Problem Set IV

Huy Quang Lai
132000359

Texas A&M University

27 September 2022

Tree Traversal

Problem 1

Prefix: $- \ * \ + \ a \ b \ c \ + \ d \ e$
Infix: $a + b * c - d + e$
Postfix: $a \ b \ + \ c \ * \ d \ e \ + \ -$

Binary Search Tree

Problem 1

Let N be the maximum number of nodes in a Binary Tree.
Let $P(n) := N = 2^{n+1} - 1$ nodes, when $n \geq 0$

Basis Step: $P(0)$

A Binary Tree of height zero only has its root.
$2^{0+1} - 1 = 2^1 - 1 = 1$
$\therefore P(0)$ holds.

Inductive Step: $P(k) \implies P(k+1)$

Assume $P(k)$ for some arbitrary $k \geq 0$: A Binary Tree of height k has a maximum of $2^{k+1} - 1$ nodes

Show $P(k+1)$: A Binary Tree of height $k+1$ has a maximum of $2^{(k+1)+1}-1$ nodes
Using the recursive definition of a Binary Tree, both sub-trees must be a height of k.
With this the total number of nodes in the Binary Tree must be

$$\begin{aligned} 1 + (2^{k+1} - 1) + (2^{k+1} - 1) &= 2 \cdot 2^{k+1} - 2 + 1 \\ &= 2^{k+2} - 1 \end{aligned}$$

$\therefore P(k) \implies P(k+1)$ holds for arbitrary $k \geq 0$.

$\therefore P(n)$ holds for all $n \geq 0$ by mathematical induction. □

Problem 2

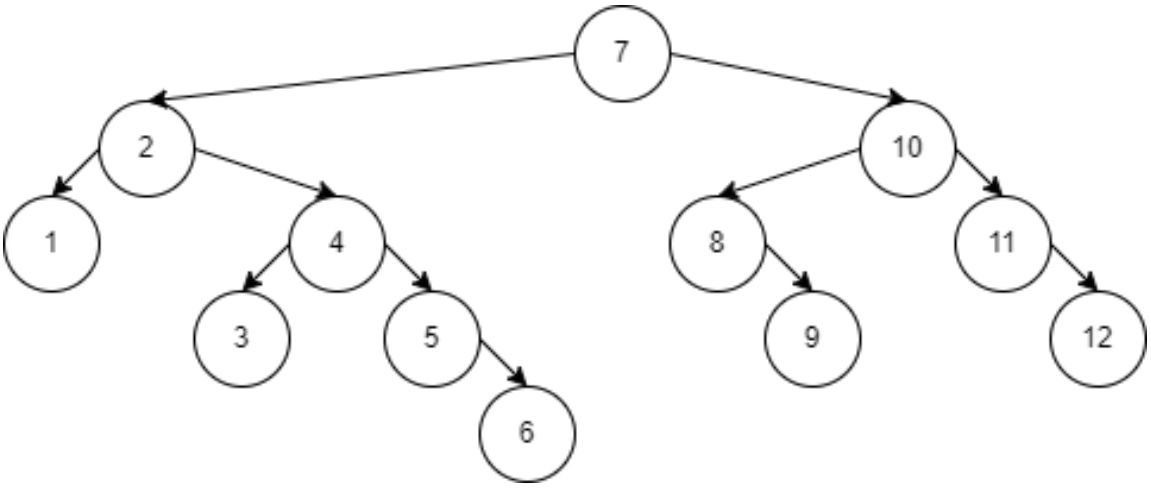

Figure 1: Binary Search Tree after Insert

The Root: 7
The Leaves: $\{1, 3, 6, 9, 12\}$
The Siblings: $\{(1, 4), (3, 5), (8, 11), (2, 10)\}$
Depths: $\{(1, 3), (2, 2), (3, 4), (4, 3), (5, 4), (6, 5), (7, 1)$
$, (8, 3), (9, 4), (10, 2), (11, 3), (12, 4)\}$
Heights: $\{(1, 1), (2, 4), (3, 1), (4, 3), (5, 2), (6, 1), (7, 5)$
$, (8, 2), (9, 1), (10, 3), (11, 2), (12, 1)\}$

