搜索引擎原型系统设计与实现

匡亚明学院 翟道京 151250189

一、问题描述

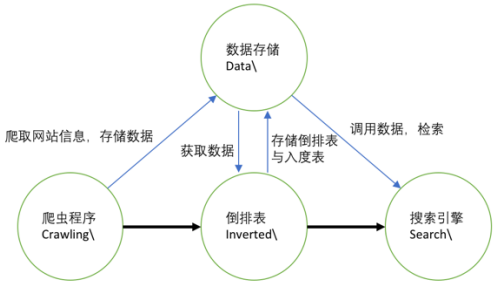
本次作业的要求是实现一个简单的搜索引擎。具体功能包括

1) 使用爬虫程序自动爬取相关的中文文档集合
2) 设计中文分词算法，实现分词。
3) 基于爬取的文档集合和分词结果，构建倒排索引(inverted index)。
4) 实现布尔检索(Boolean retrieval)功能，至少得支持“与(AND)”和“或(OR)”操作。
5) 基于文档之间的有向链接图，实现基于入度的排序算法，用来对布尔检索返回的结果(文档)进行排序。
6) 设计搜索引擎进行搜索

二、实现模块

Daojing Search Engine 是一个中文搜索引擎的原型系统（demo 版本），功能包括网络爬虫，建立倒排索引，网页查询。爬虫部分使用 Python 完成，建立倒排索引和搜索通过 C++完成，同时使用 C++自行搭建了搜索引擎入口。该软件在 Mac 平台上编写，通过 MacVim Makefile 文件将三部分结合，在终端中通过运行指令进行测试。

其中系统架构如下图所示



终端指令	make crawl/make clean	make inv	make sch
使用文件		vocab\ content\ html\ database.txt	index.txt indegree.txt content\ html\
产生文件	vocab\ content\ html\ database.txt	index.txt indegree.txt	main.unix

图 1. Daojing Search Engine 系统架构

三、数据结构

软件中使用的数据结构包括

1) Graph

文章之间的引用关系和网页之间的链接关系自然构成了图的结构。在搜索引擎中，由于需要对检索结果进行排序，需要通过图结果对每个节点的出度和入度进行统计，因此图机构也是本次作业中需要实现的数据结构之一。考虑到在网页链接构成的图结构中边的数量比较稀疏，因此采用邻接表来实现图结构。另外，网页的链接明显是有向的，因此本次作业中需要实现的是基于邻接表的有向图结构，并提供查询结点出度和入度的函数接口。

Graph 在爬虫存储数据过程中得以建立，我们存储结构为

网页 ID：	网页 URL	下层网站 ID

其中每个网页 ID 读取网页 URL 的信息，同时指向下层网站的 ID。

Graph 的建立通过 Python 自行实现(见 Crawler.py)。

2) String

涉及到信息检索，我们必须使用字符串数据结构进行处理字符变量。

3) Vector

数组具有能够随机访问其中元素的特点，使用非常方便。但是由于基本数组结构的最大容量不可变，在使用中的灵活性较差，因此有必要实现容量可变的数组结构，这样在编程时就不需考虑数组的大小。在本次作业中，可变数组应用到了一下几个方面:一是用于实现字符串结构，二是用于存储论文的信息，三是用于在倒排表中存储词条对应的文章。

Vector 通过 C++自行编写，见 djvector.h/djvector.cpp。

4) Map

映射关系在整个系统中得到广泛应用。在数据库建立时，网页的 ID 和其网址以及该网页中的下层链接相映射，在搜索时快速通过网页 ID 找到网页的其余信息;在倒排表和入度表建立时，建立起了关键词与网页 ID 的映射关系、网页 ID 与网页入度的映射关系，从而可以根据关键词快速找到该关键词对应的网页 ID，从入度表中可以读取相应网页的入度，实现布尔检索。

本次大作业允许使用 Map，我们从 STL 库中调用了 Map 容器。

5) Set

在布尔检索过程中，我们求 AND 和 OR 的过程实际上为集合交并过程，我们没有建立类，而是使用了 0/1 数组存储某关键词下的网站 ID，之后对该数组求交集与并集。

