Text Retrieval and Web Search
IIR 2: The term vocabulary and postings lists

Mihai Surdeanu
(Based on slides by Hinrich Schütze at `informationretrieval.org`)

Spring 2017

Overview

1 Tokenization

2 Normalization

3 Skip pointers

4 Phrase queries

Motivation

"In 2000, the Institute of Medicine reported that an estimated 98,000 preventable patient deaths occur annually in US hospitals due to . . ."

Take-away

- Understanding of the basic unit of classical information retrieval systems: what is a token?
- Tokenization: how to get from raw text to words (or tokens)
- More complex indexes: skip pointers and phrases

Outline

1 Tokenization

2 Normalization

3 Skip pointers

4 Phrase queries

Definitions

- Document – File, email, newspaper article, tweet, Facebook post, etc. A column in the term-document incidence matrix.
- Token == Word – A delimited string of characters as it appears in a document.
- Term – A "**normalized**" (case, morphology, spelling etc) and **unique** word. It is included in the index.
- Type – An equivalence class of tokens (e.g., "USA" and "U.S.A"). Not necessarily in the index.

Normalization

- Need to "normalize" words in indexed text as well as query terms into the same form.
- Example: We want to match *U.S.A.* and *USA*
- We most commonly implicitly define equivalence classes of terms, which are created during normalization.
- There are also explicit equivalence classes:
  - Soundex: IIR 3 (phonetic equivalence, Muller = Mueller)
  - Thesauri: IIR 9 (semantic equivalence, car = automobile)
- What's the best way to handle (explicit) equivalence classes?

Normalization: Other languages

- Normalization and language detection interact.
- *PETER WILL NICHT MIT.* $\rightarrow$ MIT $=$ mit
- *He got his PhD from MIT.* $\rightarrow$ MIT $\neq$ mit

Recall: Inverted index construction

- Input:

  Friends, Romans, countrymen. | So let it be with Caesar | . . .

- Output:

  friend | roman | countryman | so | . . .

- Each token is a candidate for a postings entry.
- What are valid tokens to emit?

Exercises

In June, the dog likes to chase the cat in the barn. – How many word tokens? How many terms?

Why tokenization is difficult – even in English. Tokenize: *Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.*

Tokenization problems: One word or two? (or several)

- Hewlett-Packard
- State-of-the-art
- co-education
- the hold-him-back-and-drag-him-away maneuver
- data base
- San Francisco
- Los Angeles-based company
- cheap San Francisco-Los Angeles fares
- York University vs. New York University

- What is a simple heuristic?

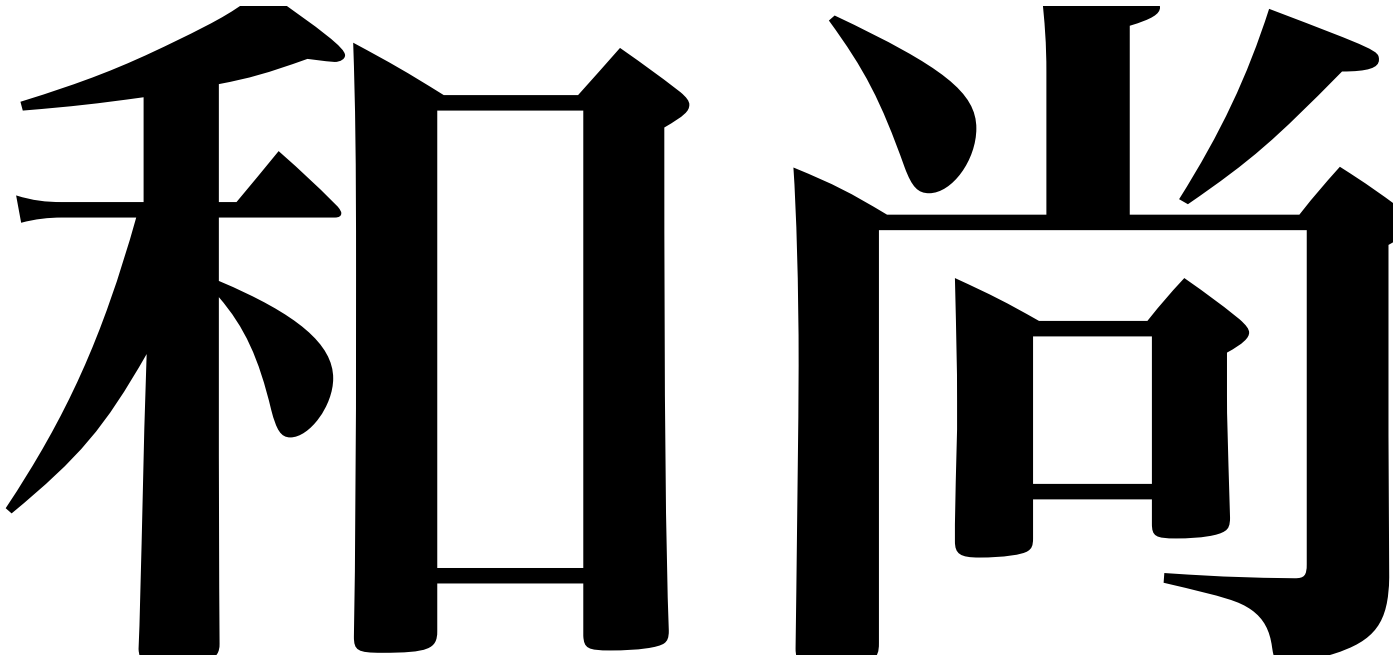
Numbers

- 3/20/91
- 20/3/91
- Mar 20, 1991
- 100.2.86.144
- (800) 234-2333
- 800.234.2333
- Older IR systems may not index numbers . . .
- . . . but generally it's a useful feature.
- Google example

Chinese: No whitespace

莎拉波娃现在居住在美国东南部的佛罗里达。今年４月９日，莎拉波娃在美国第一大城市纽约度过了１８岁生日。生日派对上，莎拉波娃露出了甜美的微笑。

Ambiguous segmentation in Chinese



The two characters can be treated as one word meaning ‘monk’ or as a sequence of two words meaning ‘and’ and ‘still’.

