

Henry's awesome summary for comp6714 16s2.

Someone complained a lot about sth(originaly published 5 years ago) I found on google, so I deleted them, if u need it, send email to me: me@changchen.me

Tip for exam: Write down intermediate steps, so that we can give you partial marks even if the final answer is wrong.

Outline available, open it in Tools → Document outline.

Ass1 spec:

<https://drive.google.com/file/d/0B8nBg-44S4maTFV6cVA2U3NpLW8/view?usp=sharing>

Ass2 spec:

<https://drive.google.com/file/d/0B8nBg-44S4maejF4SW4yb3l6NIE/view?usp=sharing>

Log:

03/11/16 Compression
04/11/16 Tolerant Retrieval
05/11/16 Information retrieval
07/11/16 Vector Space Model
08/11/16 Evaluation

09/11/16 Probabilistic Model and Language Model (I am not confident about this part)

10/11/16 Web Search Basics and Link Analysis(done by anonymous hero)

Question: Is Spelling Correction not going to be examined?

See Tolerant Retrieval → 2 (spelling correction)

Contact: me@changchen.me

Finally. Thx everyone who contributes to this summary. didn't expect it at all :)

A. Boolean Model

1. incidence vector

Term-document incidence -> bitwise operation:

	Doc1	D2	D3	D4	D5	D6
Term1	1	0	1	1	0	0

So, Brutus, Caesar and Calpurnia (complemented) bitwise AND.

$$\underline{110100} \text{ AND } \underline{110111} \text{ AND } \underline{101111} = \underline{100100}$$

2. semantics of the query model (AND/OR/NOT, and other operators, e.g., /k, /S)

Question (What's the meaning of semantics, difference with q4 below??)

- 1) Views each document as a set of words
- 2) Is precise: document matches condition or not.

/k, /s see A.4

3. inverted index, positional inverted index

The matrix above (in 1.) is space consuming, so we introduced

1) Inverted index (dictionary and postings).

Brutus	→	2 4 8 16 32 64 128
Caesar	→	1 2 3 5 8 16 21 34
Calpurnia	→	13 16

2) positional inverted index:

fools : ②: <1, 17, 74, 222>; ④: <8, 78, 108, 458>; ⑦: <3, 13, 23, 193>;
rush : 2: <2, 66, 194, 321, 702>; 4: <9, 69, 149, 429, 569>; 7: <4, 14, 404>;
in : 2: <3, 37, 76, 444, 851>; 4: <10, 20, 110, 470, 500>; 7: <5, 15, 25, 195>;

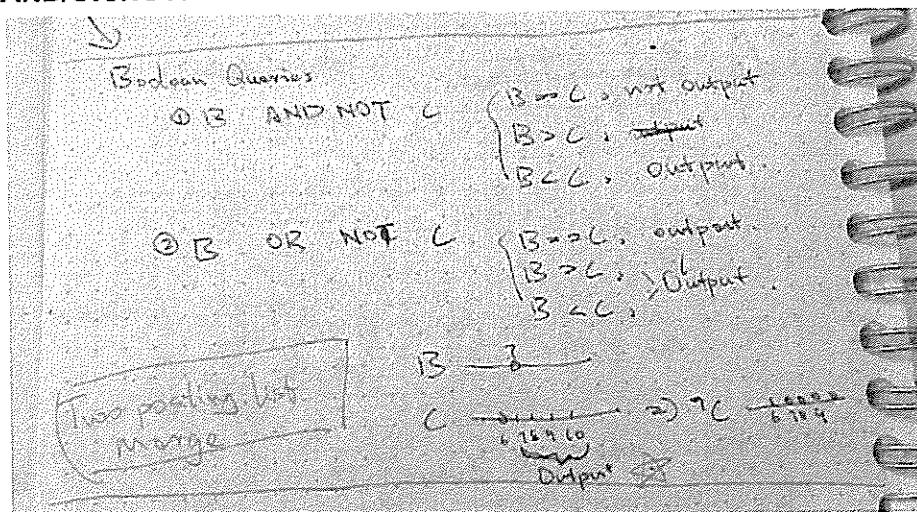
4. query processing methods for basic and advanced boolean queries (including phrase query, queries with /S operator, etc.)

P33?

P30 estimate??

Merge of two postings simultaneously. time complexity: O(x+y):

AND/OR/NOT:



/k means "within k words of":

/S means "in same sentence":

/P means "same paragraph"

The Q3(2) of ass1 and here are **two solutions**:

1) recording word id, sentence id and paragraph id as well as position id (but it's space consuming).

2) including special tokens such as "end of word, end of sen, end of paragraph".

This method is also known as the **extent list approach**.

During the query, once match or anyone encounter end we will move cursor to next begin(same position).

5. query optimization methods (list merge order, skip pointers)

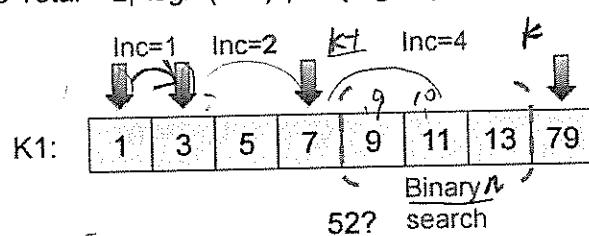
1) **list merge order:** start with the smallest set, same idea as join in database.

2) **skip pointers: Galloping search**

[Stage 1] Doubling the search range until you overshoot

[Stage 2] Perform binary search in the last range

So Total = $2 \lceil \log_2(n+1) \rceil = O(\log_2 n)$



$$f(k) = 1 + 2 + \dots + 2^{k-1} = \frac{1(1-2^k)}{1-2} = 2^k - 1$$

$$\therefore 2^{k-1} < n \leq 2^k$$

$$\therefore 2^{k-1} < n+1 \leq 2^k$$

$$\therefore k-1 < \log_2(n+1)$$

$$\therefore k < \log_2(n+1) + 1$$

Binary =

$$2^k - 2^{k-1} + 1$$

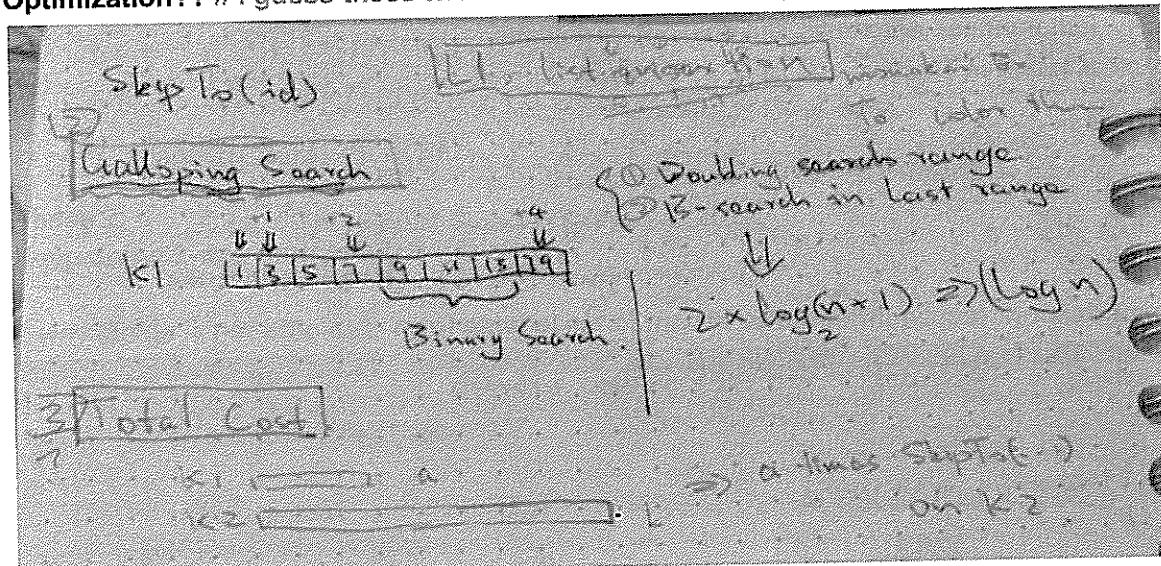
$$= 2^{k-1} + 1$$

$$\log_2(2^{k-1} + 1) < k$$

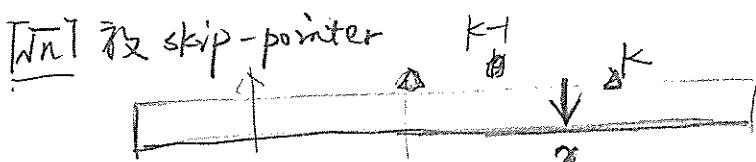
Total $\geq k$

$O(\log n)$

Optimization?? // i guess these two are considered to be optimizations -hengji



Not required: next-word index



$$\therefore (k-1)J_n < x < k \cdot J_n$$

$$\therefore k \cdot J_n < x + J_n$$

$$\therefore k < \frac{x + J_n}{J_n} < \frac{x}{J_n} + 1$$

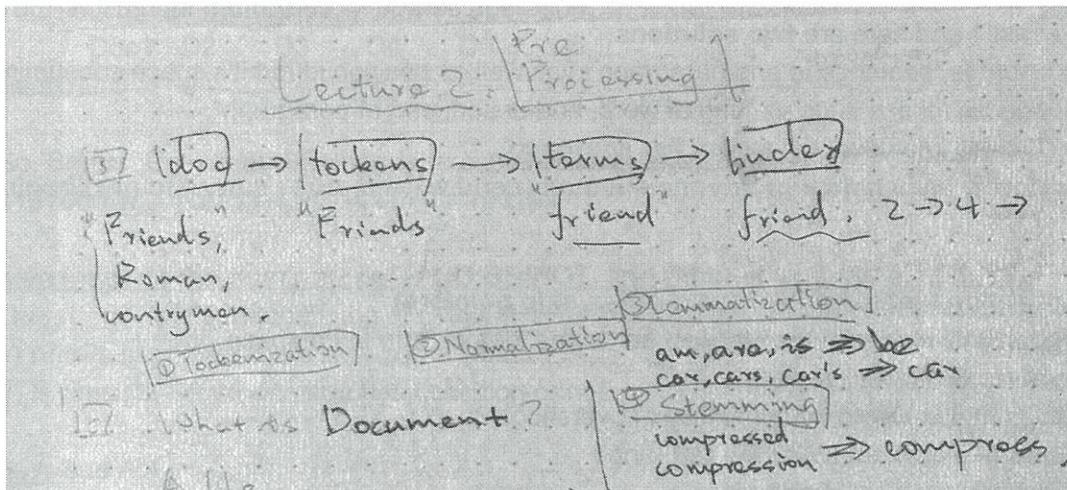
$$\text{binary search} = \frac{\log(J_n)}{J_n}$$

$O(J_n)$

$$\therefore \text{Total Cost} = O(J_n)$$

B. Preprocessing

1. typical preprocessing steps: tokenization, stopword removal, stemming/lemmatization



How does stemming typically affect recall? Why? improve ↑

Normalization increases recall but reduce precision.

