Searching the Integers (2006; Pătraşcu, Thorup)

Mihai Pătraşcu, MIT, `web.mit.edu/∼mip/www/`

Index terms: cell-probe lower bounds, rank space, orthogonal range queries
Synonyms: predecessor problem, successor problem, IP lookup

1 Problem Definition

Consider an ordered universe U, and a set $T \subset U$ with $|T| = n$. The goal is to preprocess T, such that the following query can be answered efficiently: given $x \in U$, report the predecessor of x, i.e. $\max\{y \in T \mid y < x\}$. One can also consider the dynamic problem, where elements are inserted and deleted into T. Let t_q be the query time, and t_u the update time.

This is a fundamental search problem, with an impressive number of applications. Later, this entry discusses IP lookup (forwarding packets on the Internet), orthogonal range queries and persistent data structures as examples.

The problem was considered in many computational models. In fact, most models below were initially defined to study the predecessor problem.

comparison model: The problem can be solved through binary search in $\Theta(\lg n)$ comparisons. There is a lot of work on adaptive bounds, which may be sublogarithmic. Such bounds may depend on the finger distance, the working set, entropy etc.

binary search trees: Predecessor search is one of the fundamental motivations for binary search trees. In this restrictive model, one can hope for an instance optimal (competitive) algorithm. Attempts to achieve this are described in a separate entry.[1]

word RAM: Memory is organized as words of b bits, and can be accessed through indirection. Constant-time operations include the standard operations in a language such as C (addition, multiplication, shifts and bitwise operations).

It is standard to assume the universe is $U = \{1, \dots, 2^\ell\}$, i.e. one deals with ℓ-bit integers. The floating point representation was designed so that order is preserved when values are interpreted as integers, so any algorithm will also work for ℓ-bit floating point numbers.

The standard *transdichotomous* assumption is that $b = \ell$, so that an input integer is represented in a word. This implies $b \geq \lg n$.

cell-probe model: This is a nonuniform model stronger than the word RAM, in which the operations are arbitrary functions on the memory words (cells) which have already been probed. Thus, t_q only counts the number of cell probes. This is an ideal model for lower bounds, since it does not depend on the operations implemented by a particular computer.

[1] $O(\log\log n)$*-competitive Binary Search Trees* (2004; Demaine, Harmon, Iacono, Pătraşcu)

communication games: Let Alice have the query x, and Bob have the set T. They are trying to find the predecessor of x through τ rounds of communication, where in each round Alice sends m_A bits, and Bob replies with m_B bits.

This can simulate the cell-probe model when $m_B = b$ and m_A is the logarithm of the memory size. Then $\tau \le t_q$ and one can use communication complexity to obtain cell-probe lower bounds.

external memory: The unit of access is a page, containing B words of ℓ bits each. B-trees solve the problem with query and update time $O(\log_B n)$, and one can also achieve this oblivious to the value of B.[2] The cell-probe model with $b = B \cdot \ell$ is stronger than this model.

AC^0 **RAM:** This is a variant of the word RAM in which allowable operations are functions that have constant depth, unbounded fan-in circuits. This excludes multiplication from the standard set of operations.

RAMBO: this is a variant of the RAM with a nonstandard memory, where words of memory can overlap in their bits. In the static case this is essentially equivalent to a normal RAM. However, in the dynamic case updates can be faster due to the word overlap [5].

The worst-case logarithmic bound for comparison search is not particularly informative when efficiency really matters. In practice, B-trees and variants are standard when dealing with huge data sets. Solutions based on RAM tricks are essential when the data set is not too large, but a fast query time is crucial, such as in software solutions to IP lookup [7].

2 Key Results

Building on a long line of research, Pătraşcu and Thorup [15, 16] finally obtained matching upper and lower bounds for the static problem in the word RAM, cell-probe, external memory and communication game models.

