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：**	

一、 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。

二、 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。

三、 实验平台

Windows 10 Pro 1803；
MacOS Mojave。

四、 算法设计

4.1 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$G=(V,\ E)$，如果存在一条简单链过图G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G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G有欧拉圈，则称图G为欧拉图（Eulerian Graph）。

定理 7.1（Euler 定理） 设图G是连通图，且边数$\|G\|>0$，则G是欧拉图，当且仅当G不含奇点[1]。

证明

(1) G是欧拉图$\Rightarrow$ G不含奇点

很显然，如果G是欧拉图，那么G本身就是一个欧拉圈。很显然，圈不含奇点。

(2) G不含奇点$\Rightarrow$ G是欧拉图

假设G是连通图，边数$\|G\|>0$，且G不含奇点，但是同时它不是欧拉图，同时假设G是所有这种图中含边最少的一个。由图连通、$\|G\|>0$以及都是偶数点可知，图的最小度$\delta(G)\geq 2$。所以图G含有圈（也可以这么认为：树是最简单的连通图，最小度为1，往树上随便加一条原先不存在的边，都会产生圈）。假设C是G中含有边数最多的简单圈，因为假设G不是欧拉图，所以根据假设可知：$G-E(C)$含有一个边数大于零的连通分图H，且所有点的度均为偶数，由于$\|H\|<\|G\|$，所以H必须含有欧拉圈，

否则就与G是所有这种图中含边最少的一个相矛盾；现断言：$V(C)\cap V(C')\neq\emptyset$。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$V(C)$与他们的交也为空，于是这些圈之间彼此点不交，这导致G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C边数还多的圈，这与C含边最多矛盾，于是这种图不存在，所以G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$G=(V,\,E)$，每个边e上有非负权值$w(e)$，要求出G的一个（未必是简单的）圈C，使得过每个边至少一次，并且使得圈C的总权重$w(C)=\sum_{e\in E(C)}w(e)$最小。

可以这样思考：加入连通图G不含奇点，那么这个图中有欧拉圈C，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G含有奇点，那么所求的圈必然过某条边多于一次，若在边$e=v_iv_j$上通过了k次，我们就在v_i与v_j之间添加$k-1$条新的边，并且令新添加的边的权重等于原来的边。对圈C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G命名为G^*，那么C就是多重图G^*中的欧拉圈。这时得到一个很显然的事实：$w(C)$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$G=(V,\,E)$，每个边e上有非负权$w(e)$，求$E_1\subseteq E$，满足条件：在G中，在E_1的每个边上添加一个重边，使得这样得到的图无奇点。称E_1为可行集（feasible set），并使得$w(E_1)$达到最小。

可行集E_1是最优集，当且仅当对G的每个初等圈C，都有

$$\sum_{e\in E_1\cap E(C)}w(e)\leq\sum_{e\in E(C)\setminus E_1}w(e)$$

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E_1是最优集，但这时存在一个初等圈C，使得

$$\sum_{e\in E_1\cap E(C)}w(e)>\sum_{e\in E(C)\setminus E_1}w(e)$$

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$E_2=E_1\oplus E(C)$，也就是将初等圈C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，E_2也是一个可行解，且$w(E_2)<w(E_1)$，这与E_1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E_1与E_2时，只需证明两者的权重是相等的。对每个点v，其关联边在E_1与E_2中的数目是同奇偶的（否则该点在E_1与E_2的分别作用下度不同）。所以$G[E_1 \oplus E_2]$不含奇点，进而$G[E_1 \oplus E_2]$可以剖分为若干个子集，使得每个子集都是初等圈，并设$C_1, C_2, \cdots, C_k$是所有这样的圈，于是由最优条件，任意的i都有

$$\sum_{e \in E_1 \cap E(C_i)} w(e) \leq \sum_{e \in E(C_i) \setminus E_1} w(e) = \sum_{e \in E_2 \cap E(C_i)} w(e)$$

反过来就有

$$\sum_{e \in E_2 \cap E(C_i)} w(e) \leq \sum_{e \in E(C_i) \setminus E_2} w(e) = \sum_{e \in E_1 \cap E(C_i)} w(e)$$

