Introduction to Algorithms

Exercise Class

Homework 4

Homework 5

Class Exercise 2

2021-04-09

Homework 4

Problem

Question

Consider an n-node complete binary tree T, where $n = 2^d - 1$ for some d. Each node v of T is labeled with a real number x_v. You may assume that the real numbers labeling the nodes are all distinct. A node v of T is a local minimum if the label x_v is less than the label x_w for all nodes w that are joined to v by an edge.

You are given such a complete binary tree T, but the labeling is only specified in the following implicit way: for each node v, you can determine the value x_v by probing the node v. Show how to find a local minimum of T using only $O(\log n)$ probes to the nodes of T.

Problem

Solution

For simplicity, we will say *u is smaller than v*, or $u < v$, if $x_u < x_v$.

The algorithm is the following. We begin at the root r of the tree, and see if r is smaller than its two children. If so, the root is a local minimum. Otherwise, we move to any smaller child and iterate.

Problem

The algorithm terminates when either (1) we reach a node v that is smaller than both its children, or (2) we reach a leaf w. In the former case, we return v; in the latter case, we return w.

The algorithm performs $O(d) = O(\log n)$ probes of the tree. If the root r is returned, then it is a local minimum as explained above. If we terminate in case (1), v is a local minimum because v is smaller than its parent (since it was chosen in the previous iteration)and its two children (since we terminated).If we terminate in case (2), w is a local minimum because w is smaller than its parent(again since it was chosen in the previous iteration).

6.4-3

Question

What is the running time of HEAPSORT on an array A of length n that is already sorted in increasing order? What about decreasing order?

Solution

Since an array that is sorted in increasing order is not a max-heap, BUILD-MAX-HEAP will break that ordering. The BUILDMAX-HEAP procedure will take $\Theta(n)$ to build the max-heap and the for loop of lines $2-5$ will take $O(n \lg n)$, which gives a total running time of $\Theta(n) + O(n \lg n) = O(n \lg n)$.

6.4-3

An array sorted in drecreasing order is already a max-heap, but even in that case BUILD-MAX-HEAP will take $\Theta(n)$. Note that, although the input is sorted in decreasing order, the intent of the algorithm is to sort in increasing order. In each iteration of the for loop of lines 2-5, it will exchange $A[1]$ with $A[i]$ and will call MAX-HEAPIFY on $A[1]$. Since $A[1]$ is not anymore the largest element of A, each call to MAX-HEAPIFY may cover the entire height of the heap and thus will take $O(\lg n)$. Thus, the for loop of lines 2-5 will run in $O(n \lg n)$ and the algorithm will take $\Theta(n) + O(n \lg n) = O(n \lg n)$.

6.4-4

Question

Show that the worst-case running time of HEAPSORT is $\Omega(n \lg n)$.

Solution

The worst-case is when every call to MAX-HEAPIFY covers the entire height of the heap. In that case, HEAPSORT will take

$$\sum_{i=1}^{n-1} \lfloor \lg i \rfloor \leq \sum_{i=1}^{n-1} \lg i = \lg((n-1)!) = \Theta((n-1)\lg(n-1)) = \Omega(n \lg n)$$

Homework 5

Problem 7-5

One way to improve the RANDOMIZED-QUICKSORT procedure is to partition around a pivot that is chosen more carefully than by picking a random element from the subarray. One common approach is the **median-of-3** method: choose the pivot as the median (middle element) of a set of 3 elements randomly selected from the subarray. (See Exercise 7.4-6.) For this problem, let us assume that the elements in the input array $A[1 \ldots n]$ are distinct and that $n \geq 3$. We denote the sorted output array by $A'[1 \ldots n]$. Using the median-of- 3 method to choose the pivot element x, define $p_i = \Pr\{x = A'[i]\}$.

