

关于分析民航城市重要性的上机报告

罗颖

2023 年 4 月 4 日

目录

1	问题提出	2
1.1	问题背景	2
1.1.1	全国民航不同城市的基本特点	2
1.2	提出问题	3
1.2.1	建立简单的问题模型	3
1.3	问题假设	4
1.3.1	对于问题的基本假设	4
1.4	数据采集	4
1.4.1	如何采集和保存数据	4
2	算法原理	5
2.1	Hits 算法	5
2.1.1	Hits 算法的由来	5
2.1.2	Hits 算法的思想	5
2.1.3	如何在问题中使用 Hits 算法	6
2.2	选择排序算法	7
2.2.1	选择排序的思想	7
2.2.2	选择排序的时间复杂度	7
2.2.3	在问题中使用选择排序算法	7
3	问题解决	8
3.1	Python 实现	8
3.1.1	使用到的基础知识	8
3.1.2	导入数据	8
3.1.3	分析数据	9
3.1.4	利用 Hits 算法处理数据	10

3.1.5	利用选择排序算法处理数据	10
3.2	结果分析与验证	13
3.2.1	结果分析	13
3.2.2	结果验证	14
4	方法完善与改进	14
4.1	城市数据的改进	14
4.2	算法效率的改进	14
4.3	如何跳出循环	14
4.3.1	第二种方式	14
4.3.2	两种方式的差异	16
4.4	数据分析	17
4.4.1	对 Hub 值的分析	17
4.4.2	综合分析	17

1 问题提出

1.1 问题背景

1.1.1 全国民航不同城市的基本特点

民用航空,是指使用航空器从事除了国防、警察和海关等国家航空活动以外的航空活动,民用航空活动是航空活动的一部分,同时以“使用”航空器界定了它和航空制造业的界限,用“非军事等性质”表明了它和军事航空等国家航空活动不同。

20 世纪 50 年代以来,民用航空的服务范围不断扩大,成为一个国家的重要经济部门。商业航空的发展主要表现在客货运输量的迅速增长,定期航线密布于世界各大洲。由于快速、安全、舒适和不受地形限制等一系列优点,商业航空在交通运输结构中占有独特的地位,它促进了国内和国际贸易、旅游和各种交往活动的发展,使在短期内开发边远地区成为可能。

到如今,中国许多城市的民航服务都已经非常完善,部分城市的民航建设已经成为了城市的名片,因此,掌握主要城市的民航有关指数,有利于促进中国民航整体事业的发展和规划。

1.2 提出问题

1.2.1 建立简单的问题模型

现在我们来研究中国主要城市的民航水准，为了便于分析数据，我们将城市与一个个编号对应，假设我们研究的城市为：

$$A_k, k = 1, 2, 3...$$

对于民用航班，如 15074 次航班，我们主要分析其四个指标属性：
始发站城市的编号：

$$A_i, i = 1, 2, 3...$$

目的地城市的编号：

$$A_j, j = 1, 2, 3...$$

航班里程：

$$Flight_Mileage(A_i \rightarrow A_j), i, j = 1, 2, 3..., i \neq j$$

每周的班次：

$$Count(A_i \rightarrow A_j), i, j = 1, 2, 3..., i \neq j$$

当该航班在中国不同城市往返时，统计其一周之内从一个城市飞往另一个城市的班次之和：

$$Numbers(A_i \rightarrow A_j) = \sum Count(A_i \rightarrow A_j), i, j = 1, 2, 3..., i \neq j$$

