

数据结构大作业报告 第三组

蒋雨霖：151220045 支原：151220173 马泽坤：151140042

一. 问题描述:

sh 该搜索引擎原型系统的主要设计思路:

(a):确定爬虫目标网站, 爬虫语言采用 Python, 爬取内容主要包括网页的主题内容、出入度链接; 采用 txt 文档的形式存取; 并且在爬虫爬取的同时利用提供的分词软件进行分词存入 txt 文档。

(b):利用分词后的 txt 文件构建倒排表, 基于倒排表实现布尔检索。

(c):利用爬取的文档当中的链接关系, 确定每个文档的出度入度, 建立相应的邻接矩阵, 设定误差因子常数, 通过迭代实现对文档的 PR 值计算。

(d):基于布尔检索返回的结果, 对每个文档赋予其对应 PR 值和入度, 实现 PR 值排序和入度排序, 输出排序后的结果。

二. 成员分工:

马泽坤: 爬虫和分词部分。

支原: 构建倒排索引以及实现布尔检索

蒋雨霖: 实现 Pagerank 排序和入度排序, 整合代码及报告。

三. 数据结构的设计:

(1):爬虫部分:

1. 考虑到时间有限, 选择上手快捷、内置方法丰富的脚本语言 Python 编写网络爬虫, 其在 Unix 内核的操作系统中可以自动执行。
2. 考虑到目标站点默认编码为 UTF-8, 实验环境 Windows 系统的文件名编码为 GBK, 程序代码、网页内容采用 UTF-8 编码保存, 网页标题采用 GBK 编码保存。
3. 考虑到分词需要, 导入外部模块 PyNLPIR。
4. 考虑到本次试验不涉及多媒体与请求构建, 使用基本的 urllib 库。
5. 考虑到可读性, 变量全部用表示其作用的单词命名:

数据对象	数据类型	初值	解释
page	dict	{}	包含各网页的记录, 也是最终文档

			的信息源。考虑到纵向比较的效率问题，选择 dict 作为 page 的类型
url	string		保存网址，唯一地确定某网页
page[url]	dict		保存 url 指向的网页的记录。考虑到横向比较的效率问题，选择 dict 作为 page[url] 的类型
page[url]['title']	string		保存 url 指向的网页的标题
page[url]['content']	string		保存 url 指向的网页的内容
page[url]['outlink']	set		保存 url 指向的网页的出链。考虑到两网页间的同向链接最多只有一条，选择 set 作为 page[url]['outlink'] 和 page[url]['inlink'] 的数据类型
page[url]['inlink']	set		保存 url 指向的网页的入链
run	bool	True	运行标志，传递 stop 的控制消息。

			当且仅当 url 为 True 时，crawl 运行
temporary	dict	<code>crawl('http://zh.harrypotter.wikia.com/wiki/%E5%93%88%E5%88%A9%C2%B7%E6%B3%A2%E7%89%B9')</code>	临时保存入口网页的记录，形式同 <code>page[url]</code>
i			即 item，用于迭代操作
j	string		用于嵌套的迭代操作
k	string		用于嵌套的迭代操作
f	file	<code>open(page[i]['title'] + '.txt', 'w')</code>	即 file，用于文件操作
predecessor	string		保存 crawl 当前获取的网页的网址，形式同 url
text	string	<code>urllib.urlopen(predecessor).read()</code>	保存 predecessor 指向的网页的源码
destination	dict	<code>{predecessor: 'title:', 'content:', 'outline': set(), 'inline': set()}}</code>	包含 predecessor 指向的网页的记录，形式同 <code>page</code>

destination[predecessor]	dict	{'title':'','content':'','outlink':set(),'inlink':set()}	保存 predecessor 指向的网页的记录, 形式同 page[url]
destination[predecessor]['title']	string	''	保存 predecessor 指向的网页的标题, 形式同 page[url]['title']
destination[predecessor]['content']	string	''	保存 predecessor 指向的网页的内容, 形式同 page[url]['content']
destination[predecessor]['outlink']	set	set()	保存 predecessor 指向的网页的出链, 形式同 page[url]['outlink']
destination[predecessor]['inlink']	set	set()	保存 predecessor 指向的网页的入链, 形式同 page[url]['inlink']
section	list	re.findall(r'</div><div style="display: table-cell; text-align:	保存 predecessor 指向的网页的引言

