Towards Tight Lower Bounds for Range Reporting on the RAM*†

Allan Grønlund[1] and Kasper Green Larsen[2]

1 **Department of Computer Science, Aarhus University, Aarhus, Denmark**
jallan@cs.au.dk
2 **Department of Computer Science, Aarhus University, Aarhus, Denmark**
larsen@cs.au.dk

Abstract

In the orthogonal range reporting problem, we are to preprocess a set of n points with integer coordinates on a $U \times U$ grid. The goal is to support reporting all k points inside an axis-aligned query rectangle. This is one of the most fundamental data structure problems in databases and computational geometry. Despite the importance of the problem its complexity remains unresolved in the word-RAM.

On the upper bound side, three best tradeoffs exist, all derived by reducing range reporting to a ball-inheritance problem. Ball-inheritance is a problem that essentially encapsulates all previous attempts at solving range reporting in the word-RAM. In this paper we make progress towards closing the gap between the upper and lower bounds for range reporting by proving cell probe lower bounds for ball-inheritance. Our lower bounds are tight for a large range of parameters, excluding any further progress for range reporting using the ball-inheritance reduction.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Data Structures, Lower Bounds, Cell Probe Model, Range Reporting

Digital Object Identifier 10.4230/LIPIcs.ICALP.2016.92

1 Introduction

In the orthogonal range reporting problem, we are to preprocess a set of n points with integer coordinates on a $U \times U$ grid. The goal is to support reporting all k points inside an axis-aligned query rectangle. This is one of the most fundamental data structure problems in databases and computational geometry. Given the importance of the problem, it has been extensively studied in all the relevant models of computation including e.g. the word-RAM, pointer machine and external memory model. In the latter two models, we typically work under an assumption of *indivisibility*, meaning that input points have to be stored as they are, i.e. compression techniques such as rank-space reduction and word-packing cannot be used to reduce the space consumption of data structures. The indivisibility assumption greatly alleviates the task of proving lower bounds, which has resulted in a completely tight characterisation of the complexity of orthogonal range reporting in these two models. More specifically, Chazelle [7] presented a pointer machine data structure answering queries in

* A full version of this paper is available at http://arxiv.org/abs/1411.0644.
† Both authors are supported in part by Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation, grant DNRF84. Larsen is also supported by a Villum Young Investigator Grant and an AUFF Starting Grant. Grønlund is also supported by an Industrial Post Doc Grant from Innovation Fund Denmark.

43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016).
Editors: Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi;
Article No. 92; pp. 92:1–92:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

optimal $O(\lg n + k)$ time using $O(n \lg n / \lg\lg n)$ space and later proved that this space bound is optimal for any query time of the form $O(\lg^c n + k)$, where $c \geq 1$ is an arbitrary constant [8]. In the external memory model, Arge et al. [2] presented a data structure answering queries in optimal $O(\lg_B n + k/B)$ I/Os with $O(n \lg n / \lg\lg_B n)$ space and also proved the space bound to be optimal for any query time of the form $O(\lg_B^c n + k/B)$, where $c \geq 1$ is a constant. Here B is the disk block size. Thus the orthogonal range reporting problem has been completely closed for at least 15 years in both these models of computation. If we instead abandon the indivisibility assumption and consider orthogonal range reporting in the arguably more realistic model of computation, the word-RAM, our understanding of the problem is much more disappointing. Assuming the coordinates are polynomial in n ($U = n^{O(1)}$), the current best word-RAM data structures, by Chan et al. [5], achieve the following tradeoffs:

1. Optimal query time $O(\lg\lg n + k)$ with $O(n \lg^{\varepsilon} n)$ words of space for any constant $\varepsilon > 0$.
2. Query time $O((1+k)\lg\lg n)$ with $O(n \lg\lg n)$ words of space.
3. Query time $O((1+k)\lg^{\varepsilon} n)$ with optimal $O(n)$ words of space.

Thus we can achieve linear space by paying a $\lg^{\varepsilon} n$ penalty per point reported. And even if we insist on an optimal $O(\lg\lg n + k)$ query time, it is possible to improve over the optimal space bound in the pointer machine and external memory model by almost a $\lg n$ factor. Naturally the improvements rely heavily on points not being indivisible.

On the lower bounds side, Pătraşcu and Thorup [12, 14] proved that the query time must be $\Omega(\lg\lg n + k)$ for space $n \lg^{O(1)} n$. This lower bound was obtained by reduction from the predecessor search problem. For predecessor search, the query time of $\lg\lg n$ is known to be achievable with linear space. Thus the reduction is incapable of distinguishing the three space regimes of the current best data structures for range reporting. Perhaps it might just be possible to construct a linear space data structure with $O(\lg\lg n + k)$ query time. This would have a huge impact in practice, since the non-linear space solutions are most often abandoned for the kd-trees [3], using linear space and answering queries in $O(\sqrt{n} + k)$ time. This is simply because more than a constant factor above linear space is prohibitive for most applications. Thus ruling out the existence of fast linear space data structures would be a major contribution. The focus of this paper is on understanding this gap and the complexity of orthogonal range reporting in the word-RAM. This boils down to understanding how much compression and word-packing techniques can help us in the regime between linear space and $O(n \lg^{\varepsilon} n)$ space. Since our results concern definitions made by Chan et al. [5], we first give a more formal definition of the word-RAM and briefly review the technique of rank space reduction and the main ideas in [5].