此处的

$$Numbers(A_i \rightarrow A_j), i, j = 1, 2, 3..., i \neq j$$

是一个非常重要的指标，对于一个城市 A_{k_0} 而言，与之有联系的 Numbers 指标属性有两种：

$$Numbers(A_{k_0} \rightarrow A_k), j = 1, 2, 3..., k \neq k_0$$

$$Numbers(A_k \rightarrow A_{k_0}), i = 1, 2, 3..., k \neq k_0$$

前者表明从该城市飞往其他城市的航班数，后者表明其他城市飞往该城市的航班数。

1.3 问题假设

1.3.1 对于问题的基本假设

基于上述模型，我们做出以下假设：

1. 采集数据过程中，忽略因意外导致的航班更改，取消等情况
2. 各个航班的航线相互独立：
3. 航班仅在所研究的城市间往返：

$$flight(A_i \rightarrow A_j), i, j = 1, 2, 3, \dots, n, i \neq j$$

式中 n 为所研究的城市数。

4. 忽略航班飞往除所研究的城市外其他城市（如纽约）而产生的数据影响

1.4 数据采集

1.4.1 如何采集和保存数据

现通过航班查询等手段，统计了中国的 185 个城市间航班的往返信息，主要包括前述的四个指标属性：始发站的编号、目的地的编号、航班里程（单位 km）、每周的班次。

将数据存至 A.csv 文件中，文件中四个指标属性用‘,’隔开，分别为：

始发站的编号： $A_i, i = 1, 2, 3, \dots$

目的地的编号： $A_j, j = 1, 2, 3, \dots$

航班里程： $Flight_Mileage(A_i \rightarrow A_j), i, j = 1, 2, 3, \dots, i \neq j$

每周的班次： $Count(A_i \rightarrow A_j), i, j = 1, 2, 3, \dots, i \neq j$

同时为了便于数据处理，作如下映射：

城市编号 \rightarrow 城市名称,

$$A_i \rightarrow Name_i, i = 1, 2, 3, \dots$$

并将其存入 City_name.csv 文件中。

2 算法原理

2.1 Hits 算法

2.1.1 Hits 算法的由来

HITS 算法是由康奈尔大学 (Cornell University) 的 Jon Kleinberg 博士于 1997 年首先提出的, 为 IBM 公司阿尔马登研究中心 (IBM Almaden Research Center) 的名为 “CLEVER” 的研究项目中的一部分。

按照 HITS 算法, 用户输入关键词后, 算法对返回的匹配页面计算两种值, 一种是枢纽值 (Hub Scores), 另一种是权威值 (Authority Scores), 这两种值是互相依存、互相影响的。

所谓枢纽值, 指的是页面上所有导出链接指向页面的权威值之和。权威值是指所有导入链接所在的页面中枢纽之和。

Hits 算法是一种重要的网页重要性分析算法。

在限定范围之后根据网页的出度和入度建立一个矩阵, 通过矩阵的迭代运算和定义收敛的阈值不断对两个向量 Authority 和 Hub 值进行更新直至收敛。

2.1.2 Hits 算法的思想

对于初始集合 G_0 , 用矩阵 M 表示 G_0 中网页之间的关系, 对于矩阵 M 的 (i, j) 元, 取

$$m_{(i,j)} = 1$$

表示网页 i 指向网页 j , 否则为 0。用

$$\vec{H} = (h_1, h_2, \dots, h_i, \dots), \quad i = 1, 2, 3, \dots$$

表示所有页面的 Hub 值, 其中第 i 个分量 h_i 表示网页 i 的 Hub 值, 用

$$\vec{A} = (a_1, a_2, \dots, a_i, \dots), \quad i = 1, 2, 3, \dots$$

表示所有页面的 Authority 值, 其中第 i 个分量 a_i 表示网页 i 的 Authority 值。初始时, 可设所有页面的 Hub 值和 Authority 值都为 1 (或者随机生成):

$$\vec{H}_0 = (1, 1, \dots, 1)$$

$$\vec{A}_0 = (1, 1, \dots, 1)$$

对于前述矩阵 M , 依次计算

$$A_1 = M^T H_0,$$

$$H_1 = M A_1,$$

不断重复下去，一般的，我们有：

$$A_i = M^T H_{i-1}, \quad i = 1, 2, 3, \dots$$

$$H_i = M A_i, \quad i = 1, 2, 3, \dots$$

为了防止迭代次数过多导致数据过大溢出，采用归一化操作：

$$A_i = \frac{A_i}{\sum_{i=1} (A_i)}, \quad i = 1, 2, 3, \dots$$

$$H_i = \frac{H_i}{\sum_{i=1} (H_i)}, \quad i = 1, 2, 3, \dots$$

设置一个阈值或指定一个迭代次数用于结束迭代，如指定阈值为 ε ，当

$$|A_i - A_{i-1}| < \varepsilon, \quad i = 1, 2, 3, \dots$$

且

$$|H_i - H_{i-1}| < \varepsilon, \quad i = 1, 2, 3, \dots$$

时，迭代结束。得到 G_0 中各个网页的 Hub 值和 Authority 值。

或者指定一个迭代次数 k ， k 次迭代后的

$$\vec{H}_k, \vec{A}_k,$$

即近似为 G_0 中各个网页的 Hub 值和 Authority 值。（证明略）

2.1.3 如何在问题中使用 Hits 算法

一方面，结合前述的问题模型和 hits 算法来构建矩阵 M ：

从采集的数据（A.csv）里分析，对于矩阵 M 的 (i, j) 元，取

$$M(i, j) = \begin{cases} \text{Numbers}(A_i \rightarrow A_j), & i, j = 1, 2, \dots, n, \quad i \neq j \\ 0, & i = j \end{cases} \quad (1)$$

