Unit1

Information Retrieval
Unit 1

Define :

Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

IR Architecture

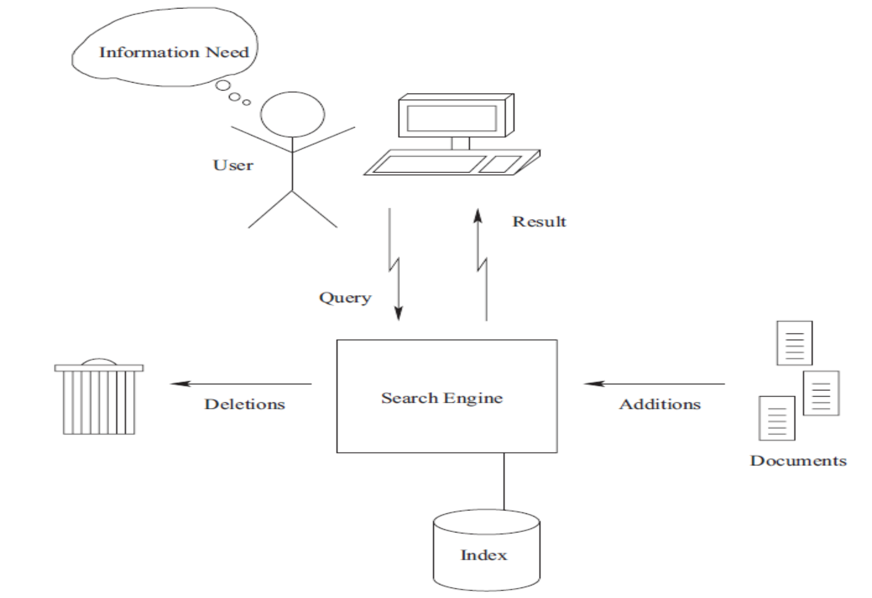


Figure 1.1 Components of an IR system.

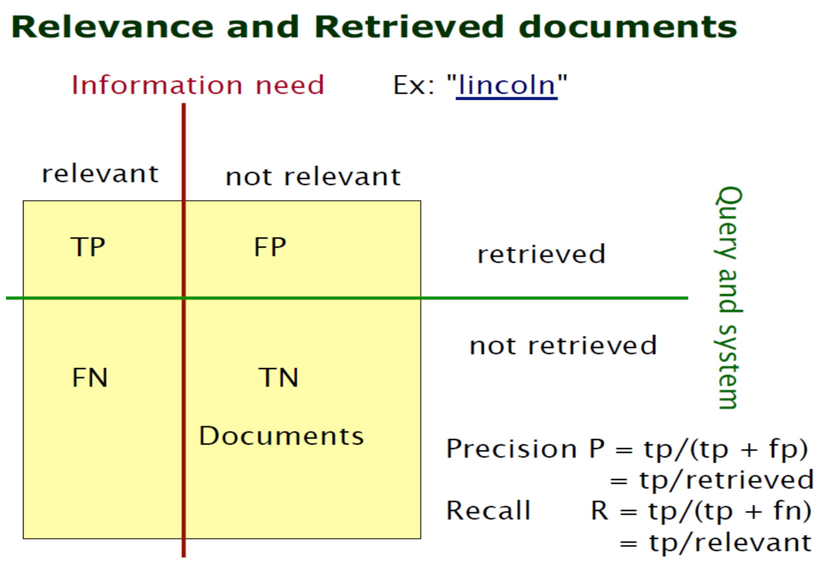
Issues with IR

- Uncertainty and Vagueness: Information representations may not perfectly reflect the true meaning of documents, especially for multimedia content like images or videos. Similarly, users may have trouble clearly expressing their information needs in a query.
- Precision vs. Recall: The trade-off between retrieving relevant documents (recall) and avoiding irrelevant ones (precision).
- System Evaluation Metrics: How to measure the effectiveness of an information retrieval system using metrics like precision, recall, F1 score, and user satisfaction surveys.
- Information Overload: The vast amount of information available can make it difficult for users to find what they need.
- Relevance Ranking: Ranking retrieved documents based on their relevance to the user's query, considering factors like keywords, document structure, and user preferences.
- Evolving Nature of Information: Information is constantly being created and updated, requiring IR systems to adapt and efficiently handle new data.

Precision vs Recall

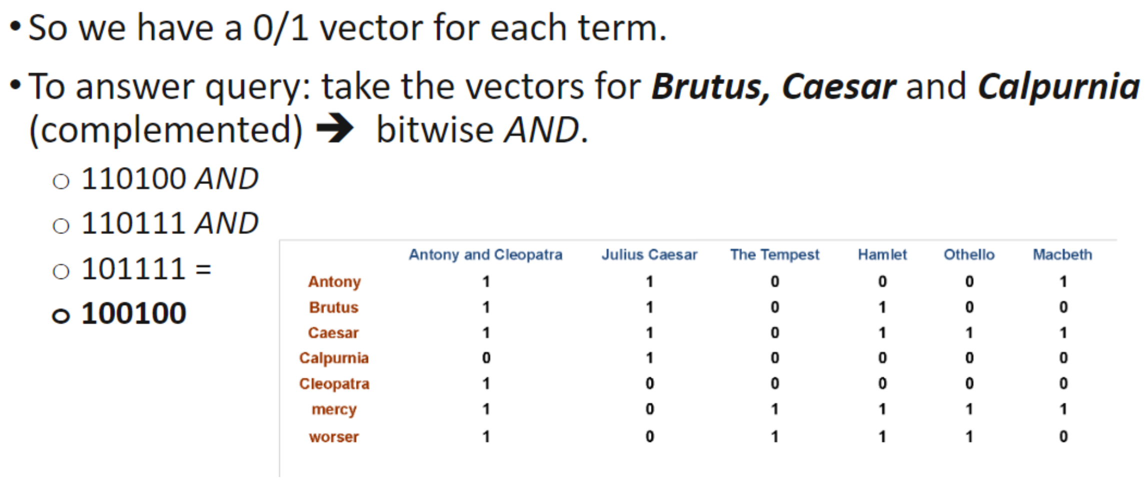
Precision : Fraction of retrieved docs that are relevant to the user's information need

Recall : Fraction of relevant docs in collection that are retrieved



Precision P = tp/(tp + fp)
= tp/retrieved
Recall R = tp/(tp + fn)
= tp/relevant

AD HOC RETRIEVAL:
System aims to provide documents from within the collection that are relevant to an arbitrary user information need, communicated to the system by means of a one-off, user-initiated query.
Incidence Matrix



- So we have a 0/1 vector for each term.
- To answer query: take the vectors for ***Brutus, Caesar*** and ***Calpurnia*** (complemented) ➔ bitwise *AND.*
  - 110100 *AND*
  - 110111 *AND*
  - 101111 =
  - **100100**

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

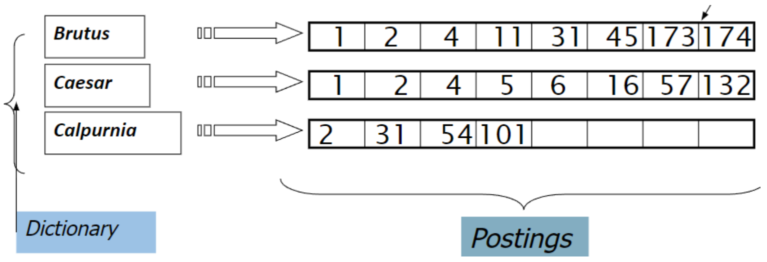
Why is incidence matrix not used ??
Bigger collections

- Consider N = 106 documents, each with about 1000 tokens
- ⇒ total of 109 tokens
- On average 6 bytes per token, including spaces and
- punctuation ⇒ size of document collection is about 6 · 109 = 6 GB
- Assume there are M = 500,000 distinct terms in the collection
- (Notice that we are making a term/token distinction.)
- M = 500,000 × 106 = half a trillion 0s and 1s.
- But the matrix has no more than one billion 1s.
- Matrix is extremely sparse.

Solution to above problem -

Inverted Index

For each term t, we store a list of all documents that containt.Identify each doc by a docID, a document serial number.



- We need variable-size postings lists On disk, a continuous run of postings is normal and best In memory, can use linked lists or variable length arrays
- Each item in the list – which records that a term appeared in a document– is conventionally called a **posting.**

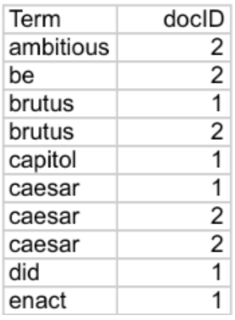
Tokenization and Normalisation

Main problem is that there are way too many ways to write the same thing Email == email == e-mail and other things like complex chemical names and huge floating point numbers
Solution to this is tokenization and normalising the content of the query to get relevant answers
Techniques:

1. **Case folding :**
    a. {The,tHe,THE,thE} all are same as the.
    b. Problem : here is if you search the term General Motors (Car company) its not same as general motors (information related to motors)
    c. Case folding will fail here as we want info related to Car company
2. **Stopword Removal :**
    a. They are terms that appear way to frequently and have huge term frequencies but don't add any value to the meaning of the query examples like - to,is,in,a,an,the etc
    b. There is a list of stopwords defined by many libraries that can removed before hand form queries and documents to avoid huge computations and space taken by stopwords
    c. Problem sometimes group of stopwords might be the whole query for example - to be or not to be is a famous dialogue but all words will be removed because of stopword removal
3. **Stemming :**
    a. Strips down words to their root form
    b. Example -{compressed,compression}->compress,{walking,walked,walks}->walk
    c. Author names and magazine catalogues are text that should not undergo stemming
    d. Main two for english - Koverts and porter

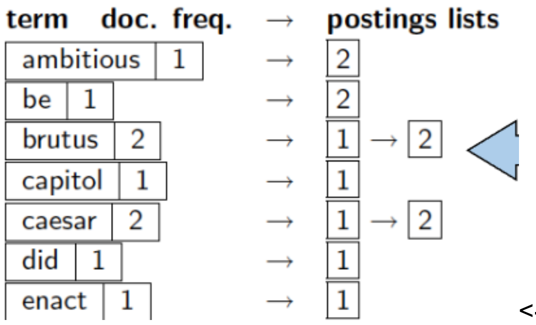
Steps to make indexes (can ask in exam)/ construct an inverted index or postings list

1. Take the terms in each document and arrange in two lists
2. Start merging both lists in alphabetical order with doc id next to it



Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1

3. Make posting lists out of it



term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1

<-output

Query processing

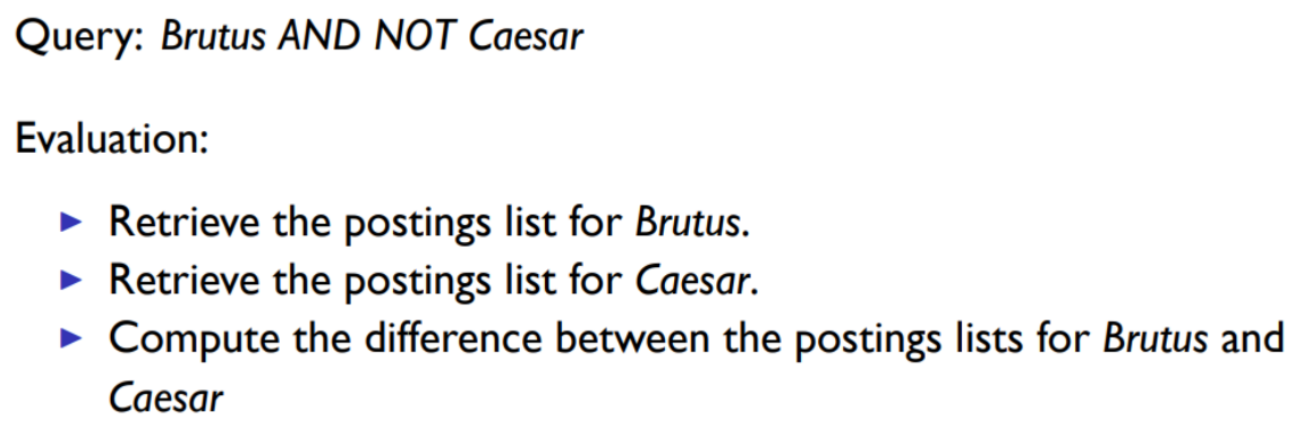
- **Brutus AND Caesar**
  - Locate Brutus in the Dictionary;
- Retrieve its postings.
  - Locate Caesar in the Dictionary;
- Retrieve its postings.
  - "Merge" the two postings

Intersect two posting lists

```
INTERSECT($p_1, p_2$)
answer ← ⟨ ⟩
while $p_1 \neq$ NIL and $p_2 \neq$ NIL
do if $docID(p_1) = docID(p_2)$
      then ADD(answer, $docID(p_1)$)
           $p_1 \leftarrow next(p_1)$
           $p_2 \leftarrow next(p_2)$
      else if $docID(p_1) < docID(p_2)$
             then $p_1 \leftarrow next(p_1)$
             else $p_2 \leftarrow next(p_2)$
return answer
```

Query: *Brutus OR Caesar*

- Retrieve the postings list for *Brutus.*
- Retrieve the postings list for *Caesar.*
- Compute the union of the postings lists.



Query: *Brutus AND NOT Caesar*

Evaluation:

- Retrieve the postings list for *Brutus*.
- Retrieve the postings list for *Caesar*.
- Compute the difference between the postings lists for *Brutus* and *Caesar*

Dictionary data structures

Two main choices:

- Hash Tables
  - Each vocabulary term is hashed to an integer
  - **Pros:**
  - Lookup is faster than for a tree: O(1)
  - **Cons:**
  - No easy way to find minor variants:
  - judgment/judgement
  - No prefix search   [tolerant retrieval]
  - If vocabulary keeps growing, need to occasionally do the expensive operation of rehashing everything
- Trees
  - Simplest: binary tree
  - More usual: B-trees
  - Trees require a standard ordering of characters and hence strings … but we typically have one
  - **Pros:**
  - Solves the prefix problem (terms starting with hyp)
  - **Cons:**
  - Slower: O(log M) [and this requires balanced tree]
  - Rebalancing binary trees is expensive
  - But B-trees mitigate the rebalancing problem

Wild card queries

WIld card Queries are used when :

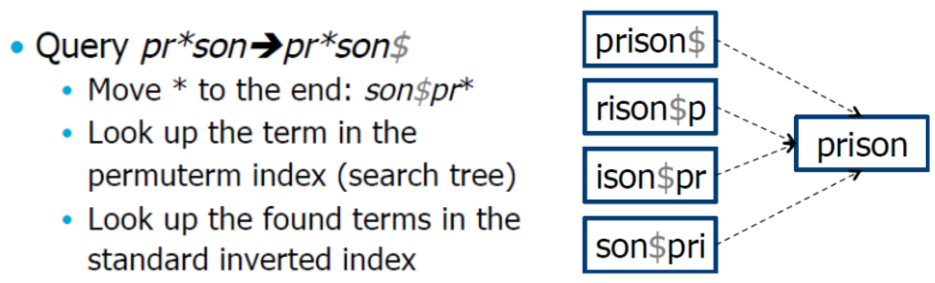
- There is uncertainty about the spelling of a term (Dantes vs. Dantès)
- Multiple spelling variants of a term exist (labour vs. labor)
- All terms with the same stem are sought (restoration and restore)

Types :

- **Trailing wildcard queries**, also known as suffix wildcard queries, are a type of search query that uses a wildcard symbol (*) at the end of a search term. This symbol tells the search engine to find all terms that begin with the letters you provided and have any combination of characters following them.

- **Leading Wildcard (*building):**
    - Looks for terms ending with what you provide (* = any characters).
    - Image uses a reverse B-tree for efficient searching (stores terms backwards).
    - Example: "*building" finds "building", "rebuilding" but not "rebuilt".
- **Single wildcard Wildcard (analy*ed):**
    - Traverse regular -Btree for (analy)
    - Traverse reverse B-tree for (ed)
    - Intersect the two results
- Remember:
    - Leading wildcards use reverse B-trees for efficiency.
    - Trailing wildcards work well with standard B-trees.

- How can we handle *'s in the middle of query term?(co*tion)
- We could look up co* AND *tion in a B-tree and intersect the two term sets
    - Expensive approach
- The solution: transform wild-card queries so that the *'s occur at the end
- This gives rise to the **Permuterm Index.**
- For term hello, index under:
- hello$, ello$h, llohe, lohel, o$hell, $hello
- where $ is a special symbol.



- Query *pr*son*➔*pr*son$*
    - Move * to the end: *son$pr**
    - Look up the term in the permuterm index (search tree)
    - Look up the found terms in the standard inverted index

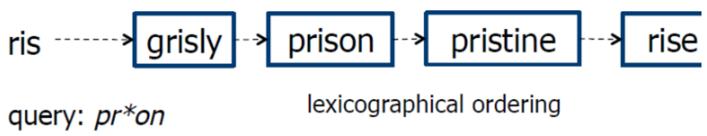
- Rotate query wild-card to the right
- Now use B-tree lookup as before.
- Permuterm problem: ≈ quadruples lexicon size

Bigram (k-gram)

- Enumerate all k-grams (sequence of k chars) occurring in any term
- e.g., from text “April is the cruellest month” we get the 2-grams (bigrams)
- a,ap,pr,ri,il,l,i,is,s,t,th,he,e,$c,cr,ru,
- ue,el,le,es,st,t$, m,mo,on,nt,h
- $ is a special word boundary symbol
- Maintain a second inverted index from bigrams to dictionary terms that match each bigram.
- Query mon* can now be run as
- $m AND mo AND on

- Gets terms that match AND version of our wildcard query.
- But we’d enumerate moon.
- Must post-filter these terms against query.
- Surviving enumerated terms are then looked up in the term-document inverted index.
- Fast, space efficient (compared to permuterm).

- each N-gram in the dictionary points to all terms containing the N-gram



- Wildcard query: *pr*on*
  - Boolean query *$pr AND on$*
  - Look up in a 3-gram index yields a list of matching terms
  - Look up the matching terms in a standard inverted index

- Wildcard query: red*
  - Boolean query *$re AND red* (also retrieves *retired*)
  - Post-filtering step to ensure enumerated terms match red*

Spell correction

- Two principal uses
  - Correcting document(s) being indexed
  - Correcting user queries to retrieve “right” answers
- Two main flavours:
- **Isolated word**
  - Check each word on its own for misspelling
  - Will not catch typos resulting in correctly spelled words
  - e.g., from → form
- **Context-sensitive**
  - Look at surrounding words,
  - e.g., I flew form Heathrow to Narita.

Document based

- Goal: the dictionary contains fewer misspellings
- But often we don’t change the documents and instead fix the query-document mapping

Query misspelling

- We can either
- Retrieve documents indexed by the correct spelling, OR
- Return several suggested alternative queries with the correct spelling
  - Did you mean … ?

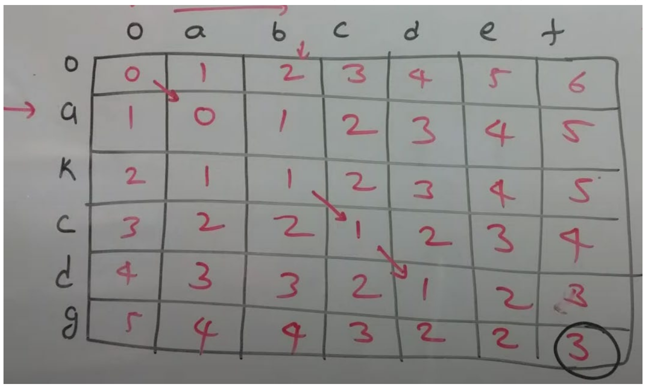
Isolated word correction

- Fundamental premise – there is a lexicon from which the correct spellings come
- Two basic choices for this
- A standard lexicon such as
  - Webster’s English Dictionary
  - An “industry-specific” lexicon – hand-maintained
- The lexicon of the indexed corpus
  - E.g., all words on the web
  - All names, acronyms etc.

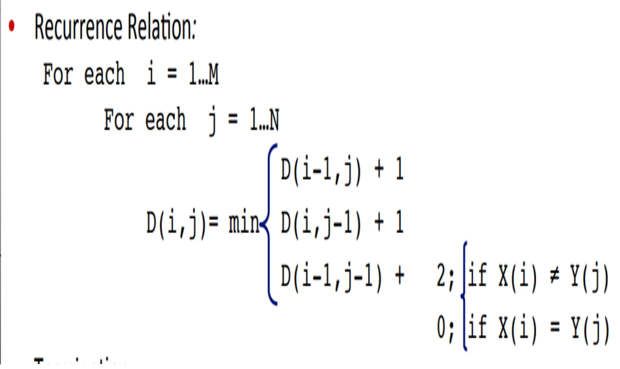
    - (Including the mis-spellings)
- Given a lexicon and a character sequence Q, return the words in the lexicon closest to Q

Edit distance

- Given two strings S1 and S2, the minimum number of operations to convert one to the other
- Operations are typically character-level
    - Insert, Delete, Replace, (Transposition)
- E.g., the edit distance from dof to dog is 1
- From cat to act is 2   (Just 1 with transpose.)
- from cat to dog is 3.
- Generally found by dynamic programming.



	0	a	b	c	d	e	f
0	0	1	2	3	4	5	6
a	1	0	1	2	3	4	5
k	2	1	1	2	3	4	5
c	3	2	2	1	2	3	4
d	4	3	3	2	1	2	3
g	5	4	4	3	2	2	3



- Recurrence Relation:

```
For each  i = 1…M
    For each  j = 1…N
```

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} \end{cases}$$

Steps (in recurrence relation its +1 not 2)

- Start by adding 0 and 0 to both words and then make a matrix
- 1st row and column just write 0,1,2,3,... till len
- For other elements the rules are :
    - Check the two things intersecting at the point for example at (1,1) two a's are matching so then take the exact value diagonal to them (i-1,j-1)here ==0
    - Else if intersecting values are not matching can take the min of upper,upper-left , left value and then add one to it foe example - (1,2)its b and a so options are (1,0,2)so take 0 and add 1
    - Do this and check the bottom right value that is the edit distance for the same

Weighted edit distance

- As above, but the weight of an operation depends on the character(s) involved
- Meant to capture OCR or keyboard errors
    - Example: m more likely to be mis-typed as n than as q
    - Therefore, replacing m by n is a smaller edit distance than by q
- This may be formulated as a probability model
- Requires weight matrix as input
- Modify dynamic programming to handle weights

n-gram overlap

- Enumerate all the n-grams in the query string as well as in the lexicon
- Use the n-gram index (recall wild-card search) to retrieve all lexicon terms matching any of the query n-grams
- Threshold by number of matching n-grams
    - Variants – weight by keyboard layout, etc.

