算法导论实验报告：图论算法

一、实验内容

实验 1：实现求有向图的强连通分量的算法。有向图的顶点数 N 的取值分别为： 9、27、81、243、729， 弧 的数目为 Nlog_3N， 随机生成 Nlog_3N 条弧，统计算法所需运行时间 ，画出时间曲线。

实验 2：实现求所有点对最短路径的 Floyd-Warshall 算法。有向图的顶点数 N 的取值分别为: 9、27、81、243、729 ， 弧的数目为 Nlog_3N， 随机生成 Nlog_3N 条弧，统计算法所需运行时间 ，画出时间曲线。

二、实验环境

编译环境：Microsoft Visual Studio 2015

机器内存：8G

时钟主频：2.5GHz

三、实验要求

a)两个实验分别建立 project1，project2 文件夹，每个文件夹分别包含 3 个文件夹：

Input 文件夹： 存放输入的图的数据

Source 文件夹：源程序

Output 文件夹：输出信息

b)input

实验一：为每种输入规模分别建立一个子文件夹，实验数据规模从小到大分别为 size1,size2,size3,size4,size5,随机生成的有向图信息分别存放到对应数据规模文件夹里面的 input.txt 文件，每行存放一对节点 i,j 序号(数字表示)，表示存在一条节点 i 指向节点 j 的边。分别读取这五个规模的图数据进行求解最强连通分量的实验.

实验二：同实验一为每种输入规模分别建立一个子文件夹,随机生成边上的权值，权值范围统一在（-10,30）之间。随机生成的有向图分别存放到对应数据规模文件夹里面的 input.txt 文件，每行存放一对节点 i,j 序号，表示这两个节点之间存在着一条边相连,以及边的权值"ω" $_{ij}$。分别读取这五个规模的图数据进行求解所有点对最短路径的实验.