* Difference of stemming and lemmatization:

(re edited on 10/11/16 based on lecturer's slides)

Stemming: suggest crude **affix chopping** (Reduce terms to their "roots")

E.g. [automate(s), automatic, automation] → automat

Typical rules in Porter

- * sses → ss
- * ies → i
- * ational → ate
- * tional → tion

Lemmatization: reduce inflectional/variant forms to base form

e.g. [am, are, is] → be; [car, cars, car's] → car

- the boy's cars are different colors → the boy car be different color
- Lemmatization implies doing "proper" reduction to dictionary headword form

2. Not required: details of the (Porter's) stemming algorithm.

. Index Construction

1. Why we need dedicated algorithms to build the index?

Many design decisions in information retrieval are based on the characteristics of hardware.

2. BSBI: Blocked sort-based indexing

Phase 1: Step 1 - step 4

Step 1: all documents \rightarrow 10 blocks

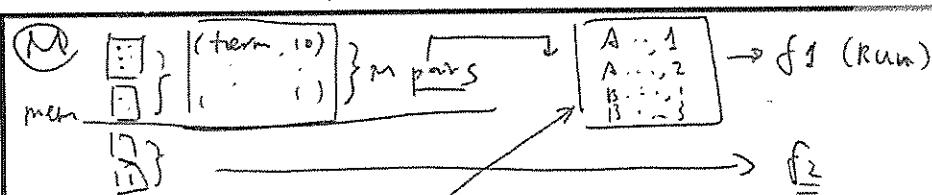
Step 2: read 2 blocks at one time

Step 3: get m pairs and **sort them on the first column**(term id#), it's also the meaning of inverted

Step 4: save them to file.

Total number of files = Corpus size / by M(memory size)

The corpus is huge, so the size of (term, docID) pairs are huge, too. So it is hard to do in-memory sort due to the large size. We need external sort.



BSBINDEXCONSTRUCTION()

```

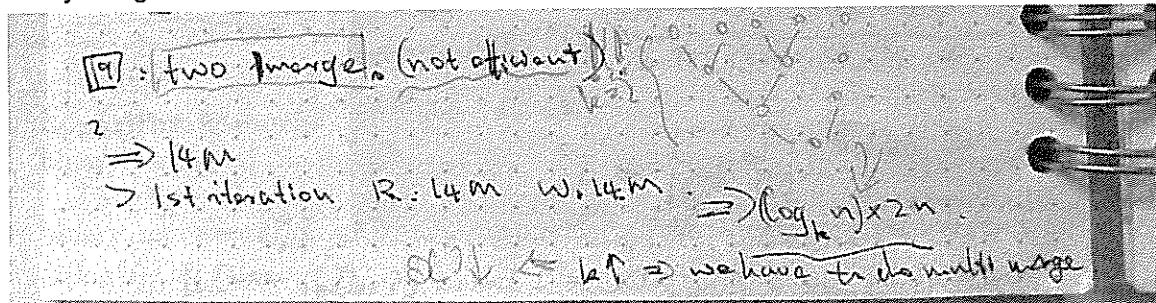
1  n ← 0
2  while (all documents have not been processed)
3    do n ← n + 1
4      block ← PARSENEXTBLOCK()
5      BSBI-INVERT(block)
6      WRITEBLOCKTODISK(block,  $f_n$ )
7      MERGEBLOCKS( $f_1, \dots, f_n; f_{\text{merged}}$ )
  
```

$h = 0$ (Corpus size)
 m

Phase 2:

Keep doing 2-way merge(OR)

2-way merge is **not efficient**:

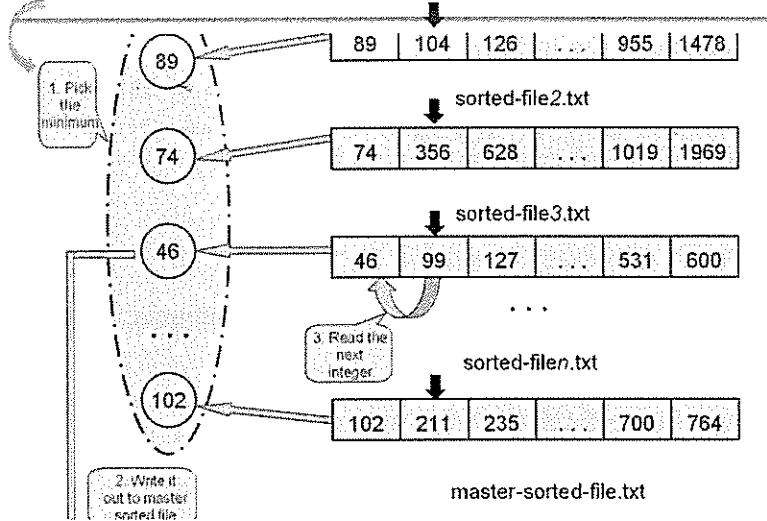


Use a **min-heap** to perform **n-way merge**:

```

.. 3-way merge, initial page size = 2
.. Initial runs = [[15, 46], [19, 93], [64, 73], [33, 80], [26, 73], [22, 77], [27, 83], [7, 5
2], [12, 51], [7, 80], [19, 95], [36]]
.. New runs size = 4. Runs = [[15, 19, 46, 64, 73, 93], [22, 26, 33, 73, 77, 80], [7, 12, 27, 5
1, 52, 83], [7, 19, 36, 80, 95]]
.. New runs size = 2. Runs = [[7, 12, 15, 19, 22, 26, 27, 33, 46, 51, 52, 64, 73, 73, 77, 80, 8
3, 93], [7, 19, 36, 80, 95]]
.. New runs size = 1. Runs = [[7, 7, 12, 15, 19, 19, 22, 26, 27, 33, 36, 46, 51, 52, 64, 73, 73,
77, 80, 80, 83, 93, 95]]
[7, 7, 12, 15, 19, 19, 22, 26, 27, 33, 36, 46, 51, 52, 64, 73, 73, 77, 80, 80, 83, 93, 95]
  
```

Use a min-heap to perform n-way merge



N-way merge time complexity analyse: In summary:

step 1: (finding the minimum element and delete it) is $O(\log k)$,

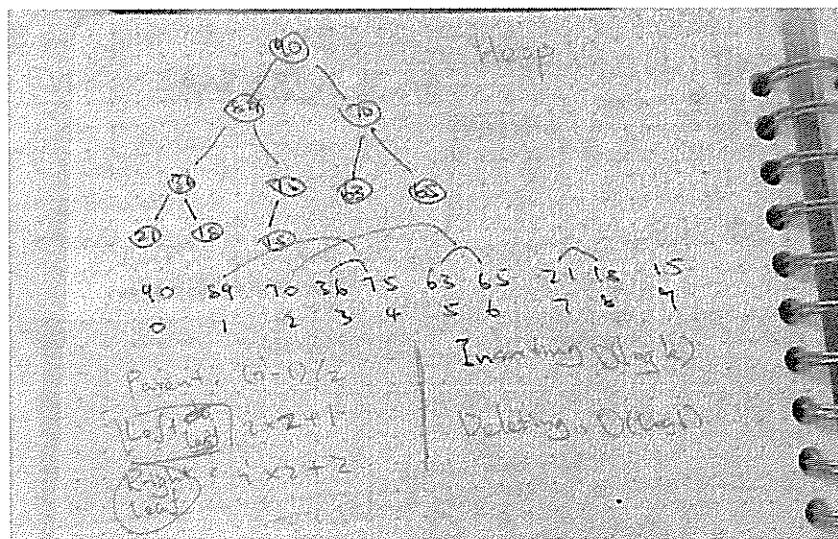
step 2: (inserting a new element from list) is $O(\log k)$, <https://piazza.com/class/iqvpyd4946j42y?cid=14>

As a result, the final cost of heap is $O(\log k)$.

Otherwise we have to scan the whole list to find the min element $O(k)$.

