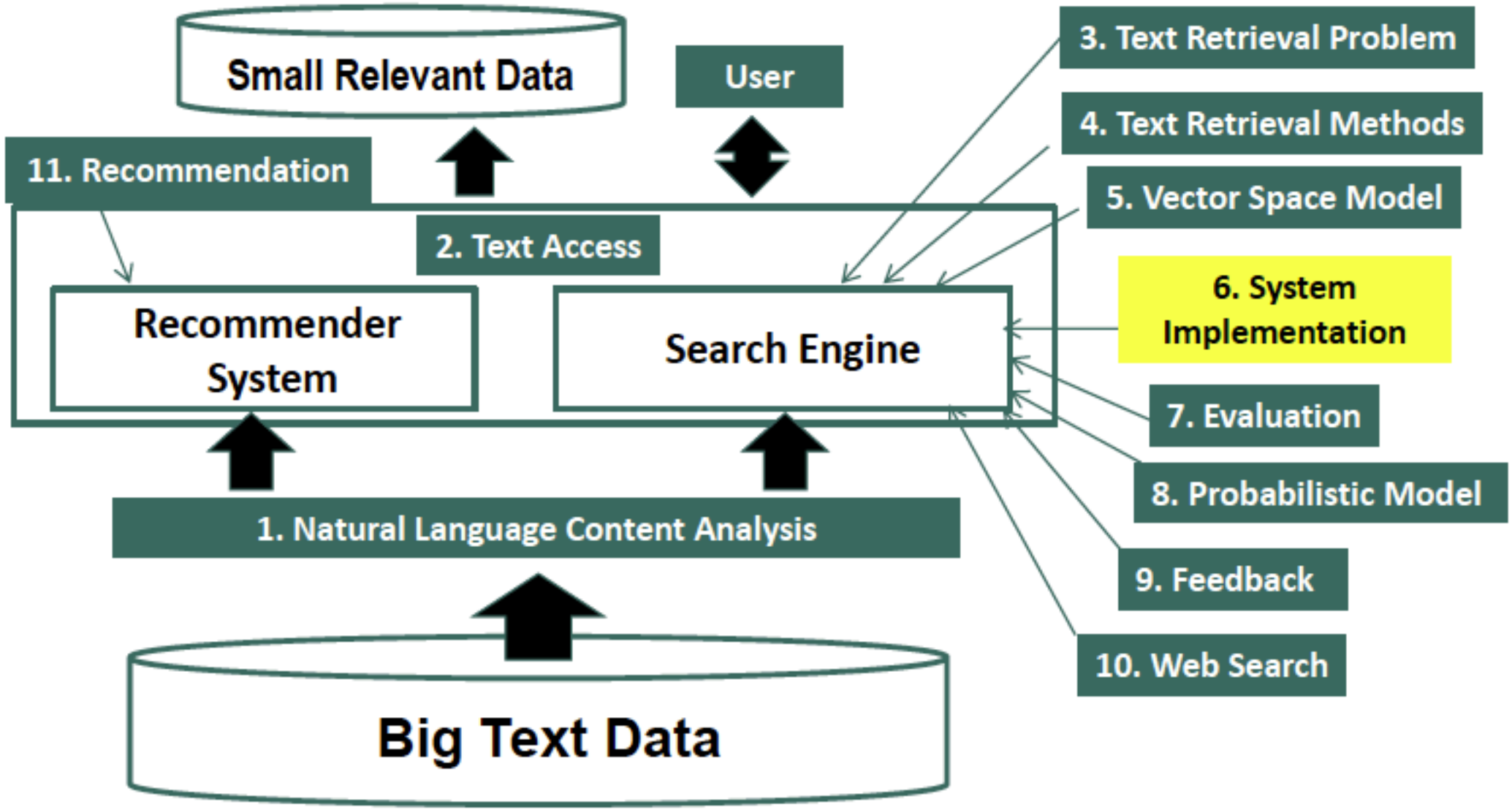
Information Retrieval

System Implementation: Inverted Index Compression

Dr. Iqra Safder

Implementation of Text Retrieval Systems



Inverted Index Example

doc 1

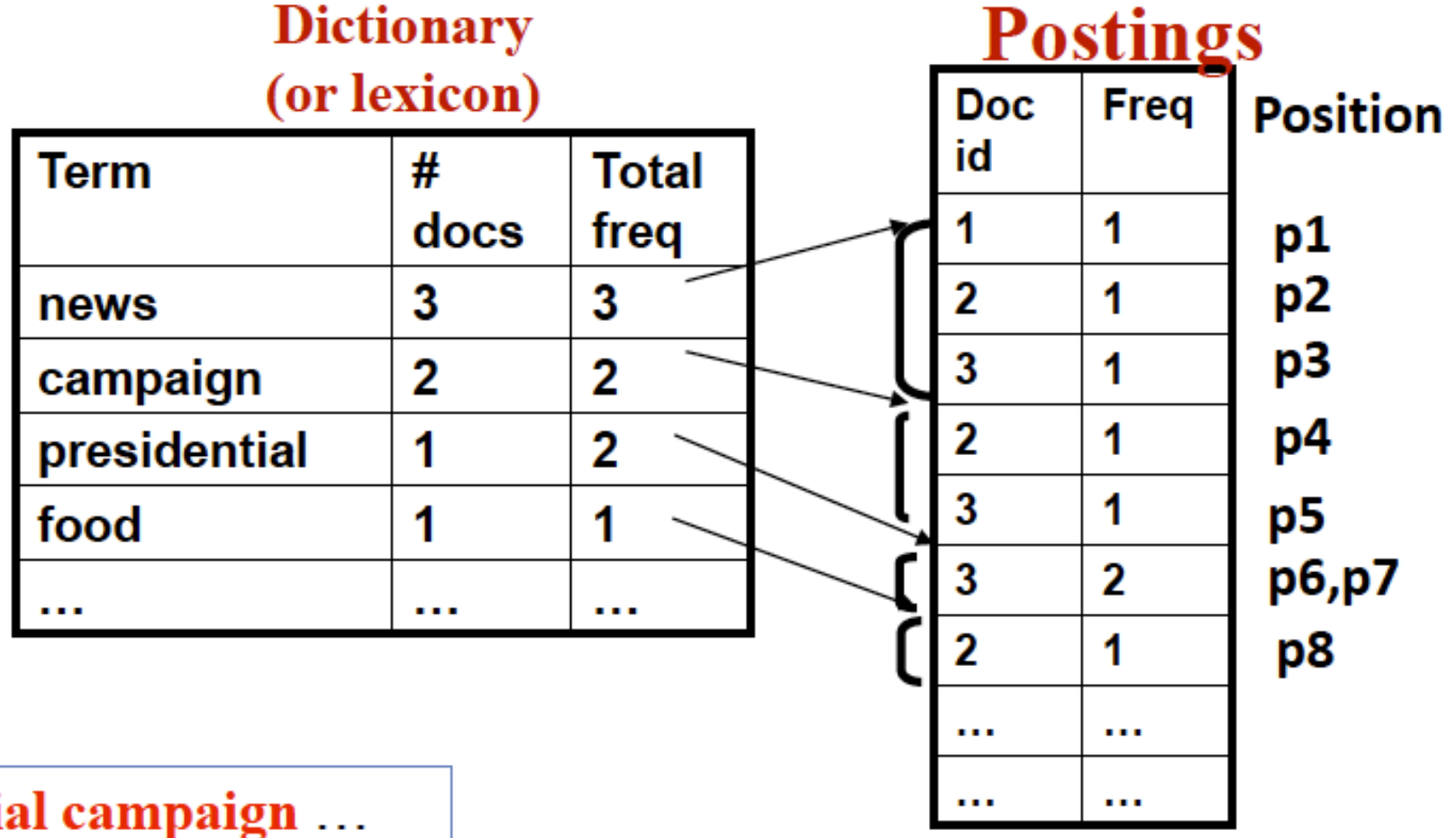
… **news about**

doc 2

… **news about** organic food **campaign**…

doc 3

… **news** of **presidential campaign** …
… **presidential** candidate …



IDF is calculated using # docs

Inverted Index Compression

- In general, leverage skewed distribution of values and use variable-length encoding
- TF compression
  - Small numbers tend to occur far more frequently than large numbers (why?)
  - Fewer bits for small (high frequency) integers at the cost of more bits for large integers
- Doc ID compression
  - "d-gap" (store difference): d1, d2-d1, d3-d2,…
  - Feasible due to sequential access
- Methods: Binary code, unary code, γ-code, δ-code, …

GAP Encoded Compression

It is important that all of our compression methods need to support random access decoding; that is, we could like to seek to a particular position in the postings file and start decompressing without having to decompress all the previous data.

$$\{23, 25, 34, 35, 39, 43, 49, 51, 57, 59, \ldots\}.$$

$$\{23, 2, 9, 1, 4, 4, 6, 2, 6, 2, \ldots\}.$$

To get the actual document ID values, simply add the offset to the previous value. So the first ID is 23 and the second is $23 + 2 = 25$. The third is $25 + 9 = 34$, and so on.