		<pre> left; padding: 10px;”&gt;[\s\S]*? &lt;/div&gt;&lt;/div&gt;&lt;/d iv&gt;\n’, text) </pre>	
successor	string	<pre> 'http://zh.harr ypotter.wikia.c om' + j </pre>	保存 crawl 即将获取的网页的网址，形式同 url
source	dict	<pre> crawl(successor ) </pre>	包含 successor 指向的网页的记录，形式同 page
source[source	dict		保存 successor 指向的网页的记录，形式同 page[url]
source[source	string		保存 successor 指向的网页的标题，形式同 page[url]['title']
source[source	string		保存 successor 指向的网页的内容，形式同 page[url]['content']
source[source	set		保存 successor 指向的网页的出链，形式同 page[url]['outlink']
source[source	set		保存 successor 指

r]['inlink']			向的网页的入链， 形式同 page[url]['inli nk']
--------------	--	--	--

6. 考虑到网速和程序运行时间，连网时延被设定成 100s。
7. 考虑到后期调试，一旦成功保存一条网页记录，便打印出网址。
8. 考虑到用户友好性，在程序运行时随时可以按回车键终止。
9. 考虑到可操作性，选择“哈利•波特”词条  
(<http://zh.harrypotter.wikia.com/wiki/%E5%93%88%E5%88%A9%C2%B7%E6%B3%A2%E7%89%B9>)作为入口网页，选择各词条的引言作为解析对象，网页内容和链接关系均从中收集。
10. 考虑到 inlink 的计算和写文件的时耗，crawl 执行完毕后，程序才会将网页记录存入文档。
11. 考虑到文件组织的条理性，网络爬虫在同级目录下的同名文件夹内工作，生成的文档与网页一一对应，文档名是网页标题，文本是网页内容、出链、入链。
12. 考虑到数据特征，各变量的空值在条件允许时皆为同类型的空集，如 destination[predecessor]['outlink']预设成 set()。None 予以保留。
13. 考虑到健壮度，如果出现异常情况，淘汰处理，即将返回结果置空并继续运行。代码中对九种异常情况进行捕捉：
  - 1) 用户终止程序。
  - 2) 连网失败。
  - 3) 标题缺失。
  - 4) 标题转码失败。
  - 5) 引言缺失。
  - 6) 内容缺失。
  - 7) 内容转码失败。
  - 8) 网页记录为空。
  - 9) 文件操作失败。

14. 考虑到需求有限，屏蔽了分词系统的词性标注和关键词筛选功能。

(2): 倒排索引部分:

采用结点保存关键词信息和文档信息:

```
typedef struct
{
    int nHashA;
    int nHashB;
    char bExists;
}SOMESTRUCTTRUE; //nHashA和nHashB用来校验
typedef struct key_node
{
    char *pkey; // 关键词实体
    int count; // 关键词出现次数
    int pos; // 关键词在hash表中位置
    struct doc_node *next; // 指向文档结点
}KEYNODE, *key_list;

typedef struct doc_node
{
    int id; //文档ID
    char *name; //文档名
    char *address; //文档网页地址
    double PR; //pagerank值
    int in; //入度值
    struct doc_node *next;
}DOCNODE, *doc_list;
```

(3): 布尔检索部分:

a): 栈 b): 链表

(4): Pagerank 及入度排序部分:

a): 采用结构数组保存文档信息及文档的 PR 值, 出入度信息, 结构体如下:

```
typedef struct
{
    string str1; //文件路径名
    string str2; //文件名
}Files;
typedef struct //入度及出度
{
    string str; //文件名
    int in; //入度
    int out; //出度
    double value;
}Files1;
```

b): 二维数组建立邻接矩阵

#### 四. 算法设计:

(1): 爬虫部分:

网络爬虫的算法经过一次较大改动, 原方案是 crawl 根据参数 cu (current url) 连网获取内容以及出链 nu (next url), 然后递归处理出链:

```

page = {'http://zh.harrypotter.wikia.com/wiki/%F5%93%88%E5%88%A9%C2%B7%E6%B3%A2%E7%89%B9':{'message':'','ol':set(),'il':set()}}
number = 1000

def crawl(cu):
    global number
    print 'get ' + cu + ' ...'
    text = urllib.urlopen(cu).read()
    for i in re.findall(r'</div><div style="display: table-cell; text-align: left; padding: 10px;">.*?</div></div></div>\n',text):
        for j in re.findall(r'>(.*)<',i):
            page[cu]['message'] += j
            for j in re.findall(r'href="(.*?)"[\^>]*?title',i):
                nu = 'http://zh.harrypotter.wikia.com' + j
                if nu in page:
                    page[cu]['ol'].add(nu)
                    page[nu]['il'].add(cu)
                else:
                    if number > 0:
                        number -= 1
                        page[cu]['ol'] = nu
                        page[nu] = {'message':'','ol':set(),'il':set(cu)}
                        crawl(nu)
                    else:
                        continue

crawl(page.keys()[0])