四、算法设计与复杂度分析

本次大作业核心算法包括

1) 快速排序算法

快速排序使用分治法策略来把一个序列分为两个子序列，步骤为:

1. 从数列中挑出一个元素，称为"基准"。

2. 重新排序数列，所有比基准值小的元素摆放在基准前面，所有比基准值大的元素摆在基准后面（相同的数可以到任何一边）。在这个分割结束之后，该基准就处于数列的中间位置。这个称为分割操作。

3. 递归地把小于基准值元素的子数列和大于基准值元素的子数列排序。 递归到最底部时，数列的大小是零或一，也就是已经排序好了。这个演算法一定会结束，因为在每次的迭代中，它至少会把一个元素摆到它最后的位置去。

在平均状况下，排序 n 个项目要$O(nlog_2n)$次比较。在最坏状况下则需$O(n^2)$次比较，但这种状况并不常见。事实上，快速排序$O(nlog_2n)$通常明显比其他算法更快，因为它的内部循环可以在大部分的架构上很有效率地达成。

排序过程可以在建立倒排表时，也可以在得到搜索结果后。其中对于较少的搜索量，得到搜索结果后进行排序时间复杂度显然要低。我在得到搜索网页 ID 后才进行按入度排序操作。

我的快速排序算法在 djheaper.cpp 中，见 QuickSort 函数。

2) 布尔检索算法

我的搜索引擎对**多关键词(>=2)**实现了 OR/AND 布尔检索。

OR 检索

OR 检索即为集合交集操作，对于用户输入的检索关键词，通过倒排表获得包含关键词的所有网页 ID 并建立 0/1 数组。OR 检索对两个数组取交集，考虑到建立倒排表时我们从 ID 地址 0 开始扫描，所以倒排表中 ID 从小到大排序，因此程序遍历两个数组每个元素一次，故执行一次两关键词检索过程，时间复杂度为$O(\max(l_1,l_2))$，其中l_1,l_2分别是两个链表的长度。我们没有使用插入操作，而是建立了第三个表存储最后的结果，因此空间复杂度为$O(\max(l_1,l_2))$。

关键算法见 djsearch.cpp。

AND 检索

AND 检索即为集合并集操作，操作方法与时间复杂度与 OR 检索类似，空间复杂度为$O(\min(l_1,l_2))$。关键算法见 djsearch.cpp。

五、实现模块

1. 网络爬虫

网络爬虫程序由 Python 完成(见 crawler.py)，主要分为爬取（Crawl）与分词（Segmentation）两个部分。我总共爬取了 CSDN 数据库板块论坛共计 10000 个网站，464282 个词汇，包括中文和英文，我将自己的电脑视作服务器，记录了这些网站的源文件、预处理结果与分词结果。

a) 爬取部分

爬取部分主要通过队列实现。我们以网站的访问次序为该网站 ID，每次抓取一个网站时，在存储区域判断该网站是否被爬取过，如果被爬取过，该网站进队列，

记录该网站；访问一个网站时，获得他所有的下层网站，选取一定广度的下层网站进入队列，并在这个网站 url 后面记录所有的下层网站，方便建立图的数据结构。最终爬取结束的标准是爬取了满足设定数量要求的网站。在爬取部分实现了对网页关系的记录，这其实就构建了图的数据结构。”database.txt”这个文件记录了该数据结构，“html/”文件夹记录了网站源代码。

b) 分词部分

分词部分主要提取文本信息、同时调用了 jieba 软件进行分词，“content/”文件夹记载了经过处理、剔除代码之后得到了网站文本信息，“vocab/”文件夹记载了分词之后的“词袋”。

2. 构建倒排表与入度表

构建倒排表与入度表过程中，我们主要使用了 Vector 与 map 两种数据结构，其中按照要求，Vector 结构由自行编写(djvector.h, djvector.cpp)。通过 map 容器实现了映射关系，而 vector 良好的动态属性给了我们很大便利。最终得到了入度表“indegree.txt”存储网站的入度，倒排表“index.txt”记录包含一定关键词的网站。

3. 搜索引擎的搭建

我们使用 C++语言写构建搜索引擎，搜索引擎阅读用户输入的关键词，建立 0/1 数组存储相应倒排表下的网站 ID 序列。根据用户要求，对数组进行交/并处理，输出布尔检索后的结果。同时允许用户调用网站、阅读网站内容。