与网页分析不同的是，此处不采用“有无链接关系”0 或 1 填充矩阵 M ，而是采用航班总次数作为权重来构建矩阵 M 。

另一方面，借助 Hits 算法，作如下对应：

Authority 值 \rightarrow 其他城市飞往该城市的航班数，

Hub 值 \rightarrow 该城市飞往其他城市的航班数，

在民航领域中，Authority 值和 Hub 值是两个比较重要的指标，通过对这两个指标的分析 and 计算，可以帮助了解不同城市的民航水准。

2.2 选择排序算法

2.2.1 选择排序的思想

选择排序 (Selection sort) 是一种简单直观的排序算法。它的工作原理是：第一次从待排序的数据元素中选出最小（或最大）的一个元素，存放在序列的起始位置，然后再从剩余的未排序元素中寻找到最小（大）元素，然后放到已排序的序列的末尾。以此类推，直到全部待排序的数据元素的个数为零。

选择排序属于不稳定排序。这是显然的。举个例子，对于序列

$$5, 8, 5, 2, 9,$$

第一遍选择第 1 个元素 5 会和 2 交换，那么原序列中两个 5 的前后顺序就被破坏了。

2.2.2 选择排序的时间复杂度

关于时间复杂度。选择排序的交换次数介于

$$0 \sim (n - 1)$$

之间，选择排序的比较次数介于

$$0 \sim \frac{n(n - 1)}{2}$$

之间。选择排序的赋值次数介于

$$0 \sim 3(n - 1)$$

之间。比较次数为 $O(n^2)$ ，与关键字的初始状态无关，总的比较次数为

$$N = (n - 1) + (n - 2) + \dots + 1 = \frac{n(n - 1)}{2}$$

交换次数为 $O(n)$ 。最好的情况是，已经有序，交换 0 次；最坏的情况是，交换 $n-1$ 次，逆序交换 $\frac{n}{2}$ 次。交换次数比冒泡排序少，由于交换所需 CPU 时间比较所需的 CPU 时间多， n 值较小时，选择排序比冒泡排序快。

2.2.3 在问题中使用选择排序算法

如前所述，当获取了某个城市的 Authority 值和 Hub 值后，数据本身无法看出规律，需对其进行排序，此处使用选择排序。理论上，Authority 值和 Hub 值均可以在一定程度上反应某个城市的民航水准，本报告主要依据 Authority 值对所有城市的民航水准进行比较，进而把握各个城市的民航水准。

3 问题解决

3.1 Python 实现

由于 Hits 算法中涉及到矩阵运算，此处我们采用 Python 实现读取数据，并借助 Numpy 来处理数据（矩阵运算）。

3.1.1 使用到的基础知识

1. 借助 Python 的 csv 包中的 read 类导入 csv 数据；
2. 借助 Python 中有关的文件操作类读取数据；
3. 借助 Python 中 math 包进行科学计算；
4. 借助 Numpy 进行矩阵迭代运算。

3.1.2 导入数据

首先导入 csv 包中的 reader 类：

```
from csv import reader
```

reader 类用于读取 csv 文件中的数据：

```
# 打开City_name.csv文件
with open('City_name.csv', 'rt', encoding='UTF-8') as f:
    City_name = reader(f, delimiter=',')
    City_list = list(City_name)
```

```
# 打开A.csv文件
with open('A.csv', 'rt', encoding='UTF-8') as f:
    data = list(reader(f, delimiter=','))
```

