软件工程课程设计（II）实验报告

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 姓名 | 林镕琛 | 学号 | | 181491117 |
| 班级 | 软件1班 | 时间 | | 2020-05-24 |
| 题目 | 无向图的操作 | | | |
| 实验成绩 |  | 指导教师 | 杜岳峰 | |
| 实验内容：   1. 实验程序：   /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  @Author：林镕琛  @Date： 2020-05-23  @Description：   无向图的操作   给定无向图：graph=（V，E）   V={A,B,C,D,E,F},E={(A,C,5),(A,D,1),(A,F,2),(B,D,3),(C,D,4),(C,E,7),(D,E,1),(E,F,2)}  （1）创建一个连通的无向图（边上的权都是正整数，表示某种代价）；  （2）对这个无向图可以实施深度或广度优先遍历；  （3）求这个无向网图的一棵最小生成树（普利姆或者迪杰斯克拉）。  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/    #include<iostream>  #include<stdlib.h>  using namespace std;  #define MAX\_NUM 10 // 设置最多的边和顶点数目  #define INFINITY 9999 // 设置无穷远的边  int visited[MAX\_NUM] = {0}; // 设置访问标志数组      /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  @Funciton :创建无向图结构  @Parm：vertexArray 顶点表  @Parm edgeArray 邻接矩阵 可看作边表  @Parm numVertexes; 顶点数目  @Parm numEdges; 边的数目  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  typedef struct  {   char vertexArray[MAX\_NUM]; // 顶点表   int edgeArray[MAX\_NUM][MAX\_NUM]; // 邻接矩阵 可看作边表   int numVertexes; // 顶点数目   int numEdges; // 边的数目  }graphType;      /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  @Funciton :利用邻接矩阵的方式建立无向图  @Parm: graph 图这个结构体指针  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void graphCreate(graphType \*graph)  {   int leftEdge,rightEdge,weight; // 边的左右结点和权值的定义   printf("请输入顶点数和边数:\n");   scanf("%d%d",&graph->numVertexes,&graph->numEdges);   fflush(stdin);   // 输入顶点   for(int i = 0;i < graph->numVertexes;i++){   printf("\n第%d个顶点",i+1);   scanf("%c",&graph->vertexArray[i]);   getchar();   }   // 矩阵初始化   for (int i = 0;i < graph->numVertexes;i++)   for(int j=0;j<graph->numVertexes;j++)   graph->edgeArray[i][j] = INFINITY;   // 输入边   for(int k = 0;k < graph->numEdges;k++)   {   printf("输入边[i][j]和权w(空格隔开)：");   scanf("%d%d%d",&leftEdge,&rightEdge,&weight);   graph->edgeArray[leftEdge][rightEdge] = weight;   // 创建无向图的步骤   graph->edgeArray[rightEdge][leftEdge]   = graph->edgeArray[leftEdge][rightEdge];   }  }      /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  @Funciton :输出无向图邻接矩阵  @Parm: graph 图这个结构体指针  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void outputGraph(graphType \*graph)  {   int count=0;   for (int i = 0;i < graph->numVertexes;i++){   printf("\t%c",graph->vertexArray[i]);   }   printf("\n");   for (int i = 0;i < graph->numVertexes;i++){   printf("%4c",graph->vertexArray[i]);   for (int j = 0;j < graph->numVertexes;j++){   printf("\t%d",graph->edgeArray[i][j]);   count++;   if(count%graph->numVertexes==0)   printf("\n");   }   }   }      /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  @Funciton: DFSTraverse 对无向图深度优先遍历    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void DFS(graphType graph,int i)  {   visited[i]=1; // 设置被访问的标记   printf("%c->",graph.vertexArray[i]);   for (int j = 0;j < graph.numVertexes;j++)   { // 边存在且未被访问过   if(graph.edgeArray[i][j] != INFINITY&&!visited[j])   DFS(graph,j); // 递归遍历   }  }    void DFSTraverse(graphType graph){   for (int i =0;i < graph.numVertexes;i++)   if (!visited[i])   DFS(graph,i);  }      /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  @Funciton: 普里姆算法实现最小生成树 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void prim(graphType \*graph)  {   int min;   int close[graph->numVertexes]; // 保存相关顶点下标   int lowcost[graph->numVertexes]; // 保存相关顶点间边的权值   // 将v0加入最小生成树集合   lowcost[0] = 0;   close[0] = 0; // 初始化第一个顶点下标为0   for(int i = 1; i < graph->numVertexes; i++)   {   lowcost[i] = graph->edgeArray[0][i];   close[i] = 0;   }   for(int i = 1; i < graph->numVertexes; i++)   {   min = INFINITY;   int j = 1, k = 0;   while(j < graph->numVertexes)   { // 如果权值不为0（说明未加入）,且权值小于min,说明是相邻的边   if(lowcost[j]!=0 && lowcost[j] < min)   {   min = lowcost[j]; // 将当前权值成为最小值   k = j; // 将当前最小值的下标设置为k   }   j++;   }   printf("\t%d<-->%d\n", close[k], k);   lowcost[k] = 0; // 顶点k已经纳入   for(j = 1; j < graph->numVertexes; j++) // 更新最短的边   { // 如果下标为k顶点各边权值小于此前这些顶点未被加入生成树权值   if(lowcost[j]!=0 && graph->edgeArray[k][j] < lowcost[j])   {   lowcost[j] = graph->edgeArray[k][j];   close[j] = k;   }   }   }  }      /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  @Funciton: 主函数调用  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* int main()  {   graphType graph;   graphCreate(&graph);   printf("邻接矩阵数据如下：\n");   outputGraph(&graph);   printf("深度优先遍历：");   DFSTraverse(graph);   printf("\n普里姆算法求最小生成树：\n");   prim(&graph);   return 0;  }   1. 实验运行结果： | | | | |