算法基础LAB4实验报告

PB19071535徐昊天

一.实验内容

1. 给定一个带权无向图G，输出最小生成树和次小生成树的权重。
2. 给定一个带权无向图G，判断图G中的顶点u是否能到达另一顶点v。
3. 实现计算所有结点对最短路径长度的Floyd-Warshall和Johnson算法。
4. 实现在课上讲过的网络流的Ford-Fulkerson算法的两种贪心改进。

二.实验目的

1. 利用OJ编程，强化编程能力，培养编程细节，提升编程素养。
2. 熟悉书本和课堂上涉及的图论算法，进一步巩固算法知识。

三.实验内容

I.次小生成树

1. 首先通过调用算法导论中的kruskal算法求出图G的最小生成树，即可得到最小生成树的权重。
2. 在图G中遍历不在最小生成树中的边，将其加入最小生成树中，得到的图中会存在环，从环路中取出一条除新加入的边以外最长的路并将其删除，即会得到一棵新的生成树。对于遍历得到的所有生成树取最小值，即可得到次小生成树及其权值。

II.可达性查询

1. 首先寻找图G中的所有强连通分量，利用算法导论中提供的SCC算法，连续调用两次DFS算法，找到所有强连通分量后得到一个经过缩点的新图G'，图中顶点为每一个强连通分量。
2. 调用算法导论中的TRANSITIVE-CLOSURE算法，得到图G'中的每一个顶点与其他顶点的可达性关系。
3. 对于每一组输入的测试样例，直接判断各自所在的强连通分量之间的可达性关系即可得到两个顶点之间的可达性关系。

III.顶点距离

1. Floyd-Warshall算法

   直接调用算法导论中的算法，通过三轮循环得到每一个结点对之间的最短路径，时间复杂度为$O(n^3)$。

2. Johnson算法

   - 首先生成图G'，而后在图G'上执行bellman-ford算法，根据计算出来的最短路径权重计算出新的权重。
   - 在通过一层循环对G中的每一个顶点执行dijkstra算法来计算最短路径权重，而后通过得到的最短路径权重计算出G中所有结点对之间的最短路径。

   时间复杂度为$O(n^2lgn+nm)$。

通过编写程序在本地生成样例数据，对以上两种算法性能进行比较。

生成样例数据的程序主体部分如下：

```
int n, m, edge, w;
int num = m / n;
ofstream outfile;
vector<unordered_set<int>> s(n);
outfile.open("file3.txt");
outfile << n << " " << m << endl;
srand((unsigned)time(NULL));
for (int i = 1; i <= n; i++)
{
    for (int j = 0; j < num; j++)
    {
        edge = rand() % n + 1;
        while (edge == i || s[i - 1].find(edge) != s[i - 1].end())
            edge = rand() % n + 1;
        s[i - 1].insert(edge);
        w = rand() % 1001;
        outfile << i << " " << edge << " " << w << endl;
    }
}
outfile.close();
```

在程序中手动设置顶点数n和边数m，为方便生成数据，可令m为n的倍数，num即为以每一个顶点为起点的边数，利用两层循环，对每一个顶点分别随机生成以它为起点的num条边，并用一个集合存储已经生成的随机数以防止生成已经存在的边(为防止得到负环导致程序直接结束妨碍计时，权重w的随机赋值范围保持在[0,1000]中)。将生成的随机数据存储到输出文件中，并在以上两个程序中添加计时器用于计时。生成同样数据分别放入以上两种算法中运行，计时并比较运行时间。

生成随机数据考虑两种情况:稀疏图和非稀疏图。并用同样的顶点数和边数分别生成三组随机数据用于比较，计时结果如下：

1. n=32,m=640

   计时结果如下表所示：

	Floyd-Warshall算法	Johnson算法
1	4.2153ms	21.392ms
2	3.4079ms	20.8893ms
3	4.907ms	32.2117ms

2. n=320,m=640

   计时结果如下表所示：

	Floyd-Warshall算法	Johnson算法
1	1056.13ms	651.594ms
2	741.666ms	580.728ms
3	1966.22ms	749.75ms

根据以上结果可知，图较为稀疏时，Johnson算法的运行时间表现比Floyd-Warshall算法好；图较为稠密时，Floyd-Warshall算法的运行时间表现比Johnson算法好。

