数据结构实验报告

实验名称： 实验四——图的实现

学生姓名： 郑宇博

班 级： 2017211128

班内序号： 07

学 号： 2017210103

日 期： 2018年6月11日

1．实验要求

实验目的：

本实验为可选实验，用于提高同学们使用数据结构解决实际问题的能力。通过选择下面3个题目之一进行实现，掌握如下内容：

* 线性表的深度应用
* 了解图的思想和算法实现
* 矩阵和相关算法在BMP图像中的应用
* 进一步提高编程能力

2. 程序分析

2.1 存储结构

顺序表；栈(递归)

2.2 关键算法分析

1、关键算法：

①．深度优先遍历：

1. 从图中任意顶点v出发，标记v为已访问；
2. 访问v中的第一个未访问的邻接点w，标记w为已访问；
3. 访问w的第一个未访问的邻接点u，标记u，为已访问；
4. 若当前顶点的所有邻接点都被访问过，则回溯，从上一级顶点的下一个未访问过的顶点开始深度优先遍历。

②．广度优先遍历

1. 初始化队列Q。
2. 访问顶点v，visited[v]=true.
3. While（队列不为空）
   1. v = 队头元素出队；
   2. 访问队头元素的所有未访问的邻接点。

③．Prime算法

1. 初始化辅助数据结构
2. 选择辅助数组lowcost中的最小值，即U->V-U的最小权值边集合中的最小权值
3. 迭代的更新辅助数据结构adjvex和lowcost
4. 重复B、C知道集合{V-U}为空

④．Kruskal算法

1. 输入为n个结点m条边的图，每条边都带有一个权值w
2. 对所有边按照权值大小进行排序，排序结果为{e1,e2,...,em}
3. 初始化MergeQuery数据结构mq，结点数量为n
4. m = {};
   1. for i=1 to m, do
      1. A = VSET(A.X);
      2. B = VEST(A.Y);
      3. if A != B
         1. mq.merge(A, B, ei);
         2. 将边A加入m;
      4. endIf
   2. endFor
   3. return m;

⑤．迪杰斯特拉算法

//  初始化,设从0开始

for i=[0,n)

    dist[i] = map[0][i]

visit[0] = true;

for i=[1,n)

    //  寻找最短路径(s,t)，同时把t加入S集合

    min = MAX\_VALUE

    for j=[0,n)

        if !visit[j] && dist[j]<min

            min = dist[j]//记录最小值和最小值的下标

min\_j = j

    visit[j] = true

    //  松弛边(t,v)，其中v为顶点

    for k=[0,n)

        if !visit[k] && dist[k]>dist[j]+tab[j][k]

            dist[k] = dist[j]+tab[j][k]

⑥．非递归深度优先遍历

1. 查找v的邻接点：遍历邻接矩阵，寻找arc[v][i]!=0(v和i邻接)且bVisited[i]==true的结点i；
2. 若找到i则进栈。
3. 若找不到i则出栈

2、代码详细分析：

①． void Prim(MGraph G){

for (int i = 0; i < G.vNum; i++){

adjvex[i] = 0; lowcost[i] = G.arcs[0][i]; //存储所有到v0的边

}

lowcost[0] = 0; //初始化U={V0}

for (int i = 1; i < G.vNum; i++){

int k = mininum(G, lowcost); //找下一个边权值最小的邻接点

cout << 'V' << adjvex[k] << "->V" << k << endl;

result += lowcost[k];

lowcost[k] = 0; //U=U+{Vk}

for (int j = 0; j < G.vNum; j++) { //更新辅助数组

if (lowcost[j] != 0 && G.arcs[k][j] < lowcost[j]){

lowcost[j] = G.arcs[k][j];

adjvex[j] = k; //增加该路径

}

}

}

cout << result; //输出最短路径长度

}

②．克鲁斯卡尔算法

while (k < n - 1){

int m = EdgeList[j].fromV, l = EdgeList[j].endV; //当前最短边的起点和终点

int sn1 = vset[m]; //起点颜色

int sn2 = vset[l]; //终点颜色

if (sn1 != sn2) { //颜色不等

cout << 'V' << m << "->V" << l << endl; //输出起点和终点

k++;

result += EdgeList[j].weight;

for (int i = 0; i < n; i++){

if (vset[i] == sn2){ //给所有和终点颜色一样的点涂色为起点颜色

vset[i] = sn1;

}

}

}

j++;

3、 计算关键算法的时间、空间复杂度

A. 时间复杂度：

①．深度优先遍历：O(n2)

②．广度优先遍历：O(n2)

③. 普里姆算法：O(n2)

④．克鲁斯卡尔算法：max{O(eloge),O(en)}

⑤．最短路径迪杰斯特拉算法：O(n2)

⑥．非递归深度优先遍历：O(n2)

B. 空间复杂度：

①．深度优先遍历：O(n)

②．广度优先遍历：O(n)

2.3 其他

思考题：

如果边数远小于n^2,对此可以用堆这种数据结构进行优化，取出最短路径的复杂度降为O(1)；每次调整的复杂度降为O（elogn）；e为该点的边数，所以复杂度降为O((m+n)logn)。

