

程序说明及用例分析测试

1 测试用例说明

我们把测试用例分为三个文件：topo.csv, demand.csv, result.csv。

1、topo.csv: 存储无向图信息，每一行为： 边编号，节点 1，节点 2，边权重； 节点编号从 0 开始，边编号从 1 开始；

2、demand.csv: 限制条件信息

第一行： 起点，终点；

第二行： 必经节点编号，以逗号隔开；

第三行： 必经边集合，边的两个节点用逗号隔开，边中间用|隔开；没有必经边此行填 NA；

第四行： 禁止边集合，边的两个节点用逗号隔开，边中间用|隔开;没有必经边此行填 NA；

第五行： 最多经过的节点数；

3. result.csv: 存储运行结果

程序运行时读取这三个文件，topo.csv 为无向图信息，demand.csv 为限制信息，输出结果存在 result.csv 中。

如下图所示，为一个简单 case，及其文件表示形式，表示起点 2，终点 3，必经点 1，必经边 (0, 1)，禁止边 (2, 3)，节点限制 5 的一个 case。

demand.csv	x	...	topo.csv	x
1	2,3		1	1,0,1,1
2	1		2	2,1,2,2
3	0,1		3	3,1,3,2
4	2,3		4	4,2,3,1
5	5			

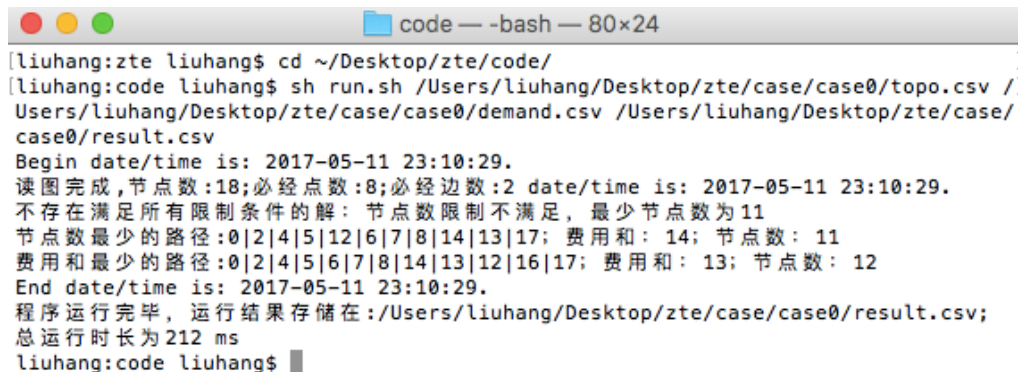
2 程序说明

我们的程序读取图信息和限制信息，如果得到满足所有条件的最优解，输出；如果不能满足所有条件，输出近似解，并给出不能满足的原因。由于我们使用了两种算法，所以给出两个程序，具体说明如下。

2.1 指派模型结合分支定界算法

2.1.1 使用说明

运行环境：Linux /mac osx 系统，程序用 Java 语言编写，运行环境需预先安装 java 运行环境(jdk/jre 1.8 版本)；我们提供运行脚本，使用如下：在终端中直接运行命令；run.sh /xxx/topo.csv /xxx/demand.csv /xxx/result.csv；如下图所示，为运行官方用例时的情况：

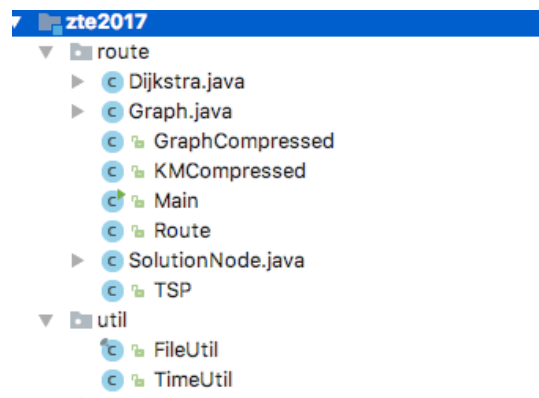


```
[liuhang:zte liuhang$ cd ~/Desktop/zte/code/
[liuhang:code liuhang$ sh run.sh /Users/liuhang/Desktop/zte/case/case0/topo.csv /Users/liuhang/Desktop/zte/case/case0/demand.csv /Users/liuhang/Desktop/zte/case/case0/result.csv
Begin date/time is: 2017-05-11 23:10:29.
读图完成,节点数:18;必经点数:8;必经边数:2 date/time is: 2017-05-11 23:10:29.
不存在满足所有限制条件的解: 节点数限制不满足, 最少节点数为 11
节点数最少的路径:0|2|4|5|12|6|7|8|14|13|17; 费用和: 14; 节点数: 11
费用和最少的路径:0|2|4|5|6|7|8|14|13|12|16|17; 费用和: 13; 节点数: 12
End date/time is: 2017-05-11 23:10:29.
程序运行完毕, 运行结果存储在:/Users/liuhang/Desktop/zte/case/case0/result.csv;
总运行时长为 212 ms
liuhang:code liuhang$
```