Other cases of "no whitespace"

- Compounds in Dutch, German, Swedish
- Computerlinguistik → Computer + Linguistik
- Lebensversicherungsgesellschaftsangestellter (life insurance company employee)
- → leben + versicherung + gesellschaft + angestellter
- Inuit: tusaatsiarunnanngittualuujunga (I can't hear very well.)
- Many other languages with segmentation difficulties: Finnish, Urdu, . . .

- Have you read "The Awful German Language" by Mark Twain?

Japanese

ノーベル平和賞を受賞したワンガリ・マータイさんが名誉会長を務めるＭＯＴＴＡＩＮＡＩキャンペーンの一環として、毎日新聞社とマガジンハウスは「私の、もったいない」を募集します。皆様が日ごろ「もったいない」と感じて実践していることや、それにまつわるエピソードを８００字以内の文章にまとめ、簡単な写真、イラスト、図などを添えて１０月２０日までにお送りください。大賞受賞者には、５０万円相当の旅行券とエコ製品２点の副賞が贈られます。

Japanese

ノーベル平和賞を受賞したワンガリ・マータイさんが名誉会長を務めるＭＯＴＴＡＩＮＡＩキャンペーンの一環として、毎日新聞社とマガジンハウスは「私の、もったいない」を募集します。皆様が日ごろ「もったいない」と感じて実践していることや、それにまつわるエピソードを８００字以内の文章にまとめ、簡単な写真、イラスト、図などを添えて１０月２０日までにお送りください。大賞受賞者には、５０万円相当の旅行券とエコ製品２点の副賞が贈られます。

4 different "alphabets": Chinese characters, hiragana syllabary for inflectional endings and function words, katakana syllabary for transcription of foreign words and other uses, and latin. No spaces (as in Chinese).

Arabic script: Bidirectionality

استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.

← START ← → ← →

'Algeria achieved its independence in 1962 after 132 years of French occupation.'

It's hard and boring!

- This is a big pain for any IR/NLP software. Let's look at Stanford's CoreNLP.
- It's best if you don't do this at home but instead use existing software:
  - NLTK in Python: `http://www.nltk.org/book/ch03.html`
  - CoreNLP in Java: `http://stanfordnlp.github.io/CoreNLP/simple.html`

Outline

1 Tokenization

2 Normalization

3 Skip pointers

4 Phrase queries

Accents and diacritics

- Accents: résumé vs. resume (simple omission of accent)
- Umlauts: Universität vs. Universitaet (substitution with special letter sequence "ae")
- Most important criterion: How are users likely to write their queries for these words?
- Even in languages that standardly have accents, users often do not type them. (Polish, Romanian)

Case folding

- Reduce all letters to lower case
- Even though case can be semantically meaningful
  - capitalized words in mid-sentence
  - MIT vs. mit
  - Fed vs. fed
  - ...
- It's often best to lowercase everything. Why?

Stop words

- stop words = extremely common words which would appear to be of little value in helping select documents matching a user need
- Examples: *a*, *an*, *and*, *are*, *as*, *at*, *be*, *by*, *for*, *from*, *has*, *he*, *in*, *is*, *it*, *its*, *of*, *on*, *that*, *the*, *to*, *was*, *were*, *will*, *with*
- Stop word elimination used to be standard in older IR systems.
- But you need stop words for phrase queries, e.g. "King of Denmark"
- Most web search engines index stop words.

Lemmatization

- Reduce inflectional/variant forms to base form
- Example: *am*, *are*, *is* $\rightarrow$ *be*
- Example: *car*, *cars*, *car's*, *cars'* $\rightarrow$ *car*
- Example: *the boy's cars are different colors* $\rightarrow$ *the boy car be different color*
- Lemmatization implies doing "proper" reduction to dictionary headword form (the lemma).
- Inflectional morphology (*cutting* $\rightarrow$ *cut*) vs. derivational morphology (*destruction* $\rightarrow$ *destroy*)
- Use WordNet for proper lemmatization. For a quick hack...

Stemming

- Definition of stemming: Crude heuristic process that chops off the ends of words in the hope of achieving what "principled" lemmatization attempts to do with a lot of linguistic knowledge.
- Language dependent
- Often inflectional and derivational
  - Example for derivational: *automate*, *automatic*, *automation* all reduce to *automat*
  - Example for inflectional: *am*, *are*, *is* reduce to *be*

Porter algorithm

- Most common algorithm for stemming English
- Results suggest that it is at least as good as other stemming options
- Conventions + 5 phases of reductions
- Phases are applied sequentially
- Each phase consists of a set of commands.
  - Sample command: Delete final *ement* if what remains is longer than 1 character
  - replacement → replac
  - cement → cement
- Sample convention: Of the rules in a compound command, select the one that applies to the longest suffix.

