {  
 M[i][j] = -1;  
 mapCells[i][j].setWay(false);  
 }  
 else  
 M[i][j] = -2;  
 }  
 }  
 }  
 int totalCost = 0;  
 boolean isFind = false;  
 while (activ && !isFind) {  
 activ = false;  
 for (int i = 0; i < width && !isFind; i++) {  
 for (int j = 0; j < height && !isFind; j++) {  
 Location newPos = new Location(i, j);  
 if (M[i][j] == totalCost) {  
 activ = true;  
 try {  
 if (M[i + 1][j] == -1) {  
 M[i + 1][j] = totalCost + 1;  
 }  
 if (M[i - 1][j] == -1) {  
 M[i - 1][j] = totalCost + 1;  
 }  
 if (M[i][j + 1] == -1) {  
 M[i][j + 1] = totalCost + 1;  
 }  
 if (M[i][j - 1] == -1) {  
 M[i][j - 1] = totalCost + 1;  
 }  
 }  
 catch(IndexOutOfBoundsException ignored){}  
 }  
 if (newPos.equals(finishLoc)&&M[i][j] != -1) {  
 isFind = true;  
 }  
 }  
 }  
 totalCost++;  
 }  
 *time*();  
 Location L = finishLoc;  
 while (totalCost != 0) {  
 double dist = Double.*MAX\_VALUE*;  
 if (L.equals(startLoc)) {  
 break;  
 }  
 else {  
 int i = L.xCoord;  
 int j = L.yCoord;  
 if (i-1 >= 0)  
 if (M[i-1][j] == (totalCost-1) && dist >= dis(new Location(i-1,j))) {  
 L = new Location(i-1,j);  
 mapCells[L.xCoord][L.yCoord].setWay(true);  
 totalCost--;  
 dist = dis(L);  
 }  
 if (j-1 >= 0)  
 if (M[i][j-1] == (totalCost-1) && dist >= dis(new Location(i,j-1))) {  
 L = new Location(i,j-1);  
 mapCells[L.xCoord][L.yCoord].setWay(true);  
 totalCost--;  
 dist = dis(L);  
 }  
 if (j+1 < height)  
 if (M[i][j+1] == (totalCost-1) && dist >= dis(new Location(i,j+1))) {  
 L = new Location(i,j+1);  
 mapCells[L.xCoord][L.yCoord].setWay(true);  
 totalCost--;  
 dist = dis(L);  
 }  
 if (i+1 < width)  
 if (M[i+1][j] == (totalCost-1) && dist >= dis(new Location(i+1,j))) {  
 L = new Location(i+1,j);  
 mapCells[L.xCoord][L.yCoord].setWay(true);  
 totalCost--;  
 }  
 }  
 }  
 }  
 public static void main(String[] args) {  
 LeeAlgorithm Lee = new LeeAlgorithm(40,30);  
 }  
 static long *timer* = 0;  
 public static void time() {  
 if (*timer* == 0)  
 *timer* = System.*nanoTime*();  
 else {  
 double val = (double)  
 (System.*nanoTime*()-*timer*);  
 System.*out*.println(val/1000000);  
 *timer* = 0;  
 }  
 }  
 }

1. Location

import java.util.Objects;  
  
public class Location {  
 public int xCoord;  
 public int yCoord;  
 public Location(int x, int y) {  
 xCoord = x;  
 yCoord = y;  
 }  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Location location = (Location) o;  
 return xCoord == location.xCoord &&  
 yCoord == location.yCoord;  
 }  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(xCoord, yCoord);  
 }  
}

1. JMapCell

import javax.swing.\*;  
import java.awt.\*;  
  
public class JMapCell extends JComponent {  
 private static final Dimension *CELL\_SIZE* = new Dimension(12, 12);  
 boolean endpoint = false;  
 boolean passable = true;  
 boolean path = false;  
 boolean way = false;  
 public JMapCell(boolean pass) {  
 setPreferredSize(*CELL\_SIZE*);  
 setPassable(pass);  
 }  
 public JMapCell() {  
 this(true);  
 }  
 public void setEndpoint(boolean end) {  
 endpoint = end;  
 updateAppearance();  
 }  
 public void setPassable(boolean pass) {  
 passable = pass;  
 updateAppearance();  
 }  
 public boolean isPassable() {  
 return passable;  
 }  
 public void setWay(boolean way) {  
 this.way = way;  
 this.path = false;  
 updateAppearance();  
 }  
 private void updateAppearance() {  
 if (passable) {  
 setBackground(Color.*WHITE*);  
 if (endpoint)  
 setBackground(Color.*CYAN*);  
 else if (this.way)  
 setBackground(Color.*GREEN*);  
 }  
 else {  
 setBackground(Color.*RED*);  
 }  
 }  
 protected void paintComponent(Graphics g) {  
 g.setColor(getBackground());  
 g.fillRect(0, 0, getWidth(), getHeight());  
 }  
}

Результат программы:

Таблица зависимости времени построения от числа клеток

|  |  |  |
| --- | --- | --- |
| **Количество клеток** | **Время** | **Рисунок** |
| 30х20 | 6.7107, мc |  |
| 3.2873, мc |  |
| 60х50 | 14.2836, мc |  |
| 12.3361, мc |  |
| 100х55 | 23.5517, мc |  |
| 38.4086, мc |  |

Вывод

В ходе работы мы реализовал алгоритм поиска кратчайшего расстояния между двумя клетками в соответствии с вариантом и алгоритм A\*. Проанализировав результаты программы, можно сделать вывод, что с малым количеством клеток алгоритм Ли работает намного быстрее, чем алгоритм А\*, но с большим числом клеток более оптимально использовать алгоритм А\*.