Efficient Implementation of Postings Lists
Inverted Indices

Query “Brutus” AND “Calpurnia”

| Brutus | → | 1 | 2 | 4 | 11 | 31 | 45 | 173 | 174 |
| Caesar | → | 1 | 2 | 4 | 5  | 6  | 16 | 57  | 132 | ...
| Calpurnia | → | 2 | 31 | 54 | 101 |

Dictionary | Postings
Skip Pointers

Brutus
16
2 4 8 16 19 23 28 43

Caesar
5 51 98
1 2 3 5 8 41 51 60 71
Using Skip Pointers

\textbf{IntersectWithSkips}(p_1, p_2)

1 \hspace{1em} answer \leftarrow \langle \rangle \\
2 \hspace{1em} \textbf{while} \ p_1 \neq \text{NIL} \ \text{and} \ p_2 \neq \text{NIL} \\
3 \hspace{1em} \textbf{do if} \ \text{docID}(p_1) = \text{docID}(p_2) \\
4 \hspace{1.5em} \textbf{then} \ \text{ADD}(\text{answer}, \ \text{docID}(p_1)) \\
5 \hspace{1.5em} p_1 \leftarrow \text{next}(p_1) \\
6 \hspace{1.5em} p_2 \leftarrow \text{next}(p_2) \\
7 \hspace{1em} \textbf{else if} \ \text{docID}(p_1) < \text{docID}(p_2) \\
8 \hspace{1.5em} \textbf{then if} \ \text{hasSkip}(p_1) \ \text{and} \ (\text{docID}(\text{skip}(p_1)) \leq \text{docID}(p_2)) \\
9 \hspace{1.5em} \hspace{1em} \textbf{then while} \ \text{hasSkip}(p_1) \ \text{and} \ (\text{docID}(\text{skip}(p_1)) \leq \text{docID}(p_2)) \\
10 \hspace{1.5em} \hspace{1.5em} \hspace{1em} \text{do} \ p_1 \leftarrow \text{skip}(p_1) \\
11 \hspace{1.5em} \hspace{1.5em} \hspace{1em} \text{else} \ p_1 \leftarrow \text{next}(p_1) \\
12 \hspace{1.5em} \textbf{else if} \ \text{hasSkip}(p_2) \ \text{and} \ (\text{docID}(\text{skip}(p_2)) \leq \text{docID}(p_1)) \\
13 \hspace{1.5em} \hspace{1em} \textbf{then while} \ \text{hasSkip}(p_2) \ \text{and} \ (\text{docID}(\text{skip}(p_2)) \leq \text{docID}(p_1)) \\
14 \hspace{1.5em} \hspace{1.5em} \hspace{1em} \text{do} \ p_2 \leftarrow \text{skip}(p_2) \\
15 \hspace{1.5em} \hspace{1.5em} \hspace{1em} \text{else} \ p_2 \leftarrow \text{next}(p_2) \\
16 \hspace{1em} \textbf{return} \ \text{answer}
Efficiency of Using Skip Pointers

• What is the best case?
  – W1: 2(40) → 5 → 8 → 40(76) → 42 → 65 → 76
  – W2: 40(120) → 85 → 100 → 120

• What is the worst case?
  – W1: 1(7) → 3 → 5 → 7(13) → 9 → 11 → 13
  – W2: 2(8) → 4 → 6 → 8(14) → 10 → 12 → 14

• Can you propose an idea to make use of skip pointers more aggressively?
  – Then, what is the cost associated with the aggressive usage?
Where Do We Place Skip Pointers?