Porter stemmer: A few rules

Rule			**Example**		
SSES	$\rightarrow$	SS	caresses	$\rightarrow$	caress
IES	$\rightarrow$	I	ponies	$\rightarrow$	poni
SS	$\rightarrow$	SS	caress	$\rightarrow$	caress
S	$\rightarrow$		cats	$\rightarrow$	cat

Three stemmers: A comparison

Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Porter stemmer: such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

Lovins stemmer: such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

Paice stemmer: such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

WordNet Lemmatization

- Based on morphological analysis + dictionary lookup rather than "chopping" suffixes
- What are the advantages/disadvantages?
- Demo: `http://textanalysisonline.com/nltk-wordnet-lemmatizer`
- Code:
  - Python: `http://www.nltk.org/_modules/nltk/stem/wordnet.html`
  - Java: `http://stanfordnlp.github.io/CoreNLP/simple.html`

Does stemming improve effectiveness?

- In general, stemming increases effectiveness for some queries, and decreases effectiveness for others.
- Queries where stemming is likely to help: [tartan sweaters], [sightseeing tour san francisco]
- (equivalence classes: {sweater,sweaters}, {tour,tours})
- Porter Stemmer equivalence class *oper* contains all of *operate operating operates operation operative operatives operational*.
- Queries where stemming hurts: [operational AND research], [operating AND system], [operative AND dentistry]

Exercise: What does Google do?

- Stop words (??)
- Normalization
- Tokenization
- Lowercasing
- Stemming Do they use WordNet?
- Non-latin alphabets
- Umlauts
- Compounds
- Numbers

