Introduction to Information Retrieval

Index Compression
(Dictionary/Posting)

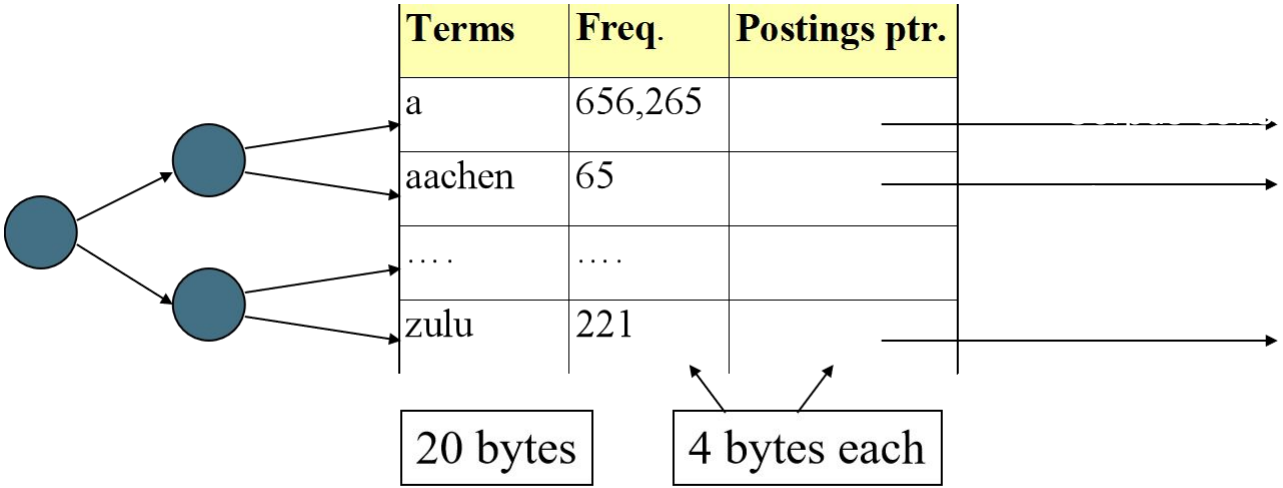
Dictionary Compression

Why compress the dictionary?

- Search begins with the dictionary
- We want to keep it in memory
- Memory footprint competition with other applications
- Embedded/mobile devices may have very little memory
- Even if the dictionary isn't in memory, we want it to be small for a fast search startup time
- So, compressing the dictionary is important

Dictionary storage - first cut

- Array of fixed-width entries
- ~400,000 terms; 28 bytes/term = 11.2 MB.



Fixed-width terms are wasteful

- Most of the bytes in the Term column are wasted – we allot 20 bytes for 1 letter terms. The , than, umbrella
    - And we still can't handle supercalifragilisticexpialidocious or hydrochlorofluorocarbons.
- Written English averages ~4.5 characters/word.
    - Exercise: Why is/isn't this the number to use for estimating the dictionary size?

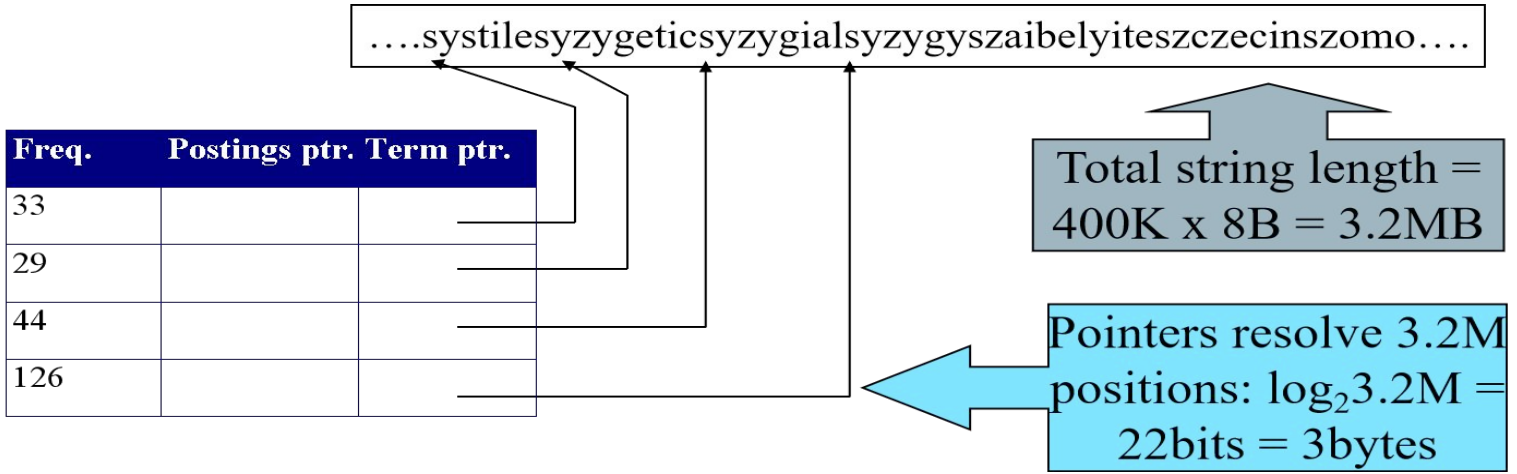
Fixed-width terms are wasteful

- Ave. dictionary word in English: ~8 characters
    - How do we use ~8 characters per dictionary term?
- Short words dominate token counts but not type average.

