Assignment 13, Data-Structures

Oleg Sivokon

<2016-04-09 Sat>

Contents

1 Problems

1.1 Problem 1

Find tight bounds on the given recurrences. Assume $T(n)$ is constant for $n = 1$.

$$
\begin{aligned}
T(n) &= 8T\left(\frac{n}{2}\right) + n + n^3 \\
T(n) &= kT\left(\frac{n}{2}\right) + (k-2)n^3 \\
&\quad \textit{where } k \in \mathbb{Z} : k \geq 2 \\
T(n) &= 2T\left(\frac{n}{4}\right) + \sqrt{n} \times \lg n \\
T(n) &= T\left(n-1\right) + n \lg n + n \\
T(n) &= n^2\sqrt{n} \times T\left(\sqrt{n}\right) + n^5 \lg^3 n + \lg^5 n
\end{aligned}
$$

1.2 Problem 2

Find upper and lower bounds on:

$$
\begin{aligned}
T(n) &= 2T(\frac{n}{2}) + n^3 \\
T(n) &= T(\frac{9n}{10}) + n \\
T(n) &= 16T(\frac{n}{4}) + n^2 \\
T(n) &= 7T(\frac{n}{3}) + n^2 \\
T(n) &= 7T(\frac{n}{2}) + n^2 \\
T(n) &= 2T(\frac{n}{4}) + \sqrt{n} \\
T(n) &= T(n-1) + n \\
T(n) &= T(\sqrt{n}) + 1
\end{aligned}
$$

1.3 Problem 3

Suggest a data-structure with the following properties:

1. Populate in $O(n)$ time.
2. Insert in $O(n \lg n)$ time.
3. Extract minimal element in $O(\lg n)$ time.
4. Extract median element in $O(\lg n)$ time.
5. Extract maximal element in $O(\lg n)$ time.

1.3.1 Answer 6

There is a simple, but impractical way of doing this—have four heaps:

- A is a `min-heap` containing elements greater than median.
- B is a `max-heap` containing elments smaller than median.

- C is a `max-heap` tracking the heap A.
- D is a `min-heap` tracking the heap B.

Creation and insertion are essentially the same as they are in the regular `max-heap` and `mean-heap`. Median element is either the root of A or the root of B, depending on which heap has more elements. Maximum element is the root of C and minimal element is the root of D, so their extraction is just the glorified `extract-max` and `extract-min` correspondingly.

Tracking is achieved using the following mechanism: Each node in each heap has an additional field that has a position of the tracked node in the other heap in it. Once the position of the node is modified, in addition to `heapify-min` or `heapify-max`, the procedure also updates the index in the tracking node (this takes only constant time).

Whenever a node is deleted, it also needs to be deleted from the tracking heap. In this case, the rightmost element in the heap is placed in the cell previously occupied by the node being deleted. Then `heapify-min` or `heapify-max` is performed, depending on the kind of heap it was.

Note that this solution is impractical since it requires saving a lot of additional information, but if we were to relax the requirement of $O(n)$ allowing $O(n \lg n)$ for population, then we could use something like order-statistic tree.

1.4 Problem 4

1. Given binary heap A of size n prove that `extract-max` requires roughly $2 \lg n$ comparisons.
2. Write an alternative `extract-max` which only uses $\lg n + \lg \lg n + O(1)$ comparisons.
3. Improve the previous `extrac-max` s.t. its running time is $\lg n + \lg \lg \lg n + O(1)$ wrt. comparisons.
4. Is it possible to improve this procedure further? Is it worth it wrt. the amount of code that it requires?

1.5 Problem 5

1. Given a sequence of real numbers: $(a_0, a_1, a_2, \ldots, a_n)$ define:

$$m = \min\{a_i \mid 0 \leq i \leq n\}$$
$$M = \max\{a_i \mid 0 \leq i \leq n\}$$

   Show that there exist a_i, a_j s.t.:

$$\left|a_i - a_j\right| \leq \frac{M-m}{n}$$

2. Write the algorithm for finding the a_i, a_j from the question above.

1.5.1 Answer 5

Note that the sequence with the given M and m will have $M - m = k \geq n$ elements. Suppose now we arrange the elements in increasing order. Suppose, for contradiction, that none of the elements of the sequence satisfies $\left|a_i - a_j\right| \leq \frac{M-m}{n}$, then it also means that the difference between every two adjacent elements must be at least $\frac{M-m+\epsilon}{n}$ for some positive ϵ. since there are k pairs of adjoint elements, we get that:

$$\begin{aligned}\sum_{i=1}^{k}(a_i - a_{i-1}) &= \sum_{i=1}^{k}\left(\frac{M-m+\epsilon}{n}\right) \\ &= \frac{k}{n}(M-m) + k\epsilon \geq M - m\end{aligned}$$

contrary to assumed. Hence, by contradiction, there must be at least one pair a_i, a_j s.t. $\left|a_i - a_j\right| \leq \frac{M-m}{n}$.

1.5.2 Answer 6

The general idea for the algorithm is to normalize all members of the given sequence by subtracting the minimum and mutliplying by the ratio of one

less than the length of the sequence and the difference between the maximum and the minumum. Once done, do the insertion sort: two elements which fall in the same cell will have a distance between them less than the one between the minimum and the maximum divided into one less than the number of elements.

Algorithm 1 Find $x, y \in Elts$ s.t. $|x - y| \leq \frac{\max(Elts) - \min(Elts)}{|Elts| - 1}$

```
procedure min-pair(elements)
    max ← max(elements)
    min ← min(elements)
    size ← size(elements)
    copy ← make-vector(size, nil)
    for element ∈ elements do
        index ← ⌊((elt−min)×(size−1)) / (max−min)⌋
        if copy_index = nil then
            copy_index = element
        elsereturn element, copy_index
        end if
    end for
end procedure
```