Outline

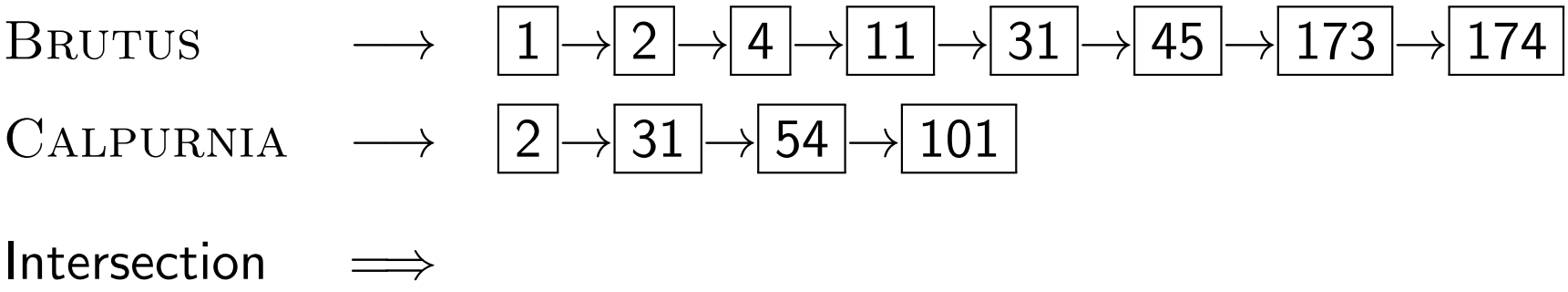
1 Tokenization

2 Normalization

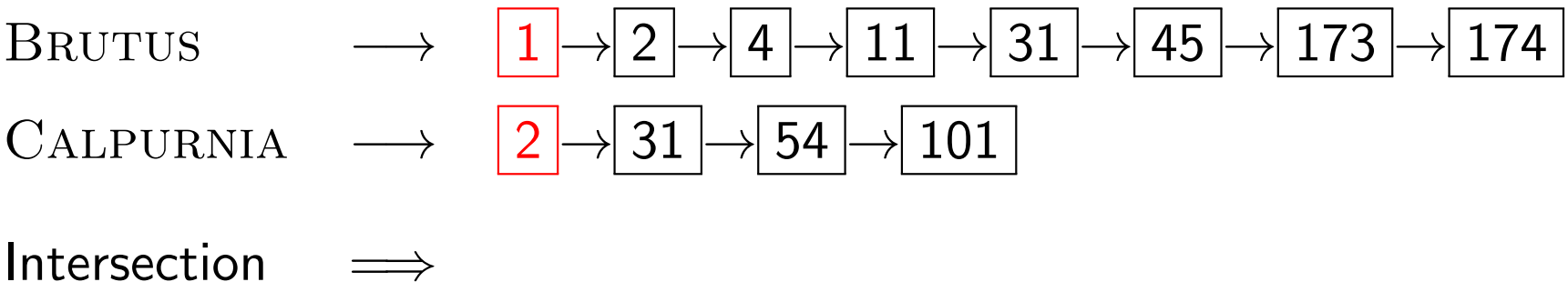
3 Skip pointers

4 Phrase queries

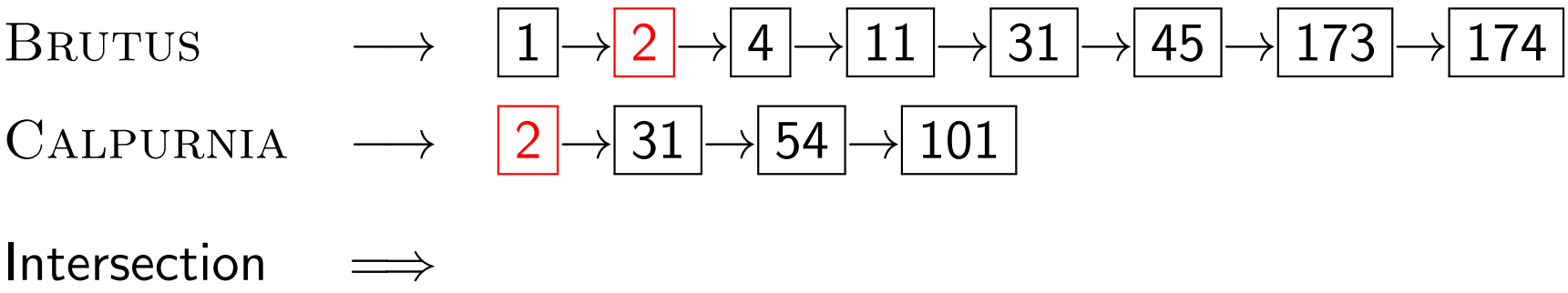
Recall basic intersection algorithm



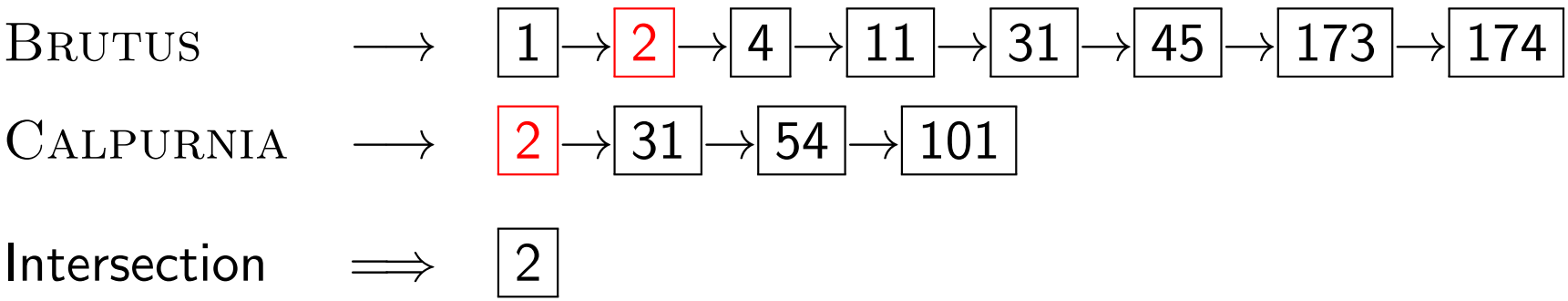
Recall basic intersection algorithm



