Note 5 - Inverted index

DS behind modern search engines

- Match query with list of documents
- Searching document against query vs query against document
- every document is a bag of words
- d1={w1,w2,...} d2={w2,w3,...}
- traverse perpendicularly
- locally parse document into bag of words into a tuple <termID,docID,count>.
- first by doc ID then term ID. descending order of terms , then doc ID will be in descending order
- sort by document, merge sort by word hash, then do global merge sort per machine

Compression & Why we sort the documents

- We want to encode effeciently. For example or most frequent words
- When we have: w1->{d1,d2,...} we can not use documentID and use small integers (gaps) so... w1->{d1,+1,+1,...}
- Benefits of compression: Save storage space, increase cache effeciency, improve disk-mem transfer
- Target: Postings file
- In pratice: We get both time and space complexity

Search via inverted index

1. Query processing: Tokenize query, normalize, stem it, remove stop words ,Represent as bag of words to match against your dictionary
2. Procedure: Look at some terms and their IDs, move pointers to find matching documents, merge sort from before steps will allow you to scan through it linearly with respect to the length of the postings. List won't be long so intersection

3. Speed up? Start with word that has lower frequency. Because once we finish traversing that word we can stop
4. Phrases? In inverted index tuple instead of count store positions

Spelling Correction

- Tolerate mispelled queries
- Principles: Choose nearest then most common/popular
- Edit distance? Too costly