**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： 2020.05.27 实验学时： 2学时**

**实验项目：11 图的最短路径 成 绩：**

|  |
| --- |
| **实验目的：**  1．掌握图的最短路径的应用。  2．求每一对顶点的最短路径算法应用。 |
| 实验内容和要求:  1、将该实验报告以“**11**+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. 有四个点A.B.C.D，如下图，图中弧上的数值为两点间距离，现要选择在某点建立一点应位于图的中心， 2. 求出各点之间最短路径，并以矩阵的形式输出。 3. 确定要选取的某点位置，并求各点到此点的路径和路径长度。补充：A到C的权是3      1. 编程实现求取如图6.25所示的AOE网的关键路径。(见教材P232-P234 例题6.6)     **图6.25 AOE网**    **图6.26 图6.25 所示AOE网的两条关键路径** |
| **实验过程:**   1. 方法：   **public** **static** **void** singleSourceShortestPath(MatrixGraph g, Vertex start, Vertex end) {  **boolean**[] S = **new** **boolean**[g.vertexList.length];  **int**[] partentPath = **new** **int**[g.vertexList.length];  **int**[] dist = **new** **int**[g.vertexList.length];  **for** (**int** i = 0; i < S.length; i++) {  dist[i] = g.edgeList[Vertex.*vertexIndexOF*(g.vertexList, start)][i];  **if** (dist[i] < Integer.***MAX\_VALUE***) {  S[i] = **false**;  partentPath[i] = Vertex.*vertexIndexOF*(g.vertexList, start);  } **else** **if** (i != Vertex.*vertexIndexOF*(g.vertexList, start)) {  S[i] = **false**;  partentPath[i] = -1;  } **else** {  S[i] = **true**;  partentPath[i] = -1;  }  }  **int** minVertex = -1;  **for** (**int** i = 0; i < S.length - 1; i++) {  **int** min = Integer.***MAX\_VALUE***;  **for** (**int** j = 0; j < S.length; j++) {  **if** (!S[j] && dist[j] < min) {  min = dist[j];  minVertex = j;  }  }  S[minVertex] = **true**;  **for** (**int** j = 0; j < S.length; j++) {  **if** (!S[j] && g.edgeList[minVertex][j] < Integer.***MAX\_VALUE***) {  dist[j] = g.edgeList[minVertex][j];  partentPath[j] = minVertex;  }  }  }  StringBuffer sb = **new** StringBuffer();  **int** length = 0;  sb.append(end.getVertexName());  **for** (**int** i = Vertex.*vertexIndexOF*(g.vertexList, end); partentPath[i] > -1; i = partentPath[i]) {  sb.append(g.vertexList[partentPath[i]].getVertexName());  length += dist[i];  }  sb.reverse();  sb.append("," + length);  System.***out***.println(sb.toString());  }  **public** **static** **int**[][] floyd(MatrixGraph g) {  **boolean** P[][][] = **new** **boolean**[g.vertexList.length][g.vertexList.length][g.vertexList.length];  **int**[][] D = **new** **int**[g.vertexList.length][g.vertexList.length];  **for** (**int** v = 0; v < g.vertexList.length; v++) {  **for** (**int** w = 0; w < g.vertexList.length; w++) {  D[v][w] = g.edgeList[v][w];  **for** (**int** u = 0; u < g.vertexList.length; u++) {  P[v][w][u] = **false**;  }  **if** (D[v][w] < Integer.***MAX\_VALUE***) {  P[v][w][v] = **true**;  P[v][w][w] = **true**;  }  }  }  **for** (**int** u = 0; u < g.vertexList.length; u++) {  **for** (**int** v = 0; v < g.vertexList.length; v++) {  **for** (**int** w = 0; w < g.vertexList.length; w++) {  **if** (D[v][u] < Integer.***MAX\_VALUE*** && D[u][w] < Integer.***MAX\_VALUE*** && D[v][u] + D[u][w] < D[v][w]) {  D[v][w] = D[v][u] + D[u][w];  **for** (**int** i = 0; i < g.vertexList.length; i++) {  P[v][w][i] = P[v][u][i] || P[u][w][i];  }  }  }  }  }  **return** D;  }  测试类：  Vertex[] vs3 = { **new** Vertex("A"), **new** Vertex("B"), **new** Vertex("C"), **new** Vertex("D") };  **int**[][] es3 = { { Integer.***MAX\_VALUE***, 15, 3, Integer.***MAX\_VALUE*** },  { 10, Integer.***MAX\_VALUE***, 2, Integer.***MAX\_VALUE*** },  { Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, 2 },  { Integer.***MAX\_VALUE***, 8, 4, Integer.***MAX\_VALUE*** }  };  MatrixGraph g3 = **new** MatrixGraph(vs3, es3);  System.***out***.println("各点间最短距离矩阵如下：");  **int**[][] D = MatrixGraph.*floyd*(g3);  **for** (**int** i = 0; i < D.length; i++) {  **for** (**int** j = 0; j < D.length; j++) {  **if** (j == D.length - 1) {  System.***out***.println(D[i][j]);  } **else** {  System.***out***.print(D[i][j] + ",");  }  }  }  System.***out***.println("各点到D点的路径及路径长为：");  **for** (Vertex v : g3.getVertexList()) {  **if** (!v.getVertexName().equals("D"))  MatrixGraph.