第 16 课
二叉搜索树

薛浩

xuehao0618@outlook.com

阅读

- Programming Abstraction in C++ *Chapter 16*

今日话题

- 二叉搜索树

动机

在链表中删除一个节点?

链表删除节点复杂度

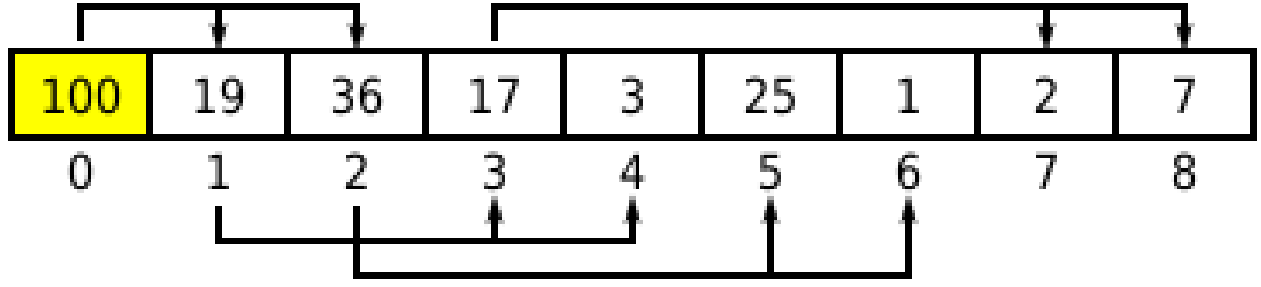
$O(n)$

作业 4 实现策略

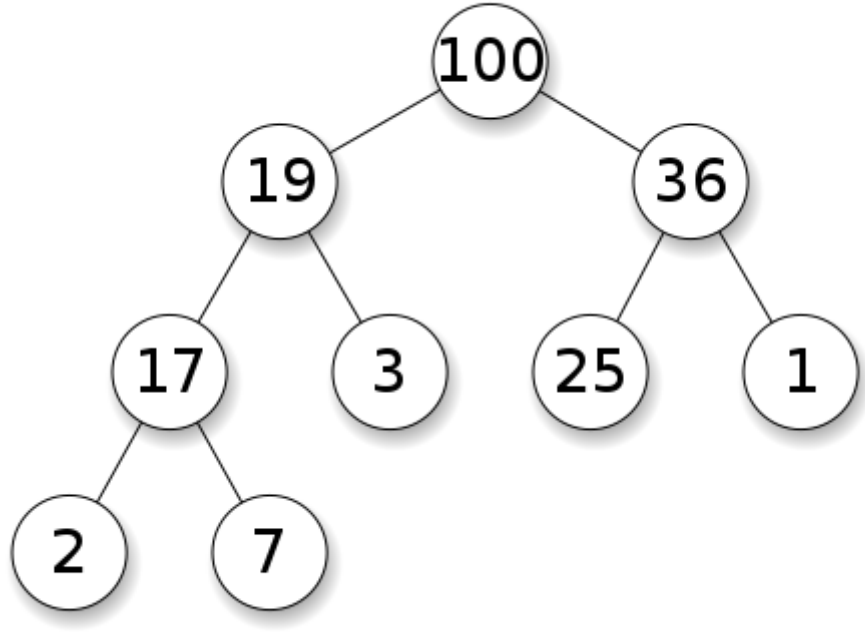
- Array
- Binary Heap

	Array	Heap
pqSort	O(N^2)	O(N log N)
topK	O(N^2)	O(N log N)

聪明的想法



树 Tree



动机

如何以树的形式组织链表节点?