- Suppose the text is november
    - Trigrams are nov, ove, vem, emb, mbe, ber.
    - The query is december
    - Trigrams are dec, ece, cem, emb, mbe, ber.
    - So 3 trigrams overlap (of 6 in each term)
- **Jaccard coefficient**

- A commonly-used measure of overlap
- Let *X* and *Y* be two sets; then the J.C. is

$$|X \cap Y| / |X \cup Y|$$

- Equals 1 when *X* and *Y* have the same elements and zero when they are disjoint
- *X* and *Y* don't have to be of the same size
- Always assigns a number between 0 and 1
    - Now threshold to decide if you have a match
    - E.g., if J.C. > 0.8, declare a match

Context-sensitive spell correction

- Text: I flew from Heathrow to Narita.
- Consider the phrase query "flew form Heathrow"
- We'd like to respond
    - Did you mean "flew from Heathrow"?
    - because no docs matched the query phrase.
- Need surrounding context to catch this.
- First idea: retrieve dictionary terms close (in weighted edit distance) to each query term
- Now try all possible resulting phrases with one word "fixed" at a time
    - flew from heathrow
    - fled form heathrow
    - flea form heathrow
- **Hit-based spelling correction:** Suggest the alternative that has lots of hits.
- In the above example flew from heathrow will have more text documents and results so the return those instead

Phonetic Correction

Soundex

- Turn every token to be indexed into a 4-character reduced form
- Do the same with query terms
- Build and search an index on the reduced forms

1. Retain the first letter of the word.
2. Change all occurrences of the following letters to '0' (zero):
   'A', E', 'I', 'O', 'U', 'H', 'W', 'Y'.
3. Change letters to digits as follows:

- B, F, P, V → 1
- C, G, J, K, Q, S, X, Z → 2
- D,T → 3
- L → 4
- M, N → 5
- R → 6

- Remove all pairs of consecutive digits.

- Remove all zeros from the resulting string.
- Pad the resulting string with trailing zeros and return the first four positions, which will be of the form <uppercase letter> <digit> <digit> <digit>.
- E.g., Herman becomes H655.

Feature	Normalisation	Stemming
Goal	Reduce words to their base or canonical form	Reduce words to their root form
Preserves Meaning	Aims to preserve the original meaning	May alter the meaning of the word
Technique	Removes prefixes, suffixes, and converts to lowercase	Chops off suffixes based on rules
Accuracy	High accuracy, often rule-based	Less accurate, may create incorrect words
Complexity	Simpler process	More complex rules required
Example	"jumps" -> "jump" (both noun and verb)	"running" -> "run" (changes verb tense)
Use Case	Text cleaning, duplicate detection	Information retrieval, topic modeling

Stemming vs. Lemmatization

Feature	Stemming	Lemmatization
Goal	Reduce a word to its base or root form	Reduce a word to its dictionary or canonical form (lemma)
Process	Chops off suffixes (sometimes prefixes) based on predefined rules	Uses morphological analysis to identify the base word and its part of speech
Output	May not be a real word	Always a real word found in a dictionary
Accuracy	Less accurate, can create unrecognizable words	More accurate, preserves meaning
Complexity	Simpler and faster	More complex and slower
Use Cases	Initial information retrieval, data mining (when speed is crucial)	Tasks requiring higher accuracy like text summarization, sentiment analysis

Skip pointers

More skips → shorter skip spans ⇒ more likely to skip. But lots of comparisons to skip pointers.
Fewer skips → few pointer comparison, but then long skip spans ⇒ few successful skips

- Simple heuristic: for postings of length L, use √L evenly-spaced skip pointers   [Moffat and Zobel 1996]

- Easy if the index is relatively static; harder if L keeps changing because of updates.
- This definitely used to help; with modern hardware it may not unless you're memory-based [Bahle et al. 2002]
- The I/O cost of loading a bigger postings list can outweigh the gains from quicker in memory merging!

Phase Queries

- We want to answer a query such as [stanford university] – as a phrase.
- Thus The inventor Stanford Ovshinsky never went to university should not be a match.
- The concept of phrase query has proven easily understood by users.
- About 10% of web queries are phrase queries.
- Consequence for inverted index: it no longer suffices to store docIDs in postings lists for terms.
- Two ways of extending the inverted index:
    - biword index
    - positional index

Biword index

- Index every consecutive pair of terms in the text as a phrase.
- For example, Friends, Romans, Countrymen would generate two biwords: "friends romans" and "romans countrymen"
- Each of these biwords is now a vocabulary term.
- Two-word phrases can now easily be answered.
- A long phrase like "stanford university palo alto" can be represented as the Boolean query "STANFORD UNIVERSITY" AND "UNIVERSITY PALO" AND "PALO ALTO".
- **Extended Biwords**
- Identifying Extended Biwords: The system searches for a specific pattern within each document: "NX*N". This means:
- N: The first and last terms must be nouns (N).
- X:* Any number (zero or more) of words from the "X" bucket (articles/prepositions) can be present in between the nouns.
- Biword Creation: If the pattern "NX*N" is found, the entire sequence is treated as a single unit called an "extended biword".
- Vocabulary Inclusion: These extended biwords are then added to the overall vocabulary used for indexing and retrieval purposes.
- Query Processing: When a user searches for terms, the system considers both individual words and extended biwords for matching.
- **Issues with biwords:**
    - False positives, as noted above
    - Index blowup due to very large term vocabulary

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions.

Proximity search

- For example: employment /4 place

- Find all documents that contain EMPLOYMENT and PLACE within 4 words of each other.
- Employment agencies that place healthcare workers are seeing growth is a hit.
- Employment agencies that have learned to adapt now place healthcare workers is not a hit.
- Use the positional index
- Simplest algorithm: look at cross-product of positions of (i) EMPLOYMENT in document and (ii) PLACE in document
- Very inefficient for frequent words, especially stop words
- Note that we want to return the actual matching positions, not just a list of documents.

Combination scheme

- Biword indexes and positional indexes can be profitably combined.
- Many biwords are extremely frequent: Michael Jackson etc
- For these biwords, increased speed compared to positional postings intersection is substantial.
- Combination scheme: Include frequent biwords as vocabulary terms in the index. Do all other phrases by positional intersection.
- Williams et al. (2004) evaluate a more sophisticated mixed indexing scheme – Next Word Index. Faster than a positional index, at a cost of 26% more space for index.

Unit 2

Index Construction

It is the process of creating an organised data structure that enables fast and efficient retrieval of relevant documents from a large collection of text or other data.
P.s its mainly efficient algorithms used to construct indexes

Symbol	statistic	value
N	documents	800,000
L	avg. # tokens per doc	200
M	terms (= word types)	400,000

Sort-based index construction

- As we build the index, we parse docs one at a time.
- The final postings for any term are incomplete until the end.
- At 8 bytes per (termID, docID), demands a lot of space for large collections.
- T = 100,000,000 in the case of RCV1

Hardware basics

- Access to data in memory is much faster than access to data on disk.
- Disk seeks: No data is transferred from disk while the disk head is being positioned.
- Therefore: Transferring one large chunk of data from disk to memory is faster than transferring many small chunks.
- Disk I/O is block-based: Reading and writing of entire blocks (as opposed to smaller chunks).
- Block sizes: 8KB to 256 KB.

BSBI: Blocked sort-based Indexing

- 8-byte records (termID, docID)
- These are generated as we parse docs
- Must now sort 100M such 8-byte records by termID
- Define a Block ~ 10M such records
- Can easily fit a couple into memory

- Will have 10 such blocks to start with
- Basic idea of algorithm:
    - Accumulate postings for each block, sort, write to disk
    - Then merge the blocks into one long sorted order

Sorting 10 blocks of 10M records

- First, read each block and sort within:
- Quicksort takes O(N ln N) expected steps
- In our case N=10M
    - 10 times this estimate – gives us 10 sorted runs of 10M records each.

How to merge the sorted runs?

- it is more efficient to do a multi-way merge, where you are reading from all blocks simultaneously
    - Open all block files simultaneously and maintain a read buffer for each one and a write buffer for the output file
    - In each iteration, pick the lowest termID that hasn't been processed using a priority queue
    - Merge all postings lists for that termID and write it out
- Providing you read decent-sized chunks of each block into memory and then write out a decent-sized output chunk, then you're not killed by disk seeks

Problems :

- Our assumption was: we can keep the dictionary in memory.
- We need the dictionary (which grows dynamically) in order to implement a term to termID mapping.

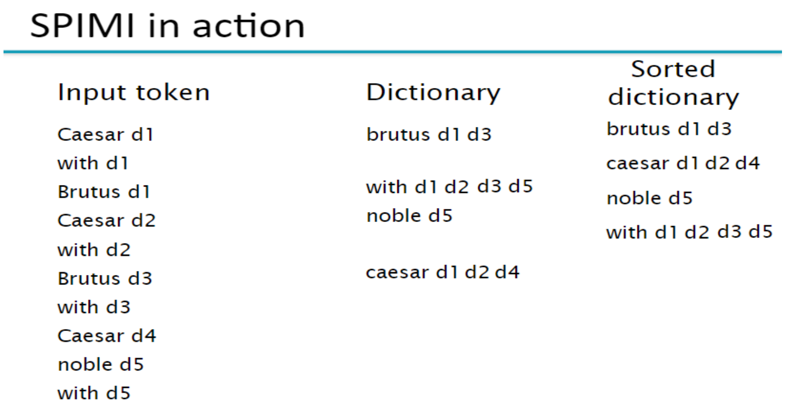
SPIMI: Single-pass in-memory indexing

- The core idea of SPIMI is to build the inverted index incrementally, one document at a time.
- As it processes tokens from each document, it keeps track of terms and their corresponding document IDs in the postings lists.
- This avoids the need for multiple passes over the data and sorting terms like other indexing methods.
- Benefits of SPIMI:
    - Single-pass processing makes it efficient for large document collections.
    - In-memory processing allows for faster manipulation of data structures.
- Challenges:
    - Requires sufficient memory to hold the entire in-memory data structures during indexing.
    - Might not be suitable for very large collections that cannot fit in memory.

Steps of Spimi

- output_file = new_file(): Creates a new file to store the inverted index.
- dictionary = new_hash(): Initialises a hash table to store dictionary terms and their corresponding postings lists.
- while (free_memory_available): The loop continues as long as there's enough memory to process tokens.
- token = next(token_stream): Reads the next token from the document stream.
- if term(token) not in dictionary:
    - postings_list = add_to_dictionary(dictionary, term(token)):
    - If the term isn't in the dictionary, it's added along with a new postings list.

- Else:
  - postings_list = get_postings_list(dictionary, term(token)):
  - If the term already exists, retrieves the corresponding postings list.
- add_to_posting_list(postings_list, doc_id(token)):
  - The document ID associated with the current token is added to the postings list for that term.
- After processing all tokens, the in-memory dictionary needs to be written to the output file to create the final inverted index.

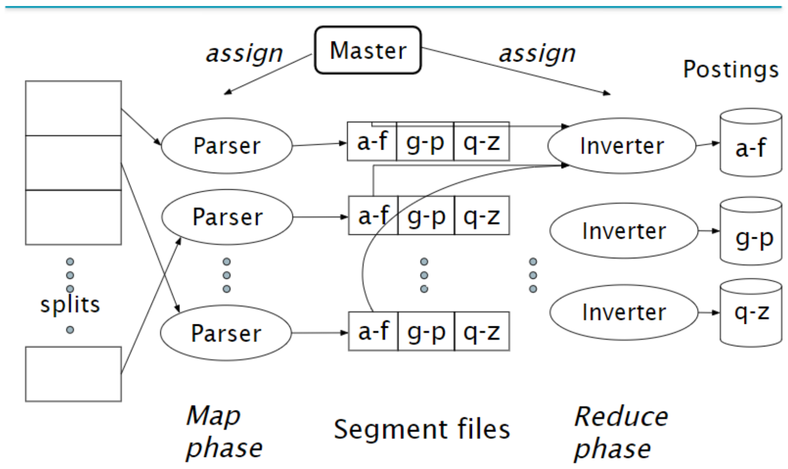


SPIMI in action

Input token	Dictionary	Sorted dictionary
Caesar d1	brutus d1 d3	brutus d1 d3
with d1		caesar d1 d2 d4
Brutus d1	with d1 d2 d3 d5	noble d5
Caesar d2	noble d5	with d1 d2 d3 d5
with d2		
Brutus d3	caesar d1 d2 d4	
with d3		
Caesar d4		
noble d5		
with d5		

Distributed Indexing

- Maintain a master machine directing the indexing job – considered “safe”.
- Break up indexing into sets of (parallel) tasks.
- Master machine assigns each task to an idle machine from a pool.
- We will use two sets of parallel tasks
  - Parsers
    - Master assigns a split to an idle parser machine
    - Parser reads a document at a time and emits (term, doc) pairs
    - Parser writes pairs into j partitions
    - Example: Each partition is for a range of terms’ first letters
      - (e.g., a-f, g-p, q-z) – here j = 3.
  - Inverters
    - An inverter collects all (term,doc) pairs (= postings) for one term-partition.
    - Sorts and writes to postings lists
- Break the input document collection into splits
- Each split is a subset of documents (corresponding to blocks in BSBI/SPIMI)



Dynamic indexing

- Up to now, we have assumed that collections are static.(They aren't)
- Documents come in over time and need to be inserted.
- Documents are deleted and modified.
- This means that the dictionary and postings lists have to be modified:
- Postings updates for terms already in dictionary
- New terms added to dictionary

Logarithmic merge(read frm ppt)

Index Compression

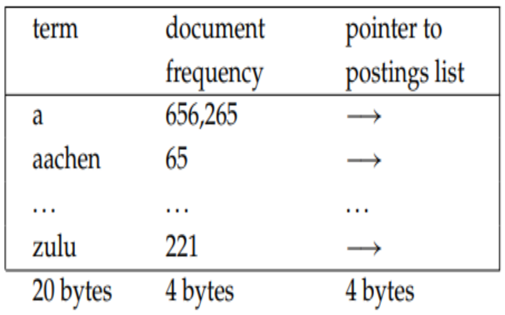
Index compression is a technique that reduces the size of data indexes without sacrificing performance. It's like packing a suitcase efficiently to fit more clothes in the same space.
Advantages:

- Space savings: Compressed indexes take up less storage, lowering costs and improving backup times.
- Faster searches: Smaller indexes often load and process queries quicker, leading to faster search results.
- Reduced network traffic: Less data needs to be transferred across networks, improving performance.

Two main Parts : Dictionary Compression And Posting Compression

Dictionary Compression

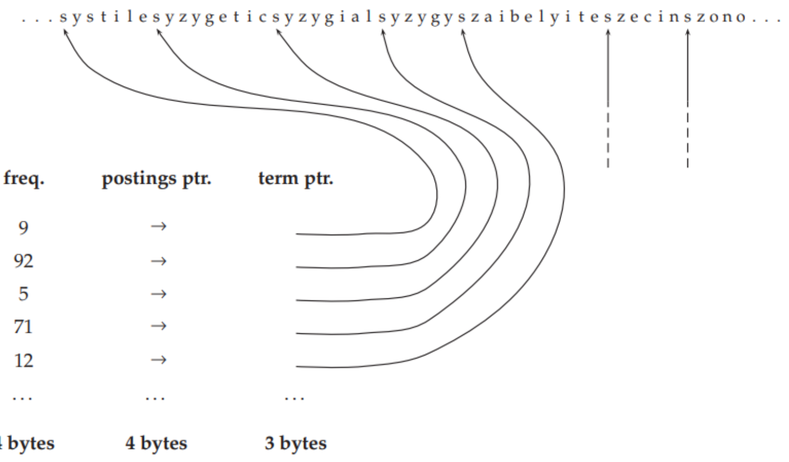
1. Dictionary as a sorted Array
   a. It means arranging all distinct terms (words or phrases) in ascending alphabetical order within a continuous memory block.
   b. This structure allows for efficient binary search to locate terms quickly.
   c. Disadvantages -
      i. Space inefficiency for varying term lengths
      ii. Limited compression opportunities



term	document frequency	pointer to postings list
a	656,265	⟶
aachen	65	⟶
…	…	…
zulu	221	⟶
20 bytes	4 bytes	4 bytes

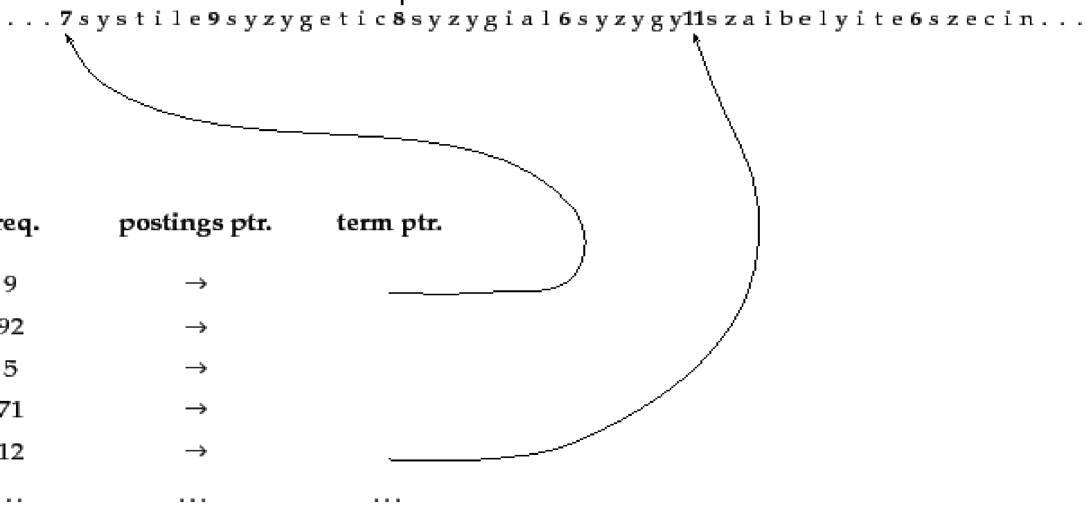
Here the term ‘a’ takes as much space as term ‘aachen ’which both take 20 bytes as a whole so its not good for storing data efficiently

2. Dictionary as a String
    a. Here instead of giving a total 20 byte storage per element we store them in a long continue=os string with pointers to each new word



. . . s y s t i l e s y z y g e t i c s y z y g i a l s y z y g y s z a i b e l y i t e s z e c i n s z o n o . . .
freq.
postings ptr.
term ptr.
9
92
5
71
12
. . .
. . .
. . .
4 bytes
4 bytes
3 bytes

3. Blocked Storage
    a. Here we further get down the size by storing blocks of words to one pointer
        i. K = block size
        ii. Here k = 4 terms
    b. Avoids K -1 term pointers



. . . 7 s y s t i l e 9 s y z y g e t i c 8 s y z y g i a l 6 s y z y g y 11 s z a i b e l y i t e 6 s z e c i n . . .
freq.
postings ptr.
term ptr.
9
92
5
71
12
. . .
. . .
. . .

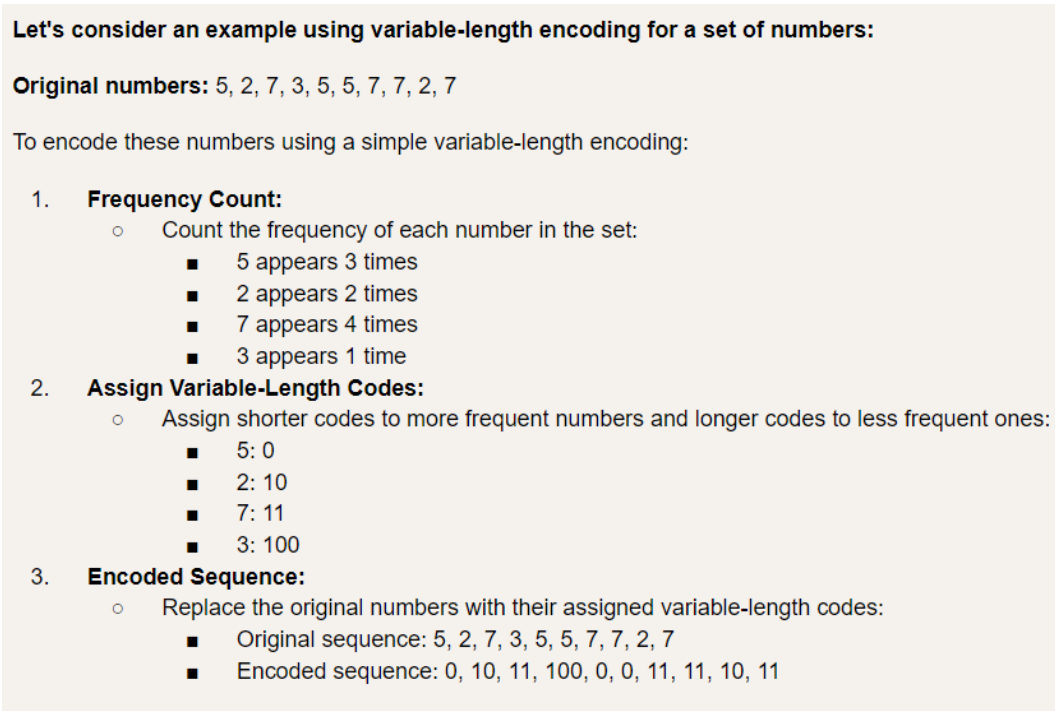
4. Font Coding
    a. Here we further cut down common text in a block cause its alphabetically sorted
    b. One block in blocked compression (k = 4) . . .

   c. 8 automata 8 automate 9 automatic 10 automation
      i. Can further be compressed with font coding :
      ii. 8 automat ∗ a 1 ⋄ e 2 ⋄ i c 3 ⋄ ion

Posting Compression

It specifically focuses on compressing the "postings lists" in inverted indexes.
The primary goal of posting compression is to reduce the storage space required for the postings lists while ensuring quick access to relevant documents during search operations.

1. Gap Coding
   a. Store the differences (gaps) between successive docIDs rather than storing the entire IDs.
   b. This method is effective when docIDs are sorted, reducing the numbers' magnitude and hence the space needed to represent them.
   c. For example if storing doc ids like 10000,10500,10510 we can store it as 10000,500,10
   d. The gaps between two numbers will consume less space to store then the numbers themselves
2. Variable Length Encoding
   a. Variable-length encoding (VLE) is a technique used in data compression where different symbols (such as numbers or characters) are represented using varying numbers of bits.
   b. Symbols that occur more frequently are encoded with shorter bit sequences, while less common symbols are encoded with longer bit sequences.