With bitwise compression, instead of writing out strings representing numbers (like "1624"), or fixed byte-width chunks (like a 4-byte integer as "00000658"), we are writing raw binary numbers. When the representation ends, the next number begins. There is no fixed width, or length, of the number representations. Using bitwise compression means performing some bit operations for every bit that is encoded in order to "build" the compressed integer back into its original form.

Unary. Unary encoding is the simplest method. To write the integer k, we simply write $k-1$ zeros followed by a one. The one acts as a delimiter and lets us know when to stop reading:

$$\begin{aligned} 1 &\to 1 \\ 2 &\to 01 \\ 3 &\to 001 \\ 4 &\to 0001 \\ 5 &\to 00001 \\ 19 &\to 0000000000000000001 \end{aligned}$$

Note that we can't encode the number zero—this is true of most other methods as well. An example of a unary-encoded sequence is

$$0001001000100000001010001000001 = 4, 3, 4, 8, 2, 4, 5.$$

Gamma. To encode a number with γ-encoding, first simply write the number in binary. Let k be the number of bits in your binary string. Then, prepend $k-1$ zeros to the binary number:

$$
\begin{aligned}
1 &\rightarrow 1 \\
2 &\rightarrow 010 \\
3 &\rightarrow 011 \\
4 &\rightarrow 00100 \\
5 &\rightarrow 00101 \\