1.1 Range Reporting in the word-RAM

The word-RAM model was designed to mimic what is possible in modern imperative programming languages such as C. In the word-RAM, the memory is divided into words of $\Theta(\lg n)$ bits. The words have integer addresses and we allow random access to any word in constant time. We also assume all standard word operations from modern programming languages takes constant time. This includes e.g. integer addition, subtraction, multiplication, division, bit-wise AND, OR, XOR, SHIFT etc. Having $\Theta(\lg n)$ bit words is a reasonable assumption since machine words on standard computers have enough bits to address the input and to store pointers into a data structure.

Rank Space Reduction. Most of the previous range reporting data structures for the word-RAM have used rank space reduction (or variants thereof) to save space, see e.g. [1, 11]. Rank space reduction is the following: Given a set P of n points on a $U \times U$ grid, compute for each point $(x, y) \in P$ the rank $r_x(x)$ of x amongst the x-coordinates of points in P. Similarly compute the rank $r_y(x)$ of y amongst the y-coordinates of points in P. Construct a new point set P^* with each point $(x, y) \in P$ replaced by $(r_x(x), r_y(y))$. The point set P^* is said to be in *rank space*. A point $(x, y) \in P$ lies inside a query range $q = [x_0; x_1] \times [y_0; y_1]$ precisely if $(r_x(x), r_y(y))$ lies inside the range $q^* = [r_x(x_0); r_x(x_1)] \times [r_y(y_0); r_y(y_1)]$. Thus if we store a data structure for mapping q to q^* and a table mapping points in P^* back to points in P, the output of a query q can be computed from the output of the query q^* on P^*. Since the coordinates of a point in P^* can be represented using $\lg n$ bits, this gives a saving in space if $\lg n \ll \lg U$.

In previous range reporting data structures, rank space reductions are often used recursively on smaller and smaller point sets $P_t \subset P_{t-1} \subset \cdots \subset P_1 \subset P$. Applying t rounds of rank space reduction however results in a query time of $O(f(n) + tk)$ since each reported point has to be *decompressed* through t rank space reduction tables.

The Ball-Inheritance Problem. In the following, we present the main ideas of the current best data structures, due to Chan et al. [5]. Their solution is based on an elegant way of combining rank space reductions over all levels of a range tree:

Construct a complete binary tree with the points of P stored in the leaves ordered from left to right by their x-coordinate. Every internal node v is associated with the subset of points P_v stored in the leaves of the subtree rooted at v. For every internal node v, map the points P_v to rank space and denote the resulting set of points P_v^*. Store in v a data structure for answering 3-sided range queries on P_v^*. Here a 3-sided query is either of the form $[x_0; \infty) \times [y_0; y_1]$ or $(-\infty, x_1] \times [y_0; y_1]$. If we require that only the rank space y-coordinate of a point is reported (and not the rank space x-coordinate), these 3-sided data structures can be implemented in $O(n)$ bits and with $O(k)$ query time using succinct data structures for range minimum queries, see e.g. [9]. For each leaf, we simply store the associated point. The total space usage is $O(n \lg n + n \lg U)$ bits, which is $O(n)$ words.

To answer a query $q = [x_0; x_1] \times [y_0; y_1]$, find the lowest common ancestor, w, of the leaves storing the successor of x_0 and the predecessor of x_1 respectively. Let w_ℓ be the left child of w and w_r the right child. The points inside q are precisely the points $P_{w_\ell} \cap [x_0; \infty) \times [y_0; y_1]$ plus $P_{w_r} \cap (-\infty, x_1] \times [y_0; y_1]$. The data structures of Chan et al. now proceeds by mapping these two 3-sided queries to rank space amongst points in $P_{w_\ell}^*$ and $P_{w_r}^*$ respectively and answering the two queries using the 3-sided data structures stored at w_ℓ and w_r. This reports, for every point $(x, y) \in P_{w_\ell} \cap q$ (and $(x, y) \in P_{w_r} \cap q$), the rank of y amongst the y-coordinates of all points in P_{w_ℓ} (P_{w_r}). Assuming one can build an S word auxiliary data structure that supports mapping these rank space y-coordinates back to the original points in t time per point (i.e. through t rank space decompressions), this gives a data structure for orthogonal range reporting that answers queries in $O(\lg \lg n + t(1 + k))$ time using $S + O(n)$ space, see [5] for full details. Chan et al. named this abstract decompression problem *the ball-inheritance problem* and defined it as follows:

▶ **Definition 1** (Chan et al. [5]). In the ball-inheritance problem, the input is a complete binary tree with n leaves. In the root node, there is an ordered list of n balls. Each ball is associated with a unique leaf of the tree and we say the ball *reaches* that leaf. Every internal node v also has an associated list of balls, containing those balls reaching a leaf in the subtree rooted at v. The ordering of the balls in v's list is the same as their ordering in the root's list. We think of each ball in v's list as being inherited from v's parent.

A query is specified by a pair (v, i) where v is a node in the tree and i is an index into v's list of balls. The goal is to return the index of the leaf reached by the i'th ball in v's list of balls.

It is not hard to see that a solution to the ball-inheritance problem is precisely what is needed in Chan et al.'s data structures: We have one ball per point. The ball corresponding to a point (x, y) reaches the $r_x(x)$'th leaf, where $r_x(x)$ is the rank of x amongst all x-coordinates. The ordering of the balls inside the lists is just the ordering on the y-coordinates of the corresponding points. Thus answering a ball-inheritance query (v, i) corresponds exactly to determining the leaf storing the point from P_v having a rank space y-coordinate of i. Since Chan et al. stored the points in the leaves, this also recovers the original point.

