计算机应用编程实验报告

计算机应用编程实验4：网络VIP节点分析

2020110656 高垚 2020111562 杨恩

# 实验目标与意义

# 随着网络技术的普及，互联网的信息逐渐庞大，如何在海量且冗余的信息中找到用户所需的结果是互联网每天都要应对的难题。信息的基数与增长速度，用户查找内容的多样性，查询结果的实时动态变化都使精确搜索成为互联网面临的挑战。

# 本实验的目的是设计一个图节点分析程序的算法，从十四万多个网页中，构建页面链接网络，然后计算前20个权重最高的页面。本实验需要构造每个网页的文件名和出链的信息，并持久化存储到文件graph.bin，读取graph.bin并在result.txt输出介数最高的前20的网页。

# 系统设计与实现

## Bin文件的构造

Bin文件的格式是第一块写入有效的网页总数，然后是遍历到的网页的路径，以及对应的出链信息，Bin文件的格式如图1所示。本部分将通过三次遍历，借助指针数组，AC自动机和队列完成对Bin文件的构造。

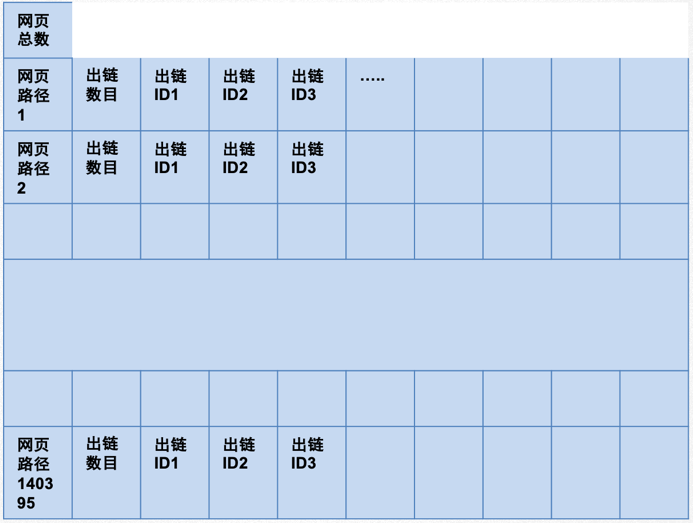


图 1 Bin文件的格式

### 实验思路

本节概括了本实验部分的流程。

- 为了保存海量网页文件的信息，构建Bin文件，我们需要进行以下几个操作。

1. 第一遍遍历，生成队列，队列中包含文件路径和文件名。

2. 第二遍遍历，根据上一步生成的队列，建立AC自动机，为每一个结束的节点，设置对应的ID，利用一个一维数组，数组的第n个元素指向队列中，id为n的节点。

3. 第三遍遍历，遍历第一轮的队列，根据文件路径信息打开相应文件，去ac自动机查id，并把该ID用链表的方式串在队列中。

4. 读取建立的队列，计算出网页文件的总数，遍历的每个网页文件的路径，以及出链的数目，出链的每一个网页文件的ID，并根据该信息生成Bin文件。

//遍历文件夹，构建FileNode队列

printf("Start map all webpage...\n");

travel\_file(base\_path, create\_file\_queue);

printf("Map webpage finished, the webpage number is %d.\n", file\_num);

//初始化ac状态机

printf("Start build ac-state-machine.\n");

ac\_init\_tree();

//文件信息插入ac状态机

build\_ac\_tree();

//构建ac状态机失效指针

ac\_make\_turn();

//读取每个网页文件并在ac状态机中匹配

printf("Start matching in the ac-state-machine.\n");

match\_link();

//输出所有网页出链信息到graph.bin

printf("Start exporting all information to the graph.bin.\n");

其中涉及到的函数有：

|  |
| --- |
| /\*根据传入的文件名，文件路径创建对应的FileNode结点，并放入文件结点队列中\*/  void create\_file\_queue(char \*file\_name, char \*file\_path)  /\*根据传入路径信息，遍历此路径下所有文件并执行相关操作\*/  void travel\_file(char \*path, void (\*operator)(char \*file\_name, char \*file\_path))  /\*创建指向文本的指针和内存缓冲区\*/  void match\_link()  int read\_line(FileHandler\* file\_handler, char\*\* str) \*\*//读取文本的内容，保存至内存。  int write\_line(FileHandler\* file\_handler, char\* str)//把内存中的值写入创建的文本中。  int close\_file(FileHandler\* file\_handler)//关闭文件，释放空间。 |

