Journal of Discrete Algorithms 30 (2015) 128–133

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Adaptive search over sorted sets

Biagio Bonasera [a], Emilio Ferrara [b], Giacomo Fiumara [a], Francesco Pagano [c], Alessandro Provetti [a,*]

[a] Dept. of Mathematics and Informatics, Univ. of Messina, V.le F. Stagno D'Alcontres 31, I-98166 Messina, Italy
[b] Center for Complex Networks and Systems Research, School of Informatics and Computing, Indiana Univ., Bloomington, USA
[c] Dept. of Informatics, Univ. of Milan, Via Comelico 39, I-20135 Milan, Italy

ARTICLE INFO

Article history:
Accepted 19 November 2012
Available online 7 January 2015

Keywords:
Sorting
Searching sorted sets

ABSTRACT

We revisit the classical algorithms for searching over sorted sets to introduce an algorithm refinement, called Adaptive search, that combines the good features of Interpolation search and those of Binary search. W.r.t. Interpolation search, only a constant number of extra comparisons is introduced. Yet, under diverse input data distributions our algorithm shows costs 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. On benchmarks drawn from large datasets, both synthetic and real-life, Adaptive search scores better times and lesser memory accesses even than Santoro and Sidney's Interpolation–Binary search.

1. Introduction

We revisit the classical algorithms for searching over sorted sets to introduce a new algorithm, called Adaptive search (AS), that combines the good features of Interpolation search and those of Binary search [3].

The membership problem can be formally defined as follows.

Instance:

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

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

There exist two classical algorithms for searching over sorted sets: Binary search (BS) [3] and Interpolation search (IS) [1]; 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 of 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 Interpolation search algorithm is more efficient than BS when the elements of $\mathcal{S}$ are distributed uniformly or *quasi-uniformly* [1] over the $[a_1, a_n]$ interval; the computational cost is in $O(\log\log n)$.

* Corresponding author.
E-mail addresses: ferrarae@indiana.edu (E. Ferrara), gfiumara@unime.it (G. Fiumara), francesco.pagano@unimi.it (F. Pagano), ale@unime.it (A. Provetti).

[1] By quasi-uniform data distribution we intend, informally, that the distance between two consecutive values of $\mathcal{S}$ does not vary much.

http://dx.doi.org/10.1016/j.jda.2014.12.007
1570-8667/© 2015 Elsevier B.V. All rights reserved.

Unfortunately, Interpolation search degrades to $O(n)$ when data is not uniformly distributed (in the sense above). This is particularly inconvenient when searching over indexes of large databases, where it is crucial to minimize the number of accesses. [2]

In this work we propose an algorithm, called Adaptive search (AS) that refines Interpolation search and minimizes the number of memory accesses needed to complete a search. AS is **adaptive** to the values by means of a *mixed* behavior: it combines the independence from the distribution of BS with the good average costs of IS.

W.r.t. Interpolation search, AS requires only a constant number of extra comparisons. Yet, under several relevant input data distributions our algorithm shows average case costs comparable to those of interpolation, i.e., $O(\log\log n)$, while the worst-case cost remains in $O(\log n)$, as with Binary search.

Comparison with a more recent literature is also encouraging: both on synthetic and real datasets AS has better times and lesser memory accesses than Santoro and Sidney's Interpolation–Binary search [8]. Also, it is easier to implement and more broadly applicable that the approach of Demaine et al. [5] to searching non-independent data.

2. The adaptive search algorithm

Given an ordered set $\mathcal{S}$, allocated on an array A, and an element *key* that is searched, we define the following:

A[bot]:	the minimum element of the subset (at the beginning, $bot = 1$);
A[top]:	the maximum element of the subset (at the beginning, $top = \|\mathcal{S}\|$);
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 consists, essentially, of a while cycle. At each iteration, we consider $\mathcal{S} = \{A[bot], .., A[top]\}$ and we set:

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

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

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

To do so, we set the new boundaries of the segment containing $\mathcal{S}'$:

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

The computation is now restricted to the segment that would have been considered by IS. Next, 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 keep the result of the interpolation step:

```
if $|\mathcal{S}'| > \frac{|\mathcal{S}|}{2}$ then $next = med = bot + \frac{top-bot}{2}$;
    elseif $key = A[next]$ then $key$ is found and we terminate;
        elseif $key > A[next]$ then $bot = next + 1$;
            else $top = next - 1$ (must be $key < A[next]$).
```

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

```
if $A[bot] < key < A[top]$ then iterate search on $\mathcal{S}'$;
    else $key \notin \mathcal{S}$ and we terminate with no.
```

From the point of view of computational costs, we could summarize the following: our algorithm may spend up to double number of operations than IS in carefully finding out the best halving of the search segment, which in turn will mean that less iterations shall be needed to complete. By means of standard cost analysis techniques, we have the following results:

- Best case: *key* is found, with a constant number of comparisons: $\Theta(1)$;
- Worst case: the intervals between 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 (but more operations at each level), and

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