Chapter 3

Phrase Retrieval

Information Retrieval

Amin Nazari

Spring 2025

Recall

- A large corpus is a challenge.
- Phrase retrieval
- Deterministic vs Probabilistic(Ranking)

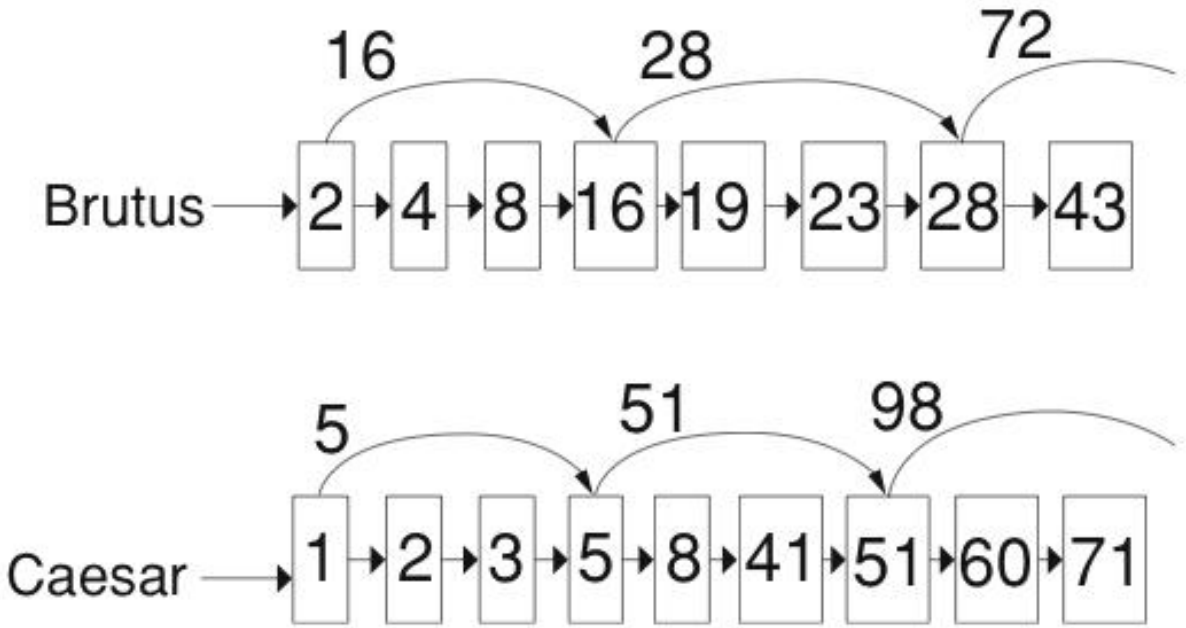
Outline

- Skip pointers
- Phrase retrieval

Skip pointers

- Skip pointers allow us to skip postings that will not figure in the search results.
- This makes intersecting postings lists more efficient.
- Some postings lists contain several million entries – so efficiency can be an issue even if basic intersection is linear.
- Where do we put skip pointers?
- How do we make sure intersection results are correct?

Skip lists: Larger example



Intersection with skip pointers

```
INTERSECTWITHSKIPS(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then if hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2))
 9                     then while hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2))
10                          do p1 ← skip(p1)
11                     else p1 ← next(p1)
12                else if hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1))
13                     then while hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1))
14                          do p2 ← skip(p2)
15                     else p2 ← next(p2)
16  return answer
```

Where do we place skips?

- Tradeoff: number of items skipped vs. frequency skip can be taken.
- More skips: Each skip pointer skips only a few items, but we can frequently use it.
- Fewer skips: Each skip pointer skips many items, but we can not use it very often.

Where do we place skips?

- Simple heuristic: for postings list of length *P,* use $\sqrt{p}$ evenly-spaced skip pointers.
- Easy if the index is static; harder in a dynamic environment because of updates.
- With today's fast CPUs, they don't help that much anymore.

Quiz

- For the following example, specify the skip pointers and execute the intersection algorithm step by step. Which skip pointers are used?

- apple: [2, 5, 9, 15, 18, 20, 25, 30, 40]
- banana: [5, 9, 20, 25, 30, 35]

Phrase 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.
- Two ways of extending the inverted index:
  - biword index
  - positional index

Biword indexes

- 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.

Longer phrase queries

- A long phrase like *“stanford university palo alto”* can be represented as the Boolean query “STANFORD UNIVERSITY” AND “UNIVERSITY PALO” AND “PALO ALTO”
- We need to do post-filtering of hits to identify subset that actually contains the 4-word phrase.

Issues with biword indexes

- Why are biword indexes rarely used?
  - False positives(longer phrase)
  - 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

Positional indexes: Example

TO, 993427:

‹ 1: ‹7, 18, 33, 72, 86, 231›;

2: ‹1, 17, 74, 222, 255›;

4: ‹8, 16, 190, 429, 433›;

5: ‹363, 367›;

7: ‹13, 23, 191›; . . . ›

BE, 178239:

‹ 1: ‹17, 25›;

4: ‹17, 191, 291, 430, 434›;

5: ‹14, 19, 101›; . . . ›

“Proximity” intersection

```
POSITIONALINTERSECT(p1, p2, k)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then l ← ⟨ ⟩
 5             pp1 ← positions(p1)
 6             pp2 ← positions(p2)
 7             while pp1 ≠ NIL
 8             do while pp2 ≠ NIL
 9                do if |pos(pp1) − pos(pp2)| ≤ k
10                      then ADD(l, pos(pp2))
11                      else if pos(pp2) > pos(pp1)
12                              then break
13                   pp2 ← next(pp2)
14                while l ≠ ⟨ ⟩ and |l[0] − pos(pp1)| > k
15                do DELETE(l[0])
16                for each ps ∈ l
17                do ADD(answer, ⟨docID(p1), pos(pp1), ps⟩)
18                pp1 ← next(pp1)
19             p1 ← next(p1)
20             p2 ← next(p2)
21        else if docID(p1) < docID(p2)
22                then p1 ← next(p1)
23                else p2 ← next(p2)
24  return answer
```

Python

- Biword retrieval
- Positional retrieval

Summary

- A large corpus is a challenge.
- Phrase retrieval
  - With a positional index, we can answer phrase queries.
  - With a positional index, we can answer proximity queries.
- Deterministic vs Probabilistic(Ranking)