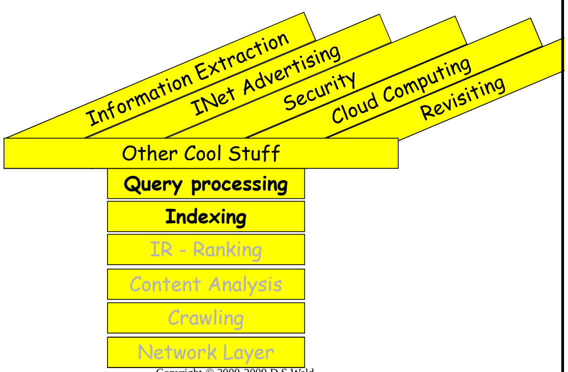
CSE 454

Index Compression
Alta Vista
PageRank

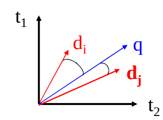
Administrivia

- **No class Tues 10/26**
  - Instead go to today's colloquium
  - Group Meetings
- **Never-Ending Language Learning**
  - Today 3:30pm EEB 105

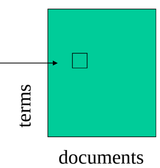
Class Overview



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?**

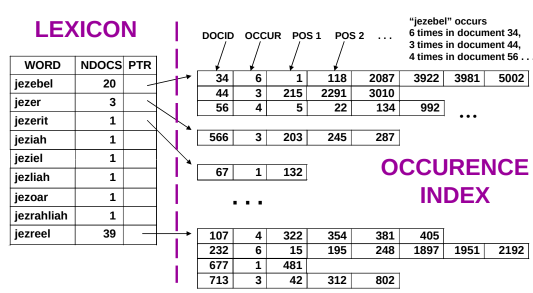
Retrieval

Document-term matrix

	t_1	t_2	. . .	t_j	. . .	t_m	nf
d_1	w_{11}	w_{12}	. . .	w_{1j}	. . .	w_{1m}	$1/\|d_1\|$
d_2	w_{21}	w_{22}	. . .	w_{2j}	. . .	w_{2m}	$1/\|d_2\|$
. . .							
d_i	w_{i1}	w_{i2}	. . .	w_{ij}	. . .	w_{im}	$1/\|d_i\|$
. . .							
d_n	w_{n1}	w_{n2}	. . .	w_{nj}	. . .	w_{nm}	$1/\|d_n\|$

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



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**

Compression

- **What Should We Compress?**
  - Repository
  - Lexicon
  - Inv Index
- **What properties do we want?**
  - Compression ratio
  - Compression speed
  - Decompression speed
  - Memory requirements
  - Pattern matching on compressed text
  - Random access

Inverted File Compression

Each inverted list has the form $< f_t \,;\, d_1 \,,\, d_2 \,,\, d_3 \,,\, \ldots \,,\, d_{f_t} >$

A naïve representation results in a storage overhead of $(f + n) * \lceil \log N \rceil$

This can also be stored as $< f_t; d_1, d_2 - d_1, \ldots, d_{f_t} - d_{f_t - 1} >$

Each difference is called a d-gap. Since $\sum (d - gaps) \leq N$,

each pointer requires fewer than $\lceil \log N \rceil$ bits.

Trick is encoding …. since worst case ….

Assume d-gap representation for the rest of the talk, unless stated otherwise

Slides adapted from Tapas Kanungo and David Mount, Univ Maryland

Text Compression

Two classes of text compression methods

- Symbolwise (or statistical) methods
  - **Estimate probabilities of symbols - modeling step**
  - **Code one symbol at a time - coding step**
  - **Use shorter code for the most likely symbol**
  - **Usually based on either arithmetic or Huffman coding**
- Dictionary methods
  - **Replace fragments of text with a single code word**
  - **Typically an index to an entry in the dictionary.**
    - **eg: Ziv-Lempel coding: replaces strings of characters with a pointer to a previous occurrence of the string.**
  - **No probability estimates needed**

Symbolwise methods are more suited for coding d-gaps

Classifying d-gap Compression Methods:

- **Global: each list compressed using same model**
  - **non-parameterized**: probability distribution for d-gap sizes is predetermined.
  - **parameterized**: probability distribution is adjusted according to certain parameters of the collection.
- **Local: model is adjusted according to some parameter, like the frequency of the term**
- **By definition, local methods are parameterized.**

Conclusion

- **Local methods best**
- **Parameterized global models ~ non-parameterized**
  - Pointers not scattered randomly in file
- **In practice, best index compression algorithm is:**
  - Local Bernoulli method (using Golomb coding)
- **Compressed inverted indices usually faster+smaller than**
  - Signature files
  - Bitmaps

