《搜索引擎原型系统设计与实现》设计报告

丁俊峰 141180016

吕航 141210022

摘要

本系统基于数组、树、图等基本数据结构，设计了简易的基于python语言的爬虫程序，采用python语言的中文开源分词算法，构建倒排索引，实现了简易的布尔检索功能，并实现对入度和PageRank的统计，最终搭建了搜索引擎原型系统的测试版本。风格简约，具有一定的人机交互性，代码有一定可读性。

一、程序框架

根据文档出度，递归统计PageRank

生成有向图，根据网页链接，统计文档入度

爬取网页，得到本地文档

对爬取结果进行分词

根据分词结果，遍历所有词，构建以AVL搜索树为主结构的倒排索引结构

布尔检索？

输入检索内容

检测布尔操作符的类型和个数，根据个数生成数组

搜索

依次在AVL树中搜索

搜索成功

搜索成功

文档编号加入相应数组

根据操作符对所有数组进行与/或/非操作

输出搜索失败信息

得到结果，打印文档名称、入度、PageRank等信息

二、数据结构

1.数组

存储方式：顺序存储。

包含功能（主要）：尾部插入、判断空、删除（任意位置元素）、查找（折半搜索）、重载操作符[]。

2.AVL树

存储方式：链式存储。

包含功能（主要）：插入、旋转（四种）、搜索。

3.有向图（邻接矩阵）

存储方式：顺序存储（二维数组）。

包含功能（主要）：插入（顶点、边）、查找（顺序搜索）。

4.稀疏矩阵

存储方式：顺序存储。

包含功能：矩阵乘法。

三、关键模块

1.爬虫

使用Python中的BeautuifulSoup进行爬虫，选择南京大学交换交流网站作为源网站进行链接和文字内容的爬取。

seed\_url='http://stuex.nju.edu.cn' #首页地址

response = urllib.request.urlopen(seed\_url)

html = response.read()

soup = BeautifulSoup(html)

crawl=[seed\_url] #链接数组

seen=set(crawl) #非重复链接存放集合

while crawl:

url=crawl.pop()

urls=get\_links(url)

if len(urls)==0:

break

for link in urls:

if link not in seen:

seen.add(link)

crawl.append(link)

链接的爬取：

for lin in soup.find\_all('a'):

url1=lin.get('href')

文字内容的爬取：

response = urllib.request.urlopen(link)

s=soup.text

2.分词

使用了Python的jieba分词。

分词之前先对文字内容进行数据清洗，清除所有空格和换行：

blank\_line=re.compile('\n+') #数据清洗

s=blank\_line.sub('',s)

blank\_line\_l=re.compile('\n')

s=blank\_line\_l.sub('',s)

blank\_kon=re.compile('\t')

s=blank\_kon.sub('',s)

blank\_one=re.compile('\r\n')

s=blank\_one.sub('',s)

blank\_two=re.compile('\r')

s=blank\_two.sub('',s)

blank\_three=re.compile(' ')

s=blank\_three.sub('',s)

然后进行分词：

seg\_list = jieba.cut\_for\_search(s) # 搜索引擎模式

ss=" ".join(seg\_list) #词之间用空格分隔

3.倒排索引

针对所有文档进行分词后，依次检索所有文档的词语，构建词语的AVL树。AVL树的定义由自己用C++代码实现。

树结点定义如下：

class WordNode{

public:

string name;

Vector<DocNode> \*addr;

bool operator<(WordNode d){

return name<d.name?true:false;

}

bool operator==(WordNode d){

return name==d.name?true:false;

}

bool operator>(WordNode d){

return name>d.name?true:false;

}

};

其中addr指向此包含单词的文件编号的int型数组，并且结点重载了判断大小符号，便于建树。如此搜索词语的平均搜索长度为。

4.布尔检索

实现了对单种多个操作符’+’（or）或’\*’（and）或’-‘（not）的布尔检索功能。得到操作符的类型和个数后，将搜索内容分成相应的词语，分别建立对应包含词语的文件的编号的数组，并进行对应操作符的操作。下面用伪代码说明：

（1）或

i=0,j=0;

do {

do{

if 数组[i][j]不在生成数组中

将元素加入生成数组;

else ;

j++;

}

while j<数组i的长度

i++;

