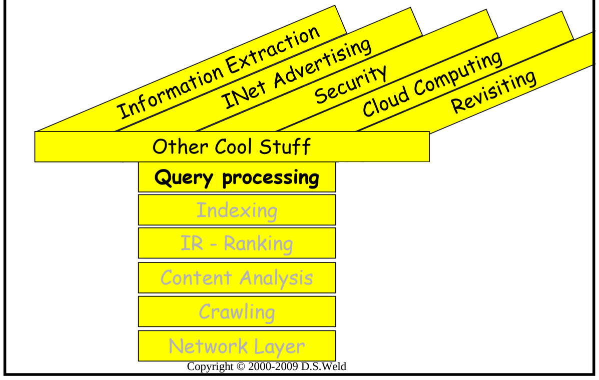
CSE 454 - Case Studies

Design of Alta Vista

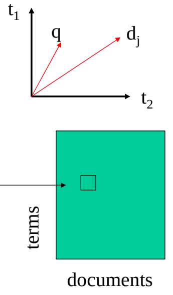
Based on a talk by Mike Burrows

Class Overview



Review

- **Vector Space Representation**
  – Dot Product as Similarity Metric
- **TF-IDF for Computing Weights**
  – $w_{ij} = f(i,j) * \log(N/n_i)$
  – Where q = … word$_i$…
  – N = |docs| n_i = |docs with word$_i$|
- **But How Process Efficiently?**



Retrieval

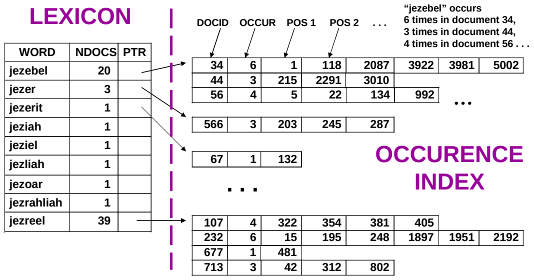
Document-term matrix

$$\begin{array}{c|cccccc|c}
 & t_1 & t_2 & \dots & t_j & \dots & t_m & nf \\
\hline
d_1 & w_{11} & w_{12} & \dots & w_{1j} & \dots & w_{1m} & 1/|d_1| \\
d_2 & w_{21} & w_{22} & \dots & w_{2j} & \dots & w_{2m} & 1/|d_2| \\
 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \\
d_i & w_{i1} & w_{i2} & \dots & w_{ij} & \dots & w_{im} & 1/|d_i| \\
 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \\
d_n & w_{n1} & w_{n2} & \dots & w_{nj} & \dots & w_{nm} & 1/|d_n|
\end{array}$$

w_{ij} is the weight of term t_j in document d_i

Most w_{ij}'s will be zero.

Inverted Files for Multiple Documents



"jezebel" occurs
6 times in document 34,
3 times in document 44,
4 times in document 56 . . .

WORD	NDOCS	PTR
jezebel	20	
jezer	3	
jezerit	1	
jeziah	1	
jeziel	1	
jezliah	1	
jezoar	1	
jezrahliah	1	
jezreel	39	

DOCID	OCCUR	POS 1	POS 2	. . .			
34	6	1	118	2087	3922	3981	5002
44	3	215	2291	3010			
56	4	5	22	134	992		
566	3	203	245	287			
67	1	132					
107	4	322	354	381	405		
232	6	15	195	248	1897	1951	2192
677	1	481					
713	3	42	312	802			

Many Variations Possible

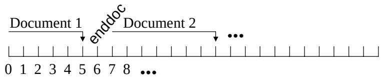
- **Address space (flat, hierarchical)**
  – Alta Vista uses flat approach
- **Record term-position information**
- **Precalculate TF-IDF info**
- **Stored header, font & tag info**
- **Compression strategies**

AltaVista: Inverted Files

- **Map each word to list of locations where it occurs**
- **Words = null-terminated byte strings**
- **Locations = 64 bit unsigned ints**
  - Layer above gives interpretation for location
    - URL
    - Index into text specifying word number
- **Slides adapted from talk by Mike Burrows**

Documents

- **A document is a region of location space**
  - Contiguous
  - No overlap
  - Densely allocated (first doc is location 1)
- **All document structure encoded with words**
  - enddoc at last location of document
  - begintitle, endtitle mark document title



Format of Inverted Files

- **Words ordered lexicographically**
- **Each word followed by list of locations**
- **Common word prefixes are compressed**
- **Locations encoded as deltas**
  - Stored in as few bytes as possible
  - 2 bytes is common
  - Sneaky assembly code for operations on inverted files
    - Pack deltas into aligned 64 bit word
    - First byte contains continuation bits
    - Table lookup on byte => no branch instructs, no mispredicts
    - 35 parallelized instructions/ 64 bit word = 10 cycles/word
- **Index ~ 10% of text size**