All three tradeoffs by Chan et al. come from solving the ball-inheritance problem with the following bounds:

▶ **Theorem 2** (Chan et al. [5]). *For any $2 \le B \le \lg^{\varepsilon} n$, we can solve the ball-inheritance problem with:* (1) *space $O(nB \lg\lg n)$ and query time $O(\lg_B \lg n)$; or* (2) *space $O(n \lg_B \lg n)$ and query time $O(B \lg\lg n)$.*

While not all previous range reporting data structures directly solve the ball-inheritance problem, they are all based on rank space reductions and decompression of one point at a time, just in less efficient ways. Thus the ball-inheritance problem in some sense captures the essence of all previous approaches to solving range reporting and the bounds obtained for the ball-inheritance problem also sets the current limits for orthogonal range reporting.

We remark that the ball-inheritance problem also has been used to improve the upper bounds for various other problems with a range reporting flavor to them, see e.g. [6, 4]. Thus in light of the lack of progress in proving tight lower bounds for range reporting, it seems like a natural goal to understand the complexity of the ball-inheritance problem.

1.2 Our Results

In this paper, we prove a lower bound for the ball-inheritance problem. Our lower bound is tight for a large range of parameters and is as follows:

▶ **Theorem 3.** *Any word-RAM data structure for the ball-inheritance problem which uses S words of space, must have query time t satisfying:*

$$t = \Omega\left(\frac{\lg\lg n}{\lg(S/n) + \lg\lg\lg n}\right).$$

Comparing to the ball-inheritance upper bounds of Chan et al. (Theorem 2), we see that this essentially matches their first tradeoff and is tight for any $S = \Omega(n \lg^{1+\varepsilon} \lg n)$ where $\varepsilon > 0$ is an arbitrarily small constant. Most importantly, it implies that for constant query time, one needs space $n \lg^{\varepsilon} n$ words. Thus any range reporting data structure based on the ball-inheritance problem cannot improve over the bounds of Chan et al. in the regime of space $S = \Omega(n \lg^{1+\varepsilon} \lg n)$ words. We believe this holds true for any data structure that is based on *decompressing* one point at a time from some subproblem in rank space. Since decompressing from a subproblem in rank space is hard to formalize exactly, we leave it at this.

One can view our lower bound in two ways: Either as a strong indicator that the data structure of Chan et al. is optimal, or as a suggestion for how to find better upper bounds. The lower bound above shows that if we want to develop faster data structures, we have to find a technique that in some sense allows us to decompress $\omega(1)$ points in one batch, faster than decompressing each point in turn. This is not necessarily impossible given the large success of batched evaluations in other problems such as matrix multiplication and multipoint evaluation of polynomials.

We also want to make a remark regarding the gap between the second tradeoff of Chan et al. and our lower bound. We conjecture that the upper bound of Chan et al. is tight, but note that current lower bound techniques (in the cell probe model) are incapable of proving any lower bounds exceeding the one we obtain in Theorem 3: The ball-inheritance problem has only $n \lg n$ queries and the strongest lower bound for any data structure problem with m queries (for any m) is $t = \Omega(\lg(m/n)/\lg(S/n))$ [10], thus apart from our $\lg \lg \lg n$ "dirt factor", our lower bound is as strong as it possibly can be with current techniques. Note that previous papers have stated their lower bounds with a $\lg m$ rather than $\lg(m/n)$ since the data structure problems considered there have m polynomially larger than n. It is however not too difficult to see that previous techniques really are limited to $\lg(m/n)$. For the cell sampling technique [10], one needs to have $n/\operatorname{poly}(\lg n)$ queries surviving the cell sampling, hence queries have to survive the sampling with probability at least $(n/(m \operatorname{poly}(\lg n)))$, thus requiring an m/n term. For the communication game by Pătraşcu and Thorup [13], the player holding queries has $n/\operatorname{poly}(\lg n)$ queries, thus she can specify her set of queries with $\lg \binom{m}{n/\operatorname{poly}(\lg n)}$ bits of communication. This again results in a $\lg(m/n)$ term rather than $\lg m$.

Technical Contributions. As a side remark, we believe our lower bound proof has interest from a purely technical point of view. In the lower bound proof, we carefully exploit that a data structure is *not* non-deterministic. While this might sound odd at first, Wang and Yin [15] recently showed that, with only few exceptions (e.g. the predecessor lower bounds), all previous lower bound techniques yield lower bounds that hold non-deterministically. Thus having a new proof outside this category is an important contribution and may hopefully help in closing fundamental problems where avoiding non-determinism in proofs is crucial. This is e.g. the case for the deterministic dictionaries problem, which is amongst the most fundamental open problems in the field of data structures. This problem is trivially solved with constant update time and query time non-deterministically (just maintain a sorted linked list) and hence lower bound proofs must use ideas similar to those we present in this paper to prove super constant lower bounds for this important problem.

2 Lower Bound Proof

We prove our lower bound in the cell probe model [16], where the complexity of a data structure is the number of cells it reads/probes. More specifically, a data structure with query time t and space S consists of memory of S cells with consecutive integer addresses $0, \ldots, S-1$. Each cell stores w bits and we assume $w = \Omega(\lg n)$. When answering a query, the data structure may probe up to t cells and must announce the answer to the query solely based on the contents of the probed cells. The cell to probe in each step may depend arbitrarily on the query and the contents of previously probed cells. Thus computation is free of charge in the cell probe model and lower bounds proved in this model clearly applies to word-RAM data structures.