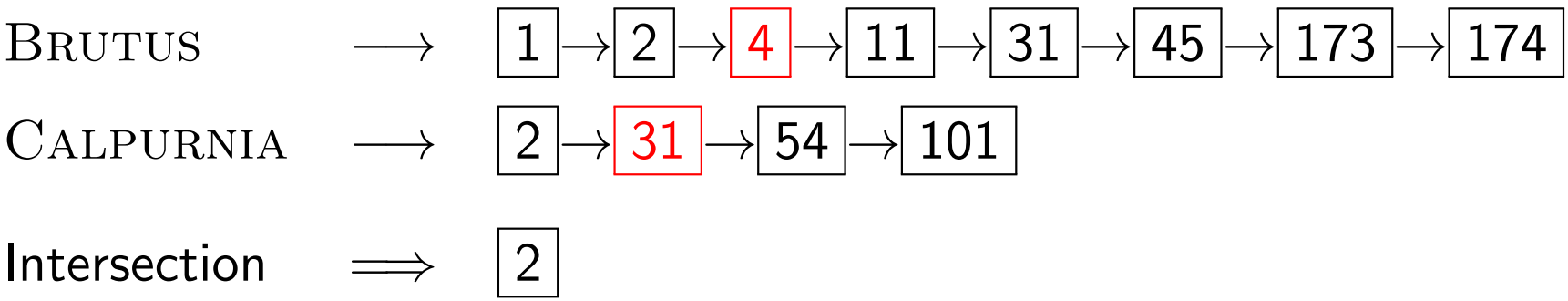
Recall basic intersection algorithm



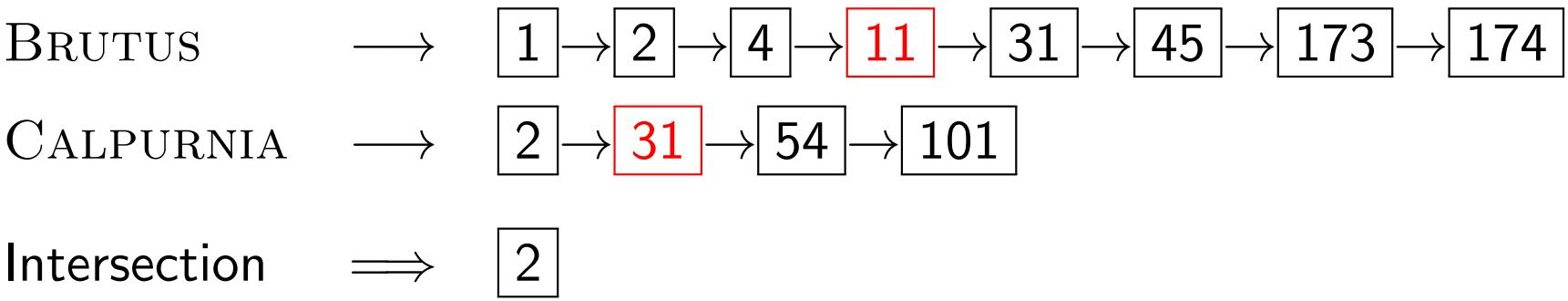
Recall basic intersection algorithm



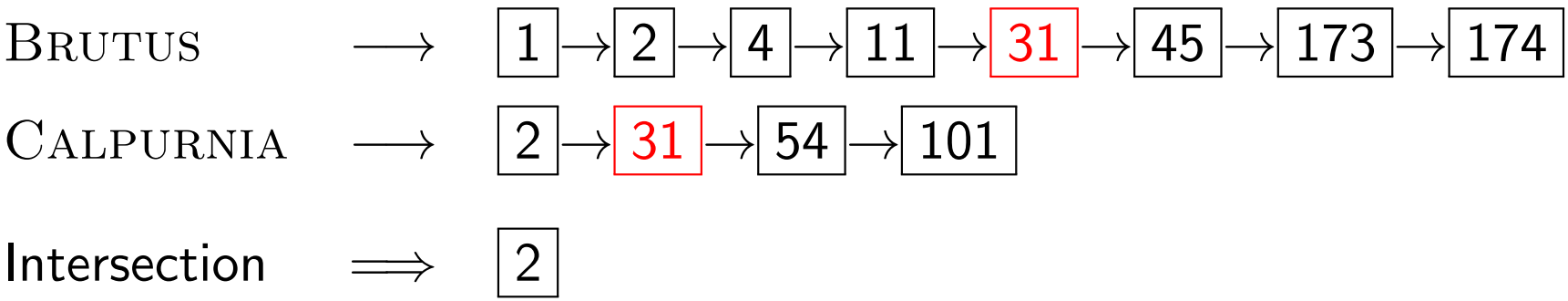
Recall basic intersection algorithm



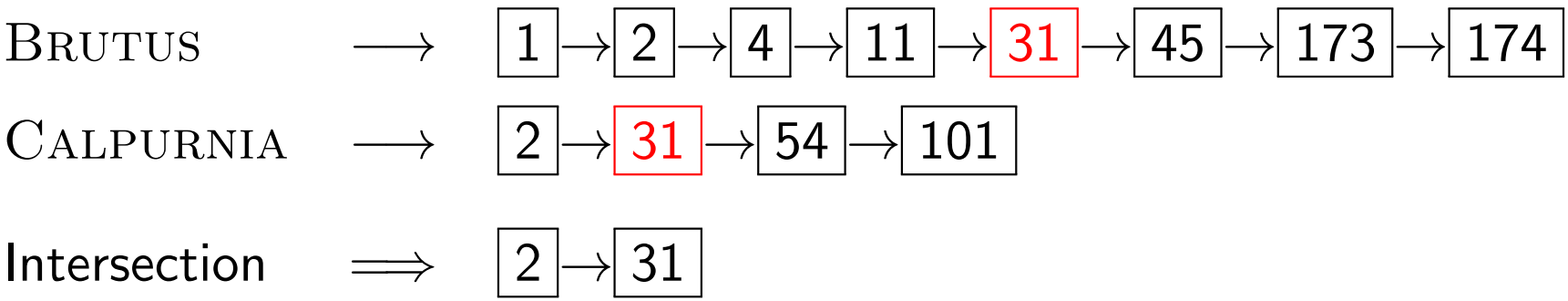
Recall basic intersection algorithm



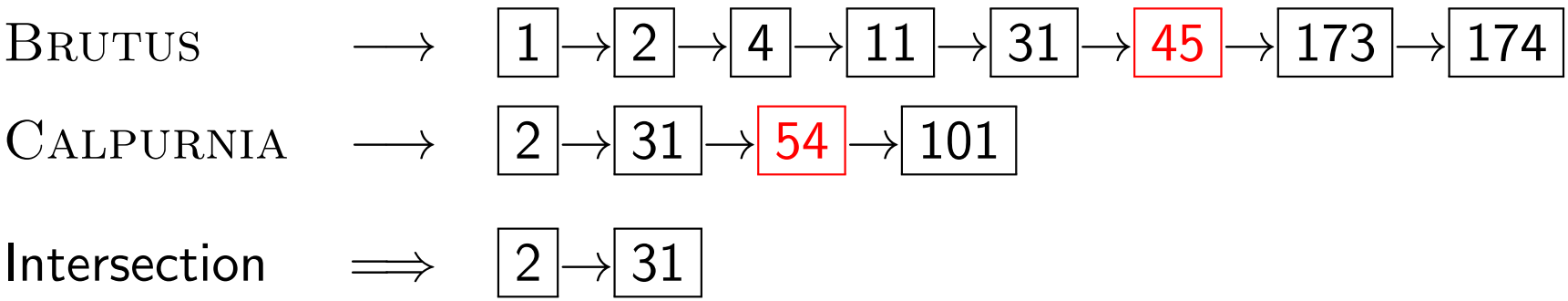
