实验三：图算法

PB19030888张舒恒

实验设备和环境

PC一台，Win11企业版操作系统，gcc 9.1.0编译器，Clion 2021.2.2代码编辑器，Excel绘图工具

实验内容

1.Bellman-Ford算法

2.Johnson算法

实验要求

代码限制C/C++，建立根文件夹80-张舒恒-PB19030888-project3，在根文件夹下建立本实验报告，ex1和ex2实验文件夹，每个实验文件夹中建立3个子文件夹：input文件夹：存放输入数据，src文件夹：源程序，output文件夹：输出数据。

实验步骤及方法

Bellman-Ford算法

1.文件输入输出

采用绝对路径的输入输出流

```
    string from =
"C:/Users/ASUS/Desktop/connecting/vscode/test/ex1/input/input";
    string dest =
"C:/Users/ASUS/Desktop/connecting/vscode/test/ex1/output/result";
    string time =
"C:/Users/ASUS/Desktop/connecting/vscode/test/ex1/output/time.txt";
    ofstream time_out;
    time_out.open(time);
```

2.Bellman-Ford算法

对每条边进行一次relax操作，并重复|V-1|次迭代

```
void Bellman_Ford(vector<vector<int>> w, vector<int> &d, vector<int> &pi,
vector<pair<int, int>> E){
    //cout << w.size() << endl << E.size()<< endl;
    for(auto i = 0; i < w.size() ; i++)
        for(auto edge: E){
            int u = edge.first, v = edge.second;
            if(d[v] > d[u] + w[u][v] && d[u] != 20000000){
                d[v] = d[u] + w[u][v];
                pi[v] = u;
            }
        }
}
```

3.打印距离和最短路径

由π数组求出前驱结点压入栈中，最后输出栈中所有结点

```
    for(auto j = 0; j < dimension[i]; j++){
            if(pi[j] != -1){
                file_out << "0," << j << "," << d[j] << ";";
                int tmp = j;
                stack<int> path;
                while(tmp != 0)
                    path.push(tmp = pi[tmp]);
                while(!path.empty()){
                    file_out << path.top() << ",";
                    path.pop();
                }
                file_out << j << endl;
            }
        }
```

4.打印运行时间，画出时间曲线并分析

Johnson算法

1.文件输入输出

采用绝对路径的输入输出流

```
string from = "C:/Users/ASUS/Desktop/connecting/vscode/test/ex2/input/input";
string dest = "C:/Users/ASUS/Desktop/connecting/vscode/test/ex2/output/result";
string time =
"C:/Users/ASUS/Desktop/connecting/vscode/test/ex2/output/time.txt";
ofstream time_out;
time_out.open(time);
```

2.Johnson算法

新增一个结点s，其到其他结点的边的权值是0，更新边集合E。调用Bellman_Ford算法求出s结点到其余结点的距离，并将其作为h数组，由h数组求出新的权值数组w^*。对于每个结点调用Dijkstra算法求出其到其余结点的距离。

```
        for(auto k = 0; k < dimension[i]; k++){
            w[dimension[i]][k] = 0;
```

```
            E.push_back(pair<int, int>(dimension[i], k));
        }
        Bellman_Ford(w, d, E);
        vector<int> h(d);
        for(auto edge: E){
            int u = edge.first, v = edge.second;
            w[u][v] = w[u][v] + h[u] - h[v];

        }
        file_out.open(dest + suffix[i] + ".txt");
        for(auto j = 0; j < dimension[i]; j++){

            vector<pair<int, int>> dj(dimension[i]);
            for(auto k = 0; k < dimension[i]; k++)
                dj[k] = pair<int, int>(k, 20000000);
            dj[j] = pair<int, int>(j, 0);
            vector<int> result(dimension[i], 20000000);
            Dijkstra(w, dj, result);
            for(auto k = 0; k < dimension[i]; k++){
                if(result[k]!=20000000)
                    file_out << result[k] + h[k] - h[j] << " ";
                else
                    file_out << "X" << " ";
            }
            file_out << endl;

        }
```

3.Dijkstra算法

每次迭代都从dj数组中抽出源点距离最小的结点，并对其邻接结点做relax操作。

```
void Dijkstra(vector<vector<int>> w, vector<pair<int, int>> &dj, vector<int>
&result){
    while(!dj.empty()){
        auto min = dj.front();
        int min_at = 0;
        for(auto i = 0; i < dj.size(); i++){
            if(dj[i].second < min.second){
                min = dj[i];
                min_at = i;
            }
        }
        result[min.first] = min.second;
        dj.erase(dj.begin() + min_at);
        for(auto i = 0; i < dj.size(); i++)
            if(dj[i].second > min.second + w[min.first][dj[i].first])
                dj[i].second = min.second + w[min.first][dj[i].first];
    }
}
```