$k \log(k)$

How heap works:

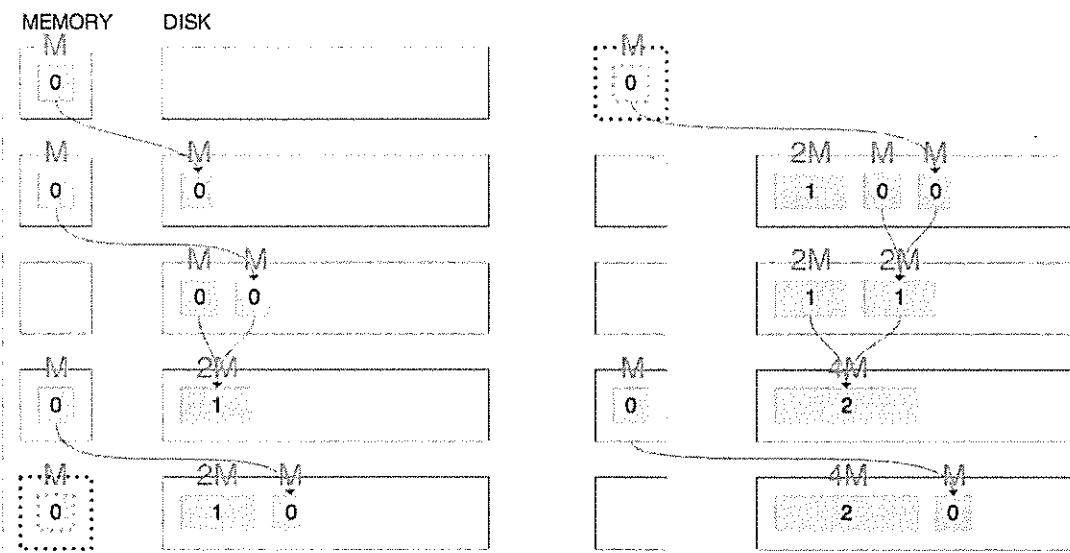


3. SPIMI: Single-pass in-memory indexing

Phase 1: Nothing special but hash-based algorithm, store hash instead of sorted indexes.

Phase 2: Merging of blocks is similar to BSBI

2-WAY LOGARITHMIC MERGE (UNOPTIMIZED)

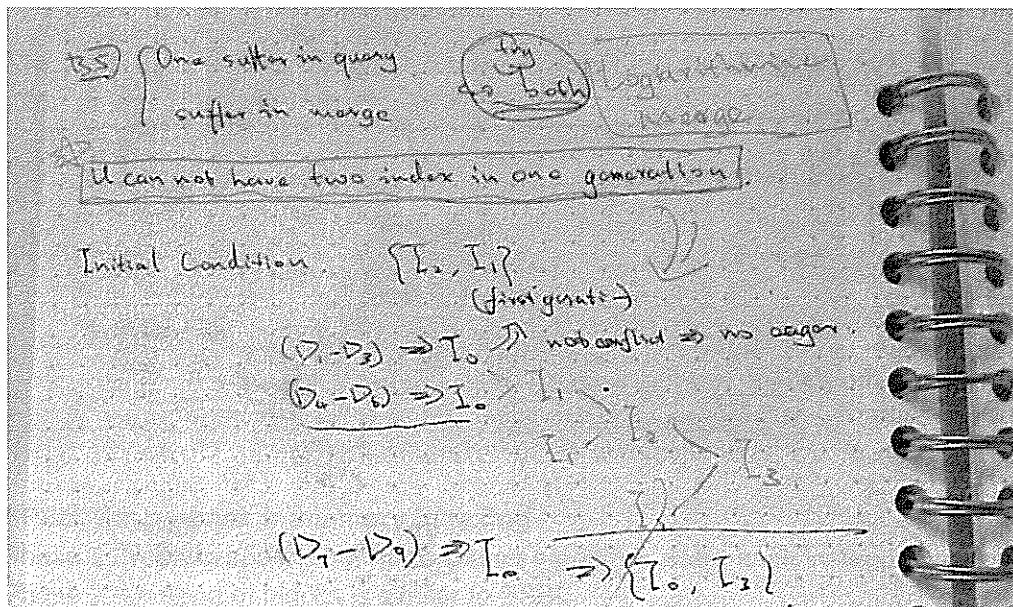


(continues on right side)

```

>>> bin(1)
'0b1'
>>> bin(10)
'0b1010'
>>> bin(10).count('1')
2

```



. Compression

1. Why compression?

The speed of IO is always the bottleneck.

- 1) Use less disk space → save money
- 2) keep more stuff on memory → increase speed
- 3) import from disk to memory, Premise: Decompression algorithms are fast.

2. Heap's Laws and Zipf's Laws

1) Heap's Laws(growing at the square root size):

Unique vocabulary vs total collection size in real documents

- Heaps' law: $M = kT^b$
- M is the size of the vocabulary, T is the number of tokens in the collection
- Typical values: $30 \leq k \leq 100$ and $b \approx 0.5$
- In a log-log plot of vocabulary size M vs. T , Heaps' law predicts a line with slope about $\frac{1}{2}$

Exercise:

- Compute the vocabulary size M for this scenario:
 - Looking at a collection of web pages, you find that there are 3000 different terms in the first 10,000 tokens and 30,000 different terms in the first 1,000,000 tokens.
 - Assume a search engine indexes a total of $20,000,000,000$ (2×10^{10}) pages, containing 200 tokens on average
 - What is the size of the vocabulary of the indexed collection as predicted by Heaps' law?

Solution: $k=30$, so according to the formula, the size of vocabulary is 6×10^7 .

2) estimate the length of posting list

If we sort the posting lists by size, the first is u , second is $u/2$, third is $u/3$

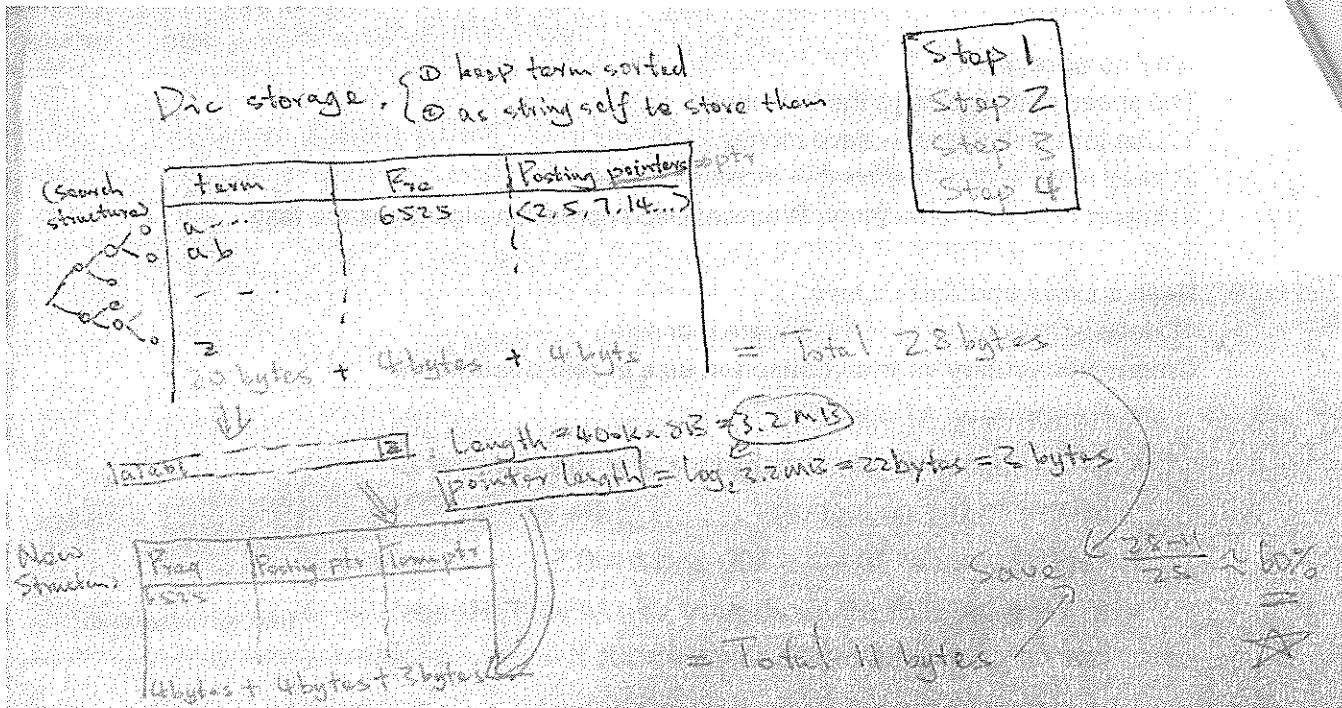
Equivalent: $cf_i = K/i$ where K is a normalizing factor,

so

- $\log cf_i = \log K - \log i$
- Linear relationship between $\log cf_i$ and $\log i$

Any **Exercise??**

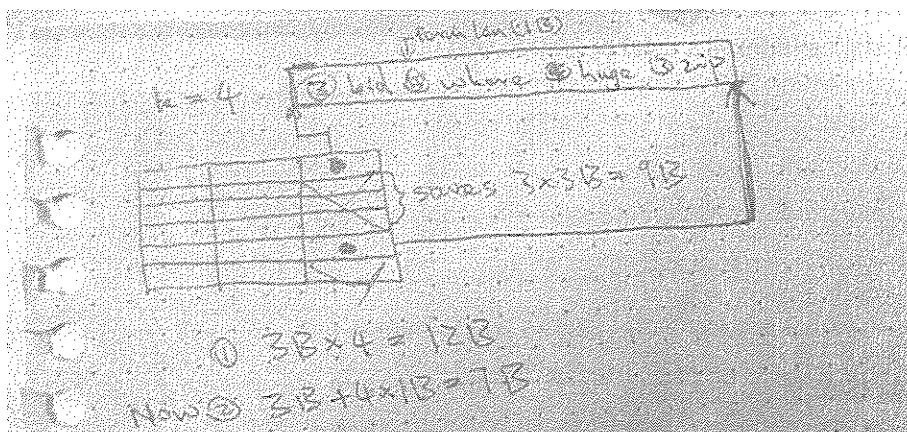
3. Compression & organization of the dictionary



The graph above shows why 1) **Dictionary-as-string** save up to 60% of dictionary space.

