电子科技大学信息与软件工程学院

课 程 报 告

课程名称	算法设计与分析
学生姓名	任超
学　　号	202021090315
学　　期	2020-2021-1 学期
考核方式	考试/考查

一、算法名称

算法名称：Floyd 算法；

算法名称来源：使用该算法的创始人之一、1978 年图灵奖获得者、斯坦福大学计算机科学系教授罗伯特·弗洛伊德名字命名。

二、算法简要介绍、研究背景与意义

目前，最短路径问题有多种表现形式，不同形式的问题会采用不同的算法。常见的最短路径问题形式有单源最短路径问题，单源负权最短路径问题，多源最短路径问题等。对于单源最短路径问题，常用的算法有 Dijkstra 算法，其基本思想是每次找到离源点（如 1 号顶点）最近的顶点，然后以该顶点为中心进行扩展，最终找到源点到其余所有顶点的最短路径；对于两个顶点之间存在负权的最短路径问题，常采用的算法是 Bellman-Ford 算法，该算法的主要思想是对所有的边进行 n-1 轮松弛操作，因为在一个含有 n 个顶点的图中，任意两点之间的最短路径最多包含 n-1 个边。松弛操作的基本思想是在第 1 轮对所有的边进行松弛之后，得到从 1 号顶点经过一条边到达其余各顶点的最短路径；第 2 轮对所有的边进行松弛之后，得到从 1 号顶点只经过两条边到达其余顶点的最短路径，以此类推从而得到 1 号顶点到其他所有顶点的最短路径。对于多源最短路径问题常采用的算法为 Floyd 算法，两点之间的权重值无论为正还是为负，该算法都能很好地解决此类问题。该算法的基本思想是当求解任意两个顶点之间最短路径时，遍历其他所有顶点，使其成为初始顶点与目的顶点之间的转折点，从而动态地更新任意两点之间的最短路径，直至找到任意两顶点间距离的最优值。在解决双权值问题时，如果同时考虑两个权值会比较麻烦，此时应该枚举一个权值再计算另外一个权值。Floyd 算法解决双权值问题的核心在于巧妙地变化最外层循环参数 k 的含义，假设需要求解任意两个顶点 i、j 之间的最短路径，该路径是不断由顶点 k 中转更新得到的，即 i、j 的最短路径经过的顶点不断由 1、2、…、k-1、k 松弛得到，此时相当于不断枚举 k 值来更新 i、j 之间的最短路径。

Floyd 算法在实际生活中的应用极其广泛，可以解决传递闭包问题（利用元素的传递性求出尽可能多的元素的关系问题），解决各个城市之间的最短路径规划问题以及解决不同地点之间的物资配送问题等等，具有很强的实用性。

三、问题描述

多个顶点相互连接组成一个带权重的图，根据已有的权重值计算图中任意两点之间连接的最小权重值，并且记录所经过的所有顶点，也可看做是求任意两点之间的最短路径问题。

四、算法设计与步骤（包含流程图）

1、算法基本原理

算法的基本原理介绍如下：

假设所有顶点组成的集合为T，起始顶点为u，终点为v，起点与终点的最短路径记作$D_T(u,v)$，将集合T中除起点和终点外的其他任意顶点记作x。此时最短路径有两种形态，经过顶点x，不经过顶点x，因此最短路径的公式见式（1）。

$$D_T(u,v) = min\begin{cases} D_{T-\{x\}}(u,x) + D_{T-\{x\}}(x,v) \\ D_{T-\{x\}}(u,v) \end{cases} \quad x \in T \qquad (1)$$

2、算法流程图

假设 i、j、k 分别表示起点、终点、转折点，P[i][j]记录记录顶点 i 到顶点 j 所经过的第一个点，D[i][j]表示顶点 i 到顶点 j 的最短距离，N 表示顶点的个数，算法的流程图如图 1 所示。

