云南大学数学与统计学院  
《算法图论实验》上机实践报告

|  |  |  |
| --- | --- | --- |
| 课程名称：算法图论实验 | 年级：2015级 | 上机实践成绩： |
| 指导教师：李建平 | 姓名：刘鹏 | 专业：信息与计算科学 |
| 上机实践名称：中国邮递员问题 | 学号：20151910042 | 上机实践日期：2019-01-01 |
| 上机实践编号：7 | 组号： |  |

# 实验目的

1. 了解一般“中国邮递员问题”及其算法；
2. 了解“有向中国邮递员问题”及其算法。

# 实验内容

1. 写出求最短路的最小插点问题的动态规划算法；
2. 用C语言实现上述算法。

# 实验平台

Windows 10 Pro 1803；

MacOS Mojave。

# 算法设计

## 前期知识

中国邮递员问题起源于实际需求。比之更早的一个问题是“戈尼斯堡七桥问题”，该问题由欧拉解决。给定一个图，如果存在一条简单链过图的每条边一次并且仅仅一次，那么这个链称为欧拉链（Eulerian Chain）；如果存在一个简单圈过图的每条边一次并且仅仅一次，那么这个圈称为欧拉圈（Eulerian Cycle）；若图有欧拉圈，则称图为欧拉图（Eulerian Graph）。

**定理7.1** （Euler定理） 设图是连通图，且边数，则是欧拉图，当且仅当不含奇点[1]。

**证明**

(1) 是欧拉图不含奇点

很显然，如果是欧拉图，那么本身就是一个欧拉圈。很显然，圈不含奇点。

(2) 不含奇点是欧拉图

假设是连通图，边数，且不含奇点，但是同时它不是欧拉图，同时假设是所有这种图中含边最少的一个。由图连通、以及都是偶数点可知，图的最小度。所以图含有圈（也可以这么认为：树是最简单的连通图，最小度为，往树上随便加一条原先不存在的边，都会产生圈）。假设是中含有边数最多的简单圈，因为假设不是欧拉图，所以根据假设可知：含有一个边数大于零的连通分图，且所有点的度均为偶数，由于，所以必须含有欧拉圈，否则就与是所有这种图中含边最少的一个相矛盾；现断言：。该断言可以解释如下：如果交为空，那么依此类推，其余连通分图也含欧拉圈，且与他们的交也为空，于是这些圈之间彼此点不交，这导致不是连通图，显然不可能，所以交非空。如此，可以通过走8字的方式构造一个比边数还多的圈，这与含边最多矛盾，于是这种图不存在，所以是欧拉图。证毕。 □

所谓的中国邮递员问题，指的是一个邮递员每次送邮件，要走遍他负责的投递范围的所有街道，完成任务之后，他应该按照什么样的路线走才可以使得走的总路程最短？把这个问题抽象成图论问题，就是给当一个图，每个边上有非负权值，要求出的一个（未必是简单的）圈，使得过每个边至少一次，并且使得圈的总权重最小。

可以这样思考：加入连通图不含奇点，那么这个图中有欧拉圈，即这是个欧拉图，所含的欧拉圈就是所要求的最佳邮递路线。如果连通图含有奇点，那么所求的圈必然过某条边多于一次，若在边上通过了次，我们就在与之间添加条新的边，并且令新添加的边的权重等于原来的边。对圈上每个这样的边都进行如此的操作，将得到的扩充之后的图命名为，那么就是多重图中的欧拉圈。这时得到一个很显然的事实：是由所有添加边的总权重所决定的。

如果边上的添加边数目不止一条，那么从中删除偶数条添加边，得到的图仍旧是欧拉图，且这个删减后的图的欧拉圈的权重不会变大。因此可以假设每个边上至多有一条添加边。于是，中国邮递员问题又被归结为如下的图论问题：给定连通图，每个边上有非负权，求，满足条件：在中，在的每个边上添加一个重边，使得这样得到的图无奇点。称为可行集（feasible set），并使得达到最小。

