倒排索引 Inverted File Index

1.Structure

搜索引擎主要做三件事：spider，爬虫获取内容； 倒排索引； 文档排序;

方法1

扫描每一个网页，看他是否包含我的关键字?

方法2

建立 Term-Document Incidence Matrix

〖Example〗 Document sets

Doc	Text
1	Gold silver truck
2	Shipment of gold damaged in a fire
3	Delivery of silver arrived in a silver truck
4	Shipment of gold arrived in a truck

	1	2	3	4
a	0	1	1	1
arrived	0	0	1	1
damaged	0	1	0	0
delivery	0	0	1	0
fire	0	1	0	0
gold	1	1	0	1
of	0	1	1	1
in	0	1	1	1
shipment	0	1	0	1
silver	1	0	1	0
truck	1	0	1	1

eg: 查找包含silver & truck?

做与运算即可 1010&1011 = 1010

所以在document 1、3 中存在silver & truck

问题：稀疏矩阵过于稀疏，浪费时间空间

方法3：Compact(精简) Version Inverted File Index

index:有点像指针，a mechanism for locating a given term in a text.

Inverted File：包含指针（存储出现次数和出现文件及位置）

Doc	Text
1	Gold silver truck
2	Shipment of gold damaged in a fire
3	Delivery of silver arrived in a silver truck
4	Shipment of gold arrived in a truck

No.	Term	Times; Documents
1	a	<3; 2,3,4>
2	arrived	<2; 3,4>
3	damaged	<1; 2>
4	delivery	<1; 3>
5	fire	<1; 2>
6	gold	<3; 1,2,4>
7	of	<3; 2,3,4>
8	in	<3; 2,3,4>
9	shipment	<2; 2,4>
10	silver	<2; 1,3>
11	truck	<3; 1,3,4>

Doc	Text
1	Gold silver truck
2	Shipment of gold damaged in a fire
3	Delivery of silver arrived in a silver truck
4	Shipment of gold arrived in a truck

No.	Term	Times; Documents Words
1	a	<3; (2;6),(3;6),(4;6)>
2	arrived	<2; (3;4),(4;4)>
3	damaged	<1; (2;4)>
4	delivery	<1; (3;1)>
5	fire	<1; (2;7)>
6	gold	<3; (1;1),(2;3),(4;3)>
7	of	<3; (2;2),(3;2),(4;2)>
8	in	<3; (2;5),(3;5),(4;5)>
9	shipment	<2; (2;1),(4;1)>
10	silver	<2; (1;2),(3;3,7)>
11	truck	<3; (1;3),(3;8),(4;7)>

为什么要保留 frequency?

查找多个term 时，从frequency低的排除不包含关键词的doc

```
while(read a document D){
    while(read a term T in D){
        if(Find(Dictionary,T)==false) Insert(Dictionary,T);
        Get T's posting list;
        insert a node to T's posting list;
    }
}
Write the inverted index to disk;//index 也很大
```

难点:

// 怎么判断 term? 英语还行，中文真难哇;

Token Analyzer Stop Filter;

// find(Dictionary ,T) 如何快速找到 term

Vocabulary Scanner

// Insert(Dictionary,T);

Vocabulary Insertor

// 写入磁盘

Memory Management

2.Modules

2.1 word stemming and stop filter

word stemming 词干

处理一个词 仅保留root or stem

stop words

不索引它们，而是删除他们 比如 a the it

2.2 accessing the term

s1: search tree (B- trees,B+ trees,tries) 数据量很大的时候红黑树、AVL树都不合适

s2: hashing

pros and cons : faster for a single word but expensive for range search (不同的term hash到不同的地方，不适合范围查找

B+ tree 局部搜索表现更好

2.3存储空间不够怎么办?

