Rank and Select Operations on Binary Strings (1974; Elias)

Naila Rahman, University of Leicester, www.cs.le.ac.uk/~nyr1
Rajeev Raman, University of Leicester, www.cs.le.ac.uk/~rraman

entry editor: Paolo Ferragina

INDEX TERMS: Succinct data structures, bit-vectors, predecessor search, sets, multisets.

Synonyms: Binary bit-vector, compressed bit-vector, rank and select dictionary, fully indexable dictionary (FID).

1 PROBLEM DEFINITION

Given a static sequence $\boldsymbol{b} = b_1 \dots b_m$ of m bits, to preprocess the sequence and to create a space-efficient data structure that supports the following operations rapidly:

$\mathsf{rank}_1(i)$ takes an index i as input, $1 \leq i \leq m$, and returns the number of **1**s among $b_1 \dots b_i$.

$\mathsf{select}_1(i)$ takes an index $i \geq 1$ as input, and returns the position of the i-th **1** in $\boldsymbol{b}$, and -1 if i is greater than the number of **1**s in $\boldsymbol{b}$.

The operations rank_0 and select_0 are defined analogously for the **0**s in $\boldsymbol{b}$. As $\mathsf{rank}_0(i) = i - \mathsf{rank}_1(i)$, one considers just rank_1 (abbreviated to rank), and refers to select_0 and select_1 collectively as select. In what follows, $|\boldsymbol{x}|$ denotes the length of a bit sequence $\boldsymbol{x}$ and $w(\boldsymbol{x})$ denotes the number of **1**s in it. $\boldsymbol{b}$ is always used to denote the input bit sequence, m to denote $|\boldsymbol{b}|$ and n to denote $w(\boldsymbol{b})$.

Models of Computation, Time and Space Bounds. Two models of computation are commonly considered. One is the *unit-cost RAM* model with word size $O(\lg m)$ bits [1]. The other model, which is particularly useful for proving lower bounds, is the *bit-probe* model, where the data structure is stored in bit-addressable memory, and the complexity of answering a query is the worst-case number of bits of the data structure that are probed by the algorithm to answer that query. In the RAM model, the algorithm can read $O(\lg m)$ *consecutive* bits in one step, so supporting all operations in $O(1)$ time on the RAM model implies a solution that uses $O(\lg m)$ bit-probes, but the converse is not true.

This entry considers three variants of the problem: in each variant, rank and select must be supported in $O(1)$ time on the RAM model, or in $O(\lg m)$ bit-probes. However, the use of memory varies:

Problem 1 (Bit-Vector). *The overall space used must be $m + o(m)$ bits.*

Problem 2 (Bit-Vector Index). $\boldsymbol{b}$ *is given in* read-only *memory and the algorithm can create auxiliary data structures (called* indices*) which must use $o(m)$ bits.*

Indices allow the representation of $\boldsymbol{b}$ to be de-coupled from the auxiliary data structure, e.g., $\boldsymbol{b}$ can be stored (in a potentially highly compressed form) in a data structure such as that of

[6, 9, 10] which allows access to $O(\lg m)$ consecutive bits of $\boldsymbol{b}$ in $O(1)$ time on the RAM model. Most bit-vectors developed to date are bit-vector indices.

Recalling that $n = w(\boldsymbol{b})$, observe that if m and n are known to an algorithm, there are only $l = \binom{m}{n}$ possibities for $\boldsymbol{b}$, so an information-theoretically optimal encoding of $\boldsymbol{b}$ would require $B(m,n) = \lceil \lg l \rceil$ bits (it can be verified that $B(m,n) < m$ for all m, n). The next problem is:

Problem 3 (Compressed Bit-Vector). *The overall space used must be $B(m,n) + o(n)$ bits.*

It is helpful to understand the asymptotics of $B(m,n)$ in order to appreciate the difference between the bit-vector and the compressed bit-vector problems:

- Using standard approximations of the factorial function, one can show [15] that:

$$B(m,n) = n \lg(m/n) + n \lg e + O(n^2/m) \tag{1}$$

  If $n = o(m)$, then $B(m,n) = o(m)$, and if such a sparse sequence $\boldsymbol{b}$ were represented as a compressed bit-vector, then it would occupy $o(m)$ bits, rather than $m + o(m)$ bits.

- $B(m,n) = m - O(\lg m)$, whenever $|m/2 - n| = O(\sqrt{m \lg m})$. In such cases, a compressed bit-vector will take about the same amount of space as a bit-vector.