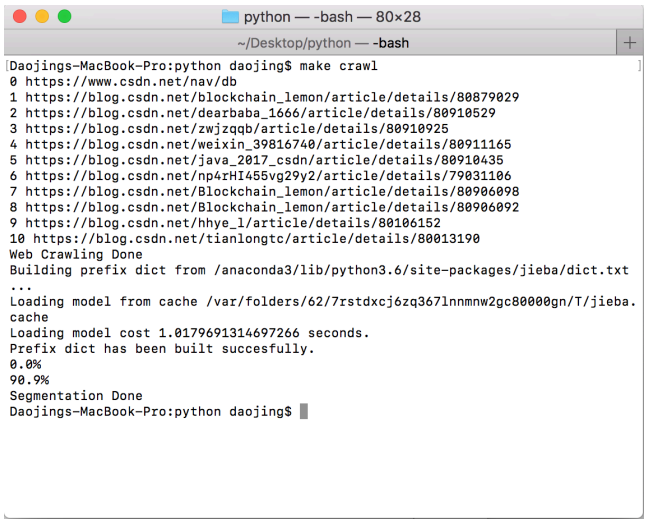
六、使用示例

本软件在 Mac OS 环境下编写，通过 MacVim 在终端下实现集成编译，编译指令为

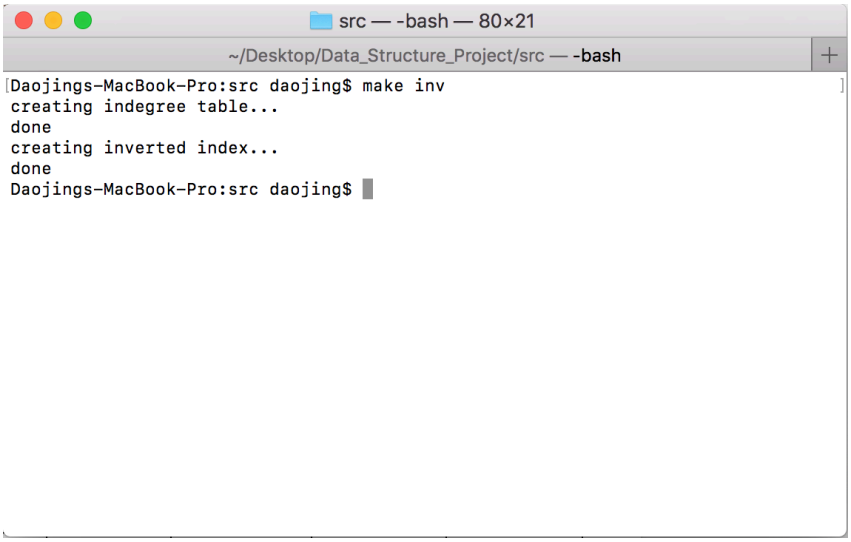
执行三个模块	执行挖掘模块	执行建立倒排表	执行搜索程序	清空文件
make run	make crawl	make inv	make sch	make clean

执行情况如图所示

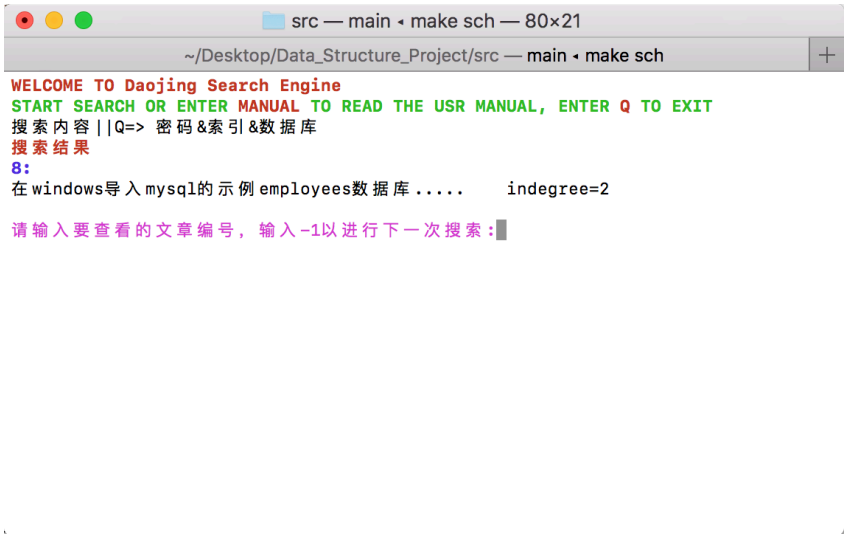
模块一：数据挖掘(这个 demo 版本我只挖掘了 10 个网站)