*singleSourceShortestPath*(g3, v, **new** Vertex("D"));  }  2. 关键路径  方法类：  **package** graph;  **import** stack.SeqStack;  **public** **class** CriticalPath {  **private** SeqStack T = **new** SeqStack();  **private** **int**[] ve, vl;  **public** **boolean** topologicalOrder(MatrixGraph g) **throws** Exception {  **int** count = 0;  **int**[][] vs = g.getEdgeList();  **int** v\_len = g.getVertexList().length;  **int**[] indegree = **new** **int**[v\_len];  ve = **new** **int**[v\_len];  SeqStack S = **new** SeqStack();  // 入度统计  **for** (**int** i = 0; i < v\_len; i++) {  **for** (**int** j = 0; j < v\_len; j++) {  **if** (vs[i][j] < Integer.***MAX\_VALUE***) {  indegree[j]++;  }  }  }  **for** (**int** i = 0; i < v\_len; i++) {  **if** (indegree[i] == 0) {  S.push(i);  }  }  **while** (!S.isEmpty()) {  **int** j = (Integer) S.pop();  T.push(j);  ++count;  **for** (**int** a = 0; a < v\_len; a++) {  **if** (g.getEdgeList()[j][a] < Integer.***MAX\_VALUE***) {  **if** (--indegree[a] == 0) {  S.push(a);  }  **if** (ve[j] + g.getEdgeList()[j][a] > ve[a]) {  ve[a] = ve[j] + g.getEdgeList()[j][a];  }  }  }  }      **return** !(count<v\_len);  }    **public** **boolean** importantPath(MatrixGraph g) **throws** Exception {  **if**(!topologicalOrder(g)) {  **return** **false**;  }  **int** v\_len = g.getVertexList().length;  vl = **new** **int**[v\_len];  **for**(**int** i=0;i<v\_len;i++) {  vl[i] = ve[v\_len-1];  }  **while**(!T.isEmpty()) {  **int** j = (Integer) T.pop();  **for**(**int** a=0;a<v\_len;a++) {  **if**(g.getEdgeList()[a][j]<Integer.***MAX\_VALUE***) {  **int** value = g.getEdgeList()[a][j];  **if**(vl[j]-value<vl[a]) {  vl[a] = vl[j] -value;  }  }  }  }    System.***out***.println("关键路径如下：");  **for**(**int** j=0;j<v\_len;j++) {  **for**(**int** k=0;k<v\_len;k++ ) {  **if**(g.getEdgeList()[j][k]<Integer.***MAX\_VALUE***) {  **int** value = g.getEdgeList()[j][k];  **int** ee = ve[j];  **int** el = vl[k]-value;  **if**(ee==el) {  System.***out***.println(g.getVertexList()[j].getVertexName()+"->"+g.getVertexList()[k].getVertexName());  }  }  }  }  **return** **true**;  }    }  测试类：  **package** graph;  **import** stack.SeqStack;  **public** **class** MainTest {  **public** **static** **void** main(String[] args) **throws** Exception {  Vertex[] vs4 = { **new** Vertex("v0"), **new** Vertex("v1"), **new** Vertex("v2"), **new** Vertex("v3"), **new** Vertex("v4"),  **new** Vertex("v5"), **new** Vertex("v6"), **new** Vertex("v7"), **new** Vertex("v8") };  **int**[][] es4 = {  { Integer.***MAX\_VALUE***, 6, 4, 5, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***,  Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE*** },  { Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, 1, Integer.***MAX\_VALUE***,  Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE*** },  { Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, 1, Integer.***MAX\_VALUE***,  Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE*** },  { Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, 2,  Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE*** },  { Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***,  Integer.***MAX\_VALUE***, 9, 7, Integer.***MAX\_VALUE*** },  { Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***,  Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, 4, Integer.***MAX\_VALUE*** },  { Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***,  Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, 2 },  { Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***,  Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, 4 },  { Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***,  Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE*** } };  MatrixGraph g4 = **new** MatrixGraph(vs4, es4);  CriticalPath cp = **new** CriticalPath();  cp.importantPath(g4);  }  } |
| **实验结果：** |
| **实验总结**  **通过这次实验我掌握了对图最短路径的求法和关键路径的求法，谢谢老师的细心指导。** |