Flexible Interpolated-Binary Search over Sorted Sets

Biagio Bonasera, Francesco Pagano[1], and Alessandro Provetti[2]

[1] Dip. di Scienze dell'Informazione, Università degli Studi di Milano
Via Comelico, 39. I-20135 Milan, Italy
francesco.pagano@unimi.it
[2] Dip. di Fisica, Università degli Studi di Messina.
Sal. Sperone 31. S. Agata di Messina, I-98166 Italy
ale@unime.it

Abstract. We revisit the classical algorithms for searching over sorted sets and introduce an algorithm refinement, called adaptive search, that combines the good features of Interpolation search and those of Binary search. W.r.t. Insertion sort, only a constant number of extra comparisons is introduced. Yet, under several relevant input data distributions our algorithm shows average case cost comparable to that of Interpolation Search, i.e., $O(\log \log n)$ while the worst case cost is always in $O(\log n)$, as with Binary search. This result compares well with the traditional result of Santoro and Sidney Interpolation-Binary Search and the recent approach of Demaine et al.[**?**] on searching non-independent data.

1 Introduction

We revisit the classical algorithms for searching over sorted sets and introduce a new algorithm, called adaptive search (AS), that combines the good features of Interpolation search and those of Binary search [**?**]. W.r.t. Insertion sort, only a constant number of extra comparisons is introduced. Yet, under several relevant input data distributions our algorithm shows average case cost comparable to that of Interpolation Search, i.e., $O(\log \log n)$ while the worst case cost is always in $O(\log n)$, as with Binary search. This result compares well with the traditional result of Santoro and Sidney [**?**] and the recent approach of Demaine et al.[**?**] on searching non-independent data.

Definition 1. *The membership problem is defined as follows.* ***instance****:*

- $\mathcal{L} = \{a_1, a_2, ..., a_n\}$*, a set of* n *distinct, sorted elements, with* $a_i < a_{i+1}$ *,* $1 \leq i \leq n-1$*;*
- *an element key*

question*: Does key belong to the set represented by* $\mathcal{L}$ *(key* $\in \mathcal{L}$*) ?*

In literature there exist two classical algorithms for searching over sorted sets: binary search (BS) [**?**] and interpolation search (IS) [**?**]; both take advantage of the ordering of the instance to minimize the number of keys that must be accessed.

In BS, the worst-case computational cost is $O(log\ n)$; This result is independent from data distribution over the instance. Notice that in search the worst-case is rather important as it corresponds to an unsuccessful membership query.

Vice versa, the IS algorithm is more efficient than BS on *quasi-uniform* data distributions[3] with average case computational complexity in $O(\log \log n)$. Unfortunately, IS degrades to $O(n)$ when data is not uniformly distributed. This is particularly inconvenient when searching over indexes of large databases, it is fundamental to minimize the number of accesses[4].

2 The adaptive search algorithm

The crucial variables of this algorithm are described next. First, given $\mathcal{S} = \{A[bot], ..., A[top]\}$ we define:

key: the element being searched;
A[bot]: the minimum element of the subset (at the beginning, $bot = 1$);
A[top]: the maximum element of the subset (at the beginning, $top = |\mathcal{L}|$);
A[next]: interpolation element, i.e. what IS would choose, and
A[med]: the el. halfway between *bot* and *top*, i.e., what BS would choose.

Our algorithm consist, essentially, of a while cycle. At each iteration, we set:

$$next = \left\lfloor \frac{key - A[bot]}{A[top] - A[bot]} * (top - bot) + bot \right\rfloor$$

Variable *next* defined above contains the index value that bounds the array segment on which our AS algorithm will recur on. As with IS, the instance is now *clipped:*

$$\mathcal{S}' = \begin{cases} \{A[bot], ..., A[next]\} \text{ if } A[bot] \leq key \leq A[next] \\ \\ \{A[next], ..., A[top]\} \text{ otherwise} \end{cases}$$

To do so, we set the new boundary of the segment:

[3] By quasi-uniform data distribution here the following property is intended: the distance between two consecutive values in the input set does not vary much wrt. the considered values. In our benchmark we replicated this behavior by generating random instances with the following assignment: $A[1] = 1$; $A[i+1] := A[i] + random(16) + 1$.

