实验 4

1. 实验要求

1、输入输出格式:

- **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,size6,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 a,b 序号(数字表示)，表示存在一条节点 a 指向节点 b 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,随机生成的无向图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 a,b 序号，表示这两个节点之间存在着一条边相连。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,size6 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。**

2、实验细节

- **a)进行算法实现时选取合适的数据结构和实现方法来表示图。**
- **b)实验一中输出的连通分量数据要表示清楚，同一个连通分量的节点序列**

用一对括号括起来输出到 output.txt 文件中，如果可以实现图形化显示每个连通分量并正确清楚的表示出来可以给予加分。

- **c)实验一中输出的最短路径要表示清楚，在一条最短路径的节点序列用一对括号括起来输出到 output.txt 文件中，并输出路径的长度。**
- **d)针对书上给的算法能够进行部分改进或创新并正确实现的，可以给予加分。**

2. 实验环境

编译环境：Xcode11.0
机器内存：8G
时钟主频：2.9GHz

3. 实验过程

实验一：强连通分量

实验中我尝试使用了 class 的结构，并用 list 来表示领接表，其余算法细节基本与课本相同，先 DFS 计算出 finishTime，再构造转置图，然后再 DFS 一次

实验二：Johnson 算法

实验一中领接矩阵的结构在这时不是很需要了，就使用了领接矩阵，其他的好像也没什么好说的，initialze-single-source，relax，bellman-For，dijkstra 都是按课本一行一行写的，然后主函数的地方如果 bellman-Ford 返回了 false，说明出现了负权值回路，这时重新设置权值，再返回主循环就好了

4. 实验关键代码截图（结合文字说明）

实验一：

图的结构：

```
class Graph {
    int V;    // 顶点个数
    list<int> *adj;    //邻接表 最晚的完成时间的顶点放在栈顶
    void fillOrder(int v, bool visited[], stack<int> &Stack);// DFS打印以v起点的边
    void DFSUtil(int v, bool visited[]);
public:
    Graph(int V);
    void addEdge(int v, int w);//打印所有的强连通分量
    void printSCCs();//得到当前图的转置图
    Graph getTranspose();
};
```

DFS：

```
void Graph::DFSUtil(int v, bool visited[]) {
    visited[v] = true;
    fout << v << " ";
    list<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
            DFSUtil(*i, visited);
}
```

转置：

```
Graph Graph::getTranspose() {
    Graph g(V);
    for (int v = 0; v < V; v++) {
        list<int>::iterator i;
        for (i = adj[v].begin(); i != adj[v].end(); ++i) {
            g.adj[*i].push_back(v);
        }
    }
    return g;
}
```

计算强连通分量：

```
void Graph::printSCCs() {
    stack<int> Stack;

    bool *visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;

    // 根据完成时间压入栈中，栈顶是完成时间最晚的顶点
    for (int i = 0; i < V; i++)
        if (visited[i] == false)
            fillOrder(i, visited, Stack);

    // 创建转置图
    Graph gr = getTranspose();
    // 准备第二次DFS
    for (int i = 0; i < V; i++)
        visited[i] = false;

    while (Stack.empty() == false) {
        int v = Stack.top();
        Stack.pop();
        // 打印以v为起点的强连通分量
        if (visited[v] == false) {
            fout <<"以下一行为强连通分量"<<endl;
            gr.DFSUtil(v, visited);
            fout << endl;
        }
    }
}
```

实验二：**Johnson 算法**

图结构：

```
typedef struct{
    int d[258];//书本上的d
    int p[258];//书本上的π
    int weight[258][258];
    int vertixNum;
}Graph;
```

```
void Initialize_Single_Source(Graph &G, int s)
{
    for(int i=1; i<=G.vertixNum; i++)
    {
        G.d[i] = INT_MAX;
        G.p[i] = -1;
    }
    G.d[s] = 0;
}
```

```
void Relax(int u, int v, Graph &G)
{
    if(G.d[v] > G.d[u] + G.weight[u][v])
    {
        G.d[v] = G.d[u] + G.weight[u][v];
        G.p[v] = u;
    }
}
```

```
bool Bellman_Ford(Graph &G, int s)
{
    Initialize_Single_Source(G, s);
    for(int i=1; i<G.vertixNum; i++)
    {
        for(int u=1; u<=G.vertixNum; u++)
            for(int v=1; v<=G.vertixNum; v++)
            {
                if(G.weight[u][v] != INT_MAX)
                    Relax(u, v, G);
            }
    }
    for(int u=1; u<=G.vertixNum; u++)
        for(int v=1; v<=G.vertixNum; v++)
        {
            if(G.weight[u][v] != INT_MAX)
            {
                if(G.d[v] > G.d[u] + G.weight[u][v])
                    return false;
            }
        }
    return true;
}
```

```
bool dijkstra(Graph &G, int s)
{
    int flag[300];
    int q = G.vertixNum;
    int u;
    Initialize_Single_Source(G, s);
    for(int i=1; i<=G.vertixNum; i++)
        flag[i] = 1;
    while(q--)
    {
        u = FindMin(G.d, flag, G.vertixNum);
        flag[u] = 0;
        for(int v=1; v<=G.vertixNum; v++)
            if(flag[v] && G.weight[u][v] != INT_MAX)
                Relax(u, v, G);
    }
    for(int i=1; i<=G.vertixNum; i++)
        if(G.p[i] == -1 && i != s)
            return false;
    return true;
}
```

