运筹学 project2 测试报告

苏茂江 PB21000340

2023.12.18

1　问题介绍

用 C++ 编写 Dijkstra 算法求解最短路问题的通用子程序，并求解最短路问题。

2　算法原理

单纯形法的原理参考课程讲义《Lecture_06_ 最短路和最大流》，主要是 Dijkstra 算法。

3　编译环境及使用方法

3.1　编译环境

编译环境本人采用的是 Vscode，编译器为 g++（Mingw）。

而 project2 中文件.vscode 中 task.json 为 vscode 自动生成的编译文件，eigen-3.4.0 为 eigen 第三方库。

剩余的.cpp 和.h 文件是主要代码，.exe 文件是 vscode 生成的可执行文件，如果运行报错，请删除.exe 文件再试一次。

3.2　使用方法

project2.cpp 是主要测试文件，主要函数的定义和声明都在 function2.h 里面，testtimewithn.cpp 是用来随机生成测试案例，测试 Dijkstra 算法时间与阶数关系。而 comparisiontest.cpp 引用 function2.h 来求解标准 LP 问题，对比两种求解方法的效率。

```
37    // fourth example  connected2
38    // G.resize(4, 4);
39    // G << 0, 1, 4, 6,
40    //     INF, 0, INF, 2,
41    //     INF, INF, 0, 1,
42    //     INF, INF, INF, 0;
43
44    // fifth example connected3 (home work example)
45    G.resize(5, 5);
46    G << 0, 4000, 5400, 9800, INF,
47        INF, 0, 4300, 6200, 8700,
48        INF, INF, 0, 4800, 7100,
49        INF, INF, INF, 0, 4900,
50        INF, INF, INF, INF, 0;
51
52    int flag = checkifconnectedgraph(G); // 0==connected, 1==not connected ,-1==negative weight edge
53    if (flag == -1)
54    {
55        std::cout << "There is negative weight edge in the graph." << std::endl;
56        return 0;
57    }
```

G 输入是系数矩阵，第一个点固定为起始点，填充规则是 $g(i,j)$ 对应的是从点 i 到点 j 的距离长度，如果两点之间无连接，那么则为 INF(infinity), 定义为 C++ 整型的最大值。输入后即可得到结果。五个测试案例均以注释形式给出。

输出的结果是起始点到其余点的最小距离。

3.3　各模块详解

3.3.1　判断图是否连通，是否有负权重边

由函数 checkifconnectedgraph(MatrixXd &G) 实现，位置在 function2.h 的 54-90 行。

实现的方法是先判断是否有负权重边，令一个是判断图是否连通。

3.3.2 Dijkstra 算法

Dijkstra 算法由 project2.cpp 第 73-77 行实现，采用的是下图中的详细算法。

算法 3.18 (Dijkstra 算法)

1. 初始化：令 $d_i = \begin{cases} 0 & i = s \\ \infty & i \neq s \end{cases}$，集合 $P = \varnothing$。
2. 固定最短路：记 i_0 为 $V \backslash P$ 中使 d_i 最小的 i，$P = P \cup \{i_0\}$。
3. 更新：对所有满足 $(i_0, t) \in E, t \in V \backslash P$ 中的 t 作更新 $d_t = \min\{d_t, d_{i_0} + c_{i_0 t}\}$。
4. 终止判定：$P = V$ 时终止，否则回到第二步 (也即二三两步重复 $|V|$ 次)。
5. 结果：终止时的 d 即为 s 到每点的最短路径长度。

其中固定最短路由函数 Fixed_Shortest_Circuit(VectorXd &d, std::vector<int> &P) 实现，位置在 function2.h 的 92-115 行。

更新部分由函数 update(MatrixXd &G, VectorXd &d, std::vector<int> &P) 实现，位置在 function2.h 的 117-137 行。

4 数据集说明

本次测试采用的数据集是自己构造的，还有一道作业题。

5 测试结果

以下实验均在代码中用注释形式保留

5.1 测试案例

5.1.1 连通图 1

```
// first example connected graph1
// G.resize(4, 4);
// G << 0, 1, 2, 3,
//      1, 0, 4, 5,
//      2, 4, 0, 6,
//      3, 5, 6, 0;
```

结果为

```
Shortest circuit:
0
1
2
3
```

5.1.2 有有负权重边

```
// second example negative weight edge
// G.resize(4, 4);
// G << 0, 1, 2, 3,
//      1, 0, 4, 5,
//      2, 4, 0, -6,
//      3, 5, -6, 0;
```

结果为

```
PS D:\STudy\the first semester of junior\Oprational research\homework>  & 'c:\Users\26557\.vscode\extensions\ms-vscode.cpptools-
bugLauncher.exe' '--stdin=Microsoft-MIEngine-In-zoff1ya2.ngt' '--stdout=Microsoft-MIEngine-Out-05b3is2d.1vo' '--stderr=Microsoft
ft-MIEngine-Pid-afh35jwo.ijt' '--dbgExe=C:\Program Files\mingw\mingw64\bin\gdb.exe' '--interpreter=mi'
There is negative weight edge in the graph.
PS D:\STudy\the first semester of junior\Oprational research\homework>
```