Johnson算法

优点：对于稀疏图有着较好的性能，较为灵活。

缺点：算法思想较为复杂，且对于非稀疏图时间复杂度较高。

Floyd-Warshall算法

优点：代码量较小，算法思想较为简单，对于非稀疏图有较高的性能。

缺点：思想较为暴力，处理稀疏图时间复杂度过高，效率低。

IV.最大流

1. 最短增广路径

   调用算法导论中的FORD-FULKERSON算法，通过BFS搜索是否存在s到t的路径，这样可以保证每次搜索到的路径都是最短路径。时间复杂度为$O(nm^2)$。

2. 最大剩余容量增广路径

   对dijkstra算法进行修改，查找出从s出发到其他所有可达结点的路径的最大剩余容量，若可达结点t，则结点t对应的最大剩余容量即为此轮循环应当增加的流量。时间复杂度为$O(m^2lognlog|f^*|)$。

通过编写程序在本地生成样例数据，对以上两种算法性能进行比较。

生成样例数据的程序主体部分如下：

```
vector<int> S;
void dfs_visit(int u, int n, vector<unordered_set<int>> &s, int color[])
{
    color[u] = -1;
    for (auto i = s[u].begin(); i != s[u].end(); i++)
    {
        if (color[*i] == 0)
        {
            S.push_back(*i);
            dfs_visit(*i, n, s, color);
        }
    }
    color[u] = 1;
}

int main()
{
    int n = 600, m = 24000, edge, w;
    int num = m / n;
    int color[n] = {0};
    ofstream outfile;
    vector<unordered_set<int>> s(n);
    outfile.open("file4.txt");
    outfile << n << " " << m << endl;
    srand((unsigned)time(NULL));
    for (int i = 1; i <= n; i++)
    {
        for (int j = 0; j < num; j++)
        {
```

```
            edge = rand() % n + 1;
            while (edge == i || s[i - 1].find(edge) != s[i - 1].end())
                edge = rand() % n + 1;
            s[i - 1].insert(edge);
            w = rand() % 1001;
            outfile << i << " " << edge << " " << w << endl;
        }
    }
    S.clear();
    int u = rand() % n + 1;
    dfs_visit(u, n, s, color);
    int t = rand() % S.size();
    int v = S[t];
    outfile << u << " " << v << endl;
    outfile.close();
    return 0;
}
```

在程序中手动设置顶点数n和边数m，为方便生成数据，可令m为n的倍数，num即为以每一个顶点为起点的边数，利用两层循环，对每一个顶点分别随机生成以它为起点的num条边，并用一个集合存储已经生成的随机数以防止生成已经存在的边。随机生成图信息结束后随意选取一个顶点u，并通过深度优先搜索将u可达的结点存储到容器中，并在容器中随机选取一个可达的结点v，作为最大流测试数据中的s、t。将生成的随机数据存储到输出文件中，并在以上两个程序中添加计时器用于计时。生成同样数据分别放入以上两种算法中运行，计时并比较运行时间。

生成随机数据考虑两种图的的顶点数和边数情况，并用同样的顶点数和边数分别生成三组随机数据用于比较，计时结果如下：

1. n=60,m=600

   计时结果如下表所示：

	最短增广路径	最大剩余容量增广路径
1	3.2145ms	22.3289ms
2	1.0128ms	16.5037ms
3	1.3822ms	11.4168ms

2. n=600,m=6000

   计时结果如下表所示：

	最短增广路径	最大剩余容量增广路径
1	169.798ms	111.264ms
2	136.329ms	106.913ms
3	104.129ms	97.7462ms

根据以上结果可知，顶点数较少时，最短增广路径算法的运行时间表现比最大剩余容量增广路径算法好；顶点数较多时，最大剩余容量增广路径算法的运行时间表现比最短增广路径算法好。

最短增广路径算法

优点：寻找路径的思想较为简单，直接通过BFS即可，且适用于处理顶点数较小的图。

缺点：对于顶点数较大的图，时间复杂度较大。

最大剩余容量增广路径算法

优点：处理顶点数较大的图时效率较高。

缺点：查找最大剩余容量增广路径较为复杂，思路较为繁琐。

四.实验收获与感想

1. 利用OJ编程，养成了更好的编程习惯，促进自身追求更优美的代码风格，提升了编程素养。
2. 提升了编程能力，提前适应上机编程的难度，为将来刷算法题或机试打下了基础。
3. 增强了对代码细节的把控能力，对程序的追求不止于结果正确，而是追求更低的时间空间消耗、数据的完整性和精确性等。
4. 增强了对经典图论算法的理解，通过复现算法更加深入了解了算法的原理和过程。