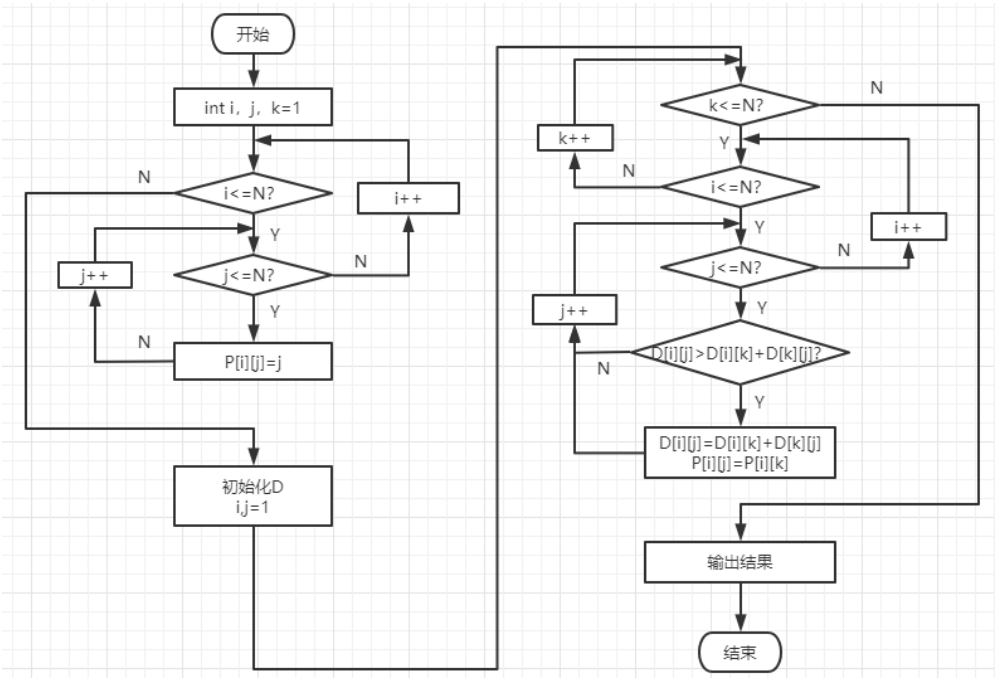

图 1 算法流程图

3、算法演示

以图 2 所示权重图对算法进行演示。

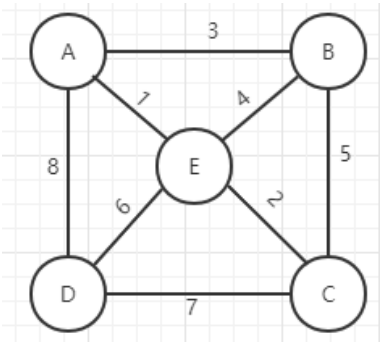

图 2 样例图

Step1：初始化权重矩阵 D，结果如图 3 所示。

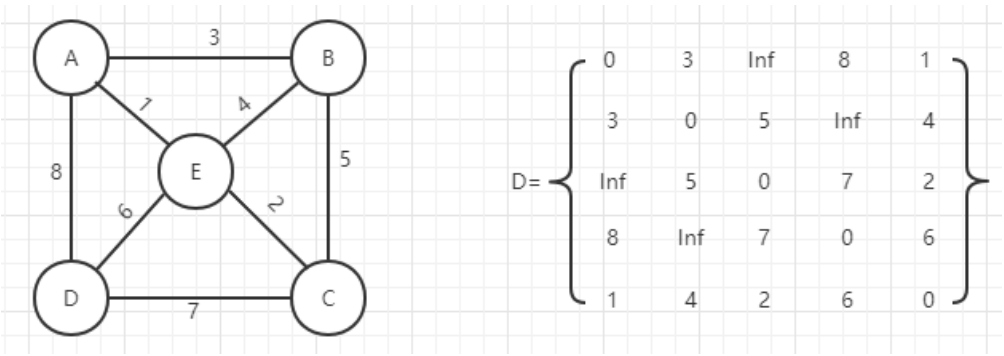

图 3 *Step1* 结果图

Step2：选择 A 为转折点更新矩阵 D，结果如图 4 所示。

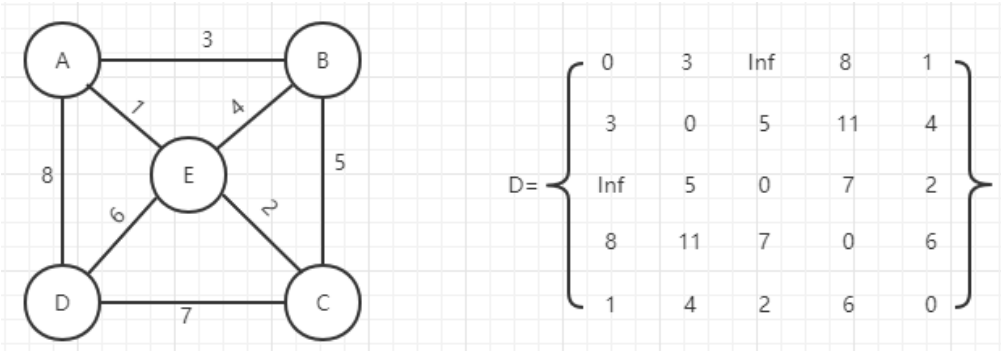

图 4 *Step2* 结果图

Step3：选择 B 为转折点更新矩阵 D，结果如图 5 所示。

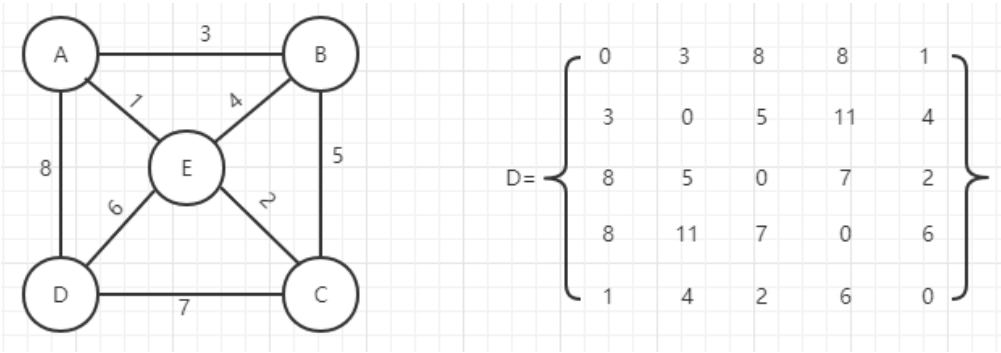

图 5 *Step3* 结果图

Step4：选择 C 为转折点更新矩阵 D，结果如图 6 所示。

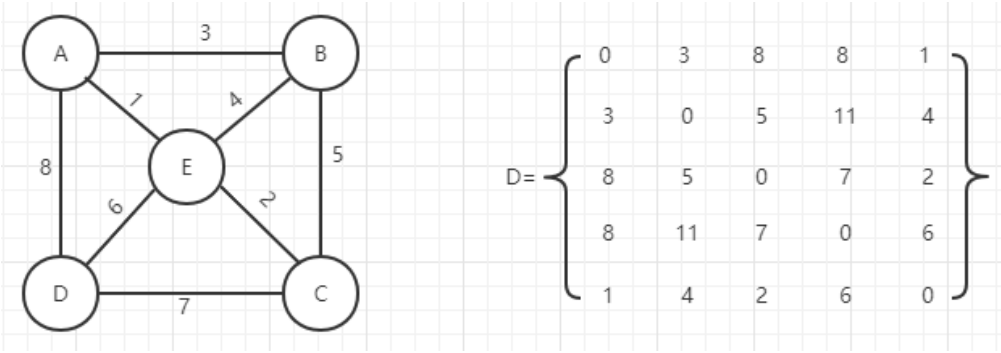

图 6 *Step4* 结果图

Step5：选择 D 为转折点更新矩阵 D，结果如图 7 所示。

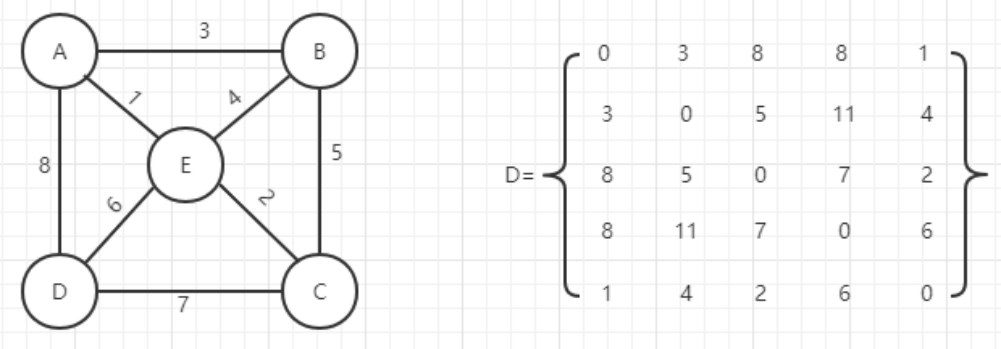

图 7 *Step5* 结果图

Step6：选择 E 为转折点更新矩阵 D，结果如图 8 所示。

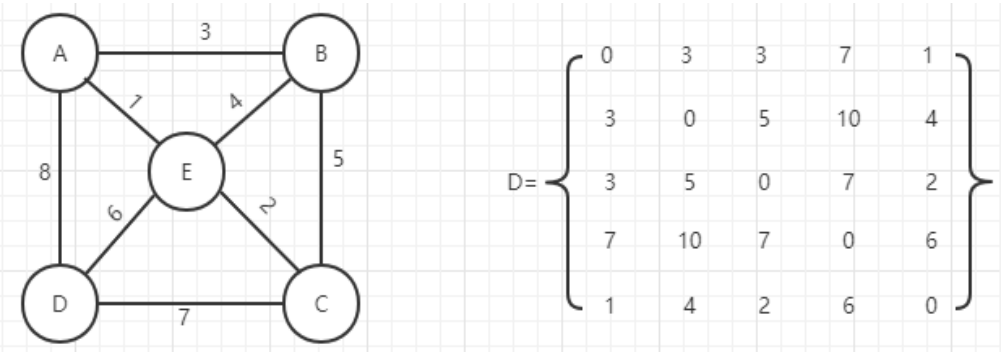

图 8 *Step6* 结果图

五、实验设计与实验结果

1.实验环境

编程语言：Java

编程软件：Eclipse

2.数据集