5.1.3 非连通图

```
// third example not connected
// G.resize(4, 4);
// G << 0, 1, INF, INF,
//      1, 0, INF, INF,
//      INF, INF, 0, 6,
//      INF, INF, 6, 0;
```

结果为

```
PS D:\STudy\the first semester of junior\Oprational research\homework>  & 'c:\Users\26557\.vscode\extensions\ms-vscode.cpptools-1.
bugLauncher.exe' '--stdin=Microsoft-MIEngine-In-z3q5upy3.ewo' '--stdout=Microsoft-MIEngine-Out-2ro1hn0d.wgv' '--stderr=Microsoft-M
ft-MIEngine-Pid-01kjpqqn.nva' '--dbgExe=C:\Program Files\mingw\mingw64\bin\gdb.exe' '--interpreter=mi'
The graph is not connected.
PS D:\STudy\the first semester of junior\Oprational research\homework>
```

5.1.4 连通图 2

```
// fourth example  connected2
// G.resize(4, 4);
// G << 0, 1, 4, 6,
//      INF, 0, INF, 2,
//      INF, INF, 0, 1,
//      INF, INF, INF, 0;
```

结果为

```
Shortest circuit:
0
1
4
3
PS D:\STudy\the first semester of junior\Oprational research\homework>
```

注意到上面的矩阵是模块 1 和模块 2 工作过后的新矩阵 A，可见两模块均完成工作。

5.1.5 连通图 3

```
// fifth example connected3 (home work example)
G.resize(5, 5);
G << 0, 4000, 5400, 9800, INF,
     INF, 0, 4300, 6200, 8700,
     INF, INF, 0, 4800, 7100,
     INF, INF, INF, 0, 4900,
     INF, INF, INF, INF, 0;
```

这是作业题，答案与预期符合。

```
Shortest circuit:
    0
 4000
 5400
 9800
12500
PS D:\STudy\the first semester of junior\Oprational research\homework>
```

5.2 关于求解时间测试结果

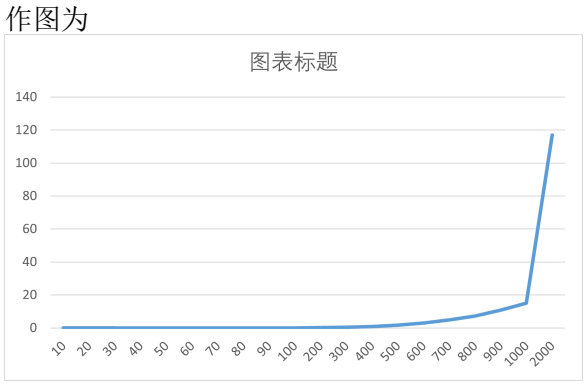
5.2.1 Dijkstra 算法时间与阶数关系

在 testtimewithn.cpp 内测试随机系数矩阵结果，采取随机生成系数矩阵，如果没得到连通图就不计时间，和 project1 的 LP 求解函数比较。

Dijkstra 算法原始求解时间数据为 (单位为 s)

```
     0
     0
     0
 0.001
 0.003
 0.004
 0.009
 0.012
 0.014
 0.019
 0.132
 0.415
 0.938
 1.766
 3.031
 4.916
 7.161
10.815
14.948
117.047
```

这分别是 $n = 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000$ 的情况作图为



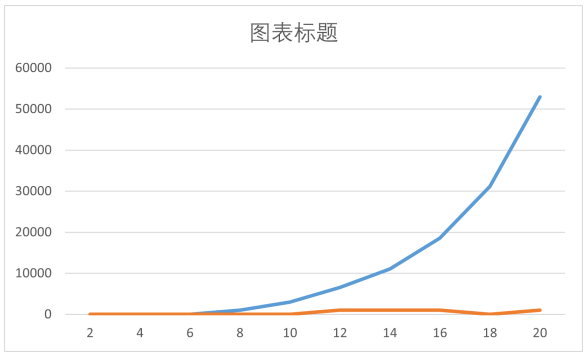
横坐标为 n，纵坐标为时间（s）。

5.2.2 Dijkstra 算法与 project1 单纯形法对比

测试代码在 comparisiontest.cpp 内。原始测试数据为（时间单位为微秒）：

```
n:
2
time_di(micro):
0
time_lp(micro):
0
n:
4
time_di(micro):
0
time_lp(micro):
0
n:
6
time_di(micro):
0
time_lp(micro):
0
n:
8
time_di(micro):
0
time_lp(micro):
1000
n:
10
time_di(micro):
0
time_lp(micro):
3000
n:
12
time_di(micro):
1001
time_lp(micro):
6519
n:
14
time_di(micro):
999
time_lp(micro):
11109
n:
16
time_di(micro):
1000
time_lp(micro):
18560
n:
18
time_di(micro):
0
time_lp(micro):
31172
n:
20
time_di(micro):
1000
time_lp(micro):
52942
```

作图得到（时间单位为微秒）



6 分析与总结

Dijkstra 算法比用 lp 求解快得多。甚至在 20 阶以内没有体现出正常相关关系，认为是在误差范围内。我认为 Dijkstra 算法比 lp 快的主要原因是将最短路问题化为标准 lp 时，系数矩阵是 $n * n^2$ 的，大大增加了计算量。

7 Code

代码见文件