### 2.1.2 队列的建立

遍历news.sohu.com文件，建立存储了网页路径，网页名，网页出链ID等信息的队列。其数据结构设计如下：

|  |
| --- |
| /\*定义队列的结构\*/  typedef struct file\_node //代表网页文件的数据结点  {  char \*path; //此网页的路径  int id;  struct file\_node \*next; //队列中下一个结点  } FileNode;  /\*根据传入的文件名，文件路径创建对应的FileNode结点，并放入文件结点队列中\*/  void create\_file\_queue(char \*file\_name, char \*file\_path){  //创建FileNode结点并赋初值  FileNode \*file\_node = (FileNode \*)malloc(sizeof(FileNode));  file\_node->id = file\_num;  file\_node->path = ch;  file\_node->next = NULL;  }  /\*遍历FileNode队列，根据FileNode中存放的文件路径打开相应文件，并把文件作为字符串在构建好的ac状态机中查询，查询函数ac\_query\_recoder负责处理输出信息\*/  void create\_file\_queue(char \*file\_name, char \*file\_path){} |

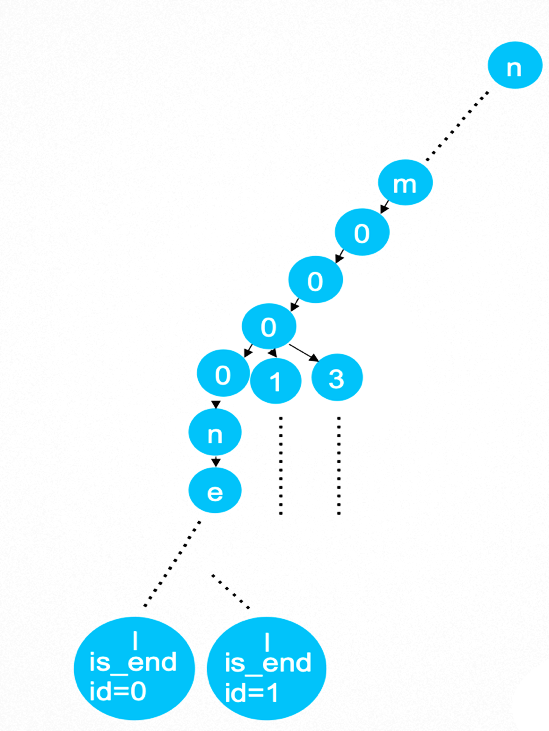


图3 AC树 的数据结构表示

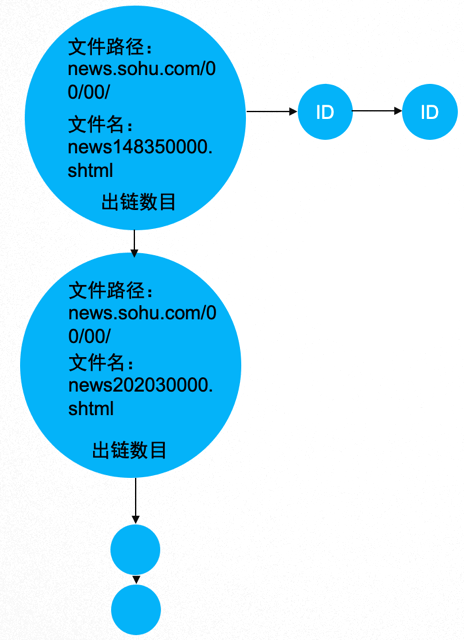


图2 队列的数据结构表示

### 2.1.3 AC树的建立

AC查找能够帮助我们以很快的速度进行搜索，因此本实验利用AC查找树。本实验中，利用的文件操作涉及到了指针。其数据结构设计如下：

|  |
| --- |
| /\* ac状态机树的初始化 \*/  void ac\_init\_tree(){  root = ac\_create\_newnode(); //创建树根  }  /\*创建AC树\*/  void build\_ac\_tree(){  FileNode \*file;  while (file)  {  ac\_insert\_recoder(file); //路径作为字符串插入状态机  file = file->next;  }  }  /\*AC状态机树中插入一条记录\*/  void ac\_insert\_recoder(FileNode \*file){  point->is\_end = 1; //最后一个节点打个结束标记  }  /\*遍历AC状态机每层的节点，构建它们的失效指针\*/  void ac\_make\_turn (){} |