```

这个版本无法判断网页的有效性，由于采用尾递归，每个网页在访问前就已经被添进记录，同时后期不做检查，容易出现崩溃或空文档。通过变更递归方式、附加函数返回值问题得到解决

(2):倒排索引部分:

1. 采用结点保存关键词信息和文档信息

2. 在 hash.h 中的主要函数:

函数 prepareCryptTable 以下的函数生成一个长度为 0x1000000 的 cryptTable[0x1000000]

void PrepareCryptTable()

函数 HashString 以下函数计算 lpszFileName 字符串的 hash 值，其中 dwHashType 为 hash 的类型，unsigned long HashString(const char \*lpszkeyName, unsigned long dwHashType )

按关键字查询， 如果成功返回 hash 表中索引位置

key\_list SearchByString(const char \*string\_in)插入关键字， 如果成功返回 hash 值

int InsertString(const char \*str)

(3):布尔检索部分:

1. 采用栈保存表达式

2. 用链表的交与并求值

(4):Pagerank 排序及入度排序部分:

1. 利用 getFiles 函数读取文件夹中所有文件名，并且创建结构数组，通过读文件的方式，得到 outlink 个数和 inlink 个数，以及文档名存入结构数组。



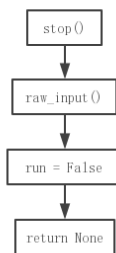
2. 根据得到的结构数组建立邻接矩阵, 并且设置 **Pagerank** 算法所需的特征向量, 以及防止陷阱的修正误差因子  $q$ . 进行迭代矩阵运算(邻接矩阵与特征向量), 当迭代停止时, 特征向量中的值即为每个文档的 **PR** 值。

3. 结构数组得到每个文档 **PR** 值及入度值, 对布尔检索返回的结果进行单链表冒泡排序即可。

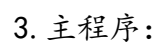
## 五. 实现模块:

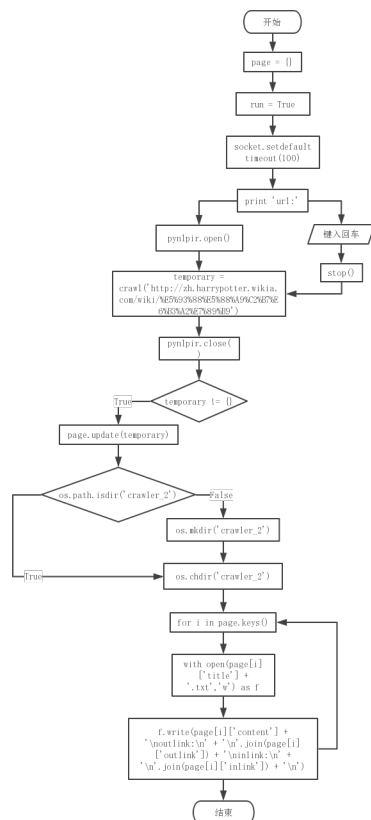
(1) 爬虫部分:

1. **stop** 函数:



2. **crawl** 函数:





## (2) 倒排索引部分：

1. 设计 Hash 算法。其中包括较好的解决冲突，快速定位
2. 在打开文本时，遇到了文本名是 gbk 编码，txt 的内容按 utf-8 编码的情况，这其中涉及到 wchar\_t 与 char 相互转换的问题。
3. 讨论关键词是否第一次出现和对哈希表的查询插入。
4. 采用了 SPIMI 算法，内存式单遍扫描索引算法

将每个块的词典写入磁盘，对于写一块则重新采用新的词典，只要硬盘空间足够大，它能索引任何大小的文档集。

倒排索引 = 词典（关键词或词项+词项频率）+倒排记录表。建倒排索引的步骤如下：

- A) 从头开始扫描每一个词项-文档 ID（信息）对，遇一词，构建索引；
- B) 继续扫描，若遇一新词，则再建一新索引块（加入词典，通过 Hash 表实现，同时，建一新的倒排记录表）；若遇一旧词，则找到其倒排记录表的位置，添加其后
- C) 在内存内基于分块完成排序，后合并分块；

