

Stemming & Lemmatization:

Root form of a words
→ stem / lemma.

~~Lemmatization~~ Lemmatization

→ proper reduction of the word to its lemma (usually

am, are, is → be
car, car's, cars, cars' → car

by matching the headword from a dictionary).

inflections.
(morphology)

Stemming

→ surface form
of the word).

→ chops the end of a word and assumes that the stem has been obtained.

- ① car ^(N) → cars ^(N)
② automate ^(V) → automation ^(N)
- derivational morphology

→ heuristics.

→ inflectional morphology.

Text processing for IR systems-

Stop words & ~~their~~ their removal.

→ words (do not have ~~their~~ a meaning of their own)
→ Render meaning to other words in a query/document

Articles, prepositions, conjunction, pronouns
auxiliary verbs ← stop words

{ a, an, the, will, was, were,
has, he, in, ... }

→ very frequent in a document
Some of the most frequent words in documents.

Design policy: not separately index these words.
Caveats: phrase queries
"Joan of Arc"

automation, automatic → automat
(also) Algo that does stemming → stemmer.

(Porter Stemmer)

Stage 1

✓ ss es → ss (caresses → caress)

ies → i (ponies → poni)

ss → s (vivess → vivess) (caress → caress)

✓ s → ϕ (cat's → cat)

Carries ← longest match from the end first.

~~Stage~~ Stage 2

~~(*)ing)ed~~

vowel.
(*)ing → ϕ [walking → walk, King → king]

(*)ed → ϕ

[played → play]

Stage 3

only 3 among many rules

relational \rightarrow ate (relational \rightarrow relate)
 izer \rightarrow ize (digitizer \rightarrow digitize)
 ator \rightarrow ate (operator \rightarrow operate)

Stage 4

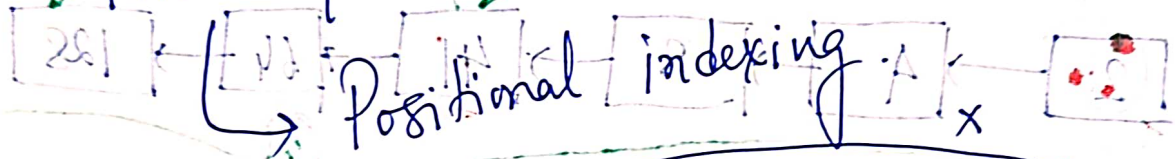
al $\rightarrow \phi$ (revival \rightarrow revive)
 able $\rightarrow \phi$ (adjustable \rightarrow adjust)
 ate $\rightarrow \phi$ (activate \rightarrow active)

automate \rightarrow automate

morphology

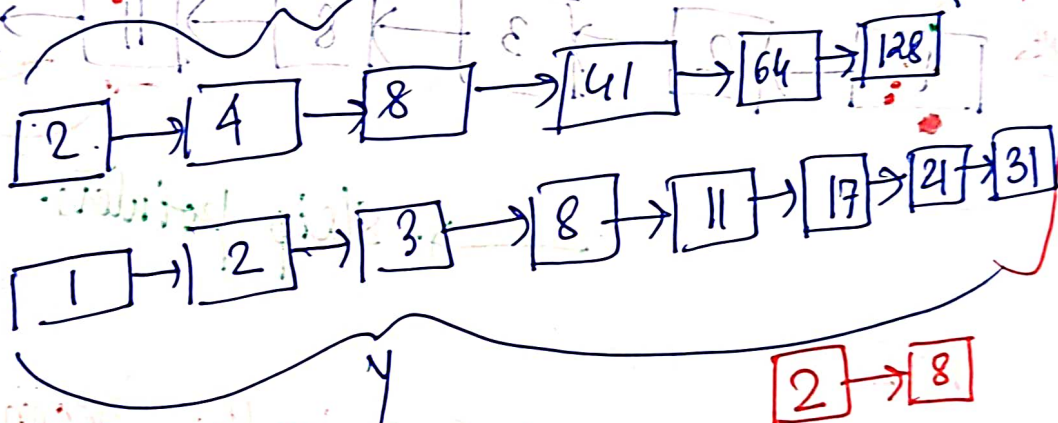
Making postings list operations more efficient \rightarrow skip pointers