Local < Parameterized Global < Non-parameterized Global

Not by much

CSE 454 - Case Studies

Design of Alta Vista

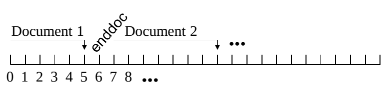
Based on a talk by Mike Burrows

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)

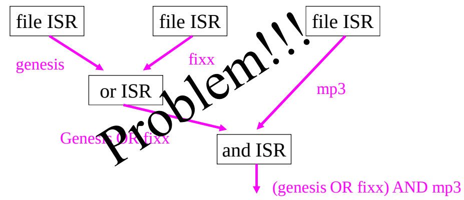
- **Interface for**
  - Reading result of query
  - Return ascending sequence of locations
  - Implemented using lazy evaluation
- **Methods**
  - loc(ISR) — return current location
  - next(ISR) — advance to next location
  - seek(ISR, X) — advance to next loc after X ⇐ !
  - prev(ISR) — return previous location

Processing Simple Queries

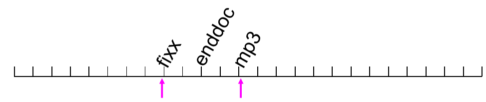
- **User searches for "mp3"**
- **Open ISR on "mp3"**
  - Uses hash table to avoid scanning entire file
- **Next(), next(), next()**
  - returns locations containing the word

Combining ISRs

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

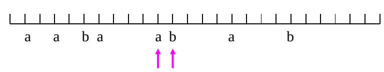


What About File Boundaries?



ISR Constraint Solver

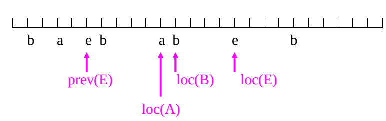
- **Inputs:**
  - Set of ISRs: A, B, ...
  - Set of Constraints
- **Constraint Types**
  - loc(A) ≤ loc(B) + K
  - prev(A) ≤ loc(B) + K
  - loc(A) ≤ prev(B) + K
  - prev(A) ≤ prev(B) + K
- **For example: phrase "a b"**
  - loc(A) ≤ loc(B), loc(B) ≤ loc(A) + 1



Two words on one page

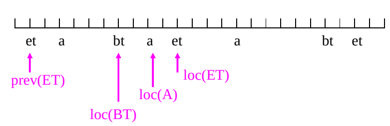
- **Let E be ISR for word** enddoc
- **Constraints for conjunction a AND b**
  - prev(E) ≤ loc(A)
  - loc(A) ≤ loc(E)
  - prev(E) ≤ loc(B)
  - loc(B) ≤ loc(E)

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 : return it
prev(ISR)	return pre loc

while (unsatisfied_constraints)
  satisfy_constraint(choose_unsat_constraint())

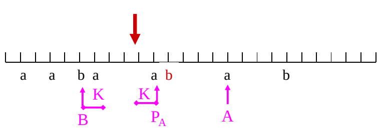
- **To satisfy: loc(A) ≤ loc(B) + K**
  - Execute: seek(B, loc(A) - K)

Solver Algorithm

loc(ISR)	return cur loc
next(ISR)	adv to nxt loc return it
seek(ISR, X)	adv to nxt loc : 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)

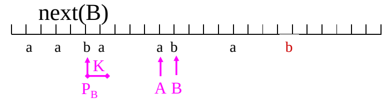


Solver Algorithm

loc(ISR)	return cur loc
next(ISR)	adv to nxt loc return it
seek(ISR, X)	adv to nxt loc : 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 : 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?

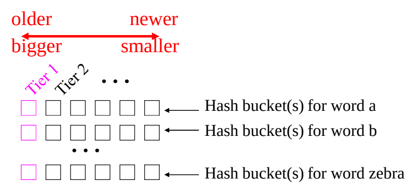
- **To satisfy: loc(A) ≤ loc(B) + K**
  - Execute: seek(B, loc(A) - K)
- **To satisfy: prev(A) ≤ loc(B) + K**
  - Execute: seek(B, prev(A) - K)
- **To satisfy: loc(A) ≤ prev(B) + K**
  - Execute: seek(B, loc(A) - K),
  - next(B)
- **To satisfy: prev(A) ≤ prev(B) + K**
  - Execute: seek(B, prev(A) - K)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

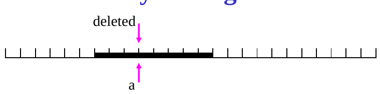
Buckets & Tiers

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

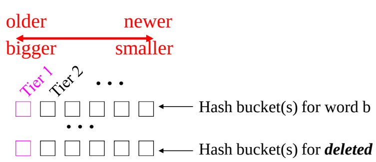


What if Word Removed from Doc?

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



- **Expunge deletions when merging tier 1**



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/N[th] of index