Lazy Selection

Carlos Requena López

December 5, 2018

1 Introduction

This assignment tries to empirically verify Theorem 3.5 (here 1) from [1, p. 49], running the **LazySelect** algorithm for different values of n and k to establish the asymptotic behaviour of the number of comparisons performed.

The **LazySelect** algorithm finds the kth smallest element in a set S (assumed to have a total order) of size n. It is essentially outputs the element with rank k. Concerning notation, $A_{(i)}$ will refer to the element with rank i in set A and $r_A(j)$ will refer to the rank of element j in set A.

2 Analysis and expectations

We detail the steps of the aforementioned algorithm in Algorithm 1.

Algorithm 1: LazySelect

Input: An (unsorted) array S of size n and and integer $k < n$

Output: The element in S that has rank k

1 Take a random sample of $n^{3/4}$ elements of S. Call it R;

2 Sort R;

3 Define $x = kn^{-1/4}$. With $\ell = \max\{\lfloor x - \sqrt{n} \rfloor, 1\}$ and $h = \min\{\lceil x - \sqrt{n} \; \rceil, n^{3/4}\}$;

4 Assign $a = R_{(\ell)}$ and $b = R_{(h)}$;

5 Determine the rank of a and b in S. That is, find $r_S(a)$ and $r_S(b)$;

6 **if** $k < n^{1/4}$ **then**

  $P = \{y \in S \mid y \leq b\}$

else if $k > n - n^{1/4}$ **then**

  $P = \{y \in S \mid y \geq a\}$

else if $k \in [n^{1/4} \, n - n^{1/4}]$ **then**

  $P = \{y \in S \mid a \leq y \leq b\}$

end

7 Check whether $S_{(k)} \in P$ *and* $|P| \leq 4n^{3/4} + 2$. If not, repeat steps 1 through 6;

8 If the conditions are satisfied: sort P and look for $S_{(k)}$ in P;

We can make a few observations about this algorithm:

- By taking R, we hope it will be a good representative of the original set S.

- Any optimal sorting algorithm works for step 2. The number of comparisons performed is sublinear: $\mathcal{O}(n^{3/4}\log(n))$
- l and j

Also, as proved in [1, p. 49], we have:

Theorem 1 *With probability* $1 - \mathcal{O}(n^{-1/4})$, **LazySelect** *finds* $S_{(k)}$ *on the first pass, and thus performs only* $2n + o(n)$ *comparisons.*

The elements are put back to simplify the analysis, since the random variables used become independent.

x is a sort of rank scaling. Rank x is to R what k is to S (roughly).

3 Implementation

4 Results

5 Conclusion

References

[1] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995.

A Appendix - code listing

core.clj

```clojure
(ns lazy-selection.core
  (:gen-class)
  (:require [clojure.math.numeric-tower :as math]))

(defn foo
  "I don't do a whole lot."
  [x]
  (println x "Hello, World!"))

(defn random_array
  "Returns a random permutation of an array of n unique elements. In
 ↪ the
  range 1..inf, element i is chosen with probability p."
  [n p]
  (shuffle (take n (random-sample p (range)))))

(random_array 100 0.5)

(defn rank_of
  "Returns the rank of element e in array a. Takes O(n). Assumes
 ↪ element
```

```
  is in array, otherwise it returns the rank it would have if it was"
  [e a]
  (reduce (fn [e1 e2] (if (< e2 e) (+ e1 1) e1)) 0 a))

(defn R
  "Returns R, which are the randomly sampled n^{3/4} elements of array
  S"
  [S]
  (let [n (count S)
        sample (math/ceil (math/expt n (/ 3 4)))]
    (take sample (shuffle S))))



(defn -main [] (foo 0))
```