2) Blocking: Store pointers to every kth term string

So for **each term** ptr, it saves(4 term positions → 1 block position) $3 \times 3B$, but also lost 4 bytes on **term lengths**.



3) Front coding: Sorted words commonly have long common prefix – store differences suffix only

- * Complete front encoding
 - = (prefix-len, suffix-len, suffix)
- * Partial 3-in-4 front encoding
 - = No encoding/compression for the first string in a block
 - = Enables binary search

Assume previous string is "auto"

String	Complete Front Encoding	Partial 3-in-4 Front Encoding	Aux
8, automata	4, 4, mata	, 8, automata	query
9, automate	7, 1, e	7, 1, e	
10, automatic	7, 2, ic	7, 2, ic	
11, automation	8, 2, on	8, , on	
	aux	2, 1, X	15

Why the last word's length is blank?

The length of the last string in the block can be inferred from the next block pointer. In compression, we try to squeeze out the last bit that is unnecessary.

Advantage of **partial** front encoding: Skip in searching.

"**Enable binary search**" means that binary search can be used with partial front encoding, but *not* on the complete front encoding. This is because the former is used in conjunction with the block-based schemes, and the first string of each block is stored as a plain string (e.g., "automata") here. So **binary search** is possible to jump to the right block (and then do sequential search within). For complete front encoding, you have to decode all the strings before a string in question to get its plain content, hence not possible to do binary search.

Summary:

Fixed: $400K * (20B + 4B + 4B) = 11.2MB$

1) dict: $400K * (4B + 4B + 3B) + 400K * 8B (\text{term average len}) = 7.6MB$

2) Block: $7.6MB - 400K/k * (3B * k - 3B - k) = 7.1MB \quad (k=4)$

3) 5.9MB

(question) P26 search impact....

4. Compression & organization of the inverted index: byte vs bit aligned codes.

Difference of byte and bit:

For byte aligned, *every* input must be encoded in bytes (i.e., cannot be say 13 bits). The rest is bit-aligned.

1. VB(variable Byte) encoding

Continuation bit:

1: it's the last set

0: need to read next set

Example

Hex(214577)=0x00034631

posting	824	829	214577
gaps	5	5	214577
VB code	00000110 10111000	10000101	00001101 00001100 10110001

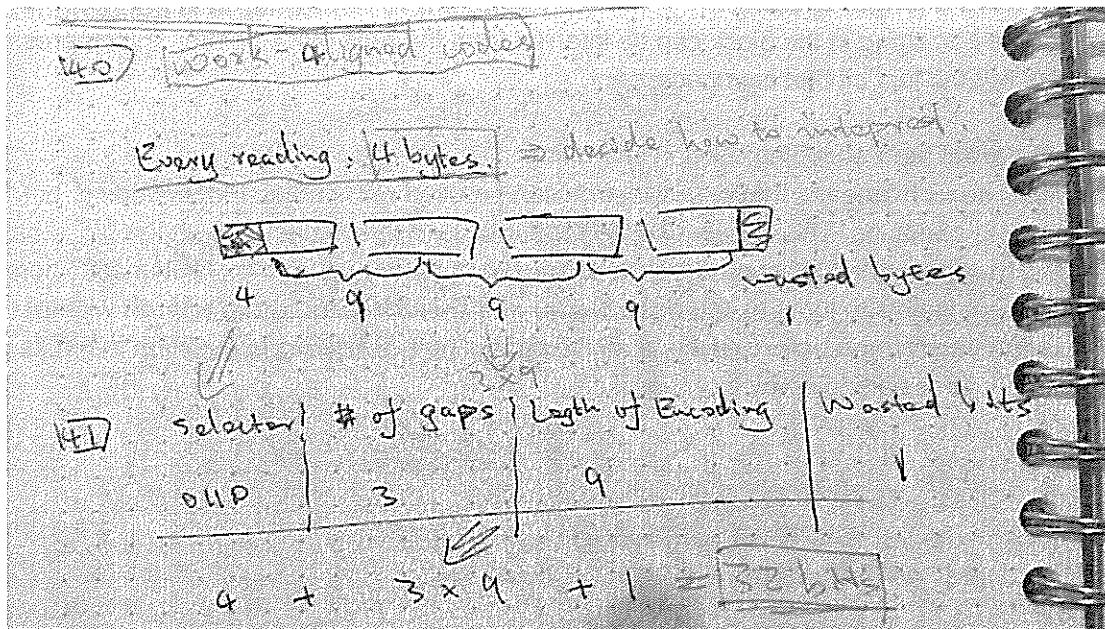
Postings stored as the byte concatenation

000001101011100010000101000011010000110010110001

Key property: VB-encoded postings are uniquely prefix-decodable.

For a small gap (5), VB uses a whole byte.

2. Word-aligned codes



3. Unary Code

Unary = Represent n as n 1s with a final 0 e.g. 3 \rightarrow 1110.

Unary is very efficient for small numbers but expensive for large number.

Binary is more efficient for large numbers but can be ambiguous.

Elias- γ Code

$K = \text{concat}(K_d, K_r)$ where $K_d = \lfloor \log_2(K) \rfloor$ in unary and $K_r = K - 2^{\lfloor \log_2(K) \rfloor}$ in binary

E.g. 15; $K_d = 3 \rightarrow 1110$, $K_r = 7 \rightarrow 111$; 1110 111

γ Code uses $2\lfloor \log_2(K) \rfloor + 1$ bits

Elias- γ Code

$$\text{Num}(K) = 2^{K_d} + K_r$$

$$15 = 2^3 + 7 = 1110 + 111$$

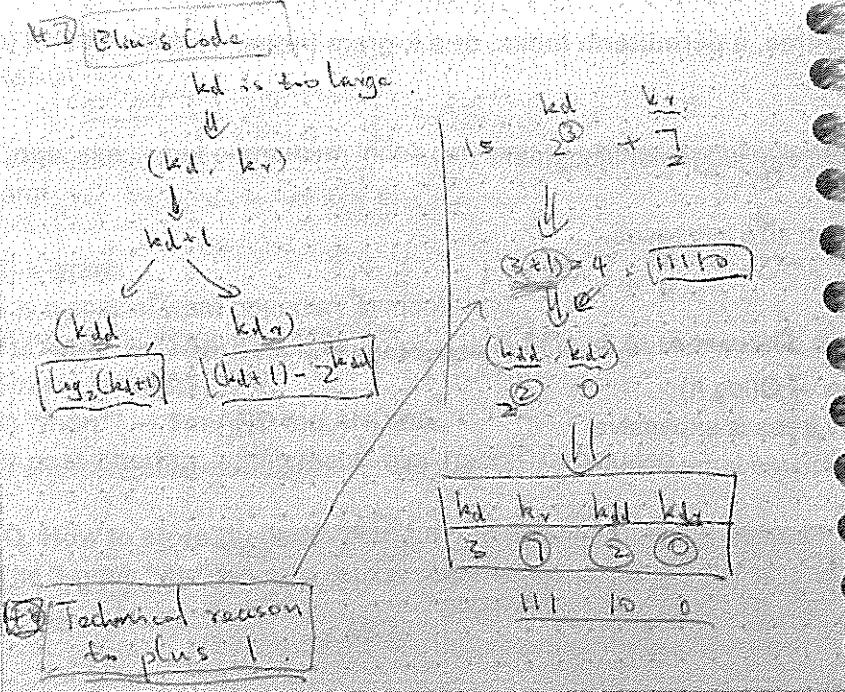
unary code binary code

because a lead is not big.

Elias- δ Code

$K = \text{concat}(K_{dd}, K_{dr}, K_r)$ where $K_r = K - 2^{\lfloor \log_2(K) \rfloor}$ in binary and

$K_{dd} = \lfloor \log_2(K_d + 1) \rfloor$ in unary, $K_{dr} = (K_d + 1) - 2^{\lfloor \log_2(K_d + 1) \rfloor}$ in binary



Not required: Simple, Details of the Golomb code, Google's group varint encoding (the WSDM09 slides Pages 55–63)

E. Tolerant Retrieval

1. Wildcard queries using a B-tree, a permuterm index, or a k-gram index.

1) Hash:

Pro: fast($O(1)$)

- Cons: i. Hard to find variance, e.g. judgement/judgment
 ii. No prefix search
 iii. have to rehash when size increases.

2) B-tree:

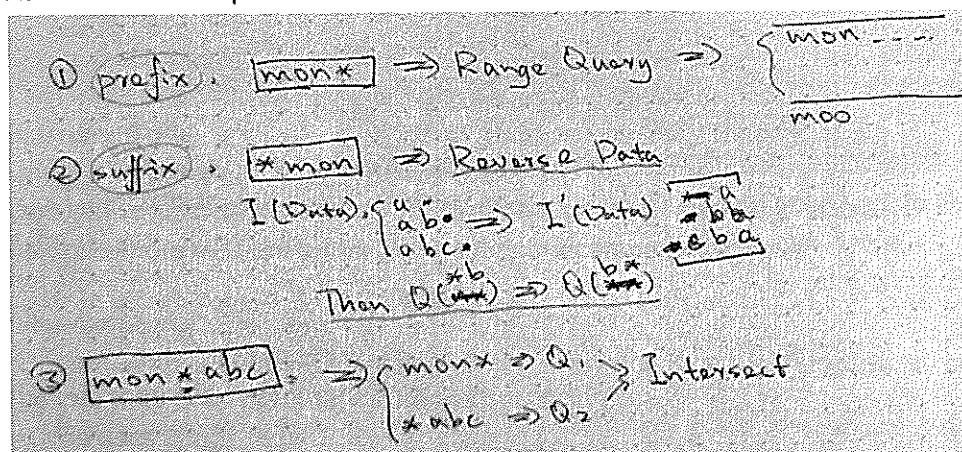
Pro: Prefix search available

- Cons: i: Slow, balanced tree $\rightarrow O(\log n)$

- ii: Rebalancing binary trees is expensive

// just to clarify, binary tree requires large balancing time, but B-tree does not require that much -hengji

Handle wild-card queries:



2.1) tier? Not required?