Let's consider an example using variable-length encoding for a set of numbers:

Original numbers: 5, 2, 7, 3, 5, 5, 7, 7, 2, 7

To encode these numbers using a simple variable-length encoding:

1. **Frequency Count:**
   - Count the frequency of each number in the set:
     - 5 appears 3 times
     - 2 appears 2 times
     - 7 appears 4 times
     - 3 appears 1 time
2. **Assign Variable-Length Codes:**
   - Assign shorter codes to more frequent numbers and longer codes to less frequent ones:
     - 5: 0
     - 2: 10
     - 7: 11
     - 3: 100
3. **Encoded Sequence:**
   - Replace the original numbers with their assigned variable-length codes:
     - Original sequence: 5, 2, 7, 3, 5, 5, 7, 7, 2, 7
     - Encoded sequence: 0, 10, 11, 100, 0, 0, 11, 11, 10, 11

Personalised Search

Feature	Traditional Search	Personalized Search
Results	Uniform for all users	Tailored to user preferences and interests
User Experience	Generic, one-size-fits-all	Relevant, efficient, and satisfying
Search History	Not considered	Analyzed to refine results
User Location	Not factored in	Used to suggest local options
Demographics	Not relevant	May influence certain results (e.g., news or music)
Overall Focus	Content matching keywords	User needs and intentions

- Two main ways:
  - Query expansion
    - Modify or augment user query
      E.g., query term "IR" can be augmented with either "information retrieval" or "Ingersoll-Rand" depending on user interest
    - Ensures that there are enough personalised results
  - Reranking
    - Issue same query, fetch same results
    - Rerank on basis of user profile
    - Allows personalised + globally relevant results
- Search engines do:
  - USER PROFILING:
    - Create a profile based on user interests
    - Tracks:
      - History
      - Explicit prefs: set preferences
      - Implicit: analysed from engagement
      - Contextual data: location, time, device
  - Considerations:
    - Transparency: Users should be informed about data collection and profiling practices.
    - Control: Users should have the right to access, review, and modify their profile data.
    - Privacy: Data should be collected and used responsibly, minimising risks of misuse or discrimination.

RANKING IN P.S.

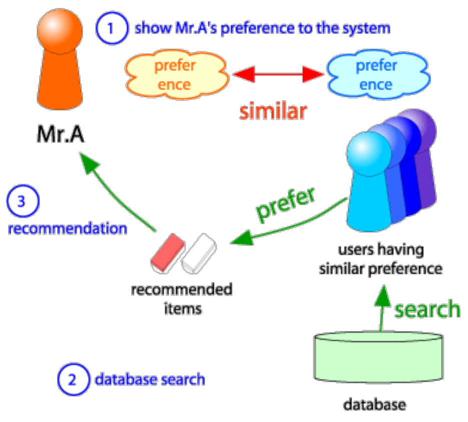
- Individualised ranking to users
- 5 categories:
  - Personal:
    - Info about user
    - Signal: language, demographics
  - Social:
    - Info about users social network
    - Signals: Social network connections
  - Activity:

        - Info about queries by user + OTHER users
    - Signals:
        - Query suggestion: if user enters a + b, b is a good suggestion after a
        - Spelling Correction: if b immediately after a, b might be correct spelling of a
    - Context:
        - Info about context in which search is done
        - Signals: Location, device, date and time
    - Learning how to rank:
        - Learning optimal combination of all
        - Goal: do this continuously,automatically using ML
        - Predict: for each query-result pair -> is result relevant to user needs
        - Detect common patterns -> specific outcomes

RECOMMENDER SYSTEMS

- If user a does something (x) , then does something (y)
- Now, if user b does (x), recommend (y)
- Eg amazon. Spotify etc
- Motivation:
    - Automates suggestion
    - Extends user knowledge field
    - Explore new things
    - Find more things fast
    - Ecomm -> improve sales
    - Also opp: remove movies and etc which will def not be enjoyed

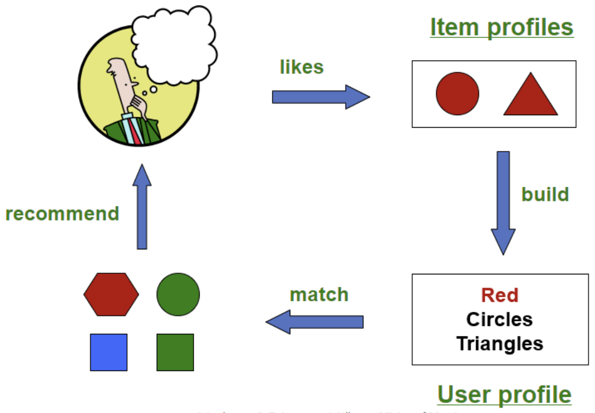
TYPES OF RECOMMENDER SYSTEMS

- Collaborative / social filtering
    - Aggregation of consumers prefs AND
    - Recommendations to others based on
        - Similarity in behavioural patterns
    - Consider user X
        - Find set n of other users whose "Ratings" are similar to x's "ratings"
        - Estimate x's ratings based on ratings of set n users
    - 

    - Problem:
        - Predict how well user will like, if no rating is available

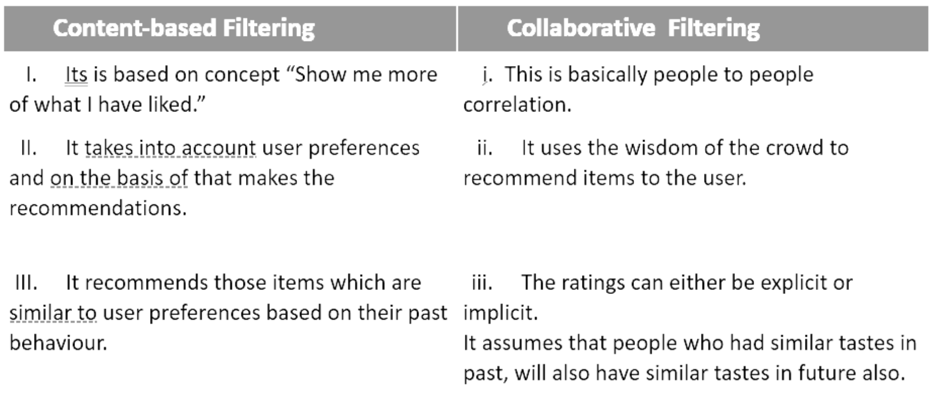
- TWO TYPES:
    - User based cf:
        - Recommend items by finding users sim to *active users*
    - Item based cf:
        - For item i, find sim items
        - Estimate rating by rating of sim items
- Pros:
    - Works for any kind of item
- Cons:
    - Cold start: need enough users
    - Sparsity: user rating is sparse
    - First rate: cannot recommend item which has never been rated: new items, eccentric items
    - Popularity bias: can't recommend to someone with a unique taste

- Content Based Filtering
    - Supervised ML used to induce a classifier to discriminate between interesting vs not for user
    - Main idea:
        - Recommend items to X similar to previous items rated highly by x



J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive

    - PROS:
        - No need for data on other users
        - Able to recommend to user w unique taste
        - Recommend new and unpopular items
        - Provide explanations - list content features
    - CONS:
        - Finding appt features is hard
        - How to build profiles for new users
        - Overspecialization:
            - Same same things
            - Might have multiple interests
            - Unable to exploit quality judgement of other users



Content-based Filtering	Collaborative Filtering
I. Its is based on concept “Show me more of what I have liked.”	i. This is basically people to people correlation.
II. It takes into account user preferences and on the basis of that makes the recommendations.	ii. It uses the wisdom of the crowd to recommend items to the user.
III. It recommends those items which are similar to user preferences based on their past behaviour.	iii. The ratings can either be explicit or implicit. It assumes that people who had similar tastes in past, will also have similar tastes in future also.

SNIPPET GENERATION

- "Snippet generation" in the context of Information Retrieval (IR) typically refers to the process of creating concise and meaningful text excerpts, or snippets, from longer documents in response to a user query.
- Query dependent doc summary
- Simple approach
  - Rank each sentence in doc -> use significance factor
  - Select top sentences for summary
  - Proposed by Luhn in the 50s
- Length control - concise but enough info
- Sentence selection:
  - Significance factor for a sentence is calculated based on the occurrence of significant words
  - If fd,w is the frequency of word w in document d, then w is a significant word if it is not a stopword and

$$f_{d,w} \geq \begin{cases} 7 - 0.1 \times (25 - s_d), & \text{if } s_d < 25 \\ 7, & \text{if } 25 \leq s_d \leq 40 \\ 7 + 0.1 \times (s_d - 40), & \text{otherwise} \end{cases}$$

where sd is the number of sentences in document d
text is bracketed by significant words (limit on number of non-significant words in bracket)

- Involves more than just significance factor
- e.g. for a news story, could use
  - whether the sentence is a heading
  - whether it is the first or second line of the document
  - the total number of query terms occurring in the sentence
  - the number of unique query terms in the sentence
  - the longest contiguous run of query words in the sentence
  - a density measure of query words (significance factor)
- Weighted combination of features used to rank sentences
- Web pages are less structured than news stories
  - can be difficult to find good summary sentences
- Snippet sentences are often selected from other sources
  - metadata associated with the web page

        - e.g., <meta name="description" content= ...>
    - external sources such as web directories
        - e.g., Open Directory Project, http://www.dmoz.org
- Snippets can be generated from text of pages like Wikipedia
- GUIDELINES:
    - All query terms should appear, showing reln to retrieved doc
    - When terms in title, need not be repeated
    - Highlight query terms
    - Should be readable, not lists of keywords

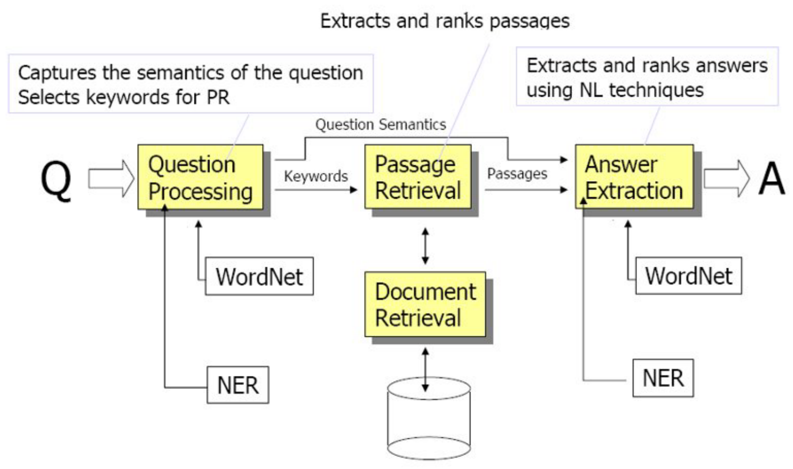
SUMMARIZATION

- A summary is a text that is produced from one or more texts, that contains a significant portion of the information in the original text(s), and that is no longer than half of the original text(s).

- Can be
    - Extractive
        - Created by reusing portions of text verbatim
        - Eg: search engines typically generate extractive summaries from webpages.
        - Most summarization is of this form
    - Abstractive
        - Info from src is rephrased
        - Humans generally write abstractive summaries
        - Like, paraphrasing
        - Semantic rep, inference and natural lang gen are harder

QUESTION ANSWERING

- Type of information retrieval.
- Given a collection of documents ,the system should be able to retrieve answers to questions posed in natural language.
- Generic q.a arch:



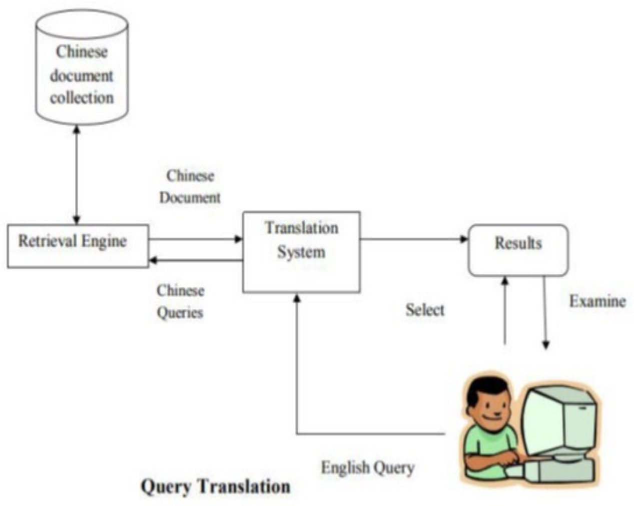
-
- NER is a natural language processing (NLP) task that involves identifying and classifying entities, such as names of people, organizations, locations, dates, numerical values, and other specific terms, within a text.

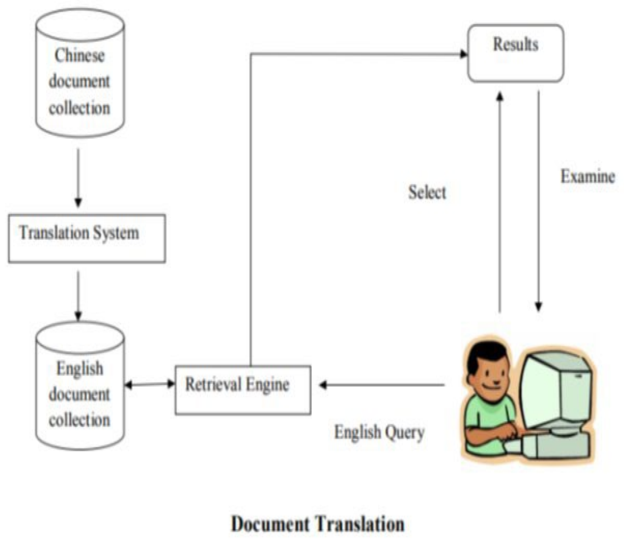
- WordNet is a lexical database of the English language that groups words into sets of synonyms, called synsets, and provides definitions, usage examples, and relationships between words.
- QA SYS:
  - Pull ans from unstructured collection of NL docs
  - Attempts to deal w wide range of ques:
    - Fact list defn, how why etc

CROSS LINGUAL SEARCH

- Cross-lingual search in Information Retrieval (IR) refers to the process of retrieving information written in a language different from the language used in the search query.
- TWO METHODS:
  - Query Translation:
    - Translate query into lang of doc
    - Search doc collection
    - Translate retrieved result into query lang
    - Translation Methods:
      - Various methods can be used for translation:
      - Statistical Machine Translation (SMT): Utilises statistical models trained on bilingual corpora to generate translations.
      - Neural Machine Translation (NMT): Employs neural networks for more context-aware translations.
      - Rule-Based Translation: Uses predefined rules for translation based on linguistic structures and patterns.
      - Hybrid Approaches: Combines multiple techniques for improved accuracy.

  - Document Translation:
    - Convert all docs into query lang
    - Then search
    - Convert back

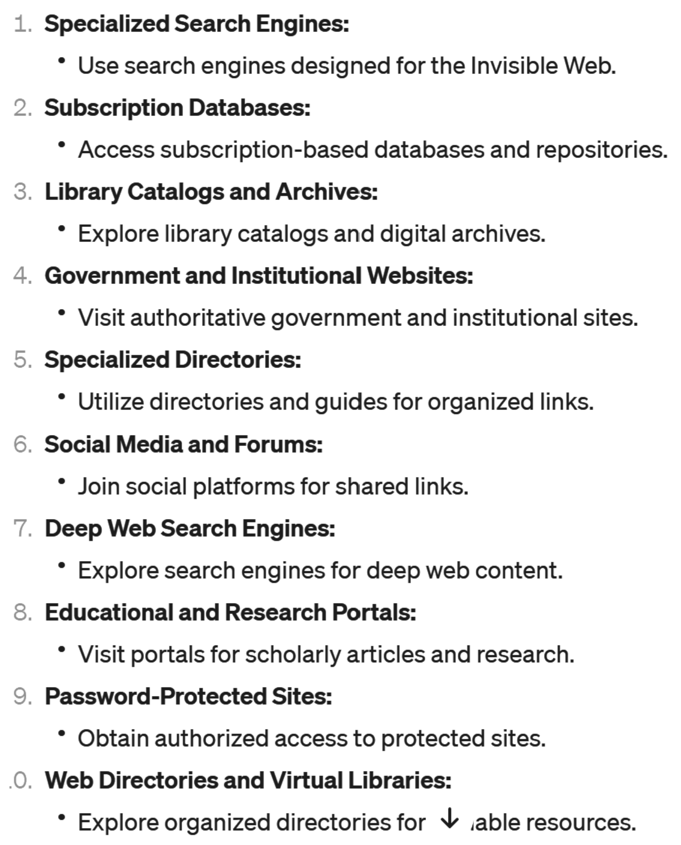




HIDDEN WEB / INVISIBLE WEB

- Part of the www, which is not indexable therefore invisible
- Opp of surface web
- It has data and info that cannot be searched w s.e for various reasons
- Users cant access by traditional s.e
- Non-indexed websites, apps, and resources include protected information in the areas of email, online banking, specialised databases, and other paid services

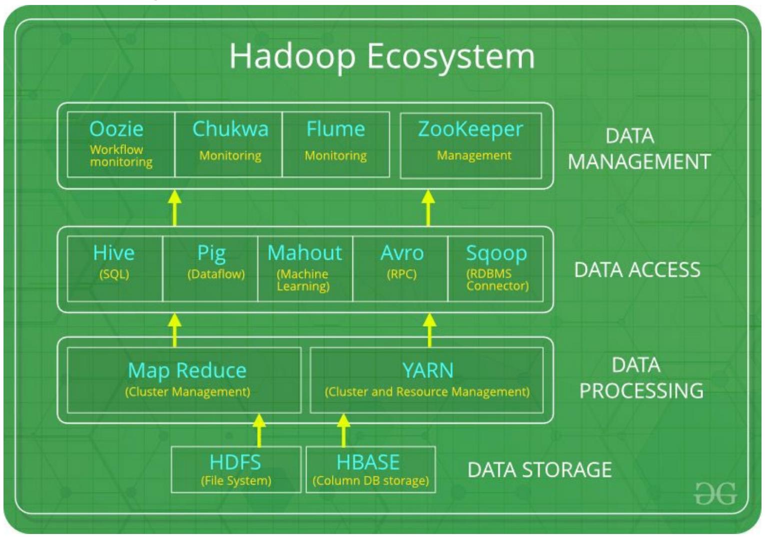
WAYS TO ACCESS:


1. **Specialized Search Engines:**
   - Use search engines designed for the Invisible Web.
2. **Subscription Databases:**
   - Access subscription-based databases and repositories.
3. **Library Catalogs and Archives:**
   - Explore library catalogs and digital archives.
4. **Government and Institutional Websites:**
   - Visit authoritative government and institutional sites.
5. **Specialized Directories:**
   - Utilize directories and guides for organized links.
6. **Social Media and Forums:**
   - Join social platforms for shared links.
7. **Deep Web Search Engines:**
   - Explore search engines for deep web content.
8. **Educational and Research Portals:**
   - Visit portals for scholarly articles and research.
9. **Password-Protected Sites:**
   - Obtain authorized access to protected sites.

.0. **Web Directories and Virtual Libraries:**
   - Explore organized directories for ↓ ,able resources.

HADOOP ECOSYSTEM(Do this properly again !!)