实现

1. 将与源点相连的点加入堆，并调整堆。

2. 选出堆顶元素u（即代价最小的元素），从堆中删除，并对堆进行调整。

3. 处理与u相邻的，未被访问过的，满足三角不等式的顶点

1):若该点在堆里，更新距离，并调整该元素在堆中的位置。

2):若该点不在堆里，加入堆，更新堆。

4. 若取到的u为终点，结束算法；否则重复步骤2、3。

3. 程序运行结果

* 1. 测试主函数流程：流程图如图2所示

开始

等待用户输入节点数和邻接数组

输出所有结果

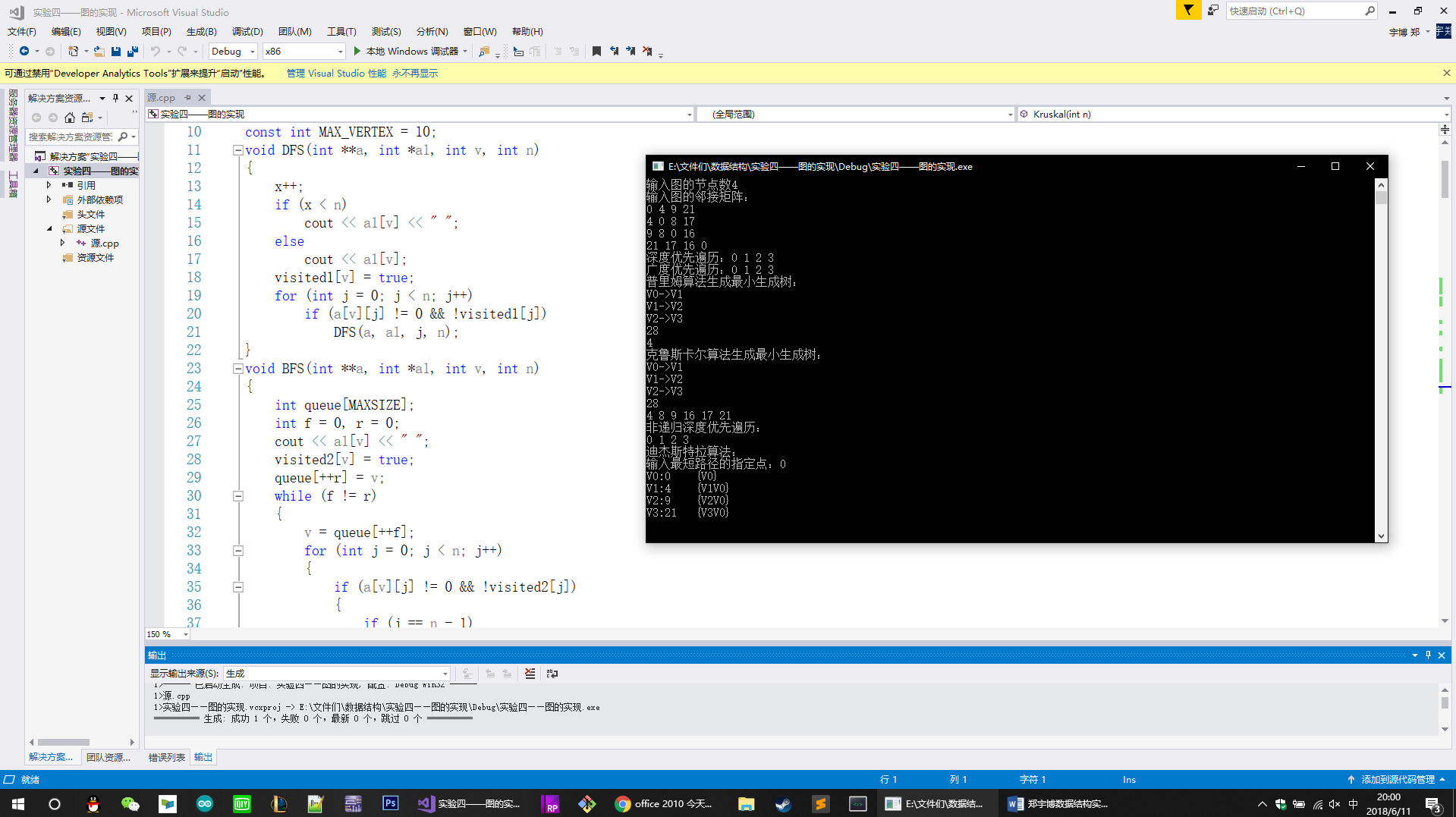
等待用户输入最短路径长度的指定终点

输出到指定最短路径

结束

图2 流程图示意图

* 1. 测试结果：



4. 总结

1. 调试时出现的问题及解决的方法

邻接数组的表示方式有很多种，对代码有不同的影响，要及时注意区分

要避免数组越界

1. 心得体会

图的实现训练了我对于二维数组的直观感觉，可以加强我对于通信中“组播”技术和计算机网络中的路由协议的最短路径优先算法的理解，有助于我今后对于通信网络的学习

1. 下一步的改进

尝试进一步改进迪杰斯特拉算法，使用堆空间。