c)output：

为每种数据规模建立一个子文件夹，分别为 size1,size2,size3,size4,size5 其输出结果数据导出到其对应子文件下面

output1.txt ：输出对应规模图中存在的所有连通分量

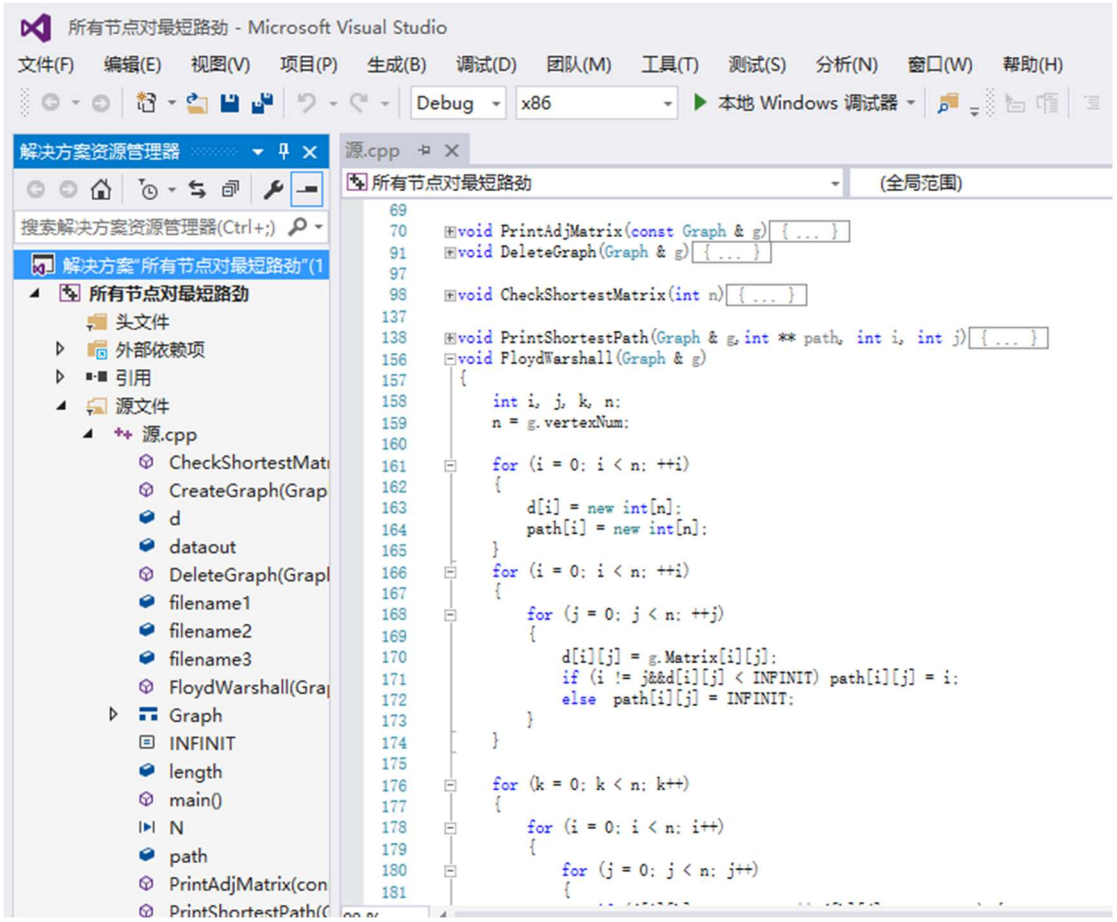
time1.txt：输出测试求解出每个连通分量所花费的时间。

第二个实验输出结果同样是导入到相同的对应子文件夹下面

output2.txt :输出对应规模图中所有点对之间的最短路径包含的节点序列及路径长。

time2.txt: 输出测试程序求解出对应规模图所有点对最短路径所消耗的时间。

四、实验过程截图



五、实验过程

1. 求强连通分量的 scc 算法实验

1) 采用邻接链表来存储图 G。

```
//边结构
struct Edge
{
    int start;
    int end;
    Edge *next;//同一个起点开始的下一条边
    int type;//边的类型
    Edge(int s, int e) :start(s), end(e), next(NULL) {}
};
//顶点结构
struct Vertex
{
    int id;
    Edge *head;//以该顶点为起点的下一条边
    int color;
    Vertex *p;//指向遍历结果的父结点
    int d, f;//第一次被发现的时间和结束检查的时间
    Vertex() :head(NULL), color(WHITE), p(NULL), d(0x7fffffff), id(0) {}
};
```

2) 图的深度优先遍历算法 DFS，使用 flag 标志判断是第一次

```
//深度优先搜索
void DFS(Graph *G){
    int i;
    for (i = 1; i <= N; i++){
        G->V[i].id = i;
        G->V[i].color = WHITE;
        G->V[i].p = NULL;
    }
    T = 0;
    //依次检索V中的顶点，发现白色顶点时，调用DFS_Visit访问该顶点
    for (i = 1; i <= N; i++){
        int j;//第一次是以正常顺序按点1->2->3->.....->8的顺序调用DFS_Visit函数
        if (flag == 0) j = i;
        //第二次是以f从大到小的顺序，这个顺序在第一次dfs次保存于Sort数组中
        else j = Sort[i];
        //发现白色顶点时，调用DFS_Visit访问该顶点
        if (G->V[j].color == WHITE){
            if (flag)  dataout << "强连通分量为：( ";
            DFS_Visit(G, &G->V[j]);
            //flag == 1时，第二次调用DFS，此时每次调用DFS_Visit 就会输出一个强连通分量
            if (flag)  dataout <<")"<< endl;
        }
    }
}
```

3) SCC 算法

```
void Strongly_Connected_Component(Graph *G)
{
        DFS(G); //第一次DFS，计算每个顶点的f
        Graph *G2 = Reverse(G); //转置，计算GT
    flag = 1; //第一次的DFS和第二次的DFS不同，用flag区分
        DFS(G2); //第二次的DFS，按照f从大到小的顺序调用

}
```