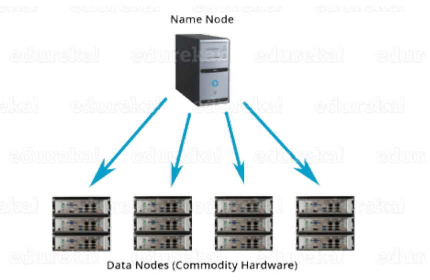
- Framework to handle big data
- Big data is a term given to the data sets which can't be processed in an efficient manner with the help of traditional methodology such as RDBMS.
- open-source framework for distributed storage and processing of large data sets
- Characteristics
    - Distributed:
        - Processes data across multiple machines for improved efficiency and parallelism.
    - Scalable:
        - Capable of handling growing amounts of data by adding resources without a fundamental change in architecture.
    - Reliable:
        - Ensures consistent performance by being fault-tolerant, recovering from errors, and maintaining data integrity.
- Hadoop is made up of several modules that are supported by a large ecosystem of technologies.
- There are four major elements of Hadoop i.e. HDFS, MapReduce, YARN, and Hadoop Common Utilities.
- components that collectively form a Hadoop ecosystem:
    - HDFS: Hadoop Distributed File System
    - YARN: Yet Another Resource Negotiator
    - MapReduce: Programming based Data Processing
    - Spark: In-Memory data processing
    - PIG, HIVE: Query based processing of data services
    - HBase: NoSQL Database
    - Mahout, Spark MLLib: Machine Learning algorithm libraries
    - Solar, Lucene: Searching and Indexing
    - Zookeeper: Managing cluster
    - Oozie: Job Scheduling



- **HDFS**
    - Primary / major comp
    - Resp for storing large data sets of structured/unstructued data across various nodes
    - Two core comps:
        - Name Node

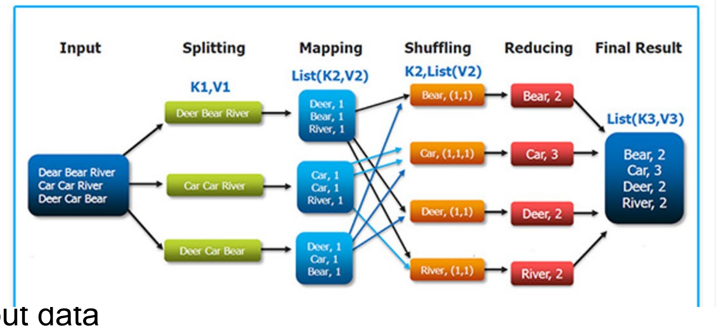
        - prime node which contains metadata (data about data)
        - comparatively fewer resources than the data nodes that stores the actual data
        - Manages the filesystem namespace
        - Maintains the filesystem tree and the metadata for all the files and directories in the tree
        - Stored in RAM
        - knows the data nodes on which all the blocks for a given file are located

    - Data Node:
        - Stores actual data

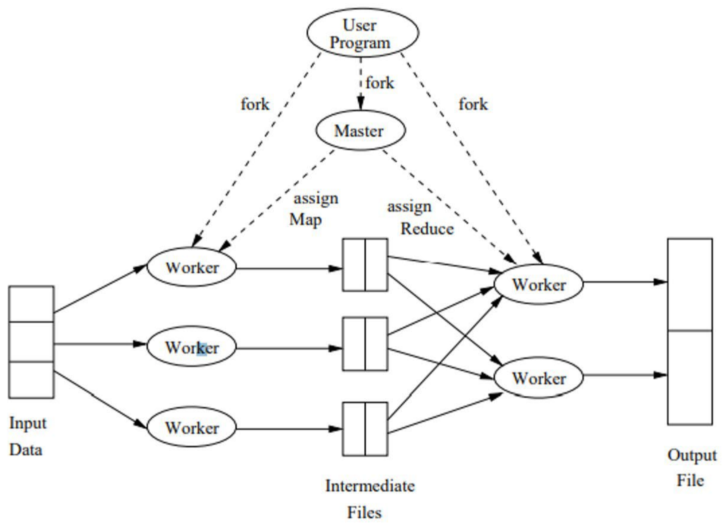


- Data Distribution:
    - Large files are divided into fixed-size blocks (typically 128 MB or 256 MB) to facilitate parallel processing and efficient storage.
    - These blocks are then distributed across multiple nodes in the Hadoop cluster. Each block is replicated to ensure fault tolerance in case a node fails.
- NameNode and DataNode:
    - The HDFS architecture consists of two main components: the NameNode and DataNodes.
        - NameNode: Manages the metadata and namespace of the file system, including information about the location and replication level of each block.
        - DataNodes: Store the actual data blocks. They receive instructions from the NameNode on where to store, replicate, and retrieve data.
- Read and Write Operations:
    - Write Operation:
        - When a client wants to store a file in HDFS, it communicates with the NameNode to determine the locations for block storage.
        - The client sends the data to the designated DataNodes, which store the data blocks and acknowledge the successful write.
        - The NameNode updates the metadata with the new file information.
    - Read Operation:
        - When a client wants to read a file, it contacts the NameNode for the metadata.
        - The NameNode provides the client with the locations of the relevant DataNodes.
        - The client then retrieves the data directly from the DataNodes in parallel.
- Fault Tolerance:

    - HDFS achieves fault tolerance through block replication. By default, each block is replicated three times across different DataNodes.
    - If a DataNode fails or a block becomes unavailable, HDFS can retrieve the data from one of the replicas stored on another node.
- Balancing and Maintenance:
    - HDFS periodically checks the health and storage capacity of DataNodes.
    - The system rebalances data by moving blocks between nodes to ensure even distribution and optimal resource utilisation.
- YARN:
    - Yet Another Resource Negotiator, as the name implies, YARN is the one who helps to manage the resources across the clusters. In short, it performs scheduling and resource allocation for the Hadoop System.
    - Three components:
        - Resource Manager
            - privilege of allocating resources for the applications in a system
        - Node Manager
            - on the allocation of resources such as CPU, memory, bandwidth per machine and later on acknowledges the resource manager.
        - Application Manager
            - Application manager works as an interface between the resource manager and node manager and performs negotiations as per the requirement of the two.
- Map Reduce
- Programming paradigm
- To help solve Big data problems
- Specifically sorting intensive jobs or disk read intensive
- Two functions needed:
    - Mapper: converts input into key-value pairs
    - Reducer: Aggregate all values for keys
- WORD COI



- Map-Reduce environmen
    - Partitioning the input data
    - Scheduling the program's execution across a set of machines
    - Performing the group by key step
    - Handling machine failures
    - Managing required inter-machine communication



- Data flow:
  - Input and final output are stored on a distributed file system (FS):
  - Scheduler tries to schedule map tasks “close” to physical storage location of input data
  - Intermediate results are stored on local FS of Map and Reduce workers
  - Output is often input to another MapReduce task
- MAster node: coordination
  - Task status: (idle, in-progress, completed)
  - Idle tasks get scheduled as workers become available
  - When a map task completes, it sends the master the location and sizes of its R intermediate files, one for each reducer
  - Master pushes this info to reducers
  - Master pings workers periodically to detect failures
- Failures:
  - Map worker failure
    - Map tasks completed or in-progress at worker are reset to idle
    - Reduce workers are notified when task is rescheduled on another worker
  - Reduce worker failure
    - Only in-progress tasks are reset to idle
    - Reduce task is restarted
  - Master failure
    - MapReduce task is aborted and client is notified

- **PIG**
  - Pig was basically developed by Yahoo which works on a pig Latin language, which is Query based language similar to SQL.
  - It is a platform for structuring the data flow, processing and analyzing huge data sets.

  - Pig does the work of executing commands and in the background, all the activities of MapReduce are taken care of. After the processing, pig stores the result in HDFS.
  - Pig Latin language is specially designed for this framework which runs on Pig Runtime. Just the way Java runs on the JVM.
  - Pig helps to achieve ease of programming and optimization and hence is a major segment of the Hadoop Ecosystem.
- Others ARE HIVE, MAHOUT,APACHE SPARK APACHE HBASE ETC
- Hadoop Common utilities include tools like DistCp for data copying, HAR for archiving, FileSystem Shell for command-line HDFS interaction, IOUtils for input/output operations, RPC for communication, security utilities, metrics for monitoring, and annotations for metadata in the Hadoop ecosystem. These tools enhance Hadoop's capabilities in data handling, system management, and security.

Unit 3

Vector space model

- Boolean queries: a document either matches or does not match a query
  - Large document collections, the resulting number of matching documents can far exceed the number a human user could possibly sift through.
  - essential for a search engine to rank-order the documents matching a query. To do this, the search engine computes, for each matching document, a score with respect to the query at hand
- Therefore, calc a score for every (query,doc) pair
- Parametric and zone indexes -> allow us to index and retrieve docs by metadata; simple means of scoring
- Each doc as a vector of weights (importance of term based on statistics occurrences of term) -> compute score between query and each doc == Vector space scoring

Here's a breakdown of the key aspects of VSM:

1. Documents and Queries as Vectors:

- Each document in a collection is represented as a vector.
- This vector has a dimension for every unique term in the entire collection.
- The value at each dimension indicates the weight or importance of that term for that specific document.

2. Term Weighting:

- Not all terms are created equal. A simple count of term occurrences (term frequency) isn't enough.
- Term weighting schemes like tf-idf (term frequency-inverse document frequency) are used to assign importance to terms.
- tf-idf considers both how often a term appears in a document (tf) and how rare it is across the entire document collection (idf).

3. Cosine Similarity:

- Once documents and queries are represented as vectors, we can compare them using cosine similarity.
- Cosine similarity measures the angle between two vectors.

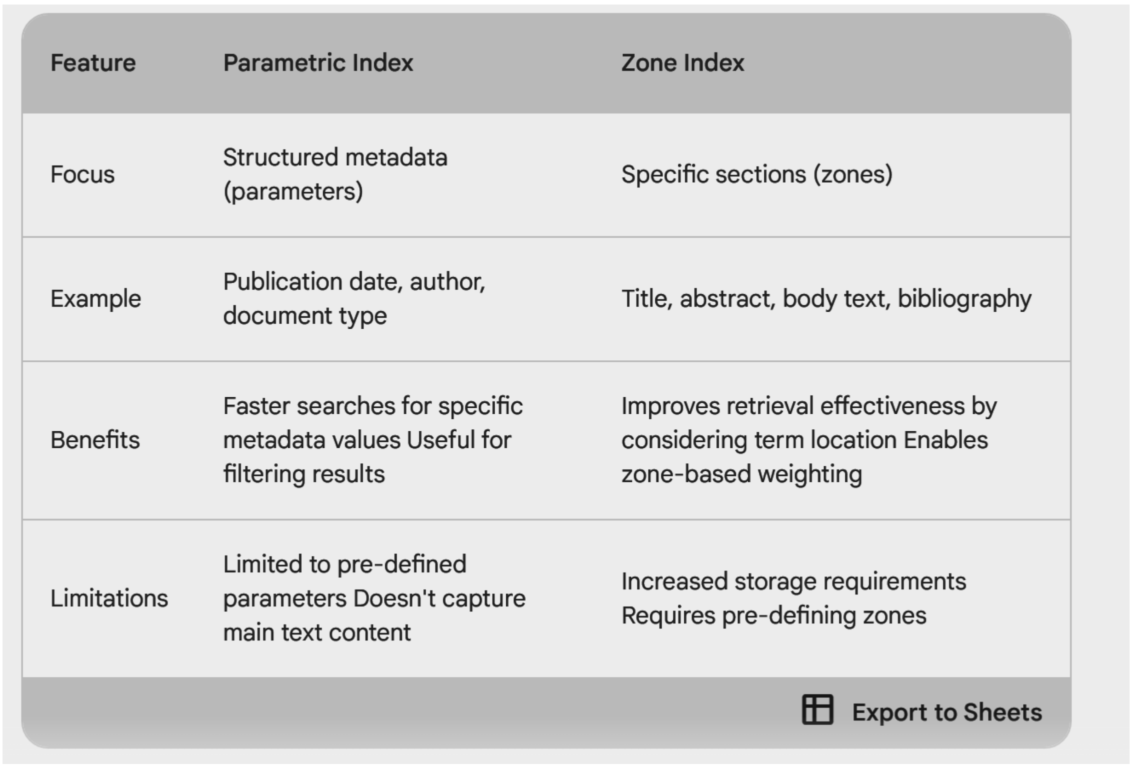
- A higher cosine similarity score indicates a greater closeness between the document and query vectors, suggesting higher relevance.

4. Benefits of VSM:

- Enables efficient retrieval of documents based on their semantic similarity to a query.
- Relatively simple to implement and computationally efficient.
- Flexible - can be adapted to incorporate different weighting schemes and similarity measures.

5. Limitations of VSM:

- Ignores word order and context within documents.
- Doesn't capture synonyms or related terms.
- May struggle with polysemy (words with multiple meanings).



Feature	Parametric Index	Zone Index
Focus	Structured metadata (parameters)	Specific sections (zones)
Example	Publication date, author, document type	Title, abstract, body text, bibliography
Benefits	Faster searches for specific metadata values Useful for filtering results	Improves retrieval effectiveness by considering term location Enables zone-based weighting
Limitations	Limited to pre-defined parameters Doesn't capture main text content	Increased storage requirements Requires pre-defining zones

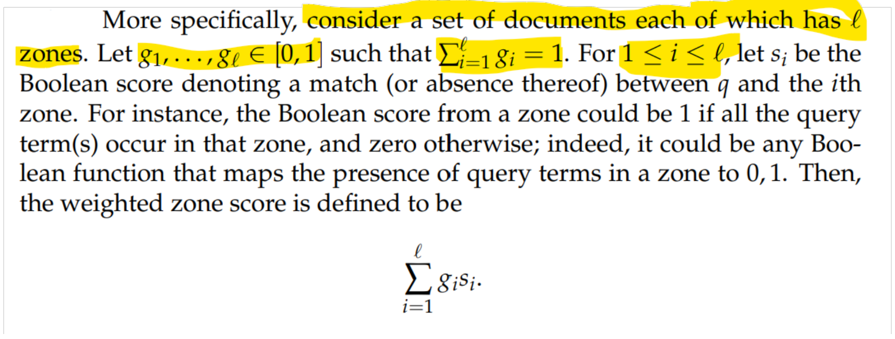
PARAMETRIC AND ZONE INDEXES

- Digital docs have more structure than just seq of terms
- Generally encode metadata
    - Data about data
    - fields such as the date of creation and the format of the document, as well the author and possibly the title of the document
    - Finite possible values
- Parametric indexes are inverted indexes built for specific parameters, or fields, that support parametric search (e.g. "all documents from author Z containing. word Y").
    - a structure like a B-tree may be used for the field’s dictionary

- Zones are a similar concept applied to arbitrary free text (e.g. portion of a. document).
- . Whereas a field may take on a relatively small set of values, a zone
- can be thought of as an arbitrary, unbounded amount of text.
- reduce the size of the dictionary by encoding the zone in which a term occurs in the postings
- the
- WEIGHTED ZONE
    - efficient computation of scores using a technique we will call weighted zone scoring

WEIGHTED ZONE SCORING

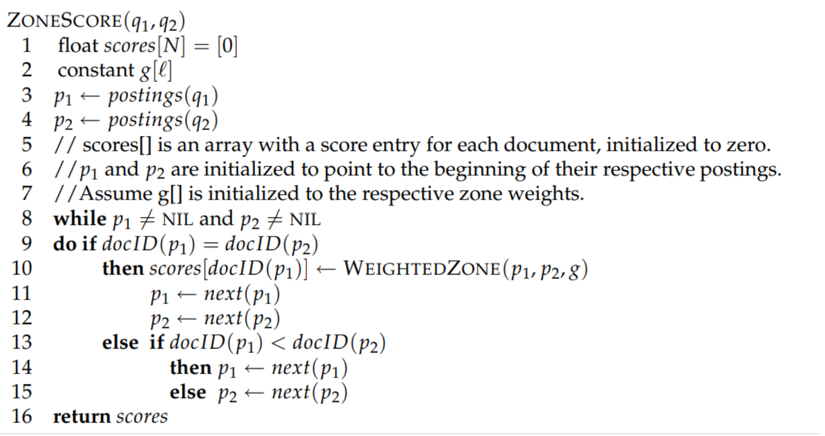
- Weighted zone scoring is a technique used in information retrieval (IR) to rank documents based on how well they match a user's query, considering the importance of different sections of the document (zones) and the frequency of query terms within those zones.
- Given query q and doc d, weighted zone scoring assigns to the pair (q,d) a score in the interval [0,1] by computing a linear combination of zone scores
    - Each zone of doc contribs a boolean val



More specifically, consider a set of documents each of which has ℓ zones. Let $g_1, \ldots, g_\ell \in [0, 1]$ such that $\sum_{i=1}^{\ell} g_i = 1$. For $1 \leq i \leq \ell$, let s_i be the Boolean score denoting a match (or absence thereof) between q and the ith zone. For instance, the Boolean score from a zone could be 1 if all the query term(s) occur in that zone, and zero otherwise; indeed, it could be any Boolean function that maps the presence of query terms in a zone to 0, 1. Then, the weighted zone score is defined to be

$$\sum_{i=1}^{\ell} g_i s_i.$$

- Computation:
    - Simple approach:
        - compute score for each doc in turn
        - Add all contribs from various zones
    - we may compute weighted zone scores directly from inverted indexes.
    - Eg algo:
        - when the query q is a two term query consisting of query terms q1 and q2, and the Boolean function is
        - AND: 1 if both query terms are present in a zone and 0 otherwise.



```
ZONESCORE($q_1, q_2$)
float scores[$N$] = [0]
constant g[$\ell$]
$p_1 \leftarrow postings(q_1)$
$p_2 \leftarrow postings(q_2)$
// scores[] is an array with a score entry for each document, initialized to zero.
//$p_1$ and $p_2$ are initialized to point to the beginning of their respective postings.
//Assume g[] is initialized to the respective zone weights.
while $p_1 \neq$ NIL and $p_2 \neq$ NIL
do if $docID(p_1) = docID(p_2)$
      then $scores[docID(p_1)] \leftarrow$ WEIGHTEDZONE($p_1, p_2, g$)
           $p_1 \leftarrow next(p_1)$
           $p_2 \leftarrow next(p_2)$
      else if $docID(p_1) < docID(p_2)$
             then $p_1 \leftarrow next(p_1)$
             else $p_2 \leftarrow next(p_2)$
return scores
```

    -
    - Score(document, query) = ∑ (zone_weight_i * tf(query_term, zone_i)) for all zones i

        - For each zone, a score is calculated based on how well it matches the query. This typically involves a Boolean function (AND, OR) that considers if all or any query terms are present in the zone.
        - The weight of the zone is then multiplied by this zone score to get a weighted zone contribution.
        - The final score for a document is computed by summing the weighted zone contributions from all its zones. Documents with a higher score are considered more relevant to the user's query.
        - Benefits:
            - Improved Ranking: Weighted zone scoring helps prioritise documents where query terms appear in important sections (e.g., title) compared to less important sections (e.g., body text).
            - Flexibility: You can adjust zone weights to reflect the specific needs of your information retrieval system.

Term frequency - inverse document frequency(TF-IDF)

Concept:

- In information retrieval (IR), the relevance of a document (d) to a query (q) is often determined by how well the document mentions the query terms.
- Term frequency (TF) and inverse document frequency (IDF) are weighting schemes used to assess the importance of a term within a document and across the document collection.

Free Text Queries:

- Users typically enter queries as a set of words without specific operators (e.g., "information retrieval").
- Scoring these queries involves considering the presence and importance of individual terms within documents.

Term Frequency (TF):

- The simplest approach to weighting is based on term frequency (TF).
- TF(t, d) represents the number of times term t appears in document d.

Example:

- Consider a document (d) with the sentence: "The information retrieval system is efficient."
    - TF("information", d) = 1 (appears once)
    - TF("system", d) = 1 (appears once)
    - TF("the", d) = 2 (appears twice)

Problem with TF:

- While TF captures the importance of a term within a document, it doesn't consider the term's overall prevalence in the collection.
- Frequent words like "the" or "a" might have high TF but contribute little to relevance.
- Longer documents naturally have more terms, potentially inflating TF values.

Inverse Document Frequency (IDF):

- To address this issue, IDF considers how common a term is across the entire document collection.
- IDF(t) is calculated as the logarithm of the total number of documents (N) in the collection divided by the number of documents (df(t)) containing term t.
- **Formula: IDF(t) = log(N / df(t))**
- Here, a high IDF value indicates a less frequent term, potentially carrying more weight for relevance.

Example (assuming 10 documents in the collection):

- If "information" appears in 5 documents (df("information") = 5), then IDF("information") = log(10 / 5) = 0.3
- If "the" appears in all 10 documents (df("the") = 10), then IDF("the") = log(10 / 10) = 0 (logarithm of 1 is 0)

TF-IDF Weighting:

- TF-IDF combines the strengths of TF and IDF to provide a more comprehensive weight for a term.
- TF-IDF(t, d) is calculated by multiplying TF(t, d) and IDF(t).
- **Formula: TF-IDF(t, d) = TF(t, d) * IDF(t)**

Overlap score measure :

- score of soc d is sum, over all query terms of the number of times each of the query terms occur in d

$$\text{Score}(q,d) = \sum_{t \in q} \text{tf-idf}_{t,d}.$$

Vector Space Model for Scoring

Concept:

- The Vector Space Model (VSM) represents documents and queries as vectors in a high-dimensional space.
- Each dimension corresponds to a term in the vocabulary, and the weight of a term in a document or query determines its value on that axis.

Documents and Queries as Vectors:

- Imagine a space with axes representing terms from your document collection (potentially millions of dimensions).
- Each document is represented as a vector in this space. The value on each axis (dimension) reflects the weight (importance) of the corresponding term within that document.
- Similarly, a user's query is also represented as a vector in the same space, with weights for each query term.

Example:

Consider a document about "space exploration" and a query "planets around distant stars."

- The document vector might have high weights for terms like "space," "exploration," and potentially lower weights for "stars" and "planets."
- The query vector would likely have high weights for "planets" and "stars," and potentially lower weights for "exploration" and related terms not explicitly mentioned in the query.

Similarity and Ranking:

- VSM ranks documents based on their similarity to the query vector.
- Intuitively, documents with vectors closer to the query vector in this space are considered more relevant.

Measuring Similarity: Cosine Similarity

- Euclidean distance might not be the best measure for high-dimensional spaces.

- VSM often uses cosine similarity to assess the angle between the query and document vectors.

Formula Breakdown:

- Cosine Similarity (query, document) = dot_product(query, document) / (||query|| * ||document||)
- **dot_product(query, document):** This calculates the sum of the products of corresponding elements from the query and document vectors. It captures how well the vectors align in terms of direction and weight of their components.
- **||query|| and ||document||:** These represent the lengths (magnitudes) of the query and document vectors, respectively. They are calculated using the L2 norm, which involves squaring each component, summing them up, and then taking the square root.

Real World Example:

Imagine two documents (d1 and d2) about astronomy:

- Doc 1: "Exploring the Milky Way galaxy and its billions of stars." (Focuses on Milky Way)
- Doc 2: "Scientists discover a new planet orbiting a distant star." (Focuses on exoplanets)
- Let's assume the query is "planets around distant stars."
- Using TF-IDF weights for terms, the query vector might have higher weights for "planets" and "stars" compared to "Milky Way."
- Doc 2, which talks about exoplanets, would likely have a vector closer to the query vector in terms of both direction and weight of terms.
- Therefore, based on cosine similarity, Doc 2 would be ranked higher than Doc 1 for this particular query.

Normalisation:

- VSM often normalises document and query vectors (dividing each component by its L2 norm) to ensure all vectors have a unit length. This makes cosine similarity calculation independent of the overall length of the vectors and focuses purely on the directional alignment.

EVALUATION IN IR(not so useful ?)

- To measure ir effectiveness,we need a test collection of 3 things:
  - Doc collection
  - Test suite of info needs, expressible as queries
  - Set of relevance judgements, binary assessments of either relevant or non relevant for each query-doc pair
- Gold Standard / Ground truth judgement of relevance
  - a document in the test collection is given a binary classification as either relevant or nonrelevant. W.r.t to user info need
- Relevance - assessed rel to info need NOT query

Tuning System Parameters:

- Many IR systems have various weights or parameters that can be adjusted to influence how the system retrieves documents.
- Adjusting these parameters can improve the system's performance.

Overfitting on Test Collections:

        - The passage warns against tuning parameters directly on the test collection used for evaluation.
        - This is because if you optimise the system for a specific set of queries (the test collection), it might perform poorly on unseen queries encountered in real-world use.

Development Test Collection:

- To avoid overfitting, the passage suggests using a separate development test collection.
- This collection is used specifically for tuning the system's parameters.
- You can adjust the weights and parameters on this development collection to improve the system's general performance.

Evaluation on Separate Test Collection:

- Once the parameters are tuned on the development collection, the system's performance is evaluated on a separate test collection.
- This test collection should not have been used for tuning and should represent a more realistic sample of user queries.
- The results obtained on this separate test collection provide a more unbiased estimate of the system's expected performance in real-world scenarios.

Benefits of This Approach:

- Prevents overfitting and provides a more realistic picture of the system's effectiveness.
- Ensures the system performs well on a broader range of queries, not just the specific ones used for tuning.

