

CMPS 102 Solutions to Homework 5

Solutions by Cormen and us

November 3, 2005

Problem 1. 9.3-6 p.192

The k th quantiles of an n -element set are the $k - 1$ order statistics that divide the sorted set into k equal-sized sets (to within 1).

The k quantiles of an n -element sorted array A are:

$$A[\lfloor 1 \cdot n/k \rfloor], A[\lfloor 2 \cdot n/k \rfloor], \dots, A[\lfloor (k-1) \cdot n/k \rfloor].$$

We have as inputs an unsorted array A of distinct keys, an integer k , and an empty array Q of length $k - 1$ and we want to find the k th quantiles of A .

QUANTILES(A, k, Q)

1. **if** $k = 1$ **then return**
2. **else**
3. $n \leftarrow \text{length}[A]$
4. $i \leftarrow \lfloor \frac{k}{2} \rfloor$
5. $x \leftarrow \text{SELECT}(A, \lfloor i \cdot n/k \rfloor)$
6. PARTITION(A, x)
7. Add to list Q : QUANTILES($A[1..A[\lfloor i \cdot n/k \rfloor]], \lfloor k/2 \rfloor, Q$)
8. Add to list Q : QUANTILES($A[\lfloor i \cdot n/k \rfloor + 1..A[n]], \lceil k/2 \rceil, Q$)
9. **return** x

The output Q of the recursive algorithm QUANTILES contains the k th quantiles of A . The algorithm first finds the key that turns out to be the lower median of Q , and then partitions the input array about this key. So we have two smaller arrays that each contain at most $(k-1)/2$ of the $k-1$ order statistics of the original array A . At each level of the recursion the number of order statistics in the array is at most half the number of the previous array.

Consider a recursion tree for this algorithm. At the top level we need to find $k-1$ order statistics, and it costs $O(n)$ to find one. The root has two children, one contains at most $\lfloor (k-1)/2 \rfloor$ order statistics, and the other $\lceil (k-1)/2 \rceil$ order statistics. The sum of the costs for these two nodes is $O(n)$.

At depth i we find 2^i order statistics. The sum of the costs of all nodes at depth i is $O(n)$, for $0 \leq i \leq \log_2(k-1)$, because the total number of elements at any depth is n . The depth of the tree is $d = \log_2(k-1)$. Hence, the worst-case running time of QAUNTILES is $\Theta(n \lg k)$.

Problem 2. 9.3-9 p.193

By Cormen:

In order to find the optimal placement for Professor Olay's pipeline, we need only find the median(s) of the y -coordinates of his oil wells, as the following proof explains.

Claim

The optimal y -coordinate for Professor Olay's east-west oil pipeline is as follows:

1. If n is even, then on either the oil well whose y -coordinate is the lower median or the one whose y -coordinate is the upper median, or anywhere between them.
2. If n is odd, then the oil well whose y -coordinate is the median.

Proof

We examine various cases. In each case, we will start out with the pipeline at a particular y -coordinate and see what happens when we move it. We'll denote by s the sum of the north-south spurs with the pipeline at the starting location and s' will denote the sum after moving the pipeline.

We start with the case in which n is even. Let us start with the pipeline somewhere on or between the two oil wells whose y -coordinates are the lower and upper medians. If we move the pipeline by a vertical distance d without crossing either of the median wells, then $n/2$ of the wells become d farther from the pipeline and $n/2$ become d closer, and so $s' = s + dn/2 - dn/2 = s$; thus, all locations on or between the two medians are equally good.

Now suppose that the pipeline goes through the oil well whose y -coordinate is the upper median. What happens when we increase the y -coordinate of the pipeline by $d > 0$ units, so that it moves above the oil well that achieves the upper median? All oil wells whose y -coordinates are at or below the upper median become d units farther from the pipeline, and there are at least $n/2 + 1$ such oil wells (the upper median, and every well at or below the lower median). There are at most $n/2 - 1$ oil wells whose y -coordinates are above the upper median, and each of these oil wells becomes at most d units closer to the pipeline when it moves up. Thus, we have a lower bound on $s' \geq s + d(n/2 + 1) - d(n/2 - 1) = s + 2d > s$.

We conclude that moving the pipeline up from the oil well at the upper median increases the total spur length. A symmetric argument shows that if we start with the pipeline going through the oil well whose y -coordinate is the lower median and move it down, then the total spur length increases.

We see, therefore, that when n is even, an optimal placement of the pipeline is anywhere on or between the two medians.

Now we consider the case when n is odd. We start with the pipeline going through the oil well whose y -coordinate is the median, and we consider what happens when we move it up by $d > 0$ units. All oil wells at or below the median become d units farther from the pipeline, and there are at least $(n + 1)/2$ such wells (the one at the median and the $(n - 1)/2$ at or below the median). There are at most $(n - 1)/2$ oil wells above the median, and each of these becomes at most d units closer to the pipeline. We get a lower bound on s' of $s' \geq s + d(n + 1)/2 - d(n - 1)/2 = s + d > s$, and we conclude that moving the pipeline up from the oil well at the median increases the total spur length. A symmetric argument shows that moving the pipeline down from the median also increases the total spur length, and so the optimal placement of the pipeline is on the median.