}

while i<词语个数

（2）与

i=1,j=0;

do {

遍历数组i

if 数组i中第j个元素在数组i-1中存在

加入生成数组

else;

遍历数组 i-1

if 数组i-1中第j个元素在数组i中存在

加入生成数组

}

while i<数组个数

（3）非

生成数组=第一个数组

i=1;

do {

遍历数组i

if 第j个元素存在于生成数组

在生成数组中删去该元素;

}

while i<数组个数

其中与操作假设操作的两个数组长度分别为m、n，时间复杂度本应当为，但是采用了折半搜索，时间复杂度降为。

5.入度统计

使用了邻接矩阵来存储有向图，用C++代码实现。

邻接矩阵定义如下：

template<class T, class E>

class Graph

{

private:

T \*VerticesList;

E \*\*Edge;

int maxVertices;

int numEdges;

int numVertices;

public:

Graph(int sz);

~Graph() { delete[]VerticesList; delete[]Edge; }

bool insertVertex(const T & vertex);

bool insertEdge(int v1, int v2, E cost);

int getWeight(int v1, int v2);

int getInDegree(int v);

int getOutDegree(int v);

int getVertexPos(const T &vertex)

{

for (int i = 0; i < numVertices; i++)

if (VerticesList[i] == vertex) return i;

return -1;

}

};

其中getInDegree用来统计结点的入度。getOutDegree用来统计结点出度，便于后面进行PageRank值统计时转移矩阵的计算。

6.PageRank统计

根据各网页的出度初始化二维转移矩阵google。

for (i = 0; i < 283; i++)

{

j = g.getOutDegree(i);

cout << i << " " << j << endl;

if (j == 0)

{

g.Edge[i][151] = 1;

j =1 ;

}

for (k = 0; k < 283; k++)

{

if (g.Edge[i][k] > 0 && g.Edge[i][k] < 100) google[k][i] = (double)1 / j;

}

}

为了解决终止点问题和陷阱问题，将迭代公式变为：

IMG_256

迭代直到最后收敛。

for (i = 0; i < 283; i++)

{

for (j = 0; j < 283; j++)

{

prob1[i] += (double)(0.8\*(double)google[i][j] \* (double)prob[j]);

#矩阵乘法

}

prob1[i] += (double)(0.2/283);

}

}

7.快速排序

Partition函数用来分割数组并返回中间索引，QuickSort进行递归地排序。

void QuickSort(LinkNode\* list, int left, int right)//快速排序

{

if (left < right)

{

int pivotpos = Partition(list,left, right);

QuickSort(list, left, pivotpos - 1);

QuickSort(list, pivotpos+1,right);

}

}

int Partition(LinkNode\* list,int low, int high)//快速排序中的分割

{

LinkNode temp,pivot;

int pivotpos = low;

pivot.degree = list[low].degree;

pivot.name = list[low].name;

for (int i = low + 1; i <= high;i++)

if (list[i].degree < pivot.degree)

{

pivotpos++;

if (pivotpos != i)

{

temp = list[i];

list[i] = list[pivotpos];

list[pivotpos] = temp;

}

}

list[low] = list[pivotpos];

list[pivotpos] = pivot;

return pivotpos;