2. 求所有节点对最短路径的 floyd-warshall 算法实验

1) Floyd-warshall 算法

```
void FloydWarshall(Graph & g){
    int i, j, k, n;
    n = g.vertexNum;
    for (i = 0; i < n; ++i){
        d[i] = new int[n];
        path[i] = new int[n];
    }
    for (i = 0; i < n; ++i)     {
        for (j = 0; j < n; ++j){
            d[i][j] = g.Matrix[i][j];
            if (i != j&&d[i][j] < INFINIT) path[i][j] = i;   //矩阵d和path初始化
            else  path[i][j] = INFINIT;
        }
    }
    for (k = 0; k < n; k++)    {
        for (i = 0; i < n; i++){
            for (j = 0; j < n; j++)    {
                if (d[i][k] == INFINIT || d[k][j] == INFINIT) {
                    /*若d[i][k]或d[k][j]等于无穷，则不能用d[i][k] + d[k][j] 做比较运
算 ，否则可能会出现 负数+infinit<infinit， infinit设定的是一个很大的数*/
                    //有不可达的点对，直接跳过
                }
                else if (d[i][k] + d[k][j] < d[i][j]){
                    d[i][j] = d[i][k] + d[k][j];
                    path[i][j] = path[k][j];
                }
            }
        }
    }
}
```

2) 图中负环路的检查

```
void CheckShortestMatrix(int n){
    int i, j, k;
    for (k = 0; k < n; k++){
        for (i = 0; i < n; i++){
            for (j = 0; j < n; j++){
                if (d[i][k] == INFINIT || d[k][j] == INFINIT)
                else   if (d[i][k] + d[k][j] < d[i][j]){
                    cout << "error:graph中存在负环路" << endl;   /*再对矩阵d进行循
环，若还有更短的路径,则说明存在负环路*/
                    exit(1);
                }
}}}}
```

3) 所有节点对的最短路径的输出

```
void PrintShortestPath(Graph & g,int ** path, int i, int j){
    if (i == j)  dataout << i << " ";
    else if (path[i][j] == INFINIT) {
            dataout << "INF"; return;
    }
    else {
    int k = path[i][j];
    length+=g.Matrix[k][j];
    PrintShortestPath(g,path, i,k );
    dataout << j<<" ";
    }
}
```

六、实验结果与分析

顶点数V	9	27	81	243	729
边数目E	18	81	324	1215	4374
V+E	27	108	405	1458	5103
SCC用时	2610	4942	8275	22445	44199
FW用时	251	1310	20935	683993	16384204

1、 求连通分量的 SCC 算法

实验结果：

以 N=9 为例：

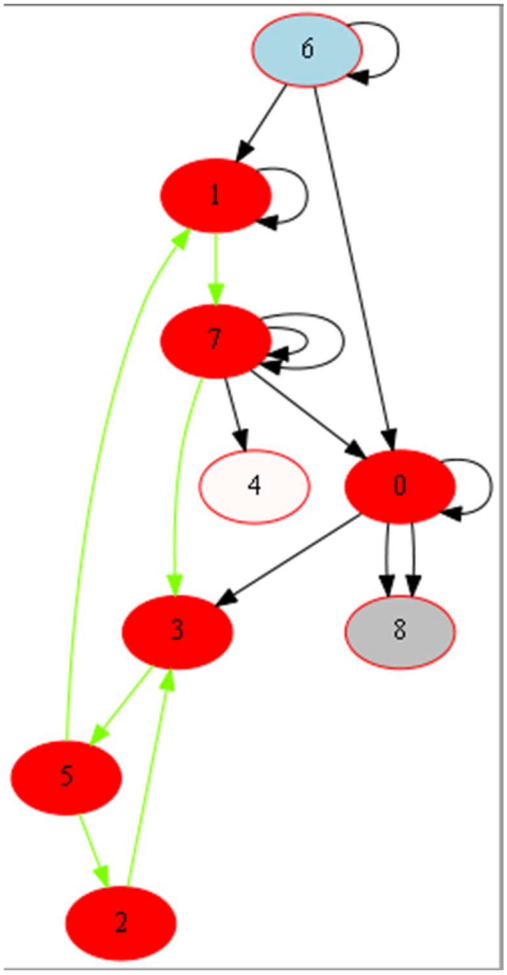
强连通分量为：(6)