EVALUATION OF UNRANKED RETRIEVAL SETS

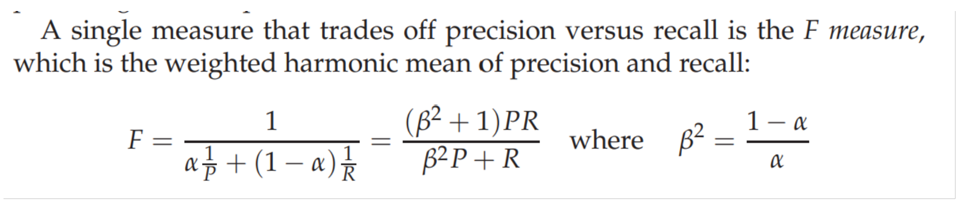
- Most freq and basic measures for ir:
    - Precision
    - Recall
- Precision
    - Fraction of retrieved docs which are relevant
    - $$\text{Precision} = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})} = P(\text{relevant}|\text{retrieved})$$
- Recall
    - Fraction of rel docs that are retrieved
    - $$\text{Recall} = \frac{\#(\text{relevant items retrieved})}{\#(\text{relevant items})} = P(\text{retrieved}|\text{relevant})$$
    -

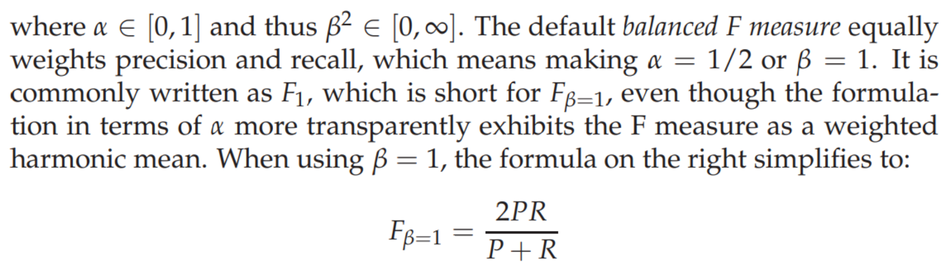
	Relevant	Nonrelevant
Retrieved	true positives (tp)	false positives (fp)
Not retrieved	false negatives (fn)	true negatives (tn)

Then:

$$\begin{aligned} P &= tp/(tp+fp) \\ R &= tp/(tp+fn) \end{aligned}$$

- Accuracy
    - Fraction of classifications that are correct
        - (tp + tn) / (tp + f p + f n + tn).
        -
    - Not a good measure
- F measure

- 
  A single measure that trades off precision versus recall is the *F measure*, which is the weighted harmonic mean of precision and recall:

  $$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R} \quad \text{where} \quad \beta^2 = \frac{1-\alpha}{\alpha}$$
- 
  where $\alpha \in [0,1]$ and thus $\beta^2 \in [0,\infty]$. The default *balanced F measure* equally weights precision and recall, which means making $\alpha = 1/2$ or $\beta = 1$. It is commonly written as F_1, which is short for $F_{\beta=1}$, even though the formulation in terms of α more transparently exhibits the F measure as a weighted harmonic mean. When using $\beta = 1$, the formula on the right simplifies to:

  $$F_{\beta=1} = \frac{2PR}{P+R}$$
- **Why not just use arithmetic mean?**
- The arithmetic mean (average) of precision and recall wouldn't be ideal because:
- It doesn't penalise models that have very low values in either precision or recall.
- A high value in one metric could mask a poor value in the other.
- The advantage of the harmonic mean:
- The harmonic mean gives more weight to lower values of precision and recall. This ensures that a model can't achieve a good F-measure by having a very high value in one metric and a very low value in the other.

For Ranked retrieval results

- In ranked retrieval, documents are retrieved and ordered based on their estimated relevance to a user's query.
- The top K retrieved documents form a ranked list, where K represents the number of documents displayed to the user (e.g., top 10 search results).

1. Precision@K (P@K):

- Precision@K measures the accuracy of the top K retrieved documents.
- It essentially tells us what proportion of the initial K responses are actually relevant to the query.
- P@K = (Number of relevant items in the top K results) / (Total number of items retrieved in the top K results)
- A higher P@K value indicates that a greater proportion of the top K retrieved documents are relevant. In this case, 60% of the top 10 results are relevant.

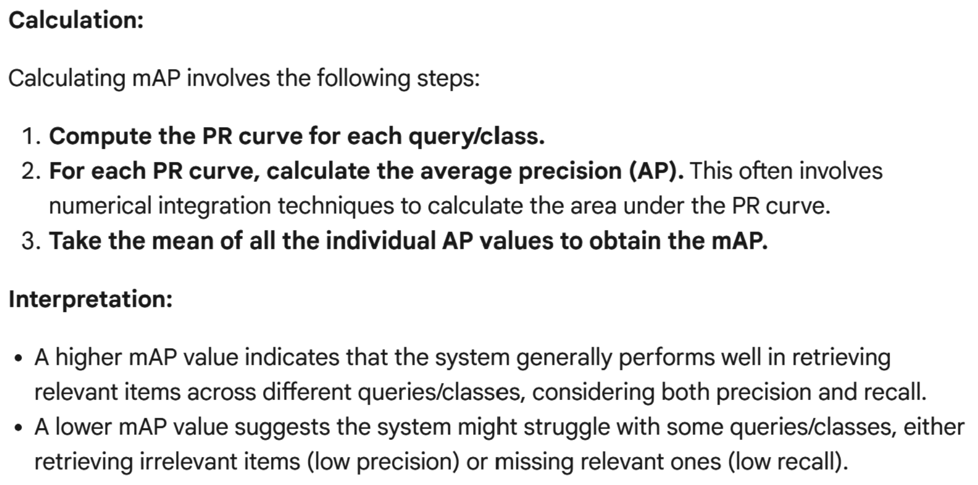
2. Recall@K (R@K):

- Recall@K focuses on completeness. It measures the proportion of all relevant items in the entire dataset that the system retrieved within the top K results.
- R@K = (Number of relevant items in the top K results) / (Total number of relevant items in the entire collection)
- Recall@K indicates how well the system retrieves relevant documents. Here, the system retrieved 6 out of 20 relevant documents within the top 10 results (30%).

Mean Average Precision (MAP)

- Consider rank posn of each rel doc, k1,k1,....kn
- Compute precisiomn@k for each
- Take avg
- MAP is avg precision across multiple queries/rankings
- Average Precision:
  - This is a single value calculated for each query or class in the dataset.

        - It summarises the precision-recall curve (PR curve) for a specific query/class by averaging the precision values obtained at various recall levels.
    - The mAP takes the average of the AP values calculated for all queries or classes in the dataset.
    - This provides a single score that represents the overall performance of the system across different retrieval difficulties.



Calculation:

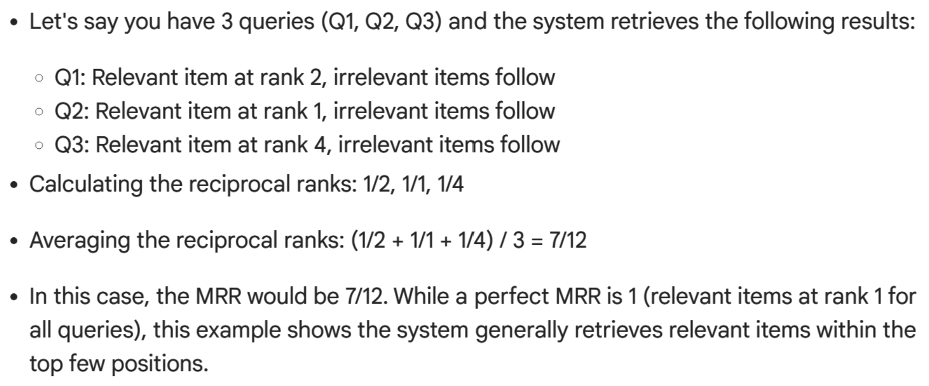
Calculating mAP involves the following steps:

1. **Compute the PR curve for each query/class.**
2. **For each PR curve, calculate the average precision (AP).** This often involves numerical integration techniques to calculate the area under the PR curve.
3. **Take the mean of all the individual AP values to obtain the mAP.**

Interpretation:

- A higher mAP value indicates that the system generally performs well in retrieving relevant items across different queries/classes, considering both precision and recall.
- A lower mAP value suggests the system might struggle with some queries/classes, either retrieving irrelevant items (low precision) or missing relevant ones (low recall).

Mean Reciprocal Rank(MRR)

- evaluate the effectiveness of a system in ranking relevant items
- position of the first relevant item in the retrieved list for each query.
- MRR tells you, on average, how high you need to go down the ranked list (reciprocal rank) to find the first relevant item across all your searches (queries).
- Formula:
    - MRR = 1 / |Q| * Σ (1 / rank_i)
- Where
    - |Q| is the total number of queries
    - rank_i is the rank position of the first relevant item for the i-th query
- 

    - Let's say you have 3 queries (Q1, Q2, Q3) and the system retrieves the following results:
        - Q1: Relevant item at rank 2, irrelevant items follow
        - Q2: Relevant item at rank 1, irrelevant items follow
        - Q3: Relevant item at rank 4, irrelevant items follow
    - Calculating the reciprocal ranks: 1/2, 1/1, 1/4
    - Averaging the reciprocal ranks: (1/2 + 1/1 + 1/4) / 3 = 7/12
    - In this case, the MRR would be 7/12. While a perfect MRR is 1 (relevant items at rank 1 for all queries), this example shows the system generally retrieves relevant items within the top few positions.
-
- Multiple Levels of relevance
    - Normalised Discounted Cumulative Gain (NDCG)
        - evaluate the system's ability to rank relevant items based on their degree of relevance.

            - goes beyond simply measuring if an item is relevant or not, and considers how well the system ranks highly relevant items over marginally relevant ones.
            - Two crucial aspects/assumptions:
                - Highly relevant documents are more useful than marginally relevant documents
                - the lower the ranked position of a relevant document, the less useful it is for the user, since it is less likely to be examined
    - **Discounted Cumulative Gain (DCG)**:
        - DCG is a metric used in information retrieval (IR) to evaluate the quality of ranked lists of documents.
        - It considers both the relevance of retrieved documents and their position in the ranking.
        - Documents ranked higher in the list contribute more to the overall gain.
    - **Ideal Discounted Cumulative Gain (IDCG):**
        - IDCG represents the highest possible DCG score achievable for a given set of documents and their relevance scores.
        - It assumes a perfect ranking where all documents are listed in descending order of relevance.
    - **Normalised Discounted Cumulative Gain (NDCG):**
        - DCG scores can vary depending on the number and relevance levels of documents retrieved for a query.
        - NDCG addresses this by normalising DCG against the IDCG for the same set of documents.
        - NDCG@k = DCG@k / IDCG@k

User Behaviour

- Features based on user behaviour:
    - Click-through features
        - Click frequency, click probability, click deviation
        - Click on next result? previous result? above? below>?
    - Browsing features
        - Cumulative and average time on page, on domain, on URL prefix; deviation from average times
        - Browse path features
    - Query-text features
        - Query overlap with title, snippet, URL, domain, next query
        - Query length

Evaluating IR systems goes beyond simply measuring how well they find relevant documents. Here are other crucial factors:

- **Speed:** How fast can the system process documents (indexing) and respond to queries (search speed)?
- **Query Language:** Can users express complex search criteria?
- **Document Collection:** Is the collection large, diverse, and covers a broad range of topics?
- **User Interface:** How easy and intuitive is the system to interact with?

Measuring User Satisfaction:

- **Return Rate:** Do users come back for future searches (indicates user preference)?
- **Click-Through Rate (CTR):** For web search, how often do users click results, especially on the first page?
- **Time to Purchase (E-commerce):** Does the search experience lead to faster purchases?
- **User Studies:** Observing user behaviour and gathering feedback provides valuable insights.

A/B Testing:

- This technique helps evaluate changes to a deployed system:
    - Create two versions: original and modified.
    - Randomly assign users to each version.
    - Analyse user behaviour (CTR, search time) to see if the change improves things.

Result Snippets:

- These are summaries of retrieved documents displayed in search results.
- They help users assess relevance without opening the document.
- Two types exist:
    - Static Snippets: Remain constant, often showing title and first few sentences.
    - Dynamic Snippets (Query-Dependent): Customised based on the query, highlighting relevant sections.

RELEVANCE IN IR

- Problem:
    - Synonymy (different words referring to the same concept) affects recall in information retrieval systems.
    - Users might miss relevant documents due to using different terms for the same concept.
- Solutions:
    - Query refinement: Users can manually refine queries to include synonyms.
    - Automatic query refinement by the system
- Two Main Approaches:
- Global Methods:
    - Expand or reformulate the query terms independent of the specific search.
    - Examples:
    - Using a thesaurus or WordNet.
    - Automatic thesaurus generation.
    - Spelling correction
- Local Methods:
    - Adjust the query based on the initially retrieved documents.
    - Examples:
    - Relevance feedback
    - Pseudo relevance feedback (blind relevance feedback).
    - Global indirect relevance feedback.

RELEVANCE FEEDBACK

- The idea of relevance feedback (RF) is to involve the user in the retrieval process so as to improve the final result set
- User gives feedback on rel of docs
- Procedure:
  - The user issues a (short, simple) query.
  - The system returns an initial set of retrieval results.
  - The user marks some returned documents as relevant or nonrelevant.
  - The system computes a better representation of the information need based on the user feedback.
  - The system displays a revised set of retrieval results.
- Relevance feedback can go through one or more iterations of this sort.
- The process exploits the idea that it may be difficult to formulate a good query when you don't know the collection well, but it is easy to judge particular documents, and so it makes sense to engage in iterative query refinement of this sort. I
- Rocchio Algo for relevance feedback:
  - classic algorithm for implementing relevance Feedback.
  - It models a way of incorporating relevance feedback information into the vector space model
  - Underlying theory: We want to find a query vector, denoted as ~q, that maximises similarity with relevant documents while minimising similarity with nonrelevant documents.
  - Steps:
    - Representing Documents and Query as Vectors:
      - Each document and the user's query are transformed into vectors. These vectors contain terms (words) from the documents and query, with each term's weight reflecting its importance.
    - Calculating Centroids:
      - Relevant Centroid: This represents the "center" of the relevant documents identified by the user. It's calculated by averaging the weight vectors of all relevant documents.
      - Non-Relevant Centroid: This represents the "center" of the irrelevant documents. It's calculated by averaging the weight vectors of all irrelevant documents.
    - Updating the Query Vector:
      - The Rocchio algorithm modifies the initial query vector by:
        - Moving it closer to the relevant centroid using a positive weight factor (a).
        - Moving it away from the non-relevant centroid using a negative weight factor (b).
      - Optionally, the original query terms can also be included with a weight factor (c) to maintain some stability in the search.
    - New Query for Next Search:
      - This adjusted query vector is then used for the next search iteration, aiming to retrieve documents more aligned with the user's intent.

Relevance Feedback Effectiveness:

- The effectiveness of relevance feedback depends on several factors:
    - Initial Query Relevance: The user's initial query should be reasonably close to the desired documents. A very broad or irrelevant query might not provide enough context for the feedback to be effective.
    - Similarity of Relevant Documents: Ideally, the relevant documents should share vocabulary and topic (form a cluster in vector space). This strengthens the association between terms and relevance.
- Limitations:
    - Misspellings: Relevance feedback can't address misspelt terms if they aren't present in any document.
    - Cross-Language Retrieval: Documents in different languages won't be close together in the vector space due to vocabulary differences.
    - Vocabulary Mismatch: If the user's search terms don't match the document vocabulary (e.g., "laptop" vs. "notebook computer"), feedback won't be helpful.
    - Multimodal Relevant Documents:
        - The Rocchio model assumes relevant documents form a single cluster.
        - It struggles if relevant documents fall into multiple clusters with distinct vocabularies (e.g., "Burma" vs. "Myanmar").
        - Disjunctive queries (e.g., "Pop stars who worked at Burger King") can also lead to multimodal clusters.
        - Well-structured collections with informative content can help bridge these vocabulary gaps between clusters.
- User Considerations:
    - Users might be reluctant to provide explicit feedback due to time constraints or lack of interest in prolonging the search interaction.
    - Understanding why a particular document is retrieved after feedback can be difficult for users.
- Practical Issues:
    - The long queries generated by traditional relevance feedback can be inefficient for IR systems, leading to high processing costs and longer response times.
    - A potential solution is to reweight only a limited number of prominent terms from the relevant documents.

RELEVANCE ON THE WEB

- Limited Adoption:
    - Unlike some specialised search engines, most web search engines haven't widely adopted relevance feedback features.
    - One reason is that users generally prefer to complete their searches in a single interaction and are less likely to use advanced features.
    - Even when offered (like the "More like this" option), user uptake of relevance feedback is low.
- Clickstream Data as Indirect Feedback:
    - Some search engines analyse user clicks to infer their preferences and indirectly improve search results.
- Evaluation Challenges:
- Evaluating the effectiveness of relevance feedback is tricky due to factors like:
    - Users might see judged relevant documents during feedback, artificially inflating later results.
    - Fairness: Ideally, evaluation shouldn't consider documents already seen by the user.

    - Difficulty in comparing performance with and without feedback due to a change in the effective collection size (known relevant documents are excluded after feedback).

PSEUDO RELEVANCE FEEDBACK

- Also known as blind rel feedback
- Automatic local analysis
- automates the manual part of relevance feedback, so that the user gets improved retrieval performance without an extended interaction
- Method:
    - do normal retrieval to
    - find an initial set of most relevant documents,
    - to then assume that the top k ranked documents are relevant,
    - and finally to do relevance feedback as before under this assumption.

GLOBAL METHODS FOR QUERY REFORMULATIONS

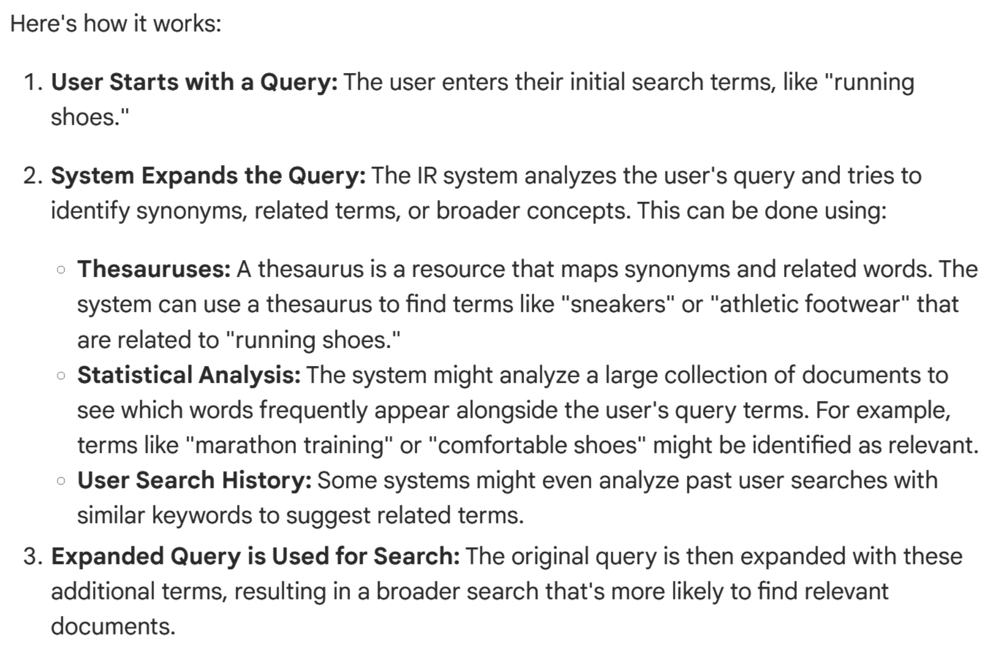
- improve user queries in information retrieval (IR) systems, specifically focusing on global methods that don't require analysing individual documents in the search results.

User support tools:

- IR systems can provide various features to help users understand their search process:
    - Information on omitted words (stop words)
    - Stemmed words
    - Number of hits per term/phrase
    - Option to browse terms in the inverted index for finding relevant terms

QUERY EXPANSION

- Query expansion is a technique used in information retrieval (IR) systems to improve the search process by broadening the user's initial query. It essentially helps the system understand what the user is really looking for, even if their initial phrasing isn't perfect.



Here's how it works:

1. **User Starts with a Query:** The user enters their initial search terms, like "running shoes."
2. **System Expands the Query:** The IR system analyzes the user's query and tries to identify synonyms, related terms, or broader concepts. This can be done using:
    - **Thesauruses:** A thesaurus is a resource that maps synonyms and related words. The system can use a thesaurus to find terms like "sneakers" or "athletic footwear" that are related to "running shoes."
    - **Statistical Analysis:** The system might analyze a large collection of documents to see which words frequently appear alongside the user's query terms. For example, terms like "marathon training" or "comfortable shoes" might be identified as relevant.
    - **User Search History:** Some systems might even analyze past user searches with similar keywords to suggest related terms.
3. **Expanded Query is Used for Search:** The original query is then expanded with these additional terms, resulting in a broader search that's more likely to find relevant documents.

- There are two main benefits to query expansion:
    - Increased Recall: By including more relevant terms, the system can find documents that wouldn't have been retrieved with the original query alone. This is helpful when the user's initial phrasing might be too narrow or miss some key aspects of their information need.

    - Improved Understanding: Even if the user doesn't find the exact documents they were looking for, the expanded terms can help them discover related information they might find useful.
- However, there are also some potential drawbacks to consider:
    - Reduced Precision: Expanding the query can sometimes lead to retrieving irrelevant documents that don't actually match the user's intent. This is especially true if the system picks up on unrelated synonyms or broadens the search too much.
    - Thesaurus Limitations: The quality of the thesaurus or the statistical analysis can significantly impact the effectiveness of query expansion. A poorly maintained thesaurus might introduce irrelevant terms, while limited statistical analysis might miss important related concepts.

Query expansion vs relevance feedback

- Query expansion involves users suggesting additional terms related to their initial query.
- In contrast, relevance feedback involves users directly marking documents as relevant or not to refine the search based on their judgement.
- Search engines might suggest related queries based on a thesaurus or controlled vocabulary.

Thesaurus-Based Query Expansion:

- A thesaurus is a resource that maps synonyms and related words to concepts.
- By analysing the user's query terms, the system can automatically expand the query using synonyms and related terms from the thesaurus.
    - Terms can be weighted differently, with original query terms having higher weight than added terms.

Types of Thesauruses:

- Controlled Vocabulary:
    - Maintained by human editors, with a single canonical term for each concept (e.g., Library of Congress Subject Headings).
- Manual Thesaurus:
    - Human-built lists of synonyms and related words without a designated canonical term (e.g., UMLS metathesaurus).
- Automatically Derived Thesaurus:
    - Statistical analysis of word co-occurrence in documents to identify relationships between words.
- Query Log Mining Thesaurus:
    - Exploits manual reformulations from other users to suggest related queries.