可行集是最优集，当且仅当对的每个初等圈，都有

这个事实是指任意简单圈，其包含的添加边的权重之和小于等于原先就存在的边的权重之和。这个定理是奇偶点图上作业法[2]的核心思想来源，下面简单地证明一下。

**Proof**:

**必要性**：设可行集是最优集，但这时存在一个初等圈，使得

这个不等式认为，存在一个圈，圈的边与最优集的交，比圈排除最优集中所有边的剩余还要更大。根据这个假设，可以令，也就是将初等圈中的添加边都舍弃，然后在没有添加过边的相邻点之间都加上点对。这样一来，不会改变整个图中的点都是偶度点的事实（因为两点之间顶多有一个添加边，所以这个操作仅仅会改变这个圈，对于圈外的点不影响奇偶度；对于圈内的点，要么两边的边都是添加边，要么一端是添加边但是另一端不是，在这种情况下进行进行如上操作，两边都是添加边的把这两个添加边都去掉，该点在原图中的奇偶度不变，另外的只有一端是添加边的，只是左右边由“原始边-添加边”变成了“添加边-原始边”或相反，这也不改变这个点的奇偶度），于是，也是一个可行解，且，这与是最优的相矛盾。

**充分性**：只需证明，当存在两个不同的最优集与时，只需证明两者的权重是相等的。对每个点，其关联边在与中的数目是同奇偶的（否则该点在与的分别作用下度不同）。所以不含奇点，进而可以剖分为若干个子集，使得每个子集都是初等圈，并设是所有这样的圈，于是由最优条件，任意的都有

反过来就有

所以

综上所述，充要性得证。

□

## 奇偶点图上作业法

山东师范学院的管梅谷教授提出的邮递员最优化投递路线问题，可以由管教授的奇偶点图上作业法来解决。此算法调用了算法、两点之间求最短路的算法。

|  |  |
| --- | --- |
| **Algorithm**  **Input**  **Output**  **Begin**  **Step 1**  **Step 2**  **Step 3**  **End** | 奇偶点图上作业法求解中国邮递员问题，记此算法为-  ，允许这个图为任意的图  图的一个圈，记-  // 计数并存储奇度点  **list**  **for** **each** vertex :  **if** :    **if** :    **goto** End  **else**:  **for** **through** :  **for** **each** edge :  **edge**        **for** **each** vertex :    **for** **each** vertex :  **for** **each** edge :  **if** **and** :      **else** **if** :    // 圈的对称差  **for** **each** vertex :  **for** **each** vertex :    **if** -:      **for** **each** edge :    **for** **each** **edge** :    **if** : |

这个算法的两个核心问题：如何证明最优化条件，如何找出所有的圈。第一个问题的证明已经给出，但是第二个问题需要设计算法。由于这里找的是简单圈，所以考虑某个点的反圈，从中任取一个点，在中寻找从到的路，这条路如果找到，那么就是一个简单圈。通过循环，可以找到所有的圈，然后挨个验证即可。

