МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм A*

Студент гр. 7383 Александров Р.А.

Преподаватель Жангиров Т.Р.

Санкт-Петербург

2019

Цель работы.

Познакомиться с алгоритмом поиска A* в графе и его реализацией.

Постановка задачи.

Необходимо разработать программу, которая решает задачу построения кратчайшего пути в ориентированном графе методом A*. Каждая вершина в графе имеет буквенное обозначение ("a", "b", "c"...), каждое ребро имеет неотрицательный вес. В качестве эвристической функции следует взять близость символов, обозначающих вершины графа, в таблице ASCII.

Вар. 1м. Матрица смежности. В A* вершины именуются целыми числами (в т. ч. отрицательными).

Реализация задачи.

В ходе работы были написаны классы Runner, Vertex и Graph.

Класс Runner начинает работу программы, предоставляет консольный ввод и заполнение вершин графа.

Класс Vertex хранит название вершины, индекс, значения функций f, g, h.

Класс Graph хранит матрицу смежности adjMat, являющую собой граф; массив вершин vertexArr, количество вершин totalVerts, стартовую и конечные веришны и их индексы.

Метод void defineStartEndIndexes() инициализирует индексы начальной и конечной вершин.

Метод Vertex getMinVert(ArrayList<Vertex> openSet) возвращает вершину с минимальной функцией f из ArrayList`a открытых вершин.

Метод void removeFromList(ArrayList<Vertex> openSet, Vertex v) удаляет вершину из ArrayList`a открытых вершин.

Метод void searchAStar() осуществляет основной алгоритм A*.

Метод void printAnswer(ArrayList<Vertex> closedSet) выводит результат работы алгоритма.

Метод ArrayList<Vertex> getChildren(int index) возвращает дочерние вершины для заданной вершины.

Метод void addVertex(String label) добавляет вершины в граф.

Метод void addEdge(String start, String end, double weight) добавляет ребра с весом в матрицу смежности графа.

Исследование алгоритма.

Так как алгоритм проходит по всем вершинам, поэтому сложность алгоритма равна O(V+E). В худшем случае, число вершин, исследуемых алгоритмом, растёт экспоненциально по сравнению с длиной оптимального пути. A* может пройти по всем вершинам, поэтому сложность вырастает до $O(|E|^2+|V||E|)$.

Тестирование.

На табл. 1 представлен результат тестирования алгоритма.

Таблица 1 – Результаты работы алгоритма

Входные данные	Результат
a e a b 3.0 b c 1.0 c d 1.0 a d 5.0 d e 1.0	ade

Выводы.

