**重 庆 大 学**

**学 生 实 验 报 告**

**实验课程名称 数据结构与算法**

**开课实验室 D1501**

**学 院 软件学院 年级 2016 专业班 软件2班**

**学 生 姓 名 丁子元 学 号 20161616**

**开 课 时 间 2016 至 2017 学年第 1 学期**

|  |  |
| --- | --- |
| **总 成 绩** |  |
| **教师签名** | **文俊浩** |

**软件学院制**

**《数据结构与算法》实验报告**

**开课实验室：DS1501 2017 年 10 月　10 日**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 学院 | | 软件学院 | 年级、专业、班 | | 2016级2班 | 姓名 | 丁子元 | | 成绩 | |  |
| 课程  名称 | | 数据结构与算法 | | 实验项目  名 称 | 图的相邻矩阵表示法和邻接表表示法的C++实现,  图的拓扑排序和单源最短路径算法的C++实现 | | | 指导教师 | | **文俊浩** | |
| 教师评语 | 教师签名：  年 月 日 | | | | | | | | | | |
| **一、实验目的**  1 练习图的相邻矩阵表示法和邻接表表示法的C++实现  2 练习图的拓扑排序和单源最短路径算法的C++实现  **二、实验原理**  1 图的相邻矩阵表示法和邻接表表示法  2 图的拓扑排序和单源最短路径算法 | | | | | | | | | | | |

|  |
| --- |
| **三、使用仪器、材料**  Vs2017  **四、实验步骤**  •(1) 参考教材中的代码实现图的相邻矩阵表示法；  •(2) 输入图11.26，输出对应的相邻矩阵；  •(3) 参考教材中的代码实现图的邻接表表示法；  •(4) 输入图11.26，输出对应的邻接表。  •(1) 参考教材代码实现图的拓扑排序算法；  •(2) 参考教材代码实现图的单源最短路径算法；  •(3) 设计程序用教材的图11.26对上述实现进行测试；  •(4) 修改单源最短路径算法，使得程序不仅能给出最短路径的值，而且能够给出具体的路径， |

|  |
| --- |
| **五、实验过程原始记录(数据、图表、计算等)**  1.图的实现  1>相邻矩阵  在main.cpp中  int main()  {  Graphm test(6);  cout << "节点数:" << test.n() << endl;  cout << "边数:" << test.e() << endl << endl;  test.setEdge(0, 1, 10);  test.setEdge(0, 5, 2);  test.setEdge(0, 3, 20);  test.setEdge(1, 2, 3);  test.setEdge(1, 3, 5);  test.setEdge(2, 4, 15);  test.setEdge(3, 5, 10);  test.setEdge(3, 4, 11);  test.setEdge(5, 4, 3);  cout << "节点数:" << test.n() << endl;  cout << "边数:" << test.e() << endl << endl;  test.print();  cout << endl;  getchar();  return 0;  }  在Graphm.h中  public:  void print()  {  for (int i = 0; i < numVertex; i++)  {  for (int j = 0; j < numVertex; j++)  cout << setw(4)<< matrix[i][j]<<" ";  cout << endl;  }  }  2>邻接表  在main.cpp中  int main()  {  Graphl test(6);  cout << "节点数:"<<test.n() << endl;  cout << "边数:" << test.e() << endl<<endl;  //设置边  //cout << "设置3条权为1的边...(1,2),(1,3),(2,3)" << endl<<endl;  test.setEdge(0, 1, 10);  test.setEdge(0, 5, 2);  test.setEdge(0, 3, 20);  test.setEdge(1, 2, 3);  test.setEdge(1, 3, 5);  test.setEdge(2, 4, 15);  test.setEdge(3, 5, 10);  test.setEdge(3, 4, 11);  test.setEdge(5, 4, 3);  cout << "节点数:" << test.n() << endl;  cout << "边数:" << test.e() << endl<<endl;  for (int i = 0; i < test.n(); i++)  {  cout << i+1<< "->";  int temp = i,  temp1= test.first(temp);  if (temp1 < test.n())  cout << temp1+1<< "->";  while (test.next(temp, temp1)<test.n())  {  cout << test.next(temp, temp1)+1<<"->";  temp1 = test.next(temp, temp1);  }  cout << "/" << endl;  }  getchar();  return 0;  }  2.图的遍历  1>拓扑排序  int main()  {  void topsort(Graph\* G);  void tophelp(Graph\* G, int v);  Graphl g1(6);  cout << "节点数:" << g1.n() << endl;  cout << "边数:" << g1.e() << endl << endl;  //设置边  //cout << "设置3条权为1的边...(1,2),(1,3),(2,3)" << endl << endl;  g1.setEdge(0, 1, 10);  g1.setEdge(0, 5, 2);  g1.setEdge(0, 3, 20);  g1.setEdge(1, 2, 3);  g1.setEdge(1, 3, 5);  g1.setEdge(2, 4, 15);  g1.setEdge(3, 5, 10);  g1.setEdge(3, 4, 11);  g1.setEdge(5, 4, 3);  cout << "节点数:" << g1.n() << endl;  cout << "边数:" << g1.e() << endl << endl;  cout << "开始拓扑排序..." << endl;  topsort(&g1);    getchar();  return 0;  }  2>单源最短路径  int main()  {  void DijkstrawithQ(Graph\* G, int\* D, int s);  void Dijkstra(Graphl\* G, int \*D, int s);  int minVertex(Graphl\* G, int \*D);  Graphl g1(6);  g1.setEdge(0, 1, 10);  g1.setEdge(0, 5, 2);  g1.setEdge(0, 3, 20);  g1.setEdge(1, 2, 3);  g1.setEdge(1, 3, 5);  g1.setEdge(2, 4, 15);  g1.setEdge(3, 5, 10);  g1.setEdge(3, 4, 11);  g1.setEdge(5, 4, 3);  int \*D,s=0;  D = new int[g1.n()];  Dijkstra(&g1, D, s);  for (int i = 0; i < g1.n(); i++)  cout << D[i] << " ";  getchar();  return 0;  }  据11.11题修改  void Dijkstra(Graphl\* G, int\* D, int\* par,int s)  {//从s开始计算最短路径  int i, v, w;  for (int i = 0; i < G->n(); i++) // Initialize  {  D[i] = 65535;  }  par[s] = -1;  D[s] = 0;  for (i = 0; i<G->n(); i++)  { // Process the vertices  v = minVertex(G, D);  if (D[v] == 65535) return; // Unreachable vertices  G->setMark(v, VISITED);  for (w = G->first(v); w < G->n(); w = G->next(v, w))  {  if (D[w] > (D[v] + G->weight(v, w)))  {  D[w] = D[v] + G->weight(v, w);  par[w] = v;  }    }  }    int t;  for (i = 0; i<G->n(); i++) {  cout << "Path for " << i+1 << ": ";  for (t = i; par[t] != -1; t =par[t])  cout << t+1 << " ";  cout << s+1 << "\n";  }  } |

|  |
| --- |
| **六、实验结果及分析**  1.图的实现  1>相邻矩阵    2>邻接表    2.图的遍历  1>拓扑排序    2>单源最短路径    修改后 |