}

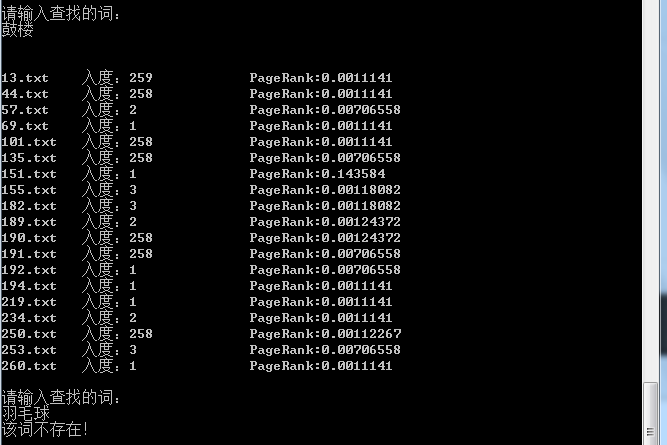
快速排序的时间复杂度是O(),比冒泡排序的O()效率高了很多。

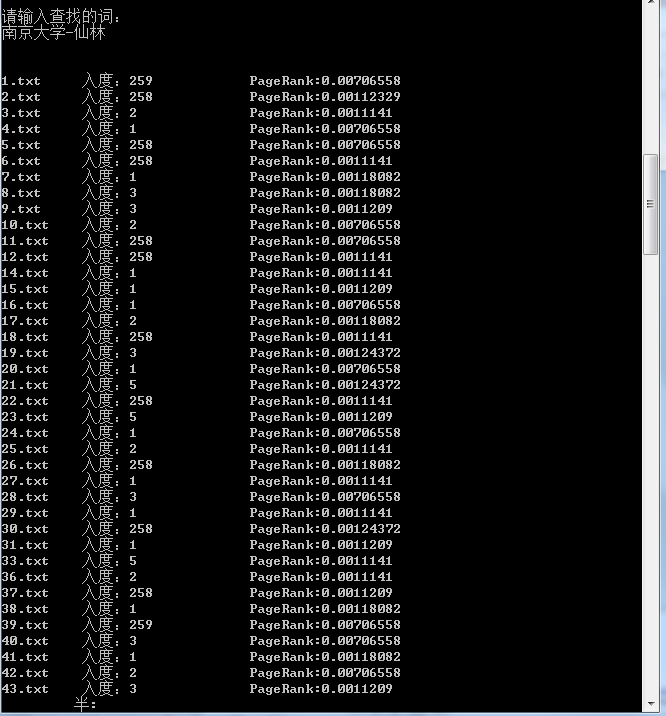
四、成员分工

丁俊峰：爬虫程序、分词算法、入度统计和PageRank统计

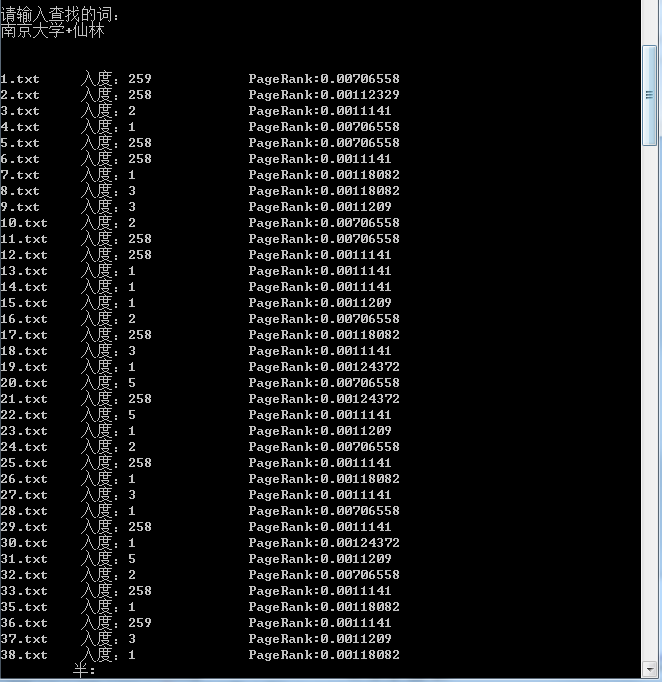
吕航：倒排索引、布尔检索。

五、测试结果

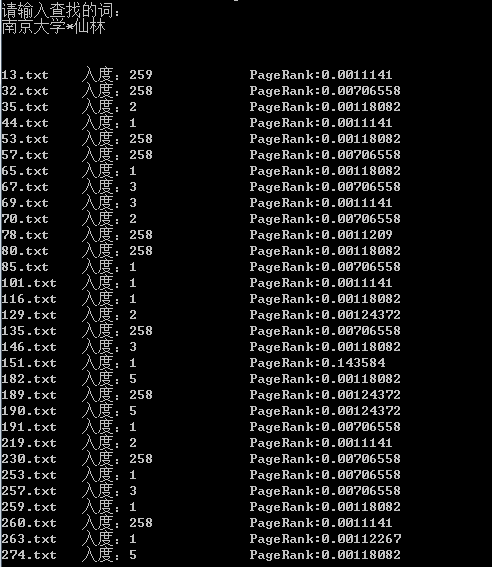




非



与



或

六、不足之处

1.没有图形界面，采用DOS窗口，人机交互友好型不够。

2.输出结果没有按照入度排序。

3.没有实现高级的布尔检索功能以及对输入内容的进一步分词。