Index Compression

Debapriyo Majumdar
Information Retrieval
Indian Statistical Institute Kolkata

Data compression

Represent the information using (hopefully) lesser number of bits than used by the original representation

- Lossy: may lose some "information"
  - Consequently, may not be able to get back the original
- Lossless: just a different representation, will be able to "decompress" and get back the original
  - We will consider only lossless compression now

Question:

Can a lossless compression algorithm "compress" any input data?

Index compression

- Use less space to store
  - In disk or in memory
  - Saving is saving anyway
- Dictionary compression
  - May store the full or most of the dictionary in memory
- Posting list compression
  - Assumption: cannot store it in memory
  - Transfer data from disk faster
  - Typically can compress index by 4 times
- Need a fast decompression
  - Without compression: transfer $4x$ bytes from disk
  - With compression: transfer x bytes and decompress

The posting lists

zuckerberg	…	3429934	3478390	3689839	…
the	…	45	46	47	…
ashish	…	54892834	394099834	828493832	…

- For a "small" dataset, need 4 bytes (or even less) to store the doc IDs
- For a "very large" dataset, may need ~8 bytes to store the doc IDs
- Observation 1: the docIDs are large integers, but the gaps are not so large!
  - Idea: Store the first doc ID, and then store only the gaps (**Gap encoding**)
- Observation 2(a): for some posting lists, the gaps are very small
  - Very frequent terms
- Observation 2(b): For some the gaps can be large as well
  - Very infrequent terms
- Some posting lists may also have some large and some small gaps
  - Fixed size for gaps would not work!

Gap encoding: variable byte encoding

doc IDs	824	829	215406
ID, gaps	824	5	214577
encoding	00000110 10111000	10000101	00001101 00001100 10110001

First indicator bit is 0 because this is not the last byte

824 is encoded as 0000110 0111000 [without the first bit of both bytes] (512 + 256 + 32 + 16 + 8)

Some gaps take more bytes, some gaps take less

- Approach: store the first doc ID, then store the gaps
- Use 1 byte units
- First bit is the indicator, next 7 bits are payload (the number)
- First bit is 1 if this is the last byte, 0 otherwise

Elias γ encoding

- Unary code: $n \mapsto n$ 1s followed by a 0
  - For example $3 \mapsto 1110$
  - Some conventions use n 0s followed by a 1 (equivalent)
- γ encoding (due to Peter Elias)
  - Consider the binary representation of n
  - It starts with a 1; chop it off (we know it, so redundant)
  - Example: $9 \mapsto 1001 \mapsto 001$ (call this part offset)
  - Encode the length of the offset in unary
  - Example: length of offset of 9 is 3. Unary encoding: 1110
  - γ encoding of $9 \mapsto \underbrace{1110}_{\text{length of offset is 3 in unary}}\underbrace{001}_{\text{the number is leading 1 and the offset = 1001}}$

Elias γ encoding

- Better than variable byte encoding because γ encoding is bit-level encoding (empirically 15% better in posting lists)
- Decompression is less efficient than variable byte encoding
- Observation: the unary encoding of length may use a lot of space
- Solution: encode length + 1 using γ encoding again (Elias δ encoding)
  - More effective for large numbers
  - Example:
    509 ↦ 11111111011111101 (γ) ↦ 11100011111101 (δ)
- Question: why length + 1?
- Omega encoding: recursively apply γ encoding on the length

References

- Primarily: IR Book by Manning, Raghavan and Schuetze: http://nlp.stanford.edu/IR-book/