**广东理工学院实验(训)报告**

**姓 名： 叶锦亮 学 号： 1812402301044**

**专业、班级： 18软工1班 课程名称： 数据结构与算法(Java)**

**指导教师： 翟允赛 实验地点：**

**实验时间： 2020.05.16 实验学时： 2学时**

**实验项目：10 图的遍历 成 绩：**

|  |
| --- |
| **实验目的：**  1．掌握图的遍历概念。  2．掌握用广度优先搜索算法和深度优先算法解决图遍历问题。 |
| 实验内容和要求  1、将该实验报告以“**10**+全部学号+姓名.doc”格式命名，完成后于本周周日24：00前交到班长处，班长再统一交给我，过期不候！！（以后实验都按照本次要求）。  2、实验内容及要求：:   1. 编程实现应用广度优先搜索算法确定无向图的连通分量。   E  G  F  A  B  D  C   1. 编程实现判断一个有向图中任意给定的两个顶点之间是否存在一条长度为k的简单路径。   A  B  F  E  C  D  （3）按上图编程实现应用深度优先搜索策略判断一个有向图是否存在环。 |
| **实验过程:**  节点类：  **package** graph;  **public** **class** Vertex {  **private** String vertexName;  **public** String getVertexName() {  **return** vertexName;  }  **public** **void** setVertexName(String vertexName) {  **this**.vertexName = vertexName;  }  **public** Vertex(String vertexName) {  **super**();  **this**.vertexName = vertexName;  }    **public** **static** **int** vertexIndexOF(Vertex[] vs,Vertex v) {  **for**(**int** i=0;i<vs.length;i++) {  **if**(vs[i].getVertexName().equals(v.getVertexName()))  **return** i;  }  **return** -1;  }  }  方法类：  package graph;  import queue.LoopSeqQueue1;  import stack.SeqStack;  public class MatrixGraph {  private Vertex[] vertexList;  private int[][] edgeList;  public MatrixGraph(Vertex[] vertexList, int[][] edgeList) {  super();  this.vertexList = vertexList;  this.edgeList = edgeList;  }  public Vertex[] getVertexList() {  return vertexList;  }  public void setVertexList(Vertex[] vertexList) {  this.vertexList = vertexList;  }  public int[][] getEdgeList() {  return edgeList;  }  public void setEdgeList(int[][] edgeList) {  this.edgeList = edgeList;  }  // 连通分量获取  public static void BFs\_findConnectedComponent(MatrixGraph g, Vertex start) throws Exception {  int[] visit = new int[g.vertexList.length];  for (int i = 0; i < visit.length; i++)  visit[i] = 0;  LoopSeqQueue1 q = new LoopSeqQueue1(g.vertexList.length);  // 起始顶点入队  q.offer(g.vertexList[Vertex.vertexIndexOF(g.vertexList, start)]);  // 标识已访问  visit[Vertex.vertexIndexOF(g.vertexList, start)] = 1;  while (!q.isEmpty()) {  // 访问队头顶点  Vertex temp = (Vertex) q.peek();  System.out.print(temp.getVertexName());  // 查找未访问过的邻接顶点入队  for (int i = 0; i < g.edgeList[0].length; i++) {  if (g.edgeList[Vertex.vertexIndexOF(g.vertexList, temp)][i] == 1 && !(visit[i] > 0)) {  q.offer(g.vertexList[i]);  // 标识已访问  visit[i] = 1;  }  }  // 访问结束出队.  q.poll();  // 查找下一连通分量  if (q.isEmpty()) {  for (int i = 0; i < visit.length; i++) {  if (visit[i] == 0) {  q.offer(g.vertexList[i]);  visit[i] = 1;  System.out.println();  break;  }  }  }  }  }  // 路径判断  public static boolean DFs\_findPath(MatrixGraph g, Vertex start, Vertex end, int length) throws Exception {  // 确定顶点X到顶点Y是否存在一条长度为n的路径  SeqStack s = new SeqStack(g.vertexList.length);  int[] visit = new int[g.vertexList.length];  for (int i = 0; i < visit.length; i++) {  visit[i] = 0;  }  s.push(start);  visit[Vertex.vertexIndexOF(g.vertexList, start)] = 1;  while (!s.isEmpty()) {  // 访问队头顶点  Vertex temp = (Vertex) s.peek();  // 如果路径长为length（这里使用栈的top指针作为路径长）并且当前位置的顶点为目标顶点时返回true  if (temp.getVertexName().equals(end.getVertexName()) && s.getTop() == length) {  return true;  // 如果当前顶点不是目标顶点但是路径长已经等于length时，对栈指针进行回退  } else if (s.getTop() == length) {  s.pop();  temp = (Vertex) s.peek();  }  // 查找未访问过的邻接顶点入队  for (int i = 0; i < g.edgeList[0].length; i++) {  if (g.edgeList[Vertex.vertexIndexOF(g.vertexList, temp)][i] == 1 && !