[4] In this discussion we do not consider the advanced techniques, viz. the exploitation of locality, that underlie search over large database indexes.

$$\begin{cases} top = next \text{ if } A[bot] \leq key \leq A[next] \\ \\ bot = next \text{ otherwise} \end{cases}$$

The computation is now restricted to the segment that would have been considered by IS; then, the median point is computed over such restricted segment, rather than on the whole input. Vice versa, if interpolation returns a shorter interval than BS would have, we eventually carry on with an IS search step:

we verify: if $|\mathcal{S}'| > \frac{top-bot}{2}$ then $next = med = bot + \frac{top-bot}{2}$;
if $key = A[next]$ then key is found and we terminate;
if $key > A[next]$ then $bot = next + 1$;
if $key < A[next]$ then $top = next - 1$.

At the end of the iteration, $\mathcal{S}' = \{A[bot], ..., A[top]\}$, and, clearly, $|\mathcal{S}'| < \frac{|\mathcal{S}|}{2}$. Finally:

if $A[bot] < key < A[top]$ then we iterate search on $\mathcal{S}'$;
else $key \notin \mathcal{L}$, hence the algorithm terminates.

2.1 Computational cost of AS

Standard cost analysis techniques yield the following results:

- Best Case: *key* is found, within a constant number of comparisons: $O(1)$;
- Average Case: values are uniformly distributed; hence, AS executes exactly as IS so its cost is in $O(\log \log n)$;
- Worst Case: The values are unevenly distributed; hence, the interval found by the BS technique is always the shortest. As a result, AS will execute essentially the same search as BS, with equal $O(\log n)$ time complexity.

3 Relation with literature

Our solution was conceived independently from the earlier work of Santoro and Sidney [**?**] who devised a very similar solution for the blending together of IS and BS. Although the asymptotically complexity is the same, there are some subtle differences between their solution and ours.

Santoro-Sidney algorithm, let's call it IBS, is based on the idea that interpolation search is useful, from the point of view of costs, only when the array searched is larger than a given threshold. When the considered search segment is smaller than a user-defined threshold (S), binary search is applied unconditionally. Vice-versa, over that threshold, an interpolation search step is applied, followed eventually by a binary search step.

Unlike IBS, our algorithm makes, at each level of its iteration, a choice about which *dicothomy* to apply. Hence, it is possible to show that for any input AS takes less elementary operations than IBS. We have sought a statistical confirmation of this fact by running a set of experiment over random-generated ordered sets. For the time being we limited the testing to successful queries with parameter θ set to 0.5, 1, 2 and 4, respectively.

4 Experimental validation and concluding remarks

We have implemented AS in order to test its efficiency (defined as average runtime) vis-à-vis those in literature. As a benchmark, we considered instances (ordered arrays) of Java double data type, double-precision 64-bit IEEE 754 floating point. The size of each instance was set to 10^i with $i = \{2, \ldots 8\}$. Instances were randomly generated, with the following distribution types[5]: uniform sparsity, increasing sparsity, stepwise sparsity and Paretian.

The results of experiments described in the previous section lead us to draw the following conclusions:

1. The performances of our AS algorithm vis-à-vis those IS and BS are very good and improve as n grows;
2. The number of accesses needed by AS is less than those of BS. The cost analysis of IS suggests that on certain instances, e.g., when sparsity grows stepwise, our algorithm needs between $\log n$ and $2 \log n$ accesses.
3. on all benchmarks our algorithm almost always outperforms both IS and BS.
4. our method for selecting the search interval succeeds in preventing the irregularities of data distribution from affecting performances; indeed, the number of accesses required remains $\cong \log \log n$.
5. Interestingly, while the asymptotically complexity of our AS algorithm is the same as Santoro's IBS, we have found that -on relatively diverse benchmarks- AS has often needed half or less of the memory accesses than IBS.

Further studies are needed in order to separate this result from a possible positive *bias* of the benchmark. An interesting question for us is whether instances that elicit the worst case ($2 \log n$ comparisons) can actually be found. Finally, the presented results are likely to be confirmed for search dictionaries (e.g., in [**?**]).

[5] The precise definition of the distributions will be in the full version of the article.