变量 City_list,data 均是一个列表：

```
[['0', '万州'], ['1', '三亚'], ['2', '上海'], ['3', '东营'], ['4', '中卫'], ['5', '临沂'], ['6', '临沧'], ['7', '丹东'], ['8', '丽江'], ['9', '义乌'], ['10', '乌兰浩特'], ['11', '乌海'], ['12', '乌鲁木齐'], ['13', '二连浩特'], ['14', '井冈山'], ['15', '伊宁'], ['16', '伊春'], ['17', '佛山'], ['18', '佳木斯'], ['19', '保山'],
```

```
[['170', '26', '3049', '4'], ['170', '159', '2697', '1'], ['170', '95', '4360', '7'], ['170', '95', '4360', '7'], ['170', '40', '470', '2'], ['170', '40', '470', '2'], ['170', '2', '4506', '7'], ['170', '12', '857', '7'], ['170', '12', '857', '7'], ['170', '12', '857', '7'], ['170', '12', '857', '7'], ['170', '12', '857', '7'], ['170', '12', '857', '6'], ['170', '12', '857', '1'],
```

至此数据已全部导入，并存入了 City_list,data 两个变量之中。

3.1.3 分析数据

分析 City_list,data 中的数据，用于构建矩阵 M：

```
# 生成一个矩阵统计任意两个城市间每周总共有多少次航班
A = np.zeros((185, 185))

# 统计
for item in data:
    A[int(item[0])][int(item[1])] += int(item[3])
```

以航班总次数作为权重处理矩阵 M ，上图中的变量 A 便是处理后的矩阵 M 。

3.1.4 利用 Hits 算法处理数据

接着，对于矩阵 M ，利用 Hits 算法，迭代计算其 Hub 值和 Authority 值，同时每迭代一次，便归一化一次，得到 k 次迭代后的 $Hub[k], Authority[k]$ 。

现取

$$k = 200$$

即迭代 200 次后，获得每个城市的近似 Hub 值和 Authority 值：

```
# Hits算法:
for i in range(200):
    # 迭代计算
    Authority0 = (A.T @ Hub)
    Hub0 = (A @ Authority)
    # 归一
    Authority = Authority0 / sum(i for i in Authority0)
    Hub = Hub0 / sum(i for i in Hub0)
```

3.1.5 利用选择排序算法处理数据

通过 Hits 算法，已经得到处理后的 Hub 值和 Authority 值。

下面仅针对 Authority 值进行分析。

首先，

```
# 通过enumerate()方法获取索引号和值构成的元组
result = list(enumerate(Authority))
```

将 Hits 算法处理后的数据转化为易于处理的列表 $result$ ，输出 $result$ (部分)：

```
[(0, 0.0010490589612719402), (1, 0.019616295276477628),
 (2, 0.0581415271081423), (3, 0.0011815756614951382),
 (4, 0.0003238708793217662), (5, 0.001984004066540052),
 (6, 0.0002832883378675286), (7, 0.0006956195680741614),
 (8, 0.008254984002292265), (9, 0.0029623516923105173),
```

可以看到，编号为 0 的城市的权重值 Authority 为 0.0010490589612719402，单从这些杂乱的数据无法获得有效的信息。现考虑将所有城市按 Authority 值的大小进行排序，此处采用选择排序：

```
# 对result做从小到大的排序处理
# 此处使用选择排序
for i in range(185):
    for j in range(i + 1, 185):
        if result[i][1] > result[j][1]:
            temp = result[i]
            result[i] = result[j]
            result[j] = temp
```

排序后，继续分析 result(部分)：

```
[(17, 1.948073650010119e-06), (101, 2.584994767274461e-05),
 (63, 6.17820389410525e-05), (47, 6.17820389410525e-05),
 (172, 6.886928265178462e-05), (174, 7.50032841656447e-05),
 (48, 7.728373615538106e-05), (171, 7.94340500670675e-05),
 (155, 0.000123564077882105), (71, 0.000123564077882105),
```

可以看到，Authority 值最小的城市的编号分别为 17, 101, 63, 47, ...，为了方便观察，现将 result 每个元组的第一个元素（城市编号）修改映射成城市名称。

由于元组不可修改，先转化为列表：

```
for i in range(185):
    result[i] = list(result[i])
```

再映射：

```
# 将Authority值与城市姓名对应
for i in range(185):
    result[i][0] = City_list[int(result[i][0))][1]
```

得到最后的处理结果，输出 result，可以看到以下结果：

```
[['佛山', 1.948073650010119e-06], ['梧州', 2.584994767274461e-05],
['富蕴', 6.17820389410525e-05], ['塔城', 6.17820389410525e-05],
['阿尔山', 6.886928265178462e-05], ['阿里', 7.50032841656447e-05],
```

