Divide and Conquer

韦俊林 (201928016029023)

2019 年 10 月 11 日

目录

1 Problem

1.1 Problem Description

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand. (i.e.,$[0, 1, 2, 4, 5, 6, 7]$ is an ascending array, then it might be rotated and become $[4, 5, 6, 7, 0, 1, 2]$.) How to find the minimum of a rotated sorted array?
(Hint: All elements in the array are distinct.)
For example, the minimum of the rotated sorted array $[4, 5, 6, 7, 0, 1, 2]$ is 0.
Please give an algorithm with $O(logn)$ complexity, prove the correctness and anlyze the complexity.

1.2 Natural Language Description:

1) 函数输入一个数组 $ARRAY$，最小下标 LOW，最大下标 $HIGH$，记录当前最符合所求的目标值 KEY，第一次调用算法将 $ARRAY[HIGH]$ 传给 KEY。
2) 若此数组是个完全升序的数组，若此数组最小值大于 KEY，则返回 KEY，反之返回最小值。
3) 若数组不是一个完全升序的数组，则将数组按下标分解为更小的数组，递归调用算法求解。

1.3 Subproblem Reduction Relationship

见图 1, 红色线条表示被剪枝。

1.4 pseudo-code:

algorithm 1.

1.5 Prove of Correctness:

由问题描述知，输入数组由两个升序数组组成，并且所求元素一定是这两个数组中的最小值。

$N = 2$ 此时数组是一个逆序数，显然数组经过一次分解后便能求解。

$N = i$ 根据分治的思想将原数组分解为两个数组后存在两种情况（图 2）：

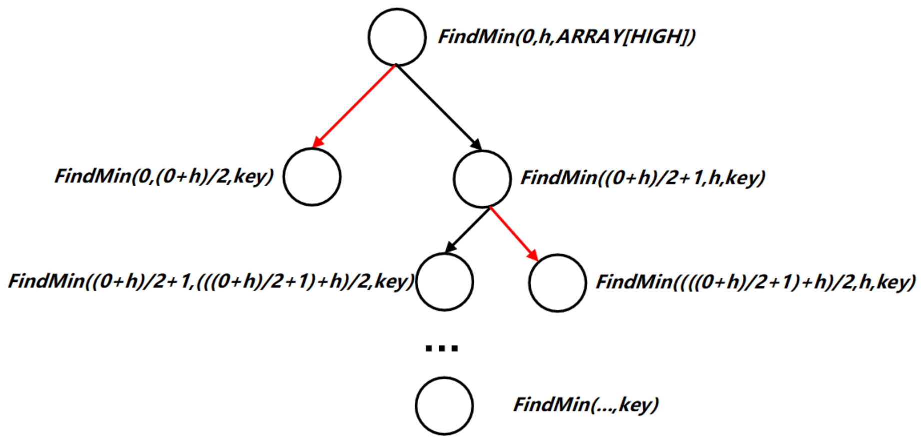

图 1: Subproblem Reduction Relationship

algorithm 1 Find the minimum of rotated sortated sorted array

INPUT: $ARRAY$, LOW, $HIGH$, KEY

OUTPUT: The minimum

```
1:  function FindMin($ARRAY, LOW, HIGH, KEY$)
2:      if $ARRAY[LOW] <= ARRAY[HIGH]$ then
3:          if $ARRAY[LOW] < KEY$ then
4:              return min($ARRAY[LOW]$,$KEY$)
5:          end if
6:      else
7:          $mid \leftarrow (LOW + HIGH)/2$
8:          $k \leftarrow$ FindMin($ARRAY, LOW, mid, KEY$)
9:          return FindMin($ARRAY, mid + 1, HIGH, k$)
10:     end if
11: end function
```