(visit[i] > 0)) {  s.push(g.vertexList[i]);  // 标识已访问  visit[i] = 1;  break;  } else if (i == g.edgeList[0].length - 1) {  s.pop();  if (s.isEmpty())  for (int j = 0; j < g.vertexList.length; j++)  if (visit[j] == 0) {  s.push(g.vertexList[j]);  visit[j] = 1;  }  }  }  }  return false;  }  public static boolean DFs\_findCycle(MatrixGraph g,Vertex start) throws Exception {  SeqStack s = new SeqStack(g.vertexList.length);  int [] visit = new int[g.vertexList.length];  for(int i=0;i<visit.length;i++) {  visit[i] = 0;  }    s.push(start);  visit[Vertex.vertexIndexOF(g.vertexList, start)] = 1;  while(!s.isEmpty()) {  //访问队头顶点  Vertex temp = (Vertex) s.peek();    //判断当前顶点的所有邻接顶点是否在栈中存在，如果存在则图中存在环  for (int i = 0; i < g.edgeList[0].length; i++) {  if(g.edgeList[Vertex.vertexIndexOF(g.vertexList, temp)][i] == 1 &&isAlive(s,g.vertexList[i])) {  return true;  }  }    //查找未访问过的邻接顶点入队  for (int i = 0; i < g.edgeList[0].length; i++) {  if (g.edgeList[Vertex.vertexIndexOF(g.vertexList, temp)][i] == 1 && !(visit[i]>0)) {  s.push(g.vertexList[i]);  //标识已访问  visit[i] = 1;  break;  //如果没有邻接顶点则进行回退  }else if(i == g.edgeList[0].length-1) {  s.pop();  //如果回退到栈为空，则寻找其他未访问的顶点开始搜索  if(s.isEmpty())  for(int j=0;j<g.vertexList.length;j++)  if(visit[j] == 0) {  s.push(g.vertexList[j]);  visit[j] = 1;  }  }  }  }  return false;  }    private static boolean isAlive(SeqStack s,Vertex v) throws Exception {  SeqStack s1 = new SeqStack();  while(!s.isEmpty()&&!((Vertex)s.peek()).getVertexName().equals(v.getVertexName())) {  s1.push(s.pop());  }  if(s.isEmpty()) {  while(!s1.isEmpty())  s.push(s1.pop());  return false;  }else  return true;  }  }  测试类：  **package** graph;  **public** **class** MainTest {  **public** **static** **void** main(String[] args) **throws** Exception {  Vertex[] vs0 = {  **new** Vertex("A"),  **new** Vertex("B"),  **new** Vertex("C"),  **new** Vertex("D"),  **new** Vertex("E"),  **new** Vertex("F"),  **new** Vertex("G")  };  **int**[][] es0 = {  {0,1,0,1,0,0,0},  {1,0,1,0,0,0,0},  {0,1,0,1,0,0,0},  {1,0,1,0,0,0,0},  {0,0,0,0,0,1,0},  {0,0,0,0,1,0,1},  {0,0,0,0,0,1,0}  };    MatrixGraph g0 = **new** MatrixGraph(vs0, es0);  System.***out***.println("图中的所有连通分量：");  MatrixGraph.*BFs\_findConnectedComponent*(g0, **new** Vertex("A"));  System.***out***.println();    Vertex[] vs = {  **new** Vertex("A"),  **new** Vertex("B"),  **new** Vertex("C"),  **new** Vertex("D"),  **new** Vertex("E"),  **new** Vertex("F")  };  **int**[][] es = {  {0,1,0,0,0,0},  {0,0,1,0,1,0},  {0,0,0,1,0,0},  {0,0,0,0,1,0},  {0,0,0,0,0,1},  {1,1,0,0,0,0}  };  MatrixGraph g = **new** MatrixGraph(vs, es);  System.***out***.println("判断图中是否存在A到F的长为3的简单路径："+MatrixGraph.*DFs\_findPath*(g, **new** Vertex("A"), **new** Vertex("F"), 3));  System.***out***.println("判断图中是否存在环："+MatrixGraph.*DFs\_findCycle*(g, **new** Vertex("A")));  }  } |
| **实验结果：** |
| **实验总结**  **这次实验深度优先搜索没有使用递归进行遍历，而是另外使用了栈来对图进行遍历，如果方法有误希望老师指正。** |