设置两种不同情况的权重样例图，第一种情况为无向图，测试样例如图 9(a)所示；第二种情况为有向图，测试样例如图 9(b)所示。

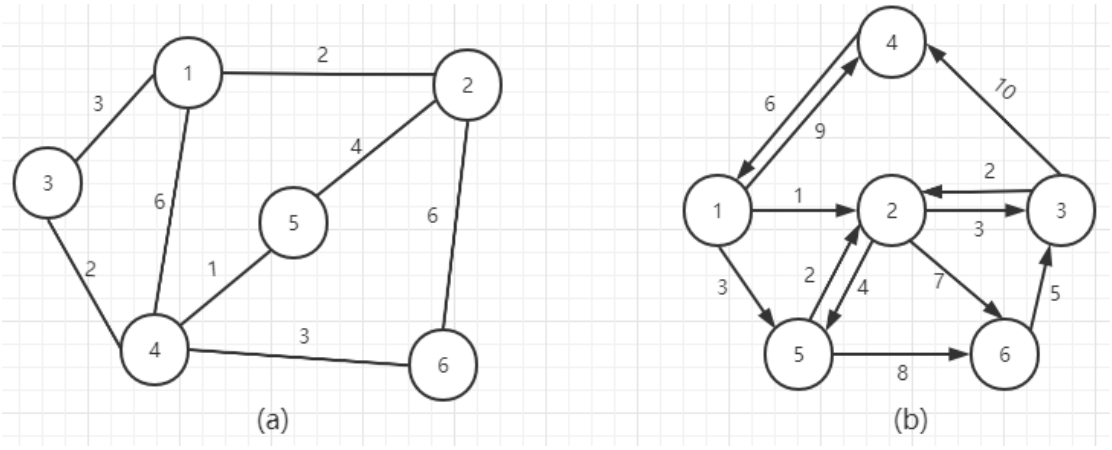

图 9 测试数据图

3.评价指标

算法评价指标一般常采用两个指标：时间复杂度以及空间复杂度。由于 Flyod 算法运行过程最多有三层循环，因此时间复杂度为$O(n^3)$，数据利用二维矩阵进行储存，因此空间复杂度为$O(n^2)$。

2.实验结果分析

2.1 实验代码

① 数据定义：若两点之间不存在连接，则定义两点之间的距离为 max，并设置 max 为 100，NUM 代表顶点个数，route 存储路径，dist 存储两点间最小值。

```
static int max=100; //定义越界最大值
static int NUM=6;
static int[][] route=new int[NUM+1][NUM+1];
static int[][] dist=new int[NUM+1][NUM+1];
```

② 初始化数据：初始化矩阵 route 以及 dist 的值，根据图 9 中的数据手动输入矩阵 dist 中的数据。

```
        //初始化数据
        for(int i=1;i<=NUM;i++){
            for(int j=1;j<=NUM;j++){
                if(i!=j){
                    dist[i][j]=max;
                }
```

```
            else{
                dist[i][j]=0;