Advantages and Limitations of Thesaurus-Based Expansion:

- Advantages:
    - Doesn't require user input.
    - Generally increases recall (finds more relevant documents), especially in scientific and engineering fields.
- Limitations:
    - Cost of manual thesaurus creation and maintenance.
    - Difficulty in capturing all relevant synonyms and relationships.
    - Potential for irrelevant terms due to word ambiguity (e.g., "Apple" computer vs. red fruit).

    - Reduced precision (may retrieve irrelevant documents) for ambiguous queries.

Automatic Thesaurus Generation:

Statistical methods can be used to automatically generate a thesaurus from a document collection.

One approach relies on word co-occurrence: frequently co-occurring words are assumed to be related.

Another approach uses shallow grammatical analysis to identify relationships based on grammatical structures.

While automatic thesauruses can identify some relevant terms, they can also introduce irrelevant terms due to ambiguity and limited understanding of context.

Overall Notes on Query Expansion:

Query expansion can be effective in improving recall but may decrease precision.

Manually creating and maintaining domain-specific thesauruses is expensive.

Query expansion is generally less successful than relevance feedback but might be more understandable for users.

Unit 4

LINK ANALYSIS

- how search engines analyse the connections between web pages (hyperlinks) to improve search results ranking.
- The structure of the web, with its interconnected pages, is like a giant graph.
- Search engines analyse these links (connections) to understand the importance and relevance of web pages.
- Why?
    - Links can be seen as a way for web pages to "vote" for each other.
    - A page with many high-quality links pointing to it is likely more trustworthy and relevant.
- Simply having a lot of links (even from low-quality pages) isn't enough.
- Link Spam Combating
    - Some websites might try to artificially inflate their ranking by creating fake links pointing to them (link spam).
    - Search engines use sophisticated techniques to identify and devalue such spammy links.
- Link Analysis for crawling
    - Link analysis can also help prioritise which web pages to crawl next.
    - By analysing links, search engines can focus on crawling pages that are likely to be more important or relevant.

Web as a graph

The web can be visualised as a massive graph, where web pages are nodes and hyperlinks act as edges connecting them. Search engines leverage this structure and the text associated with links (anchor text) to analyse and rank web pages. Here's a breakdown of key concepts:

Links as Endorsements:

- A hyperlink from page A to page B suggests that the creator of page A considers page B valuable or relevant.
- The **anchor text**, the clickable text associated with the link, often provides clues about the content of page B.
- For example, a link from a page about "cat breeds" with anchor text "Siamese cats" suggests page B likely focuses on Siamese cats.

Not All Links Are Equal:

- Internal links within a website (e.g., links to contact or privacy pages) might not be strong endorsements.
- Search engines often downplay their influence on ranking compared to links from external websites.

Anchor Text: Beyond Page Descriptions:

- Web pages don't always perfectly describe themselves, especially with limited text or text embedded in images.
- Anchor text from other pages acts as a valuable source of descriptive terms for a target page.
- Search engines can use anchor text to improve indexing, understanding a page's content even if it lacks descriptive text itself.

Leveraging the Community:

- By analysing the anchor text used by many web authors linking to a page, search engines gain insights beyond the page's content.
- Consistent anchor texts across various websites pointing to a page with "italian food recipes" suggest the page likely focuses on that topic, even if the page itself doesn't explicitly mention it.

Using Anchor Text for Indexing:

- Anchor text terms are used to index target web pages, similar to how words within a page are used.
- The frequency of a specific term in anchor texts pointing to a page can indicate its relevance to that term.
- For example, many links with anchor text containing "machine learning tutorials" pointing to a page suggest it likely offers such tutorials.

Weighting Anchor Text:

- Common anchor text words like "click here" or "visit us" are given less weight, similar to how common words within a page are downplayed using techniques like TF-IDF (term frequency-inverse document frequency).

- The actual weighting of anchor text terms often involves complex machine learning algorithms.

Anchor Text and Unexpected Results:

- Anchor text can sometimes lead to surprising search results.
- For example, a search for "big blue" might return IBM's homepage because "Big Blue" is a nickname for IBM, even though the website itself might not explicitly mention it.

Combating Spammy Anchor Text:

- Websites might try to manipulate their ranking by creating misleading anchor text pointing to themselves (e.g., stuffing keywords unrelated to their content).
- Search engines employ techniques to detect and counteract such attempts (spam).

Extended Anchor Text:

- The text surrounding an anchor text (link) can also be informative.
- Search engines might consider this "extended anchor text" for analysis, similar to anchor text itself.
- For instance, a link with anchor text "best pizza places" surrounded by text "Check out our guide to..." might provide additional context for the target page

PAGE RANK

The Core Idea:

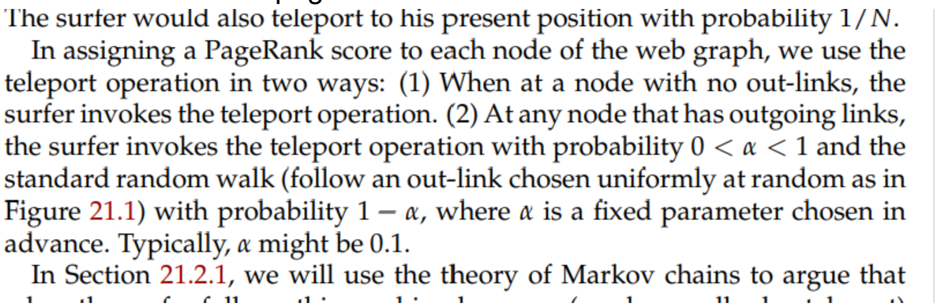
- Imagine a random surfer constantly browsing the web by randomly clicking on links.
- Pages that this surfer visits more frequently are likely more important and relevant.
- PageRank assigns a score to each web page based on this concept of "popularity" in the random walk.

Random Walk Simulation:

- The surfer starts at a random webpage and keeps clicking on outgoing links to new pages.
- At each page, there's an equal chance of clicking on any of the available links.
- Pages with more incoming links are more likely to be visited by the surfer in this random walk.

Teleporting the Surfer:

- To avoid getting stuck on pages with no outgoing links, we introduce "teleportation."
- At a dead end, jump to a random web page
- With a certain probability (e.g., 10%), the surfer jumps to a random webpage instead of following a link.
- This ensures all pages have a chance to be visited and ranked.



The surfer would also teleport to his present position with probability $1/N$.

In assigning a PageRank score to each node of the web graph, we use the teleport operation in two ways: (1) When at a node with no out-links, the surfer invokes the teleport operation. (2) At any node that has outgoing links, the surfer invokes the teleport operation with probability $0 < \alpha < 1$ and the standard random walk (follow an out-link chosen uniformly at random as in Figure 21.1) with probability $1 - \alpha$, where α is a fixed parameter chosen in advance. Typically, α might be 0.1.

In Section 21.2.1, we will use the theory of Markov chains to argue that

PageRank Score:

- The PageRank score of a webpage represents the probability that the random surfer would land on that page after a long series of clicks.
- Pages with many high-quality links pointing to them are more likely to be visited by the surfer, leading to a higher PageRank score.
-

How PageRank is Used:

- Search engines consider PageRank along with other factors (like content similarity) to rank web pages for a search query.
- Pages with higher PageRank are generally considered more relevant and authoritative.

MARKOV CHAINS

- Markov chains are used to analyse web pages and calculate their PageRank scores.

Markov Chains: The Core Concept

- A Markov chain is a series of random events where the probability of the next event depends only on the current event, not the history.
- It's like flipping a coin: the chance of heads on the next flip is always 50%, regardless of what happened in previous flips.
- In our context, each web page represents a state in the Markov chain.

Transition Probability Matrix (P):

- This matrix captures the probabilities of moving from one state (web page) to another.
- Each entry Pij represents the probability of going from page i to page j.
- All entries in a row sum to 1, as all possibilities from a current page are covered.
- The Markov property ensures Pij only depends on the current state (i).

Example:

- Imagine a simple web with 3 pages (A, B, C).
- From page A, you can go to B or C with equal probability (0.5 each).
- From B and C, you can only go back to A (probability 1).
- The transition probability matrix (P) for this example would be:

	A	B	C
A	0	0.5	0.5
B	1	0	0
C	1	0	0

Random Surfer Analogy:

- We can think of PageRank as a random surfer who keeps clicking on links.
- The transition probability matrix defines how this surfer moves between web pages.

Teleporting the Surfer:

- To avoid getting stuck on pages with no outgoing links, we introduce "teleportation."
- With a probability α (e.g., 10%), the surfer jumps to a random webpage instead of following a link.

Markov Chain and Web Graph:

- The adjacency matrix (A) of the web graph shows which pages link to each other (1 for a link, 0 otherwise).
- We can derive the transition probability matrix (P) from the adjacency matrix (A)
- Calculating P:

- **Teleporting:** We introduce the concept of "teleportation." This represents the possibility of the surfer getting bored and randomly jumping to any page on the web, regardless of links. Let α be the probability of teleportation (typically between 0.1 and 0.15).
- **No Outgoing Links:** If a page (row) has no outgoing links (all zeros), all entries in that row of P are set to 1/N (N being the total number of web pages). This ensures the surfer doesn't get stuck.
- **Link Distribution:** For pages with outgoing links, each 1 in a row of A is divided by the number of 1's in that row. This represents the probability distribution of the surfer following any link on that page.
- **Combining Probabilities:** The resulting matrix is then multiplied by 1 - α (probability of not teleporting).
- **Teleporting Chance:** Finally, α/N is added to every entry in the new matrix. This accounts for the chance of randomly teleporting to any page.

Probability Vector (~x):

- This vector represents the probability of the surfer being on each page at a given time.
- Initially, you can assume the surfer can be on any page (all entries in ~x can be set to 1/N).
- As the surfer clicks through web pages (represented by multiplying the probability vector (~x) with the transition matrix (P)), the probability vector gets updated, reflecting the likelihood of being on each page.

Steady-State Probability and PageRank:

- If the random surfer keeps walking for a long time, the probability vector converges to a fixed value called the steady-state probability.
- This represents the long-term frequency of visiting each page.
- In PageRank, the steady-state probability of a web page is considered its importance or ranking.

Ergodic Markov Chains:

- For PageRank to work, the Markov chain representing the web graph needs to be ergodic.
- An ergodic chain ensures all pages can be eventually reached from any other page (with positive probability).
- This guarantees a unique and stable steady-state probability distribution.

Theorem 21.1 and PageRank Calculation:

- This theorem states that an ergodic Markov chain has a unique steady-state probability vector (~π).
- This vector is the principal left eigenvector of the transition probability matrix (P).
- By calculating the steady-state probability vector (~π), we essentially get the PageRank score for each web page.

In summary, Markov chains provide a mathematical framework to model the random walk of a surfer on the web. By analysing the transition probabilities and ensuring ergodicity, we can calculate the PageRank score, which reflects the importance and ranking of web pages in search results.

Topic-Specific PageRank: Ranking Web Pages Based on Specific Interests

This section builds on PageRank to introduce Topic-Specific PageRank, which tailors rankings to user interests.

Beyond Uniform Teleportation:

- Standard PageRank uses a random jump (teleportation) where the surfer can land on any webpage with equal probability.

- Topic-Specific PageRank personalises this jump based on user interests.

Example: Sports Fanatic

- Imagine a user passionate about sports.
- We want webpages related to sports to rank higher in their search results.
- In the web graph, sports pages are likely interconnected (close to each other).
- A standard random surfer might get stuck in this "sports cluster" due to random jumps.

Teleporting to Specific Topics:

- Topic-Specific PageRank modifies the teleport behaviour.
- Instead of jumping to any random page, the surfer jumps to a sports-related page with a certain probability.
- This increases the chance of visiting and ranking sports pages higher.

Implementation:

- We don't need to know all sports pages, just a representative subset (S).
- The teleport probability can be directed towards this set (S) instead of random pages.
- This creates a new "sports-biassed" random walk.

Hubs and Authorities: Ranking Webpages by Importance

This passage describes HITS (Hyperlink-Induced Topic Search), a link analysis algorithm that ranks webpages based on their importance within a specific topic. It identifies two types of important pages:

- Hubs: These are webpages that act as central directories, pointing to many other relevant pages on a particular topic. They have a high number of outgoing links to authoritative sources.
- Authorities: These are webpages that are considered credible sources of information on a topic. They have a high number of incoming links from other relevant pages.

The Algorithm: A Mutually Reinforcing Process

1. Initialization: Assign equal weights (scores) to all webpages relevant to the search topic. This is typically done by starting with a set of webpages containing the query term and expanding it to include linked pages.
2. Hub Score Update: For each webpage, calculate its hub score as the sum of the authority scores of the pages it links to. In simpler terms, a webpage's hub score increases if it links to many high-authority pages.
3. Authority Score Update: For each webpage, calculate its authority score as the sum of the hub scores of the pages that link to it. Essentially, a webpage's authority score increases if it is linked to by many high-scoring hubs.
4. Normalisation: Normalise both hub and authority scores to fall between 0 and 1.
5. Iteration: Repeat steps 2-4 until the scores converge (stabilise). This iterative process allows hubs and authorities to "reinforce" each other's importance.

Intuition Behind the Process:

- Imagine a student studying for an exam. Hubs are like well-organised study guides with links to many relevant resources. Authorities are the trusted textbooks and websites containing in-depth information.
- HITS helps identify both types of valuable resources within a specific topic area.

Benefits of HITS:

- Improves search result ranking by considering both in-links and out-links of a webpage.

- Helps identify high-quality topic-specific resources within a website or the wider web.

Limitations of HITS:

- Can be computationally expensive for large web graphs.
- May struggle with webpages that have few links or are part of a tightly knit cluster of pages with circular links.

Key Differences from PageRank:

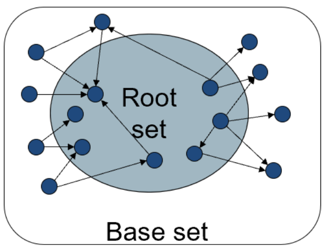
- PageRank focuses on the overall importance of a webpage based on its backlinks.
- HITS differentiates between hubs and authorities, providing more specific insights into a webpage's role within a topic.

SCHEME OF HITS

- Extract from the web a base set of pages that could be good hubs or authorities.
- From these, identify a small set of top hub and authority pages;
- iterative algorithm.

BASE SET

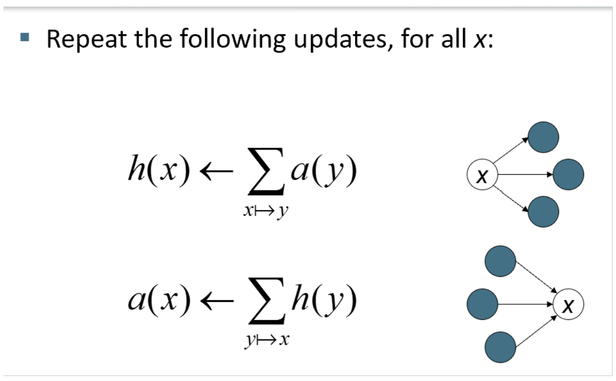
- Given text query (say browser), use a text index to get all pages containing browser.
  - Call this the root set of pages.
- Add in any page that either
  - points to a page in the root set, or
  - is pointed to by a page in the root set.
- Call this the base set.



Get in-links (and out-links) from a *connectivity server*

DISTILLING HUBS AND AUTHORITIES

- Compute, for each page x in the base set, a hub score h(x) and an authority score a(x).
- Initialize: for all x, h(x)1; a(x) 1;
- Iteratively update all h(x), a(x);
- After iterations
  - output pages with highest h() scores as top hubs
  - highest a() scores as top authorities.
-



- Repeat the following updates, for all *x*:

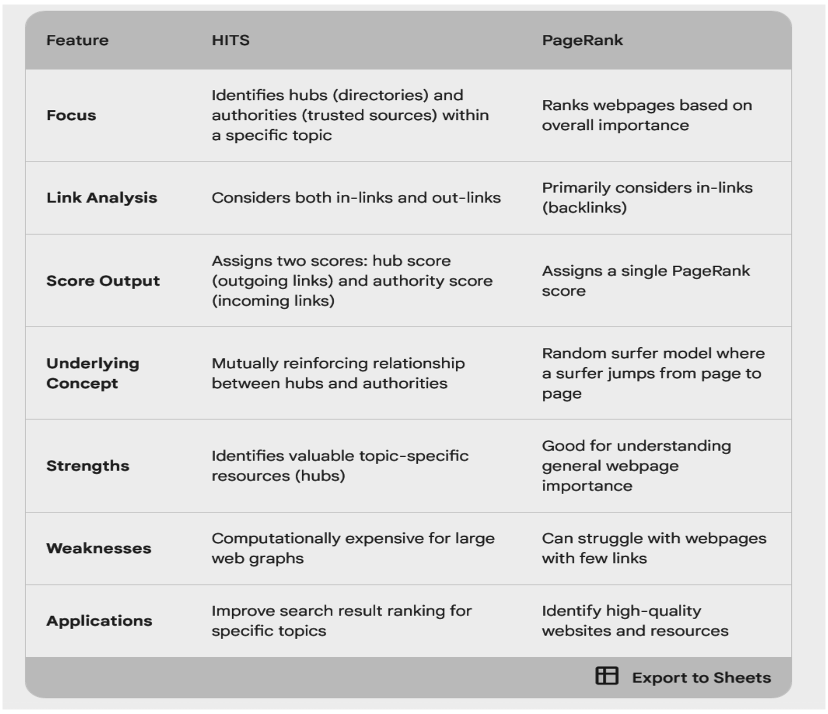
$$h(x) \leftarrow \sum_{x \mapsto y} a(y)$$

$$a(x) \leftarrow \sum_{y \mapsto x} h(y)$$

- **SCALING:**
  - To prevent the h() and a() values from getting too big, can scale down after each iteration.
  - Scaling factor doesn't really matter:
  - we only care about the relative values of the scores.
- HOW MANY ITERATIONS?
  - Claim: relative values of scores will converge after a few iterations:
    - in fact, suitably scaled, h() and a() scores settle into a steady state!
  - In practice, ~5 iterations get you close to stability.

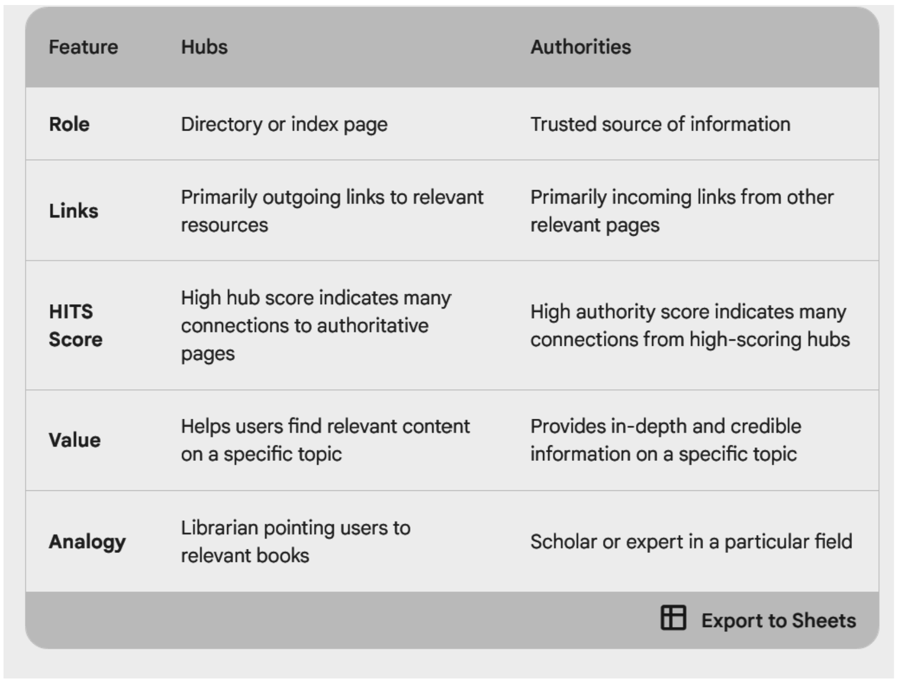
ISSUES IN HITS

- Topic Drift
  - Off-topic pages can cause off-topic "authorities" to be returned
    - E.g., the neighbourhood graph can be about a "super topic"
- Mutually Reinforcing Affiliates
  - Affiliated pages/sites can boost each others' scores
    - Linkage between affiliated pages is not a useful signal



Feature	HITS	PageRank
Focus	Identifies hubs (directories) and authorities (trusted sources) within a specific topic	Ranks webpages based on overall importance
Link Analysis	Considers both in-links and out-links	Primarily considers in-links (backlinks)
Score Output	Assigns two scores: hub score (outgoing links) and authority score (incoming links)	Assigns a single PageRank score
Underlying Concept	Mutually reinforcing relationship between hubs and authorities	Random surfer model where a surfer jumps from page to page
Strengths	Identifies valuable topic-specific resources (hubs)	Good for understanding general webpage importance
Weaknesses	Computationally expensive for large web graphs	Can struggle with webpages with few links
Applications	Improve search result ranking for specific topics	Identify high-quality websites and resources

Export to Sheets



Feature	Hubs	Authorities
Role	Directory or index page	Trusted source of information
Links	Primarily outgoing links to relevant resources	Primarily incoming links from other relevant pages
HITS Score	High hub score indicates many connections to authoritative pages	High authority score indicates many connections from high-scoring hubs
Value	Helps users find relevant content on a specific topic	Provides in-depth and credible information on a specific topic
Analogy	Librarian pointing users to relevant books	Scholar or expert in a particular field

WEB SEARCH BASICS

- The Web's decentralised and uncontrolled nature creates challenges for search engines compared to structured information retrieval systems.
- Early search engines focused on scaling to handle the vast amount of web content.
- Web page creation lacked editorial control, leading to diverse content quality and formats (languages, styles, structure).
- Search engines needed new techniques to rank results based on relevance and trustworthiness.
- Browsers and their features (ignoring unknown elements, user-friendly HTML viewing) contributed to the Web's growth.
- Search engines transitioned from keyword matching to ranking based on factors like authoritativeness.
- The size of the Web and how search engines measure it are complex issues.
- Static vs. Dynamic web pages: Static pages have consistent content, while dynamic pages are generated based on user queries.

