《数据结构》上机报告

2018 年 12 月 13 日

姓名： 赵得泽 学号： 1753642 班级： 电子 2 班 得分：

<table>
<tr><td>实验题目</td><td colspan="2">图</td></tr>
<tr><td>问题描述</td><td colspan="2">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。图的基本操作包括图的创建、销毁、添加顶点、删除顶点、插入边、删除边、图的遍历。</td></tr>
<tr><td rowspan="2">基本要求</td><td colspan="2">1. 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。</td></tr>
<tr><td>已完成基本内容（序号）：</td><td>1</td></tr>
<tr><td rowspan="2">选做要求</td><td colspan="2">1. 圣诞节马上到了，我们用一笔画画出圣诞老人的房子吧。现在的问题是，一共有多少种画法呢？请同学们写一个程序，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，要求一条边不能画两次。
2. 六度空间理论又称小世界理论。理论通俗地解释为：“你和世界上任何一个陌生人之间所间隔的人不会超过 6 个人，也就是说，最多通过五个人你就能够认识任何一个陌生人。”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。</td></tr>
<tr><td>已完成选做内容（序号）</td><td>1,2</td></tr>
<tr><td>数据结构设计</td><td colspan="2">

```
typedef  int AdjMatrix[MAX_VERTEX_NUM][MAX_VERTEX_NUM];
//邻接矩阵类型
typedef struct {
    VertexType vexs[MAX_VERTEX_NUM];      //顶点表
    AdjMatrix arcs;     //邻接矩阵
    int vexnum, arcnum;  //图的顶点数和边/弧数
    int Graphkind;
}  MGraph;
typedef struct {
    VertexType adjvex;
    int lowcost;
}closedge[MAX_VERTEX_NUM];
```

本次实验主要用到的数据结构是图，应用主要是最小生成树（上述数据结构即求最小生成树的数据结构）、深度优先和广度优先遍历。数据结构皆以邻</td></tr>
</table>

	接矩阵为主，这样可以简单快捷的知道各个顶点之间的关系，从而有利于问题的求解。在最小生成树的求解过程中又增加了一个结构体，最小代价边辅助数组，成员包括顶点及最小代价边权值。后面两种方法主要是邻接矩阵，其构造方法在前面的实验中已经交代过，在此不做赘述。