            }
            route[i][j]=j;
        }
    }
```

```
    //图(a)初始数据
    dist[1][2]=2;dist[1][3]=3;dist[1][4]=6;
    dist[2][1]=2;dist[2][5]=4;dist[2][6]=6;
    dist[3][1]=3;dist[3][4]=2;
    dist[4][1]=6;dist[4][3]=2;dist[4][5]=1;dist[4][6]=3;
    dist[5][2]=4;dist[5][4]=1;
    dist[6][2]=6;dist[6][4]=3;
```

```
    //图(b)初始数据
    dist[1][2]=1;dist[1][4]=9;dist[1][5]=3;
    dist[2][3]=3;dist[2][5]=4;dist[2][6]=7;
    dist[3][2]=2;dist[3][4]=10;
    dist[4][1]=6;
    dist[5][2]=2;dist[5][6]=8;
    dist[6][3]=5;
```

③ 迭代求解：

```
    //求解最短路径及对应的值
    for(int k=1;k<=NUM;k++){
        for(int i=1;i<=NUM;i++){
            for(int j=1;j<=NUM;j++){
                if(dist[i][j]>dist[i][k]+dist[k][j]){
                    dist[i][j]=dist[i][k]+dist[k][j];
                    route[i][j]=route[i][k];
                }
            }
        }
    }
```

④ 输出结果：

```
    //输出最短路径及结果
    for(int i=1;i<=NUM;i++){
        String log=new String();
        int temp=0;
```

```
            for(int j=1;j<=NUM;j++){
                log=Integer.toString(i)+"->"+Integer.toString(j)+"\t"
+"最短距离: "+dist[i][j]+"\t";
                log=log+"最短路径为:"+Integer.toString(i)+"->";
                temp=route[i][j];
                while(temp!=j){
                    log=log+Integer.toString(temp)+"->";
                    temp=route[temp][j];
                }
                log=log+Integer.toString(j);
                System.out.println(log);
            }
        }
```

