Chapter 14 Solusion

frc6.com
i@frc6.com
https://github.com/frc123/CLRS-code-solution

10/20/2021

14.1

14.1-1

recursion	$x.key$	r	i
1	26	13	10
2	17	8	10
3	21	3	2
4	19	1	2
5	20	1	1

The result is 20.

14.1-2

iteration	$y.key$	r
1	35	1
2	38	1
3	30	3
4	41	3
5	26	16

The result is 16.

14.1-3

```
template <class Key, class T>
typename OrderStatisticsTree<Key, T>::Node* OrderStatisticsTree<Key, T>::Select
    (Node* subtree_root_node, size_t rank)
{
    size_t root_rank;
    root_rank = subtree_root_node->left->size + 1;
    while (rank != root_rank)
```

```
    {
        if (rank < root_rank)
        {
            subtree_root_node = subtree_root_node->left;
        }
        else
        {
            subtree_root_node = subtree_root_node->right;
            rank -= root_rank;
        }
        root_rank = subtree_root_node->left->size + 1;
    }
    return subtree_root_node;
}
```

14.1-4

```
template <class Key, class T>
size_t OrderStatisticsTree<Key, T>::Rank(Node* node)
{
    if (node == root_)
        return node->left->size + 1;
    else if (node == node->parent->left)
        return Rank(node->parent) - node->right->size - 1;
    else
        return Rank(node->parent) + node->left->size + 1;
}
```

14.1-5

1 $r = \text{OS-Rank}(T, x)$

2 $succ = \text{OS-Select}(T.root, r + i)$

The result is *succ*.

14.1-6

For RB-Insert(T, z), set $z.rank = 1$, and change the while loop to the following code:

while $x \neq T.nil$
    $y = x$
    if $z.key < x.key$
        $x.rank = x.rank + 1$
        $x = x.left$
    else $x = x.right$

For RB-DELETE(T, z), add the following code right befire line 18 (in the else branch):

$y.rank = z.rank$

And invoke RB-DELETE-FIX-RANK(T, x) right before line 21.

RB-DELETE-FIX-RANK(T, x)

while $x \neq T.root$
    if $x == x.p.left$
        $x.p.rank = x.p.rank - 1$
    $x = x.p$

For LEFT-ROTATE(T, x), add the following code to the end of the procedure:

$y.rank = y.rank + x.rank$

For RIGHT-ROTATE(T, y), add the following code to the end of the procedure:

$y.rank = y.rank - x.rank$

14.1-7

```cpp
template <typename Key>
size_t CountInversions(std::vector<Key> array)
{
    size_t inversions, i, rank;
    OrderStatisticsTree<Key, int> tree;
    std::pair<typename OrderStatisticsTree<Key, int>::Iterator, bool> insert_result;
    inversions = 0;
    for (i = 0; i < array.size(); ++i)
    {
        insert_result = tree.Insert({array[i], 0});
        rank = tree.Rank(insert_result.first);
        inversions += (1 + i - rank);
    }
    return inversions;
}
```

14.1-8

```
size_t CountIntersections(std::vector<Chord> chords)
{
    size_t intersections, i, rank_a, rank_b, rank_diff_1, rank_diff_2;
    OSTree tree;
    std::pair<typename OSTree::Iterator, bool> insert_result_a, insert_result_b;
    intersections = 0;
    for (i = 0; i < chords.size(); ++i)
    {
        insert_result_a = tree.Insert({chords[i].endpoint_a, 0});
        insert_result_b = tree.Insert({chords[i].endpoint_b, 0});
        rank_a = tree.Rank(insert_result_a.first);
        rank_b = tree.Rank(insert_result_b.first);
        if (rank_a > rank_b) std::swap (rank_a, rank_b);
        // rank_a must smaller than rank_b
        rank_diff_1 = rank_b - rank_a - 1;
        rank_diff_2 = tree.Size() - rank_b + rank_a - 1;
        intersections += std::min(rank_diff_1, rank_diff_2);
    }
    return intersections;
}
```

14.2

14.2-1

Add *prev* and *succ* attributes to each node in the tree. Let *prev* points to predecessor of the node, and let *succ* points to successor of the node. Let $T.nil.succ$ points to the minumum element in the tree, and let $T.nil.prev$ points to the maximum element in the tree. A circular doubly linked list is formed.

In order to maintain these informations, we just need to modify RB-INSERT and RB-DELETE.

For RB-INSERT(T, z), modify line 9 - 13 to the following code:

```
if y == T.nil
    T.root = z
    z.succ = T.nil
    z.prev = T.nil
elseif z.key < y.key
    y.left = z
    z.succ = y
    z.prev = y.prev
else y.right = z
    z.prev = y
    z.succ = y.succ
z.succ.prev = z
z.prev.succ = z
```

For RB-DELETE(T, z), add the following code right before line 21:

```
z.prev.succ = z.succ
z.succ.prev = z.prev
```

14.2-2

We can maintain black-heights of nodes without affecting the asymptotic performance since a change to $x.bh$ propagates only to ancestors of x in the tree.
For RB-INSERT(T, z), add the following code right before line 17:

```
z.bh = 1
```

For RB-INSERT-FIXUP(T, z), add the following code right before line 8 (in the if branch) (case 1):

```
z.p.p.bh = z.p.p.bh + 1
```

For RB-DELETE-FIXUP(T, z), add the following code right before line 11 (in the if branch) (case 2):

```
x.p.bh = x.p.bh - 1
```

And add the following code right before line 21 (in the else branch) (case 4):

```
x.p.bh = x.p.bh - 1
x.p.p.bh = x.p.p.bh + 1
```

We cannot maintain depths of nodes without affecting the asymptotic performance since a change to $x.bh$ propagates to descendants of x in the tree.

14.2-3

After the rotation on x is performed, run the following code:

$x.p.f = x.f$
$x.f = x.\mathit{left}.f \otimes x.\mathit{right}.f \otimes x.a$

Apply to the *size* attributes in order-statistic trees, we just need to change f to size, change $\otimes$ to $+$, and attibute a of each node will be 1; the code will be:

$x.p.\mathit{size} = x.\mathit{size}$
$x.\mathit{size} = x.\mathit{left}.\mathit{size} + x.\mathit{right}.\mathit{size} + 1$

14.2-4

The following procedure takes $\Theta(m+\log(n))$ time (to understand this asymptotic performance, refer to theorem 12.1 and exercise 12.2-8):

RB-ENUMERATE(x, a, b)
if $a \le x.\mathit{key}$ and $x.\mathit{key} \le b$
    OUTPUT(x)
if $a \le x.\mathit{key}$ and $x.\mathit{left} \ne T.\mathit{nil}$
    RB-ENUMERATE$(x.\mathit{left}, a, b)$
if $x.\mathit{key} \le b$ and $x.\mathit{right} \ne T.\mathit{nil}$
    RB-ENUMERATE$(x.\mathit{right}, a, b)$

Note that we need to implement RB-ENUMERATE(x, a, b) in $\Theta(m+\log(n))$ time, so it does not meet the requirement of the question if we implement the procedure in the following ways (augment the tree in the way of exercise 14.2-1) since it takes $\Theta(m)$ time only:

RB-ENUMERATE(T, a, b)
$k = a$
OUTPUT(k)
repeat
    $k = k.\mathit{succ}$
    OUTPUT(k)
until $k == b$