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：欧拉图判断与寻找欧拉回路 | 学号：20151910042 | 上机实践日期：2018-12-31 |
| 上机实践编号：5 | 组号： |  |

# 实验目的

1. 了解欧拉图的来历、定义以及图论表述；
2. 能快速写出求最短路的最小插点问题的动态规划算法。

## 实验内容

1. 写出判定一个给定的图是否是欧拉图的算法；
2. 写出寻找含有欧拉回路的图的欧拉回路的算法。

# 实验平台

Windows 10 Pro 1803；

MacOS Mojave。

# 算法设计

## 图的一般概念

如果一个图不含有环（loop）和多重边（multiple edge），则称这种图为简单图（simple graph）。但是如果图中允许有环和多重边，那么这种图被称为伪图（pseudograph）。

给定伪图，若存在一条简单链过图的每条边一次并且仅仅一次，则称这个链为欧拉链（Eulerian chain）。特别地，如果欧拉链变成了一个圈，那么称这个圈为欧拉圈（Eulerian cycle）。如果一个图有欧拉圈，则这个图是欧拉图（Eulerian graph）。如果连通伪图是欧拉图，当且仅当中不含奇点。

Fleury提供了一个有效的在含欧拉圈的图中找出欧拉圈的算法[1]。

## Fleury算法

在本算法的叙述中，以表示在第步得到的简单链。在这个算法中使用了连通性判断算法-

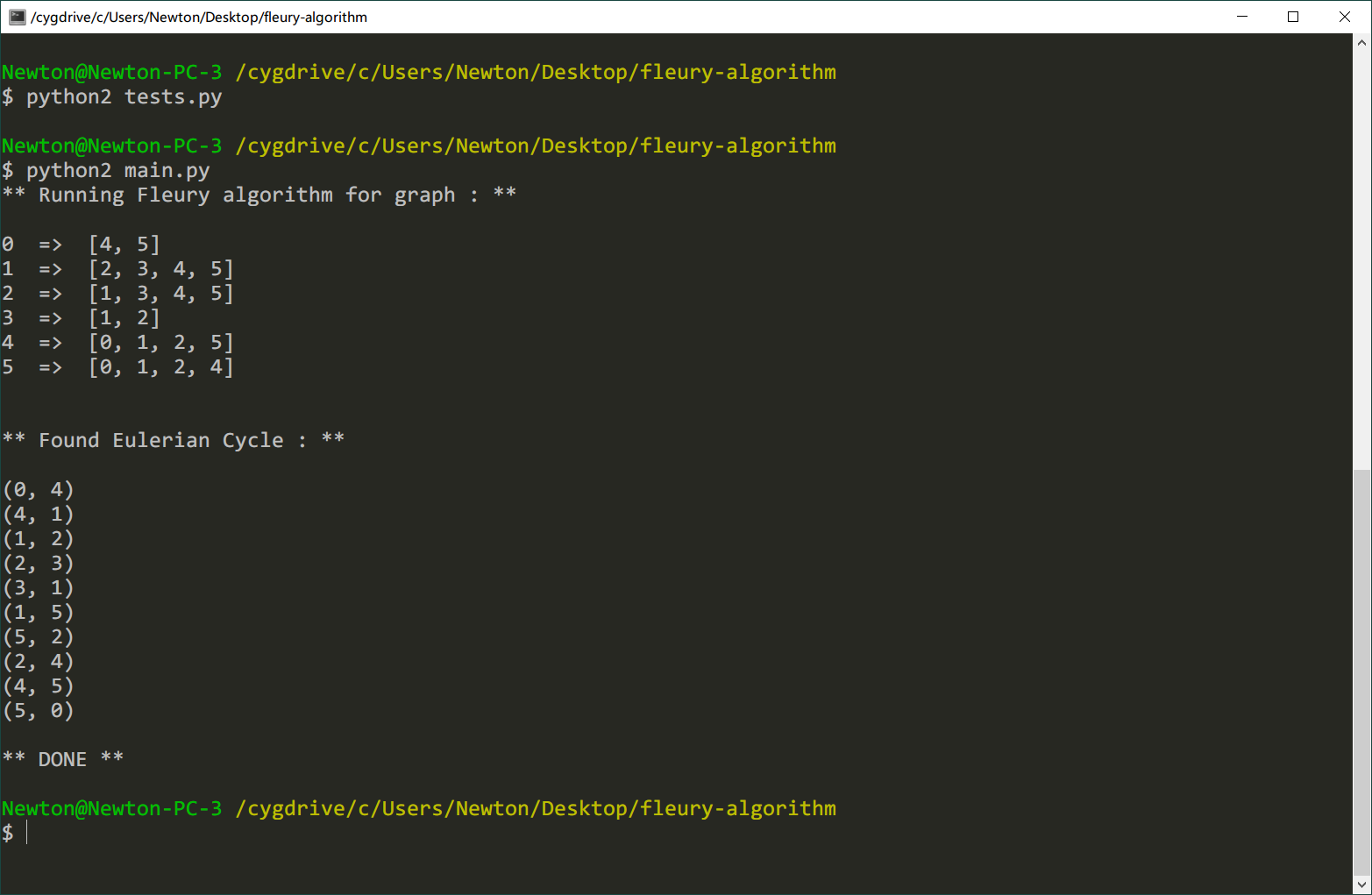
|  |  |
| --- | --- |
| **Algorithm**  **Input**  **Output**  **Begin**  **Step 1**  **Step 2**  **End** | 求含欧拉圈图的欧拉圈，记此算法为  ，允许这个图为任意的图  图的一个欧拉圈，记  如果不含欧拉圈，就输出“This graph is not Eulerian graph”  // 选择初始点  **while** **true**:    **for** **each** edge :  **if** -:              **break**  **if** :  **if** :    **else**:  **output** This graph is not Eulerian graph |