Let S be the number of words of space available. (In external memory, this is equivalent to $\frac{S}{B}$ pages.) Define $a = \lg \frac{S \cdot \ell}{n}$. Also define $\lg x = \lceil \log_2(x+2) \rceil$, so that $\lg x \ge 1$ even if $x \in [0,1]$. Then the optimal search time is, up to constant factors:

$$\min \begin{cases} \log_b n \qquad = \Theta(\min\{\log_B n, \log_\ell n\}) \\ \lg \frac{\ell - \lg n}{a} \\ \dfrac{\lg \frac{\ell}{a}}{\lg\left(\frac{a}{\lg n} \cdot \lg \frac{\ell}{a}\right)} \\ \dfrac{\lg \frac{\ell}{a}}{\lg\left(\lg \frac{\ell}{a} \middle/ \lg \frac{\lg n}{a}\right)} \end{cases} \tag{1}$$

The bound is achieved by a deterministic query algorithm. For any space S, the data structure can be constructed in time $O(S)$ by a randomized algorithm, starting with the set T given in sorted order. Updates are supported in expected time $t_q + O(\frac{S}{n})$. Thus, besides locating the element through one predecessor query, updates change a minimal fraction of the data structure.

[2]See *Cache-oblivious B-tree* (2005; Bender, Demaine, Farach-Colton).

Lower bounds hold in the powerful cell-probe model, and hold even for randomized algorithms. When $S \geq n^{1+\varepsilon}$, the optimal trade-off for communication games coincides to (1). Note that the case $S = n^{1+o(1)}$ essentially disappears in the reduction to communication complexity, because Alice's messages only depends on $\lg S$. Thus, there is no asymptotic difference between $S = O(n)$ and, say, $S = O(n^2)$.

Upper Bounds. The following algorithmic techniques give the optimal result:

- *B-trees* give $O(\log_B n)$ query time with linear space.
- *fusion trees*, by Fredman and Willard [10], achieve a query time of $O(\log_b n)$. The basis of this is a *fusion node*, a structure which can search among b^ε values in constant time. This is done by recognizing that only a few bits of each value are essential, and packing the relevant information about all values in a single word.
- *van Emde Boas search* [18] can solve the problem in $O(\lg \ell)$ time by binary searching for the length of the longest common prefix between the query and a value in T. Beginning the search with a table lookup based on the first $\lg n$ bits, and ending when there is enough space to store all answers, the query time is reduced to $O(\lg \frac{\ell - \lg n}{a})$.
- a technique by *Beame and Fich* [4] can perform a multiway search for the longest common prefix, by maintaining a careful balance between ℓ and n. This is relevant when the space is at least $n^{1+\varepsilon}$, and gives the third branch of (1). Pătraşcu and Thorup [15] show how related ideas can be implemented with smaller space, yielding the last branch of (1).

Observe that external memory only features in the optimal trade-off through the $O(\log_B n)$ term coming from B-trees. Thus, it is optimal to either use the standard, comparison-based B-trees, or use the best word RAM strategy which completely ignores external memory.

Lower Bounds. All lower bounds before [15] where shown in the communication game model. Ajtai [1] was the first to prove a superconstant lower bound. His results, with a correction by Miltersen [12], show that for polynomial space, there exists n as a function of ℓ making the query time $\Omega(\sqrt{\lg \ell})$, and likewise there exists ℓ a function of n making the query complexity $\Omega(\sqrt[3]{\lg n})$.

Miltersen et al [13] revisited Ajtai's proof, extending it to randomized algorithms. More importantly, they captured the essence of the proof in an independent *round elimination lemma*, which is an important tool for proving lower bounds in asymmetric communication.

Beame and Fich [4] improved Ajtai's lower bounds to $\Omega(\frac{\lg \ell}{\lg \lg \ell})$ and $\Omega(\sqrt{\frac{\lg n}{\lg \lg n}})$ respectively. Sen and Venkatesh [17] later gave an improved round elimination lemma, which can reprove these lower bounds, but also for randomized algorithms.

Finally, using the message compression lemma of [6] (an alternative to round elimination), Pătraşcu and Thorup [15] showed an optimal trade-off for communication games. This is also an optimal lower bound in the other models when $S \geq n^{1+\varepsilon}$, but not for smaller space.