2.1.4 查找

每次打开一个网页文件在AC自动机中查找并输出出链信息

## 2.2 PageRank的计算

PageRank是一种由根据网页之间相互的超链接计算的技术，而作为网页排名的要素之一。Google用它来体现网页的相关性和重要性，在搜索引擎优化操作中是经常被用来评估网页优化的成效因素之一。PageRank通过网络浩瀚的超链接关系来确定一个页面的等级。例如，Google把从A页面到B页面的链接解释为A页面给B页面投票，Google根据投票来源（甚至来源的来源，即链接到A页面的页面）和投票目标的等级来决定新的等级。

### 2.2.1 实验思路

本节概括了本实验部分的流程。

- 为了计算Rank的值。我们需要进行以下几个操作。

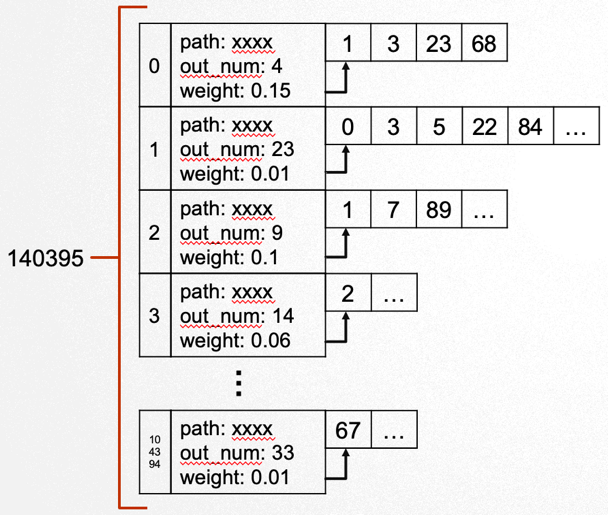
1. 打开在上一个实验步骤中建立的Bin文件

2. 读取网页结点数目，计算全局平均权重

3. 读取所有网页出链信息

4. 计算构建幂迭代的初始向量

5. 进行幂迭代计算，当误差小于

6. 输出排名前20的网页信息

其中涉及到的函数有：

|  |
| --- |
| /\*根据PageRank算法计算每个网页的权重\*/  double self\_pow(double num1, int num2)  float self\_sqrt(float x)  float \*calc\_new\_rank(float \*page\_rank)  float calc\_error(float \*page\_rank\_old, float \*page\_rank\_new)  /\*此函数调用各子函数，完成对网页排序的任务\*/  void page\_rank () |

### 2.2.2 数据的读取

根据上一部分的实验得出的Bin文件，读取网页文件的结点数目，然后计算全局平均权重。继而根据网页的出链数信息，计算权重。

|  |
| --- |
| /\*打开graph.bin\*/  if ((graph\_bin = fopen("graph.bin", "rb")) == NULL)  {  printf("Open graph.bin fail!");  return;  }  /\*读取网页结点数目\*/  fread(&page\_number, int\_len, 1, graph\_bin);  /\*计算全局平均权重\*/  average\_weight = page\_rank\_a / page\_number;  /\* 读取所有网页出链信息\*/  for (i = 0; i < page\_number; i++){  } |

### 2.2.3 rank的计算

根据读取的文件信息，迭代计算每个网页文件的权重，直到当误差小于page\_rank\_error时停止计算。实际数据结构如下所示：

|  |
| --- |
| /\* Rank值的计算 \*/  double self\_pow(double num1, int num2)  float self\_sqrt(float x)  float \*calc\_new\_rank(float \*page\_rank)  float error;  int iteration\_num = 0;  do  {  iteration\_num++;  for (i = 0; i < page\_number; i++)  {  page\_rank\_old[i] = page\_rank\_new[i];  }  page\_rank\_new = calc\_new\_rank(page\_rank\_new);  error = calc\_error(page\_rank\_old, page\_rank\_new);  } while (error > page\_rank\_error); |