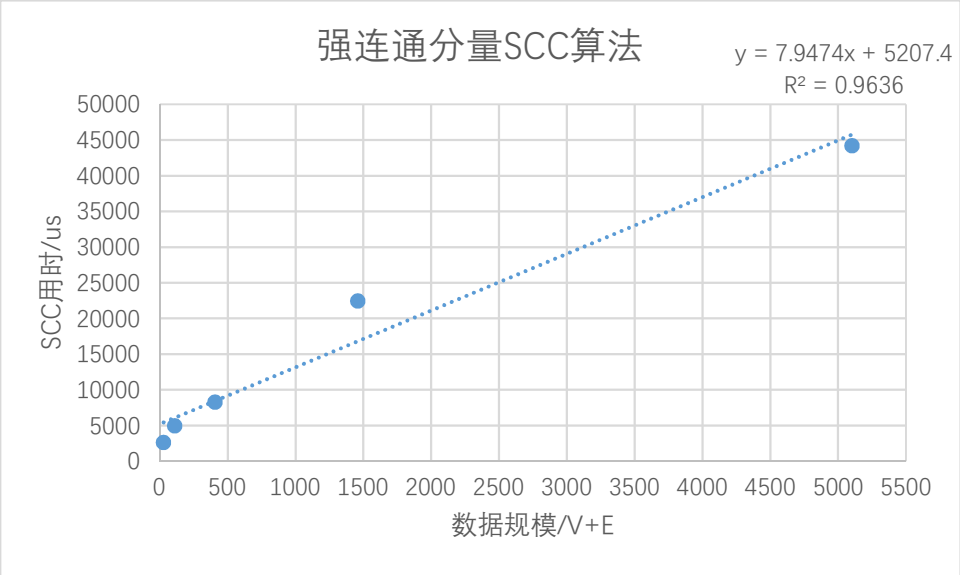
强连通分量为：(1 5 3 2 7 0)

强连通分量为：(4)

强连通分量为：(8)