# 程序代码

## 程序代码

源代码众多，这里仅仅列出核心文件的代码。所有代码均用Python 2语言写成。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196 | """  Functions relating to Eularian graphs.  This module contains functions relating to the identification  and solution of Eularian trails and Circuits.  """  **import** copy  **import** itertools  **import** random  **import** sys  **from** time **import** clock  **from** **.** **import** dijkstra**,** my\_math  **from** **.**my\_iter **import** all\_unique**,** flatten\_tuples  **def** fleury\_walk**(**graph**,** start**=None,** circuit**=False):**  """  Return an attempt at walking the edges of a graph.  Tries to walk a Circuit by making random edge choices. If the route  dead-ends, returns the route up to that point. Does not revisit  edges.  If circuit is True, route must start & end at the same node.  """  visited **=** set**()** # Edges  # Begin at a random node unless start is specified  node **=** start **if** start **else** random**.**choice**(**graph**.**node\_keys**)**  route **=** **[**node**]**  **while** len**(**visited**)** **<** len**(**graph**):**  # Fleury's algorithm tells us to preferentially select non-bridges  reduced\_graph **=** copy**.**deepcopy**(**graph**)**  reduced\_graph**.**remove\_edges**(**visited**)**  options **=** reduced\_graph**.**edge\_options**(**node**)**  bridges **=** **[**k **for** k **in** options**.**keys**()** **if** reduced\_graph**.**is\_bridge**(**k**)]**  non\_bridges **=** **[**k **for** k **in** options**.**keys**()** **if** k **not** **in** bridges**]**  **if** non\_bridges**:**  chosen\_path **=** random**.**choice**(**non\_bridges**)**  **elif** bridges**:**  chosen\_path **=** random**.**choice**(**bridges**)**  **else:**  **break** # Reached a dead-end, no path options  next\_node **=** reduced\_graph**.**edges**[**chosen\_path**].**end**(**node**)** # Other end  visited**.**add**(**chosen\_path**)** # Never revisit this edge  route**.**append**(**next\_node**)**  node **=** next\_node  **return** route  **def** eularian\_path**(**graph**,** start**=None,** circuit**=False):**  """  Return an Eularian Trail or Eularian Circuit through a graph, if found.  Return the route if it visits every edge, else give up after 1000 tries.  If `start` is set, force start at that Node.  """  **for** i **in** range**(**1**,** 1001**):**  route **=** fleury\_walk**(**graph**,** start**,** circuit**)**  **if** len**(**route**)** **==** len**(**graph**)** **+** 1**:** # We visited every edge  **return** route**,** i  **return** **[],** i # Never found a solution  **def** find\_dead\_ends**(**graph**):**  """  Return a list of dead-ended edges.  Find paths that are dead-ends. We know we have to double them, since  they are all order 1, so we'll do this ahead of time to alleviate  odd pair set finding.  """  single\_nodes **=** **[**k **for** k**,** order **in** graph**.**node\_orders**.**items**()** **if** order **==** 1**]**  **return** set**([**x **for** k **in** single\_nodes **for** x **in** graph**.**edges**.**values**()** \  **if** k **in** **(**x**.**head**,** x**.**tail**)])**  **def** build\_node\_pairs**(**graph**):**  """ Builds all possible odd node pairs. """  odd\_nodes **=** graph**.**odd\_nodes  **return** **[**x **for** x **in** itertools**.**combinations**(**odd\_nodes**,** 2**)]**  **def** build\_path\_sets**(**node\_pairs**,** set\_size**):**  """ Builds all possible sets of odd node pairs. """  **return** **(**x **for** x **in** itertools**.**combinations**(**node\_pairs**,** set\_size**)** \  **if** all\_unique**(**sum**(**x**,** **())))**  **def** unique\_pairs**(**items**):**  """ Generate sets of unique pairs of odd nodes. """  **for** item **in** items**[**1**:]:**  pair **=** items**[**0**],** item  leftovers **=** **[**a **for** a **in** items **if** a **not** **in** pair**]**  **if** leftovers**:**  # Python 2.7 version? Are they equivalent??  **for** tail **in** unique\_pairs**(**leftovers**):**  **yield** **[**pair**]** **+** tail  # Python 3 version:  # yield from ([pair] + tail for tail in unique\_pairs(leftovers))  **else:**  **yield** **[**pair**]**  **def** find\_node\_pair\_solutions**(**node\_pairs**,** graph**):**  """ Return path and cost for all node pairs in the path sets. """  node\_pair\_solutions **=** **{}**  **for** node\_pair **in** node\_pairs**:**  **if** node\_pair **not** **in** node\_pair\_solutions**:**  cost**,** path **=** dijkstra**.**find\_cost**(**node\_pair**,** graph**)**  node\_pair\_solutions**[**node\_pair**]** **=** **(**cost**,** path**)**  # Also store the reverse pair  node\_pair\_solutions**[**node\_pair**[::-**1**]]** **=** **(**cost**,** path**[::-**1**])**  **return** node\_pair\_solutions  **def** build\_min\_set**(**node\_solutions**):**  """ Order pairs by cheapest first and build a set by pulling  pairs until every node is covered. """  # Doesn't actually work... bad algorithm. What if last node  # has insane path cost?  odd\_nodes **=** set**([**x **for** pair **in** node\_solutions**.**keys**()** **for** x **in** pair**])**  # Sort by node\_pair cost  sorted\_solutions **=** sorted**(**node\_solutions**.**items**(),** key**=lambda** x**:**x**[**1**][**0**])**  path\_set **=** **[]**  **for** node\_pair**,** solution **in** sorted\_solutions**:**  **if** **not** all**(**x **in** odd\_nodes **for** x **in** node\_pair**):**  **continue**  path\_set**.**append**((**node\_pair**,** solution**))**  **for** node **in** node\_pair**:**  odd\_nodes**.**remove**(**node**)**  **if** **not** odd\_nodes**:** # We've got a pair for every node  **break**  **return** path\_set  **def** find\_minimum\_path\_set**(**pair\_sets**,** pair\_solutions**):**  """ Return cheapest cost & route for all sets of node pairs. """  cheapest\_set **=** **None**  min\_cost **=** float**(**'inf'**)**  min\_route **=** **[]**  **for** pair\_set **in** pair\_sets**:**  set\_cost **=** sum**(**pair\_solutions**[**pair**][**0**]** **for** pair **in** pair\_set**)**  **if** set\_cost **<** min\_cost**:**  cheapest\_set **=** pair\_set  min\_cost **=** set\_cost  min\_route **=** **[**pair\_solutions**[**pair**][**1**]** **for** pair **in** pair\_set**]**  **return** cheapest\_set**,** min\_route  **def** add\_new\_edges**(**graph**,** min\_route**):**  """ Return new graph w/ new edges extracted from minimum route. """  new\_graph **=** copy**.**deepcopy**(**graph**)**  **for** node **in** min\_route**:**  **for** i **in** range**(**len**(**node**)** **-** 1**):**  start**,** end **=** node**[**i**],** node**[**i **+** 1**]**  cost **=** graph**.**edge\_cost**(**start**,** end**)** # Look up existing edge cost  new\_graph**.**add\_edge**(**start**,** end**,** cost**,** **False)** # Append new edges  **return** new\_graph  **def** make\_eularian**(**graph**):**  """ Add necessary paths to the graph such that it becomes Eularian. """  **print(**'\tDoubling dead\_ends'**)**  dead\_ends **=** **[**x**.**contents **for** x **in** find\_dead\_ends**(**graph**)]**  graph**.**add\_edges**(**dead\_ends**)** # Double our dead-ends  **print(**'\tBuilding possible odd node pairs'**)**  node\_pairs **=** list**(**build\_node\_pairs**(**graph**))**  **print(**'\t\t({} pairs)'**.**format**(**len**(**node\_pairs**)))**  **print(**'\tFinding pair solutions'**)**  pair\_solutions **=** find\_node\_pair\_solutions**(**node\_pairs**,** graph**)**  **print(**'\t\t({} solutions)'**.**format**(**len**(**pair\_solutions**)))**  **print(**'\tBuilding path sets'**)**  pair\_sets **=** **(**x **for** x **in** unique\_pairs**(**graph**.**odd\_nodes**))**  **print(**'\tFinding cheapest route'**)**  cheapest\_set**,** min\_route **=** find\_minimum\_path\_set**(**pair\_sets**,** pair\_solutions**)**  **print(**'\tAdding new edges'**)**  **return** add\_new\_edges**(**graph**,** min\_route**),** len**(**dead\_ends**)** # Add our new edges  **if** \_\_name\_\_ **==** '\_\_main\_\_'**:**  **import** tests**.**run\_tests  tests**.**run\_tests**.**run**([**'eularian'**])** |