SEARCH IN WEB

- Without search, content is hard to find.
- Without search, there is no incentive to create content.
  - Why publish something if nobody will read it?
  - Why publish something if I don't get ad revenue from it?
- Interest aggregation
  - Unique feature of the web: A small number of geographically dispersed people with similar interests can find each other
- Somebody needs to pay for the web.
  - Servers, web infrastructure, content creation
  - A large part today is paid by search ads.
- Web search engine issues:
  - Data is dynamic - ever changing
  - Quality is variable - upto the user to make judgement
  - Scope is HUGE

WEB GRAPH

- Web - giant network of interconnected pages
- Each page -> **node** in the graph

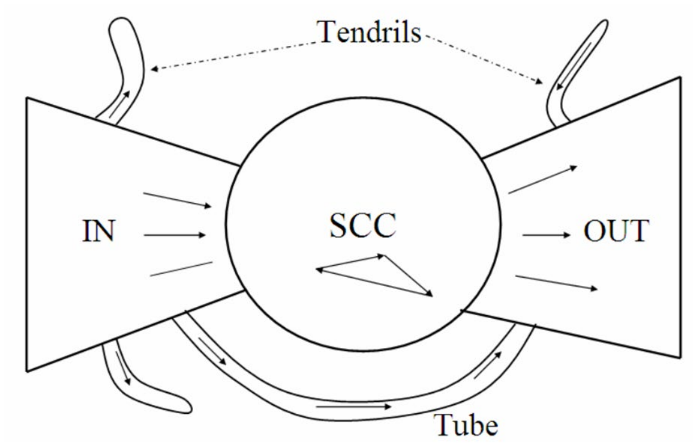
- Hyperlinks -> **directed edges,**show direction
- Properties
    - Not strongly connected : can't necessarily navigate from any page to any other page by just following links.
    - Each page -> two types of links:
        - IN Links : from other pages to this page
        - OUT links : from this to other
    - Num of in links: in degree of the page
    - Num of out links: out degree of the page
- the distribution of in-links is not random.
- follows a power law where most pages have few in-links, while a smaller number of pages have many in-links
    - some pages are more central and interconnected than others.

STRUCTURE OF THE WEB

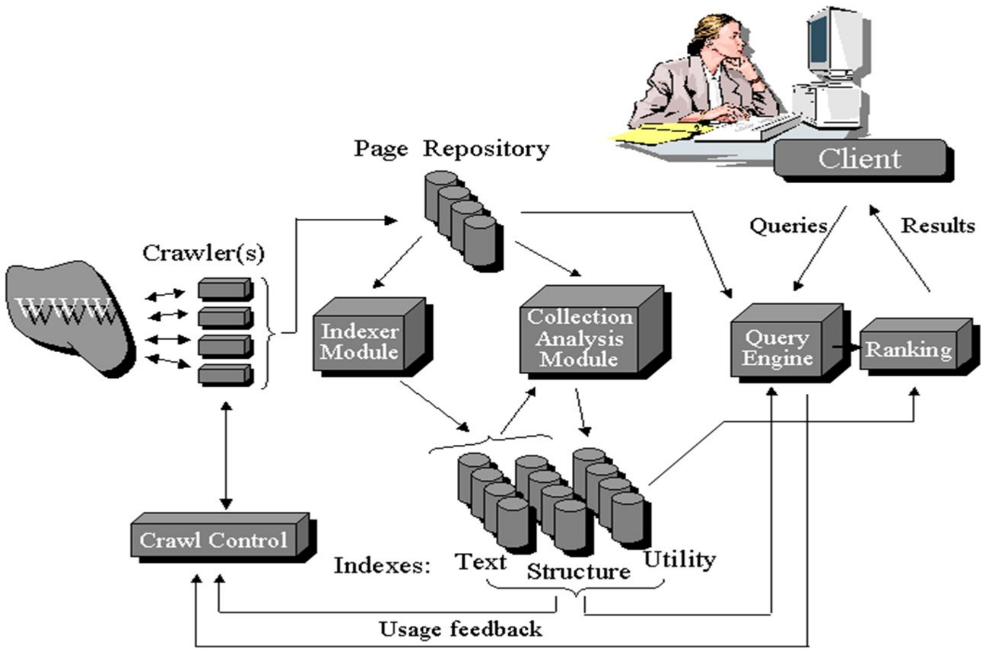
- The web can be visualized as having a bow-tie structure, which describes the distribution of web pages based on how they link to each other. Here's a breakdown:
- Components:
    - SCC - STRONGLY CONNECTED CORE
        - Central + most imp
        - Consists of web pages densely interconnect
        - Navigate between them using links
    - IN LINKS (IN)
        - Web pages that point to scc but not necessarily receive links back
        - Niche websites -> link to bigger platforms
    - OUT LINKS (OUT)
        - linked from the SCC but don't necessarily link back.
        - include forums, product pages, or specific articles that are referenced by major websites but don't have many outbound links themselves.

    - TENDRILS
        - smaller groups of web pages that branch out from the main "In" and "Out" sections of the bow-tie.
        - connect to and from the Strongly Connected Core (SCC) but are not directly part of it.
        - clusters of web pages that are somewhat related to the core topics of the SCC but have their own niche focus.
        - tendril branching from the "News" section of the SCC might connect various sports news websites.
    - TUBES
        - groups of web pages that can be reached from the "In" section and link to the "Out" section but are not part of either.
        - act as alternative pathways for navigating between the two peripheral areas of the bow-tie.
        - bridges that connect specific topics within the "In" and "Out" sections without directly interacting with the core.
        - travel blog that links to various airline websites ("Out") but gets referenced by general travel forums ("In").
- WHY BOWTIE?

    - Imagine a bow-tie with the knot as the SCC.
    - The "wings" on either side represent the In-Links and Out-Links.
    - Most web pages fall into these peripheral categories, with a smaller portion forming the tightly knit core.



Web Search Architecture

- 

- complex system with different parts working together to find information on the web
- Components:
    - Crawler:
        - Automated bots -> constantly traverse the web
        - Job -> discover new web pages + update search engines knowledge of existing ones
        - Follow links from known pages to find new ones
            - Expanding search engine's indexes
        - Programmed -> prioritise crawling fresh content / revisit pages based on update freq

        - Page Repository / Index
            - database where the information discovered by crawlers is stored.
            - It doesn't just store the entire webpage, but extracts and organises relevant information like text content, links, and metadata.
            - This allows the search engine to efficiently find web pages relevant to a user's query
            - Indexer Module:
                - takes the raw text extracted from web pages by crawlers and transforms it into a format suitable for efficient searching.
                - Does:
                    - Tokenization
                    - Stop word removal
                    - stemming/lemmatization
                    - Part of speech tagging
            - Collection analysis module
                - analyses the entire collection of indexed pages to identify patterns and improve search effectiveness.
                - uses/does:
                    - Term freq
                    - Idf
                    - Phrase detection
        - Query Engine
            - Algorithmic heart
            - Interface b/w search index, user and web
            - Two steps:
                - Retrieves results as per marching keywords
                - Ranking the web pages
            - factors like keyword matching, page authority, and user intent.
        - Search Interface
            - Look n feel of s.e
            - Allows user to submit queries
            - Browse result list
            - Click on chosen web page
            - User -> diff between sponsored and organic linksi

5 mark answer here

- **Crawler(s):** These are automated programs that constantly discover and fetch new web pages. They likely point to a module responsible for managing the crawling process.
- **Indexer:** This module takes the raw text extracted from web pages by crawlers and processes it for efficient searching (e.g., tokenization, stop word removal, stemming).
- **Analysis Module (Collection Analysis)**: This module analyses the entire collection of indexed pages to identify patterns and improve search effectiveness (e.g., term frequency, inverse document frequency).
- **Engine (Ranking Engine):** This module takes a user's search query and analyses the indexed pages to determine which ones are most relevant and should be displayed first in the search results. It likely considers factors like keyword matching, page authority, and user intent.
- **User Interface:** This is the search engine's interface where users enter their queries and receive results.

- **Queries:** This refers to the search terms or phrases users enter to find information.
- **Results:** These are the web pages the search engine retrieves and displays in response to a user's query, ranked by their relevance.
- **Text Structure Utility:** This likely refers to the data structures used within the index to organise the processed text for fast retrieval (e.g., inverted index, n-grams).
- **Usage Feedback:** This might represent user interactions or search logs that are fed back into the system to improve search algorithms and ranking over time.

flow of data through a web search engine:

- Crawlers discover and fetch web pages.
- The indexer processes the extracted text from these pages.
- The collection analysis module analyzes all indexed pages to understand content patterns.
- The ranking engine analyzes user queries and the indexed pages to determine relevant results.
- Users interact with the search engine interface to submit queries and receive results.
- User interactions might be used to further improve the search experience.

Advertising Models:

- **Branding Ads:** Aim to create a positive association with a company's brand (e.g., banner ads).
- **Transaction-Oriented Ads:** Encourage users to click and purchase something (e.g., early click-based web ads).
- **CPM (Cost Per Mille):** Pricing model where advertisers pay per thousand impressions (ad views).
- **CPC (Cost Per Click):** Pricing model where advertisers pay only when users click on their ad.

Early Search Advertising:

- **Goto (became Overture, acquired by Yahoo!):** A pioneer in search advertising, allowing companies to bid for ad placement on search queries.
- **Sponsored Search/Search Advertising:** Search results with paid placement based on advertiser bids.

Modern Search Engines:

- **Pure Search Results:** Results ranked by algorithms based on relevance to the user's query.
- **Algorithmic Search Results:** Another term for pure search results.
- **Sponsored Search Results:** Paid listings displayed separately from pure search results.

Search Engine Marketing (SEM): The process of understanding how search engines rank sponsored ads and allocating marketing budgets for them.

Click Spam: Artificial clicks on sponsored ads, often generated by bots, to disrupt the system or exhaust competitors' budgets.

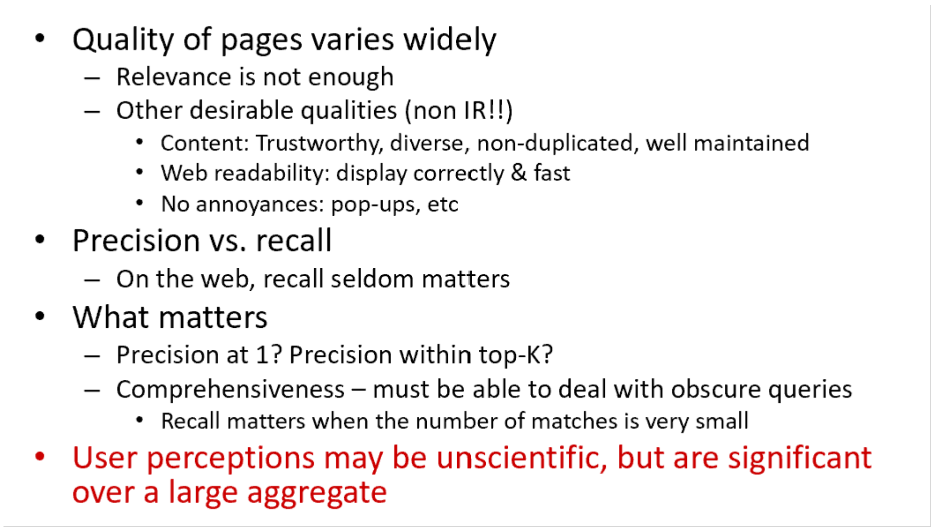
Other Important Terms:

- **User Intent:** The underlying goal or need behind a user's search query.
- **Bids:** The amount advertisers offer to pay for their ad to be displayed on a search query.

USER NEEDS

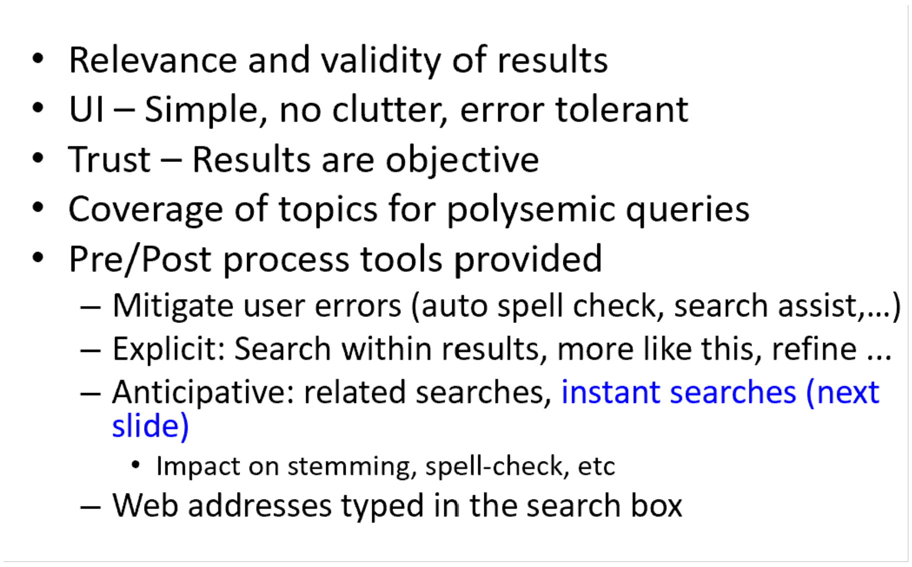
- Three broad categories:
  - Informational
    - General info about a broad topic
    - Typically not a single web page
    - Users try to assimilate info from multiple pages
  - Navigational
    - Seek website/home page of a single entity -> user has in mind
    - User expectation -> first page should be the web page they are looking for
    - Not interested in plethora of docs
  - Transactional
    - Prelude to the user performing a transaction on the web
    - Like -> purchasing a prod, downloading a file or making a reservation
    - Engine -> return results listing services that provide interfaces for such
  - Also gray areas - exploratory searches
- Query under which category? Difficult
- category not only governs the algorithmic search results, but the
- suitability of the query for sponsored search results
- For navigational queries, some have argued that
  the search engine should return only a single result or even the target web page directly

- Users empirical evaluation of results:



- Quality of pages varies widely
  - Relevance is not enough
  - Other desirable qualities (non IR!!)
    - Content: Trustworthy, diverse, non-duplicated, well maintained
    - Web readability: display correctly & fast
    - No annoyances: pop-ups, etc
- Precision vs. recall
  - On the web, recall seldom matters
- What matters
  - Precision at 1? Precision within top-K?
  - Comprehensiveness – must be able to deal with obscure queries
    - Recall matters when the number of matches is very small
- User perceptions may be unscientific, but are significant over a large aggregate

- Users empirical evaluation of engines



- Relevance and validity of results
- UI – Simple, no clutter, error tolerant
- Trust – Results are objective
- Coverage of topics for polysemic queries
- Pre/Post process tools provided
  - Mitigate user errors (auto spell check, search assist,…)
  - Explicit: Search within results, more like this, refine …
  - Anticipative: related searches, instant searches (next slide)
    - Impact on stemming, spell-check, etc
  - Web addresses typed in the search box

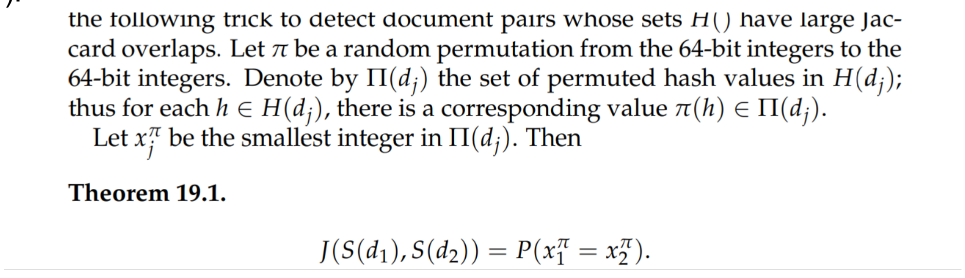
DUPLICATION IN THE WEB

- Duplication: web contains multiple copies of same content
- About 40% of content is duplicates
- Strict duplicate detection = exact match
  - Not as common
- But many, many cases of near duplicates
  - E.g., Last modified date the only difference between two copies of a page
- Search engines try to avoid indexing multiple copies of the same content,
- to keep down storage and processing overheads.
- SIMPLEST APPROACH TO DUPLICATES:
  - Fingerprints
    - Succinct 64 bit digest of characters on that page
    - If two fingerprints of web pages r equal
    - Test if pages are equal and one of them gets labelled a copy
    - DOES NOT HELP W NEAR DUPLICATION
- Duplication: exact match, can be detected w fingerprints
- Near-duplication: approx match
  - V few chars are diff

COMBATING NEAR DUPLICATION

- Done using smth known as *shingles / shingling*

  Given a positive integer k and a sequence of terms in a document d, define the k-shingles of d to be the set of all consecutive sequences of k terms in d
- As an example,
  - consider the following text: a rose is a rose is a rose. The 4-shingles for this text
    - (k = 4 is a typical value used in the detection of near-duplicate web pages)
  - are **a rose is a, rose is a rose and is a rose is**.
  - The first two of these shingles each occur twice in the text.
- Two documents are near duplicates if the sets of shingles generated from them are nearly the same.
- Method:
  - Let S(dj) denote the set of shingles of document dj
  - For two docs d1, d2 find jaccard coeff

- measures the degree of overlap between
- the sets S(d1) and S(d2) as
  - |S(d1) ∩ S(d2)|/|S(d1) ∪ S(d2)|;
- test for near duplication between d1 and d2 is to compute this Jaccard coefficient;
  - if it exceeds a preset threshold (say, 0.9), we
  - declare them near duplicates and eliminate one from indexing.
- BUT, computing jaccard coeffs pairwise -> lot of work
- To avoid this, use hashing
  - we map every shingle into a hash value over a large space, say 64 bits
  - For j = 1, 2, let H(dj) be the corresponding set of 64-bit hash values derived from S(dj
  -).
  - 

    the following trick to detect document pairs whose sets $H()$ have large Jaccard overlaps. Let π be a random permutation from the 64-bit integers to the 64-bit integers. Denote by $\Pi(d_j)$ the set of permuted hash values in $H(d_j)$; thus for each $h \in H(d_j)$, there is a corresponding value $\pi(h) \in \Pi(d_j)$.

    Let x_j^π be the smallest integer in $\Pi(d_j)$. Then

    Theorem 19.1.

    $$J(S(d_1), S(d_2)) = P(x_1^\pi = x_2^\pi).$$

- Locality Sensitive Hashing (LSH) Trick:
  - Permutation: Introduce a random permutation function (π) that takes a 64-bit hash value and scrambles it to another 64-bit value.
  - Smallest Permuted Value: For each webpage's set of permuted hash values (Π(dj)), we find the smallest value (x π j).
- This theorem is the key insight: The Jaccard coefficient between two webpages (d1 and d2) is equal to the probability that their smallest permuted hash values (x π 1 and x π 2) are the same.

SPAM in WEB

- Search engines became a valuable tool for connecting advertisers with potential buyers
- Search engines initially relied on term frequency to rank webpages.
  - This led to websites stuffing their content with keywords to appear higher in search results, creating a poor user experience.
- Then to avoid irritating users
  - sophisticated spammers resorted to such tricks as rendering these repeated terms in the same color as the background.
  - search engine indexer would parse the invisible words
- Cloaking involved showing different content to search engine crawlers (indexing robots) versus human users.
  - spammer's web server returns different pages depending on whether the http request comes from a web search engine's crawler
- A doorway page
  - contains text and metadata carefully chosen to rank highly on selected search keywords.

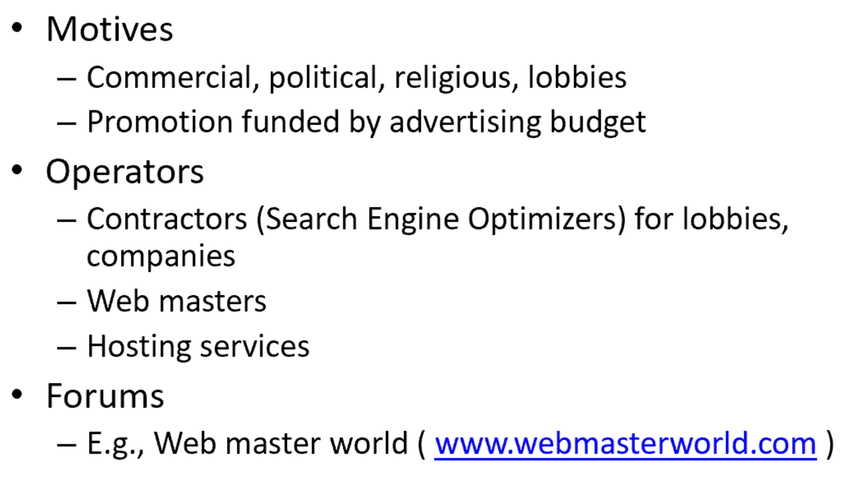
    - Browser requests the doorway page
        - it is redirected to a page containing content of a more commercial nature
    - How it Works:
    - Techniques can involve:
        - JavaScript: Delivering the relevant content to users with JavaScript, while crawlers see the keyword-stuffed version.
        - CSS: Hiding the keyword-stuffed content from users with CSS (Cascading Style Sheets) but leaving it visible to crawlers.
        - IP Address Detection: Showing different content based on the IP address of the visitor, assuming a search engine crawler has a different IP than a human user.
    - Unethical, Ineffective, Bad UX
- More complex spamming - manipulation of metadata

PAID INCLUSION

- Some search engines offered "paid inclusion," allowing websites to pay to be included in the index, raising fairness concerns.