2.1 Main Ideas

In the following, we sketch the overall approach in our proof. Assume we have a data structure for the ball-inheritance problem, having space S cells of w bits and with query time t. Assume furthermore that the data structure performs very poorly in the following sense: For every input I to the ball-inheritance problem and every leaf index $b \in [n] = \{0, \ldots, n-1\}$, let $Q(b, I)$ be the set of queries that have b as its answer. Each such query probes at most t

cells of the data structure on input I. Assume these sets of cells are *disjoint*, i.e. information about the leaf b is stored in $|Q(b,I)| = \lg n$ disjoint t-sized locations in the memory.

Now pick a uniform random set C of $\lg(n!)/(4w)$ memory cells. For a query q, we say that q survives if all its t probes lie in C. Then by the disjointness of the probed cells, there will be a surviving query in $Q(b,I)$ with probability roughly $1-(1-(|C|/S)^t)^{\lg n}$. If $t = o(\lg\lg n/\lg(S/|C|))$, this is about $1-\exp(\lg n \cdot (|C|/S)^t) = 1-\exp(\lg^{1-o(1)} n)$, i.e. each leaf index is almost certainly the answer to a surviving query. Thus C must basically store the entire input. But $|C|$ is too small for this and we get a contradiction, i.e. $t = \Omega(\lg\lg n/\lg(Sw/(n\lg n)))$, which roughly equals the lower bound we claim. There are obviously a few more details to it, but this is the main idea.

Of course any realistic attempt at designing a data structure for the ball-inheritance problem would try to make the queries in $Q(b,I)$ probe the same cells (which is exactly what Chan et al.'s solution does [5]). In our actual proof, we get around this using the following observation: Consider two queries q_1, q_2 to the ball-inheritance problem, where q_2 is asked in a node d levels below the node of q_1. The probability q_1 and q_2 return the same leaf index is exponentially decreasing in d for a uniform random input. In particular this means that for the very first probe, the queries in $Q(b,I)$ will almost certainly read different cells, which is precisely the property we exploited above. If we pick a random sample of cells, there will be many queries in $Q(b,I)$ that have their first probe in the sample. To handle the remaining $t-1$ probes, we follow [12] and extend the cell probe model with the concepts *published bits* and *accepted queries*. A data structure is allowed to publish bits at preprocessing time that the query algorithm may read free of charge. After inspecting a given query and the published bits, a data structure can choose to reject the query and not return an answer. Otherwise, the query is accepted and the algorithm must output the correct answer. Note that it is only allowed to reject queries before performing any probes.

The crucial idea is now the following: If the data structure has few published bits, then for most leaves $b \in [n]$, the published bits simply contains too little information to make the queries in $Q(b,I)$ probe the same cells. Thus for t rounds, we can pick a random sample of cells and *publish* their contents. For every accepted query, we check if its first probe is amongst the published cells. If so, we continue to accept it and may skip the first probe since we know the contents of the requested cell. Otherwise we simply reject it. If the published cell sets are small enough, there continues to be too little information in the published bits to make the queries in $Q(b,I)$ meet. Since this holds for all t probes, the argument above for the poorly performing data structures carry through and we get our lower bound.

2.2 Deriving the Lower Bound

With the ideas from Section 2.1 in mind, we present our technical lemma that allows us to publish bits for t rounds to eliminate probes while ensuring that most leaves are still the answer to many accepted queries. Before we present the lemma, consider partitioning the ball-inheritance tree into into $\lg n/Y$ disjoint layers of Y consecutive tree levels and group the accepted queries by these layers. Think of Y as looking at the queries at a given zoom level. To measure how much information we have left about the different leaves, we count for each leaf $b \in [n]$ how many layers that have at least one accepted query with b as its answer. If this count is large, then intuitively the answers to all accepted queries carry much information.

Formally, given a data structure for the ball-inheritance problem, define for every $1 \le Y \le \lg n$ and index $i \in [\lg n/Y]$ the *query-support set* of a leaf $b \in [n]$ on an input I as the set $Q_i^Y(b,I)$ of accepted queries in the tree levels $\{iY, \ldots, (i+1)Y-1\}$ that has b as its answer. Observe that $|Q_i^Y(b,I)| \in \{0,\dots,Y\}$ since there is precisely one query in each tree level that has b as its answer (it may be less than Y since some queries might be rejected). Define also the Y-*level-support* of an input I, denoted $L^Y(I)$, as the the number of pairs (b,i) such that $Q_i^Y(b,I)$ is non-empty.

With this notation in hand we are ready to state our main Probe Elimination Lemma.

▶ **Lemma 4.** *Let $\mathcal{I}$ be a set of inputs to the ball-inheritance problem where $|\mathcal{I}| \geq n!/2^n$. Assume a ball-inheritance data structure uses S cells of w bits, answers queries in t probes, has $p < n\lg n/\lg^9 \lg n$ published bits and satisfies $L^Y(I) \geq (1-1/Z)n\lg n/Y$ for all $I \in \mathcal{I}$ for some parameters $Z \geq 2$ and $64\lg w \leq Y \leq \lg n/\alpha$, where $\alpha = (Sw\lg^{18}\lg n)/(n\lg n)$. Then there exists a subset of inputs $\mathcal{I}^* \subseteq \mathcal{I}$, with $|\mathcal{I}^*| \geq |\mathcal{I}|/2$, and another ball-inheritance data structure using S cells of w bits, answering queries in $t-1$ probes with $p+O(n\lg n/\lg^{10}\lg n)$ published bits, and satisfying $L^{\alpha Y}(I) \geq (1-1/Z-1/\lg\lg^3 n)n\lg n/(\alpha Y)$ for all $I \in \mathcal{I}^*$.*

In laymans terms, the lemma states that we can decrease the number of probes of a data structure by one, while only increasing the published bits with a lower order term. When we do this, we maintain the essential property that the leaves still have high support, just on a coarser zoom level. The Z factor is basically just a *dirt* factor. The proof of Lemma 4 can be found in the full version of the paper. In the following we use Lemma 4 to prove our main result, Theorem 3.

