Interpolation Search for Non-Independent Data

Erik D. Demaine[*] Thouis Jones[*] Mihai Pătraşcu[*]

Abstract

We define a deterministic metric of "well-behaved data" that enables searching along the lines of interpolation search. Specifically, define Δ to be the ratio of distances between the farthest and nearest pair of adjacent elements. We develop a data structure that stores a dynamic set of n integers subject to insertions, deletions, and predecessor/successor queries in $O(\lg \Delta)$ time per operation. This result generalizes interpolation search and interpolation search trees smoothly to nonrandom (in particular, non-independent) input data. In this sense, we capture the amount of "pseudorandomness" required for effective interpolation search.

1 Introduction

Interpolation search is a classic method for searching through ordered random data and attains a running time is $O(\lg \lg n)$, which is exponentially better than binary search. The original method [5] was analyzed only for uniformly distributed data [7, 4, 2]. Willard [6] later generalized the analysis to arbitrary "regular distributions", without the algorithm having to know the distribution. Mehlhorn and Tsakalidis [3] further generalized interpolation search to handle the dynamic case of insertions and deletions in addition to searches and to a wider class of "smooth" distributions. Andersson and Mattsson [1] further extend the technique of Mehlhorn and Tsakalidis to a larger class of distributions and better bounds on searches and updates.

All of these results rely on the data being drawn independently from some probability distribution. In this paper, we remove this independence assumption and instead capture the necessary properties purely deterministically via a sort of "pseudorandomness" measure. We start with the static case in Section 2 and generalize to the dynamic case in Section 3.

2 Searching Static Data

We first present a method for searching static data. Suppose we are given n distinct values $x_1 < x_2 < \cdots < x_n$ in sorted order. Define the maximum gap ratio Δ of the data as $\frac{\max(x_i - x_{i-1})}{\min(x_i - x_{i-1})}$. We make no requirements on the data being independently drawn.

Evenly subdivide the interval $[x_1, x_n]$ into n bins $B_1, B_2, \ldots, B_n$ each representing a range of size $\frac{x_n - x_1}{n}$. Each bin stores a balanced binary search tree for the elements lying within its range, plus the nearest neighbor above and below. Searching for an element y then proceeds by interpolating on y to find the bin B_i that it lies in, $i = \lceil \frac{y - x_1}{x_n - x_1} \rceil$, and performing a search in the binary search tree of that bin. The array of bins can be constructed in $O(n)$ time and space, assuming the data are already sorted.

Theorem 2.1. *The worst-case search time is $O(\lg \Delta)$.*

Proof. The maximum distance between two adjacent values is at least $\frac{x_n - x_1}{n}$. Given Δ, the minimum distance between two values must be at least $\frac{x_n - x_1}{n\Delta}$. Because the bins are of size $\frac{x_n - x_1}{n}$, no more than Δ values can land in a single bin, and the search time of the binary search tree in a bin occupied by Δ values is $O(\lg \Delta)$. ∎

We note the following interesting traits of this algorithm:

1. The algorithm is oblivious to the value of Δ.
2. The worst-case search time is also $O(\lg n)$ and thus $O(\lg \min\{\Delta, n\})$.
3. The algorithm reproduces the $O(\lg \lg n)$ performance of interpolation search on data drawn independently from the uniform distribution, by the following lemma:

Lemma 2.1. *For data drawn independently from the uniform distribution, $\Delta = O(\text{polylog}(n))$ with high probability.*

Proof. We use standard Chernoff bounds on n balls thrown into m bins. Consider the uniform distribution over $[0, u]$. If $m = O(n/\lg n)$, then every bin contains at least one ball with high probability. Thus, $\max_i(x_i - x_{i-1}) = O(\frac{u \lg n}{n})$ with high probability. If $m = \Omega(n \lg n)$, then every adjacent pair of bins contains at most one ball with high probability. Thus, $\min_i(x_i - x_{i-1}) = \Omega(\frac{u}{n \lg n})$ with high probability. Taking the ratio, $\Delta = O(\lg^2 n)$ with high probability. ∎

[*]MIT Computer Science and Artificial Intelligence Laboratory, {edemaine,thouis,mip}@mit.edu

3 Searching Dynamic Data

The dynamic version of our structure supports insertions, deletions, and searches in $O(\lg \Delta_{\max})$ time per operation, where $\Delta_{\max}$ is the largest value of Δ over the lifetime of the structure, and the insertion and deletion bounds are amortized. At any point in time, the structure will be valid for sets of data with Δ less than some $\hat{\Delta}$ for which the structure was built.

The only modification to the static structure is that it spans the larger range $[x_1 - L\hat{\Delta}, x_n + L\hat{\Delta}]$, where $L = x_n - x_1$. This superinterval is uniformly subdivided into n bins each of size $(2\hat{\Delta}+1)L/n$. Each bin stores a dynamic balanced binary search tree.