- More skip pointers $\rightarrow$ shorter skip spans $\rightarrow$ more likely to skip $\rightarrow$ many comparisons to skip pointers $\rightarrow$ much space storing skip pointers
- Fewer skip points $\rightarrow$ fewer pointer comparison $\rightarrow$ long skip spans $\rightarrow$ fewer opportunities to skip
- A heuristic: for a postings list of length $P$, use $\sqrt{P}$ evenly spaced skip pointers
  - Can you propose an idea to be aware of the distribution of the postings?
Phrase Queries

• Query “Stanford University”
  – Using singleton terms, “The inventor Stanford Ovshinsky never went to university” is an answer

• 10% of web queries are phrase queries, many more are implicit phrase queries (e.g., person names)
Biword Indices

• Consider every pair of consecutive terms in a document as a phrase
  – “Friends, Romans, Countrymen” \(\rightarrow\) “friends romans”, “romans countrymen”
  – “stanford university palo alto” \(\rightarrow\) “stanford university” AND “university palo” AND “palo alto”

• Biwords using nouns
  – “renegotiation of the constitution” matches rule NX*N
  – May not always work well
    • “cost overruns on a power plant” \(\rightarrow\) “cost overruns” AND “overruns power” AND “power plan”
    • “cost overruns” AND “power plan” seem more appropriate
Phrase Indices

• Biword indices can be extended to variable length word sequences
• The longer the phrase, the lower the chance of false-positive
• Including long phrases increases the size of vocabulary
  – A tradeoff between space and query effectiveness as well as efficiency
Positional Indices

- A positional index stores, for each term in the vocabulary, postings of the form docID: <position1, position2, ...>

- Search for “to be and not to be”
  - How many times do we need to search for “to be”?

  to, 993427:
  \[ \langle 1, 6: \langle 7, 18, 33, 72, 86, 231 \rangle; 2, 5: \langle 1, 17, 74, 222, 255 \rangle; 4, 5: \langle 8, 16, 190, 429, 433 \rangle; 5, 2: \langle 363, 367 \rangle; 7, 3: \langle 13, 23, 191 \rangle; \ldots \rangle \]

  be, 178239:
  \[ \langle 1, 2: \langle 17, 25 \rangle; 4, 5: \langle 17, 191, 291, 430, 434 \rangle; 5, 3: \langle 14, 19, 101 \rangle; \ldots \rangle \]
Proximity Intersection

**POSITIONAL INTERSECT**(\(p_1, p_2, k\))

1. \(answer \leftarrow \langle \rangle\)
2. \(while \ p_1 \neq \text{NIL} \text{ and } p_2 \neq \text{NIL} \)
3. \(\text{do if } \text{docID}(p_1) = \text{docID}(p_2) \)
4. \(\text{then } l \leftarrow \langle \rangle\)
5. \(p_{p1} \leftarrow \text{positions}(p_1)\)
6. \(p_{p2} \leftarrow \text{positions}(p_2)\)
7. \(while \ p_{p1} \neq \text{NIL} \)
8. \(\text{do while } p_{p2} \neq \text{NIL} \)
9. \(\text{do if } |\text{pos}(p_{p1}) - \text{pos}(p_{p2})| \leq k\)
10. \(\text{then } \text{ADD}(l, \text{pos}(p_{p2}))\)
11. \(\text{else if } \text{pos}(p_{p2}) > \text{pos}(p_{p1}) \)
12. \(\text{then break}\)

13. \(p_{p2} \leftarrow \text{next}(p_{p2})\)
14. \(\text{while } l \neq \langle \rangle \text{ and } |l[0] - \text{pos}(p_{p1})| > k\)
15. \(\text{do } \text{DELETE}(l[0])\)
16. \(\text{for each } ps \in l\)
17. \(\text{do } \text{ADD}(answer, \langle \text{docID}(p_1), \text{pos}(p_{p1}), ps \rangle)\)
18. \(p_{p1} \leftarrow \text{next}(p_{p1})\)
19. \(p_{1} \leftarrow \text{next}(p_{1})\)
20. \(p_{2} \leftarrow \text{next}(p_{2})\)
21. \(\text{else if } \text{docID}(p_1) < \text{docID}(p_2)\)
22. \(\text{then } p_{1} \leftarrow \text{next}(p_{1})\)
23. \(\text{else } p_{2} \leftarrow \text{next}(p_{2})\)
24. \(\text{return } answer\)
Positional Index Size

• Record positions of all occurrences of a term
  – Boolean query complexity $O(T)$, where $T$ is the number of tokens in the document collection

• Suppose a term has frequency 0.1% on average
  – The term is expected to appear once in a document of 1,000 terms, 100 times in a document of 100,000 terms
  – Document size affects the positional index size

