Inverted Indexes: Compression

Yufei Tao

KAIST

March 26, 2013

The inverted index we have learned has a defect: its size can be very large such that it may not fit in memory. In this case, a part of the index will have to be stored in the disk, and thus, query processing can incur (disk) I/Os. A successful search engine aims at completely eliminating I/Os in a query, because I/Os are significantly more expensive than memory accesses.

Fortunately, the inverted index is friendly to compression. In this lecture, we will learn several compression techniques that can make inverted indexes much smaller.

Review

An inverted index consists of:

- For every term w_i in $DICT$, the value of $idf(w_i)$.
- For every term w_i in $DICT$, an inverted list, denoted as $list(w_i)$, which contains a pair

  $$(i, tf(D_i, w_i))$$

  for every document D_i that contains w_i.

We will refer to i as the *document id* of D_i.

Example (Excerpted from [Zobel and Moffat, 2006]

Suppose that our document collection is:

document ID	content
1	the old night keeper keeps the keep in the town
2	in the big old gown in the big old house
3	the house in the town had the big old keep
4	where the old night keeper never did sleep
5	the night keeper keeps the keep in the night
6	and keeps in the dark and sleeps in the light

Example (Excerpted from [Zobel and Moffat, 2006]

term w	inverted list for w
and	(6, 2)
big	(2, 2), (3, 1)
dark	(6, 1)
did	(4, 1)
gown	(2, 1)
had	(3, 1)
house	(2, 1), (3, 1)
in	(1, 1), (2, 2), (3, 1), (5, 1), (6, 2)
keep	(1, 1), (3, 1), (5, 1)
keeper	(1, 1), (4, 1), (5, 1)
keeps	(1, 1), (5, 1), (6, 1)
light	(6, 1)
never	(4, 1)
night	(1, 1), (4, 1), (5, 2)
old	(1, 1), (2, 2), (3, 1), (4, 1)
sleep	(4, 1)
sleeps	(6, 1)
the	(1, 3), (2, 2), (3, 3), (4, 1), (5, 3), (6, 2)
town	(1, 1), (3, 1)
where	(4, 1)

Today, an integer typically requires 64 bits to store (e.g., in C++). Thus, an inverted list like:

$$\text{the: } (1, 3), (2, 2), (3, 3), (4, 1), (5, 3), (6, 2)$$

requires $12 \times 64 = 768$ bits.

In a computer, an integer is represented in binary. In other words, an integer $x \geq 1$ requires at least $\lceil \log_2(x+1) \rceil$ bits to store. Thus, pair (1, 3) in the inverted list of the previous slide could be stored as a bit string that starts with a 1 (for number 1), followed by 11 (for number 3). This storage method uses the minimum number of bits (i.e., 3 bits). But does it really work?

Think

The answer is no. Given a bit string 111, how would you decompress it into a pair (1, 3)?

Next, we will learn some effective encoding schemes that use a small number (but more than $\lceil \log_2(x+1) \rceil$) of bits to represent x

Elias' Gamma Code

Let x be an integer at least 1. Elias' gamma code represents x as follows. First, let y be the largest power of 2 that is at most x, and $z = x - y$. Then, x is represented by a bit string $gamma(x)$:

- $gamma(x)$ starts with $\log_2 y$ 1's followed by a single 0.
- Then, $gamma(x)$ continues with $\log_2 y$ bits representing z in binary.

Example

- Consider $x = 13$. Then, $y = 8$ and $z = 5$. Hence, $gamma(13) = 1110101$.
- As another example, consider $x = 57$. Then, $y = 32$ and $z = 25$. Hence, $gamma(57) = 11111011001$.

Elias' Gamma Code

Given a bit string $gamma(x)$, we can decompress it back to x as follows

1. Search for the first 0 in $gamma(x)$.
2. Count how many 1's there are before the 0. Let the number by b.
3. Then, $y = 2^b$.
4. Let z be the number represented by the next b bits of $gamma(x)$ in binary.
5. $x = y + z$.

Think

Convince yourself that 1110101111111011001 decompresses into two numbers: 13 and then 57.

Lemma

Let $\ell = \lceil \log_2(x+1) \rceil$. Then, *gamma*$(x)$ has $2\ell - 1$ bits.

Elias' gamma code has some variants. A well-known example is Elias' delta code.

Elias' Delta Code

Let x be an integer at least 1. Elias' delta code represents x as follows. First, let y be the largest power of 2 that is at most x, and $z = x - y$. Then, x is represented by a bit string $delta(x)$ decided as follows:

- $delta(x)$ starts with $gamma(1 + \log_2 y)$.
- Then, $delta(x)$ continues with $\log_2 y$ bits representing z in binary.

Example

- Consider $x = 13$. Then, $y = 8$, $z = 5$, and $gamma(1 + \log_2 y) = 11000$. Hence, $delta(13) = 11000101$.
- As another example, consider $x = 57$. Then, $y = 32$, $z = 25$, and $gamma(1 + \log_2 32) = 11010$. Hence, $delta(57) = 1101011001$.

Elias' Delta Code

Think

Convince yourself that 11000101111010011001 decompresses into 13 and then 57.

Lemma

Let $\ell = \lceil \log_2(x+1) \rceil$. Then, *delta*$(x)$ has $2\lceil \log_2(\ell+1) \rceil + \ell - 2$ bits.

The delta code uses fewer bits than the gamma code for all $x \geq 32$.

Now, we are ready to explain how to compress an inverted list. Recall that the list is a set of (id, frequency) pairs. For example:

$$(6, 2), (4, 1), (2, 2), (3, 3), (1, 3), (5, 3)$$

In practice, frequency values are small (i.e., most words appear just a few times in a document). They are stored directly using Elias' gamma or delta code.

IDs, on the other hand, are typically large integers. Of course, we can represent each ID again using Elias' gamma or delta code, but there is a way to do better.

Let us sort all the pairs in an inverted list by id:

$$(1, 3), (2, 2), (3, 3), (4, 1), (5, 3), (6, 2)$$

Let (id_i, f_i) the i-th pair for $i \geq 1$. For each $i \geq 2$, instead of (id_i, f_i), we store:

$$(id_i - id_{i-1}, f_i)$$

namely, we store only the difference between id_i and id_{i-1}.

For example, the above inverted list will be stored as:

$$(1, 3), (1, 2), (1, 3), (1, 1), (1, 3), (1, 2)$$

Finally, all the numbers in this final sequence of pairs are stored in Elias' gamma/delta code.

Think

Convince yourself that we can decompress the resulting bit sequence into the original inverted list precisely.