Since we know we are looking for the median, we can use the linear-time median finding algorithm.

Problem 3. 11.4-1 p.244

Set of keys: $\{10, 22, 31, 4, 15, 28, 17, 88, 59\}$.

$m = 11$

$h'(k) = k \bmod m$.

$i = \{0, 1, 2, \dots, m - 1\}$

Linear probing:

$h(k, i) = (h'(k) + i) \bmod m$

$$\begin{array}{r|l|l|l}
 h(10, 0) & = 10 & h(28, 0) & = 6 & h(59, 0) & = 4 \\
 h(22, 0) & = 0 & h(17, 0) & = 6 & h(59, 1) & = 5 \\
 h(31, 0) & = 9 & h(17, 1) & = 7 & h(59, 2) & = 6 \\
 h(4, 0) & = 4 & h(88, 0) & = 0 & h(59, 3) & = 7 \\
 h(15, 0) & = 4 & h(88, 1) & = 1 & h(59, 4) & = 8 \\
 h(15, 1) & = 5 & & & &
 \end{array}$$

The hash table resulting from linear probing is

$$H = \{22, 88, \text{nil}, \text{nil}, 4, 15, 28, 17, 59, 31, 10\}.$$

Quadratic probing:

$$c_1 = 1, c_2 = 3$$

$$h(k, i) = (h'(k) + c_1i + c_2i^2) \bmod m$$

$h(10, 0) = 10$	$h(17, 0) = 6$	$h(88, 3) = 8$
$h(22, 0) = 0$	$h(17, 1) = 10$	$h(88, 4) = 8$
$h(31, 0) = 9$	$h(17, 2) = 9$	$h(88, 5) = 3$
$h(4, 0) = 4$	$h(17, 3) = 3$	$h(88, 6) = 4$
$h(15, 0) = 4$	$h(88, 0) = 0$	$h(88, 7) = 0$
$h(15, 1) = 8$	$h(88, 1) = 4$	$h(88, 8) = 2$
$h(28, 0) = 6$	$h(88, 2) = 3$	$h(59, 0) = 4$
		$h(59, 1) = 8$
		$h(59, 2) = 7$

The hash table resulting from quadratic probing is

$$H = \{22, \text{nil}, 88, 17, 4, \text{nil}, 28, 59, 15, 31, 10\}.$$

Double hashing:

$$h(k, i) = (h_1(k) + ih_2(k)) \bmod m$$

where $h_2 = 1 + k \bmod (m - 1)$.

$h(10, 0) = 10$	$h(15, 2) = 5$	$h(88, 2) = 7$
$h(22, 0) = 0$	$h(28, 0) = 6$	$h(59, 0) = 4$
$h(31, 0) = 9$	$h(17, 0) = 6$	$h(59, 1) = 3$
$h(4, 0) = 4$	$h(17, 1) = 3$	
$h(15, 0) = 4$	$h(88, 0) = 0$	
$h(15, 1) = 10$	$h(88, 1) = 9$	

The hash table resulting from double hashing is

$$H = \{22, \text{nil}, 59, 17, 4, 15, 28, 88, \text{nil}, 31, 10\}.$$

Problem 4. 12.3-4 p.264

Suppose that another data structure contains a pointer to a node y in a binary search tree, and suppose that y 's predecessor z is deleted from the tree by the procedure TREE-DELETE.

What problem can arise?

A binary tree is represented as a linked list where each node is an object with a key and pointers to its left child, right child and parent.

If z has two children and the successor of z is y , then y is the node spliced out. When spliced out y 's key and satellite data are moved to z . Node y is

returned in line 17 so that it may be recycled. If another data structure has a pointer to this node and it is recycled, this is a problem.

How can TREE-DELETE be rewritten to solve this problem?

When y 's key and satellite data are moved to z any pointers to y need to be updated to point to the new y node.

How will we know which pointers to update? Instead of updating the pointers, we could maintain an intermediate array S . The satellite data of a node y could include an index j to S , and $S[j]$ contains the key of y . The other data structure has a pointer to $S[j]$ which remains unchanged while the nodes of the binary search tree are moved and deleted.

Problem 5. 3.3-4 p.57

Let ϕ be the golden ratio and $\hat{\phi}$ be its conjugate.

$$\phi = \frac{1+\sqrt{5}}{2}, \quad \hat{\phi} = \frac{1-\sqrt{5}}{2}.$$

The Fibonacci numbers are defined by the following recurrence:

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_i &= F_{i-1} + F_{i-2} \quad \text{for } i \geq 2. \end{aligned}$$

Claim: The i^{th} Fibonacci number satisfies

$$F_i = \frac{\phi^i - \hat{\phi}^i}{\sqrt{5}}$$

Proof by induction on i .

Base case: Let $i = 2$.