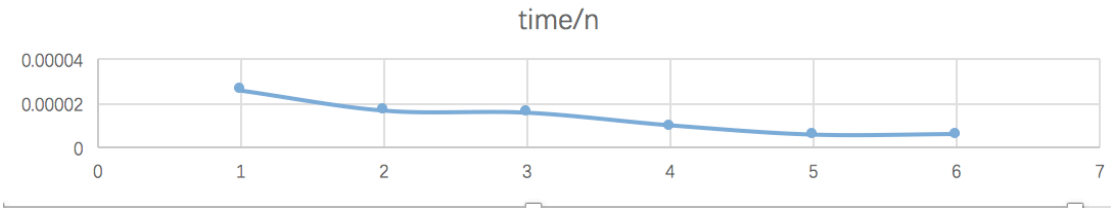
```
bool Johnson(Graph &G)
{
    int h[300];
    G.vertixNum++;
    for(int i=1; i<G.vertixNum; i++)
        G.weight[G.vertixNum][i] = 0;
    if(!Bellman_Ford(G, G.vertixNum))
        return false;
    else
    {
        for(int i=1; i<G.vertixNum; i++)
            h[i] = G.weight[G.vertixNum][i];
        G.vertixNum--;
        for(int u=1; u<=G.vertixNum; u++)
            for(int v=1; v<=G.vertixNum; v++)
            {
                if(G.weight[u][v] != INT_MAX)
                    G.weight[u][v] = G.weight[u][v] + h[u] - h[v];
            }
        for(int u=1; u<=G.vertixNum; u++)
        {
            if(!dijkstra(G, u))
                return false;
            for(int v=1; v<=G.vertixNum; v++)
            {
                Print_Path(G, v, u);
                if(v == u)
                    fout << "->" << u;
                G.d[v] += h[v] - h[u];
                fout << "   Total Length:" << G.d[v] << endl;
            }
        }
    }
    return true;
}
```

5. 实验结果、分析（结合相关数据图表分析）

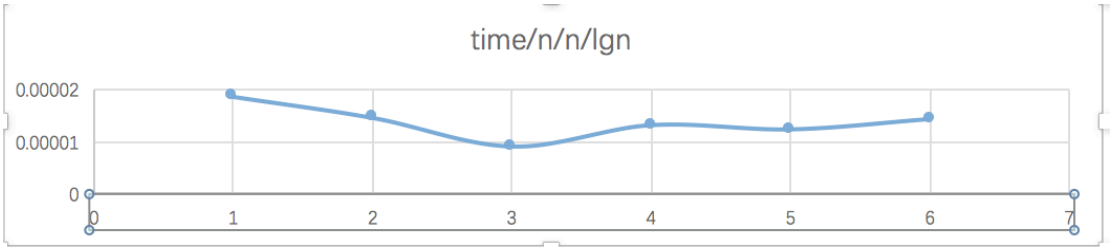
耗时：

project1						
size	8	16	32	64	128	256
timecost	0.000211	0.000273	0.000515	0.000651	0.000751	0.001575
time/n	0.000026375	1.7063E-05	1.6094E-05	1.0172E-05	5.8672E-06	6.1523E-06
project2						
size	8	16	32	64	128	256
timecost	0.001201	0.003732	0.009125	0.053896	0.201798	0.941519
time/n/n	1.87656E-05	1.4578E-05	8.9111E-06	1.3158E-05	1.2317E-05	1.4366E-05

强连通分量的时间复杂度为 O(V+E)，处理过后，6 个 size 接近于直线



Johnson 算法的运行时间为 O(V2logV + VE)。经过处理后时间也接近一条直线，说明两实验都是较为成功的。



6. 实验心得

这次实验经过助教修改条件以后较为简单，实验一实际上我是想锻炼一下自己写 class 的能力但是效果并不好，浪费了很多时间在这上面，实验二中我使用了较为简单的数据结构，做得比较顺利。谢谢老师与助教的指导！谢谢一个学期以来的指导！