COMP6714 Lecture 1

Information Retrieval and Search Engines

Lecturer: Wei Wang *Date:*

1 Galloping Search

1.1 Description

Algorithm 1: skipTo(x)

Data: cur is the position of the current docID under the cursor in the posting list L. $L[pos]$ returns the docID at postion pos.

`/* Stage 1: Find a range where` x `will appear by trying exponentially larger search range (from the current cursor position) */`

1 $\Delta \leftarrow 1$;
2 **while** $cur + \Delta \leq |L|$ **and** $L[cur + \Delta] < x$ **do**
3 $\quad \Delta \leftarrow 2 \cdot \Delta$;

`/* Stage 2: (Binary) search within` $[cur + \delta, cur + \Delta]$ `*/`

4 **if** $cur + \Delta > |L|$ **then**
5 $\quad$ **return** `NO_MORE_DOC`;
6 **else**
7 $\quad \delta \leftarrow \lfloor \Delta/2 \rfloor$;
8 $\quad cur =$ bsearch$(x, L[cur + \delta], \Delta - \delta + 1,$ `SIZE_OF_POSTING`$)$;
9 $\quad$ **return** $L[cur]$;

1.2 Cost Analysis

Assume the answer is n positions away. Define

$$f(i) := 1 + 2 + 4 + \ldots + 2^i = 2^{i+1} - 1$$

When first stage ends after i steps, we know that $f(i-1) < n \leq f(i)$, and the cost is i. We can derive [1]

$$\begin{aligned} & f(i-1) < n \leq f(i) \\ \Longleftrightarrow & 2^i < n + 1 \leq 2^{i+1} \\ \Longleftrightarrow & i < \lg(n+1) \leq i + 1 \end{aligned}$$

The search range in the second stage is $2^i - 2$ (two end-points have already been examined), and the worst case cost is $\lceil \lg(2^i - 2) \rceil \leq i$.

So the total cost is $2i \leq 2\lg(n+1) = O(\log(n))$.

2 Skip Pointers

2.1 Motivation

In order to support efficient query processing (e.g., conjunctive queries in Boolean model and DAAT query processing in Vector Space Model), we need to support the `skipTo(x)` operation, which moves the cursor

[1] $\lg(x) := \log_2(x)$.

on a postings list to the first posting whose docID is equal to or larger than x.[2] Sequential scan is a naive implementation, but is obviously insufficient to deliver high performance.

If we do not have additional index built on the postings lists, we can use binary search or galloping search, assuming postings are not compressed.[3] The former has a cost of $O(\log(n))$, and the latter has a cost of $O(\log(\Delta))$, where Δ is the positional difference between the current cursor and the cursor returned from the `skipTo()` call.

2.2 Idea

There are several reasons skip pointers are built for long posting lists.

- They can help skipping to the first docID that satisified certain criterion using less cost than alternative methods (without indexing) (e.g., binary search or galloping search).
- This is especially helpful when the posting lists are compressed (as always in production systems). Binary or galloping search cannot be performed easily or will incur considerable cost (I/O and CPU due to decompression).

Note that

- If the posting list is short, skip pointers are not needed.
- If the posting list is very long, we may build multiple-levels of skip pointers [Moffat and Zobel, 1996].[4]

2.3 Implementation

Skip pointers can be placed at the beginning of the postings list or embedded in the postings list. The postings list consists of multiple blocks. A skip pointer for the i-th block consists of $(docID, ptr)$, where $docID$ is the first entry in the i-th block and ptr is the pointer to the first docID in the i-th block.

[Moffat and Zobel, 1996] proposes the compressed self-indexing inverted index, where both skip pointers and posting lists are compressed. It proposes to place pointers strictly every $O(\sqrt{n})$ docIDs, so that the ptr actually needs to point to a particular *bit* (rather than byte or DWORD). In addition, the skip pointers are also compressed using gap + golomb encoding.

Recently, there is a proposal to use (compressed perfect) skip lists instead [Boldi and Vigna, 2005], which is reported to deliver an increase in query processing speed between 20% to 300% with respect to the classic (i.e., square-root spaced) approach.

According to `https://issues.apache.org/jira/browse/LUCENE-866`, multi-level skipping can achieve further saving in time and I/O than the single-level skipping.

2.4 Optimization

Traditional wisdom says approximately $O(\sqrt{n})$ skip pointers are needed for a posting of n entries [Moffat and Zobel, 1996]. This is based on the following reasoning:

- Let the posting list consist of n postings, and we build skip pointers for every x-th docID. Let c be the average cost of decoding a compressed value and performing a comparison.
- Since skip pointers are compressed, we need to, on average, decode and access half of them for each `skipTo()` call. The cost is $c \cdot 2(n/x)$.

[2] c.f., Lucene's `TermDocs::skipTo(target)`.

[3] We can still perform the above types of search on block level even if posting lists are compressed though.

[4] Lucene uses multiple level skipping lists. c.f., `http://lucene.apache.org/core/3_6_1/fileformats.html`, and `http://lucene.apache.org/core/4_0_0-BETA/core/org/apache/lucene/codecs/MultiLevelSkipListWriter.html`

- Once we locate a block, we need to decode and search within the block. The cost is $c \cdot x$.[5]
- Total cost is $c \cdot \left(\frac{2n}{x} + x\right)$. The total cost is minimized when $x = \sqrt{2n}$.

[6]

2.5 Time Complexity

If we place skip pointers at $O(\sqrt{n})$ intervals, the cost of a `skipTo()` call is $O(\sqrt{n})$.

References

[Boldi and Vigna, 2005] Boldi, P. and Vigna, S. (2005). Compressed perfect embedded skip lists for quick inverted-index lookups. In *SPIRE*, pages 25–28.

[Moffat and Zobel, 1996] Moffat, A. and Zobel, J. (1996). Self-indexing inverted files for fast text retrieval. *ACM Trans. Inf. Syst.*, 14(4):349–379.

[5] In [Moffat and Zobel, 1996], each posting consists of $\langle docID, tf \rangle$. So the cost need to be doubled.

[6] The above analysis ignores factors such as caching the decoded skip pointers, and the extra time needed to read in the skip pointers. However, these does not substantially change the conclusion here.