此外，源代码在 code/src 文件夹下，我们提供一键编译打包的脚本，直接运行 build.sh 即可得到 jar 包。

2.1.2 源代码说明

源代码结构如下：



各代码文件说明：

- 1、Main.java 为程序入口，通过 main 函数带参数运行，调用 Route.searchRoute()运行算法；
- 2、Graph.java 操作文件，构建原始图和读取限制条件信息；
- 3、TSP.java 为求解 TSP 模型代码，主要包括分支定界法的实现 (branchAndBound 函数)；
- 4、KMCompressed.java 为用 KM 算法求解指派问题的实现；

- 5、Route.java 整体算法入口，实现迭代求解，控制算法流程；
- 6、Dijkstra.java 为堆优化的 Dijkstra 算法实现；
- 7、GraphCompressed.java 为对原始图进行压缩，并构建新图；
- 8、SolutionNode.java 为分支定界法的解节点；
- 9、TileUtil.java 和 Timeutil.java 为工具函数，读取文件和计算时间；

2.2 线性规划结合 lpsolve 求解器

本程序用 C++编写，并调用了 lpsolve5.5.2.5 库。运行环境为 **Linux64 位系统**（其他系统，如 Linux32 位系统或 win32/64 需重新进行编译，详见 /code/lp/README.txt）。

程序使用如下：在 Linux64 位系统中（如 ubuntu 16.04），在终端中，进入 lp 文件夹下，运行：./zte.out /xxx/topo.csv /xxx/demand.csv /xxx/result.csv，如下图所示，为运行官方例子时的情况：

```
xuan@ubuntu:~/Desktop/zte2017-xu$ ./zte.out case/case0/topo.csv case/case0/demand.csv case/case0/result.csv

暂时未找到最优解（可能无解），以下为不限制访问次数时的近似解：
0|2|4|5|6|7|8|14|13|12|16|17
distance:13
visit node number:12

该题被证明无解，以下为访问次数最少情况下的近似解：
0|2|4|5|12|6|7|8|14|13|17
distance:14
visit node number:11

using time:0.042309s
不存在满足所有限制条件的解:节点数据限制不满足,最少节点数为11
节点数最少的路径:最短路径:0|2|4|5|12|6|7|8|14|13|17; 费用和: 14; 节点数: 11
费用和最少的路径:最短路径:0|2|4|5|6|7|8|14|13|12|16|17; 费用和: 13; 节点数: 12
```

3 用例分析及设计

我们总共设计了 20 个测试用例，各个测试用例的情况如下：

- 1、case0: 官方赛题上所给的用例；
- 2、case1-case3: 验证算法及程序正确性的用例，其中
 - a) case1: 用于测试是否考虑节点可重复经过，本用例必须重复经过节点才能得到可行解；
 - b) case2: 用于测试算法对限制条件“路径经过点数不超过 k”的响应，考虑节点限制得到的解中路径不能经过重复点，且路径长度比不考虑节点限制小；
 - c) case3: 测试验证，考虑节点可重复经过的情况下，是否可以得到

费用值更小的解；

3、case4-case20: 程序性能验证，验证程序可以跑多大规模的 case。

a) case4-case6 为中等规模 case，无向图点数在 1000 以内；

b) case7-case20 为大规模 case，无向图点数在 1000-2000；

各个测试用例的详细介绍，见每个 case 文件夹下的用例说明.txt/用例说明.png，每个 case 的运行结果见相应文件夹中的 result.csv。

4 程序测试

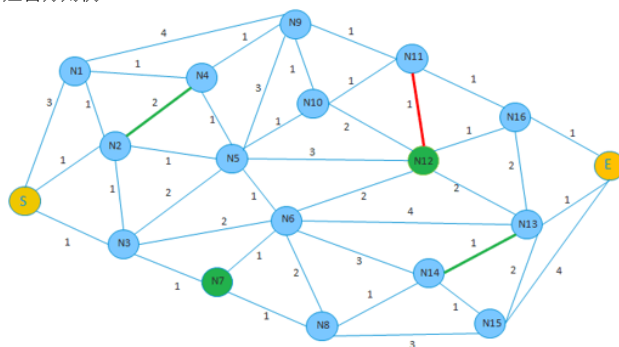
程序测试主要包括以下三个部分：

- 1、对赛题中所述示例的测试；
- 2、对程序正确性及各种特殊情况的测试；
- 3、对程序性能的测试，看看程序能跑多少节点、多少必经点、多少必经边的图以及运行的时间。

测试环境如下：CPU: intel 酷睿 i7; 操作系统: Linux Ubuntu 16.04 64 位; 内存: 8G; 内核: 4 核。

4.1 官方 case

赛题官方用例



我们把赛题上所描述的示例制作为用例(case/case0)，起点谁 0，终点设为 17，包含 18 个节点，2 条必经边，1 条禁忌边，2 个必经节点，限制经过 9 个节点。运行结果如下：

```

result.csv
不存在满足所有限制条件的解：节点数限制不满足，最少节点数为11
节点数最少的路径:0|2|4|5|12|6|7|8|14|13|17; 费用和: 14; 节点数: 11
费用和最少的路径:0|2|4|5|6|7|8|14|13|12|16|17; 费用和: 13; 节点数: 12
    
```

