Keyword Search

- Traditional B+index is good for answering 1-dimensional range or point query
- Q: What about keyword search? Geo-spatial queries?
  - Q: Documents on "Computer Science"?
  - Q: Nearby coffee shops?
- Keyword, geo-spatial queries are getting more important for Web and mobile applications

Information Retrieval (IR)

- Q: Given a set of keywords how can we return "relevant documents" quickly?

- Q: What does "relevant document" mean?

  - Fundamentally, IR problem is subjective, whose effectiveness can be measured only through human judgement

Boolean model

- User wants *all documents that contain the word*
  - Simplest interpretation of "relevant documents"
- Q: Given this model, what does the system have to do?

- Q: How can the system identify the documents that contain the keywords quickly?
  - With no preprocessing or index: string matching algorithm
    - Knuth-Morris-Pratt algorithm
    - Boyce-Moore algorithm
  - Prebuilt indexes:
    - Suffix tree

    * Suffix array
    * Inverted index

Inverted index

- *Inverted index*: An index that stores the mapping from keyword to docid
  - Inverted index diagram
    * Lexicon/Dictionary -> Posting list (or inverted list)

- Q: How can we use inverted index to answer "UCLA Computer Science"?

- Inverted index construction algorithm
  - Q: Given document collection C, how can we build an inverted index?
  - Algorithm

```
Input:  C - set of documents
Output: DIC - dictionary of inverted index
        PL(w) - posting list for word w

For each document d in C:
    Extract all words in content(d) into W
    For each w in W:
        If w in DIC, then add w to DIC
        Append id(d) to PL(w)
```

- Size estimation of inverted index
  - Example: 100M docs, 10KB/doc, 1000 unique words/doc, 10 bytes/word, 4 byte/docid
  - Q: Document collection size?

  - Q: Inverted index size?
    * Q: total # docids in posting lists?
    * Q: # words in dictionary?
    * Q: Between dictionary and posting lists, which one is larger?
  - Rule of thumb: inverted index size is roughly equivalent to corpus size

Vector model

- Problem of boolean model
  - When document collection is large, there may be too many documents matching a set of keywords. Are they equally good? How can we help users to identify that are more likely to be relevant?
- Matrix representation of the Boolean model
  - Q: But, "Princeton" and "University", are they really equally important?
- Basic idea of vector model
  - Instead of 0 and 1 for terms, give real-valued weight for each term depending on how important the term is for the document
- Q: How do we know whether a term is "important" for a document?

- Basic idea of tf-idf weighting (or tf-idf vector)
  - A term t is important for document D
    - If t appears many times in D or
    - If t is a rare term
  - *tf*: term frequency: # occurrence of t in D
  - *idf*: inverse document frequency: # documents containing t
    - More precisely, N: total # documents, n: # documents with t, f: # occurrence of t in D
    - tf.idf = f x log (N/n)
  - tf.idf weight is high if the term is rare, but appears often in the document
- *Cosine similarity*
  - Represent the query using the same tf.idf vector
    - Similarly to documents, a query term is "important" if it is a rare term
  - Take inner product of the two vectors to compute "similarity" of documents to query
    - Cosine similarity will be high only if query and documents share many rare terms
  - Example
    - idf(Princeton)=1,
    - idf(univ)=idf(car)=idf(racing)=idf(admission)=0.1
    - Q = (Princeton, university)
    - D1 = (car, racing). D1.Q = ?
    - D2 = (Princeton, admission) D2.Q = ?
    - D3 = (university, admission) D3.Q = ?
  - Sort documents by the cosine similarity value
- Q: How can we compute cosine similarity quickly?
  - Q: What additional information should we include in inverted index?

  - Inverted index for cosine similarity

Keyword search in MongoDB

- Creating inverted index

```
db.collection.createIndex({ "title": "text" })
```

- Performing keyword search

```
db.collection.find({ $text: { $search: "computer science" }})
```