3.2 结果分析与验证

3.2.1 结果分析

利用 python 获取排名最高的 20 个城市和排名最低的 5 个城市：

```
# 最高的20个城市的索引号 (注意顺序)
rank_list_high = [i for i in range(165,185)]

print("排名最高的20个城市及其Authority值:")
# 循环输出 (Authority值保留六位小数), 注意输出顺序:
for i in rank_list_high:
    print("第%d名: %s; Authority值: %.6f" %
          ((i-164), result[185-(i-164)][0], result[185-(i-164)][1]))

# 最低的5个城市的索引号
rank_list_low = [i for i in range(5)]

print("*****")

print("排名最低的5个城市及其Authority值:")
# 循环输出 (Authority值保留八位小数)
for i in rank_list_low:
    print("倒数第%d名: %s; Authority值: %.8f" % ((5-i), result[4-i][0], result[4-i][1]))
```

依据习惯，排名最高的 20 个城市正向输出，排名最低的 5 个城市反向输出，运行程序，输出结果如下（部分）：



```
排名最高的20个城市及其Authority值:  
第1名: 上海; Authority值:0.058142  
第2名: 北京; Authority值:0.048296  
第3名: 深圳; Authority值:0.043459  
第4名: 广州; Authority值:0.042614  
第5名: 昆明; Authority值:0.036857  
第6名: 重庆; Authority值:0.035902  
第7名: 成都; Authority值:0.035892  
第8名: 西安; Authority值:0.035527  
第9名: 杭州; Authority值:0.027961  
第10名: 厦门; Authority值:0.025758  
第11名: 郑州; Authority值:0.024568  
第12名: 哈尔滨; Authority值:0.023356
```

3.2.2 结果验证

中国城市发展水平不一，北上广等城市发展水平较高，上述结果符合中国当前城市民航水准的实情。

4 方法完善与改进

4.1 城市数据的改进

在 City_name.csv 文件中，‘,’分割的两个数据，后一个数据与‘,’之间有一个空格，转化为字符串时，城市名称前有一个空格，可以考虑使用 Python 字符串中的 strip() 方法去掉首尾空格。

4.2 算法效率的改进

选择排序算法和 Hits 算法在效率上还有提升的空间，此处不做讨论。

4.3 如何跳出循环

4.3.1 第二种方式

跳出迭代循环的方式有两种，一是使用 for 循环迭代 200 次，二是前后两次迭代的差值小于某个小量。前面使用的是第一种，现考虑第二种。

对于矩阵 M ，利用 Hits 算法，迭代计算其 Hub 值和 Authority 值，同时每迭代一次，便归一化一次，得到 k 次迭代后的 $Hub[k], Authority[k]$ 。

对于第 i 次迭代，令

$$\begin{aligned} Gap_Hub &= Hub[i+1] - Hub[i], \quad i = 1, 2, \dots, n-1, \\ Gap_Authority &= Authority[i+1] - Authority[i], \quad i = 1, 2, \dots, n-1 \end{aligned}$$

分别为第 $i+1$ 次迭代前后 Hub 值和 Authority 值的差值（差值列表），取

$$\begin{aligned} Sum_Gap_Authority &= \sum_{i=1}^n |Gap_Authority[i]|, \\ Sum_Gap_Hub &= \sum_{i=1}^n |Gap_Hub[i]|, \end{aligned}$$

第 k 次迭代后，记

$$Gap[k] = \sqrt{Sum_Gap_Authority^2 + Sum_Gap_Hub^2}, \quad k = 1, 2, 3, \dots$$

迭代 k_0 次后，若有

$$Gap[k_0] < 10^{-5}$$

则跳出循环，迭代结束。代码如下：

```
# Hits算法:
while True:
    temp1 = Authority
    temp2 = Hub
    Authority0 = (A.T @ Hub)
    Hub0 = (A @ Authority)
    # 归一
    Authority = Authority0 / sum(i for i in Authority0)
    Hub = Hub0 / sum(i for i in Hub0)
    # 前后两次迭代的差值列表
    Gap_Authority = Authority - temp1
    Gap_Hub = Hub - temp2
    # 后面会对差值列表求和，差值应为正值，此处需处理负值：
    for item in Gap_Authority:
        item = math.fabs(item)
    for item in Gap_Hub:
        item = math.fabs(item)
    # 求和
    Sum_Gap_Authority = sum(i for i in (Gap_Authority))
    Sum_Gap_Hub = sum(i for i in Gap_Hub)
    if math.fabs(
        math.sqrt(
            math.pow(Sum_Gap_Authority,2) + math.pow(Sum_Gap_Hub,2))) < 1e-5:
        break
```