More importantly, [15] developed the first tools for proving lower bounds exceeding communication complexity, when $S = n^{1+o(1)}$. This showed the first separation ever between a data structure or polynomial size, and one of near linear size. This is fundamentally impossible through a direct communication lower bound, since the reduction to communication games only depends on $\lg S$.

The full result of Pătraşcu and Thorup [15] it the trade-off (1). Initially, this was shown only for deterministic query algorithms, but eventually it was extended to a randomized lower bound as well [16]. Among the surprising consequences of this result was that the classic van Emde Boas search is optimal for near-linear space (and thus for dynamic data structures), whereas with quadratic space it can be beaten by the technique of Beame and Fich.

A key technical idea of [15] is to analyze many queries simultaneously. Then, one considers a communication game involving all queries, and proves a direct-sum version of the round elimination lemma. Arguably, the proof is even simpler than for the regular round elimination lemma. This is achieved by considering a stronger model for the inductive analysis, in which the algorithm is allowed to *reject* a large fraction of the queries before starting to make probes.

Bucketing. The rich recursive structure of the problem can not only be used for fast queries, but also to optimize the space and update time — of course, within the limits of (1). The idea is to place ranges of consecutive values in buckets, and include a single representative of each bucket in the predecessor structure. After performing a query on the predecessor structure (now with fewer elements), one need only search within the relevant bucket.

Because buckets of size $w^{O(1)}$ can be handled in constant time by fusion trees, it follows that factors of w in space are irrelevant. A more extreme application of the idea is given by *exponential trees* [3]. Here buckets have size $\Theta(n^{1-\gamma})$, where γ is a sufficiently small constant. Buckets are handled recursively in the same way, leading to $O(\lg \lg n)$ levels. If the initial query time is at least $t_q \geq \lg^{\varepsilon} n$, the query times at each level decrease geometrically, so overall time only grows by a constant factor. However, any polynomial space is reduced to linear, for an appropriate choice of γ. Also, the exponential tree can be updated in $O(t_q)$ time, even if the original data structure was static.

3 Applications

Perhaps the most important application of predecessor search is IP lookup. This is the problem solved by routers for each packet on the Internet, when deciding which subnetwork to forward the packet to. Thus, it is probably the most run algorithmic problem in the world. Formally, this is an *interval stabbing* query, which is equivalent to predecessor search in the static case [9]. As this is a problem where efficiency really matters, it is important to note that the fastest deployed software solutions [7] use integer search strategies (not comparison-based), as theoretical results would predict.

In addition, predecessor search is used pervasively in data structures, when reducing problems to *rank space*. Given a set T, one often wants to relabel it to the simpler $\{1, \dots, n\}$ ("rank space"), while maintaining order relations. If one is presented with new values dynamically, the need for a predecessor query arises. Here are a couple of illustrative examples:

- in orthogonal range queries, one maintains a set of points in U^d, and queries for points in some rectangle $[a_1, b_1] \times \cdots \times [a_d, b_d]$. Though bounds typically grow exponentially with the dimension, the dependence on the universe can be factored out. At query time, one first runs $2d$ predecessor queries transforming the universe to $\{1, \dots, n\}^d$.

- to make pointer data structures persistent [8], an outgoing link is replaced by a vector of pointers, each valid for some period of time. Deciding which link to follow (given the time being queried) is a predecessor problem.

Finally, it is interesting to note that the lower bounds for predecessor hold, by reductions, for all applications described above. To make these reductions possible, the lower bounds are in fact shown for the weaker *colored predecessor* problem. In this problem, the values in T are colored red or blue, and the query only needs to find the color of the predecessor.

4 Open Problems

It is known [2] how to implement fusion trees with AC^0 instructions, but not the other query strategies. What is the best query trade-off achievable on the AC^0 RAM? In particular, can van Emde Boas search be implemented with AC^0 instructions?

For the dynamic problem, can the update times be made deterministic? In particular, can van Emde Boas search be implemented with fast deterministic updates? This is a very appealing problem, with applications to deterministic dictionaries [14]. Also, can fusion nodes be updated deterministically in constant time? Atomic heaps [11] achieve this when searching only among $(\lg n)^\varepsilon$ elements, not b^ε.

