Homework 2 Solutions

September 30, 2019

1 Exercise 2.23

An array $A[1...n]$ is said to have a *majority element* if more than half of its entries are the same. Given an array, the task is to design an efficient algorithm to tell whether the array has a majority element, and, if so, to find that element. The elements of the array are not necessarily from some ordered domain like the integers, and so there can be no comparisons of the form "is $A[i] > A[j]$?". (Think of the array elements as GIF files, say.) However, you *can* answer questions of the form "is $A[i] = A[j]$?" in constant time.

(a) Show how to solve this problem in $O(n \log n)$ time. (*Hint:* Split the array A into two arrays A_1 and A_2 of half the size. Does knowing the majority elements of A_1 and A_2 help you figure out the majority element of A? If so, you can use a divide-and-conquer approach.)

Solution:

MAJORITY-ELEMENT(A)

1. Split the array A (size n) into two arrays A_1 and A_2 of half the size,
2. $m_1 \leftarrow$ MAJORITY-ELEMENT(A_1)
3. $m_2 \leftarrow$ MAJORITY-ELEMENT(A_2)
4. Case 1: Neither A_1 nor A_2 contains a majority element. Then there is no majority element in A. $m \leftarrow$ NULL
5. Case 2: $m_1 = m_2$. Then m_1 is the majority element of A: $m \leftarrow m_1$
6. Case 3: Only one of A_1 and A_2 contains a majority element m.
   If $\#m$ in $A > n/2$, then m is the majority element;
   Otherwise, there is no majority element: $m \leftarrow$ NULL
7. Case 4: $m_1 \neq m_2$.
   The majority element is m_1 only if $\#m_1$ in $A > n/2$: $m \leftarrow m_1$
   The majority element is m_2 only if $\#m_2$ in $A > n/2$: $m \leftarrow m_2$
   No majority element otherwise: $m \leftarrow$ NULL
8. Return m

The running time will be $T(n) = 2T(n/2) + O(n)$. Case 1 and 2 take constant time for combining the solutions. Case 3 and case 4 takes $O(n)$ time. We finally have $T(n) = O(n \log n)$ by the Master's theorem.

(b) Can you give a linear-time algorithm? (*Hint:* Here is another divide-and-conquer approach:

- Pair up the elements of A arbitrarily, to get $n/2$ pairs
- Look at each pair: if the two elements are different, discard both of them; if they are the same, keep just one of them.

Show that after this procedure there are at most $n/2$ elements left, and that they have a majority element if A does.)

Solution:

MAJORITY-ELEMENT-LINEAR(A)

1. If A is empty, return NULL
2. If A contains only one element m, return m
3. Pair up the elements of A arbitrarily and get $\lfloor n/2 \rfloor$ pairs
4. If n is odd, include the unpaired singleton element into B
5. Discard those pairs containing different elements
6. Include into B one element of each pair with equal elements
7. $m \leftarrow$ MAJORITY-ELEMENT-LINEAR(B)
8. If $m \neq$ NULL and $\#m$ in $A \leq n/2$, A has no majority element: $m \leftarrow$ NULL
9. Return m

Running time analysis

Let $T(n)$ be the runtime on an input array of length n. Lines 1–6 and 8–9 take $O(n)$ time. As the size of B is at most $\lceil n/2 \rceil$, line 7, the recursive call, takes $T(\lceil n/2 \rceil)$. We have

$$T(n) = T(\lceil n/2 \rceil) + O(n)$$

which is $O(n)$ by Master's theorem.

Correctness: If m is the majority element of A, it will also be the majority element of B.

(But the converse proposition is not necessarily correct, so line 8 is necessary).

Proof 1 (Longer but easy to understand). We prove the statement above separately for even and odd n.

If n is even, let $n = 2k$. If m is the majority element of A, we have

$$\#m \geq k + 1$$

After we discard $d \leq k$ pairs, we have a new array B of

$$2k - 2d \quad \text{(number of elements in, or size of } B\text{)}$$

elements. As we can discard at most one m in each pair, we have in B at least

$$\#m - d$$

copies of m. As

$$\#m - d \geq k + 1 - d = k - d + 1 > \frac{2k - 2d}{2}$$

which is thus at least more than half the size of B.