Recall basic intersection algorithm



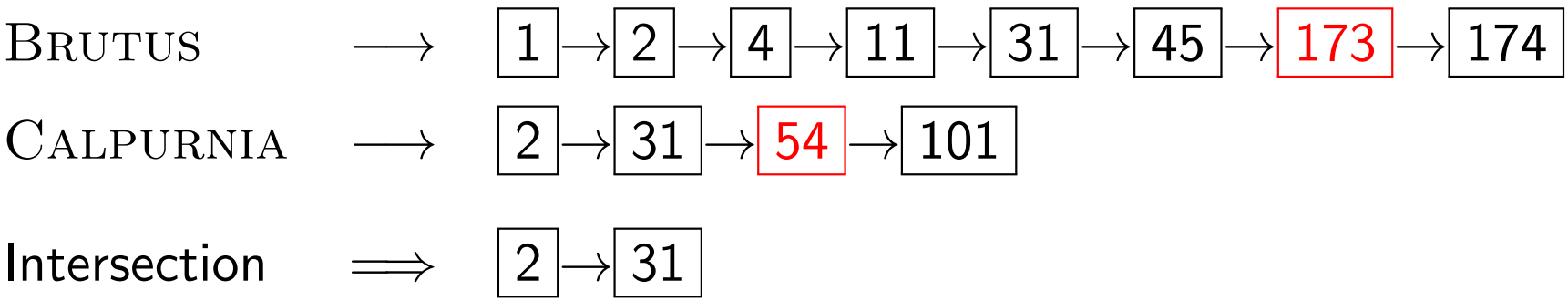
Recall basic intersection algorithm



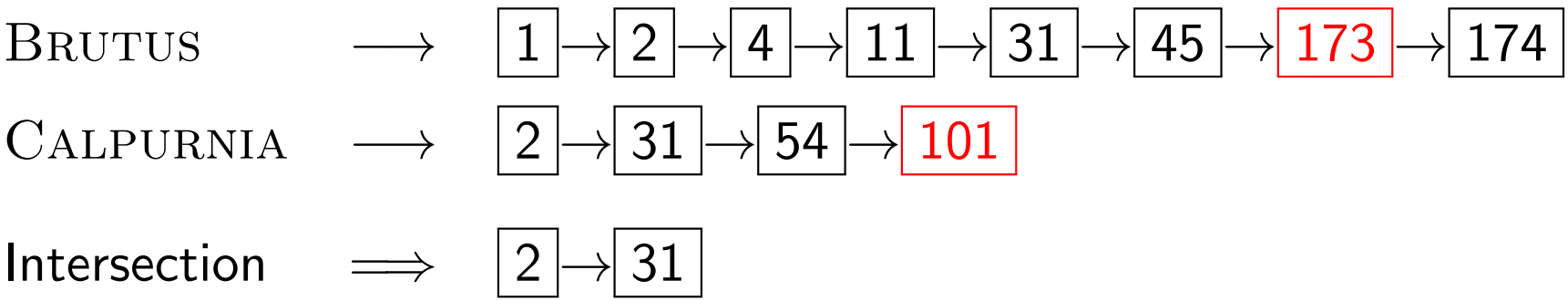
Recall basic intersection algorithm



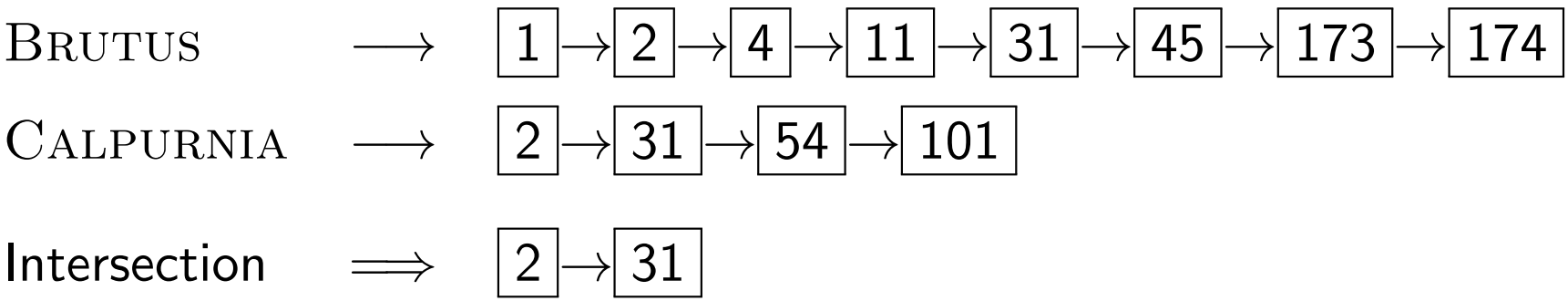
Recall basic intersection algorithm



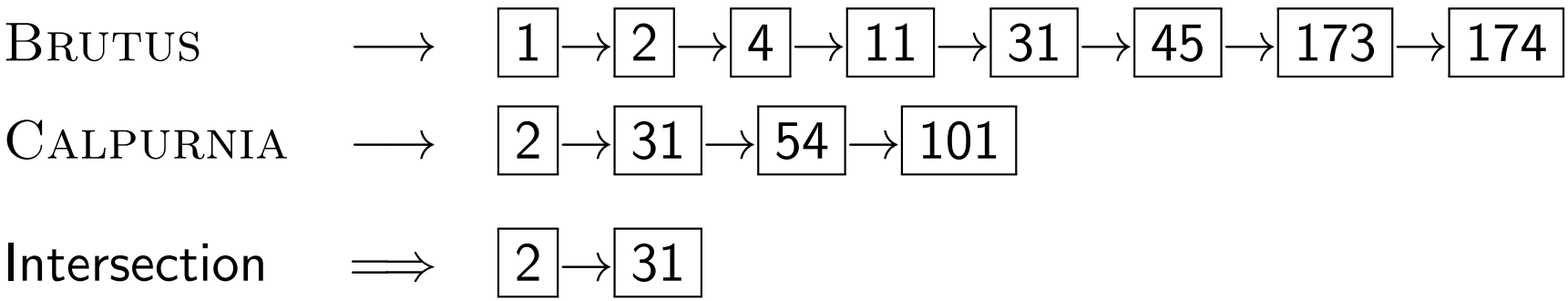