### 2.2.4 精确度的计算

当对一个网页文件的权重的两次计算差值时，如果该差值大于预先设置的阈值，则需要停止迭代，因此本部分的步骤是计算权重先后两次的差值。实际数据结构如下所示：

|  |
| --- |
| /\* 精确度 \*/  float calc\_error(float \*page\_rank\_old, float \*page\_rank\_new){  float error = 0;  int i;  for (i = 0; i < page\_number; i++)  {  error += (self\_pow(page\_rank\_new[i] - page\_rank\_old[i], 2));  }  return self\_sqrt(error);  } |

## 2.3 介数中心的计算

介数中心性是基于最短路径针对网络图中心性的衡量标准之一。针对全连接网络图，其中任意两个节点均至少存在一个最短路径，在无权重网络图中该最短路径是路径包含边的数量求和，加权网络图中该最短路径则是路径包含边的权重求和。它适用于解决网络理论中的许多问题，包括与社会网络、生物、运输和科学合作等方面相关的问题。本实验部分将通过介数中心计算每个网页的权重，给出权重最高的前二十个网页。

### 2.3.1 实验思路

本节概括了本实验部分的流程。

1. 打开在上一个实验步骤中建立的Bin文件

2. 利用广度优先遍历所有的网页

3. 遍历过程中计算节点被经过次数

4. 计算输出排名前20的网页信息

其中涉及到的函数和结构体有：

|  |
| --- |
| /\*网页结点\*/  typedef struct page {  int link\_num; //连接的网页数目  char \*path; //网页路径  int \*link\_list; //连接的网页列表  } Page;  typedef struct path  {  unsigned int node\_num; //连接的网页数目  int \*node\_list; //连接的网页列表  struct path \*next; //下一条路径信息  } Path;  /\*主函数\*/  void brandes\_between() |

### 2.3.2 构建网页数据的队列

根据上一部分的实验得出的Bin文件，读取网页文件的结点数目，读取网页信息。继而根据网页的出链数信息，计算权重。

|  |
| --- |
| /\*打开graph.bin\*/  for (i = 0; i < page\_number; i++)  {  //读取网页路径  do  {  fread(&temp\_char, char\_len, 1, graph\_bin);  } while (temp\_char);  //生成新的PageNode结点并初始化  //读取此网页的出链数目  //构建存储此网页出链指向信息的数组  //读取出链信息  for (j = 0; j < out\_link\_num; j++)  {  fread(&out\_link\_page\_id, int\_len, 1, graph\_bin);  }  } |

### 2.3.3 遍历网页数据

每轮遍历访问所有旧路经信息，找到每条老路径末尾节点能到达的新节点并以此创建新路径，直到某一轮遍历找不到任何新路径，则i节点能够到达的所有节点寻找完毕。

|  |
| --- |
| for (i = 0; i < page\_number; i++){  do{  while (old\_path\_num){ //遍历所有旧路经  /\*遍历该网页节点的出链信息，若一个出链指向的节点未被访问过，则基于当前路径创建一条新路径\*/  for (j = 0; j < tail\_page->link\_num; j++){  if (!is\_visited[tail\_page->link\_list[j]]) { //判断出链指向的节点是否被访问过  /\*未被访问，创建新路径\*/  /\*创建新路径的路径信息为老路径信息加新访问的节点\*/  /\*创建的新路径加入路径信息队列\*/  /\*新路径上所有新经过的节点它们的被经过路径数加一，总路径数加一\*/  /\*新被访问的节点加入新访问节点队列\*/  /\*释放老路径\*/  }  }  }  //遍历完老路径之后，把所有新被访问到的节点放入已被访问节点列表  }  } |

