山东大学 软件 学院

数据结构 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号： | 姓名： | | 班级： |
| 实验题目：图的操作 | | | |
| 实验学时：4小时 | | 实验日期：2018.11.25 | |
| 实验目的：  掌握无向图的创建、遍历方法。 | | | |
| 硬件环境： 笔记本电脑 | | | |
| 软件环境：Win10+Vistual Studio 2017 | | | |
| 实验步骤与内容：  实验内容：  1、创建图类，存储结构使用邻接矩阵。  2、输入图的节点数n（不超过10个）、边数m，节点分别用1-n代表。  3、采用“起始节点，终止节点，权值”输入图的m条边，创建图。  4、输出从节点1开始的BFS遍历，在遍历过程中，如有多个可以选择的节点，则优先选择编号较小的节点。  5、输出从节点1开始的DFS遍历，在遍历过程中，如有多个可以选择的节点，则优先选择编号较小的节点。  6、输出从第1节点到第n节点最短路径的长度，如果没有路经，输出0。  实验步骤：  adjacencyWDigraph.h  #pragma once  class adjacencyWDigraph  {  //加权无向图的邻接矩阵表示  public:  adjacencyWDigraph(int Vertices = 10, int noEdge = 0);  ~adjacencyWDigraph() { deleteArray(a, n + 1); }  bool exist(int i, int j)const;  adjacencyWDigraph& Add(int i, int j, const int& w);  adjacencyWDigraph& Delete(int i, int j);  void makeArray(int\*\* &x, int numberOfRows, int numberOfColumns);  void deleteArray(int \*\* &x, int numberOfRows);  void BFS(int v, int reach[], int label); //广度优先  void DFS(int v, int reach[], int label); //深度优先  int findMIN(int a[], int length);  bool findPath(int v, int w, int reach[], int label);  int findTheShortestPath(int begin, int end); //迪克斯特拉算法找最短路径  int begin(int i);  int\*\* getA() { return a; } //get方法获取私有成员变量a  int nextVertex(int i);  private:  int noEdge;//用于没有边存在的情形  int n;//顶点数目  int e;//边数  int \*\*a;//二维数组  int \*pos;//记录每个顶点的邻接顶点  };  adjacencyWDigraph.cpp  #include <iostream>  #include <queue>  #include "adjacencyWDigraph.h"  using namespace std;  int adjacencyWDigraph::nextVertex(int i)  {  //返回下一个与i邻接的顶点  if (i<1 || i>n) return 0;  for (int j = pos[i] + 1; j <= n; j++)  if (a[i][j] != noEdge)  {  pos[i] = j;  return j;  }  pos[i] = n + 1;  return 0;  }  void adjacencyWDigraph::makeArray(int\*\* &x, int numberOfRows, int numberOfColumns)  {  x = new int \*[numberOfColumns];  for (int i = 0; i < numberOfRows; i++)  {  x[i] = new int[numberOfColumns];  }  }  void adjacencyWDigraph::deleteArray(int \*\* &x, int numberOfRows)  {  for (int i = 0; i < numberOfRows; i++)  {  delete[] x[i];  }  delete[] x;  x = NULL;  }  adjacencyWDigraph::adjacencyWDigraph(int Vertices, int noEdge)  {  // 构造函数  n = Vertices;  e = 0;  noEdge = noEdge;  makeArray(a, n + 1, n + 1);//程序1-13  pos = new int[n + 1];  for (int i = 1; i <= n; i++)  pos[i] = 0;  //初始化为没有边的图  for (int i = 1; i <= n; i++)  for (int j = 1; j <= n; j++)  a[i][j] = noEdge;  }  bool adjacencyWDigraph::exist(int i, int j) const  {  // 边(i, j)是否存在  if (i<1 || j<1 || i>n || j>n || a[i][j] == noEdge)  return false;  return true;  }  adjacencyWDigraph& adjacencyWDigraph::Add(int i, int j, const int& w)  {  // 如果边(i,j) 不存在，则将该边加入无向图中  if (i<1 || j<1 || i>n || j>n || i == j || a[i][j] != noEdge)  return \*this;  a[i][j] = w;  a[j][i] = w;  e++;  return \*this;  }  adjacencyWDigraph& adjacencyWDigraph::Delete(int i, int j)  {  //删除边(i, j) .  