SEO

- Spamming is inherently an economically motivated activity, there has sprung around it an industry of
    - Search Engine Optimizers, SEOs
    - to provide consultancy services for clients who seek to have their
    - web pages rank highly on selected keywords



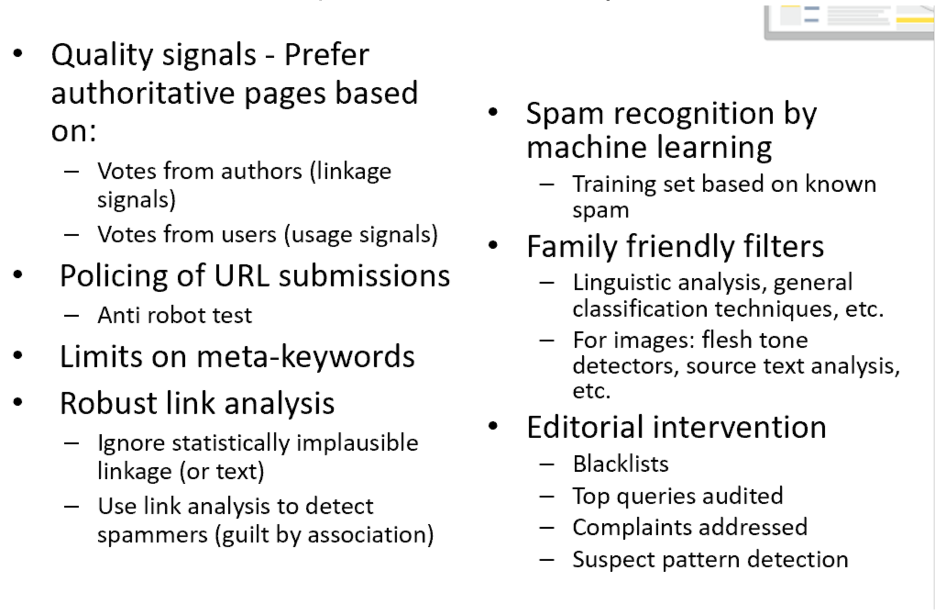
- Motives
    - Commercial, political, religious, lobbies
    - Promotion funded by advertising budget
- Operators
    - Contractors (Search Engine Optimizers) for lobbies, companies
    - Web masters
    - Hosting services
- Forums
    - E.g., Web master world (www.webmasterworld.com)

- Simplest seo:
    - Keyword Stuffing
        - First generation engines relied heavily on tf/idf
            - The top-ranked pages for the query maui resort were the ones containing the most maui's and resort's
            - SEOs -- dense repetitions of chosen terms
                - e.g., maui resort maui resort maui resort

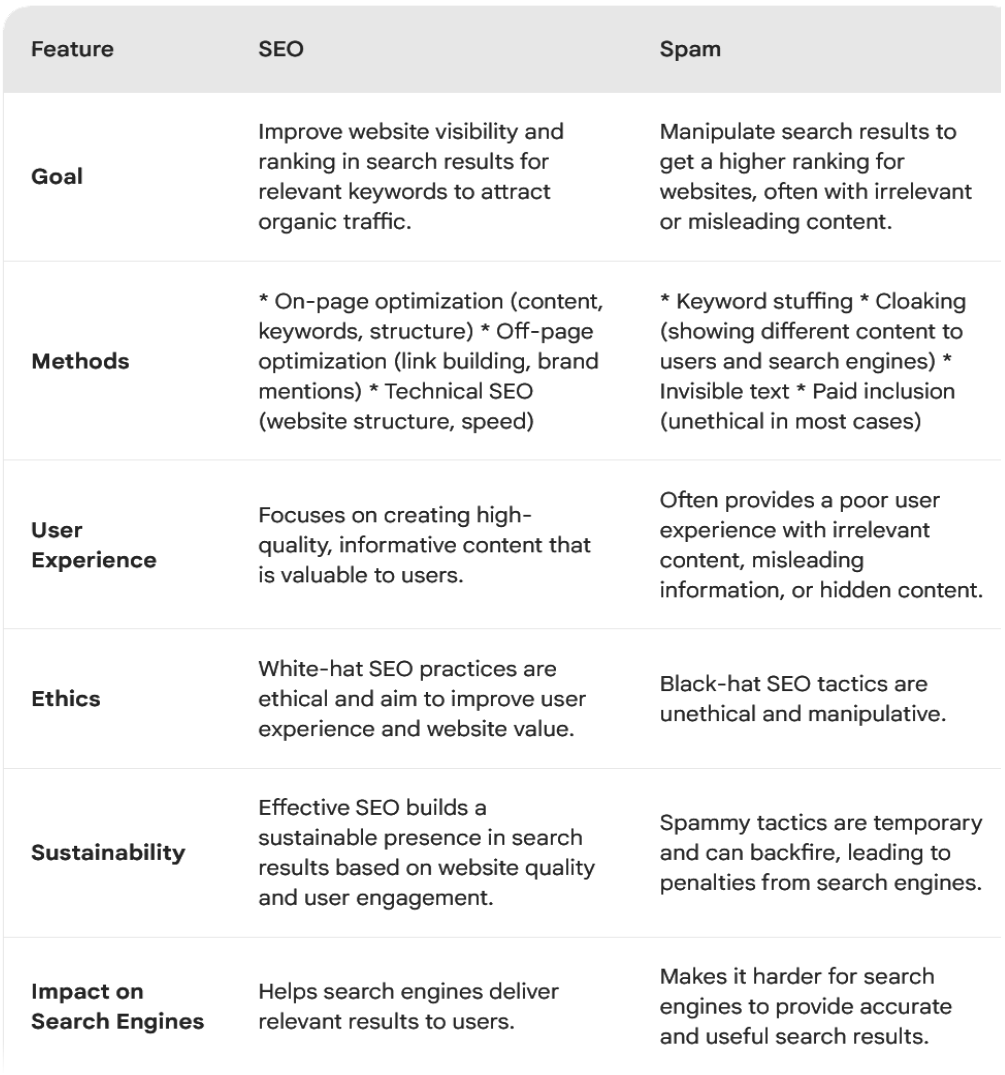
                - Often, the repetitions would be in the same color as the background of the web page
                - Repeated terms got indexed by crawlers
                - But not visible to humans on browsers
            - PURE WORD DENSITY IS NOT A GOOD IR SIGNAL
        - OTHER METHODS: CLOAKING, DOORWAY PAGES

FIGHTING SPAM

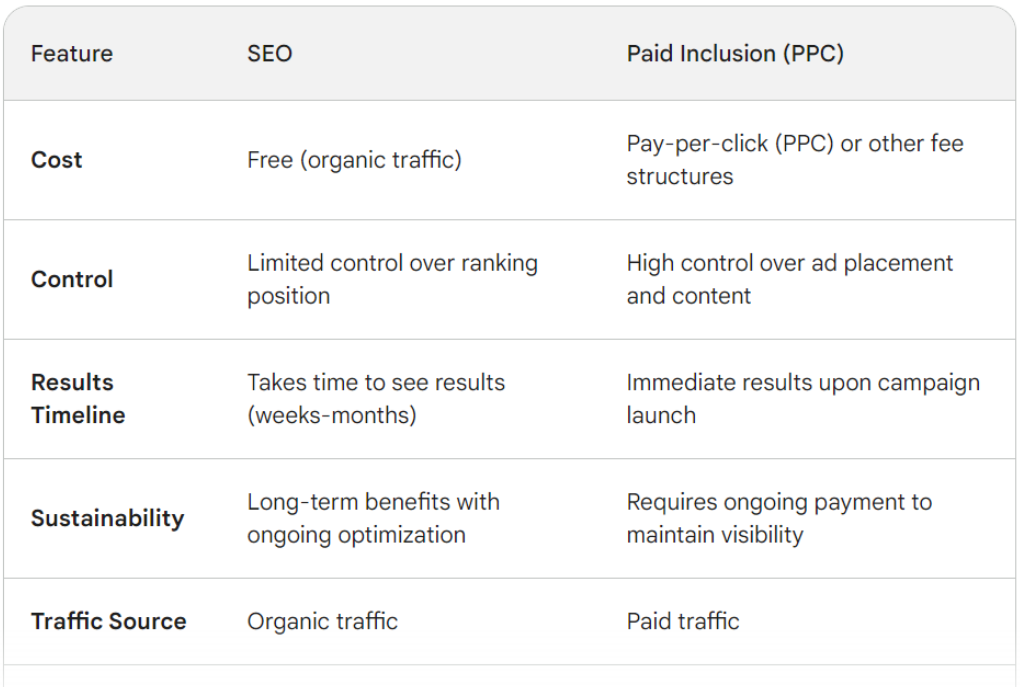
- Web search engines frown on this business of attempting to decipher and adapt to their proprietary ranking techniques and indeed announce policies on forms of SEO behaviour they do not tolerate
- To combat spammers who manipulate the text of their web pages is the exploitation of the link structure of the Web – a technique known as link analysis.



- Quality signals - Prefer authoritative pages based on:
  - Votes from authors (linkage signals)
  - Votes from users (usage signals)
- Policing of URL submissions
  - Anti robot test
- Limits on meta-keywords
- Robust link analysis
  - Ignore statistically implausible linkage (or text)
  - Use link analysis to detect spammers (guilt by association)
- Spam recognition by machine learning
  - Training set based on known spam
- Family friendly filters
  - Linguistic analysis, general classification techniques, etc.
  - For images: flesh tone detectors, source text analysis, etc.
- Editorial intervention
  - Blacklists
  - Top queries audited
  - Complaints addressed
  - Suspect pattern detection



Feature	SEO	Spam
Goal	Improve website visibility and ranking in search results for relevant keywords to attract organic traffic.	Manipulate search results to get a higher ranking for websites, often with irrelevant or misleading content.
Methods	* On-page optimization (content, keywords, structure) * Off-page optimization (link building, brand mentions) * Technical SEO (website structure, speed)	* Keyword stuffing * Cloaking (showing different content to users and search engines) * Invisible text * Paid inclusion (unethical in most cases)
User Experience	Focuses on creating high-quality, informative content that is valuable to users.	Often provides a poor user experience with irrelevant content, misleading information, or hidden content.
Ethics	White-hat SEO practices are ethical and aim to improve user experience and website value.	Black-hat SEO tactics are unethical and manipulative.
Sustainability	Effective SEO builds a sustainable presence in search results based on website quality and user engagement.	Spammy tactics are temporary and can backfire, leading to penalties from search engines.
Impact on Search Engines	Helps search engines deliver relevant results to users.	Makes it harder for search engines to provide accurate and useful search results.



Feature	SEO	Paid Inclusion (PPC)
Cost	Free (organic traffic)	Pay-per-click (PPC) or other fee structures
Control	Limited control over ranking position	High control over ad placement and content
Results Timeline	Takes time to see results (weeks-months)	Immediate results upon campaign launch
Sustainability	Long-term benefits with ongoing optimization	Requires ongoing payment to maintain visibility
Traffic Source	Organic traffic	Paid traffic

WEB CRAWLING

- Process by which we gather pages from the web,
- In order to index them + support search engine
- Objective ->
  - quickly and efficiently
  - gather as many useful web pages as possible,
  - together with the link structure that interconnects them.

Web Crawler / spider

- automated program that systematically browses the World Wide Web
- busy insects of the internet, constantly fetching and analysing information to keep search engines up-to-date
- FEATURES
  - **Must** provide:
    - Robustness:
      - Web contains servers that create spider traps,
      - which are generators of web pages that mislead crawlers into getting stuck fetching an infinite number of pages in a particular domain
      - Not all traps are malicious some r just faulty dev
      - Crawler must be resilient to such traps
    - Politeness:
      - Servers have explicit and implicit policies regulating the rate at which a crawler can visit them
      - Crawler must respect these
  - **Should** provide:
    - Distributed:
      - should have the ability to execute in a distributed fashion across multiple machines.
    - Scalable:
      - should permit scaling up the crawl rate by adding extra machines and bandwidth.
    - Performance and efficiency:
      - The crawl system should make efficient use of various system resources including processor, storage and network bandwidth.
    - Quality:
      - Given that a significant fraction of all web pages are of poor utility for serving user query needs, the crawler should be biassed towards fetching "useful" pages first.
    - Freshness:
      - In many applications, the crawler should operate in continuous mode: it should obtain fresh copies of previously fetched pages. A search engine crawler, for instance, can thus ensure that the search engine's index contains a fairly current representation of each indexed web page.
      - For such continuous crawling, a crawler should be able to crawl a page with a frequency that approximates the rate of change of that page.
    - Extensible:
      - Crawlers should be designed to be extensible in many ways –

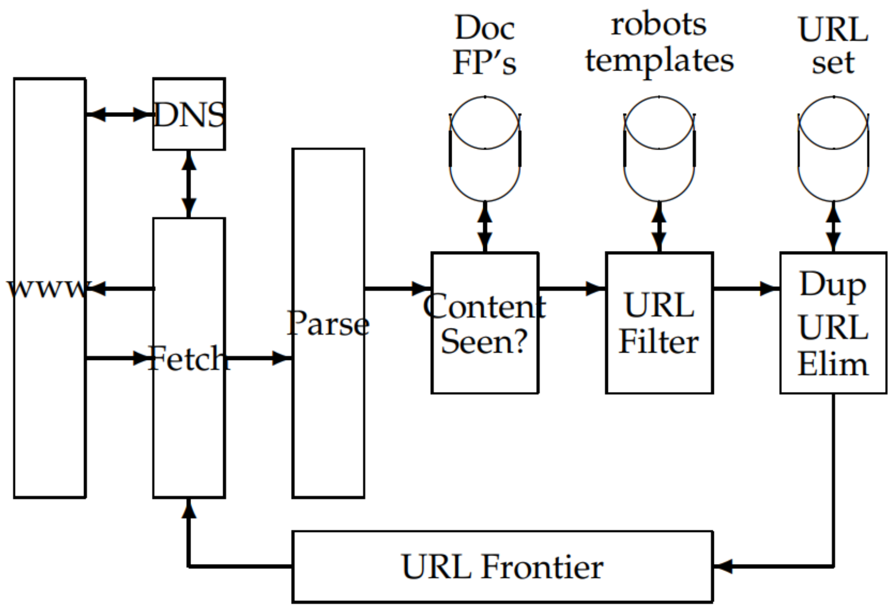
to cope with new data formats, new fetch protocols, and so on
This demands that the crawler architecture be modular.

CRAWLING

- Basic operation of any crawler:
    - Begins with one or more URLS -> seed set
    - Picks Url from this set
    - Fetches web page at this URL
    - Fetched page => parsed
        - Text from page is extracted
            - Text extracted is fed to a text indexer
        - Links from page are extracted
            - Links extracted are added to a URL FRONTIER
    - Url frontier -> consists of urls whose pages are yet to be fetched
    - Initially, url frontier has the seed set
    - Pages are fetched =>
        - corresponding urls are deleted from the frontier
        - New extracted links are added
    - entire process may be viewed as traversing the web graph
    - In continuous crawling, the URL of a fetched page is added back to the frontier for fetching again in the future.
- Basic properties needed for a professional crawler:
    - Only one connection should be open to give host at a time
    - A waiting time of few seconds should occur between successive requests to a host
    - Politeness should be obeyed

CRAWLER Architecture

- SEVERAL MODULES:
    - URL FRONTIER
        - containing URLs yet to be fetched in the current crawl
        - (in the case of continuous crawling, a URL may have been fetched previously but is back in the frontier for re-fetching).
    - DNS RESOLUTION
        - module that determines the web server from which to fetch the page specified by a URL
    - Fetch module
        - Uses http protocol -> retrieve web page at a URL
    - Parsing Module
        - Extracts text and set of links from a fetched web page
    - Duplicate elimination module
        - Determines whether an extracted link is alr in url frontier
        - Or has recently been fetched



LIFE CYCLE OF WEB CRAWLER THREAD

- steps a web crawler thread takes to process a URL and discover new links

1. Fetching the Webpage:
    a. The crawler starts by picking a URL from a queue called the "frontier."
    b. It uses the HTTP protocol to download the content of the webpage at that URL.
    c. The downloaded content is stored temporarily - IN TEMPORARY STORE
2. Extracting Information:
    a. The downloaded webpage is parsed to extract:
        i. Text content, including any HTML tags (e.g., bold text).
        ii. Links embedded within the webpage.
    b. The extracted text is sent to the indexer for further processing (indexing keywords etc.).
    c. Link information, including the surrounding text (anchor text), is also sent to the indexer for ranking purposes
3. Link Validation and Filtering:
    a. Each extracted link goes through various checks before being added to the frontier:
    b. Duplicate Content Check:
        i. The crawler verifies if a webpage with identical content has already been seen (using a checksum or more sophisticated methods like shingles).

    c. URL Filter:
        i. The URL is checked against pre-defined rules to exclude specific domains (e.g., all .com websites) or include only certain ones.
    d. Robots Exclusion Protocol (robots.txt):

            i. Websites can specify areas off-limits to crawlers through a robots.txt file. The crawler checks this file to ensure it's allowed to crawl the extracted link.
    e. the trade-off between efficiency (using a cache to avoid re-fetching robots.txt) and politeness (considering the robots.txt file might change between adding the URL to the frontier and fetching it). Ultimately, checking robots.txt before fetching is considered more polite.

4. URL Normalisation:
    a. Relative links (specifying a link within the same website) are converted into absolute URLs (including the full domain name) for proper processing.
5. Duplicate Elimination:
    a. The crawler checks if the extracted URL already exists in the frontier or has already been crawled (for non-continuous crawls). If so, it's not added again.
6. Adding to the Frontier:
    a. If a URL passes all the checks, it's assigned a priority and added to the frontier queue. This priority determines the order in which URLs are fetched next
7. Housekeeping Tasks:
    a. A separate thread periodically performs housekeeping tasks:
        i. Logging crawl progress (URLs crawled, frontier size).
        ii. Deciding when to stop crawling.
        iii. Checkpointing (saving the crawler's state, like the frontier queue) for fault tolerance in case of crashes.

Distributed Crawling:

- What it is: Distributing web crawlers across multiple machines (nodes) for efficiency and targeting specific regions.
- Benefits:
    - Scalability: Handle larger workloads by adding more machines.
    - Geographical Targeting: Focus crawling efforts on specific regions (not guaranteed due to internet routing).
- System Setup:
    - Multiple machines working together.
    - Each machine can run multiple crawler threads.
- URL Partitioning:
    - Assigning crawling responsibilities for websites to different nodes.
    - Doesn't necessarily follow geographic location due to internet complexities.
- Crawling Process per Node:
    - Each node fetches, parses, and extracts information from webpages.
    - A "host splitter" identifies the website's domain name (host) from URLs.
    - Based on partitioning, the splitter assigns URLs to the responsible node for that host.
- Sharing Duplicate URL Information:
    - All nodes share information to avoid crawling the same URL multiple times.
    - Challenges in Content Detection:
    - Distributing fingerprints (used to detect similar content) across nodes is difficult.
    - Same content on different servers breaks hostname-based partitioning.
    - Frequent communication between nodes is needed, reducing efficiency.
- Fingerprint Caching:
    - Caching fingerprints isn't effective due to constantly changing content.
- Handling Content Changes (Continuous Crawling):

    - Outdated fingerprints need to be removed for repeated crawling.
    - Solution: Store fingerprint/shingle with URL in each node's URL frontier.

NORMALISATION

In web crawlers, normalisation is like tidying up URLs. It changes them into a standard format so the crawler treats similar URLs the same way. This avoids wasting time on duplicates and ensures accurate indexing for search engines. Imagine cleaning up your room - everything has its place and is easier to find!

DNS Lookup

The Problem: Web crawlers need to translate website names (URLs) into IP addresses to access them. This process, called DNS lookup, can be slow.

Why is it Slow?

- **Distributed DNS:** DNS requests might involve multiple servers, causing delays.
- **Synchronous Resolution:** Standard libraries wait for each DNS response, blocking the crawler thread.

Solutions:

- **DNS Caching:** Store recently looked up URLs and IP addresses to avoid repeated DNS requests.
- **Asynchronous Resolution:** Crawler sends a request and continues working while waiting for a response (up to a timeout).
    - Retries with increasing wait times for slow lookups.

Benefits:

- Reduced waiting time for crawlers.
- More efficient crawling by utilising crawler threads effectively.

URL Frontier

- The URL frontier acts like a to-do list for a web crawler, holding URLs to be visited. Here's a simplified breakdown:
- What it Stores: Unvisited URLs discovered during crawling.
- URL Fronteering: The strategies to manage and prioritize these URLs.
- Goal: Efficient crawling by fetching important pages first and respecting website limits.

Key Components:

Prioritization (Front Queues):

- URLs are assigned priority scores based on factors like change rate (frequently updated pages get priority).
- Multiple queues exist, with higher-priority queues holding URLs at the front for faster crawling.

Politeness (Back Queues):

- Separate queues exist for each website to avoid overwhelming them with requests.
- A heap tracks the "cool-down" time for each website (how long to wait before fetching more URLs).
- Crawlers wait until a website's cool-down period is over before fetching more URLs from its queue.

How it Works:

- Crawler adds discovered URLs to the frontier.

- The frontier assigns a priority score and places the URL in the appropriate front queue.
- Crawler thread requests a URL from the frontier.
- The frontier checks the back queue for the website associated with the earliest cool-down time.
- If the cool-down period is over, the URL at the front of that back queue is retrieved for crawling.
- If the back queue is empty, the frontier refills it from a high-priority front queue, respecting website cool-down times.

Benefits:

- Prioritises important pages for efficient crawling.
- Avoids overloading websites by respecting their limits.
- Ensures crawlers work efficiently by keeping them busy with available URLs.

Distributing Search Indexes

Storing a giant search engine index on one machine is slow. Here's how to distribute it across multiple machines for speed:

Two Partitioning Strategies:

Term Partitioning:

- Splits dictionary terms among machines.
- Each machine stores a subset of terms and their document locations.
- Good for: Concurrent queries targeting different terms.

Challenges:

- Slow merging of long posting lists for multi-word queries.
- Difficult workload balancing due to changing term frequency.
- Complex dynamic indexing (adding new documents).

Document Partitioning:

- Each machine stores the entire index for a subset of documents.
- All queries are broadcast to all machines, which return their results.
- Good for: Less communication overhead compared to term partitioning.

Challenges:

- Requires calculating global document statistics (e.g., idf) across machines for ranking.
- Common Choice: Document partitioning due to its simplicity.

Document Partitioning Techniques:

- Host-Based: Group all pages from a website (host) on one machine (might lead to uneven workloads).
- URL Hashing: Distribute documents based on a hash of their URL for a more balanced workload.
- Score-Based: Create separate indexes for high-scoring (important) and low-scoring documents. Only search low-scoring ones if high-scoring results are insufficient.

Connectivity Servers

Search engines need to track links between webpages to understand website relationships and improve search results. But storing all this information directly would be massive!

The Problem: Efficiently answering queries about links between web pages without using too much memory.

Challenges:

- Storing all link data directly requires too much space (e.g., 32 TB for a 4 billion-page web).
- Balancing data compression for storage with fast retrieval of link information.

Solution: Connectivity Servers with Compressed Link Representation

- Unique IDs: Each webpage gets a unique ID for efficient storage and retrieval.
- Inverted Index-like Structure:
- A table stores link information similar to an inverted index for keywords.
- Each row represents a webpage with lists of linked-to pages (out-links) and pages linking to it (in-links) using IDs.

Compression Techniques:

- Similarity: Similar link structures across pages are addressed by referencing a "prototype" row with common links and specifying only differences for each similar page.
- Locality: Links often point to nearby pages (same website). Smaller integers are used for these local links, saving space.
- Gap Encoding: Instead of storing absolute positions of linked pages, the difference (gap) from the previous entry is stored, reducing redundancy.

Benefits:

- Reduces storage requirements significantly (e.g., down to 3 bits per link).
- Enables efficient retrieval of link information using connectivity servers.