# 实验分工

按照上述的系统设计，实验分工如下：

高垚实现了整体流程的设计，部分关键函数。杨恩负责文件的读写模块设计，定义bin文件结构以及部分代码。

# 实验结果与分析

### 4.1 Bin文件的创建

本实验第一部分，创建Bin文件中，一共查找到140395个有效网页，并且在极短的时间内，以及尽可能小的内存占用中，获得完整且精确的结果，符合预期效果。

## **（1） 时间消耗**

## 在创建Bin文件的第三次遍历中，时间消耗最大，因为第三次遍历中，包含读每个网页的出链文件名和网页ID，进而写入队列中，该部分项目任务重。在整个过程中，时间消耗152.8秒。

## **（2） 占用内存**

## 在实验第一部分中，内存峰值为107.2M。



## **（3） 文件大小**

## 在实验第一部分中，文件大小为13.8M。



图6 Bin文件大小

### 4.2 Page rank

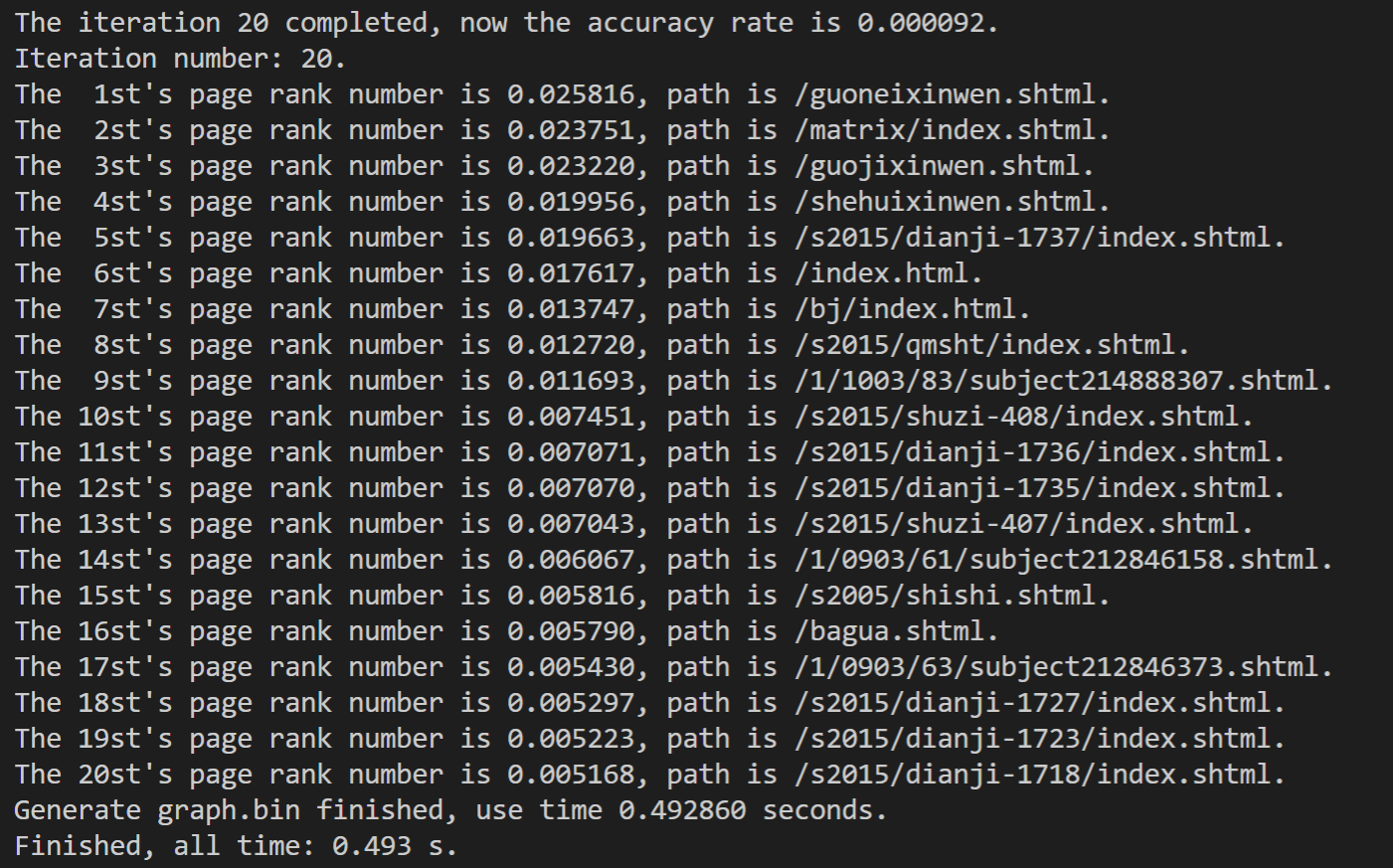
本实验第二部分，利用page rank，对所有网页排序取前二十个输出。在整个过程中，时间消耗0.493秒。内存占用28.6M。

