

问题求解（二）作业（第十三周）

161180162 许致明

2018 年 6 月 5 日

TC 第 12 章

12.1-2

BST: $x.left \leq x$ and $x.right > x$

Minimum Heap: $x.left \geq x$ and $x.right \geq x$

It's not possible, since for a heap, its left child and right child's keys doesn't have any greater or less relationship. So only by traversing the heap for $O(n)$ time is unable to output all keys in order.

12.1-5

证明. Building a BST using sorting algorithm based on comparing is just like sort these data. Since the lower bound of comparison-based sorting algorithm is $\Omega(n \lg n)$, the lower bound of building a BST by comparison is also $\Omega(n \lg n)$. \square

12.2-5

证明. Denote this Node as x , then $x.left.key \leq x.key$ and $x.right.key \geq x.key$.

a. Suppose $S = \text{SUCCESSOR}(x)$, then if $S.left$ exists, $S.left.key \leq S$, and S is located in x 's right subtree. So $S.left.key \geq x.key$ and $S.left.key \leq S.key$, in this case, $S.left$ is x 's successor, which contradicts our assumption. As a result, x 's successor doesn't have a left child.

b. Suppose $P = \text{PREDECESSOR}(x)$, and P has a right child $P.right$. We have $P.key \leq x.key$ and $P.right.key \geq P.key$. Besides, $P.right$ is located in

x 's left subtree. So $P.right.key \leq x.key$, in this case, $P.right$ is the predecessor of x , which contradicts our assumption. Therefore, x 's predecessor doesn't have a right child.

\square

12.2-9

证明. If x is the left child of y , then $x.key < y.key$ and $y.right.key > y.key > x.key$. For since x is the left child of y , all decedents in y 's right subtree is larger than x and y . So $y.key$ is the least key that is larger than $x.key$. Similarly, if x is the right child of y , $y.key$ is the largest key that is less than $x.key$. \square

12.3-5

TREE-INSERT(y, x, x)

```
1  if  $x \neq NIL$ 
2      if  $z.key < x.key$ 
3          TREE-INSERT( $x, x.left, z$ )
4      else
5          TREE-INSERT( $x, x.right, z$ )
6  if  $y == NIL$ 
7       $T.root = y$ 
8  elseif  $z.key < y.key$ 
9       $y.left = z$ 
10      $x.suce = y$ 
11  else
12      $y.right = z$ 
13      $z.suce = y.suce$ 
14      $y.suce = z$ 
```

```

    TREE-PRED( $x, k$ )
1  if  $k < x.key$ 
2       $y = x.left$ 
3  else
4       $y = x.right$ 
5  if  $y == NIL$ 
6      raise error
7  elseif  $y.key == k$ 
8      return  $x$ 
9  else
10     return TREE-PRED( $y, k$ )

```

TC 第 13 章

13.1-5

证明. Since this tree has a certain number of black node along every way downwards, the shortest way's length is the number of black nodes. According to the theorem, red nodes can't be adjacent, and red node's child must be black, the longest is that black and red nodes come up one after another. That is twice the number of the certain number black number has along one way downwards. So the longest way is at most twice the length of the shortest one. \square

13.1-7

If black and red nodes come up one after another, and all nodes at bottom is red, the ratio at most is 2:1. If all this tree's nodes is black, the ratio at least is 0.

13.2-2

证明. There are n nodes, then there are $n-1$ sides, every side can be rotated left or right, so there are $n-1$ possible ways of rotation. \square

13.3-1

If that node is black, then the fifth property of a RB Tree is violated, making it much more difficult to adjust.

13.3-5

证明. If an insertion doesn't violate any property of a RB Tree, it's obvious that this node is red. Then suppose the worst case, all of the $n-1$ nodes are adjusted and made black. Wherever the last node is inserted, since it's first made red, it won't violate any property. So the last node must be red. Therefore, there is at least one red node. \square

13.4-1

证明. If x is the root, then it will be made black. If x is red and is not root, the loop won't take effect, and the root remains black. If x is black and is not root, then the program will enter the loop. $x.p$ will be made red (see line 6). After case 1, it will enter case 2, 3, 4. In case 2, $x.p$ is the new x node and made red. Loop terminates. At last x is made black. If entering case 3, 4, (see line 18) $x.p$ is made black, if $x.p$ is root, it remains black. If it directly enters case 2, then $x.p$ is replaced by x . If the new x is red, loop terminates, at last it is made black. If it directly enters case 3, 4, if $x.p$ is root, it will made black in line 18, then in case terminates. After all, the root is still black. \square

13.4-7

Not the same. An example is shown below.

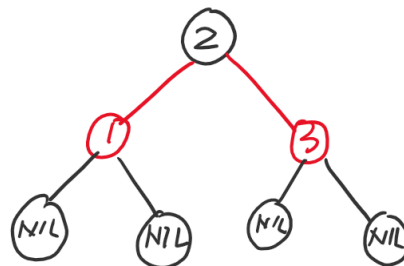


图 1: 插入顺序为 3, 2, 1

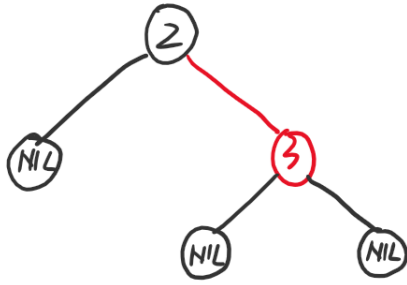


图 2: 删去节点 1

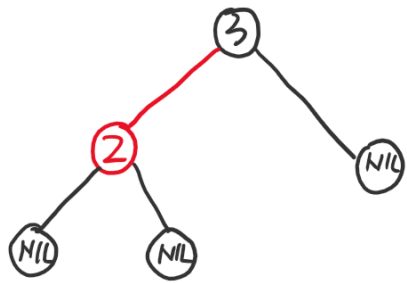


图 3: 插入顺序为 1, 3, 2 时, 删去 1

Obviously, Fig 3 is different from Fig 2.