模块二：建立倒排表



模块三：搜索窗口



注意：由于该软件在 Mac 环境下编写，最终生成 Unix executable 类型工程文件。我担心生成 exe 文件可能因为系统差别、在 Windows 版本下不能运行，所以录制了 demo_daojing.mov 视频，记录了测试的过程。

源代码在 src 文件夹中，main 为工程文件，请查看。

附录：关键功能与代码

```
//布尔检索，求交集
int CMD_AND_ENGINE(){
  int res[max_page]={1};
  for(int i=0;i<max_page;i++)
    res[i]=1;
  for(int cur_string=0;cur_string<search_list_num;cur_string++){
    int re[max_page]={0};
    memset(re,0,sizeof(re));
    search_string(search_list[cur_string],re);
    for(int i=0;i<max_page;i++)
      if(res[i]==1){
        if(re[i]==0)
          res[i]=0;
      }
  }
  int count=0;
  for(int i=0;i<max_page;i++){
    if(res[i]==1){
      search_result[i]=1;
      count++;
    }
  }
  return count;
}
```

```
//布尔检索，求并集
int CMD_OR_ENGINE(){
  int res[max_page]={1};
  for(int i=0;i<max_page;i++)
    res[i]=1;
    for(int cur_string=0;cur_string<search_list_num;cur_string++){
    int re[max_page]={0};
    memset(re,0,sizeof(re));
    search_string(search_list[cur_string],re);
    for(int i=0;i<max_page;i++)
        if(re[i]==1)
          res[i]=1;
  }
  int count=0;
  for(int i=0;i<max_page;i++){
    if(res[i]==1){
      search_result[i]=1;
      count++;
    }
  }
  return count;
}
```

图 2：布尔检索算法

```
int Partition(int low, int high,int *p,int *value){
    int pivotpos = low;
    int pivot = value[p[low]];
    int temp=p[low];
    for (int i=low+1; i<=high; i++) {
        if (value[p[i]]<pivot) {
            pivotpos++;
            if (pivotpos!=i) {
                int temp = p[pivotpos];
                p[pivotpos] = p[i];
                p[i] = temp;
            }
        }
    }
    p[low] = p[pivotpos];
    p[pivotpos] = temp;
    return pivotpos;
}


void QuickSort(int left, int right,int *p,int *value){
    if (left<right) {
        int pivotpos = Partition(left, right,p,value);
        QuickSort(left, pivotpos-1,p,value);
        QuickSort(pivotpos+1, right,p,value);
    }
    return;
}

void mysort(int num,int *p,int *value){
    QuickSort(0,num-1,p,value);
}
```

图 3：入度排序过程 QuickSort 算法

```
class DJVector {
public:
    DJVector(int sz = num_pages); //构造函数
    DJVector(DJVector & L); //构造函数函数
    ~DJVector() { delete[] data; data = NULL; last = -1; } //析构函数
    int Size() const { return maxSize; } //判断vector的大小
    int Search(int & x) const; //寻找元素，返回这个元素在vector中的下标
    bool getData(int i, int & x) const; // 找到vector第i位元素
    int Length() const { return last + 1; } //判断vector的长度
    bool Insert(int i, int x); //把x插入到vector的第i位
    bool Append(int x); //在vector后面添加一个元素
    bool Remove(int i, int & x); //把vector第i位元素删除

private:
    int * data;
    int maxSize; //记录vector的大小
    int last; //最后一个元素的下标
    friend ostream & operator <<(ostream & os, const DJVector & list); //输出函数
};
```

图 4：vector 类