功能(函数)说明	/***
	函数功能：定位顶点
	函数说明：通过遍历查找与目标定点相同的顶点的编号返回即可。
	***/
	int LocateVertex(MGraph &G, VertexType v)
	{
	int i;
	for (i = 0; i < G.vexnum; i++)
	if (v == G.vexs[i])
	return i;
	return -1;
	}
	/***
	函数功能：创建图
	函数说明：创建图有邻接表法和邻接矩阵法；邻接表法就是用一个结构数组即顺序表存储顶点，称为顶点表；创建边结点表存储在与顶点相关联的一条边（弧）的另一个顶点，以链表的方式进行头插法建立。邻接矩阵法就是用一个方阵来存储两个顶点是否相邻的信息，若i,j顶点相邻，就将矩阵的i行j列元素置1，（或者为权值，若是无向图，j行i列也做此操作）在本题目中，用的是邻接矩阵创建的有向图。
	***/
	void CreateGraph(MGraph &G1)
	{
	int i, j, k;
	VertexType v1, v2;
	int w;
	int n, m;
	cin >> n >> m;
	G1.vexnum = n;
	G1.arcnum = m;
	for (i = 0; i < G1.vexnum; i++)
	G1.vexs[i] = i + 1;
	for (i = 0; i < G1.vexnum; i++)
	for (j = 0; j < G1.vexnum; j++)
	G1.arcs[i][j] = INFINITY;
	for (k = 0; k < G1.arcnum; k++)
	{
	cin >> v1 >> v2 >> w;
	i = LocateVertex(G1, v1);
	j = LocateVertex(G1, v2);

```
        G1.arcs[i][j] = w;
        G1.arcs[j][i] = w;
    }
}
/*************************************************************
函数功能：求u到v的最小代价的边
函数说明：通过循环，在没有标记过的点中查找，一条最小代价的边，
并且返回下一个顶点的编号。
*************************************************************/
int min(closedge &cls, MGraph &G)
{
    int min = INFINITY, k = 0;
    for (int i = 0; i < G.vexnum; i++)
        if (cls[i].lowcost != 0 && cls[i].lowcost < min)
        {
            min = cls[i].lowcost;
            k = i;
        }
    return k;
}
/*************************************************************
函数功能：Prim算法求最小生成树
函数说明：具体说明见下方代码注释区域
*************************************************************/
int  MST_PRIM(MGraph &G, VertexType u)
{
    closedge cls;
    int k, j, sum = 0, arc_num = 0;
    k = LocateVertex(G, u);//定位顶点u的位置
    for (j = 0; j < G.vexnum; j++)
    {
        if (j != k)
            cls[j] = { u,G.arcs[k][j] };//辅助数组初始化为依附顶点u
的边权值
    }
    cls[k].lowcost = 0;//顶点u并入U={u};
    int i;
    for (i = 1; i < G.vexnum; i++)//选择其余G.vexnum-1个顶点
    {
        k = min(cls, G);//求出最小生成树中u顶点的下一个顶点：第k个
顶点
        sum += cls[k].lowcost;//计算最短路径长度
        arc_num++;//计数已经求过的边
        cls[k].lowcost = 0;//第k个顶点并入U集合
```

```
        for (j = 0; j < G.vexnum; j++)//第k个顶点并入集合U后，以k顶点为新的顶点，修改到其余各顶点的最小边
            if (G.arcs[k][j] < cls[j].lowcost)
                cls[j] = { G.vexs[k],G.arcs[k][j] };
    }
    if (arc_num == G.vexnum - 1)
        return sum;//最后边的个数等于顶点数-1，说明有最短路径，该图连通
    else
        return  -1;//反之，则该图不连通
}
/*****************************************************************
函数功能：深度优先递归求欧拉路径
函数说明：因为已经给定了图的具体结构，所以用邻接矩阵存储该无向图，初始化的时候，如果相邻两点之间有一条边，则相应的位置置1。共八条边，所以深度为8的时候退出一层递归，输出欧拉路径即可。在具体寻找的循环中，如果某个位置值为1，则将该位置及相应的对称位置置零，表示已经访问过，再进行递归；当一层递归退出之后，再回到上一层递归，同时将原先置0的地方恢复，以便下一次循环的时候进行查找。
*****************************************************************/
int AdjMatrix[5][5] = {
            0, 1, 1, 0, 1,
            1, 0, 1, 0, 1,
            1, 1, 0, 1, 1,
            0, 0, 1, 0, 1,
            1, 1, 1, 1, 0
};   //邻接矩阵
char s[MAX_VERTEX_NUM];//存储欧拉路径
void  DFS(int x, int k, char s[])
{
    //x:起始顶点；
    //k：遍历的深度；
    //s[]:存储路径顶点
    s[k] = x + 1 + '0';
    if (k >= 8)
    {
        cout << s << endl;
        return;
    }
    int y = 0;
    for (y = 0; y < 5; y++)
        if (AdjMatrix[x][y] == 1)
        {
            AdjMatrix[x][y] = 0;
```

```
            AdjMatrix[y][x] = 0;
            DFS(y, k + 1, s);
            AdjMatrix[x][y] = 1;
            AdjMatrix[y][x] = 1;
        }
}
/*****************************************************************
函数功能：广度优先求每个顶点最近的六层顶点
函数说明：具体说明见下函数注释区域
*****************************************************************/
float  BFSTraverse(int k, int AdjMatrix[][MAX_VERTEX_NUM])
{
    int v, depth = 0;//广度优先的层数
    float count = 1;//顶点个数
    int last_pre = k;//上一层的最后一个顶点
    int last_cur = 0;//当前层的最后一个顶点
    queue<int>Queue;
    int w = 1, u = 1;
    vis[k] = 1;//起始顶点入队
    Queue.push(k);//将起始顶点入队
    while (!Queue.empty())//若队列不空一直进行遍历操作
    {
        v = Queue.front();
        Queue.pop();//队头出队
        for (w = 1; w <= vexnum; w++)//访问与顶点v相邻的顶点
            if (!vis[w] && AdjMatrix[v][w] == 1)//对没访问过的顶点
进行访问，同时将与v相邻的顶点入队
            {
                count++;
                vis[w] = 1;
                Queue.push(w);
                last_cur = w;//记录v的下一层的最后一个顶点
            }
        if (last_pre == v)
        {
            depth++;
            last_pre = last_cur;
        }//如果上一层的最后一个顶点是v，就让上一层最后一个顶点赋值
为当前层
        //的最后一个顶点，即说明当前层遍历结束。
        if (depth == 6)//6层遍历结束，就退出循环。
            break;
    }
    return count;//返回与顶点k相距最近的6层顶点的总数}
```

开发环境	Win10 ， vs2017,C++高级程序语言设计
调试分析	Problem1： 4 5 1 2 2 1 3 2 1 4 3 2 3 4 3 4 3 7 4 5 1 2 1 1 3 1 1 4 2 2 3 4 3 4 3 4 Problem2： 134532512 135123452 135125432 135215432 135234512 135432152 135432512 152134532 152135432 152345312 152354312 153123452 153125432 153213452 153254312 153452132 153452312 154312352 154312532 154321352 154325312 154352132 154352312 Problem3： 10 9 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 10 1: 70.00% 2: 80.00% 3: 90.00% 4: 100.00% 5: 100.00% 6: 100.00% 7: 100.00% 8: 90.00% 9: 80.00% 10: 70.00%

心得体会	本次实验是图的综合运用，其中包括最小代价生成树（MST）、欧拉回路的寻找、六度空间这几个问题。通过对这类问题的求解，让我对广度优先、深度优先算法也有了更进一步的认识和体会，在解决这些问题的过程当中，还充分**认识到队列、栈等数据结构和图交汇在一起的综合应用**，以及他们解决问题的时间效应和空间效应（六度空间）。 在解决**最小生成树问题**的过程中，我觉得其中最最核心的思想便是每次对起始顶点到相邻顶点之间的权值更新为最小值，并且再求得这些最小值中的最小值，然后将该最小值对应的顶点并入已经寻找过的顶点集合，这样每次寻找最短路径求得最终的最小生成树，也即贪心算法。**一笔画问题，**其核心思想就是，深度优先进行递归，利用循环，找出所有的**欧拉路径**；**六度空间问题**，其核心思想是利用广度优先遍历，找出**离源点最近的六层顶点**，然后计数即可，当然其中的难点是，如何定位某一层的最后一个顶点（也就是**在出队过程中，如何判断某一层遍历结束**），在此我的思想是设置变量每次都记录每一层的最后一个顶点，然后在出队的过程中若是上一层最后一个顶点恰好等于我记录的上层最后一个节点，就让它层数加一，重新对 last_pre 赋值。 总之，这次实验的应用需要自己思考理解问题中蕴含的算法原理，再利用我们学习过的知识进行解决。