Assume for contradiction that a ball-inheritance data structure exists satisfying $t = o(\lg\lg_w n/\lg\alpha)$, where $\alpha = (Sw\lg^{18}\lg n)/(n\lg n)$. We proceed by repeatedly applying Lemma 4 to eliminate all t probes of the data structure. In order to guarantee we can apply Lemma 4 t times, we check the conditions for applying it. The conditions involve the number of published bits p, the parameters Z and Y and $|\mathcal{I}|$. The values of these parameters will change for each application, thus we use $p^{(i)}, Z^{(i)}, Y^{(i)}$ and $|\mathcal{I}^{(i)}|$ to denote these parameters just before the i'th invocation of the lemma. For the first round, we have $p^{(1)} = 0$ and $|\mathcal{I}^{(1)}| = n!$. Note also that $L^Y(I) = n\lg n/Y$ for any Y before the first round. Thus we choose $Y^{(1)} = 64\lg w$ to satisfy the conditions $64\lg w \leq Y^{(1)} \leq \lg n/\alpha$. This also means that we are free to choose $Z^{(1)} \geq 2$ as we wish. We simply let $Z^{(1)} = \lg^3\lg n$. Examining the lemma, we conclude that our parameters evolve in the following way (assuming we do not violate any of the conditions):

$$p^{(1+i)} = O(i(n\lg n/\lg^{10}\lg n)), \quad |\mathcal{I}^{(1+i)}| \geq n!/2^i, \quad Y^{(1+i)} = 64\lg w\cdot\alpha^i, \quad Z^{(i)} \geq \lg^3\lg n/i.$$

Since we assumed $t = o(\lg\lg_w n/\lg\alpha)$, this means that

$$p^{(1+t)} = o(n\lg n/\lg^9\lg n), \quad |\mathcal{I}^{(1+t)}| \geq n!/\lg n, \quad Y^{(1+t)} = o(\lg n), \quad Z^{(1+t)} \geq \lg^2\lg n.$$

We conclude that we can apply our lemma for t rounds under the contradictory assumption. Furthermore, the data structure we are left with answers queries in 0 probes on a subset $\mathcal{I}^* = \mathcal{I}^{(1+t)}$ of inputs, where $|\mathcal{I}^*| \geq n!/\lg n$. It has $o(n\lg n/\lg^9\lg n)$ published bits and there is some $Y^* = o(\lg n)$ such that $L^{Y^*}(I) \geq (1-1/\lg^2\lg n)n\lg n/Y^*$ for all $I \in \mathcal{I}^*$. That this is contradictory should not come as a surprise: our 0-probe data structure is capable of answering queries about almost all leaves using only the $o(n\lg n/\lg^9\lg n) \ll \lg|\mathcal{I}^*|$ published bits. The formal argument we use to reach the contradiction is as follows: we show that the 0-probe data structure can be used to uniquely encode every input $I \in \mathcal{I}^*$ into a bit string of length less than $\lg(|\mathcal{I}^*|) = \lg(n!) - \lg\lg n$ bits. This gives the contradiction since there are fewer such bit strings than inputs. We present the encoding and decoding algorithms in the following:

Encoding. Let $I \in \mathcal{I}^*$ be an input to encode. Observe that if we manage to encode the leaf index reached by each ball in the root node's list of balls, then that information completely specifies I. With this in mind, we implement the 0-probe data structure above on I and proceed as follows:

1. First we write down the published bits on input I. This cost $o(n \lg n / \lg^9 \lg n)$ bits.
2. For $i = 1, \dots, n$ consider the i'th ball in the root node's list of balls. Let b_i denote the index of the leaf reached by that ball. We write down $\lg n/2$ bits for each such ball in turn, specifying the subtree at depth $\lg n/2$ that contains the leaf b_i. This costs $n \lg n/2$ bits.
3. Finally, we go through all leaf nodes from left to right. For a leaf b, we check if there is an accepted query returning b as its answer amongst all queries in all nodes of depth at most $\lg n/2$. If so, we continue to the next leaf. Otherwise we write $\lg n$ bits denoting the rank of the ball reaching b amongst balls the root node's list of balls. If X is the number of leaves with no accepted query reporting it in tree levels $\{0, \dots, \lg n/2\}$, this step costs $X \lg n$ bits.

Decoding. To recover I from the above encoding, we do as follows.