1) 完全升序的数组，所求元素可能是 $ARRAY[LOW]$。
2) 变为一个与原数组一样只是规模更小的数组，则所求元素一定在此数组里，递归调用算法求解。

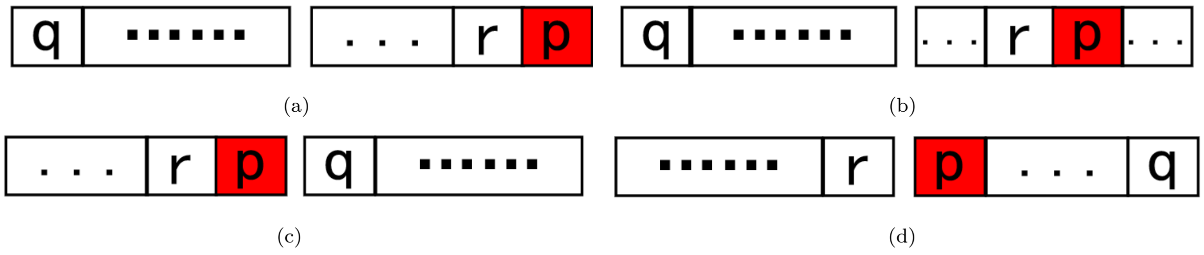

图 2: 假设 $p < q < r$，其中 p 为数组中最小值也是所求的值，r 为数组中最大值

1.6 Analyse of Complexity:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T\left(\frac{n}{2}\right) + O(1) & \text{otherwise} \end{cases}$$

运用 Unrolling the Recurrence 技术分析时间复杂度, 见图 3（红色线条表示剪枝）。二叉树高度 $logn$，经过剪枝到达叶子节点只有有限项，因此算法时间复杂度为 $O(logn)$。

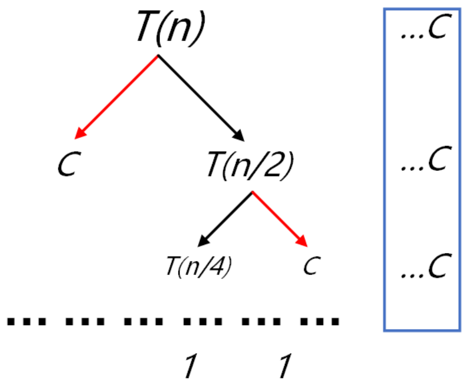

图 3: Unrolling the Recurrence

2 Problem

2.1 Problem Description:

Given a binary tree, suppose that distance between two adjacent nodes is 1, please give a solution to find the maximum distance of any two node in the binary tree.
Note: For BinaryTree, each node has three properties, a int value and two TreeNode pointers to children. You can assume that the INPUT is the root TreeNode of the tree. For example, the maximum distance of any two nodes in the below binary tree is 5.

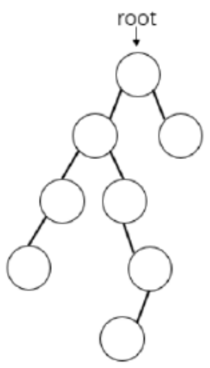

图 4: Unrolling the Recurrence

2.2 Natural Language Description:

从左子数出发，递归遍历整棵树，自底向上计算每个子树的高度以及节点距离，若此时节点距离大于已有的最大节点距离则更新最大节点距离，并且返回此子树的高度。

2.3 Subproblem Reduction Relationship

见图 5。

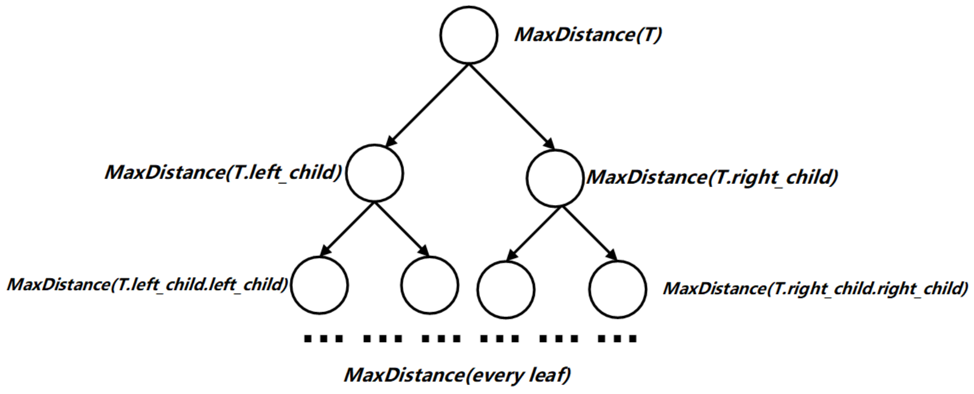
MaxDistance(T)
MaxDistance(T.left_child)
MaxDistance(T.right_child)
MaxDistance(T.left_child.left_child)
MaxDistance(T.right_child.right_child)
MaxDistance(every leaf)

图 5: Subproblem Reduction Relationship

2.4 pseudo-code:

algorithm 2.