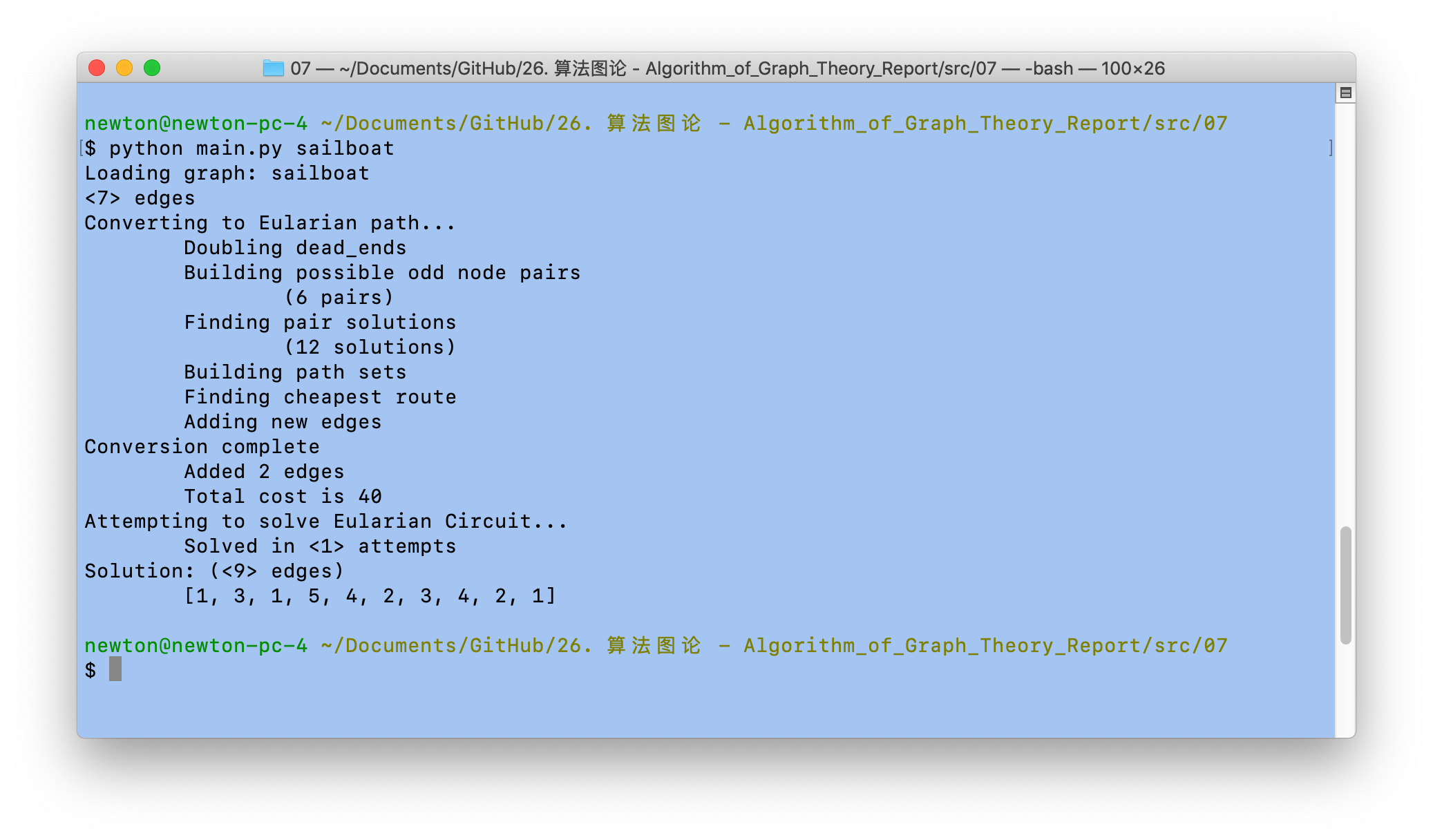
## 运行结果

由于该算法不仅仅要考虑结构，还要考虑权重，所以这里采用边表来进行建构。

(1,2,4), (1,3,3), (1,5,10), (2,3,2), (2,4,3), (3,4,3), (4,5,9)转换为图之后：



在命令行中运行可得如下结果：



# 参考文献

[1] 田丰, 张运清. 图与网络流理论 [M]. 2nd ed. 北京: 科学出版社, 2015.

[2] 管梅谷. 奇偶点图上作业法 [J]. 数学学报, 1960, 03): 263-6.

[3] https://github.com/supermitch/Chinese-Postman