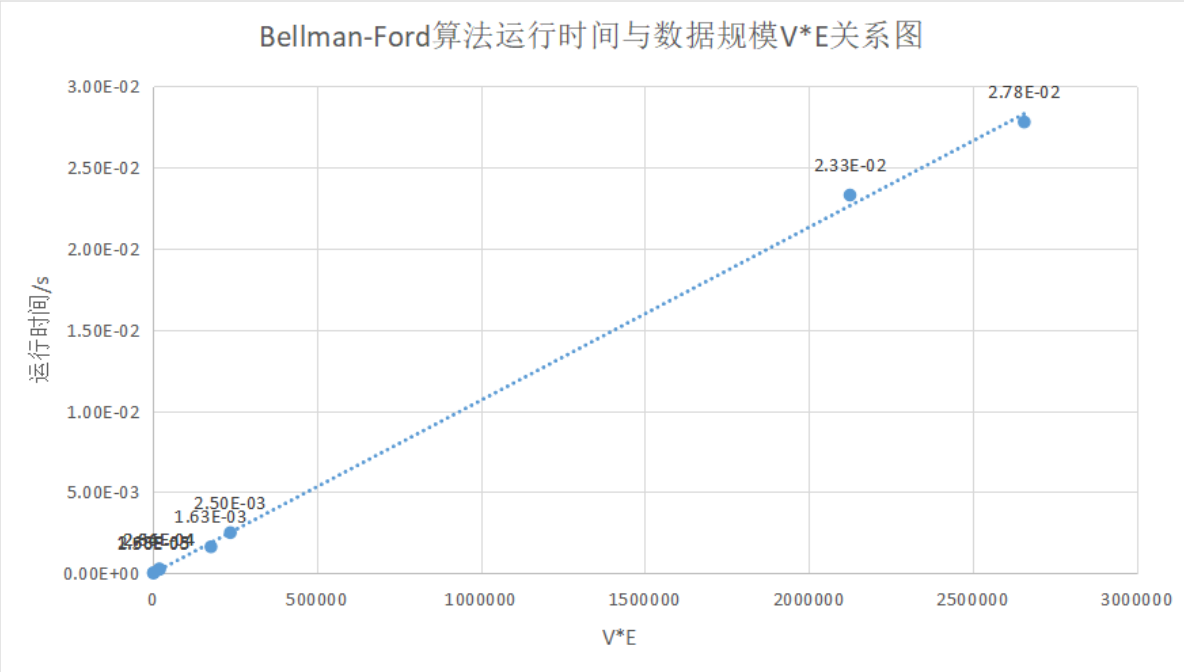
4.打印运行时间画出时间曲线并分析

实验结果分析

Bellman-Ford算法

画出时间曲线，用线性函数进行拟合，结果基本符合一次函数增长模型，所以实际时间复杂度和理论时间复杂度近似相同，均为$O(VE)$

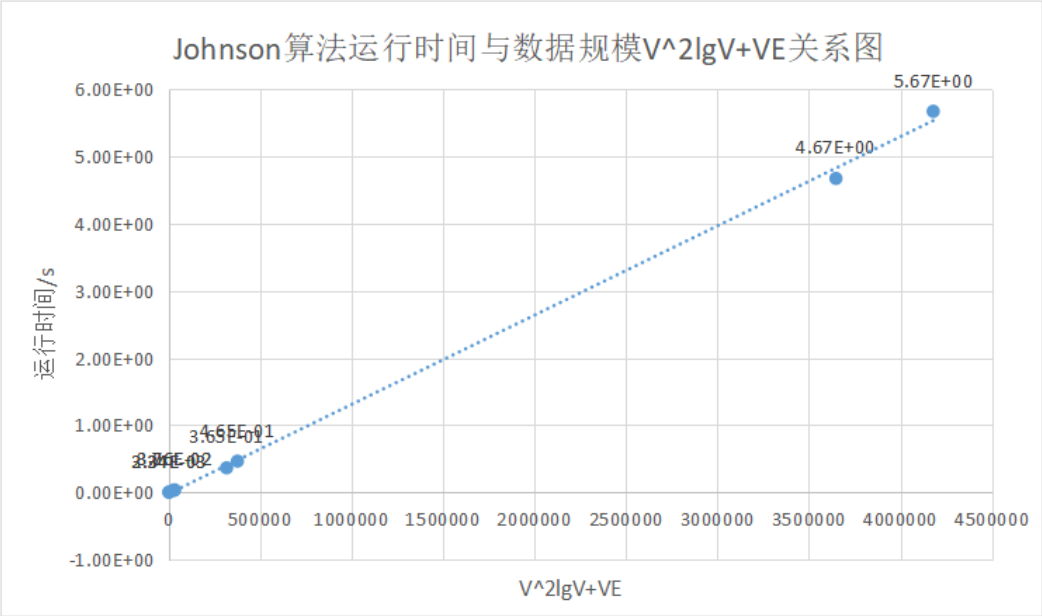
	V	E	V*E	运行时间
2	27	81	2187	2.68E-05
3	81	243	19683	2.66E-04
4	243	972	236196	2.50E-03
5	729	3645	2657205	2.78E-02
6	27	54	1458	1.56E-05
7	81	243	19683	2.34E-04
8	243	729	177147	1.63E-03
9	729	2916	2125764	2.33E-02



Johnson算法

画出时间曲线，用线性函数进行拟合，结果基本符合一次函数增长模型，V=243，E=729时有少许偏差可能是该样例数据分布特殊，但误差在可接受范围内，所以实际时间复杂度和理论时间复杂度近似相同，均为$O(V^2lgV+VE)$

10	V	E	V^2lgV+VE	运行时间
11	27	81	3230.464184	3.34E-03
12	81	243	32204.57021	3.76E-02
13	243	972	377063.6648	4.65E-01
14	729	3645	4178575.78	5.67E+00
15	27	54	2501.464184	2.34E-03
16	81	243	32204.57021	3.06E-02
17	243	729	318014.6648	3.65E-01
18	729	2916	3647134.78	4.67E+00



实验总结

通过本次实验我基本掌握了图相关算法