2.2 结果展示

图 9(a)结果如图 10 所示。

```
1->1    最短距离: 0     最短路径为:1->1
1->2    最短距离: 2     最短路径为:1->2
1->3    最短距离: 3     最短路径为:1->3
1->4    最短距离: 5     最短路径为:1->3->4
1->5    最短距离: 6     最短路径为:1->2->5
1->6    最短距离: 8     最短路径为:1->2->6
2->1    最短距离: 2     最短路径为:2->1
2->2    最短距离: 0     最短路径为:2->2
2->3    最短距离: 5     最短路径为:2->1->3
2->4    最短距离: 5     最短路径为:2->5->4
2->5    最短距离: 4     最短路径为:2->5
2->6    最短距离: 6     最短路径为:2->6
3->1    最短距离: 3     最短路径为:3->1
3->2    最短距离: 5     最短路径为:3->1->2
3->3    最短距离: 0     最短路径为:3->3
3->4    最短距离: 2     最短路径为:3->4
3->5    最短距离: 3     最短路径为:3->4->5
3->6    最短距离: 5     最短路径为:3->4->6
4->1    最短距离: 5     最短路径为:4->3->1
4->2    最短距离: 5     最短路径为:4->5->2
4->3    最短距离: 2     最短路径为:4->3
4->4    最短距离: 0     最短路径为:4->4
4->5    最短距离: 1     最短路径为:4->5
4->6    最短距离: 3     最短路径为:4->6
5->1    最短距离: 6     最短路径为:5->2->1
5->2    最短距离: 4     最短路径为:5->2
5->3    最短距离: 3     最短路径为:5->4->3
5->4    最短距离: 1     最短路径为:5->4
5->5    最短距离: 0     最短路径为:5->5
5->6    最短距离: 4     最短路径为:5->4->6
6->1    最短距离: 8     最短路径为:6->2->1
6->2    最短距离: 6     最短路径为:6->2
6->3    最短距离: 5     最短路径为:6->4->3
6->4    最短距离: 3     最短路径为:6->4
6->5    最短距离: 4     最短路径为:6->4->5
6->6    最短距离: 0     最短路径为:6->6
```

图 10 图 9(a)结果图

图 9(b)结果如图 11 所示。

```
1->1    最短距离: 0     最短路径为:1->1
1->2    最短距离: 1     最短路径为:1->2
1->3    最短距离: 4     最短路径为:1->2->3
1->4    最短距离: 9     最短路径为:1->4
1->5    最短距离: 3     最短路径为:1->5
1->6    最短距离: 8     最短路径为:1->2->6
2->1    最短距离: 19    最短路径为:2->3->4->1
2->2    最短距离: 0     最短路径为:2->2
2->3    最短距离: 3     最短路径为:2->3
2->4    最短距离: 13    最短路径为:2->3->4
2->5    最短距离: 4     最短路径为:2->5
2->6    最短距离: 7     最短路径为:2->6
3->1    最短距离: 16    最短路径为:3->4->1
3->2    最短距离: 2     最短路径为:3->2
3->3    最短距离: 0     最短路径为:3->3
3->4    最短距离: 10    最短路径为:3->4
3->5    最短距离: 6     最短路径为:3->2->5
3->6    最短距离: 9     最短路径为:3->2->6
4->1    最短距离: 6     最短路径为:4->1
4->2    最短距离: 7     最短路径为:4->1->2
4->3    最短距离: 10    最短路径为:4->1->2->3
4->4    最短距离: 0     最短路径为:4->4
4->5    最短距离: 9     最短路径为:4->1->5
4->6    最短距离: 14    最短路径为:4->1->2->6
5->1    最短距离: 21    最短路径为:5->2->3->4->1
5->2    最短距离: 2     最短路径为:5->2
5->3    最短距离: 5     最短路径为:5->2->3
5->4    最短距离: 15    最短路径为:5->2->3->4
5->5    最短距离: 0     最短路径为:5->5
5->6    最短距离: 8     最短路径为:5->6
6->1    最短距离: 21    最短路径为:6->3->4->1
6->2    最短距离: 7     最短路径为:6->3->2
6->3    最短距离: 5     最短路径为:6->3
6->4    最短距离: 15    最短路径为:6->3->4
6->5    最短距离: 11    最短路径为:6->3->2->5
6->6    最短距离: 0     最短路径为:6->6
```