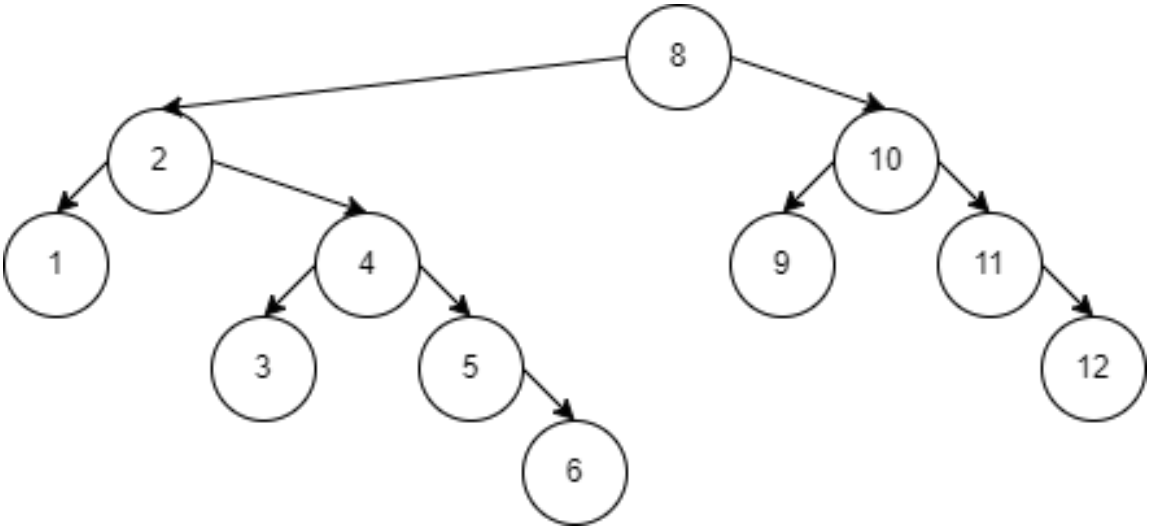

Figure 2: Binary Search Tree after Remove

Problem 3

```
size_t countNodes(const Node*& root) const {
    return root ?
        1 + countNodes(root->_left) + countNodes(root->_right) :
        0;
}
```

$O(\log_2 N)$

```
size_t countLeaves(const Node*& root) const {
    if (!root)
        return 0;
    else if (!root->_left && !root->_right)
        return 1;
    return countLeaves(root->_left) + countLeaves(root->_right);
}
```

$O(\log_2 N)$

```
size_t countFull(const Node*& root) const {
    if (!root)
        return 0;
    size_ t l = countFull(root->_left), r = countFull(root->_right)
    if (root->_left && root->_right)
        return 1 + l + r;
    return l + r;
}
```