3) permuterm index:

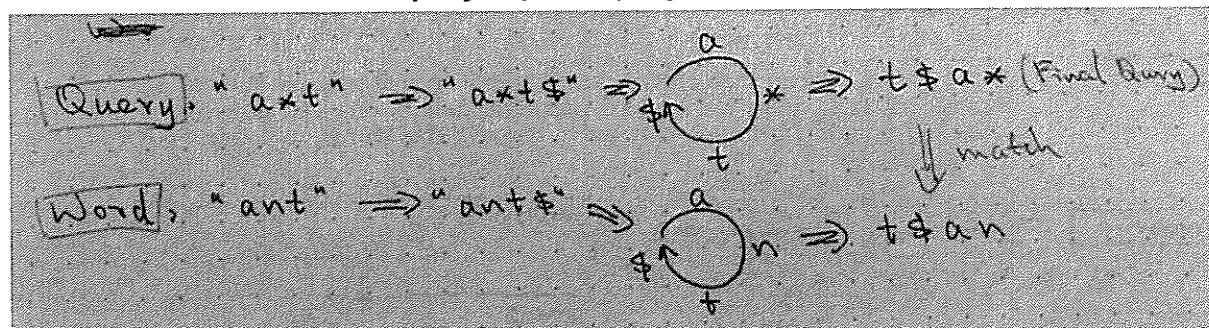
Queries:

- = P Exact match $P\$$
- = P^* Range match $\$P^*$
- = $*P$ Range match $P\*
- = $*P^*$ Range match $P^*\$$
- = P^*Q Range match $Q\$P^*$
- = P^*Q^*R ??? Exercise!

Q: Why not $P^*\* ?
 Query = hel*o
 $P=hel, Q=o$
 Lookup o\\$hel*

How it works(understand it rather remember)?

Our final aim is to transfer all query to prefix query!



a) $*P^* \rightarrow P(*)\$(*) \rightarrow P^*$ (Question: WHY: query is Public * means .+ instead of *+ ?)

// can you rephrase the question? -hengji

b) P*Q*R: (HOW)

R\$PQ*

Cons: simple but permuterm index can lead to a considerable blowup from the number of rotations per term

4) k-gram:

\$ is a special word boundary symbol

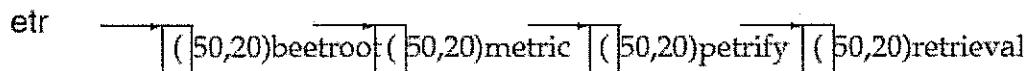
1) "April" → "\$April\$" → \$a,ap,pr,ri,il, I\$, \$i,is,s\$, \$t,th,he,e\$, \$c,cr,ru, ue,el,le,es,st,t\$, \$m,mo,on,nt,h\$

2) "mon*" → \$m AND mo AND on

3) post-filtering: wrong result. e.g., "moon" is included.

(Q. How to store the index and how to do the query??) solved

The 3-gram "etr" would point to vocabulary terms such as metric and retrieval.



2. Context-insensitive spelling correction: edit distance, Jaccard of n-grams. Context-sensitive spelling correction: Noisy channel model

1) edit distance:

```
del[x,y]: count(xy typed as x)
ins[x,y]: count(x typed as xy)
sub[x,y]: count(x typed as y)
trans[x,y]: count(xy typed as yx)
```

2) Jaccard of n-grams

<https://www.cs.utah.edu/~jeffp/teaching/cs5140-S15/cs5140/L4-Jaccard+nGram.pdf>

D1: I am Sam → [I am], [am Sam]

D2: Sam I am → [Sam I], [I am]

So JS(D1, D2) = 1/3

3) Context-sensitive spelling correction: Noisy channel model, My favourite part :)

(Only for non-word spelling error)

Main Steps:

a. Find a set of **candidates** (based on edit distance, e.g. *distance == 1*).

b. Score each word in the candidates based on the noisy channel model.

Candidate Correction	Correct Letter	Error Letter	x w	P(x word)	P(word)	$10^6 \cdot P(x w)P(w)$
actress	t	-	c ct	.000117	.0000231	2.7
cress	-	a	a #	.00000144	.000000544	.00078
caress	ca	ac	ac ca	.00000164	.00000170	.0028
access	c	r	r c	.000000209	.0000916	.019
across	o	e	e o	.0000093	.000299	2.8
acres	-	s	es e	.0000321	.0000318	1.0
acres	-	s	ss s	.0000342	.0000318	1.0

Noisy channel model:

W: Hypothesis(candidates)

X: Observation(wrong words)

$$\begin{aligned}\hat{w} &= \max_{w \in W} P(w|x) \\ &\approx \max_{w \in W} \frac{P(x|w)P(w)}{P(x)} \quad \leftarrow P(x|w)P(w) = P(w|x)P(x) \\ &\approx P(x|w)P(w) \\ &\text{Likelihood} \qquad \qquad \qquad \text{Prior} \\ &\qquad \qquad \qquad \qquad \text{(Occurrence in dict)}\end{aligned}$$

Likelihood:

a. checking the confusion matrix

BETA = 1.0

def score(err, beta = BETA):
 return exp(-beta * err) math. $\exp(x)$
Return e^{-x} .

Takes the context into consideration, also useful for real word spelling error.

Main idea of bigram language model:

$$P(ABC) = P(A)P(B|A)P(C|AB) = P(A)P(B|A)P(C|B)$$

The first = is exact, due to the chain rule. The second = holds as we use the bigram model which assumes $P(X|A, B, \dots, W) = P(X|W)$

$$\begin{aligned}P(A \rightarrow B \rightarrow C) &= P(A) \times P(B|A) \times P(C|B, A) \\&\text{assume position doesn't matter, } P(B|A) \times P(C|B) \quad \text{Mark of assumption.} \\&\text{Bigram Language Model} = P(\text{actions/variables}) \times P(\text{variables/choices})\end{aligned}$$

③ Real-word spelling errors. (an \Rightarrow and)

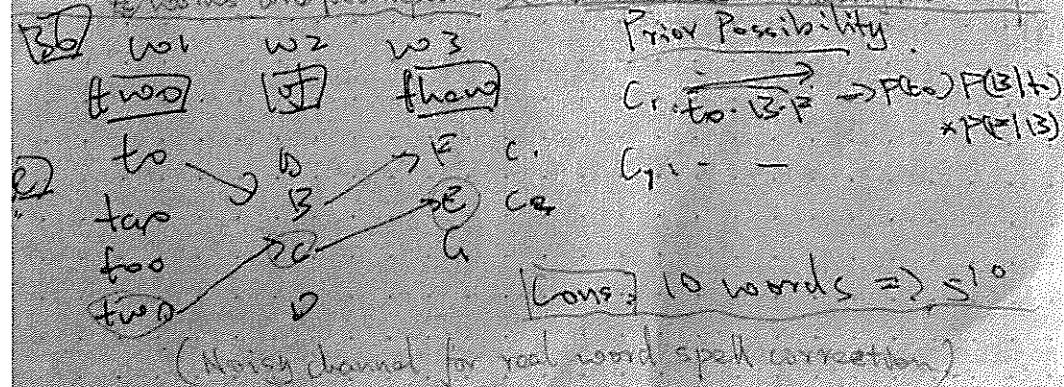
Obscure conditions

→ word stem

→ all single edits that are by words

→ base-spell form

→ words are homophones (θ) → θ X



What is the channel probability for a correctly typed word?

Possibility of no error:

1 error in 20 words: $P(\text{"the"} | \text{"the"}) = (20-1)/20$

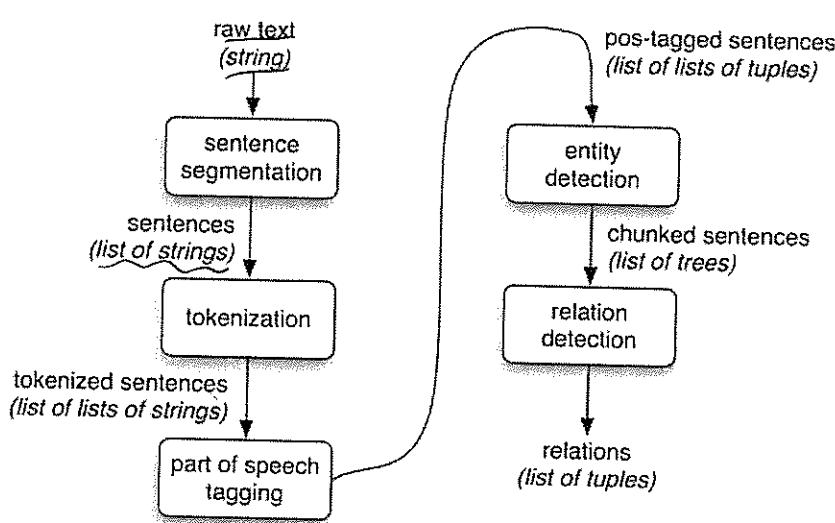
F. Information Extraction

1. Information extraction and its subtasks.

Task: Filling slots in a database from unstructured text. [NAME, TITLE, ORGANIZATION]

IE = Information extraction = segmentation + classification + association

Subtasks:



2. Named entity recognition.

Aim: identify all textual mentions of the named entities.