D) 写入磁盘。

其伪码如下：

```
SPIMI-Invert(Token_stream)
output.file=NEWFILE()
dictionary = NEWHASH()
while (free memory available)
do token <-next(token_stream) //逐一处理每个词项-文档 ID 对
if term(token) !(- dictionary
/*如果词项是第一次出现， 那么加入 hash 词典， 同时， 建立一个新的倒排
索引表*/
then postings_list = AddToDictionary(dictionary, term(token))
/*如果不是第一次出现， 那么直接返回其倒排记录表， 在下面添加其后*/
else  postings_list  =  GetPostingList(dictionary, term(token)) if
full(postings_list)
then postings_list =DoublePostingList(dictionary, term(token))
AddToPosTingsList (postings_list, docID(token))
sorted_terms <- SortTerms(dictionary)
WriteBlockToDisk(sorted_terms, dictionary, output_file)
return output_file
```

(3) 布尔检索部分：

1. 实现基于链表的交与并。
2. 识别带有关键字和符号的表达式
3. 通过栈计算表达式。
4. 主要函数如下

```
void UNION(key_list m, key_list n, doc_list &hc); //布尔检索的求并
void InterSect(key_list m, key_list n, doc_list &hc); //布尔检索的求交

bool trans_real_exp(char exp[], char real_exp[], doc_list exp_array[], int &count_key); //将用户输入的表达式转化为用字母代替关键字
void translate(char str[], char exp[]); //将中缀表达式转化为后缀表达式
void cal_value(char exp[], doc_list exp_array[], int &count_key); //通过栈计算后缀表达式
```

#### (4) Pagerank 及入度排序部分：

1. 实现获取文件信息
2. 实现邻接矩阵的建立
3. 实现迭代计算 PR 值
4. 根据返回文档进行单链表冒泡排序 (PR 排序及入度排序)
5. 主要函数：

```
void Pagerank(Files *files, Files1 *page); //计算每个文档PR值以及出入度 记录到page结构数组中
void Insert(doc_list &hc, Files1 *page); //入度值排序
void Pagesort(doc_list hc, Files1 *page); //PR值排序
void getFiles(string path, Files *files); //获取链接文档名
```

#### 六. 关键功能与代码：

##### (1)：爬虫部分：

关键功能：爬取网页文档内容并进行分词以及网页之间的链接。

代码：

```
def crawl(predecessor):

    global page,run

    if not run:

        return {}

    try:

        text = urllib.urlopen(predecessor).read()

    except:

        return {}

    destination = {predecessor: {'title': '', 'content': '', 'outlink': set(), 'inlink': set()}}

    for i in re.findall(r'<h1>([\s\S]*?)</h1>', text):

        try:

            destination[predecessor]['title'] = i.decode('utf-8').encode('gbk')

        finally:

            break

    if destination[predecessor]['title'] == '':

        return {}
```

```

        section = re.findall(r'</div><div style="display: table-cell; text-align: left; padding:
10px;">[\s\S]*?</div></div></div>\n',text)

        for i in section:

            for j in re.findall(r'>([\s\S]*?)<',i):

                destination[predecessor]['content'] += j

            if destination[predecessor]['content'] == "":

                return {}

        try:

            destination[predecessor]['content'] = '\n'.join([i[0].encode('utf-8') for i in
pynlpir.segment(destination[predecessor]['content'])])

        except:

            return {}

    print predecessor

    for i in section:

        for j in re.findall(r'href="([\s\S]*?)"[\^>]*?title',i):

            successor = 'http://zh.harrypotter.wikia.com' + j

            if successor in page:

                destination[predecessor]['outlink'].add(successor)

                page[successor]['inlink'].add(predecessor)

            else:

                page[successor] = {'title': '', 'content': '', 'outlink': set(), 'inlink': set()}

                source= crawl(successor)

                if source == {}:

                    page.pop(successor)

                for i in page.keys():

                    if successor in page[i]['outlink']:

                        page[i]['outlink'].remove(successor)

                    if successor in page[i]['inlink']:

                        page[i]['inlink'].remove(successor)

                else:

```

```

        page.update(source)

        destination[predecessor]['outlink'].add(successor)

        page[successor]['inlink'].add(predecessor)

    return destination

'''

主体：

功能：

    建立空的记录集合，设置时延，监视停止，启用分词，设置入口网页，运行爬取，
保存记录

'''

page = {}

run = True

socket.setdefaulttimeout(100)

print 'url:'

thread.start_new_thread(stop,())

pynlpir.open()

temporary =

crawl('http://zh.harrypotter.wikia.com/wiki/%E5%93%88%E5%88%A9%C2%B7%E6%B3%A2%E7%
89%B9')

pynlpir.close()

if temporary != {}:

    page.update(temporary)

    if not os.path.isdir('crawler_2'):

        os.mkdir('crawler_2')

    os.chdir('crawler_2')

    for i in page.keys():

        with open(page[i]['title'] + '.txt', 'w') as f:

            f.write(page[i]['content'] + '\noutlink:\n' + '\n'.join(page[i]['outlink'])) +
'\ninlink:\n' + '\n'.join(page[i]['inlink']) + '\n')

```

(2):倒排索引部分:

关键功能: 构建倒排表

代码:

```
preparecrypttable();    //初始化 Hash 表

int wordnum = 0;

for (int i = 0; i < filename; i++) //开始读取每一个文档的分词。
{
    fr = OpenReadFile(i, filename);

    if (fr == NULL)
    {
        break;
    }

    bool is_web = false;

    wchar_t*wc = (wchar_t*)malloc(200 * sizeof(wchar_t));

    char*address = NULL;

    while (!feof(fr))
    {
        fgetws(wc, 200, fr);

        int m = wcslen(wc);

        if (wc[m - 1] == '\n')
            wc[m - 1] = 0;

        char*pbuf = (char*)malloc(sizeof(char)*(2 * wcslen(wc) + 1));

        memset(pbuf, 0, 2 * wcslen(wc) + 1);

        setlocale(LC_ALL, "");

        wcstombs(pbuf, wc, 2 * wcslen(wc) + 1);

        if (is_web == true)
        {
            if (keylist = SearchByString(pbuf))    //到 hash 表内查询
            {
                bool flag = false;
```



```

doc_list infolist = Saveltems(i, filename, address);
doc_list com = keylist->next;
while (com != NULL)
{
    if (com->id == i)
    {
        flag = true;
        break;
    }
    else
        com = com->next;
}
if (flag == false)
{
    infolist->next = keylist->next;
    keylist->count++;
    keylist->next = infolist;
}
else
{
    keylist->count++;
}
}
else
{
    // 如果关键字第一次出现，则将其加入 hash 表
    int pos = InsertString(pbuf);      //插入 hash 表
    keylist = key_array[pos];
    doc_list infolist = Saveltems(i, filename, address);
    infolist->next = NULL;
}

```

```

        keylist->next = infolist;

        if (pos != -1)
        {
            strcpy_s(words[wordnum], 50, pbuf);

            wordnum++;
        }
    }

    else
    {
        int addr_len = strlen(pbuf);

        address = (char *)malloc(addr_len + 1);

        memset(address, 0, addr_len + 1);

        strncpy(address, pbuf, addr_len);

        is_web = true;
    }

}

fclose(fr);
}

```

### (3) 布尔检索部分：

关键功能：

解析输入的布尔表达式，实现链表的交与并搜索。

代码：

```

void UNION(key_list m, key_list n, doc_list &hc); //布尔检索的求并

void InterSect(key_list m, key_list n, doc_list &hc); //布尔检索的求交 void UNION(doc_list m,
doc_list n, doc_list &hc)
{

    doc_list pa = m->next;

    doc_list pb = n->next;

```

```

doc_list s, tc;

hc = (doc_list)malloc(sizeof(DOCNODE));

tc = hc;

while (pa != NULL && pb != NULL)
{
    if (pa->id > pb->id)
    {
        s = (doc_list)malloc(sizeof(DOCNODE));

        s->id = pa->id;

        int len = strlen(pa->name);

        s->name = (char *)malloc(len + 1);

        memset(s->name, 0, len + 1);

        strncpy(s->name, pa->name, len);

        int addr_len = strlen(pa->address);

        s->address = (char *)malloc(addr_len + 1);

        memset(s->address, 0, addr_len + 1);

        strncpy(s->address, pa->address, addr_len);

        tc->next = s;

        tc = s;

        pa = pa->next;
    }
    else if (pa->id < pb->id)
    {
        s = (doc_list)malloc(sizeof(DOCNODE));

        s->id = pb->id;

        int len = strlen(pb->name);

        s->name = (char *)malloc(len + 1);

        memset(s->name, 0, len + 1);

        strncpy(s->name, pb->name, len);

        int addr_len = strlen(pb->address);
    }
}