1. We first go through all nodes v of depth d for $d = 0, \dots, \lg n/2$. For each such node, let $q_1^v, \dots, q_{n/2^d}^v$ denote the queries we can ask at node v, i.e. q_i^v asks for the leaf reached by the i'th ball in v's list of balls. We run the query algorithm for each such query in turn using the published bits written in step 1. of the encoding procedure. Since our data structure makes 0 probes, this returns the answer to each such accepted query, i.e. we have collected a set $\mathcal{Q}$ of pairs (q_i^v, b) such that b is the index of the leaf reached by the i'th ball in v's list of balls.
2. We now partition $\mathcal{Q}$ into one set $\mathcal{Q}_b$ for each leaf index b. The set $\mathcal{Q}_b$ contains all pairs $(q_i^v, b') \in \mathcal{Q}$ such that $b' = b$. There are precisely X empty such sets.
3. For each empty set $\mathcal{Q}_b$ in turn (ordered based on b), we use the bits written in step 3. of the encoding procedure to recover the rank of the ball reaching b amongst all balls in the root node's list of balls.
4. For every non-empty set $\mathcal{Q}_b$, pick an arbitrary pair $(q_i^v, b) \in \mathcal{Q}_b$. From this pair alone, we know that the ball reaching b has rank i amongst all balls ending in a leaf of the subtree rooted at v. Now initialize a counter Δ to 0. Using the bits written in step 2. of the encoding procedure, we now go through all balls in the root node's list of balls in turn. For the r'th ball, $r = 1, \dots, n$, we check the $\lg n/2$ bits written for it and from this we determine if the ball reaches a leaf in v's subtree (possible since v can only be in the first $\lg n/2$ levels by construction). If so, we increment Δ by 1. If this causes Δ to reach i, we conclude that the ball ending in b has rank r in the root node's list of balls.
5. From the above steps, we have for every leaf b determined the rank of the ball reaching it amongst all balls in the root node's list of balls. This information completely specifies I.

Analysis. The encoding above costs

$$o(n \lg n / \lg^9 \lg n) + n \lg n/2 + X \lg n$$

bits. Now observe that if $\mathcal{Q}_b$ is empty for a leaf index b, this means $Q_i^{Y^*}(b, I)$ is empty for every $i \in \{0, \dots, \lg n/(2Y^*) - 1\}$. This gives $L^{Y^*}(I) \leq n \lg n/Y^* - X(\lg n/(2Y^*))$. But we know $L^{Y^*}(I) \geq (1 - 1/\lg^2 \lg n) n \lg n/Y^*$ and we conclude

$$X \leq 2n/\lg^2 \lg n.$$

The encoding thus costs

$$n \lg n/2 + O(n \lg n/\lg^2 \lg n).$$

Since $\lg(n!) = n \lg n - O(n)$, we conclude that our encoding uses no more than

$$\lg(|\mathcal{I}^*|) - n \lg n/2 + O(n \lg n/\lg^2 \lg n) = \lg(|\mathcal{I}^*|) - \Omega(n \lg n)$$

bits, which completes the proof.

We have thus shown $t = \Omega(\lg \lg_w n/\lg \alpha)$ where $\alpha = (Sw \lg^{18} \lg n)/(n \lg n)$. In the word-RAM, we assume $w = \Theta(\lg n)$ and the lower bound becomes the claimed $t = \Omega(\lg \lg n/(\lg(S/n) + \lg \lg \lg n))$.

2.3 Eliminating Probes

In this section we prove Lemma 4. Recalling the intuition presented in Section 2.1, we want to show that for a data structure with few published bits, the different accepted queries reporting a fixed leaf index $b \in [n]$ have to probe distinct cells in their first probe. If we can establish this, we can pick a small random sample of memory cells and there are many of the accepted queries that make their first probe in the sample.

To formalize the above, we define a memory cell c to be k-popular on input I, if at least k accepted queries make their first probe in c on I. Define for every query-support set $Q_i^Y(b, I)$ the *cell-support set* $C_i^Y(b, I)$ as the set of memory cells that are read in the first probe of a query in $Q_i^Y(b, I)$ on input I. We measure to what extend the queries in $Q_i^Y(b, I)$ probe distinct cells using the following definitions.

▶ **Definition 5.** For an input I and value $Y \in \{1, \dots, \lg n\}$, we say that a pair (b, i), where $b \in [n]$ and $i \in \{0, \dots, \lg n/Y - 1\}$, is Y-*scattered* on input I if one of the following three holds:

1. $Q_i^Y(b, I)$ contains a query making 0 probes.
2. $C_i^Y(b, I)$ contains a w^3-popular cell.
3. $|C_i^Y(b, I)| \geq \alpha/\lg^6 \lg n$.

We define the Y-*scatter-number* of I, denoted $\Gamma^Y(I)$, as the number of pairs (b, i) that are Y-scattered on I.

If a query makes zero probes, all the information needed to answer it is contained in the already published bits. There are very few w^3-popular cells, so publishing all of them costs few bits. Most interestingly, if the queries in each support $Q_i^Y(b, I)$ set probe many distinct cells in their first probe (case 3.), then a random sample of cells will contain at least one of these cells with good probability.

We need the following lemma that captures the correspondence between large support on zoom level Y, the properties maintained by our Probe Elimination Lemma, and large scattering on a higher zoom level αY.

▶ **Lemma 6.** *Let $\mathcal{I}$ be a set of inputs to the ball-inheritance problem where $|\mathcal{I}| \geq n!/2^n$. Assume a ball-inheritance data structure uses S cells of w bits, has $p < n \lg n/\lg^9 \lg n$ published bits and satisfies $L^Y(I) \geq (1 - 1/Z) n \lg n/Y$ for all $I \in \mathcal{I}$ for some parameters $Z \geq 2$ and $64 \lg w \leq Y \leq \lg n/\alpha$, where $\alpha = (Sw \lg^{18} \lg n)/(n \lg n)$. Then there exists a subset $\mathcal{I}^* \subseteq \mathcal{I}$ of inputs such that $|\mathcal{I}^*| \geq |\mathcal{I}|/2$ and*

$$\Gamma^{\alpha Y}(I) \geq \left(1 - \frac{1}{\lg^3 \lg n}\right) \cdot \left(1 - \frac{1}{Z}\right) \cdot \frac{n \lg n}{\alpha Y}.$$

for all $I \in \mathcal{I}^*$.