先判断存储空间够不够

```
while(read a document D){
    while(read a term T in D){

        if(out of memory){
            write BlockIndex[BlockCnt] to disk;
            BlockCnt ++;
            FreeMemory;
        }

        if(Find(Dictionary,T)==false) Insert(Dictionary,T);
        Get T's posting list;
        insert a node to T's posting list;
    }
}

for(i=0;i<BlockCnt;i++)
    Merge(InvertedIndex,BlockIndex[i]);
// merge all subsets
Write the inverted index to disk;//index 也很大
```

3.topics

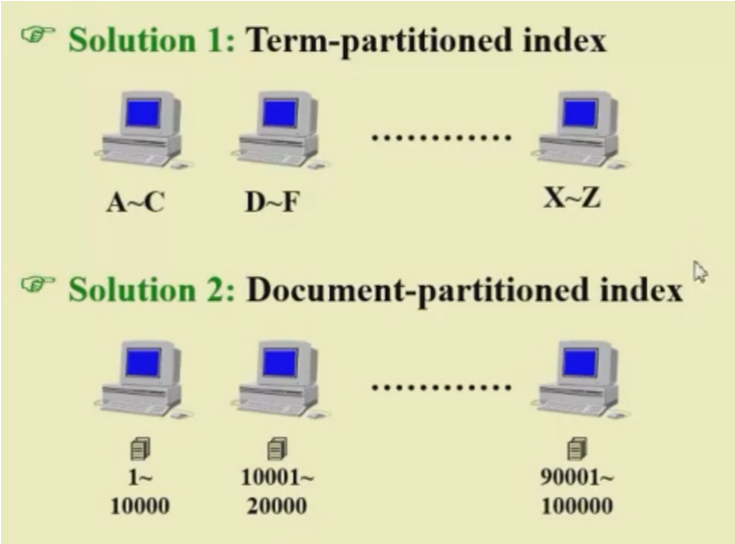
3.1 distributed indexing

(for web-scale indexing， 每一个节点都包括一个子集

s1 : term-partitioned index

s2: document-partitioned index

eg:



3.2 dynamic indexing

需求:

docs 不断加入

更新dictionary中已有postings

新的term 加入到 dictionary 中;

docs 删除;

方法:

main index (大) : search

auxiliary index (小,辅助性的): insert (new docs); search

特定时间 merge 两个index ,clean auxiliary index

困难:

什么时候merge ? 什么时候删除docs?

3.3.compression

怎么定义数组大小? 来存储string?

s1:删除stop words;存储一串字符，和字符的位置;

p1:存储下标数字过大会溢出;

s2:存储gap(差)

3.4 thresholding

(阈值！=阀值)

document: 仅检索 top x documents (ranking by weight)

缺点：

Not feasible for Boolean queries （比如 & | ）
Can miss some relevant documents due to truncation （截断修理）

query: Sort the query terms by their **frequency** in **ascending** order;search according to only some percentage of the original query terms

频率越低，term越重要；只考虑部分term,由你设置 threshold（20%，40% eg)

4.measures

4.1Measures for a search engine

4.1.1How fast does it index
——Number of documents/hour

4.1.2How fast does it search
——Latency as a function of index size(与index size有关，而不是只看多快)

4.1.3Expressveness of query language
——Ability to express complex information needs
——Speed on complex queries

eg: apple - company

4.1.4user happiness

——Data Retrieval Performance Evaluation (after establishing correctness)
Response time
Index space

——Information Retrieval Performance Evaluation

How relevant is the answer set?

(eg: red apple not red ballon)

4.2 Relevance measurement

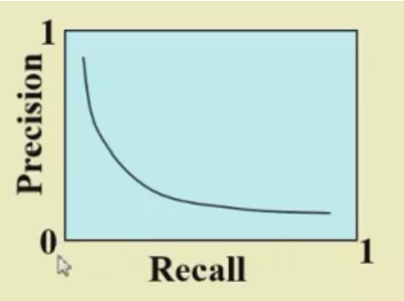
requires 3 elements:
1.A benchmark(标准集) document collection
2.A benchmark suite of queries
3.A binary assessment of either Relevant or Irrelevant for each query-doc pair

	relevent	**irrelevant**
Retrived	RR	IR
not retrieved	RN	IN

precision: P =RR / (RR+IR)

recall: R = RR(RR+RN)

*防止上一个方程只做了一次relevant的检索，敢说自己100%正确 / 100%错误



recall 太高，很多junk也return了，precision下降

理想状态：recall、precision 都很高

How to improve the relevancy of search results?

page rank, semantic（语义） web ...