

《数据结构》上机报告

2018 年 11 月 26 日

姓名：刘思源 学号：1651390 班级：电子三班 得分：_____

实验题目	图
问题描述	图是一种描述多对多关系的数据结构，图中的数据元素称作顶点，具有关系的两个顶点形成的一个二元组称作边或弧，顶点的集合 V 和关系的集合 R 构成了图，记作 $G=(V,R)$ 。图又分成有向图，无向图，有向网，无向网。图的常用存储结构有邻接矩阵、邻接表、十字链表、邻接多重表。
基本要求	<p>1. (p1) 假设要在 n 个城市之间建立通信联络网，至少需要 $n-1$ 条线路，而建立每条线路需要付出不同的经济代价。现给出一个无向图，列出了建造每一条线路的成本，求使得所有城市均连通的最小代价。</p> <p>2. (p2) 从房子的左下角开始，按照节点编号递增顺序排列，输出所有可以一笔画完的顺序，要求一条边不能画</p> <p>3. (p3) 假如给你一个社交网络图，请你对每个节点计算符合“六度空间”理论的结点占结点总数的百分比。</p>
	<div>已完成基本内容（序号）：1</div>
选做要求	
	<div>已完成选做内容（序号）2, 3</div>
数据结构设计	<pre> typedef int Status; typedef char ElemType; typedef struct LNode{ ElemType data; int num; int weight; LNode *next; }VextexNode, *AdjList; typedef struct MGraph { char vexs[MAXSIZE]; int arcs[MAXSIZE][MAXSIZE]; int vexnum, arcnum; AdjList List[MAXSIZE]; }MGraph; </pre>

功能(函数)说明	<p>1、Prim 算法</p> <p>Prim 算法求最小生成树的时候和边数无关，和顶点树有关，所以适合求解稠密网的最小生成树。</p> <p>Prim 算法的步骤包括：</p> <ol style="list-style-type: none"> 1. 将一个图分为两部分，一部分归为点集 U，一部分归为点集 V，U 的初始集合为 {V1}，V 的初始集合为 {ALL-V1} 2. 针对 U 开始找 U 中各节点的所有关联的边的权值最小的那个，然后将关联的节点 Vi 加入到 U 中，并且从 V 中删除（注意不能形成环）。 3. 递归执行步骤 2，直到 V 中的集合为空。 4. U 中所有节点构成的树就是最小生成树。 <p>讨论到该题中，首先定义一个数组 U 来用作全集的表示。首先将每个节点初始化到该全集数组中。接下来进行最小生成树的建立，对已经从全集中去除的点进行遍历，如果找到了可连接的点，证明可以生成最小生成树，对所有的边进行比较，选取最小的边存储，直到所有的点都被寻找过；或者找不到与某一个点相连接的点。</p> <pre> struct temp { int start; int end; int weight; }; int Prim(MGraph G, int begin) { temp *U = new temp[G.vexnum]; int j, sum=0; for (j = 0; j < G.vexnum; j++) { if (j != begin - 1) { U[j].start = begin - 1; U[j].end = j; U[j].weight = G.arc[begin - 1][j]; } } U[begin - 1].weight = -1; for (j = 1; j < G.vexnum; j++) { int min = MAXSIZE; int k; int index=-1; for (k = 0; k < G.vexnum; k++) { if (U[k].weight != -1) { </pre>
----------	---

```

        if (U[k].weight < min) {
            min = U[k].weight;
            index = k;
        }
    }

    if (index == -1) {
        return -1;
    }

    U[index].weight = -1;

    sum += G.arc[U[index].start][U[index].end];

    for (k = 0; k < G.vexnum; k++) {
        if (G.arc[U[index].end][k] < U[k].weight) {
            U[k].weight = G.arc[U[index].end][k];
            U[k].start = U[index].end;
            U[k].end = k;
        }
    }
}

return sum;
}
}

```

2、一笔画算法

一笔画算法实际上就是深度搜索优先遍历的变形。我们首先将邻接矩阵初始化，使用一个 string 变量储存一笔画的结果。使用一个 int 值表示寻找的深度（即一笔画的边数）。当这个边数等于 8 时，证明一笔画已经完成。接下来从第一个点开始遍历，如果找到了与之相邻的点，将这条边暂时去除（置为 0），然后将深度+1，字符串存贮结果，进行下一递归的 DFS。直到找不到连接的边；或者已经达到 8，输出结果，返回上一层遍历。

```

void DFS(MGraph G, int v, int depth, string s)
{
    if (depth >= 8) {
        cout << s << endl;
    }

    int i;
    for (i = 0; i < G.vexnum; i++) {
        if (G.arcs[v][i] == 1) {
            G.arcs[v][i] = 0;
            G.arcs[i][v] = 0;