If $n = 2k+1$ is odd, we always keep the one unpaired element. We still have

$$\#m \geq k + 1$$

majority element m in A. After we discard $d \leq k$ pairs, we have a new array B of

$$2k+1 - 2d \quad \text{(number of elements in, or size of } B\text{)}$$

elements. As we can discard at most one m in each pair, we have in B at least

$$\#m - d$$

copies of m. As

$$\#m - d \geq k + 1 - d > \frac{2k+1 - 2d}{2}$$

which is thus at least more than half the size of B. □

Proof 2. (Shorter but difficult to understand). If m is the majority element of A, we must have

$$\#m \geq \lfloor n/2 \rfloor + 1$$

After we discard $d \leq \lfloor n/2 \rfloor$ pairs, we have a new array B of

$$n - 2d \quad \text{(number of elements in, or size of } B\text{)}$$

elements. As we can discard at most one m in each pair, we have in B at least

$$\#m - d$$

copies of m. As

$$\#m - d \geq \lfloor n/2 \rfloor + 1 - d = \lfloor n/2 \rfloor - d + 1 > \frac{n - 2d}{2}$$

which is thus at least more than half the size of B.

□

2 Exercise 2.26

Professor F. Lake tells his class that it is asymptotically faster to square an n-bit integer than to multiply two n-bit integers. Should they believe them?

Solution:

Professor F. Lake is incorrect. Given any two n-bit integers a and b, we have

$$(a-b)^2 = a^2 + b^2 - 2ab$$

$$ab = \frac{a^2 + b^2 - (a-b)^2}{2}$$

implying ab can be computed via 3 squares operations, 2 additions, 1 subtraction, and 1 division by 2.

Proof by contradiction. Suppose the run time of computing the squares is asymptotically faster than computing the multiplication. Then we can call the algorithm 3 times to compute a^2, b^2, and $(a-b)^2$. As we can always represent the numbers in binary, addition, subtraction, and division by 2 take at most $O(n)$, which is no more than multiplication. Then this procedure will have computed multiplication of a and b using the same asymptotic runtime, which contradicts with the assumption that it should be slower asymptotically than computing squares.

□

3 Exercise 2.28

The *Hadamard matrices* $H_0, H_1, H_2, \ldots$ are defined as follows:

- H_0 is the 1×1 matrix [1]
- For $k > 0$, H_k is the $2^k \times 2^k$ matrix

$$H_k = \left[\begin{array}{c|c} H_{k-1} & H_{k-1} \\ \hline H_{k-1} & -H_{k-1} \end{array}\right]$$

Show that if v is a column vector of length $n = 2^k$, then the matrix-vector product of $H_k v$ can be calculated using $O(n \log n)$ operations. Assume that all the numbers involved are small enough that basic arithmetic operations like addition and multiplication take unit time.

Solution:

Let $n = 2^k$.

1. Base case: $H_0 v = v$.

2. If $k > 0$, separate v into two parts: v_1 being the top $n/2$ elements of v, and v_2 being the bottom $n/2$ elements of v.

3. Define the product $H_k v$ recursively for $k > 0$:

$$H_k v = \begin{bmatrix} H_{k-1}v_1 + H_{k-1}v_2 \\ \\ H_{k-1}v_1 - H_{k-1}v_2 \end{bmatrix}$$

Runtime: Let $T(n)$ be the runtime for $H_k v$. Calculating $H_{k-1}v_1$ and $H_{k-1}v_2$ takes $2T(n/2)$. Calculating addition in $H_{k-1}v_1 + H_{k-1}v_2$ and subtraction in $H_{k-1}v_1 - H_{k-1}v_2$ takes linear time $O(n)$. As a result, we have running time $T(n) = 2T(n/2) + O(n)$, which is $O(n \log n)$ by the Master's theorem.