Recall basic intersection algorithm



Recall basic intersection algorithm

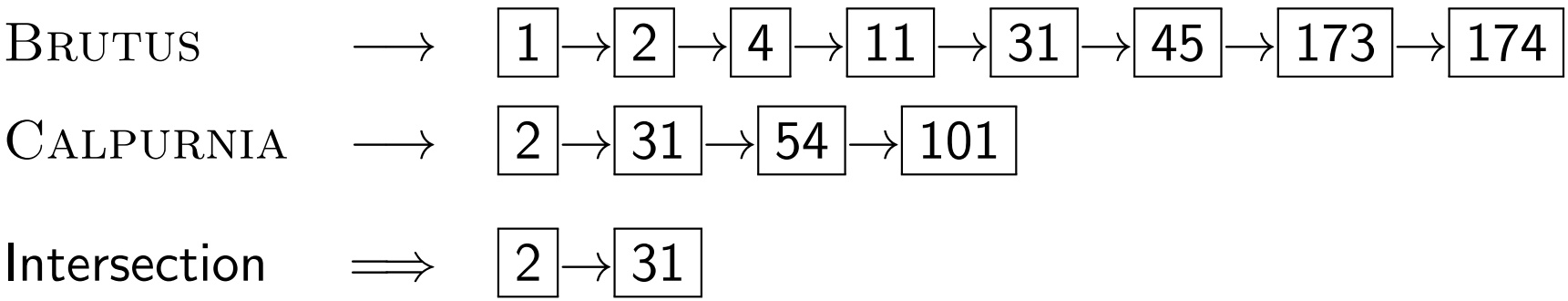


Recall basic intersection algorithm



- Linear in the length of the postings lists.

Recall basic intersection algorithm

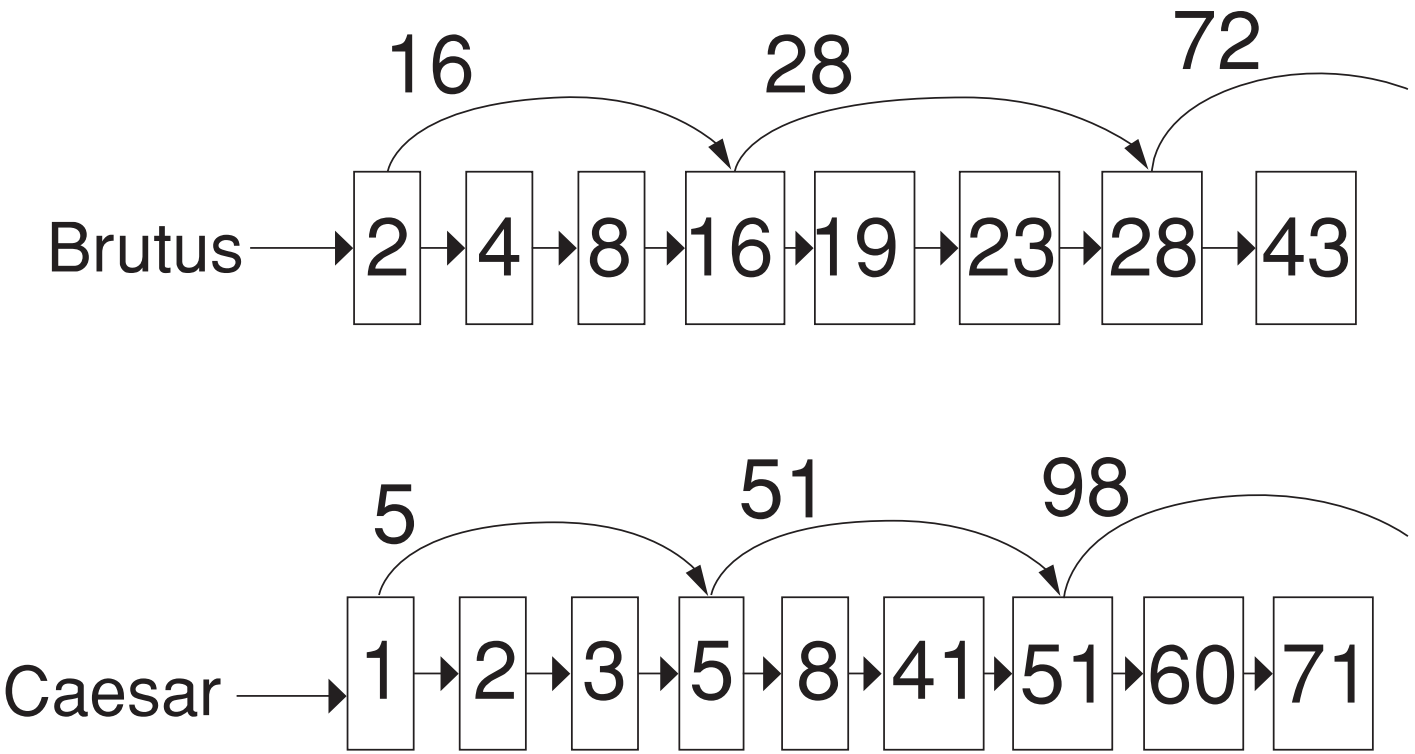


- Linear in the length of the postings lists.
- Can we do better?