Problem 7-5

a. Give an exact formula for p_i as a function of n and i for $i = 2, 3, \ldots, n-1$. (Note that $p_1 = p_n = 0$.)

p_i is the probability that a randomly selected subset of size three has the $A'[i]$ as it's middle element. There are 6 possible orderings of the three elements selected. So, suppose that S' is the set of three elements selected. We will compute the probability that the second element of S' is $A'[i]$ among all possible 3-sets we can pick, since there are exactly six ordered 3-sets corresponding to each 3 -set, these probabilities will be equal. For any such S', we would need to select the first element from $\{1, \ldots, i-1\}$ and the third from $\{i+1, \ldots, n\}$. So, there are $(i-1)(n-i)$ such 3-sets. The total number of 3-sets is $C_n^3 = \frac{n(n-1)(n-2)}{6}$. So,

$$p_i = \frac{6(n-i)(i-1)}{n(n-1)(n-2)}$$

Problem 7-5

b. By what amount have we increased the likelihood of choosing the pivot as $x = A'[\lfloor (n+1)/2 \rfloor]$, the median of $A[1 \ldots n]$, compared with the ordinary implementation? Assume that $n \to \infty$, and give the limiting ratio of these probabilities.

We have

$$p_{\lfloor (n+1)/2 \rfloor} = \frac{6 \cdot \left(\left\lfloor \frac{n+1}{2} \right\rfloor - 1\right)\left(n - \left\lfloor \frac{n+1}{2} \right\rfloor\right)}{n(n-1)(n-2)} \leq \frac{6 \cdot \left(\frac{n+1}{2} - 1\right)\left(n - \frac{n+1}{2}\right)}{n(n-1)(n-2)}$$

$$= \frac{6 \cdot \left(\frac{n-1}{2}\right)\left(\frac{n-1}{2}\right)}{n(n-1)(n-2)} = \frac{3}{2} \cdot \frac{(n-1)}{n(n-2)}$$

Then, we have the ratio

$$\lim_{n\to\infty} \frac{\frac{3}{2} \cdot \frac{n-1}{n(n-2)}}{\frac{1}{n}} = \lim_{n\to\infty} \frac{3}{2} \frac{n(n-1)}{n(n-2)} = \frac{3}{2}$$

Problem 7-5

c. If we define a "good" split to mean choosing the pivot as $x = A'[i]$, where $n/3 \le i \le 2n/3$, by what amount have we increased the likelihood of getting a good split compared with the ordinary implementation? (Hint: Approximate the sum by an integral.)

To save the messiness, suppose n is a multiple of 3 .We will approximate the sum as an integral, so,

$$\begin{aligned}\sum_{i=n/3}^{2n/3} p_i &\approx \int_{n/3}^{2n/3} \frac{6(-x^2+nx+x-n)}{n(n-1)(n-2)}dx \\ &= \frac{6(-7n^3/81+3n^3/18+3n^2/18-n^2/3)}{n(n-1)(n-2)} \\ &= \frac{13n^3/27-n^2}{n(n-1)(n-2)}\end{aligned}$$

$$\therefore \lim_{n\to\infty} \sum_{i=n/3}^{2n/3} p_i = \frac{13}{27}$$

which is greater than $\frac{1}{3}$.

Problem 7-5

d. Argue that in the $\Omega(n \lg n)$ running time of quicksort, the median-of-3 method affects only the constant factor.

The only difference is on the choice of the pivot. However, even if the middle element is always chosen as the pivot (which is the best case), the height of the recursion tree will be $\Theta(\lg n)$. Since each recursion level takes $\Theta(n)$, the running time is still $\Omega(n \lg n)$.

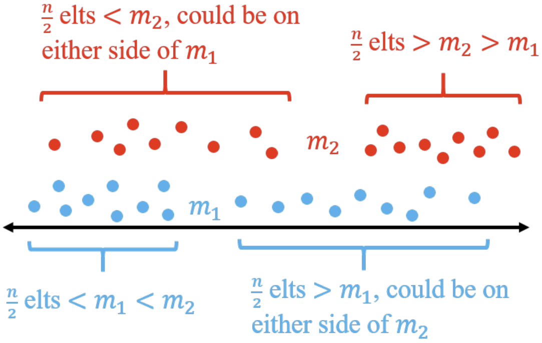
Class Exercise 2

Question

There are two arrays A and B, each with n numeric values - so there are $2n$ values total and no two values are the same. You'd like to find the *median* of this set of $2n$ values, the n^{th} smallest value. However, you can only access these values through queries to each array. E.g., you can specify a value k to array A and it will returns you the k^{th} smallest value in A. Give an algorithm that find the median value using at most $O(\log n)$ queries.