В ходе лабораторной работы был изучен и реализован алгоритм поиска путей в графе A*.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

```
 public class Runner
{

    public void start() throws IOException {
        BufferedReader input = new BufferedReader(new
InputStreamReader(System.in));
String line;         int
counter = 0;
        ArrayList<String> allVertices = new ArrayList<>();
        String startVertex = null;
        String endVertex = null;
        ArrayList<Double> weights = new ArrayList<>();
        HashSet<String> notRepeatVertex = new HashSet<>();
        while ((line = input.readLine()) != null &&
!line.trim().equals("") && line.length() > 0) {
            String[] inits = line.split(" ");
            String start = inits[0];             String end =
inits[1];             if (counter != 0) {
double weight = Double.parseDouble(inits[2]);
allVertices.add(start);                 allVertices.add(end);
weights.add(weight);
notRepeatVertex.add(start);
notRepeatVertex.add(end);
            } else {
startVertex = start;
endVertex = end;
            }
counter++;
        }         if (counter ==
0) return;

        Graph graph = new Graph(notRepeatVertex.size());
for (String s : notRepeatVertex) {
graph.addVertex(s);
        }
graph.initStartVertex(startVertex);
graph.initEndVertex(endVertex);
         for (int i = 0, j = 0; i < weights.size(); i++, j++) {
graph.addEdge(allVertices.get(j), allVertices.get(j +
1), weights.get(i));
            j += 1;
```

```
        }
graph.searchAStar();
    }      public static void main(String[] args) throws
IOException {          new Runner().start();
    }
}
  public class Vertex {
private String label;
private int index;
private Vertex parent;
     private double
f;     private double
g;     private double
h;
      public Vertex(String label) {
this.label = label;          int number =
Integer.parseInt(label);          if (number
>= 0) {              h = (int)
label.charAt(0);
        } else {              char minus =
label.charAt(0);              int indexMinusInAscii =
(int) minus;              char num = label.charAt(1);
int indexNumInAscii = (int) num;              h =
indexMinusInAscii + indexNumInAscii;
        }
    }      public String
getLabel() {          return
label;
    }      public double
getG() {          return g;
    }      public double
getH() {          return h;
    }      public void
setG(double g) {          this.g
= g;
    }      public void
setH(double h) {          this.h
= h;
    }      public void
setF(double f) {          this.f
= f;
    }      public double
getF() {          return f;
    }      public Vertex
getParent() {          return
parent;
```

```
    }         public void setParent(Vertex
parent) {          this.parent = parent;
    }      public int
getIndex() {          return
index;
    }      public void setIndex(int
index) {          this.index = index;
    }
}  public class Graph {
private Vertex[] vertexArr;
private double[][] adjMat;
private int totalVerts;
     private Vertex startVertex;
private Vertex endVertex;
private int startVertexIndex;
private int endVertexIndex;
     public  Graph(int  initSize)
{               vertexArr  =  new
Vertex[initSize];          adjMat
=                             new
double[initSize][initSize];
totalVerts = 0;          for (int
i  =  0;  i  <  initSize;  i++)  {
for  (int  j  =  0;  j  <  initSize;
j++) {               adjMat[i][j]
= Integer.MAX_VALUE;
            }
        }
    }      private void defineStartEndIndexes() {
for (int i = 0; i < vertexArr.length; i++) {
if
(vertexArr[i].getLabel().equals(startVertex.getLabel())) {
startVertexIndex = i;

startVertex.setIndex(vertexArr[startVertexIndex].getIndex());
            }          }          for (int i = 0; i <
vertexArr.length; i++) {               if
(vertexArr[i].getLabel().equals(endVertex.getLabel())) {
endVertexIndex = i;

endVertex.setIndex(vertexArr[endVertexIndex].getIndex());
            }
        }
    }      private Vertex getMinVert(ArrayList<Vertex>
openSet) {          double min = Integer.MAX_VALUE;
int index = 0;          for (int i = 0; i < openSet.size();
```

```
i++) {              if (openSet.get(i).getF() <= min) {
min = openSet.get(i).getF();                  index =
openSet.get(i).getIndex();
            }         }
return vertexArr[index];
    }      private void removeFromList(ArrayList<Vertex> openSet,
Vertex v)
{         for (int i = 0; i < openSet.size(); i++) {
if (openSet.get(i).getIndex() == v.getIndex()) {
openSet.remove(openSet.get(i));
                break;
            }
        }
    }      public void searchAStar() {
defineStartEndIndexes();         if (startVertexIndex
== endVertexIndex) return;

        ArrayList<Vertex> openList = new ArrayList<>();
ArrayList<Vertex> closedList = new ArrayList<>();
vertexArr[startVertexIndex].setH(0);
vertexArr[startVertexIndex].setG(0);
vertexArr[startVertexIndex].setF(0);

        openList.add(startVertex);
while (!openList.isEmpty()) {
            Vertex curVert = getMinVert(openList);
removeFromList(openList, curVert);             if
(curVert.getIndex() == endVertexIndex) {
closedList.add(curVert);
printAnswer(closedList);                return;
            }
closedList.add(curVert);

            ArrayList<Vertex>              children              =
getChildren(curVert.getIndex());             for (Vertex
child : children) {                 if
(closedList.contains(child)) continue;

                double   weightScore    =    curVert.getG()       +
adjMat[curVert.getIndex()][child.getIndex()];
                 if (openList.contains(child) && weightScore >=
child.getG()) continue;
                 double heuristicScore;                          if
(!openList.contains(child)) {                    heuristicScore =
Math.abs(endVertex.getH()             -             child.getH());
child.setH(heuristicScore);
```

```
                }

child.setParent(curVert);
child.setG(weightScore);
child.setF(weightScore + child.getH());
                 if (openList.contains(child)) {
for (int j = 0; j < openList.size(); j++) {
if   (openList.get(j).getIndex()   == child.getIndex()) {
openList.set(j, child);                                  break;
                        }
                    }                 }
else {
openList.add(child);
                }
            }
        }
    }
    private void printAnswer(ArrayList<Vertex> closedSet) {
Vertex lastVertex = closedSet.get(closedSet.size() - 1);
        ArrayList<String> VertexesAnswer = new ArrayList<>();
VertexesAnswer.add(lastVertex.getLabel());         while
(lastVertex.getParent() != null) {
            VertexesAnswer.add(lastVertex.getParent().getLabel());
lastVertex = lastVertex.getParent();
        }
        for (int i = VertexesAnswer.size() - 1; i >= 0; i--) {
            System.out.print(VertexesAnswer.get(i) + " ");
        }
        System.out.println();
    }
    private ArrayList<Vertex> getChildren(int index) {
ArrayList<Vertex> children = new ArrayList<>();
for (int i = 0; i < totalVerts; i++) {             if
(adjMat[index][i] != Integer.MAX_VALUE) {
children.add(vertexArr[i]);
            }         }
return children;
    }      public void addVertex(String label) {
vertexArr[totalVerts] = new Vertex(label);
vertexArr[totalVerts].setIndex(totalVerts);
totalVerts++;
    }      public void addEdge(String start, String end, double
weight) {         int startIndex = 0;         int endIndex = 0;
for (int i = 0; i < vertexArr.length; i++) {             if
```

```
(vertexArr[i].getLabel().equals(start)) {
startIndex = i;
            }             if
(vertexArr[i].getLabel().equals(end)) {
endIndex = i;
            }         }
adjMat[startIndex][endIndex] = weight;
    }
      public void initStartVertex(String startVertex)
{         this.startVertex = new Vertex(startVertex);
    }      public void initEndVertex(String
endVertex) {          this.endVertex = new
Vertex(endVertex);
    }      public void
displayAj() {
        for (int i = 0; i < vertexArr.length; i++) {
            System.out.print(vertexArr[i].getLabel() + "    ");
        }
        System.out.println();          for (int i = 0; i
< adjMat.length; i++) {             for (int j = 0; j <
adjMat[i].length; j++) {                 if (j == 0) {
                    System.out.print(vertexArr[i].getLabel() + "
");
                }
                if (adjMat[i][j] == Integer.MAX_VALUE) {
                    System.out.print("-1   ");
                } else {
                    System.out.print(adjMat[i][j] + "   ");
                }
            }
            System.out.println();
        }
    }
}
```