$O(\log_2 N)$

AVL Tree

Problem 1

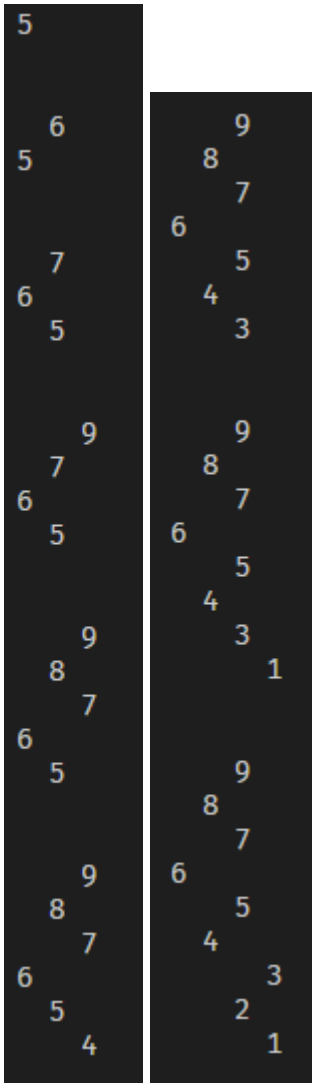

Figure 3: AVL Insert

Problem 2

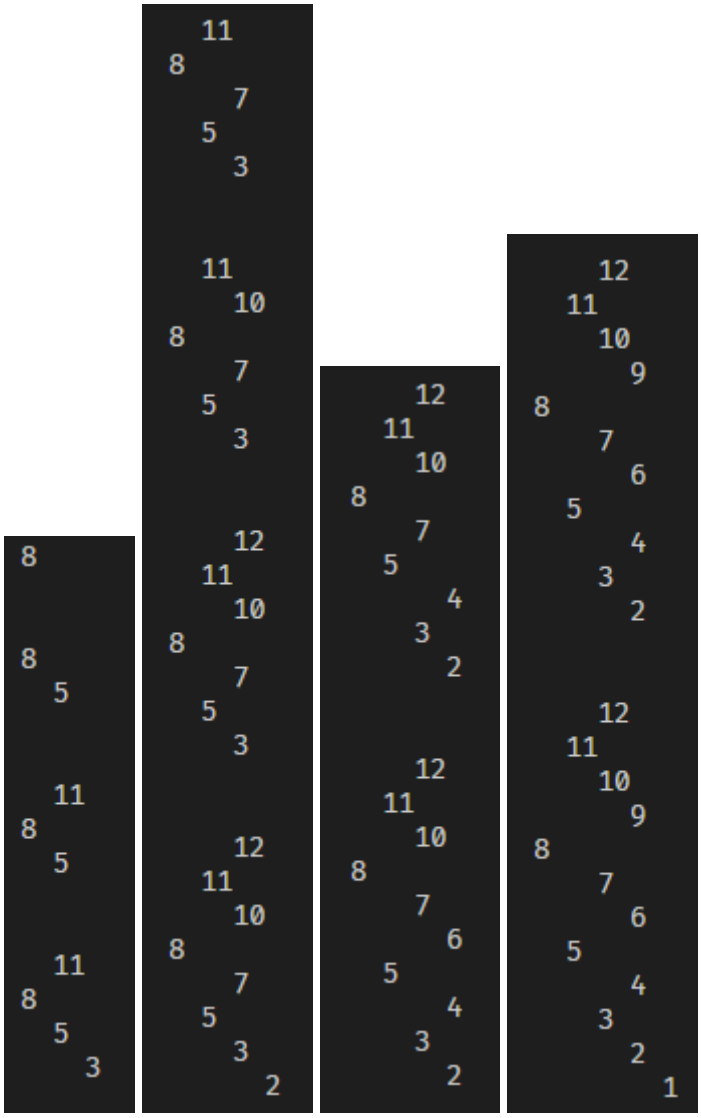

Figure 4: Caption

Problem 3

Let $N(h)$ be the minimum number of nodes in a AVL Tree of height h.
Then, $N(h) = 1 + N(h-1) + N(h-2)$, $N(0) = 1, N(1) = 2$
$N(2) = 1 + 2 + 1 = 4$
$N(3) = 1 + 4 + 2 = 7$
$N(4) = 1 + 7 + 4 = 12$
$N(5) = 1 + 12 + 7 = 20$
$N(6) = 1 + 20 + 12 = 33$
$N(7) = 1 + 33 + 20 = 54$
$N(8) = 1 + 54 + 33 = 88$
$N(9) = 1 + 88 + 54 = 143$
$N(10) = 1 + 143 + 88 = 232$
$N(11) = 1 + 232 + 143 = 376$
$N(12) = 1 + 367 + 232 = 600$
$N(13) = 1 + 600 + 367 = 968$

Red Black Tree

Problem 1

```
void remove(const Comparable& value) {
    if (this->empty())
        return;

    Node* node = this->search(value);
    if (!node)
        return;

    this->remove(node);
}

void remove(Node*& node) {
    if (node->isLeaf()) {
        delete node;
        return;
    }

    Node* replace = this->findMin(node->_right);
    swap(node->_value, replace->_value);
    this->remove(replace);
}
```

Problem 2

Prove that the height of a red-black tree is at most $2 \log N$, and that this bound cannot be substantially lowered.