所以

$$\sum_{e \in E_1 \cap E(C_i)} w(e) = \sum_{e \in E_2 \cap E(C_i)} w(e)$$

综上所述，充要性得证。

□

4.2 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FLEURY算法、两点之间求最短路的PATH算法。

Algorithm 奇偶点图上作业法求解中国邮递员问题，记此算法为POSTMAN-ROUTE

Input $G = (V, E)$，允许这个图为任意的图

Output 图G的一个圈C，记$C = \mathrm{POSTMAN\text{-}ROUTE}(G)$

Begin // 计数并存储奇度点

list $L = \emptyset$

Step 1 **for each** vertex $v \in V$:

  if $d_G(v) \bmod 2 == 1$:

    $L.\mathbf{append}(v)$

Step 2 **if** $|L| == 0$:

  $C = \mathrm{FLEURY}(G)$

  goto End

else:

  for $i = 1$ **through** $|L| - 1$:

    for each edge $e \in \mathrm{PATH}\big(L.\mathbf{get}(i), \mathrm{L}.\mathbf{get}(i+1)\big)$:

      edge $temp = e$

$temp.\textbf{color} = \textbf{Black}$
$G.\textbf{addEdge}(temp)$
$E_1 = \emptyset$
for each vertex $v \in V$:
flag $= 0$
for each vertex $u \in V \backslash$错误! 未定义书签。:
for each edge $e \in E(u,\ v)$:
if $e.\textbf{color} == \textbf{Black}$ **and** flag $== 0$:
flag $= 1$
$E_1.\textbf{append}(e)$
else if $e.\textbf{color} == \textbf{Black}$:
$G.\textbf{deleteEdge}(e)$

Step 3

// 圈的对称差
for each vertex $v \in V$:
for each vertex $u \in V \backslash v$:
$C = G[(u,\ v)] \cup \mathrm{PATH}(G - E(u,\ v);\ u,\ v)$
if IS-CYCLE$(C) ==$ **True**:
$in = 0$
$out = 0$
for each edge $e \in E_1 \cap E(C)$:
$in = in + w(e)$
for each edge $e \in E(C) \backslash E_1$:
$out = out + w(e)$
if $in \geq out$:
$E_1 = E_1 \oplus E(C)$
$C = \mathrm{FLEURY}(G)$

End

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v的反圈$\boldsymbol{\Phi}_G(v)$，从$\boldsymbol{\Phi}_G(v)$中任取一个点u，在$G[E-(u,\ v)]$中寻找从u到的路，这条路P如果找到，那么$G[P \cup \{(u,\ v)\}]$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。