We refer to the full version for the proof of Lemma 6, and use it to Prove Lemma 4 instead. Let $\mathcal{I}$ be a set of at least $n!/2^n$ inputs to the ball inheritance problem. Assume furthermore we are given a ball inheritance data structure that uses S cells of w bits, answers queries in t probes, has $p < n \lg n / \lg^9 \lg n$ published bits, and satisfies $L^Y(I) \geq (1 - 1/Z) n \lg n / Y$ for all $I \in \mathcal{I}$ for some parameters $Z \geq 2$ and $64 \lg w \leq Y \leq \lg n / \alpha$ where $\alpha = (Sw \lg^{18} \lg n)/(n \lg n)$ (as in the assumptions of Lemma 4 and Lemma 6). Let $\mathcal{I}^* \subseteq \mathcal{I}$ be the subset of $\mathcal{I}$ promised by Lemma 6. Our goal is to construct a new ball inheritance data structure answering queries in $t-1$ probes for the inputs $\mathcal{I}^*$ while publishing few bits and keeping $L^{\alpha Y}(I)$ fairly large for all $I \in \mathcal{I}^*$. Given an input $I \in \mathcal{I}^*$, we keep the (old) p published bits and publish some additional bits from our data structure as follows:

1. First we publish all memory cells that are w^3-popular on input I. Since there are no more than $n \lg n$ accepted queries, there are no more than $n \lg n / w^3$ popular cells. The addresses and contents of all such cells can be described using $O(n \lg n / w^2) = O(n / \lg n)$ bits.
2. Next we collect all αY-scattered pairs (b, i) for input I. We remove those pairs for which $Q_i^{\alpha Y}(b, I)$ contains a query making 0 probes, or $C_i^{\alpha Y}(b, I)$ contains a w^3-popular cell. By definition, the remaining αY-scattered pairs (b, i) must satisfy $|C_i^{\alpha Y}(b, I)| \geq \alpha / \lg^6 \lg n$. We now consider all subsets of $n \lg n / (w \lg^{10} \lg n)$ memory cells and publish the subset $P^* \subseteq [S]$ for which most remaining pairs (b, i) satisfies $C_i^{\alpha Y}(b, I) \cap P^* \neq \emptyset$. Specifying the addresses and contents of cells in P^* costs $O(n \lg n / \lg^{10} \lg n)$ bits.

The query algorithm of our modified data structure is simple: We start running the old query algorithm with the p "old" published bits and stop once one of the following happens:

1. If the old query algorithm rejects the query, we also reject it.
2. If the old query algorithm answers the query without any probes, we know the answer to the query and return it.
3. Otherwise the old query algorithm makes at least one memory probe. The (address of the) first cell probed, denoted c, can be determined solely from the old published bits. Before making the actual probe, we check the newly published cells to see if c is amongst them. If so, we have the contents of c in the published bits and therefore skip the probe. We then continue executing the old query algorithm and have successfully reduced the number of probes by one. If c was not published, we simply reject the query.

Clearly our new data structure answers queries in $t-1$ probes and has $p + O(n \lg n / \lg^{10} \lg n)$ published bits. What remains is to argue that $L^{\alpha Y}(I)$ is high for all $I \in \mathcal{I}^*$ for this new data structure. To distinguish the new data structure and the old, we use $\bar{L}, \bar{Q}$ and $\bar{\Gamma}$ in place of L, Q and Γ when referring to the new data structure. L, Q and Γ refers to the old data structure.

So fix an $I \in \mathcal{I}^*$. By our choice of $\mathcal{I}^*$, we have

$$\Gamma^{\alpha Y}(I) \geq \left(1 - \frac{1}{\lg^3 \lg n}\right) \cdot \left(1 - \frac{1}{Z}\right) \cdot \frac{n \lg n}{\alpha Y}.$$

i.e. the old data structure has many pairs (b, i) that are αY-scattered on input I. By definition of $\bar{L}^{\alpha Y}(I)$, we need to lower bound the number of such pairs (b, i) that have $\bar{Q}_i^{\alpha Y}(b, I)$ non-empty, i.e. at least one query reporting b in tree-levels $\{i\alpha Y, \ldots, (i+1)\alpha Y - 1\}$ is accepted by our new query algorithm. For this, let (b, i) be a pair that was αY-scattered for I in the old data structure. By definition of αY-scattered we know that $Q_i^{\alpha Y}(b, I)$ is non-empty. Now observe that if $Q_i^{\alpha Y}(b, I)$ contains a query that made 0 probes, then that query is also in $\bar{Q}^{\alpha Y}(b, I)$. Similarly if $Q_i^{\alpha Y}(b, I)$ contains a query making its first probe in a w^3-popular cell, then that query is also in $\bar{Q}_i^{\alpha Y}(b, I)$ since we publish all w^3-popular cells. Hence $\bar{Q}_i^{\alpha Y}(b, I)$ can be empty only if $Q_i^{\alpha Y}(b, I)$ contains no queries making 0 probes and no queries probing a w^3-popular cell. Since (b, i) was αY-scattered, this implies $|C_i^{\alpha Y}(b, I)| \geq \alpha/\lg^6 \lg n$. Furthermore, we get that $\bar{Q}_i^{\alpha Y}(b, I)$ becomes empty only if none of these cells are in P^*.