$F_2 = F_1 + F_0 = 1 + 0 = 1$ and

$$\begin{aligned} \phi^2 &= \frac{3+\sqrt{5}}{2} \\ \hat{\phi}^2 &= \frac{3-\sqrt{5}}{2} \\ \frac{\phi^2 - \hat{\phi}^2}{\sqrt{5}} &= \frac{\frac{3+\sqrt{5}}{2} - \frac{3-\sqrt{5}}{2}}{\sqrt{5}} = 1 \end{aligned}$$

Assume that

$$F_k = \frac{\phi^k - \hat{\phi}^k}{\sqrt{5}}$$

for $0 \leq k < i$. We will show that it is true for $k = i$.

$$\begin{array}{l}
F_i = F_{i-1} + F_{i-2} \\
= \frac{\phi^{i-1} - \hat{\phi}^{i-1}}{\sqrt{5}} + \frac{\phi^{i-2} - \hat{\phi}^{i-2}}{\sqrt{5}} \\
= \frac{(\phi^{i-1} + \phi^{i-2}) - (\hat{\phi}^{i-1} + \hat{\phi}^{i-2})}{\sqrt{5}} \\
= \frac{\phi^{i-2}(\phi+1) - \hat{\phi}^{i-2}(\hat{\phi}+1)}{\sqrt{5}} \\
= \frac{\phi^i - \hat{\phi}^i}{\sqrt{5}}
\end{array}
\left| \begin{array}{l}
\text{by definition} \\
\text{by the ind. assumption with } k = i-1 \text{ and } k = i-2 \text{ respectively.} \\
\text{If } \phi + 1 = \phi^2 \text{ and } \hat{\phi} + 1 = \hat{\phi}^2.
\end{array} \right.$$

If $\phi + 1 = \phi^2$ and $\hat{\phi} + 1 = \hat{\phi}^2$ then we are done. We computed ϕ^2 and $\hat{\phi}^2$ above, so now we compute $\phi + 1$ and $\hat{\phi} + 1$.

$$\begin{aligned}
\phi + 1 &= \frac{1 + \sqrt{5}}{2} + \frac{2}{2} \\
&= \frac{3 + \sqrt{5}}{2} \\
\hat{\phi} + 1 &= \frac{1 - \sqrt{5}}{2} + \frac{2}{2} \\
&= \frac{3 - \sqrt{5}}{2}
\end{aligned}$$

We have shown that

$$F_i = \frac{\phi^i - \hat{\phi}^i}{\sqrt{5}}$$

Extra Credit 11.1-4 p.223

We denote the huge array by T and, taking the hint from the book, we also have a stack implemented by an array S . The size of S equals the number of keys actually stored, so that S should be allocated at the dictionary's maximum size. The stack has an attribute $top[S]$, so that only entries $S[1..top[S]]$ are valid.

The idea of this scheme is that entries of T and S validate each other. If key k is actually stored in T , then $T[k]$ contains the index, say j , of a valid entry in S , and $S[j]$ contains the value k . Let us call this situation, in which $1 \leq T[k] \leq top[S]$, $S[T[k]] = k$, and $T[S[j]] = j$, a validating cycle.

Assuming that we also need to store pointers to objects in our direct-address table, we can store them in an array that is parallel to either T or S . Since S is smaller than T , we'll use an array S' , allocated to be the same size as S , for these pointers. Thus, if the dictionary contains an object x with key k , then there is a validating cycle and $S'[T[k]]$ points to x .

The operations on the dictionary work as follows:

1. Initialization: Simply set $top[S] = 0$, so that there are no valid entries in the stack.

2. SEARCH: Given key k , we check whether we have a validating cycle, i.e., whether $1 \leq T[k] \leq \text{top}[S]$ and $S[T[k]] = k$. If so, we return $S'[T[k]]$, and otherwise we return NIL.
3. INSERT: To insert object x with key k , assuming that this object is not already in the dictionary, we increment $\text{top}[S]$, set $S[\text{top}[S]] \leftarrow k$, set $S'[\text{top}[S]] \leftarrow x$, and set $T[k] \leftarrow \text{top}[S]$.
4. DELETE: To delete object x with key k , assuming this object is in the dictionary, we need to break the validating cycle. The trick is to also ensure that we don't leave a "hole" in the stack, and we solve this problem by moving the top entry of the stack into the position that we are vacating - and then fixing up that entry's validating cycle. That is, we execute the following sequence of assignments:

$$\begin{aligned}
 S[T[k]] &\leftarrow S[\text{top}[S]] \\
 S'[T[k]] &\leftarrow S'[\text{top}[S]] \\
 T[S[T[k]]] &\leftarrow T[k] \\
 T[k] &\leftarrow 0 \\
 \text{top}[S] &\leftarrow \text{top}[S] - 1
 \end{aligned}$$

Each of these operations - initialization, SEARCH, INSERT, and DELETE - takes $O(1)$ time.

Notice that we may also output the set of objects in the dictionary in time $O(\text{size of set})$. We do this with the following iteration:

1. **for** $i \leftarrow \text{top}[S]$ **downto** 1
2. key $k = T[S[i]]$ and object $x = S'[i]$