通过这个算法，可以比较简单地判定一个图是否是欧拉图，如果是欧拉图，还可以输出欧拉回路。

# 程序代码

## 运行结果

以如图所示为例子，可以看到输出的结果。



## 程序代码

使用Python 2进行编码。具体的程序非常复杂，这里仅列举核心的算法部分，具体代码参见相应的文件夹。

|  |  |
| --- | --- |
| 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 | # Fleury's Algorithm implementation  import copy  class FleuryException(Exception):  def \_\_init\_\_(self, message):  super(FleuryException, self).\_\_init\_\_(message)  self.message = message  class Fleury:  COLOR\_WHITE = 'white'  COLOR\_GRAY = 'gray'  COLOR\_BLACK = 'black'  def \_\_init\_\_(self, graph):  self.graph = graph  def run(self):  print "\*\* Running Fleury algorithm for graph : \*\* \n"  for v in self.graph:  print v, ' => ', self.graph[v]  print '\n'  output = None  try:  output = self.fleury(self.graph)  except FleuryException as (message):  print message  if output:  print '\*\* Found Eulerian Cycle : \*\*\n'  for v in output:  print v  print '\n\*\* DONE \*\*'  def is\_connected(self, G):  start\_node = list(G)[0]  color = {}  iterator = 0;  for v in G:  color[v] = Fleury.COLOR\_WHITE  color[start\_node] = Fleury.COLOR\_GRAY  S = [start\_node]  while len(S) != 0:  u = S.pop()  for v in G[u]:  if color[v] == Fleury.COLOR\_WHITE:  color[v] = Fleury.COLOR\_GRAY  S.append(v)  color[u] = Fleury.COLOR\_BLACK  return list(color.values()).count(Fleury.COLOR\_BLACK) == len(G)  def even\_degree\_nodes(self, G):  even\_degree\_nodes = []  for u in G:  if len(G[u]) % 2 == 0:  even\_degree\_nodes.append(u)  return even\_degree\_nodes  def is\_eulerian(self, even\_degree\_odes, graph\_len):  return graph\_len - len(even\_degree\_odes) == 0  def convert\_graph(self, G):  links = []  for u in G:  for v in G[u]:  links.append((u, v))  return links  def fleury(self, G):  edn = self.even\_degree\_nodes(G)    if not self.is\_eulerian(edn, len(G)):  raise FleuryException('This is not an Eulerian graph!')  g = copy.copy(G)  cycle = []    u = edn[0]  while len(self.convert\_graph(g)) > 0:  current\_vertex = u  for u in list(g[current\_vertex]):  g[current\_vertex].remove(u)  g[u].remove(current\_vertex)  bridge = not self.is\_connected(g)  if bridge:    g[current\_vertex].append(u)  g[u].append(current\_vertex)  else:  break  if bridge:    g[current\_vertex].remove(u)  g[u].remove(current\_vertex)  g.pop(current\_vertex)  cycle.append((current\_vertex, u))  return cycle |

# 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] <https://github.com/dkulig/fleury-algorithm>