• In practice, a positional index is about 2-4 times as large as a nonpositional index
Combination Schemes

• Biword indices and positional indices can be combined
  – For commonly queried phrases, use biword indices
  – For others, use positional indices
• Consider relative frequencies
  – Use biword indices for phrases where the individual words are common but the desired phrase is comparatively rare
• Partial next word index: for each term, record terms that follow it in a document
  – Save 75% query time, with 26% more space
Example Text Collection

$S_1$ Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

$S_2$ Fishkeepers often use the term tropical fish to refer only those requiring fresh water, with saltwater tropical fish referred to as marine fish.

$S_3$ Tropical fish are popular aquarium fish, due to their often bright coloration.

$S_4$ In freshwater fish, this coloration typically derives from iridescence, while salt water fish are generally pigmented.
Inverted Index with Word Counts

• Help to rank the most relevant documents
Fields and Extents

• Document fields: sections of documents that carry some kind of semantic meaning
  – Name, title, location, etc.

• Extent: a contiguous region of a document
  <author>W. Bruce Croft</author>,
  <author>Donald Metzler</author>, and
  <author>Trevor Strohman</author>

• Author: (1, 3), (4, 5), (7, 8)
Aligning Posting Lists and Field

<table>
<thead>
<tr>
<th>fish</th>
<th>1,2</th>
<th>1,4</th>
<th>2,7</th>
<th>2,18</th>
<th>2,23</th>
<th>3,2</th>
<th>3,6</th>
<th>4,3</th>
<th>4,13</th>
<th>4:(9,15)</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>1:(1,3)</td>
<td>2:(1,5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

J. Pei: Information Retrieval and Web Search -- Efficient Implementation of Postings Lists
Posting Lists of Scores

- We can store the relevance/similarity between a document and a query word
- A posting list of scores
  - Fish: (1: 3.6), (3: 2.2), …
- Increasing flexibility: computationally expensive scoring becomes possible
  - Scoring is moved to the index construction process
- Losing flexibility: the scoring mechanism is fixed once the index is built
  - Information about word proximity is lost
Ordering Using Scores

• An inverted list can be ordered by score so that the highest scored documents come first

• The query processing engine can focus only on the top part of each inverted list
  – Only need to read a small number of postings to find the top-k documents
Threshold Adjustment Algorithm

- Query: A and B, find top-2 documents
  - Scoring function: score(A) + score(B)
- Each posting list is sorted in score descending order
  - A: (D1, 3.8), (D10, 3.7), (D5, 3.6), (D8, 2.9), ...
  - B: (D10, 3.4), (D2, 2.5), (D5, 2.2), (D9, 1.9), (D1, 1.8)
- Step 1: read (D1, 3.8) from A and (D10, 3.4) from B
  - Upper bounds: any document cannot be higher than 7.2
  - Lower bounds: D1 ≥ 3.8, D10 ≥ 3.4
- Step 2: read (D10, 3.7) from A and (D2, 2.5) from B
  - D10 = 7.1
  - Upper bounds: D1 ≤ 6.3, all others ≤ 6.2
  - Lower bounds: D2 ≥ 2.5
  - D10 is the highest
Threshold Adjustment Algorithm

- Step 3: read (D5, 3.6) from A and (D5, 2.2) from B
  - D5 = 5.8
  - Upper bounds: D1 ≤ 6, D2 ≤ 6.1, all others ≤ 5.8
  - Lower bounds:
    - D5 is a candidate
- Step 4: read (D8, 2.9) from A and (D9, 1.9) from B
  - Upper bounds: D1 ≤ 5.7, D2 ≤ 5.4
  - D5 is another document in top-2
  - The search terminates
Summary

- Inverted indices can be improved by efficient implementation of postings lists
- Using skip pointers
- Phrase queries
  - Biword indices
  - Positional postings
  - The two can be combined
- Inverted indexes can be extended in many ways
- The TA algorithm
To-do List

• Give the pseudocode of the TA algorithm
  – How can the lower bounds be used? This is not discussed in the class
  – Can you generalize it for multiple lists?