The structure is rebuilt if either $\frac{n}{2}$ updates (inserts or deletes) occur without a rebuild, or an update causes the Δ of the data to be larger than $\hat{\Delta}$ (i.e., when the structure is no longer valid). In the latter case, we rebuild the structure with $\hat{\Delta} = \max(\Delta_{\text{new}}, 2\hat{\Delta}_{\text{old}})$, where Δ_{new} is computed from the data, and $\hat{\Delta}_{\text{old}}$ was the value of Δ immediately before the rebuild.

We need two main properties: (a) the structure is always valid, i.e., no x_i outside the superinterval can be inserted into the structure, and (b) the number of values landing in a single bin is polynomial in $\hat{\Delta}$.

The following two theorems establish the validity of the structure during a set of n updates that do not cause $\hat{\Delta}$ to grow.

Theorem 3.1. *None of the $\frac{n}{2}$ operations immediately after a rebuild can add a value outside the range $[x_1 - L\hat{\Delta}, x_n + L\hat{\Delta}]$.*

Proof. Consider only the initial elements confined to $[x_1, x_n]$. The initial minimum separation between these elements is no more than $\frac{L}{n-1}$. After k deletions, the minimum separation is no more than $\frac{L}{n-k-1}$.

Given k deletions, there are $\frac{n}{2} - k$ insertions before a rebuild of the structure[1]. These insertions can be only $\frac{\hat{\Delta}L}{n-k-1}$ from the previous largest value in the structure. Therefore, the maximum value that can be inserted in $\frac{n}{2}$ editing operations is $x_n + (\frac{n}{2} - k)\frac{L\hat{\Delta}}{n-k-1}$, which is bounded above by $x_n + L\hat{\Delta}$ for $k \le \frac{n}{2}$. By symmetry, we can obtain a similar bound for the minimum value inserted. ∎

Theorem 3.2. *The maximum number of elements in a bin after n insertions or deletions is $O(\hat{\Delta}^2)$.*

Proof. Again, consider the initial elements in $[x_1, x_n]$. After $k < n-2$ deletions, the maximum separation between two of these elements is at least $\frac{L}{n-k-1}$. Therefore, the minimum separation between any two elements is at least $\frac{L}{(n-k-1)\hat{\Delta}}$. The bins are of size $\frac{L(2\hat{\Delta}+1)}{n}$, so the maximum number of elements that can end up in a single bin is $\frac{\hat{\Delta}(2\hat{\Delta}+1)(n-k-1)}{n} = O(\hat{\Delta}^2)$. ∎

Finally, we establish the desired time bounds:

Theorem 3.3. *The worst-case cost of searches and the amortized costs of insertions and deletions is $O(\lg \Delta_{\max})$ time per operation.*

Proof. The cost for searches follows from theorem 3.2, as does the immediate cost of insertions and deletions. We use a charging scheme against updates to amortize the cost of rebuilds.

The structure can be rebuilt in linear time. If the structure is rebuilt because of a sequence of $\frac{n}{2}$ updates without a rebuild, then the rebuild can be charged to these updates to give an amortized cost of $O(1)$.

If a rebuild occurs because of a change in $\hat{\Delta}$, then we charge the cost of the rebuild to the original insertions of the elements currently in the structure. The cost of inserting one such element was $O(\lg \hat{\Delta}_{\text{old}})$. Increasing $\hat{\Delta}$ raises $\Delta_{\max}$ and hence the amortized cost of the original insertion by $\Theta(\lg \hat{\Delta} - \lg \hat{\Delta}_{\text{old}}) = \Theta(\lg \frac{\hat{\Delta}}{\hat{\Delta}_{\text{old}}}) = \Omega(1)$ because $\hat{\Delta} \ge 2\hat{\Delta}_{\text{old}}$, to which we can charge the rebuild cost for that element. ∎

If the value of $\Delta_{\max}$ is known *a priori*, then we can always build the data structure with $\hat{\Delta} = \Delta_{\max}$ and avoid rebuilding because of changing $\hat{\Delta}$. A standard de-amortization of the remaining global rebuilds, from performing $\frac{n}{2}$ updates, yields worst-case time bounds.

[1] Because the minimum separation cannot grow from insertions, it is sufficient to consider deletions followed by insertions.

References

[1] A. Andersson and C. Mattsson. Dynamic Interpolation Search in $o(\log\log n)$ Time. *Proceedings of ICALP*, pages 15–27, 1993.

[2] G. Gonnet, L. Rogers, and G. George. An algorithmic and complexity analysis of interpolation search. *Acta Inf.*, 13(1):39–52, 1980.

[3] K. Mehlhorn and A. Tsakalidis. Dynamic Interpolation Search. *Journal of the ACM*, 40(3):621–634, July 1993.

[4] Y. Perl, A. Itai, and H. Avni. Interpolation search – A $\log\log N$ search. *CACM*, 21(7):550–554, 1978.

[5] W. W. Peterson. Addressing for Random-Access Storage. *IBM J. Res. Development*, 1(4):130–146, 1957.

[6] D. Willard. Searching unindexed and nonuniformly generated files in log log N time. *SIAM J. Comput.*, 14:1013–1029, 1985.

[7] A. C. Yao and F. F. Yao. The complexity of searching an ordered random table. *Proceedings of FOCS*, pages 173–177, 1976.