Index Stream Readers (ISRs)

- **Interface for**
  - Reading result of query
  - Return ascending sequence of locations
  - Implemented using lazy evaluation
- **Methods**

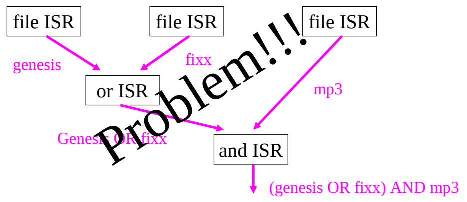
– loc(ISR)	return current location
– next(ISR)	advance to next location
– seek(ISR, X)	advance to next loc after X ⇦ !
– prev(ISR)	return previous location

Processing Simple Queries

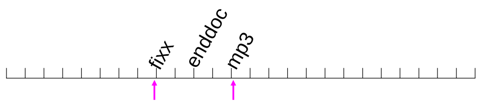
- **User searches for “mp3”**
- **Open ISR on “mp3”**
  - Uses hash table to avoid scanning entire file
- **Next(), next(), next()**
  - returns locations containing the word

Combining ISRs

• **And**	**Compare locs on two streams**
• **Or**	**Merges two or more ISRs**
• **Not**	**Returns locations not in ISR (lazily)**

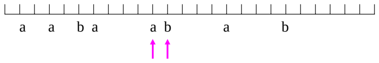


What About File Boundaries?



ISR Constraint Solver

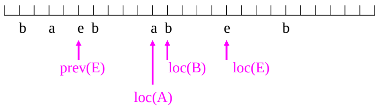
- **Inputs:**
  - – Set of ISRs: A, B, ...
  - – Set of Constraints
- **Constraint Types**
  - – loc(A) ≤ loc(B) + K
  - – prev(A) ≤ loc(B) + K
  - – loc(A) ≤ prev(B) + K
  - – prev(A) ≤ prev(B) + K
- **For example: phrase “a b”**
  - – loc(A) ≤ loc(B), loc(B) ≤ loc(A) + 1



Two words on one page

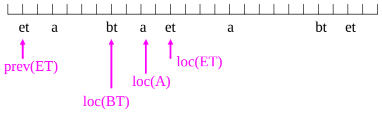
- **Let E be ISR for word** enddoc
- **Constraints for conjunction a AND b**
  - – prev(E) ≤ loc(A)
  - – loc(A) ≤ loc(E)
  - – prev(E) ≤ loc(B)
  - – loc(B) ≤ loc(E)

What if prev(E) Undefined?



Advanced Search

- **Field query: a in Title of page**
- **Let BT, ET be ISRP of words** begintitle, endtitle
- **Constraints:**
  - – loc(BT) ≤ loc(A)
  - – loc(A) ≤ loc(ET)
  - – prev(ET) ≤ loc(BT)



Implementing the Solver

Constraint Types

- – loc(A) ≤ loc(B) + K
- – prev(A) ≤ loc(B) + K
- – loc(A) ≤ prev(B) + K
- – prev(A) ≤ prev(B) + K

Remember: Index Stream Readers

- **Methods**

– loc(ISR)	return current location
– next(ISR)	advance to next location
– seek(ISR, X)	advance to next loc after X
– prev(ISR)	return previous location

Solver Algorithm

loc(ISR)	return cur loc
next(ISR)	adv to nxt loc return it
seek(ISR, X)	adv to nxt loc > X return it
prev(ISR)	return pre loc

```
while (unsatisfied_constraints)
    satisfy_constraint(choose_unsat_constraint())
```

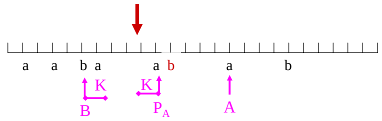
- **To satisfy: loc(A) ≤ loc(B) + K**
  - Execute: seek(B, loc(A) - K)

Solver Algorithm

loc(ISR)	return cur loc
next(ISR)	adv to nxt loc return it
seek(ISR, X)	adv to nxt loc > X return it
prev(ISR)	return pre loc

```
while (unsatisfied_constraints)
    satisfy_constraint(choose_unsat_constraint())
```

- **To satisfy: loc(A) ≤ loc(B) + K**
  - Execute: seek(B, loc(A) - K)
- **To satisfy: prev(A) ≤ loc(B) + K**
  - Execute: seek(B, prev(A) - K)