Let $bh(x)$ be the black height of node x.
Let $P(n) := N = 2^{bh(x)} - 1$

Basis Step: x is leaf, $P(0)$
$bh(x) = 0 \rightarrow N(x) = 2^0 - 1 = 0$
$\therefore P(0)$ holds

Inductive Step: $P(k) \Rightarrow P(k+1)$ Assume $P(k)$ for some arbitrary $k \geq 0$: A Red Black Tree with a black height of $bh(k)$ has a minimum of $2^{bh(k)} - 1$.
If x is black, both subtrees have a black height of $bh(x) - 1$.
If x is red, both subtrees have a black height of $bh(x)$

Therefore, the number of nodes in any subtree is

$$N(x) \geq 2^{bh(x)-1} - 1 + 2^{bh(x)-1} - 1 + 1 \geq 2^{bh(x)} - 1$$

$\therefore P(k) \Rightarrow P(k+1)$ holds for arbitrary $k \geq 0$. $\therefore P(n)$ holds for all $n \geq 0$ by mathematical induction.

Let h be the height of the red black tree. Under the red black tree properties, half of the nodes on any single path must be black ignoring the root.
Therefore, $bh(x) \geq \frac{h}{2}$ and $N \geq 2^{\frac{h}{2}} - 1 \rightarrow N + 1 \geq 2^{\frac{h}{2}}$
This implies that $\log_2(n+1) = \frac{h}{2} \equiv h \leq 2\log_2(n+1)$

Problem 3

Color the root black and its children red.
Each Red Node will have black children and each black Node will have red children.
If we apply this coloring to the entire AVL Tree, we can have the AVL tree as a Red Black Tree.

Not all Red Black Trees are AVL Trees since Red Black Trees can have more of a height different between children than AVL Trees.

Sets and Maps

Problem 1

Assume that a BST data structure already exists as a class (e.g. BinarySearchTree or AVL-Tree, or RedBlackTree) and that class implements the necessary iterators. Write pseudocode for an implementation of the set class with associated iterators using a binary search tree.

```
iterator insert(const_iterator hint, const Value& value) {
    Node* node = this->_search(value);
    if (node)
        return iterator(node);
    const Node* location = hint._node;
    if (!this->_root || !location)
       this->_root = this->insert(this->_root, value);
    else
        this->insert(location, value);
    return iterator(this->search(value));
}

iterator remove(const_iterator hint) {
    if (!this->_root)
        throw std::out_of_range("Set is empty");
    if (hint == this->end())
        throw std::out_of_range("Iterator out of bounds")
    const Key& key = *hint;
    iterator it(++iter);
    this->_root = this->remove(this->_root, key);
    --this->_size;
    return it;
}
```

Problem 2

Write pseudocode for an implementation of the map class by storing a data member of type `set<Pair<KeyType,ValueType>>`.

```
typedef pair_t pair<KeyType, ValueType>
pair<iterator, bool> insert(const pair_t& _pair) {
    if (_pair.first < this->_root->_pair->first)
        return this->insert(this->_root->left, _pair);
    else
        return this->insert(this->root->right, _pair);
}
```

```
iterator insert(const_iterator hint, const pair_t& _pair) {
    if (_pair.first < hint->first)
        return this->insert(--hint, _pair);
    else
        return this->insert(++hint, _pair);
}

size_t remove(const KeyType& key) {
    if (this->_root->_pair.first == key)
        return this->remove(this->_root);
    else if (this->_root->_pair.first < key)
        return this->remove(this->_root->left);
    else
        return this->remove(this->_root->right);
}
```