- Token - Term I want to have a I want to dinner
- Token – 9
- Terms - 6

Compressing the term list: Dictionary-as-a-String

- Store dictionary as a (long) string of characters:
  - Pointer to next word shows end of current word
  - Hope to save up to 60% of dictionary space.



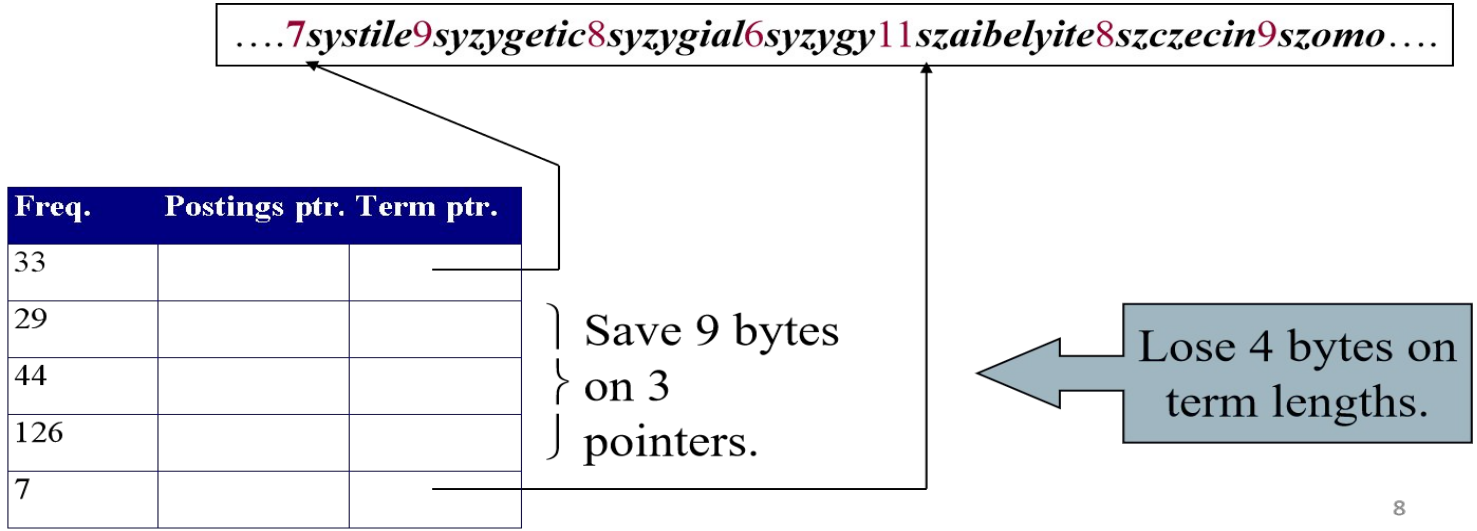
Space for dictionary as a string

- 4 bytes per term for Freq.
- 4 bytes per term for pointer to Postings.
- 3 bytes per term pointer
- Avg. 8 bytes per term in term string
- 400K terms x 19 = 7.6 MB (against 11.2MB for fixed width)

} Now avg. 11 bytes/term, not 20.

Blocking

- Store pointers to every kth term string.
  - Example below: k=4.
- Need to store term lengths (1 extra byte)



Net

- Example for block size k = 4
- Where we used 3 bytes/pointer without blocking
    - 3 x 4 = 12 bytes,

now we use 3 + 4 = 7 bytes.

For each word -□ 28 □ 19 □ 17.75

Saved another ~0.5MB. This reduces the size of the dictionary from 7.6 MB to 7.1 MB. We can save more with larger k.

Why not go with larger k?

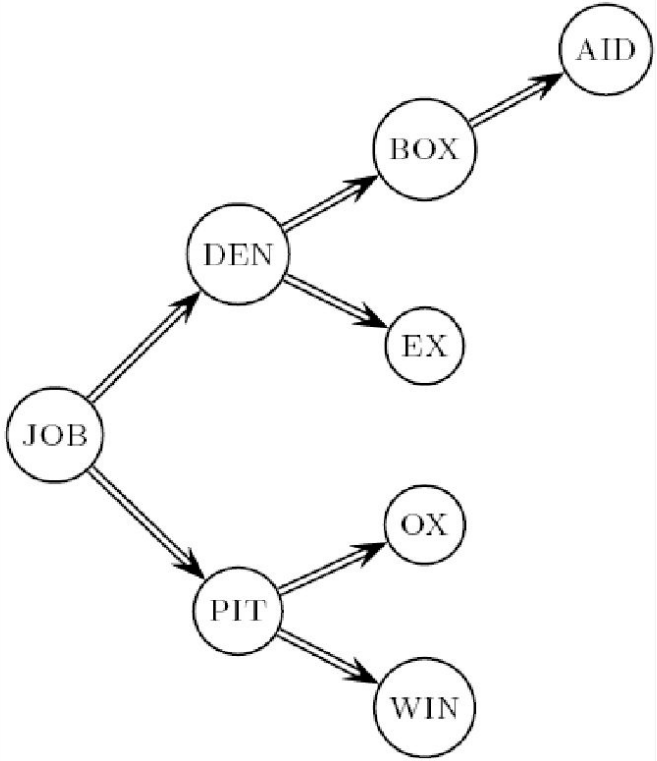
Exercise

- Estimate the space usage (and savings compared to 7.6 MB) with blocking, for block sizes of k = 4, 8 and 16.