图 11 图 9(b)结果图

六、算法优化（可选）[1]

算法的评价指标包含空间复杂度以及时间复杂度，因此算法的优化主要是针对这两方面的提升。

空间复杂度的优化策略：采用二维矩阵存储时会浪费一部分空间存储无意义的数据（如“0”和“∞”），若采用结构体的形式来存储数据，结构体中包含用以判断的权值、方向。对所有点赋初值为 0,方向为空。在每一次的计算过程中，只对结果进行比较，进而定义每个点的权值以及方向。在整个计算过程中，对每个点的数据和方向只进行一次记录，存储空间的访问次数降低，空间利用率上升。对于顶点个数为 n 的有向图，结构体需要的存储空间为 2n，因此空间复杂度降为$O(n)$。

时间复杂度优化策略：基本思想是从终点出发，向相邻点方向遍历，更新每个点的权值，求出终点与相邻点之间的路径长度，对于有多个入度和出度的点，取其中最短路径值作为该点的权值，如式(2)所示。

$$D_i = \min\ (D_j + d_{ij}) \tag{2}$$

其中D_i表示v_i的路径长度；D_j代表v_i相邻点v_j的路径长度；d_{ij}代表v_i到v_j之间计算所得的路径值。对所有与终点相邻点的权值进行比较，对权值最小的点进行标记。重复上述过程直至起点，如式(3)所示。

$$D_1 = D_k + d_{1k} \tag{3}$$

根据上述公式，可以将整条最短路径细分为局部的最短路径，即先在终点相邻的顶点中，计算与终点距离最为接近的点，记录路径方向，以此为依据，从之前标记的点出发，依次计算得到最终整条路径的结果。若m为一次计算的个数，n为迭代次数，则每一步的时间复杂度为$O(m \times n)$。在每一步计算过程中，由于不同顶点的相邻点数目不同，因此每一步的计算量都不同，当算法运行一遍后，每个顶点只遍历一遍，因此算法总时间复杂度如式(4)所示。

$$\sum_{k=1}^{t} O(m_k) = O(n) \tag{4}$$

其中 n 为顶点个数，m_k为每一步计算的个数，t为计算的总次数。

七、总结与展望

Floyd 算法常用于求解多源最短路径问题，其本质是一种动态规划算法。该算法适用的范围较广，图的边权值可正可负，图的类型可为有向图以及无向图。在某些问题中，求解多元最短路径问题可以采用多次执行 Dijkstra 算法的方式解决，但是对于稠密图来说，该算法由于三重循环结构紧凑，其执行效率要高于执行多次 Dijkstra 算法，也要高于多次执行 SPFA 算法。经过理论分析与编程实践，对于 Floyd 的算法优缺点总结如下：

优点：容易理解，可以算出任意两个节点之间的最短距离，代码编写简单。

缺点：时间复杂度偏高，不适合计算大量数据。

易理解以及代码编写简单的优点使得 Floyd 算法在计算资源充足的情况下能够快速解决各种多源最短路径问题。但是三重循环的算法结构决定了其较高的时间复杂度，因此该算法在需要计算大量数据的问题中效果不是很理想。针对此缺陷，可以考虑在算法原有的模型下进行一定的局部优化，适当降低算法的时间复杂度，使其计算效率得到提升，从而提高该算法在实际生活中的应用范围。

八、参考文献

[1] 卢立果, 刘立越, 鲁铁定, et al. 一种改进的 Floyd 算法 [J]. 东华理工大学学报(自然科学版), 2019, 42(01): 78-81.

报告评分：

指导教师签字：