- Taking $p = n/m$, $H_0(\boldsymbol{b}) = (1/p)\lg(1/p) + (1/(1-p))\lg(1/(1-p))$ is the *empirical zeroth-order entropy* of $\boldsymbol{b}$. If $\boldsymbol{b}$ is compressed using an 'entropy' compressor such as non-adaptive arithmetic coding [18], the size of the compressed output is at least $mH_0(\boldsymbol{b})$ bits. However, $B(m,n) = mH_0(\boldsymbol{b}) - O(\log m)$. Applying Golomb coding to the 'gaps' between successive **1**s, which is the best way to compress bit sequences that represent inverted lists [18], also gives a space usage close to $B(m,n)$ [4].

Related Problems. Viewing $\boldsymbol{b}$ as the characteristic vector of a set $S \subseteq U = \{1, \ldots, m\}$, note that the well-known *predecessor* problem — given $y \in U$, return $pred(y) = \max\{z \in S | z \leq y\}$ — may be implemented as $\mathsf{select}_1(\mathsf{rank}_1(y))$. One may also view $\boldsymbol{b}$ as a *multiset* of size $m - n$ over the universe $\{1, \ldots, n+1\}$ [5]. First, append a **1** to $\boldsymbol{b}$. Then, take each of the $n+1$ **1**s to be the elements of the universe, and the number of consecutive **0**s immediately preceding a **1** to indicate their multiplicities. For example, $\boldsymbol{b} = \mathbf{01100100}$ maps to the multiset $\{1, 3, 3, 4, 4\}$. Seen this way, $\mathsf{select}_1(i) - i$ on $\boldsymbol{b}$ gives the number of items in the multiset that are $\leq i$, and $\mathsf{select}_0(i) - i + 1$ gives the value of the i-th element of the multiset.

Lower-Order Terms. From an asymptotic viewpoint, the space utilization is dominated by the main terms in the space bound. However, the second (apparently lower-order) terms are of interest for several reasons, primarily because the lower-order terms are extremely significant in determining practical space usage, and also because non-trivial space bounds have been proven for the size of the lower-order terms.

2 KEY RESULTS

2.1 Reductions

It has been already noted that rank_0 and rank_1 reduce to each other, and that operations on multisets reduce to select operations on a bit sequence. Some other reductions, whereby one can support operations on $\boldsymbol{b}$ by performing operations on bit sequences derived from $\boldsymbol{b}$ are:

Theorem 1. *(a)* rank *reduces to* select_0 *on a bit sequence $\boldsymbol{c}$ such that $|\boldsymbol{c}| = m + n$ and $w(\boldsymbol{c}) = n$.*

(b) *If* $\boldsymbol{b}$ *has no consecutive* $\mathbf{1}$*s, then* select_0 *on* $\boldsymbol{b}$ *can be reduced to* rank *on a bit sequence* $\boldsymbol{c}$ *such that* $|\boldsymbol{c}| = m - n$ *and* $w(\boldsymbol{c})$ *is either* $n - 1$ *or* n*.*

(c) *From* $\boldsymbol{b}$ *one can derive two bit sequences* $\boldsymbol{b_0}$ *and* $\boldsymbol{b_1}$ *such that* $|\boldsymbol{b_0}| = m - n$*,* $|\boldsymbol{b_1}| = n$*,* $w(\boldsymbol{b_0}), w(\boldsymbol{b_1}) \leq \min\{m - n, n\}$ *and* select_0 *and* select_1 *on* $\boldsymbol{b}$ *can be supported by supporting* select_1 *and* rank *on* b_0 *and* b_1*.*

Parts (a) and (b) follow from Elias's observations on multiset representations (see the "Related Problems" paragraph), specialised to sets. For part (a), create $\boldsymbol{c}$ from $\boldsymbol{b}$ by adding a $\mathbf{0}$ after every $\mathbf{1}$. For example, if $\boldsymbol{b} = \mathbf{01100100}$ then $\boldsymbol{c} = \mathbf{01}\boldsymbol{0}\mathbf{1}\boldsymbol{0}\mathbf{00}\boldsymbol{1}\boldsymbol{0}\mathbf{00}$. Then, $\mathsf{rank}_1(i)$ on $\boldsymbol{b}$ equals $\mathsf{select}_0(i) - i$ on $\boldsymbol{c}$. For part (b), essentially invert the mapping of part (a). Part (c) is shown in [3].