19 &\rightarrow 000010011 \\
47 &\rightarrow 00000101111
\end{aligned}
$$

To decode, read and count k zeros until you hit a one. Read the one and additional k bits in binary. Note that all γ codes will have an odd number of bits.

Delta. In short, δ-encoding is γ-encoding a number and then γ-encoding the unary prefix (including the one):

$$
\begin{aligned}
1 &\rightarrow 1 \rightarrow 1 \\
2 &\rightarrow 010 \rightarrow 0100 \\
3 &\rightarrow 011 \rightarrow 0101 \\
4 &\rightarrow 00100 \rightarrow 01100 \\
5 &\rightarrow 00101 \rightarrow 01101 \\
19 &\rightarrow 000010011 \rightarrow 001010011 \\
47 &\rightarrow 00000101111 \rightarrow 0011001111
\end{aligned}
$$

To decode, decode the γ code at your start position to get an integer k. Write a one, and then read the next $k+1$ bits in binary (including the one you wrote). As you can see, the δ compression at first starts off to have more bits than the γ encoding, but eventually becomes more efficient as the numbers get larger. It probably depends on the particular dataset (the distribution of integers) as to which compression method would be better in terms of **compression ratio**. A compression ratio is simply the uncompressed size divided by the compressed size. Thus, a compression ratio of 3 is better (in that the compressed files are smaller) than a compression ratio of 2.

HOME TASK

- Understand the slide#17