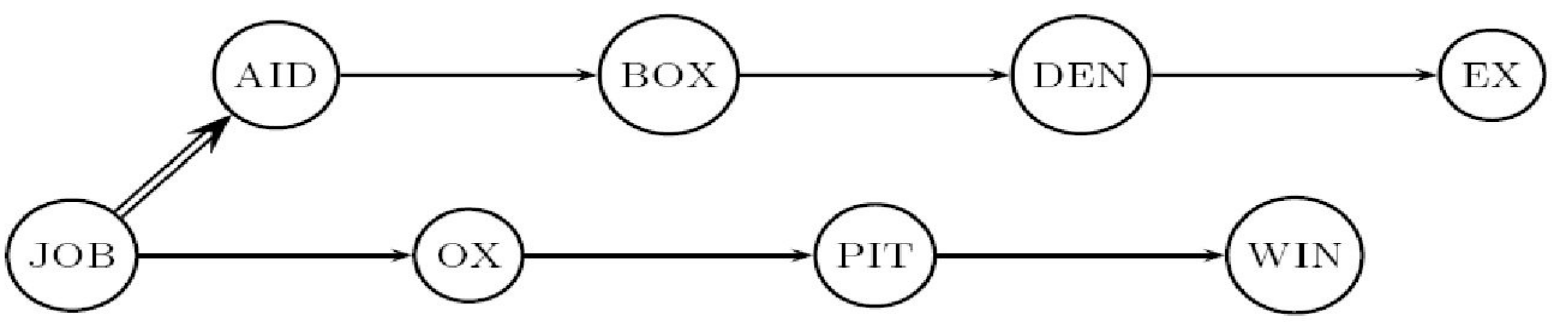
Dictionary search without blocking

- Assuming each dictionary term equally likely in query (not really so in practice!), average number of comparisons = (1+2·2+4·3+4)/8 ~2.6

Exercise: what if the frequencies of query terms were non-uniform but known, how would you structure the dictionary search tree?



Dictionary search with blocking



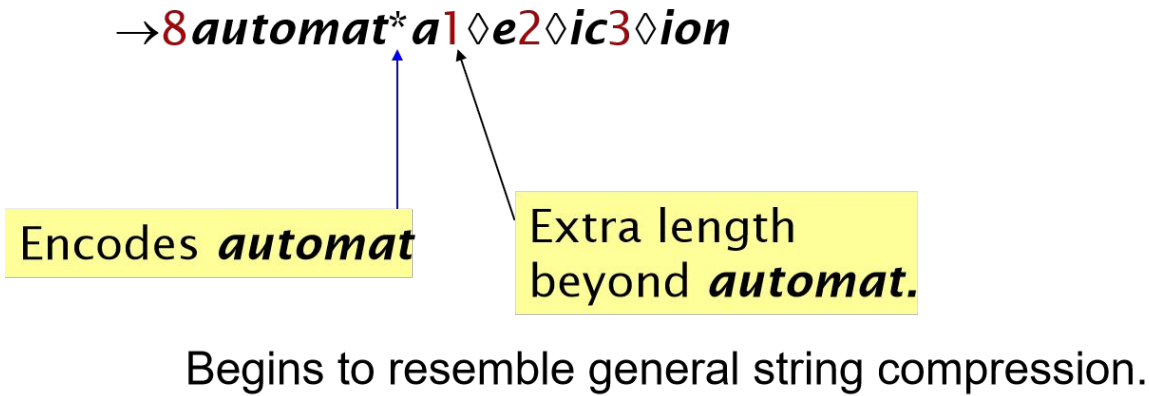
- Binary search down to 4-term block;
  - Then linear search through terms in block.
- Blocks of 4 (binary tree)
  - avg. = (1+2·2+2·3+2·4+5)/8 = 3 comparison

Exercise

- Estimate the impact on search performance (and slowdown compared to k=1) with blocking, for block sizes of k = 4, 8 and 16.

Front coding

- Front-coding:
  - Sorted words commonly have long common prefix – store differences only
  - (for last k-1 in a block of k)
    8automata**8**automate**9**automatic**10**automation



Begins to resemble general string compression.

RCV1 dictionary compression summary

Technique	Size in MB
Fixed width	11.2
Dictionary-as-String with pointers to every term	7.6
Also, blocking $k = 4$	7.1
Also, Blocking + front coding	5.9