运行结果：

```

排名最高的20个城市及其Authority值:
第1名: 上海; Authority值:0.063842
第2名: 北京; Authority值:0.059383
第3名: 广州; Authority值:0.050590
第4名: 深圳; Authority值:0.045441
第5名: 昆明; Authority值:0.040417
第6名: 重庆; Authority值:0.039658
第7名: 成都; Authority值:0.038967
第8名: 西安; Authority值:0.036510
第9名: 杭州; Authority值:0.027110
第10名: 厦门; Authority值:0.024501
第11名: 郑州; Authority值:0.024405
第12名: 海口; Authority值:0.023383

```

4.3.2 两种方式的差异

对比两种方式得到的结果，有一定的差异。

以排名第一的上海为例。经过分析数据以及估值计算发现，对于第一种方式，当

$$\text{迭代次数} > 12$$

时，上海的 Authority 值与最终稳定值 0.058142（保留 6 位小数）的误差已经不超过

$$10^{-6}$$

而对于第二种方式，记

$$\text{Authority_Precision}(k)$$

表示差值小于 k 便跳出循环时上海的 Authority 值，前面的运行结果表明

$$\text{Authority_Precision}(10^{-5}) = 0.063842$$

与稳定值 0.058142（保留 6 位小数）相差较大，因此，需要继续缩小 k 的值，调试程序，得到了以下结果：

$$\text{Authority_Precision}(10^{-5}) = 0.063842$$

$$\text{Authority_Precision}(10^{-6}) = 0.063842$$

.....

$$\text{Authority_Precision}(10^{-15}) = 0.063842$$

$$\text{Authority_Precision}(10^{-16}) = 0.058142$$

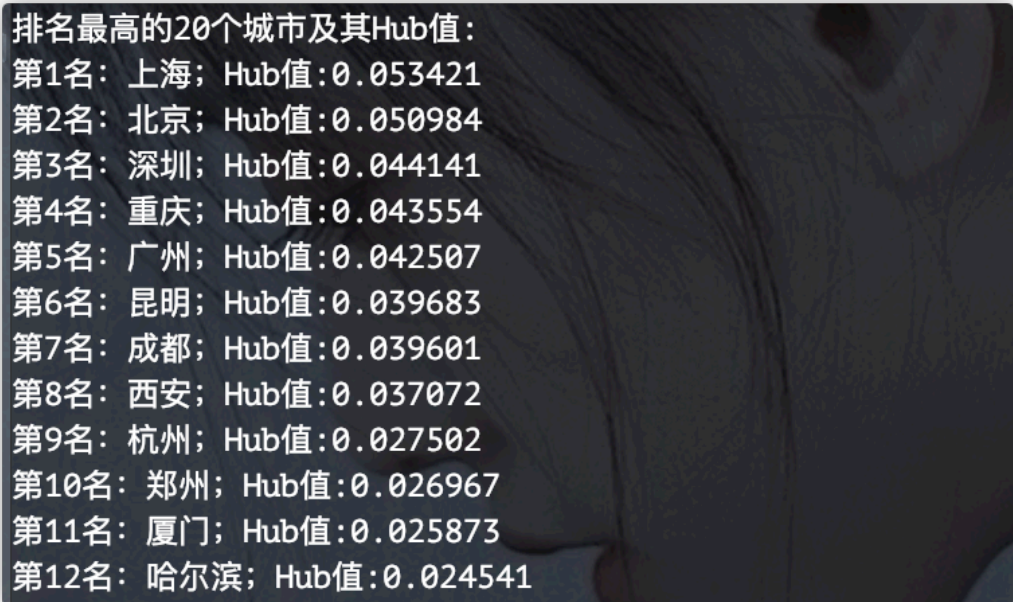
可以看到，当 k 取 10^{-16} 时，才可以达到预期效果。

但是实测发现，当 k 取 10^{-17} 时，程序运行时间大幅增加，目前测试，运行 5 分钟，仍未得到结果，实际运行过程中不太容易控制精度，因而不建议采用这种方式。对比两种方式，此题采用第一种方式更具可行性。