Named entities: definite noun phrases that refer to specific types of individuals, such as organizations, persons, dates etc..

- Can be broken down into two **sub-tasks**:

1. identifying the boundaries of the NE (segmentation)
2. identifying its type (classification)

Jack Welch	will retire as	CEO of	General Electric	tomorrow. The top role
at the	Connecticut	company	will be filled by	Jeffrey Immelt.

Method:

Look up each word in an appropriate list of names.

Ambiguity:

1. Words have different meanings in different situations.
2. Multi-word names like Stanford University.
3. Name that contains other names: Henry's awesome note.

Therefore, we need to be able to identify the beginning and end of multi-token sequences à **chunking**.

- Chunking useful for entity recognition
- Segment and label multi-token sequences

W e	s a w	t h e	y e l l o w	d o g
PRP	VBD	DT	JJ	NN
NP			NP	

- Each of these larger boxes is called a chunk

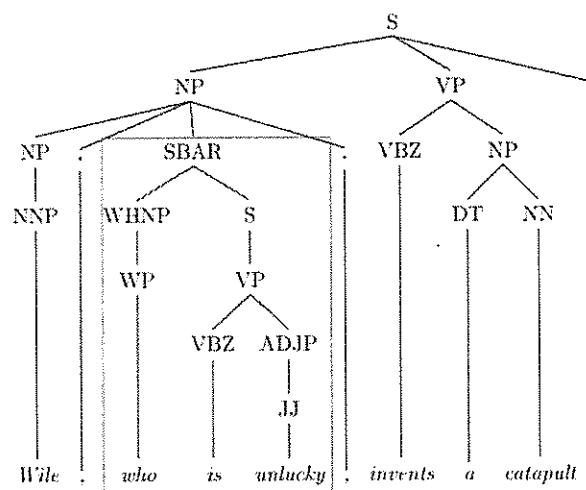
3. Pattern-based Relation Extraction: use of POS, and parsers

POS Tagging:

Word	POS	Chunk	EntityType
U.N.	NNP	I-NP	I-ORG
official	NN	I-NP	O
Eteus	NNP	I-NP	I-PER
heads	VBZ	I-VP	O
for	IN	I-PP	O
Baghdad	NNP	I-NP	I-LOC
.	O	O	O

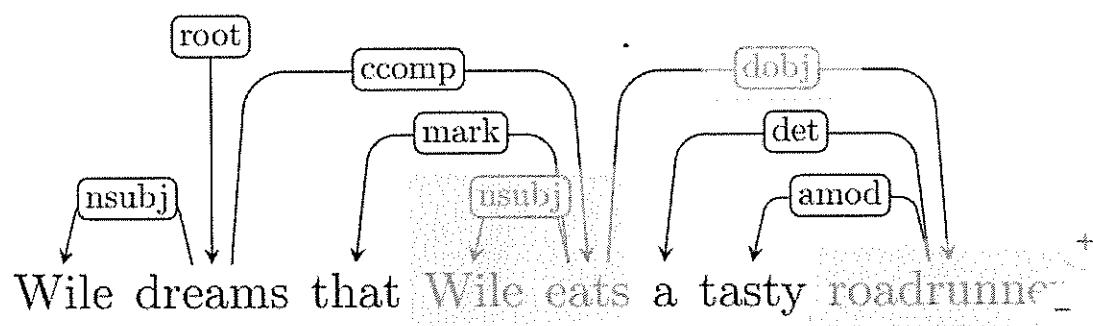
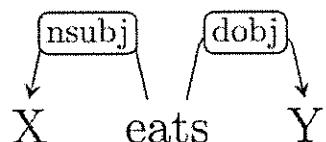
Parser(consider grammar):

1. Constituent parse



This is just a sub-ordinate clause

2. Dependency parse



G. Vector Space Model

1. What is/why ranked retrieval?

1) What:

returning an ordering of the top documents in the collection rather than the complete set

2) Because:

- Normal boolean search results in either too few (don't match) or too many results
- Normal users cannot look through the whole returned docs

3) The reason why we need weighted average:

The diagram shows two tables representing document vectors. The first table, labeled 'Q', has columns 'a' and 'b'. Row 1 (d₁) has values 2 and 3, with a sum of 5 indicated. Row 2 (d₂) has values 5 and 8, with a sum of 13 indicated. The second table, labeled '(B into)', has columns 'a' and 'b'. Row 1 (d₁) has values 7 and 3. Row 2 (d₂) has values 4 and 8. Arrows labeled 'w_a' and 'w_b' point from the second row to the first row, indicating a weighted average calculation. The text 'Scoring' is written above the second table, and 'weighted average' is written below it, with a curved arrow pointing from 'weighted average' to 'Scoring'. A circled 'Reason' is at the bottom right.

2. raw and normalized tf, idf

1) tf (term frequency)

$\text{Raw } tf = tf_{t,d} = \text{How many times term}(t) \text{ occurs in doc}(d)$

Then $tf = \log(\text{raw } tf) + 1$ or $tf = \log(\text{raw } tf + 1)$

2) df (document frequency): How many documents that contain t)

Raw df → raw idf → idf

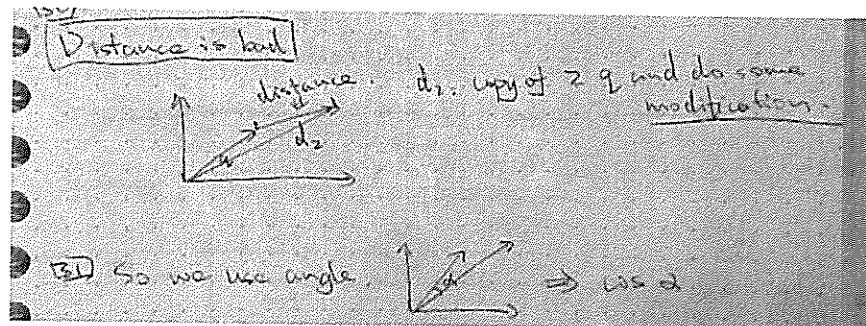
The diagram illustrates the calculation of idf from raw df. It shows a flow from 'Raw df' to 'raw idf' to 'idf'. The formula $tf = \log(N + 1) \rightarrow \log \frac{N}{df}$ is shown, with a note that N is the number of documents. A handwritten note on the right says $idf = \log \frac{N}{df}$.

1) + 2) → The final tf-idf weighting:

$$\text{Score}(q, d) = \sum_{t \in q \cap d} \underline{tf \cdot idf_{t,d}}$$

3. cosine similarity

Because distance is a bad idea:



My useful question about this part:

<https://piazza.com/class/lqvpyd4946j42y?cid=108>

And Lab for this part(just do this lab and u will understand everything):

<https://github.com/DBWangGroupUNSW/COMP6714/blob/master/L8%20-%20tf-idf.ipynb>

Calculation of cosine of two vectors.

Step 1: Length □ normalization

$$\text{Total length} = \|\vec{x}\|_2 = \sqrt{\sum_i x_i^2}$$

Then get the normalized value by dividing this length.

```
import math
def l2_normalizer(vec):
    denom = np.sum([el**2 for el in vec])
    return [(el / math.sqrt(denom)) for el in vec]
```

Step 2: cosine(query,document)

$$\cos(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{\|\vec{q}\| \|\vec{d}\|} = \frac{\vec{q} \cdot \vec{d}}{\|\vec{q}\| \|\vec{d}\|} = \frac{\sum_{i=1}^{|R|} q_i d_i}{\sqrt{\sum_{i=1}^{|R|} q_i^2} \sqrt{\sum_{i=1}^{|R|} d_i^2}}$$

For example:

Normalized query vector: [0, 0, 0, 0.4, 0.1, 0.3, 0]

Normalized doc vector: [0, 0.1, 0, 0.1, 0.5, 0, 0]

$$\cos(q, d) = (0.4 \times 0.1 + 0.1 \times 0.5) / (1 \times 1)$$

4. tf-idf variants (using SMART notation):

Format is ddd.ddd for term frequency, document frequency, normalization coding structure in weighting formula of document and query e.g., Inc.ltc means documents use logarithm term freq, no idf weighting and cosine normalization while queries use logarithm tf, idf and cosine normalization

Term frequency	Document frequency	Normalization
n (natural) $tf_{t,d}$	n (no)	1
t (logarithm) $1 + \log(tf_{t,d})$	t (idf) $\log \frac{N}{df_t}$	c (cosine) $\frac{1}{\sqrt{w_1^2 + w_2^2 + \dots + w_n^2}}$
a (augmented) $0.5 + \frac{0.5 \times tf_{t,d}}{\max_t(tf_{t,d})}$	p (prob idf) $\max\{0, \log \frac{N - df_t}{df_t}\}$	u (pivoted unique) $1/u$
b (boolean) $\begin{cases} 1 & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$.	b (byte size) $1/\text{CharLength}^\alpha$, $\alpha < 1$
L (log ave) $\frac{1 + \log(tf_{t,d})}{1 + \log(\text{ave}_{t \in d}(tf_{t,d}))}$		

We implement an equivalent form of the Inc.ltc model. Therefore, there is no idf weighting on the query terms, since it is already used in the document matrix.