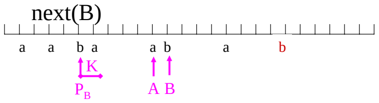


Solver Algorithm

loc(ISR)	return cur loc
next(ISR)	adv to nxt loc return it
seek(ISR, X)	adv to nxt loc > X return it
prev(ISR)	return pre loc

```
while (unsatisfied_constraints)
    satisfy_constraint(choose_unsat_constraint())
```

- **To satisfy: loc(A) ≤ loc(B) + K**
  - Execute: seek(B, loc(A) - K)
- **To satisfy: prev(A) ≤ loc(B) + K**
  - Execute: seek(B, prev(A) - K)
- **To satisfy: loc(A) ≤ prev(B) + K**
  - Execute: seek(B, loc(A) - K),
  - next(B)



Solver Algorithm

loc(ISR)	return cur loc
next(ISR)	adv to nxt loc return it
seek(ISR, X)	adv to nxt loc > X return it
prev(ISR)	return pre loc

```
while (unsatisfied_constraints)
    satisfy_constraint(choose_unsat_constraint())
```

- **To satisfy: loc(A) ≤ loc(B) + K**
  - Execute: seek(B, loc(A) - K)
- **To satisfy: prev(A) ≤ loc(B) + K**
  - Execute: seek(B, prev(A) - K)
- **To satisfy: loc(A) ≤ prev(B) + K**
  - Execute: seek(B, loc(A) - K),
  - next(B)
- **To satisfy: prev(A) ≤ prev(B) + K**
  - Execute: seek(B, prev(A) - K)
  - next(B)

Solver Algorithm

```
while (unsatisfied_constraints)
    satisfy_constraint(choose_unsat_constraint())
```

Heuristic: Which choice advances a stream the furthest?

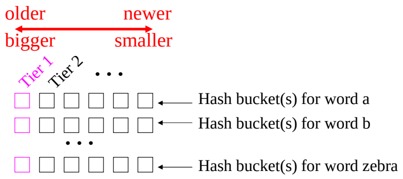
- **To satisfy: loc(A) ≤ loc(B) + K**
  - Execute: seek(B, loc(A) - K)
- **To satisfy: prev(A) ≤ loc(B) + K**
  - Execute: seek(B, prev(A) - K)
- **To satisfy: loc(A) ≤ prev(B) + K**
  - Execute: seek(B, loc(A) - K),
  - next(B)
- **To satisfy: prev(A) ≤ prev(B) + K**
  - Execute: seek(B, prev(A) - K)
  - next(B)

Update

- **Can’t insert in the middle of an inverted file**
- **Must rewrite the entire file**
  - Naïve approach: need space for two copies
  - Slow since file is huge
- **Split data along two dimensions**
  - **Buckets** solve disk space problem
  - **Tiers** alleviate small update problem

Buckets & Tiers

- **Each word is hashed to a bucket**
- **Add new documents by adding a new tier**
  - – Periodically merge tiers, bucket by bucket



older newer

bigger smaller

Tier 1 Tier 2 • • •

Hash bucket(s) for word a

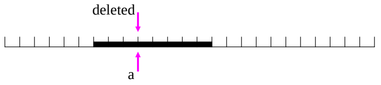
Hash bucket(s) for word b

• • •

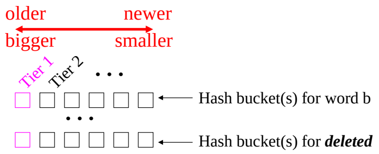
Hash bucket(s) for word zebra

What if Word Removed from Doc?

- **Delete documents by adding** deleted **word**



- **Expunge deletions when merging tier 1**



older newer

bigger smaller

Tier 1 Tier 2 • • •

Hash bucket(s) for word b

• • •

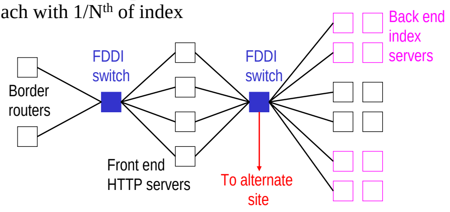
Hash bucket(s) for ***deleted***

Scaling

- **How handle huge traffic?**
  - – AltaVista Search ranked #16
  - – 10,674,000 unique visitors (Dec'99)
- **Scale across N hosts**
  1. Ubiquitous index. Query one host
  2. Split N ways. Query all, merge results
  3. Ubiquitous index. Host handles subrange of locations. Query all, merge results
  4. Hybrids

AltaVista Structure

- **Front ends**
  - – Alpha workstations
- **Back ends**
  - – 4-10 CPU Alpha servers
    - • 8GB RAM, 150GB disk
  - – Organized in groups of 4-10 machines
    - • Each with 1/Nth of index



Back end index servers

FDDI switch

FDDI switch

Border routers

Front end HTTP servers

To alternate site