五、 程序代码

5.1 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   Functions relating to Eularian graphs.
3
4   This module contains functions relating to the identification
5   and solution of Eularian trails and Circuits.
6
7   """
8   import copy
9   import itertools
10  import random
11  import sys
12  from time import clock
13
14  from . import dijkstra, my_math
15  from .my_iter import all_unique, flatten_tuples
16
17
18  def fleury_walk(graph, start=None, circuit=False):
19      """
20      Return an attempt at walking the edges of a graph.
21
22      Tries to walk a Circuit by making random edge choices. If the route
23      dead-ends, returns the route up to that point. Does not revisit
24      edges.
25
26      If circuit is True, route must start & end at the same node.
27      """
28      visited = set()  # Edges
29
30      # Begin at a random node unless start is specified
31      node = start if start else random.choice(graph.node_keys)
32
33      route = [node]
34      while len(visited) < len(graph):
35          # Fleury's algorithm tells us to preferentially select non-bridges
36          reduced_graph = copy.deepcopy(graph)
37          reduced_graph.remove_edges(visited)
38          options = reduced_graph.edge_options(node)
39          bridges = [k for k in options.keys() if reduced_graph.is_bridge(k)]
40          non_bridges = [k for k in options.keys() if k not in bridges]
41          if non_bridges:
42              chosen_path = random.choice(non_bridges)
43          elif bridges:
44              chosen_path = random.choice(bridges)
45          else:
46              break  # Reached a dead-end, no path options
47          next_node = reduced_graph.edges[chosen_path].end(node)  # Other end
48
49          visited.add(chosen_path)  # Never revisit this edge
50
```

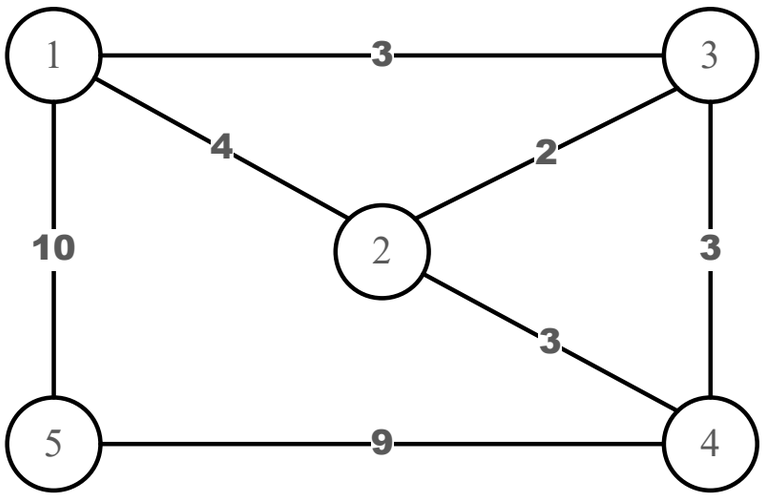
```
51          route.append(next_node)
52          node = next_node
53
54      return route
55
56
57  def eularian_path(graph, start=None, circuit=False):
58      """
59      Return an Eularian Trail or Eularian Circuit through a graph, if found.
60
61      Return the route if it visits every edge, else give up after 1000 tries.
62
63      If `start` is set, force start at that Node.
64      """
65      for i in range(1, 1001):
66          route = fleury_walk(graph, start, circuit)
67          if len(route) == len(graph) + 1:  # We visited every edge
68              return route, i
69      return [], i  # Never found a solution
70
71
72  def find_dead_ends(graph):
73      """
74      Return a list of dead-ended edges.
75
76      Find paths that are dead-ends. We know we have to double them, since
77      they are all order 1, so we'll do this ahead of time to alleviate
78      odd pair set finding.
79
80      """
81      single_nodes = [k for k, order in graph.node_orders.items() if order == 1]
82      return set([x for k in single_nodes for x in graph.edges.values() \
83              if k in (x.head, x.tail)])
84
85
86  def build_node_pairs(graph):
87      """ Builds all possible odd node pairs. """
88      odd_nodes = graph.odd_nodes
89      return [x for x in itertools.combinations(odd_nodes, 2)]
90
91
92  def build_path_sets(node_pairs, set_size):
93      """ Builds all possible sets of odd node pairs. """
94      return (x for x in itertools.combinations(node_pairs, set_size) \
95              if all_unique(sum(x, ())))
96
97
98  def unique_pairs(items):
99      """ Generate sets of unique pairs of odd nodes. """
```