4.4 数据分析

4.4.1 对 Hub 值的分析

前述结果采用的是 Authority 值对城市民航水准进行分析，也可以利用 Hub 值来分析。简单修改程序，运行程序，得到各城市的 Hub 值排名如下：



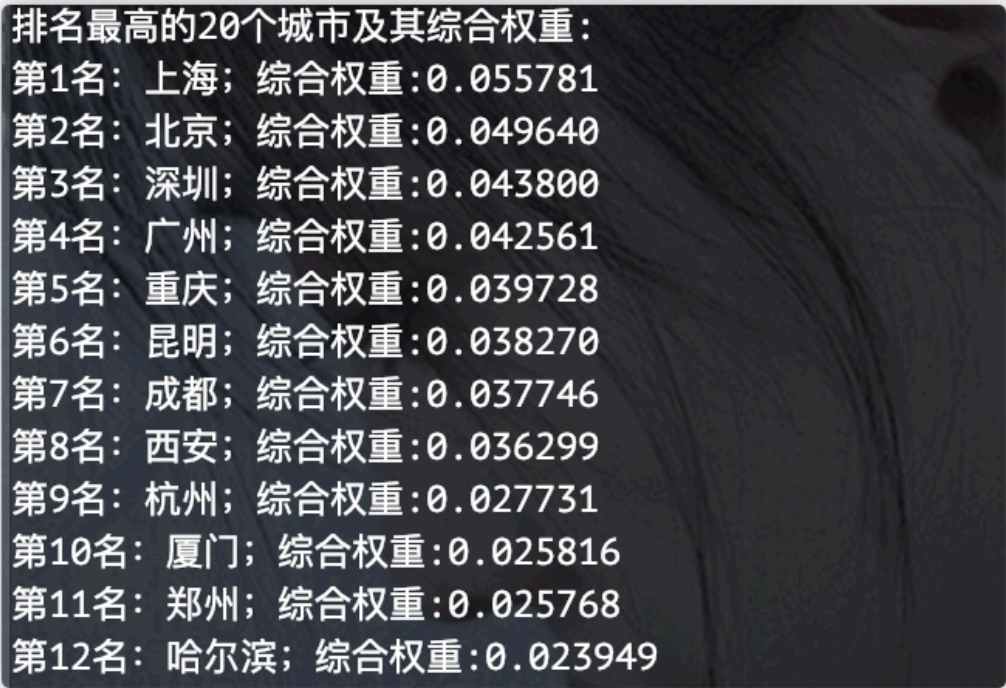
```
排名最高的20个城市及其Hub值：  
第1名：上海；Hub值：0.053421  
第2名：北京；Hub值：0.050984  
第3名：深圳；Hub值：0.044141  
第4名：重庆；Hub值：0.043554  
第5名：广州；Hub值：0.042507  
第6名：昆明；Hub值：0.039683  
第7名：成都；Hub值：0.039601  
第8名：西安；Hub值：0.037072  
第9名：杭州；Hub值：0.027502  
第10名：郑州；Hub值：0.026967  
第11名：厦门；Hub值：0.025873  
第12名：哈尔滨；Hub值：0.024541
```

可以看到，与通过 Authority 值计算得到的排名相比，两者的结果有一定差异，但仍在可接受范围内。

4.4.2 综合分析

现考虑综合 Authority 值和 Hub 值来分析数据。

实际上，Authority 和 Hub 的关系比较复杂。此处作简约化处理，取二者的平均值作为最终每个城市的民航水准指标，修改代码，运行程序得到如下结果（部分）：



排名最高的20个城市及其综合权重：
第1名：上海；综合权重：0.055781
第2名：北京；综合权重：0.049640
第3名：深圳；综合权重：0.043800
第4名：广州；综合权重：0.042561
第5名：重庆；综合权重：0.039728
第6名：昆明；综合权重：0.038270
第7名：成都；综合权重：0.037746
第8名：西安；综合权重：0.036299
第9名：杭州；综合权重：0.027731
第10名：厦门；综合权重：0.025816
第11名：郑州；综合权重：0.025768
第12名：哈尔滨；综合权重：0.023949

这个结果具有较强的可靠性，可以作为中国城市民航水准的参考依据之一。

参考文献

- [1] <https://baike.baidu.com/item/民用航空/>
- [2] [https://baike.baidu.com/item/HITS 算法/](https://baike.baidu.com/item/HITS_算法/)
- [3] <http://blog.csdn.net/rubinorth/article/details/52231620>