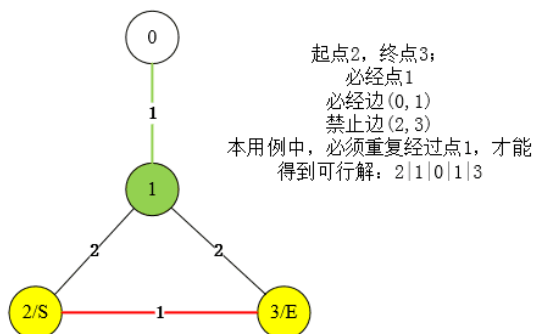
满足所有限制条件的解是不存在的，输出结果中给出了解不存在的原因，即节点

限制不满足。同时，我们给出了两个近似解，费用和最少路径和经过节点数最少的路径。

4.2 正确性测试

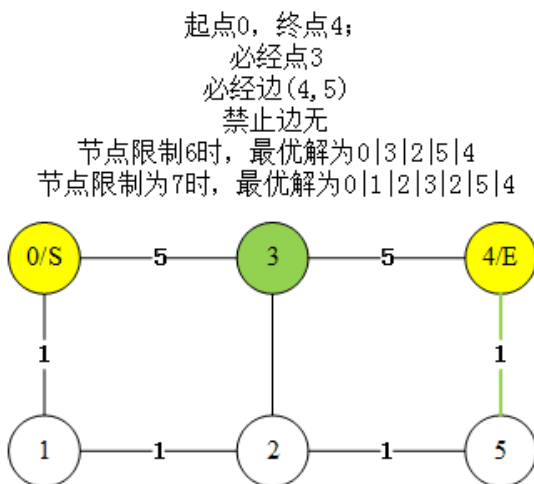
4.2.1 case1

本用例测试是否考虑节点可重复经过



本用例用于测试算法是否考虑节点可重复经过，本例中节点 1 必须重复经过才能得到可行解。经测试，我们使用的算法可以得到可行解 2|1|0|1|3，所以，我们的算法充分考虑了必须经过重复点才能得到可行解的可能性。

4.2.2 case2



本用例用于测试算法对限制条件“路径经过点数不超过 k”的响应，考虑节点限制得到的最优解的费用不是最小的且路径中没有重复点(如上图所示)。所以，我们的算法充分考虑了节点数的限制，并进行了正确的处理。

4.2.3 case3

本用例用于测试验证：考虑节点可重复经过的情况下，是否可以得到费用值

更小的解。本用例的情况如下：

节点可重复经过时路径：2|15|18|3|11|4|13|7|13|17|6|5|9|14|19|0|1|0|19；费用和：103；节点数：19

节点不可重复经过时路径：2|15|10|12|14|9|17|6|5|3|11|7|13|4|1|0|19；费用和：126；节点数：17

所以，经过验证：考虑重复经过，有可能得到费用值最小的解。

Case0-case3 的测试情况汇总如下：

case	点数	边数	必经点	必经边	线性规划+Ipsolve/s	指派模型+分支定界/s
0	18	42	2	2	0.257	0.135
1	4	4	1	1	0.0584	0.126
2	7	7	1	1	0.0614	0.136
3	20	42	6	2	0.2443	0.165

后两栏为两种算法运行的时间。经过测试，两种算法均能通过正确性测试，说明算法及模型正确有效。

4.3 性能测试

根据我们的算法，影响性能主要由三点因素：无向图的的点数，必经点数（包括起点终点），必经边数。Case4-case20 为各种不同规模的用例，测试情况如下

case	点数	边数	必经点	必经边	线性规划+Ipsolve/s	指派模型+分支定界/s
4	160	620	6	8	119.67	0.311
5	300	842	22	0	12.165	0.391
6	500	2000	24	0	X	0.628
7	1000	13333	18	0	104.21	0.739
8	1400	2484	44	0	X	1.656
9	2000	40000	100	0	X	1.286
10	2000	40000	100	100	X	4.42
11	2000	40000	100	110	X	4.94
12	2000	40000	100	120	X	5.6
13	2000	40000	100	130	x	3.48
14	2000	40000	100	140	X	4.29
15	2000	40000	100	150	X	5.4
16	2000	40000	100	180	X	9.44
17	2000	40000	100	200	x	7.87
18	2000	40000	100	300	x	38.4
19	2000	40000	100	350	x	32.99
20	2000	40000	100	400	X	30.2

如上图，为对两种算法的性能测试。经过测试，我们发现：线性规划+Ipsolve 求解器在必经点规模达到 20 时，很难得到可行解，这主要受限于 Ipsolve 求解器的性能；指派模型结合分支定界的方法，表现出非常良好的性能，在目前的测试环境下，5s 内可求解出 100 个必经点 100 条必经边的 case；10s 内可求解 100 必经点 200 必经边的 case；30s 内可求解 100 必经点 300-400 必经边的 case。