```

```

        s->address = (char *)malloc(addr_len + 1);
        memset(s->address, 0, addr_len + 1);
        strncpy(s->address, pb->address, addr_len);
        tc->next = s;
        tc = s;
        pb = pb->next;
    }
    else
    {
        s = (doc_list)malloc(sizeof(DOCNODE));
        s->id = pa->id;
        int len = strlen(pa->name);
        s->name = (char *)malloc(len + 1);
        memset(s->name, 0, len + 1);
        strncpy(s->name, pa->name, len);
        int addr_len = strlen(pa->address);
        s->address = (char *)malloc(addr_len + 1);
        memset(s->address, 0, addr_len + 1);
        strncpy(s->address, pa->address, addr_len);
        tc->next = s;
        tc = s;
        pa = pa->next;
        pb = pb->next;
    }
}

if (pb != NULL)
{
    pa = pb;
}

while (pa != NULL)

```

```

{
    s = (doc_list)malloc(sizeof(DOCNODE));

    s->id = pa->id;

    int len = strlen(pa->name);

    s->name = (char *)malloc(len + 1);

    memset(s->name, 0, len + 1);

    strncpy(s->name, pa->name, len);

    int addr_len = strlen(pa->address);

    s->address = (char *)malloc(addr_len + 1);

    memset(s->address, 0, addr_len + 1);

    strncpy(s->address, pa->address, addr_len);

    tc->next = s;

    tc = s;

    pa = pa->next;

}

tc->next = NULL;
}

```

```

void InterSect(doc_list m, doc_list n, doc_list &hc)

```

```

{
    doc_list pa = m->next;

    doc_list pb;

    doc_list s, tc;

    hc = (doc_list)malloc(sizeof(DOCNODE));

    tc = hc;

    while (pa != NULL)
    {
        pb = n->next;

        while (pb != NULL && pb->id > pa->id)
        {
            pb = pb->next;

```

```

    }

    if (pb != NULL && pb->id == pa->id)
    {
        s = (doc_list)malloc(sizeof(DOCNODE));

        s->id = pa->id;

        int len = strlen(pa->name);

        s->name = (char *)malloc(len + 1);

        memset(s->name, 0, len + 1);

        strncpy(s->name, pa->name, len);

        int addr_len = strlen(pa->address);

        s->address = (char *)malloc(addr_len + 1);

        memset(s->address, 0, addr_len + 1);

        strncpy(s->address, pa->address, addr_len);

        tc->next = s;

        tc = s;
    }

    pa = pa->next;
}

tc->next = NULL;
}

```

#### (4) Pagerank 部分:

关键功能：迭代实现 PR 值的计算

代码：(迭代部分不包含建立邻接矩阵)

```

do
{
    for (int p = 0; p < size; p++)
    {
        r[p] = page[p].value;
    }

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

```

```

{
    double sumr = 0;
    for (int j = 0; j < size; j++)
    {
        sumr = sumr + matrix[i][j] * page[j].value;
    }
    page[i].value = sumr*q + (1 - q)*e;
}

int f;
for (f = 0; f < size; f++)
{
    if (fabs(r[f] - page[f].value)>10e-6)break;
}

if (f == size)
    s = 1;
} while (s != 1);

```

## 七. 复杂度分析:

(1):爬虫部分:

目前空间复杂度为  $O(n)$ , 时间复杂度为  $O(n^2)$ , 增加函数参数可以把时间复杂度降至  $O(n)$ 。

(2):倒排索引及布尔检索部分:

Data Structure	Add	Find	Delete	GetByIndex
Array ( $T[]$ )	$O(1)$	$O(n)$	$O(n)$	$O(1)$
Linked list ( $LinkedList<T>$ )	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Stack ( $Stack<T>$ )	$O(1)$	-	$O(1)$	-
Queue ( $Queue<T>$ )	$O(1)$	-	$O(1)$	-
Hash table ( $Dictionary<K,T>$ )	$O(1)$	$O(1)$	$O(1)$	-

(3):Pagerank 及入度排序部分：

空间复杂度为：  $O(n)$

时间复杂度为： $O(n^2)$  )