Doc → [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

Query → [0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1]

tf-idf example: Inc.ltc

Document: car insurance auto insurance
 Query: best car insurance

Term	Query					Document			Prod	
	tf-raw	tf-wt	df	idf	wt	n'lize	tf-raw	tf-wt	wt	n'lize
auto	0	0	5000	2.3	0	0	1	1	1	0.52
best	1	1	50000	1.3	1.3	0.34	0	0	0	0
car	1	1	10000	2.0	2.0	0.52	1	1	1	0.52
insurance	1	1	1000	3.0	3.0	0.78	2	1.3	1.3	0.68
										0.53

Exercise: what is N , the number of docs?

$$\text{Doc length} = \sqrt{1^2 + 0^2 + 1^2 + 13^2} \approx 1.92$$

$$\text{Score} = 0+0+0.27+0.53 = 0.8$$

5. Challenges of query (add by myself):

- 1) CANDIDATES \rightarrow 80% (10 million doc) \rightarrow get result of 8 million docs
- 2) Query: $R(a \vee b \vee c)$, if c is a very common word \rightarrow score a large number of docs \rightarrow large df \rightarrow small idf \rightarrow small weight \rightarrow contribute to the final score is low
- 3) inverted index is too large \rightarrow too much I/O time

See 8(4) for optimization.

6. exact & approximate query optimization methods (heap-based top-k)

Get the top 7 numbers in $[1, 2, 45, 23, 2, 12, 5, 2, 9, 6]$

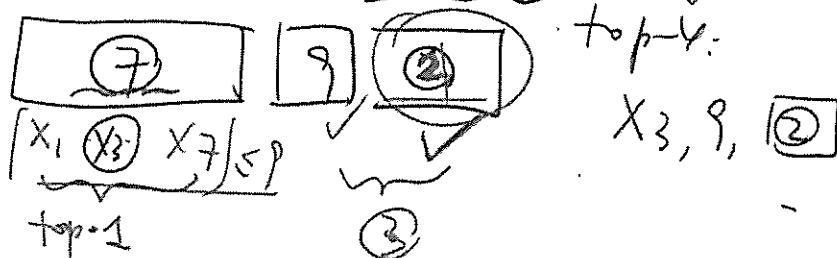
1) min heap: $n \log k$

2) if the entire array in memory: quick select

$O(n+k \log k)$

top-4:

$X_3, 9, 2$



Time complexity: $n + K \log K$ (sorting)

7. basic query processing method: document-at-a-time vs term-at-a-time

[4: 2] means this term occurs twice in the doc4.

DAAT:

salt	1:1			4:1
water	1:1	2:1		4:1
tropical	1:2	2:2	3:1	
score	1:4	2:3	3:1	4:2

TAAT:

salt	1:1	4:1
partial scores	1:1	4:1

old partial scores = $\frac{1}{1} : \frac{4}{1}$

old partial scores	1:1	4:1
water	1:1	2:1
new partial scores	1:2	2:1

old partial scores = $\frac{1}{2} : \frac{2}{1} : \frac{4}{2}$

old partial scores	1:2	2:1	4:2
tropical	1:2	2:2	3:1
final scores	1:4	2:3	3:1

8. algorithm, MaxScore algorithm, etc.)

1) Conjunction method:

Requires the result document containing all the query terms (i.e., conjunctive Boolean queries)

(20) Conjunction Processing

$$q = a \wedge b \wedge c$$

result(q) = $a \wedge b \wedge c$

\Downarrow

$\text{[AND]} \leftarrow \text{[and]}$

2) Threshold Methods

(21) Threshold Method, exact \Rightarrow always give correct answer.

1000 documents a min score can ever get :

67.14% : $\frac{\text{ass1}}{\text{ass2}} = \frac{75\%}{65\%} = 1.15$

$\frac{\text{proj1}}{\text{exam}} = \frac{70\%}{60\%} = 1.16$

student	S1	S2	S3
ass1	10	30	70
ass2	10	50	60
proj1	(0/100)	(0/100)	
exam	(0/100)	(0/100)	

$\max \left[\frac{20}{4}, \frac{22}{4} \right] = \frac{22}{4}$

we can predict S2 will be higher than S1

3) Maxscore methods

Ass2 Q1

4) Index elimination(optimization)

Only consider docs containing any query terms

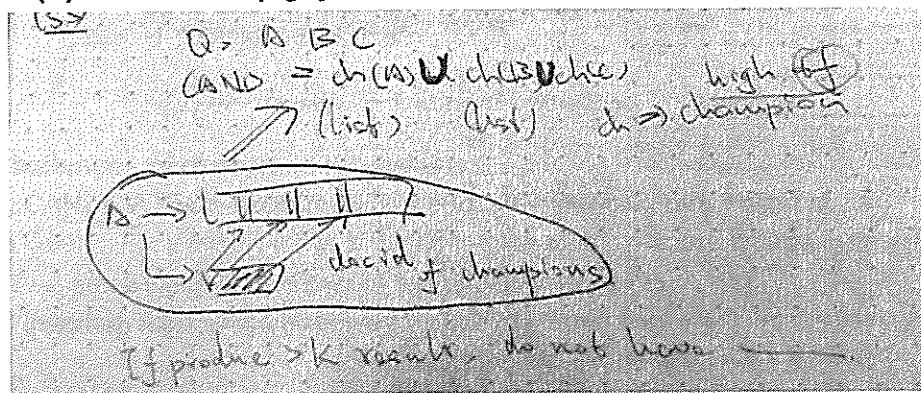
Only consider high-idf query terms

Only consider docs containing many query terms

Only consider docs containing any query terms

5) Champion lists

$ch(A)$ means the top guys of documents that contain A.



H. Evaluation

1. Existing method to prepare for the benchmark dataset, queries, and ground truth

- 1) benchmark dataset(correct answer):

Standard relevance benchmarks

- TREC - National Institute of Standards and Technology (NIST) has run a large IR test bed for many years
- Reuters and other benchmark doc collections used
- "Retrieval tasks" specified
 - sometimes as queries
- Human experts mark, for each query and for each doc, Relevant or Nonrelevant
 - or at least for subset of docs that some system returned for that query

- 2) queries:

- 3) ground truth((1)+(2)?):

2. Efficiency of a search engine(P4):

- 1) index: Number of documents/hour

- 2) How fast of search:

Give 100ms then evaluate how many results returned

- 3) Expressiveness of query language

Ability to express complex information needs

Speed on complex queries

3. For unranked results: Precision, recall, F-measure

$$\underline{P} = \frac{\# \text{relevant doc in results}}{\# \text{total retrieved docs in query results}}$$

$$\underline{R} = \frac{\# \text{relevant doc in results}}{\# \text{total relevant docs in system}}$$

$$\underline{F} = \frac{1}{\frac{\alpha}{P} + \frac{(1-\alpha)}{R}} = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} = \frac{2PR}{(P+R)}$$

Balanced F1 measure is $\alpha=0.5$, $\beta=1$

Unranked retrieval evaluation

3-2) / 3

Correct $GT = \{D_1, D_2, D_3\}$

Query $S_1 = \{D_1, D_2, D_3, D_4\}$ set.

Query \Rightarrow $\{D_2, D_4\} \cap \{D_1, D_3\}$ \Leftrightarrow Ground truth.

~~Correctness based on all queries~~

$$\text{Precision} = \frac{2}{4}$$

$$\text{Recall (Missing)} = \frac{2}{3}$$

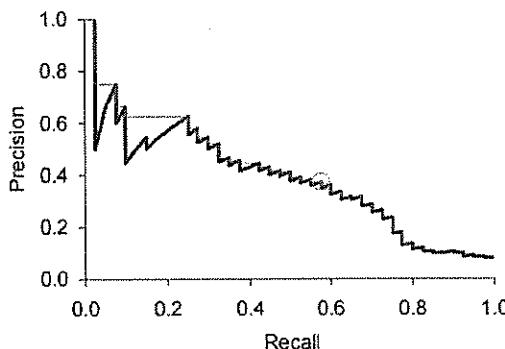
possible abuse of the system.

	Relevant	Non-relevant	Precision	Recall
Retrieved	tp	fp		
Not Retrieved	fn	tn		
			tp / (tp + fp)	tp / (tp + fn)

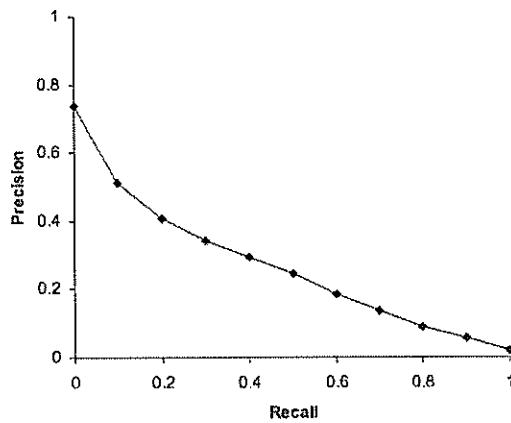
4. For ranked results: precision-recall graph, 11-point interpolated precision, MAP, etc.

$$p_{interp}(r) = \max_{r' \geq r} p(r')$$

- 1) **precision-recall graph** (the red line is interpolated precision)



- 2) **11-point interpolated precision**



how do you calculate this from the interpolations?? anyone?

It's uninterpolated. I think/

just get the 11 points(1, 0.1, 0.2, ... 0.9, 1.0) and connect them. but how to

calculate the value of precision? the plots are (0,~7.5)(0.1,~0.5)...