```

```

        DFS(G, i, depth + 1, s + char(i + 49));
        G.arcs[v][i] = 1;
        G.arcs[i][v] = 1;
    }

}

return;
}}

```

3、六度空间算法

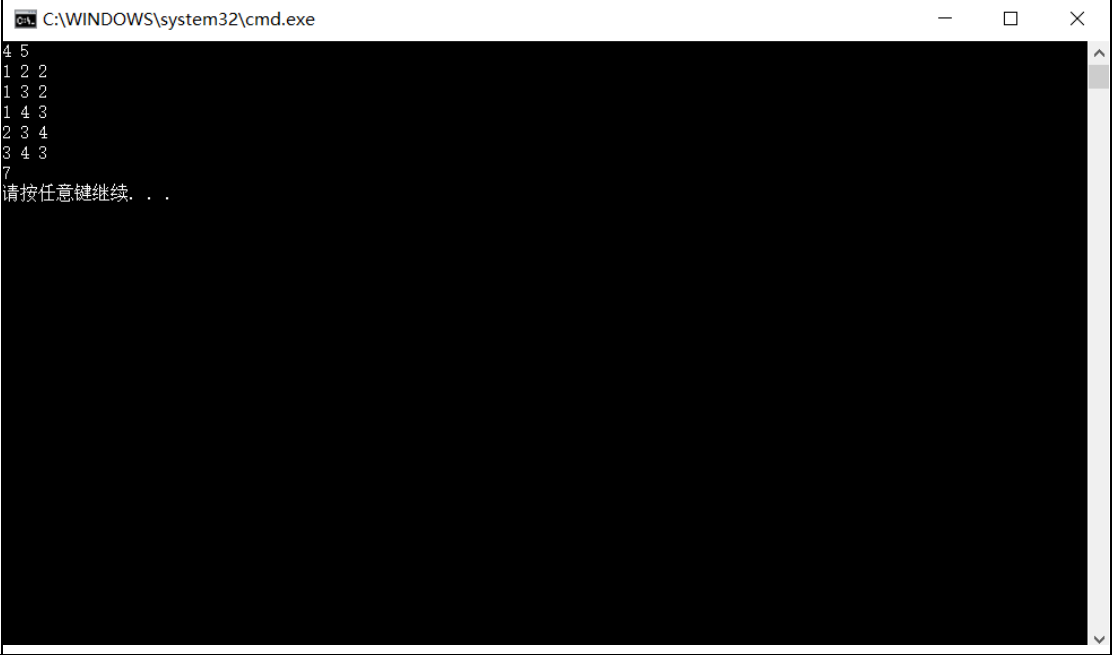
六度空间算法实际上是广度优先搜索算法的变形。通过搜索与某点联系的所有层次稍低于等于 6 的点的个数来达到要求。我们使用一个数组来模拟队列。用一个 visit 的数组来判断当前点是否遍历过。当队列不为空时，从队列中取出一个元素，层次遍历与之连接的点，将这些点入队，并进行计数，重复改操作直到队列为空。