Solution

Suppose we first query each array for its median - that is, we query for element $n/2$. Let m_1 be the median from A and let m_2 be the median from B. Suppose, without loss of generality, that $m_1 < m_2$.


$\frac{n}{2}$ elts $< m_2$, could be on either side of m_1
$\frac{n}{2}$ elts $> m_2 > m_1$
m_2
m_1
$\frac{n}{2}$ elts $< m_1 < m_2$
$\frac{n}{2}$ elts $> m_1$, could be on either side of m_2

2

Pseudocode

```
FIND-MEDIAN ($A, l1, r1, B, l2, r2$)
if $l1 == r1$
    return MIN ($A[l1], B[l2]$)
$m1$ = RANDOMIZED-SELECT ($A, l1, r1, \lfloor (l1 + r1 - 1)/2 \rfloor$)
$m2$ = RANDOMIZED-SELECT ($B, l2, r2, \lfloor (l2 + r2 - 1)/2 \rfloor$)
if $m1 < m2$
    return FIND-MEDIAN ($A, m1 + 1, r1, B, l2, m2$)
else
    return FIND-MEDIAN ($A, l1, m1, B, m2 + 1, r2$)
```

Question

What are the minimum and maximum numbers of elements in a binary heap of height h?

Solution

We define the height of a node in a heap to be the number of edges on the longest simple downward path from the node to a leaf, and we define the height of the heap to be the height of its root.

A heap of height h has the minimum number of elements when it has just one node at the lowest level. The levels above the lowest level form a complete binary tree of height $h-1$ and 2^h-1 nodes. Hence the minimum number of nodes possible in a heap of height h is $\mathbf{2^h}$.

A heap of height h has the maximum number of elements when its lowest level is completely filled. In this case the heap is a complete binary tree of height h and hence has $\mathbf{2^{h+1}-1}$ nodes.

Question

You are given a sequence of n distinct numbers $A[1,\ldots,n]$. Please find the **top $\boldsymbol{k}$** order statistics **with two different algorithms**. Present your algorithms in pseudo-code and analyze your algorithms.

Solution 1

```
FIND-TOP-K ($A$, $n$, $k$)
1   let $min\_heap$ be the first $k$ elements of $A$        // $A[1]$ to $A[k]$
2   BUILD-MIN-HEAP ($min\_heap$)        // Time complexity: $O(k)$
3   for $i = k + 1$ to $n$                // Time complexity: $O((n - k) \log k)$
4       if $A[i] > min\_heap[1]$
5           exchange $min\_heap[1]$ with $A[i]$
6           MIN-HEAPIFY ($min\_heap$)
7   return $min\_heap$
```

Solution 2

```
FIND-TOP-K $(A, n, k)$
1   $p$ = RANDOMIZED-SELECT $(A, 1, n, k)$     //Time complexity: $O(n)$
2   exchange $A[p]$ with $A[n]$
3   $i$ = PARTITION $(A, 1, n)$                // Time complexity: $O(n)$
4   **return** $A[i + 1, \ldots, n]$
```

Exercise

9.3-1

Question

In the algorithm SELECT, the input elements are divided into groups of 5. Will the algorithm work in linear time if they are divided into groups of 7? Argue that SELECT does not run in linear time if groups of 3 are used.

9.3-1

Solution

For groups of 7, the algorithm still works in linear time. The number of elements greater than x (and similarly, the number less than x) is at least

$$4\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{7} \right\rceil \right\rceil - 2\right) \geq \frac{2n}{7} - 8 ,$$

and the recurrence becomes

$$T(n) \leq T(\lceil n/7 \rceil) + T(5n/7 + 8) + O(n) ,$$

which can be shown to be $O(n)$ by substitution, as for the groups of 5 case in the text.

9.3-1

For groups of 3, however, the algorithm no longer works in linear time. The number of elements greater than x, and the number of elements less than x, is at least

$$2\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{3} \right\rceil \right\rceil - 2\right) \geq \frac{n}{3} - 4 ,$$

and the recurrence becomes

$$T(n) \leq T(\lceil n/3 \rceil) + T(2n/3 + 4) + O(n) ,$$

which does not have a linear solution.