3) MAP and Kappa(is this required)

MAP - Ex result = R R N N N N N N R N R N N N R N N N N R,

$$\text{total relevant doc} = 8; MAP = \frac{1}{8} \times \left(\frac{1}{1} + \frac{2}{2} + \frac{3}{9} + \frac{4}{11} + \frac{5}{15} + \frac{6}{20} \right)$$

Kappa measure agreement between judges

$$Kappa = \frac{[P(A) - P(E)]}{[1 - P(E)]}; A = \text{all judges agree same point}$$

Total = 400	Judge 2: R	Judge 2: NR
Judge 1: R	300	20
Judge 1: NR	10	70

$$P(A) = \frac{300+70}{400} = 0.9250; P(NR) = \frac{10+70+20+70}{400+400} = 0.2125$$

$$P(R) = \frac{300+20+300+10}{400+400} = 0.7875;$$

$$P(E) = P(NR)^2 + P(R)^2 = (0.2125)^2 + (0.7875)^2 = 0.6653$$

$$Kappa = \frac{(0.9250 - 0.6653)}{(1 - 0.6653)} = 0.7759;$$

Kappa > 0.8 = good agreement; 0.67 < Kappa < 0.8 = tentative

$$P(A) = \frac{370}{400}$$

$$P(R) = \frac{320+310}{800}$$

$$P(NR) = \frac{80+90}{800}$$

$$P(E) = \frac{370+310}{800}$$

I. Probabilistic Model and Language Model

1. probability ranking principle (intuitively, how to rank documents and when to stop)
 (not sure about this part, headache about math)

1) We can use this P to present the possibility of how relevant the doc is, then sort them.

■ Idea: Rank by probability of relevance of the document w.r.t. information need

$$\boxed{\Pr(\text{relevant} \mid \text{document}, \text{query})}$$

$R=1/k=0$ variable

2) Stop at ≥ 0.5 means only output D1(0.7) not D2(0.3).

$$P_1 \quad \Pr(R=1 \mid Q, D_1) = 0.7$$

$$P_2 \quad \Pr(R=1 \mid Q, D_2) = 0.3$$

$$\textcircled{1} \quad \Pr(R \mid Q, D_i) \rightarrow \text{answer?} \quad \textcircled{1} \text{ sort } \checkmark \text{ P vals}$$

$$\textcircled{2} \quad \Pr(R \mid Q, D) \geq 0.5$$

3) 0.5 is because of:

Bayes' Optimal Decision Rule

$$x \text{ is relevant iff } p(R|x) > p(NR|x)$$

4) The problem is how do we estimate $\Pr(\text{relevant} \mid \text{doc, query})$

a.

$$\phi(R, x) = \frac{p(x|R)p(R)}{p(x)}$$

b. Odds:

$$\text{odds}(p) = \frac{p}{1-p}$$

c. each document is a vector (only difference is that the x is either 1 or 0)

(Then I was totally lost in this part...give up)

2. derivation of the ranking formula of the probabilistic model
 (check the echo video of w11(end) w12(start))

3. the BM25 method

$$RSV_d = \sum_{t \in Q} idf_t \cdot \frac{(k_1 + 1)tf_{t,d}}{k_1 \left((1 - b) + b \frac{L_d}{L_{ave}} \right) + tf_{t,d}} \cdot \frac{(k_3 + 1)tf_{t,Q}}{k_3 + tf_{t,Q}}$$

4. Query-likelihood unigram language model with Jelinek-Mercer smoothing.

$$p(Q, d) = p(d) \prod_{t \in Q} ((1 - \lambda)p(t|d) + \lambda p(t|M_d))$$

$p(d|d) = p(d) - p(\alpha|d)$ $p(t|d)$

5. Binary independence model

(check the echo video of w11(end) w12(start))

$I(NR, q)$

constant

variables

$O(R|Q, \vec{r}) = \frac{p(R|Q, \vec{r})}{p(NR|Q, \vec{r})}$

$= \frac{p(R|Q)}{p(NR|Q)} \cdot \frac{p(\vec{r}|R, Q)}{p(\vec{r}|NR, Q)}$

$= O(p(R|Q)) \cdot \prod_{i=1}^V \frac{p(x_i|R, Q)}{p(x_i|NR, Q)}$

$= O(p(R|Q)) \cdot \prod_{x_i=1}^V \frac{p(x_i=1|R, Q)}{p(x_i=1|NR, Q)} \cdot \prod_{x_i=0}^V \frac{p(x_i=0|R, Q)}{p(x_i=0|NR, Q)}$ (4)

$= O(p(R|Q)) \cdot \prod_{x_i=1}^V \frac{p_i}{r_i} \cdot \prod_{x_i=0}^V \frac{1-p_i}{1-r_i}$ (5)

$= O(p(R|Q)) \cdot \prod_{x_i=1, x_i \in Q}^V \frac{p_i}{r_i} \cdot \prod_{x_i=0, x_i \in Q}^V \frac{1-p_i}{1-r_i}$ (6)

$= O(p(R|Q)) \cdot \prod_{x_i=1, x_i \in Q}^V \frac{p_i}{r_i} \cdot \left(\frac{\prod_{x_i \in Q} \frac{1-p_i}{1-r_i}}{\prod_{x_i=1, x_i \notin Q} \frac{1-p_i}{1-r_i}} \right)$ (7)

$= O(p(R|Q)) \cdot \prod_{x_i=1, x_i \in Q}^V \frac{p_i(1-r_i)}{r_i(1-p_i)} \cdot \prod_{x_i \in Q}^V \frac{1-p_i}{1-r_i}$ (8)

\Rightarrow

$x_i = P(x_i=1|R, g)$ (Positive)

$r_i = P(x_i=1|NR, g)$

(Rest)

$x_i = 0$

$r_i = 1 - x_i$

ORQ

constant.

$\log(A/B) = \log A - \log B$

$$\log \prod_i \left(\frac{p_i}{1-p_i} \times \frac{1-r_i}{r_i} \right) \\ = \sum \log [O(d, r_i)] - \log [O(d, x_i)]$$

adj!

COMP6714 Odd to do the ranking

Page 7 of 11

$$P_i = P(x_i=1|R, g) = \frac{\# \text{occ}(x_i \text{ in Rel doc})}{\# \text{Rel docs}} = \frac{s}{S} \quad \left. \begin{array}{l} s = \frac{n-s}{N-s} \rightarrow 0 \\ S = \frac{n-s}{N-s} \rightarrow 1 \end{array} \right\} r_i = \frac{n-s}{N-s}$$

$$\frac{p(R, D, g)}{p(NR, D, g)} = \frac{p(R|D, g)}{p(NR|D, g)} = \frac{p(D|R, g) \cdot p(R, g)}{p(D|NR, g) \cdot p(NR, g)}$$

$$\prod_i p_i = p(x_i=1|R, g) \\ r_i = p(x_i=1|NR, g)$$

$$\frac{p(R|D, g)}{p(NR|D, g)} = \frac{p(D|R, g)}{p(D|NR, g)} = O(R, g) \cdot \prod_{x_i \in D} \frac{p(x_i=1|R, g)}{p(x_i=1|NR, g)}$$

$$= O(R, g) \cdot \prod_{x_i \in D} \frac{p(x_i=1|R, g)}{p(x_i=1|NR, g)} \cdot \prod_{x_i \in R \setminus D} \frac{p(x_i=0|R, g)}{p(x_i=0|NR, g)}$$

$$= O(R, g) \cdot \prod_{x_i \in D} \frac{p_i \cdot (1-p_i)}{r_i \cdot (1-p_i)} \quad \boxed{\prod_{x_i \in D} \frac{1-p_i}{1-r_i}}$$

J. Web Search Basics and Link Analysis

- Categorization of query types: **informational**, **navigational**, and **transactional**.

From MRS08 p.432

Informational queries seek general information on a broad topic, such as leukemia or Provence. There is typically not a single web page that contains all the information sought; indeed, users with informational queries typically try to assimilate information from multiple web pages.

Navigational queries seek the website or homepage of a single entity that the user has in mind, say Lufthansa airlines. In such cases, the user's expectation is that the very first search result should be the home page of Lufthansa. The user is not interested in a plethora of documents containing the term Lufthansa; for such a user, the best measure of user satisfaction is precision at 1.

A **transactional query** is one that is a prelude to the user performing a transaction on the Web – such as purchasing a product, downloading a file or making a reservation. In such cases, the search engine should return results listing services that provide form interfaces for such transactions.

```
Thx :)  
//no worries  
// lol
```

- The pagerank algorithm and its variant (topic-specific, personalized pageranks)

Exercise

Consider a Web graph with three nodes 1, 2, and 3. The links are as follows: 1->2, 3->2, 2->1, 2->3. Write down the transition probability matrices P for the surfer's walk with teleporting, with the value of teleport probability $\alpha=0.5$.

$$\begin{array}{c}
 \text{Web Graph: } \\
 \begin{array}{c}
 \text{1} \quad \text{2} \quad \text{3}
 \end{array}
 \begin{array}{ccc}
 & \text{1} \rightarrow \text{2} & \text{3} \rightarrow \text{2} \\
 & \downarrow & \downarrow \\
 & \text{2} \rightarrow \text{1} & \text{2} \rightarrow \text{3}
 \end{array}
 \end{array}
 \quad A = \begin{pmatrix} 0 & 1 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 1 & 0 \end{pmatrix}$$

$$(1-\alpha) \begin{pmatrix} 0 & 1 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 1 & 0 \end{pmatrix} + \alpha \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix} = P$$

If surfer starts at state 1, $x_0 = (1 \ 0 \ 0)$. If no given start state, $x_0 = (1/3 \ 1/3 \ 1/3)$

\vec{x}_0	1	0	0
\vec{x}_1	$1/6$	$2/3$	$1/6$
\vec{x}_2	$1/3$	$1/3$	$1/3$
\vec{x}_3	$1/4$	$1/2$	$1/4$
\vec{x}_4	$7/24$	$5/12$	$7/24$
...
\vec{x}	$5/18$	$4/9$	$5/18$

-> steady state probability distribution

// Thx for this clear example.

The main idea of page ranking of Notebook L18