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: Example



Intersecting 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
```

Exercise

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? (cont)

- Simple heuristic: for postings list of length P, use $\sqrt{P}$ evenly-spaced skip pointers.
- This ignores the distribution of query terms.
- Easy if the index is static; harder in a dynamic environment because of updates.
- How much do skip pointers help?
- They used to help a lot.
- With today's fast CPUs and in-memory indices, they don't help that much anymore.

Outline

1 Tokenization

2 Normalization

3 Skip pointers

4 Phrase queries

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.
- Consequence for inverted index: it no longer suffices to store docIDs in postings lists.
- 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?

Issues with biword indexes

- Why are biword indexes rarely used?
- False positives, as noted above
- 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

Query: "to_1 be_2 or_3 not_4 to_5 be_6"

Positional indexes: Example

Query: "to_1 be_2 or_3 not_4 to_5 be_6"

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⟩; . . . ⟩

Positional indexes: Example

Query: "to_1 be_2 or_3 not_4 to_5 be_6"

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⟩; . . . ⟩

Positional indexes: Example

Query: "to_1 be_2 or_3 not_4 to_5 be_6"

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⟩; . . . ⟩

Positional indexes: Example

Query: "to_1 be_2 or_3 not_4 to_5 be_6"

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⟩; . . . ⟩

Positional indexes: Example

Query: "to_1 be_2 or_3 not_4 to_5 be_6"

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⟩; . . . ⟩

Positional indexes: Example

Query: "to_1 be_2 or_3 not_4 to_5 be_6"

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⟩; . . . ⟩

Positional indexes: Example

Query: *"to*$_1$ *be*$_2$ *or*$_3$ *not*$_4$ *to*$_5$ *be*$_6$*"*

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⟩; . . . ⟩

Positional indexes: Example

Query: *"to*$_1$ *be*$_2$ *or*$_3$ *not*$_4$ *to*$_5$ *be*$_6$*"*

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⟩; . . . ⟩

Positional indexes: Example

Query: *"to*$_1$ *be*$_2$ *or*$_3$ *not*$_4$ *to*$_5$ *be*$_6$*"*

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⟩; . . . ⟩

Positional indexes: Example

Query: "to_1 be_2 or_3 not_4 to_5 be_6"

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⟩; . . . ⟩

Positional indexes: Example

Query: "to_1 be_2 or_3 not_4 to_5 be_6"

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⟩; . . . ⟩

Positional indexes: Example

Query: "to_1 be_2 or_3 not_4 to_5 be_6"

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⟩; . . . ⟩

Positional indexes: Example

Query: *"to*$_1$ *be*$_2$ *or*$_3$ *not*$_4$ *to*$_5$ *be*$_6$*"*

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⟩; . . . ⟩

Positional indexes: Example

Query: *"to*$_1$ *be*$_2$ *or*$_3$ *not*$_4$ *to*$_5$ *be*$_6$*"*

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⟩; . . . ⟩

Positional indexes: Example

Query: "to_1 be_2 or_3 not_4 to_5 be_6"

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⟩; . . . ⟩

Positional indexes: Example

Query: *"to*$_1$ *be*$_2$ *or*$_3$ *not*$_4$ *to*$_5$ *be*$_6$*"*

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⟩; . . . ⟩

Positional indexes: Example

Query: "to_1 be_2 or_3 not_4 to_5 be_6"

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⟩; . . . ⟩

Positional indexes: Example

Query: "to_1 be_2 or_3 not_4 to_5 be_6"

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⟩; . . . ⟩

Document 4 is a match!

Proximity search

- We just saw how to use a positional index for phrase searches.
- We can also use it for proximity search.
- For example: employment /4 place
- Find all documents that contain EMPLOYMENT and PLACE within 4 words of each other.
- *Employment agencies that place healthcare workers are seeing growth* is a hit.
- *Employment agencies that have learned to adapt now place healthcare workers* is not a hit.

Proximity search

- Use the positional index
- Simplest algorithm: look at cross-product of positions of (i) EMPLOYMENT in document and (ii) PLACE in document
- Very inefficient for frequent words, especially stop words
- Note that we want to return the actual matching positions, not just a list of documents.
- This is important for dynamic summaries etc.

"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
```

Exercise!

Combination scheme

- Biword indexes and positional indexes can be profitably combined.
- Many biwords are extremely frequent: Michael Jackson, Britney Spears etc
- For these biwords, increased speed compared to positional postings intersection is substantial.
- Combination scheme: Include frequent biwords as vocabulary terms in the index. Do all other phrases by positional intersection.

Take-away

- Understanding of the basic unit of classical information retrieval systems: what is a token?
- Tokenization: how to get from raw text to words (or tokens)
- More complex indexes: skip pointers and phrases