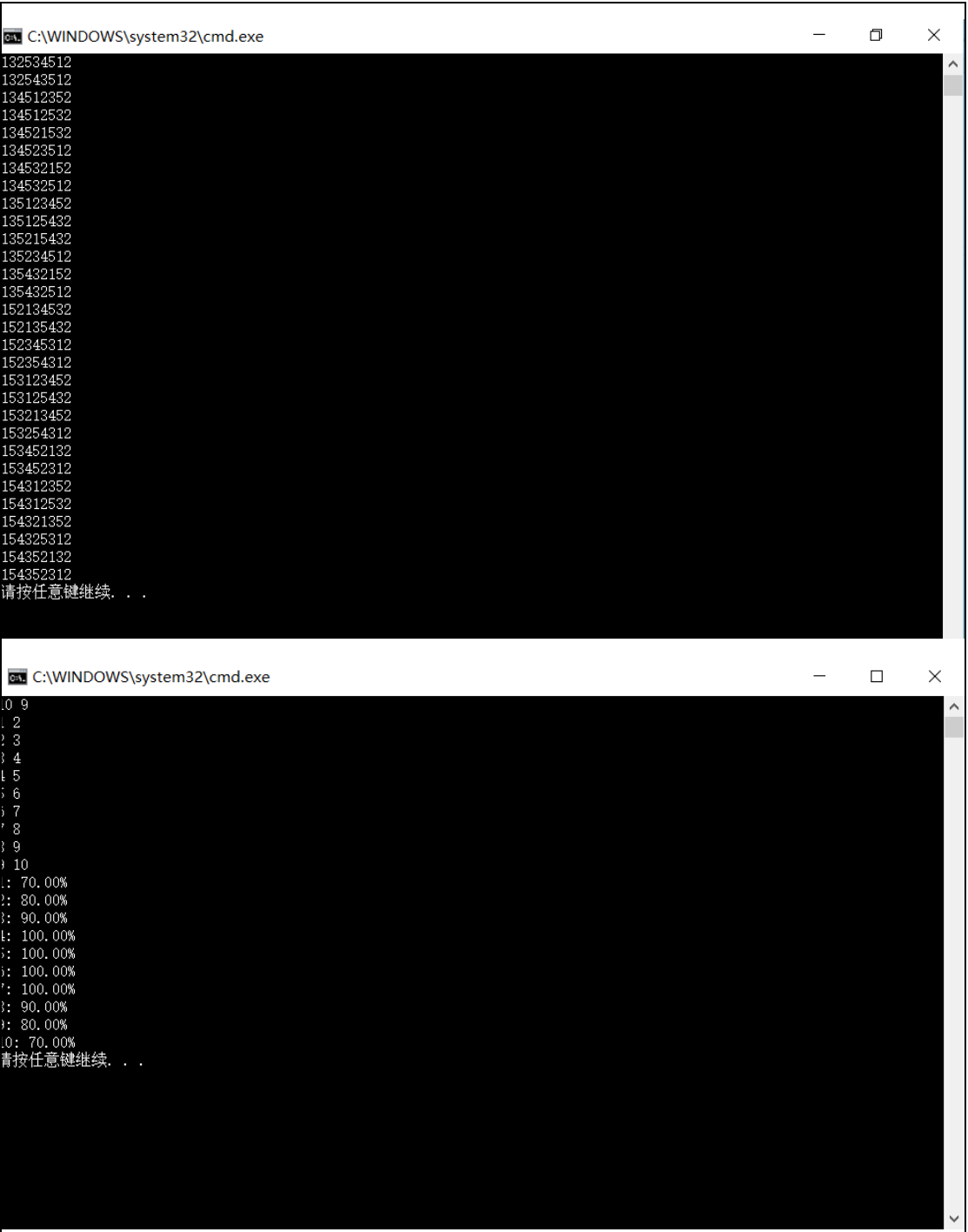
```

int BFS(MGraph G, int i)
{
    int q[MAXSIZE + 3], tail, v, j;
    bool *visit=new bool[G.vexnum];

    for (j = 0; j <= G.vexnum; j++) {
        visit[j] = false;
    }
    visit[i] = true;
    int front = -1, rear = -1;
    int count = 1;
    int level = 0;
    int last = i;
    q[++rear] = i;
    while (front < rear) {
        v = q[++front];
        for (j = 1; j <= G.vexnum; j++) {
            if (!visit[j] && G.arcs[v][j] == 1) {
                q[++rear] = j;
                visit[j] = 1;
                count++;
                tail = j;
            }
        }
        if (v == last) {
            level++;
            last = tail;
        }
        if (DU <= level) {

```

	<pre> return count; } } free(visit); return count; }</pre>
开发环境	Visual studio 2017
调试分析	

	
心得体会	<p>1、六度空间算法容易 Runtime Error。本文一开始使用邻接矩阵进行操作，但最后一项数据总是 Runtime Error。分析应该是数据量过大，但实际上边的数量并不是有申请的那么多，造成了空间的浪费。所以改为使用邻接表进行操作，可以大大减少数据空间和分配时间。</p> <p>2、一笔画算法的时候，为了避免答案不全或者是答案遗漏的情况，每次递归得到的结果只在形参中进行改变，如果在函数调用外进行，改变将直接影响到后续的每一次遍历和递归。</p>