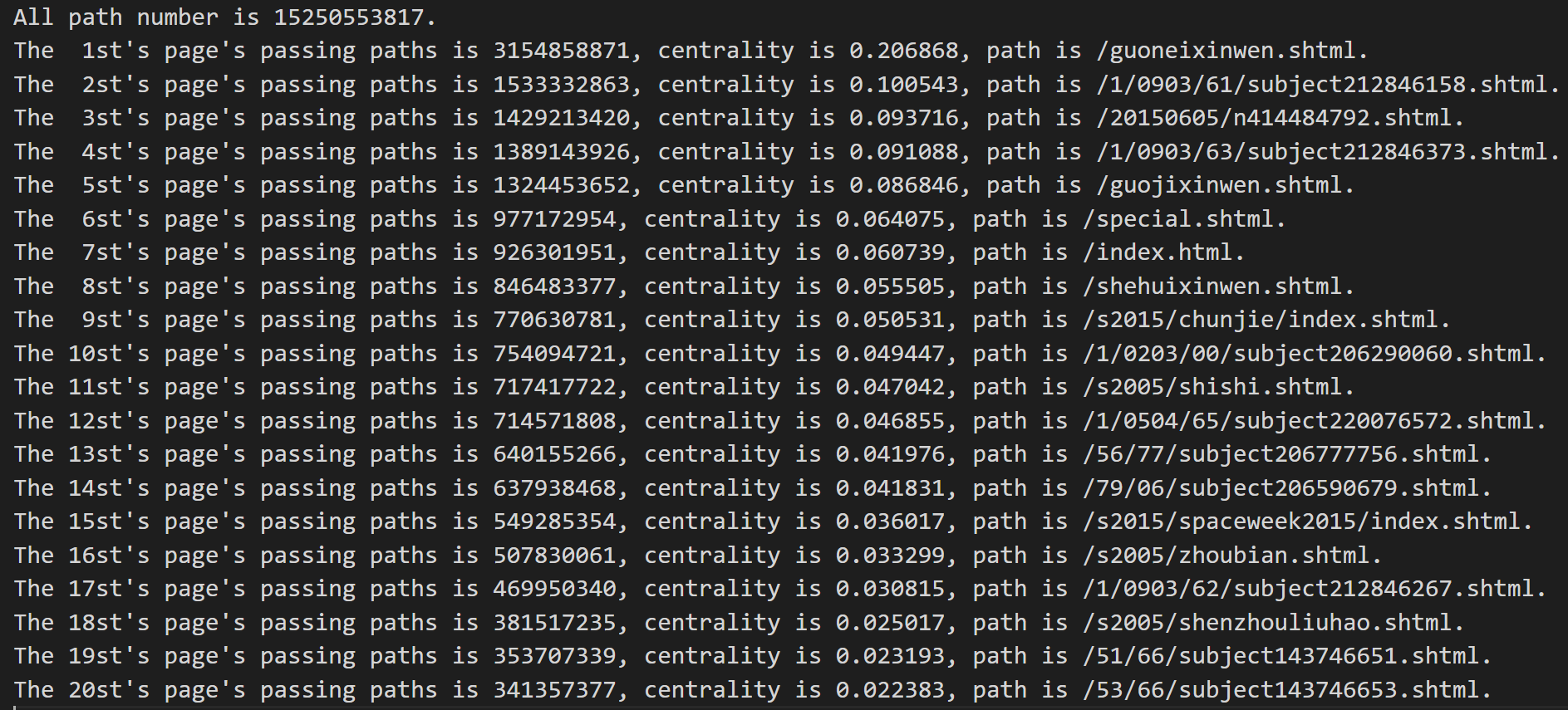
图 7 PageRank 的结果

### 4.3 Brandes

采用BFS算法遍历，由于遍历过程中同步计算经过路径数，比网上查到的较快算法的速度慢很多。其中单线程运行需要七十多小时，之后加入多线程技术，采用8线程运行需要17小时，峰值内存45.7GB。



排序结果：



# 实验结论

综上所述，实验程序能够对海量网页进行搜索和排序，并能在较快的时间内完成计算。

实验中学到了图的相关知识以及部分经典算法，其中page rank算法速度快效果好，但brandes算法速度很慢。