if (i<1 || j<1 || i>n || j>n || a[i][j] == noEdge)  return \*this;  a[i][j] = noEdge;  e--;  return \*this;  }  class OutofBounds : exception  {  public: OutofBounds() {};  };  int adjacencyWDigraph::begin(int i)  {  //返回第一个与i邻接的顶点  if (i<1 || i>n) throw OutofBounds();  for (int j = 1; j <= n; j++)  {  if (a[i][j] != noEdge)  {  pos[i] = j;  return j;  }  }  pos[i] = n + 1;//没有下一个顶点  return 0;  }  void adjacencyWDigraph::BFS(int v, int reach[], int label)  {  //宽度优先搜索  queue<int> Q;  reach[v] = label;  Q.push(v);  cout << v;  while (!Q.empty())  {  //从队列删除一个已经标记的顶点  int w = Q.front();  Q.pop();  //标记所有邻接于顶点w的还没有到达的顶点  for (int i = 1; i <= n; i++)  if (a[w][i] != noEdge && reach[i] != label)  {  Q.push(i);  cout << "," << i;  reach[i] = label;  }//标记已到达该顶点  }  }  int i = 0;  void adjacencyWDigraph::DFS(int v, int reach[], int label)  {  //reach[i]用来标记所有邻接于顶点V的可达到顶点  reach[v] = label;  if (i == 0)  {  cout << v;  }  else  cout << "," << v;  i++;  int u = begin(v);  while (u)  {  //u邻接至v  if (reach[u] != label)  DFS(u, reach, label);  u = nextVertex(v);  }  }  bool adjacencyWDigraph::findPath(int v, int w, int reach[], int label)  {  // 实际搜索v到w的路径，其中v != w.  // 按深度优先方式搜索一条路径  reach[v] = label;  int u = begin(v);  while (u)  {  if (reach[u] != label)  {  if (u == w) return true;  if (findPath(u, w, reach, label))  return true;  }  u = nextVertex(v);  }  return false;  }  int adjacencyWDigraph::findMIN(int a[], int length)  {  int min = a[0];  int positionOfMin = 0;  for (int i = 1; i <= length; i++)  {  if (a[i] <= min)  {  min = a[i];  positionOfMin = i;  }  }  return positionOfMin;  }  int adjacencyWDigraph::findTheShortestPath(int begin, int end)  {  //实现Dijkstra算法  const int MAX = 9999;  const int Label = 999;  int \*Dijkstra = new int[end + 1];  int weightOfPath = 0;  int positionOfMin = 0;  for (int i = 0; i < end + 1; i++)  {  //初始化保存路径数组 使数据全为-1  Dijkstra[i] = MAX;  }  Dijkstra[begin] = 0;  int \*reach = new int[end + 1];  if (findPath(begin, end, reach, Label)) //如果存在从begin 到end 的通路  {  while (Dijkstra[end] != Label)  {  positionOfMin = findMIN(Dijkstra, end);  weightOfPath = Dijkstra[positionOfMin];  Dijkstra[positionOfMin] = Label;  for (int i = 1; i < end + 1; i++)  {  if (Dijkstra[i] != Label && a[positionOfMin][i] != noEdge && a[positionOfMin][i] + weightOfPath < Dijkstra[i])  {  //如果没走过 且 其上一步走到的地方的临边存在 且 它的值加上之前走过的比直接到此处路途更短 就更新一遍表  Dijkstra[i] = a[positionOfMin][i] + weightOfPath;  }  }  }  }  return weightOfPath;  }  源.cpp  #include <iostream>  #include <stdio.h>  #include <queue>  #include "adjacencyWDigraph.h"  using namespace std;  int main()  {  cout << "Input" << endl;  int point;  int edge;  scanf\_s("%d,%d", &point, &edge);  adjacencyWDigraph gragh(point, 0);//初始化一个无边图  int start, end, weight;  for (int i = 1; i <= edge; i++)  {  scanf\_s("%d,%d,%d", &start, &end, &weight);  gragh.Add(start, end, weight);  }  cout << "Output" << endl;  int Label = 9;//定义reach数组中走过的点标记为9  int \*reach = new int[point + 1];  gragh.BFS(1, reach, Label);  cout << endl;  int \*reach2 = new int[point + 1];  gragh.DFS(1, reach2, Label);  cout << endl;  cout << gragh.findTheShortestPath(1, point) << endl;  cout << "End" << endl;  system("pause");  return 0;  } | | | |
| 结论分析与体会：  运行结果：    体会： | | | |