SCC 算法的时间复杂度为θ(V+E)。利用上表做出下图，可以看出 SC 算法用基本与 V+E 成线性关系。



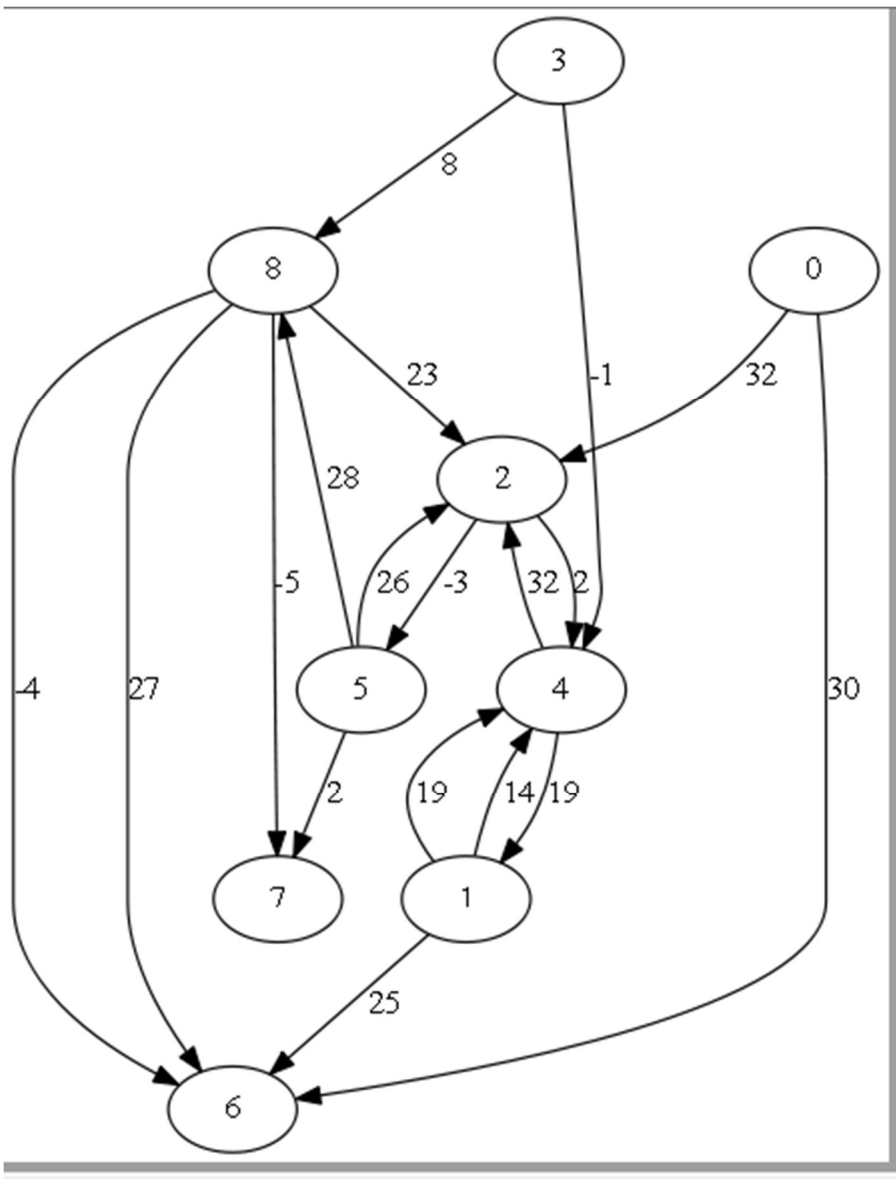


2、 求所有节点对的 Floyd-warshall 算法

实验结果：

以 N=9 为例：

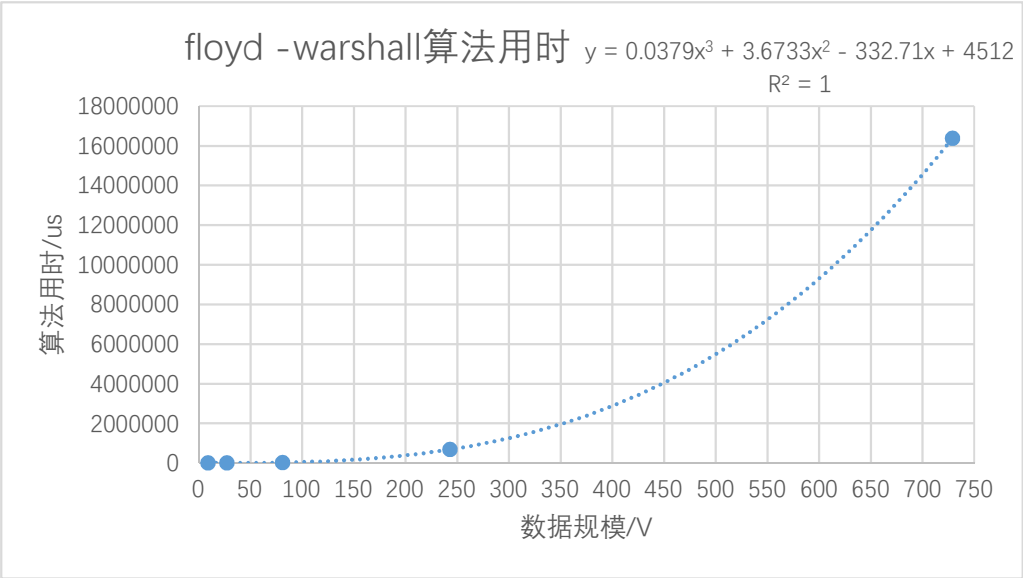
v0 to v1 is: 0 2 4 1 length=53
v0 to v2 is: 0 2 length=32
v0 to v3 is: INF length=0
v0 to v4 is: 0 2 4 length=34
v0 to v5 is: 0 2 5 length=29
v0 to v6 is: 0 6 length=30
v0 to v7 is: 0 2 5 7 length=31
v0 to v8 is: 0 2 5 8 length=57
v1 to v0 is: INF length=0
v1 to v2 is: 1 4 2 length=46
v1 to v3 is: INF length=0
v1 to v4 is: 1 4 length=14
v1 to v5 is: 1 4 2 5 length=43
v1 to v6 is: 1 6 length=25
v1 to v7 is: 1 4 2 5 7 length=45
v1 to v8 is: 1 4 2 5 8 length=71
v2 to v0 is: INF length=0
v2 to v1 is: 2 4 1 length=21
v2 to v3 is: INF length=0
v2 to v4 is: 2 4 length=2
v2 to v5 is: 2 5 length=-3
v2 to v6 is: 2 4 1 6 length=46
v2 to v7 is: 2 5 7 length=-1
v2 to v8 is: 2 5 8 length=25
v3 to v0 is: INF length=0
v3 to v1 is: 3 4 1 length=18
v3 to v2 is: 3 4 2 length=31