2.2 Bit-Vector Indices

Theorem 2 ([8]). *There is an index of size* $(1 + o(1))(m \lg \lg m / \lg m) + O(m / \lg m)$ *that supports* rank *and* select *in* $O(1)$ *time on the RAM model.*

Elias previously gave an $o(m)$-bit index that supported select in $O(\lg m)$ bit-probes on average (where the average was computed across all select queries). Jacobson gave $o(m)$-bit indices that supported rank and select in $O(\lg m)$ bit-probes in the worst case. Clark and Munro [2] gave the first $o(m)$-bit indices that support both rank and select in $O(1)$ time on the RAM. A matching lower bound on the size of indices has also been shown (this also applies to indices which support rank and select in $O(1)$ time on the RAM model):

Theorem 3 ([8]). *Any index that allows* rank *or* select_1 *to be supported in* $O(\lg m)$ *bit-probes has size* $\Omega(m \lg \lg m / \lg m)$ *bits.*

2.3 Compressed Bit-Vectors

Theorem 4. *There is a compressed bit-vector that uses:*

(a) $B(m, n) + O(m \lg \lg m / \lg m)$ *bits and supports* rank *and* select *in* $O(1)$ *time.*

(b) $B(m, n) + O(n(\lg \lg n)^2 / \lg n)$ *bits and supports* rank *in* $O(1)$ *time, when* $n = m/(\lg m)^{O(1)}$*.*

(c) $B(m, n) + O(n \lg \lg n / \sqrt{\lg n})$ *bits and supports* select_1 *in* $O(1)$ *time.*

Theorem 4(a) and (c) were shown by Raman et al. [17] and Theorem 4(b) by Pagh [15]. Note that Theorem 4(a) has a lower-order term that is $o(m)$, rather than $o(n)$ as required by the problem statement. As compressed bit-vectors must represent $\boldsymbol{b}$ compactly, they are not bit-vector indices, and the lower bound of Theorem 3 does not apply to compressed bit-vectors. Coarser lower bounds are obtained by reduction to the predecessor problem on sets of integers, for which tight upper and lower bounds in the RAM model are now known. In particular the work of [16] implies:

Theorem 5. *Let* $U = \{1, \ldots, M\}$ *and let* $S \subseteq U$*,* $|S| = N$*. Any data structure on a RAM with word size* $O(\lg M)$ *bits that occupies at most* $O(N \lg M)$ *bits of space can support predecessor queries on* S *in* $O(1)$ *time only when* $N = M/(\lg M)^{O(1)}$ *or* $N = (\lg M)^{O(1)}$*.*

As noted in the paragraph "Related Problems", the predecessor problem can be solved by the use of rank and select_1 operations. Thus, Theorem 5 has consequences for compressed bit-vector data structures, which are spelt out below:

Corollary 1. *There is no data structure that uses* $B(m, n) + o(n)$ *bits and supports either* rank *or* select_0 *in* $O(1)$ *time unless* $n = m/(\lg m)^{O(1)}$*, or* $n = (\log m)^{O(1)}$*.*

Given a set $S \subseteq U = \{1, \ldots, m\}$, $|S| = n$, we have already noted that the predecessor problem on S is equivalent to rank and select_1 on a bit-vector $\boldsymbol{c}$ with $w(\boldsymbol{c}) = n$, and $|\boldsymbol{c}| = m$. However, $B(m,n) + o(n) = O(n \lg m)$. Thus, given a bit-vector that uses $B(m,n) + o(n)$ bits and supports rank in $O(1)$ time for $m = n(\lg n)^{\omega(1)}$, we can augment it with the trivial $O(1)$-time data structure for select_1, that stores the value of $\mathsf{select}_1(i)$ for $i = 1, \ldots, n$ (which occupies a further $O(n \lg m)$ bits), solving the predecessor problem in $O(1)$ time, a contradiction. The hardness of select_0 is shown in [17], but follows easily from Theorem 1(a) and Equation 1.

3 APPLICATIONS

There are a vast number of applications of bit-vectors in succinct and compressed data structures (see e.g. [13]). Such data structures are used for, e.g., text indexing, compact representations of graphs and trees, and representations of semi-structured (XML) data.

4 EXPERIMENTAL RESULTS

Several teams have implemented bit-vectors and compressed bit-vectors. When implementing bit-vectors for good practical performance, both in terms of speed and space usage, the lower-order terms are very important, even for uncompressed bit-vectors[1], and can dominate the space usage even for bit-vector sizes that are at the limit of conceivable future practical interest. Unfortunately, this problem may not be best addressed purely by a theoretical analysis of the lower-order terms. Bit-vectors work by partitioning the input bit sequence into (usually equal-sized) blocks at several levels of granularity—usually 2-3 levels are needed to obtain a space bound of $m + o(m)$ bits. However, better space usage—as well as better speed—in practice can be obtained by reducing the number of levels, resulting in space bounds of the form $(1 + \epsilon)m$ bits, for any $\epsilon > 0$, with support for rank and select in $O(1/\epsilon)$ time.