We can prove that the worst-case time for groups of 3 is $\Omega(n \lg n)$. We do so by deriving a recurrence for a particular case that takes $\Omega(n \lg n)$ time.

9.3-1

In counting up the number of elements greater than x (and similarly, the number less than x), consider the particular case in which there are exactly $\left\lceil \frac{1}{2} \left\lceil \frac{n}{3} \right\rceil \right\rceil$ groups with medians $\geq x$ and in which the “leftover” group does contribute 2 elements greater than x. Then the number of elements greater than x is exactly $2\left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{3} \right\rceil \right\rceil - 1\right) + 1$ (the -1 discounts x's group, as usual, and the $+1$ is contributed by x's group) $= 2\lceil n/6 \rceil - 1$, and the recursive step for elements $\leq x$ has $n - (2\lceil n/6 \rceil - 1) \geq n - (2(n/6 + 1) - 1) = 2n/3 - 1$ elements. Observe also that the $O(n)$ term in the recurrence is really $\Theta(n)$, since the partitioning in step 4 takes $\Theta(n)$ (not just $O(n)$) time. Thus, we get the recurrence

$$T(n) \geq T(\lceil n/3 \rceil) + T(2n/3 - 1) + \Theta(n) \geq T(n/3) + T(2n/3 - 1) + \Theta(n) ,$$

from which you can show that $T(n) \geq cn \lg n$ by substitution. You can also see that $T(n)$ is nonlinear by noticing that each level of the recursion tree sums to n.

In fact, any odd group size ≥ 5 works in linear time.

Maximum Performance of a Team

There are `n` engineers numbered from 1 to `n` and two arrays: `speed` and `efficiency`, where `speed[i]` and `efficiency[i]` represent the speed and efficiency for the i-th engineer respectively. *Return the maximum **performance** of a team composed of at most `k` engineers, since the answer can be a huge number, return this modulo 10^9 + 7.*

The **performance** of a team is the sum of their engineers' speeds multiplied by the minimum efficiency among their engineers.

Example 1:

```
Input: n = 6, speed = [2,10,3,1,5,8], efficiency = [5,4,3,9,7,2], k = 2
Output: 60
Explanation:
We have the maximum performance of the team by selecting engineer 2 (with speed=10 and efficiency=4) and engineer 5 (with
speed=5 and efficiency=7). That is, performance = (10 + 5) * min(4, 7) = 60.
```

Example 2:

```
Input: n = 6, speed = [2,10,3,1,5,8], efficiency = [5,4,3,9,7,2], k = 3
Output: 68
Explanation:
This is the same example as the first but k = 3. We can select engineer 1, engineer 2 and engineer 5 to get the maximum
performance of the team. That is, performance = (2 + 10 + 5) * min(5, 4, 7) = 68.
```

Constraints:

- `1 <= n <= 10^5`
- `speed.length == n`
- `efficiency.length == n`
- `1 <= speed[i] <= 10^5`
- `1 <= efficiency[i] <= 10^8`
- `1 <= k <= n`

Maximum Performance of a Team

```
class Solution {
public:
    using LL = long long;

    struct Staff {
        int s, e;
        bool operator < (const Staff& rhs) const {
            return s > rhs.s;
        }
    };

    int maxPerformance(int n, vector<int>& speed, vector<int>& efficiency, int k) {
        vector<Staff> v;
        priority_queue<Staff> q;
        for (int i = 0; i < n; ++i) {
            v.push_back({speed[i], efficiency[i]});
        }
        sort(v.begin(), v.end(), [] (const Staff& u, const Staff& v) { return u.e > v.e; });
        LL ans = 0, sum = 0;
        for (int i = 0; i < n; ++i) {
            LL minE = v[i].e;
            LL sumS = sum + v[i].s;
            ans = max(ans, sumS * minE);
            q.push(v[i]);
            sum += v[i].s;
            if (q.size() == k) {
                sum -= q.top().s;
                q.pop();
            }
        }
        return ans % (int(1E9) + 7);
    }
};
```

Thank you for your attention!

Introduction to Algorithms

TA: Tianlu Fei

2021-04-09