2.5 Prove of Corrness:

一棵树最大节点距离是某个节点的最大右子树高度加最大左子数高度。遍历整棵树每个节点并计算其左右子树高度之和，取最大值。

algorithm 2 Find the maximum distance of two node

```
INPUT: The root node
OUTPUT: The maximum distance
$MAX \leftarrow 0$
function MaxDistance($T$)
    if $T$ is leaf node then
        return 0
    end if
    $LeftHigh \leftarrow$ MaxDistance($T.left$)
    $RightHigh \leftarrow$ MaxDistance($T.right$)
    $MAX \leftarrow \max(MAX, LeftHigh + RightHigh)$
    return $\max(LeftHigh, RightHigh)+1$
end function
```

2.6 Analyse of Complexity:

$$T(n) = \begin{cases} 1 & \text{if } n = 0 \\ T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + O(1) & \text{otherwise} \end{cases}$$

由图 6 分析，根据 Master Theorem 得时间复杂度为 $O(n)$。

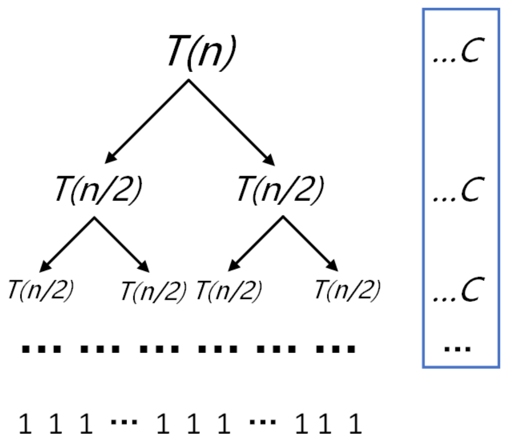

图 6: Unrolling the Recurrence

3 Problem

3.1 Problem Description:

Consider an n-node complete binary tree T, $n = 2^d - 1$ 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 *localminimum* if the label x_v is less than the label x_w for all nodes w are joined to v by an edge.

You are given such a complete 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 $O(logn)$ probes to the nodes of T.

3.2 Natural Language Description:

查找节点以及该节点的两个孩子节点中值最小的节点，如果最小值是父节点，则返回该节点，否则递归调用最小值对应的孩子节点。

3.3 Subproblem Reduction Relationship

见图 7。

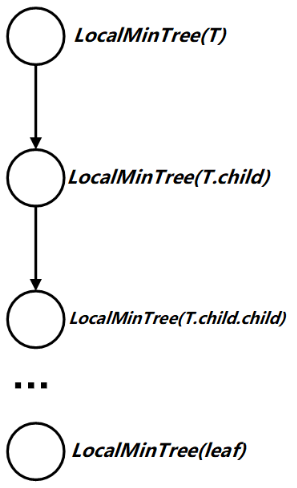

图 7: Unrolling the Recurrence

3.4 pseudo-code:

algorithm 3.

3.5 Prove of Correctness:

$N = 1$ 只有一项直接返回。

$N = i$ 当前调用此算法的节点不是根节点就是比父节点小的孩子节点，因此若此节点再同时小于左右子树便是所查找的局部最小节点，反之则必定至少有一个孩子节点比自身小。递归查询最小的孩子节点，最差情况直至叶子节点必定能找到一个局部最小节点。

3.6 Analyse of Complexity:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T\left(\frac{n}{2}\right) + O(1) & \text{otherwise} \end{cases}$$

algorithm 3 Find a local minimum of the tree

```
INPUT: The root node
OUTPUT: A local minimum of the tree
function LocalMinTree($T$)
    if $T$ is leaf node then
        return $T$
    end if
    if $T < T.left$ and $T < T.right$ then
        return $T$
    else
        LocalMinTree($min(T.left,T.right)$)
    end if
end function
```

运用 Unrolling the Recurrence 技术分析时间复杂度, 见图 8。算法只往值更小的孩子节点方向递归调用, 二叉树高度 $logn$，因此算法时间复杂度为 $O(logn)$。

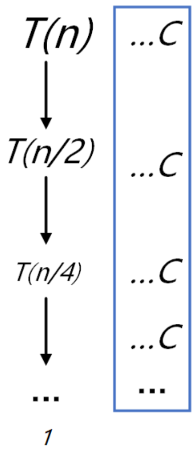

图 8: Unrolling the Recurrence