Finally, does an update to the predecessor structure require a query? In other words, can $t_u = o(t_q)$ be obtained, while still maintaining efficient query times?

5 Cross References

Please link to: *$O(\log\log n)$-competitive Binary Search Trees* (2004; Demaine, Harmon, Iacono, Pătraşcu); *Cache-oblivious B-tree* (2005; Bender, Demaine, Farach-Colton).

6 Recommended Reading

References

[1] M. Ajtai, *A lower bound for finding predecessors in Yao's cell probe model*, Combinatorica, 8 (1988), pp. 235–247.

[2] A. Andersson, P. B. Miltersen, and M. Thorup, *Fusion trees can be implemented with AC^0 instructions only*, Theoretical Computer Science, 215 (1999), pp. 337–344.

[3] A. Andersson and M. Thorup, *Dynamic ordered sets with exponential search trees*, Journal of the ACM, 54 (2007). See also FOCS'96, STOC'00.

[4] P. Beame and F. E. Fich, *Optimal bounds for the predecessor problem and related problems*, Journal of Computer and System Sciences, 65 (2002), pp. 38–72. See also STOC'99.

[5] A. Brodnik, S. Carlsson, M. L. Fredman, J. Karlsson, and J. I. Munro, *Worst case constant time priority queue*, Journal of Systems and Software, 78 (2005), pp. 249–256. See also SODA'01.

[6] A. Chakrabarti and O. Regev, *An optimal randomised cell probe lower bound for approximate nearest neighbour searching*, in Proc. 45th IEEE Symposium on Foundations of Computer Science (FOCS), 2004, pp. 473–482.

[7] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, *Small forwarding tables for fast routing lookups*, in Proc. ACM SIGCOMM, 1997, pp. 3–14.

[8] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan, *Making data structures persistent*, Journal of Computer and System Sciences, 38 (1989), pp. 86–124. See also STOC'86.

[9] A. Feldmann and S. Muthukrishnan, *Tradeoffs for packet classification*, in Proc. IEEE INFOCOM, 2000, pp. 1193–1202.

[10] M. L. Fredman and D. E. Willard, *Surpassing the information theoretic bound with fusion trees*, Journal of Computer and System Sciences, 47 (1993), pp. 424–436. See also STOC'90.

[11] ——, *Trans-dichotomous algorithms for minimum spanning trees and shortest paths*, Journal of Computer and System Sciences, 48 (1994), pp. 533–551. See also FOCS'90.

[12] P. B. Miltersen, *Lower bounds for Union-Split-Find related problems on random access machines*, in Proc. 26th ACM Symposium on Theory of Computing (STOC), 1994, pp. 625–634.

[13] P. B. Miltersen, N. Nisan, S. Safra, and A. Wigderson, *On data structures and asymmetric communication complexity*, Journal of Computer and System Sciences, 57 (1998), pp. 37–49. See also STOC'95.

[14] R. Pagh, *A trade-off for worst-case efficient dictionaries*, Nordic Journal of Computing, 7 (2000), pp. 151–163. See also SWAT'00.

[15] M. Pătraşcu and M. Thorup, *Time-space trade-offs for predecessor search*, in Proc. 38th ACM Symposium on Theory of Computing (STOC), 2006, pp. 232–240.

[16] ——, *Randomization does not help searching predecessors*, in Proc. 18th ACM/SIAM Symposium on Discrete Algorithms (SODA), 2007, pp. 555–564.

[17] P. Sen and S. Venkatesh, *Lower bounds for predecessor searching in the cell probe model*, Journal of Computer and System Sciences, 74 (2008), pp. 364–385. See also ICALP'01, CCC'03.

[18] P. van Emde Boas, R. Kaas, and E. Zijlstra, *Design and implementation of an efficient priority queue*, Mathematical Systems Theory, 10 (1977), pp. 99–127. Conference version by van Emde Boas alone in FOCS'75.