Letting $\mu = n \lg n/(w \lg^{10} \lg n)$, we get that $C_i^{\alpha Y}(b, I)$ has a non-zero intersection with the following fraction of μ-sized cell sets:

$$1 - \frac{\binom{S-|C_i^{\alpha Y}(b,I)|}{\mu}}{\binom{S}{\mu}} \geq 1 - \frac{(S - \alpha/\lg^6 \lg n)!(S-\mu)!\mu!}{S!(S - \alpha/\lg^6 \lg n - \mu)!\mu!} \geq 1 - \frac{(S-\mu)^{\alpha/\lg^6 \lg n}}{(S - \alpha/\lg^6 \lg n)^{\alpha/\lg^6 \lg n}} =$$

$$1 - \left(\frac{S - \alpha/\lg^6 \lg n + \alpha/\lg^6 \lg n - \mu}{S - \alpha/\lg^6 \lg n}\right)^{\alpha/\lg^6 \lg n} = 1 - \left(1 - \frac{\mu - \alpha/\lg^6 \lg n}{S - \alpha/\lg^6 \lg n}\right)^{\alpha/\lg^6 \lg n}.$$

Since $\alpha = (Sw \lg^{18} \lg n)/(n \lg n) = S \lg^8 \lg n/\mu \ll \mu/2$, this is at least a

$$1 - \left(1 - \frac{\mu}{2S}\right)^{\alpha/\lg^6 \lg n} \geq 1 - \exp\left(-\alpha\mu/(2S \lg^6 \lg n)\right) \geq 1 - 1/\lg n$$

fraction. Since we chose P^* to maximize the number sets $C_i^{\alpha Y}(b, I)$ having a non-empty intersection, we conclude that at least

$$\left(1 - \frac{1}{\lg n}\right) \cdot \left(1 - \frac{1}{\lg^3 \lg n}\right) \cdot \left(1 - \frac{1}{Z}\right) \cdot \frac{n \lg n}{\alpha Y} \geq \left(1 - \frac{1}{Z} - \frac{2}{Z \lg^3 \lg n}\right) \frac{n \lg n}{\alpha Y}$$

sets $\bar{Q}_i^{\alpha Y}(b, I)$ must be non-empty. Since $Z \geq 2$, we finally conclude

$$\bar{L}^{\alpha Y}(I) \geq \left(1 - \frac{1}{Z} - \frac{1}{\lg^3 \lg n}\right) \frac{n \lg n}{\alpha Y}.$$

References

1. Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. New data structures for orthogonal range searching. In *Proc. 41st IEEE Symposium on Foundations of Computer Science*, pages 198–207, 2000.
2. Lars Arge, Vasilis Samoladas, and Jeffrey Scott Vitter. On two-dimensional indexability and optimal range search indexing. In *Proc. 18th ACM Symposium on Principles of Database Systems*, pages 346–357, 1999.
3. Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.
4. Gerth Stølting Brodal and Kasper Green Larsen. Optimal planar orthogonal skyline counting queries. In *Proc. 14th Scandinavian Workshop on Algorithms Theory*, pages 110–121, 2014.
5. Timothy M. Chan, Kasper Larsen, and Mihai Pătraşcu. Orthogonal range searching on the ram, revisited. In *Proc. 27th ACM Symposium on Computational Geometry*, pages 354–363, 2011. See also arXiv:1011.5200.
6. Timothy M. Chan and Bryan T. Wilkinson. Adaptive and approximate orthogonal range counting. In *Proc. 24th ACM/SIAM Symposium on Discrete Algorithms*, pages 241–251, 2013.
7. Bernard Chazelle. Filtering search: a new approach to query answering. *SIAM Journal on Computing*, 15(3):703–724, 1986. **id**doi:10.1137/0215051.
8. Bernard Chazelle. Lower bounds for orthogonal range searching: I. the reporting case. *Journal of the ACM*, 37(2):200–212, 1990.

9 Johannes Fischer. Optimal succinctness for range minimum queries. In *Proc. 9th Latin American Theoretical Informatics Symposium*, pages 158–169, 2010.

10 Kasper Green Larsen. Higher cell probe lower bounds for evaluating polynomials. In *Proc. 53rd IEEE Symposium on Foundations of Computer Science*, pages 293–301, 2012.

11 Yakov Nekrich. Orthogonal range searching in linear and almost-linear space. *Computational Geometry: Theory and Applications*, 42:342–351, 2009.

12 Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Proc. 38th ACM Symposium on Theory of Computation*, pages 232–240, 2006.

13 Mihai Pătraşcu and Mikkel Thorup. Higher lower bounds for near-neighbor and further rich problems. *SIAM Journal on Computing*, 39(2):730–741, 2010.

14 Mihai Pătraşcu and Mikkel Thorup. Randomization does not help searching predecessors. In *Proc. 18th ACM/SIAM Symposium on Discrete Algorithms*, pages 555–564, 2007.

15 Yaoyu Wang and Yitong Yin. Certificates in data structures. In *Proc. 41st International Colloquium on Automata, Languages, and Programming*, pages 1039–1050, 2014.

16 Andrew Chi Chih Yao. Should tables be sorted? *Journal of the ACM*, 28(3):615–628, 1981.