```
100     for item in items[1:]:
101         pair = items[0], item
102         leftovers = [a for a in items if a not in pair]
103         if leftovers:
104             # Python 2.7 version? Are they equivalent??
105             for tail in unique_pairs(leftovers):
106                 yield [pair] + tail
107             # Python 3 version:
108             # yield from ([pair] + tail for tail in unique_pairs(leftovers))
109         else:
110             yield [pair]
111 
112 
113 def find_node_pair_solutions(node_pairs, graph):
114     """ Return path and cost for all node pairs in the path sets. """
115     node_pair_solutions = {}
116     for node_pair in node_pairs:
117         if node_pair not in node_pair_solutions:
118             cost, path = dijkstra.find_cost(node_pair, graph)
119             node_pair_solutions[node_pair] = (cost, path)
120             # Also store the reverse pair
121             node_pair_solutions[node_pair[::-1]] = (cost, path[::-1])
122     return node_pair_solutions
123 
124 
125 def build_min_set(node_solutions):
126     """ Order pairs by cheapest first and build a set by pulling
127     pairs until every node is covered. """
128     # Doesn't actually work... bad algorithm. What if last node
129     # has insane path cost?
130     odd_nodes = set([x for pair in node_solutions.keys() for x in pair])
131     # Sort by node_pair cost
132     sorted_solutions = sorted(node_solutions.items(), key=lambda x:x[1][0])
133     path_set = []
134     for node_pair, solution in sorted_solutions:
135         if not all(x in odd_nodes for x in node_pair):
136             continue
137         path_set.append((node_pair, solution))
138         for node in node_pair:
139             odd_nodes.remove(node)
140         if not odd_nodes:  # We've got a pair for every node
141             break
142     return path_set
143 
144 
145 def find_minimum_path_set(pair_sets, pair_solutions):
146     """ Return cheapest cost & route for all sets of node pairs. """
147     cheapest_set = None
148     min_cost = float('inf')
```

```
149     min_route = []
150     for pair_set in pair_sets:
151         set_cost = sum(pair_solutions[pair][0] for pair in pair_set)
152         if set_cost < min_cost:
153             cheapest_set = pair_set
154             min_cost = set_cost
155             min_route = [pair_solutions[pair][1] for pair in pair_set]
156 
157     return cheapest_set, min_route
158 
159 
160 def add_new_edges(graph, min_route):
161     """ Return new graph w/ new edges extracted from minimum route. """
162     new_graph = copy.deepcopy(graph)
163     for node in min_route:
164         for i in range(len(node) - 1):
165             start, end = node[i], node[i + 1]
166             cost = graph.edge_cost(start, end)  # Look up existing edge cost
167             new_graph.add_edge(start, end, cost, False)  # Append new edges
168     return new_graph
169 
170 
171 def make_eularian(graph):
172     """ Add necessary paths to the graph such that it becomes Eularian. """
173     print('\tDoubling dead_ends')
174     dead_ends = [x.contents for x in find_dead_ends(graph)]
175     graph.add_edges(dead_ends)  # Double our dead-ends
176 
177     print('\tBuilding possible odd node pairs')
178     node_pairs = list(build_node_pairs(graph))
179     print('\t\t({} pairs)'.format(len(node_pairs)))
180 
181     print('\tFinding pair solutions')
182     pair_solutions = find_node_pair_solutions(node_pairs, graph)
183     print('\t\t({} solutions)'.format(len(pair_solutions)))
184 
185     print('\tBuilding path sets')
186     pair_sets = (x for x in unique_pairs(graph.odd_nodes))
187 
188     print('\tFinding cheapest route')
189     cheapest_set, min_route = find_minimum_path_set(pair_sets, pair_solutions)
190     print('\tAdding new edges')
191     return add_new_edges(graph, min_route), len(dead_ends)  # Add our new edges
192 
193 
194 if __name__ == '__main__':
195     import tests.run_tests
196     tests.run_tests.run(['eularian'])
```

5.2 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：

```
07 — ~/Documents/GitHub/26. 算法图论 - Algorithm_of_Graph_Theory_Report/src/07 — -bash — 100×26

newton@newton-pc-4 ~/Documents/GitHub/26. 算法图论 - Algorithm_of_Graph_Theory_Report/src/07
$ python main.py sailboat
Loading graph: sailboat
<7> edges
Converting to Eularian path...
        Doubling dead_ends
        Building possible odd node pairs
                (6 pairs)
        Finding pair solutions
                (12 solutions)
        Building path sets
        Finding cheapest route
        Adding new edges
Conversion complete
        Added 2 edges
        Total cost is 40
Attempting to solve Eularian Circuit...
        Solved in <1> attempts
Solution: (<9> edges)
        [1, 3, 1, 5, 4, 2, 3, 4, 2, 1]

newton@newton-pc-4 ~/Documents/GitHub/26. 算法图论 - Algorithm_of_Graph_Theory_Report/src/07
$
```

六、 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