Clark [2] implemented bit-vectors for external-memory suffix trees. More recently, an implementation using ideas of Clark and Jacobson was used by [7], which occupied $(1 + \epsilon)m$ bits and supported operations in $O(1/\epsilon)$ time. Using a substantially different approach, Kim et al. [12] gave a bit-vector that takes $(2 + \epsilon)n$ bits to support rank and select. Experiments using bit sequences derived from real-world data in [3, 4] showed that if parameters are set to ensure that [12] and [7] use similar space — on *typical* inputs — the Clark-Jacobson implementation of [7] is somewhat faster than an implementation of [12]. On some inputs, the Clark-Jacobson implementation can use significantly more space, whereas Kim et al's bit-vector appears to have stable space usage; Kim et al's bit-vector may also be superior for somewhat sparse bit-vectors. Combining ideas from [7, 12], a third practical bit-vector (which is not a bit-vector index) was described in [4], and appears to have desirable features of both [12] and [7]. A first implementational study on compressed bit-vectors can be found in [14] (compressed bit-vectors supporting only select_1 were considered in [4]).

5 URL to CODE

Bit-vector implementations from [3, 4, 7] can be found at `http://hdl.handle.net/2381/318`.

6 CROSS REFERENCES

Arithmetic Coding for Data Compression; Compressed Text Indexing; Succinct Encoding of Permutations and its Applications to Text Indexing; Tree Compression and Indexing.

[1] For compressed bit-vectors, the 'lower-order' $o(m)$ or $o(n)$ term can dominate $B(m,n)$, but this is not our concern here.

7 RECOMMENDED READING

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[2] D. Clark and J. I. Munro. Efficient suffix trees on secondary storage. In *Proc. 7th ACM-SIAM SODA*, pp. 383–391, 1996.

[3] O. Delpratt, N. Rahman and R. Raman. Engineering the LOUDS succinct tree representation. In *Proc. WEA 2006*, LNCS 4007, pp. 134–145, Springer, 2006.

[4] O. Delpratt, N. Rahman and R. Raman. Compressed prefix sums. In *Proc. SOFSEM 2007*, LNCS 4362, pp. 235–247, 2007.

[5] P. Elias. Efficient storage retrieval by content and address of static files. *Journal of the ACM*, 21(2):246–260, April 1974.

[6] P. Ferragina and R. Venturini. A simple storage scheme for strings achieving entropy bounds. *Theoretical Computer Science* **372** (2007), pp 115–121.

[7] R. F. Geary, N. Rahman, R. Raman and V. Raman. A simple optimal representation for balanced parentheses. *Theoretical Computer Science* **368** (2006), pp. 231–246.

[8] A. Golynski. Optimal lower bounds for rank and select indexes. In *Proc. ICALP 2006, Part I.*, LNCS 4051, pp. 370–381, 2006.

[9] Rodrigo González and Gonzalo Navarro Statistical encoding of succinct data structures. In *Proc. CPM 2006*, LNCS 4009, Springer, 2006, pp. 294-305.

[10] K. Sadakane and R. Grossi. Squeezing succinct data structures into entropy bounds. In *Proc. 17th ACM-SIAM SODA*, pp. 1230–1239. ACM Press, 2006.

[11] G. Jacobson. Space-efficient static trees and graphs. In *Proc. 30th FOCS*, 549-554, 1989.

[12] D. K. Kim, J. C. Na, J. E. Kim and K. Park. Efficient implementation of Rank and Select functions for succinct representation. In *Proc. WEA 2005*, LNCS 3505, pp. 315–327, 2005.

[13] J. I. Munro and S. Srinivasa Rao. Succinct representation of data structures. Chapter 37 in *Handbook of Data Structures with Applications*, D. Mehta and S. Sahni, eds. Chapman and Hall/CRC Press, 2005.

[14] D. Okanohara and K. Sadakane. Practical entropy-compressed rank/select dictionary. In *Proc. 9th ACM-SIAM Workshop on Algorithm Engineering and Experiments (ALENEX '07)*, SIAM, to appear, 2007.

[15] R. Pagh. Low redundancy in static dictionaries with constant query time. *SIAM J. Comput*, **31** (2001), 353–363.

[16] M. Patrascu and M. Thorup, Time-space trade-offs for predecessor search. In *Proc. 38th ACM STOC*, pp. 232–240, 2006.

[17] R. Raman, V. Raman and S. S. Rao. Succinct indexable dictionaries, with applications to representing k-ary trees and multisets. In *Proc. 13th ACM-SIAM SODA*, 233–242, 2002.

[18] I. Witten, A. Moffat, I. Bell. *Managing Gigabytes*, 2e. Morgan Kaufmann, 1999.