Phrase queries



Brutus

Caesar



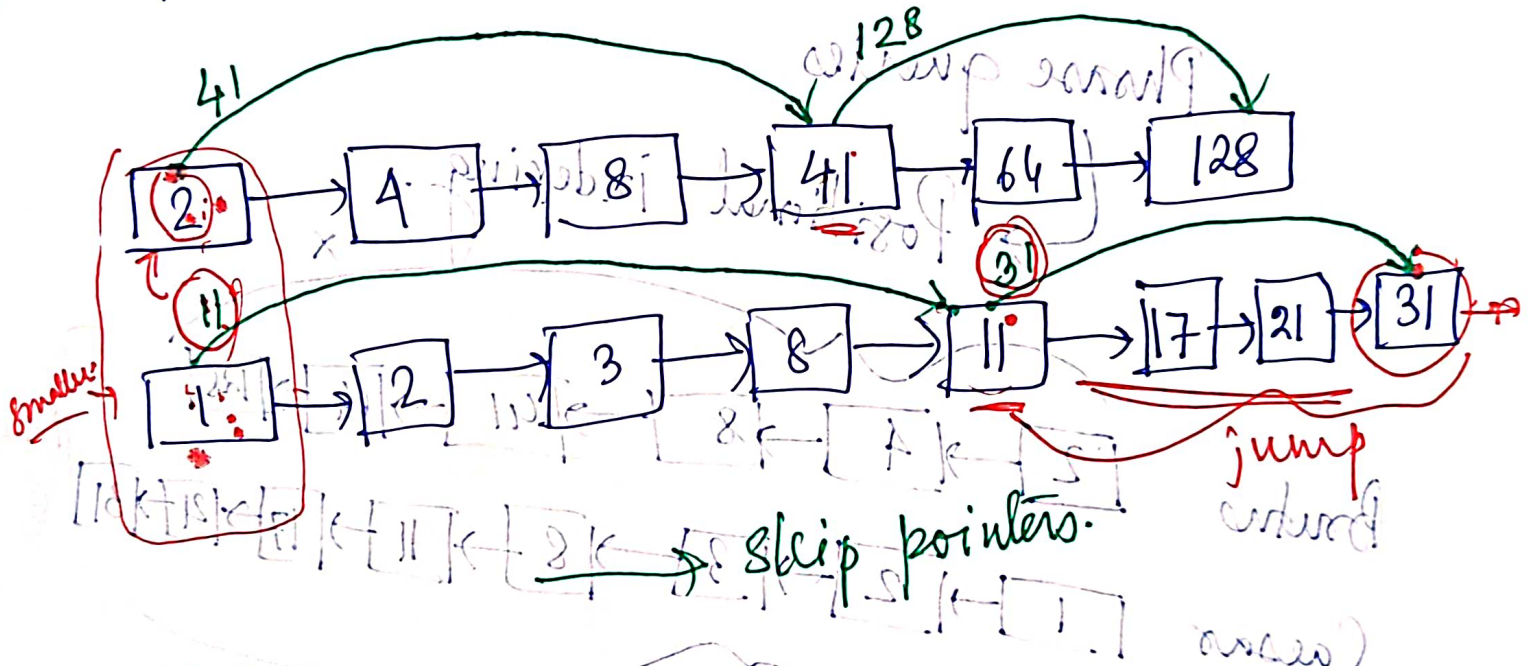
~~$O(X+Y)$~~ $O(X+Y)$

Can we make it any better?

Augment the postings list \rightarrow skip pointers

postings lists are not changing too fast.

skip those docIDs which will not feature in the search.



1172

11 > 2
Till we have reached 11 in the second list

21 < 41

Agreement the postage list —
Can we make it any better?

INTERSECT WITH SKIP (p_1, p_2)

1. answer $\leftarrow \langle \rangle$

while $p_1 \neq \text{NIL}$ & $p_2 \neq \text{NIL}$

do if $\text{docID}(p_1) = \text{docID}(p_2)$

then ADD(answer, $\text{docID}(p_2)$)

$p_1 \leftarrow \text{next}(p_1)$

$p_2 \leftarrow \text{next}(p_2)$

else if $\text{docID}(p_1) < \text{docID}(p_2)$

then if hasSkip(p_1) & $(\text{docID}(\text{skip}(p_1)) \leq \text{docID}(p_2))$

then while hasSkip(p_1) & $(\text{docID}(\text{skip}(p_1)) \leq \text{docID}(p_2))$

do $p_1 \leftarrow \text{skip}(p_1)$

else ~~$p_1 \leftarrow \text{next}(p_1)$~~

else if hasSkip(p_2) & $(\text{docID}(\text{skip}(p_2)) \leq \text{docID}(p_1))$

then while hasSkip(p_2) & $(\text{docID}(\text{skip}(p_2)) \leq \text{docID}(p_1))$

do $p_2 \leftarrow \text{skip}(p_2)$

$p_2 \leftarrow \text{next}(p_2)$

return answer

\leftarrow else

How many ship pointers?] IR designer
will have restr.

② Where to place them?

Tradeoff for ①

More ships. — ship spans will be shorter.
hardly miss anything

Many pointer comparisons.

Less ships. — Less pointer comparisons.

Unsuccessful frequently.

② If the list size is L then

[Maffat & Zobel 1996]

Keeping \sqrt{L} ship pointers evenly placed
empirically gives the best performance.

Easy \rightarrow L is not growing frequently.

More pointers means more memory.

I/O of bringing ship pointer laden postings list from disk to memory can surpass the benefit of merging time saved by these additional pointers.

Phrase queries

[Stanford University] → together as a unit.

will return such cases.

→ The inventor Stanford Orshinsky never went to university.

I/O of web queries → phrase queries.

→ backward indexing.

→ positional indexing.

Binrod indexing \rightarrow indexes bigrams. \rightarrow "Friends, Romans, Countrymen"
"friends romans", "romans countrymen"
binrods \rightarrow added to the dictionary.
"Stanford university palo alto" \rightarrow stanford university AND palo alto

Caveat. "abolition of slavery"

Extended binrods.

What is the part-of-speech of each word in query / document? \rightarrow POS-tagger.

Bucket all the terms that are nouns (N) and articles/prepositions (X).

Extended binrod: $N X^* N$

Catcher N in the eye
 X X N

King N of Denmark
 X N

New vocabulary \rightarrow all the extended binrods of the form $N X^* N$

Caveat: Could easily blow up.

Positional indexes (reconstruction of the postings list).

Postings list in a nonpositional index: each posting is just a docID.

Postings list in a positional index: each posting is a docID + a list of positions.

"(to) be or not (to) be"

to, 993427:

List of positions where "to" has appeared in doc 1.

Doc 1
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>; ...

431 432
or not

① Get the postings list of to, be, or, not.

② Progressive intersect the postings list starting with to & be

docID = 2

removed easily

merge within merge

NOT EQUALITY

the positional merge checks for difference of $K=1$