v3 to v4 is: 3 4 length=-1
v3 to v5 is: 3 4 2 5 length=28
v3 to v6 is: 3 8 6 length=35
v3 to v7 is: 3 8 7 length=3
v3 to v8 is: 3 8 length=8
v4 to v0 is: INF length=0
v4 to v1 is: 4 1 length=19
v4 to v2 is: 4 2 length=32
v4 to v3 is: INF length=0
v4 to v5 is: 4 2 5 length=29
v4 to v6 is: 4 1 6 length=44
v4 to v7 is: 4 2 5 7 length=31
v4 to v8 is: 4 2 5 8 length=57
v5 to v0 is: INF length=0
v5 to v1 is: 5 2 4 1 length=47
v5 to v2 is: 5 2 length=26
v5 to v3 is: INF length=0
v5 to v4 is: 5 2 4 length=28
v5 to v6 is: 5 8 6 length=55
v5 to v7 is: 5 7 length=2
v5 to v8 is: 5 8 length=28
v6 to v0 is: INF length=0
v6 to v1 is: INF length=0
v6 to v2 is: INF length=0
v6 to v3 is: INF length=0
v6 to v4 is: INF length=0
v6 to v5 is: INF length=0
v6 to v7 is: INF length=0
v6 to v8 is: INF length=0
v7 to v0 is: INF length=0
v7 to v1 is: INF length=0
v7 to v2 is: INF length=0
v7 to v3 is: INF length=0
v7 to v4 is: INF length=0
v7 to v5 is: INF length=0
v7 to v6 is: INF length=0
v7 to v8 is: INF length=0
v8 to v0 is: INF length=0
v8 to v1 is: 8 2 4 1 length=44
v8 to v2 is: 8 2 length=23
v8 to v3 is: INF length=0
v8 to v4 is: 8 2 4 length=25
v8 to v5 is: 8 2 5 length=20
v8 to v6 is: 8 6 length=27

v8 to v7 is: 8 7 length=-5

Floyd-washall 算法的时间复杂度为 O(V^3)。利用上表做出下图，可以看出 Floyd-washall 算法的渐进时间符合 O（V^3）。



七、实验心得

实验中，SCC 算法在求连通分量的同时将各连通分量的成员写入到文件中。但数据规模较小时，实验环境是对算法的运行时间的主要影响。Floyd-washall 算法中，当某对结点 i,j 不可到达时，设定其距离为无穷，而在实验时，INFINIT 只能设定为一个很大的数，且要大于最大的最短路径。当有权值为负的边时，必须要考虑 负数+INFINIT<INFINIT 的情形。