八. 测试流程：

如下图：



```

请输入需要查找的关键字个数
3
请输入关键字
(大作业I魔法)&考试
关键字大作业搜索不到!
搜索失败: 无您需要搜索的内容
是否需要继续查询? (输入 Y or N)
Y
请输入需要查找的关键字个数
4
请输入关键字
(哈利&波特)&(赫敏&格兰杰)
请选择查看哪种排序方式得到的结果:
1. Pagerank排序
2. 入度值排序
2
赫敏·格兰杰.txt    入度值: 11  PR值: 0.00499093
该文档地址:http://zh.harrypotter.wikia.com/wiki/%E8%B5%AB%E6%95%8F%C2%B7%E6%A0%BC%E5%85%B0%E6%9D%B0
米勒娃·麦格.txt    入度值: 3  PR值: 0.00334763
该文档地址:http://zh.harrypotter.wikia.com/wiki/%E7%B1%B3%E5%8B%92%E5%A8%83%C2%B7%E9%BA%A6%E6%A0%BC
是否需要继续查询? (输入 Y or N)
Y
请输入需要查找的关键字个数
2
请输入关键字
dasd dsad
关键字dasd搜索不到!
搜索失败: 无您需要搜索的内容
是否需要继续查询? (输入 Y or N)

```



```

----- 搜索引擎Demo -----
制作人: 蒋雨霖 马泽坤 支原

请输入需要查找的关键字个数
1
请输入关键字
这家
请选择查看哪种排序方式得到的结果:
1. Pagerank排序
2. 入度值排序
1
决斗俱乐部.txt    入度值: 1  PR值: 0.00114876
是否需要继续查询? (输入 Y or N)
Y
请输入需要查找的关键字个数
2
请输入关键字
赫敏&哈利
请选择查看哪种排序方式得到的结果:
1. Pagerank排序
2. 入度值排序
2
赫敏·格兰杰.txt    入度值: 11  PR值: 0.00499093
该文档地址:http://zh.harrypotter.wikia.com/wiki/%E8%B5%AB%E6%95%8F%C2%B7%E6%A0%BC%E5%85%B0%E6%9D%B0
芙蓉·德拉库尔.txt    入度值: 5  PR值: 0.00204269
该文档地址:http://zh.harrypotter.wikia.com/wiki/%E8%8A%99%E8%93%89%C2%B7%E5%B7%B7%E6%8B%89%E5%BA%93%E5%B0%94
格里莫广场12号.txt    入度值: 5  PR值: 0.0016433
该文档地址:http://zh.harrypotter.wikia.com/wiki/%E6%A0%BC%E9%87%8C%E8%8E%AB%E5%B9%BF%E5%9C%BA12%E5%8F%B7
罗恩·韦斯莱.txt    入度值: 4  PR值: 0.00466102
该文档地址:http://zh.harrypotter.wikia.com/wiki/%E7%BD%97%E6%81%A9%C2%B7%E9%9F%A6%E6%96%AF%E8%8E%B1
米勒娃·麦格.txt    入度值: 3  PR值: 0.00334763
该文档地址:http://zh.harrypotter.wikia.com/wiki/%E7%B1%B3%E5%8B%92%E5%A8%83%C2%B7%E9%BA%A6%E6%A0%BC
弗雷德和乔治·韦斯莱.txt    入度值: 3  PR值: 0.00370177
该文档地址:http://zh.harrypotter.wikia.com/wiki/%E4%B9%94%E6%B2%BB%C2%B7%E9%9F%A6%E6%96%AF%E8%8E%B1
复方汤剂.txt    入度值: 2  PR值: 0.00123352
该文档地址:http://zh.harrypotter.wikia.com/wiki/%E5%A4%8D%E6%96%B9%E6%B1%A4%E5%89%82
吉德罗·洛哈特.txt    入度值: 2  PR值: 0.00111065
该文档地址:http://zh.harrypotter.wikia.com/wiki/%E5%90%89%E5%BE%B7%E7%BD%97%C2%B7%E6%B4%9B%E5%93%88%E7%89%B9
速顺滑发剂.txt    入度值: 1  PR值: 0.00107522
搜狗拼音输入法 全 :h.harrypotter.wikia.com/wiki/%E9%80%9F%E9%A1%BA%E6%BB%91%E5%8F%91%E5%89%82

```

主要测试流程:

一般情况  特殊情况  错误情况