Web Search: what’s different?

- Extracting content from web-pages
  - synonymy in XML tags
  - tags embedded in content
  - Document Object Model
  - Finn’s plateau method
- Detecting duplicated content
- Link structure: PageRank and HITS
- Amount of data and query types

Extracting the Content

- Most information online is some form of XML
  - `<tag>` for content
  - `<description>` for content
  - Tag = “variable name”, Content = “its value”
  - Tag = “annotation” over a span of text
  - clarify what the content means (semantics)
  - different from HTML (tag = how text looks)
- XML parsers widely available

Tags Across Sources

- Tags and formats will vary across sources
- Need a separate parser for every source
- Sometimes parsing XML is not enough

Extracting Content from Web-pages

- No semantic tags
  - Tags = how the page looks
  - Highly varied across sources
- What if we remove all tags?

Extracting Content: DOM

- Use DOM structure
  - document object model
  - logical layout of page
- Explicit markup
  - some sites will tag content
  - rare and site-specific
  - breaks with new webmaster
- Or learn statistically
  - brittle, needs re-training

Extracting Content: Finn’s method

- Idea: text portion contains fewer HTML tags
  - pict #tags vs. #tokens
  - look for a plateau
  - formally: `page = x_{tags} + x_{non-tag}`
    - `x_{tags} = 1` a tag
    - `0 ... non-tag`
    - find a, b that maximize
  - `O(n)` approach

Finn’s method: implementation

- Naïve: `O(n^2)` ... hours per page
  - `O(n^2)` approach:
  - `O(n)` approach:
  - `O(n)` approach:
Web Search: what's different?

- Extracting content from web-pages
- Detecting duplicated content
  - exact duplicate detection: Adler32
  - near-duplicate detection: SimHash
- Statistical error bounds for LSH
- Link structure: PageRank and HITS
- Amount of data and query types

Detecting duplicates

- About 30% of web-pages are duplicates (2003)
  - mirrors, plagiarism, spam
  - also happens in news-feeds, speeches, essays
- Important to detect and remove
  - exact duplicate:
    - the content is identical bit-for-bit
    - non-content elements may differ
  - near-duplicate:
    - very similar but not exact
      - e.g. 90% of content is the same
    - reworded a few sentences, or used different keywords

Detecting duplicates Naively

- Compare every document against all others

  for b = 1...m:
    for a = 1...n:
      if dist(a,b) == 0: b is a duplicate of a
      return b is not a duplicate

- Computational complexity: O(n^2m) impractical
  - n = number of documents in a set (10^6 - 10^8)
  - d = size of each document (10^3 words / 10^4 bytes)

Duplicates via Fingerprinting

- Idea: make duplicates fall into same bucket
- For each new document:
  - compute its fingerprint
    - large number, which is
      - always same for identical docs
      - never same for two different docs
    - hash (fingerprint)
  - collision → test if duplicate
- Complexity: O(nd)

Fingerprinting: Adler32

- Generate a unique fingerprint for each doc
  - change 1 bit in a doc → get different print
  - maintain low chance of false collision
  - remove non-content (punctuation, tags)
  - add up byte values modulo a prime
- Designed for data integrity applications:
  - "magnify" small changes in input
    - transmission error, bad bit in RAM, bad sector on disk

Intuition for Adler32 / MD5 / etc.

- Data = sequence of "units" (bytes, words, ...)
  - each unit fills a subset of bits: 0 ≤ 1
  - bits scattered over fingerprint, depend on unit
  - flips depend on surrounding bits (so "UU" ≠ "UU")
- P(False Negative) = 0
  - identical units ➔ identical fingerprint ➔ identical result
- P(False Positive) = P’ (different inputs ➔ same fingerprint)
  - different unit → k flipped bits
  - FP = P (other units flip exactly those k bits back to original)
- Best possible case: completely random (but deterministic)
  - each of the 2^m fingerprints equally likely given input
  - Bad fit for near duplicates: "magnifies" small changes

Detecting near-duplicates

- Locality-sensitive hashing (LSH)
  - similar documents ➔ similar hash-code
  - For each document d:
    - generate K-bit hash-code
    - insert document into hash-table
    - collision ➔ possible duplicate
    - compare to docs in same bucket
  - Can miss near-duplicates:
    - similar hash-code = equal
    - repeat L times w. different hash-tables (randomised)

Locality Sensitive Hashing

- Locality sensitive hashing (LSH)
  - similar documents ➔ similar hash-code
  - For each document d:
    - generate K-bit hash-code
    - insert document into hash-table
    - collision ➔ possible duplicate
    - compare to docs in same bucket
  - Can miss near-duplicates:
    - similar hash-code = equal
    - repeat L times w. different hash-tables (randomised)

Computational cost: O(qk x 2^clogn)

1. Build a list of K random binary vectors: h_1, h_2,..., h_K
2. Hash code: h = h_k(a) XOR h_k(b)
3. For each vector h_k, apply L rounds:
   - Compute: h_k(a) XOR h_k(b)
   - Use vectors that are similar in k bits

LSH Error Rate

False Negative Error (we want a,b to collide, but they don't)

- P(0 of the L hashcodes for a,b match)
- (P(0) = L.P) (chance for a, b to match)
- (1 - P(0)) (chance for a, b to collide)
- (1 - P(0)) (chance for a, b to collide)

False Positive Error (we want a,b not to collide, but they do)

- P(1 of the L hashcodes for a,b match)
- (1 - (1 - q)^L)
- (1 - (1 - q)^L)
- (1 - (1 - q)^L)
- (1 - (1 - q)^L)
Simhash algorithm

- A very efficient variant of LSH
  - fast & space-efficient (no need to store hyperplanes)
  - re-created on-the-fly, only for the words in the document
  - slightly worse error bounds (sampling bias)

- Simhash fingerprint:
  - tokenize, remove stop-words, (sometimes stem)
  - assign weights to words (frequencies or TF.IDF)
  - compute a unique b-bit binary hash for every word
  - convert 0 → -1 and multiply by word weight
  - add by columns, set to 1 if Σ>0, 0 otherwise

Summary

- Content extraction: DOM and Finn's method
- Duplicates: spam, plagiarism, 30% of the web
- Exact duplicates: byte-for-byte identical
  - $O(n^2)$ approach: compare every A to every B
  - $O(n)$ fingerprint → hash, compare collisions
  - Adler32, MD5, etc: very sensitive to small changes
- Near duplicates: several words / sentences changed
  - simhash: locality-sensitive fingerprint
    - more words changed → more bits in fingerprint change
    - K, L affect chance of detection ±/ insertion / deletion
    - repeat L times with different K-bit fingerprints

Tropical fish include fish found in tropical environments around the world, including both freshwater and saltwater species.