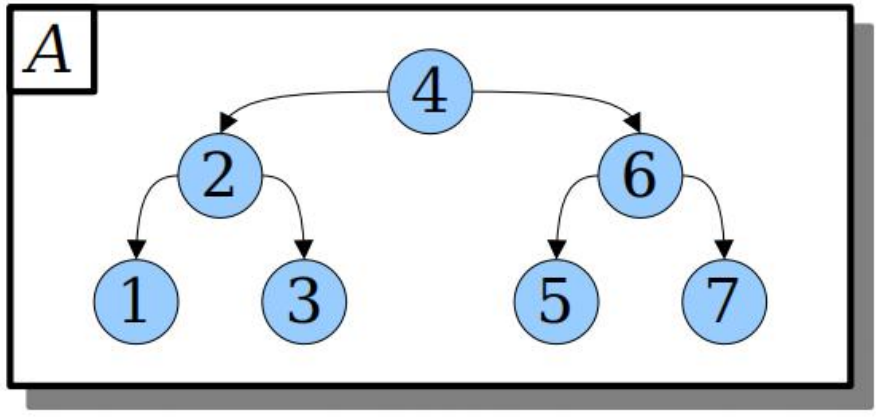
二叉搜索树

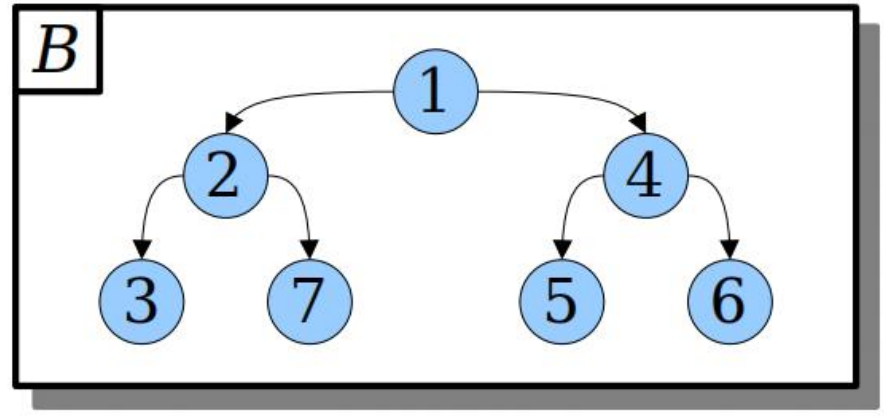
二叉搜索树

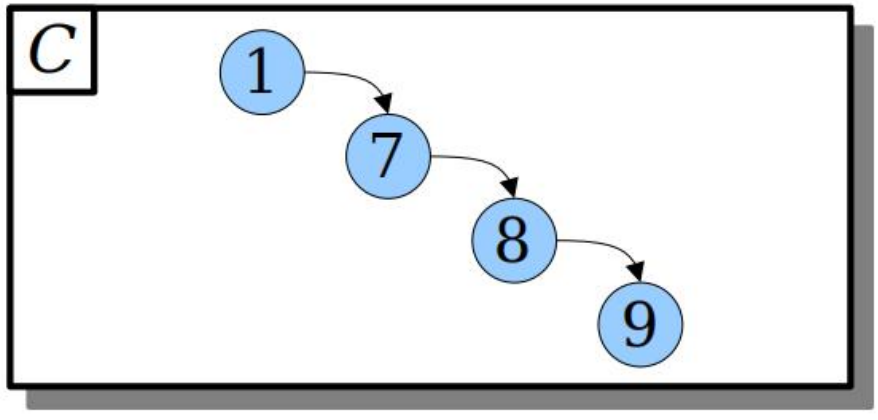
- 二叉搜索树（BST）由节点（Node）组成
- 每个节点都存储一个独一无二的（键）值
  - 节点左边的节点值，小于节点自身值
  - 节点右边的节点值，大于节点自身值
- 每个节点都有 0、1 或 2 个（叶）子节点

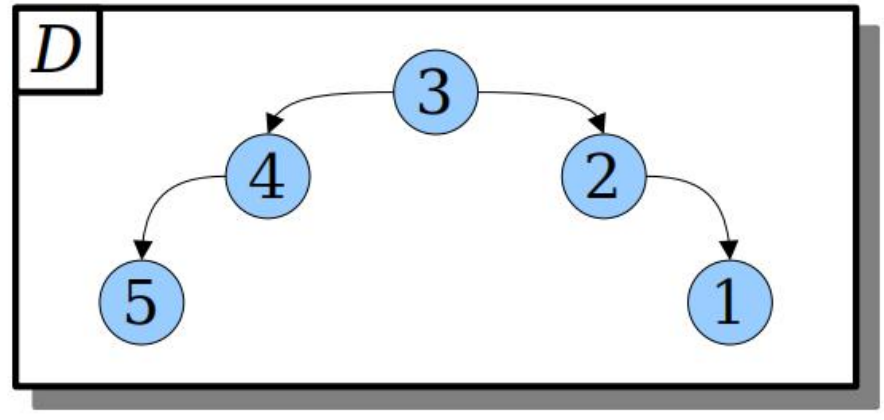
小试牛刀

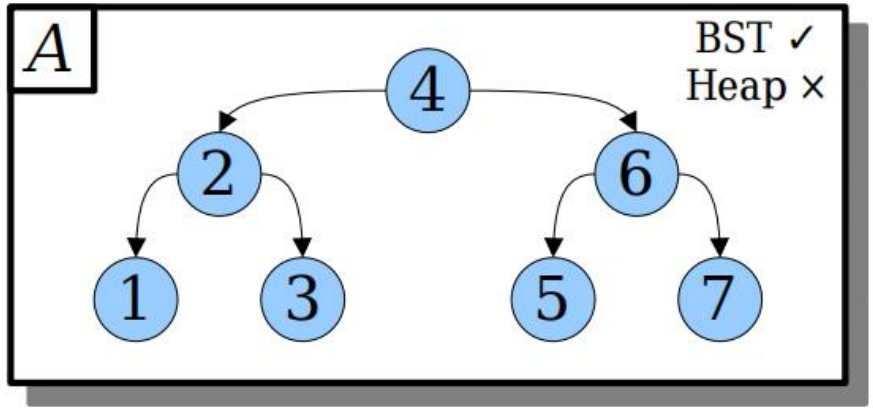
哪些是正确的树结构？

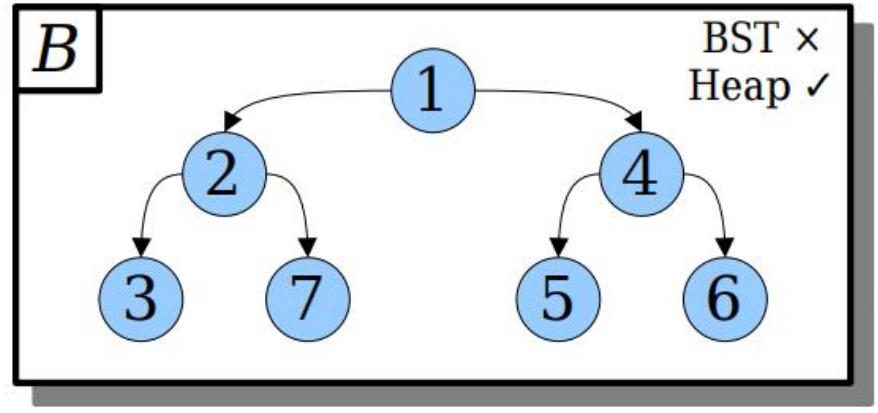

A
4
2
6
1
3
5
7

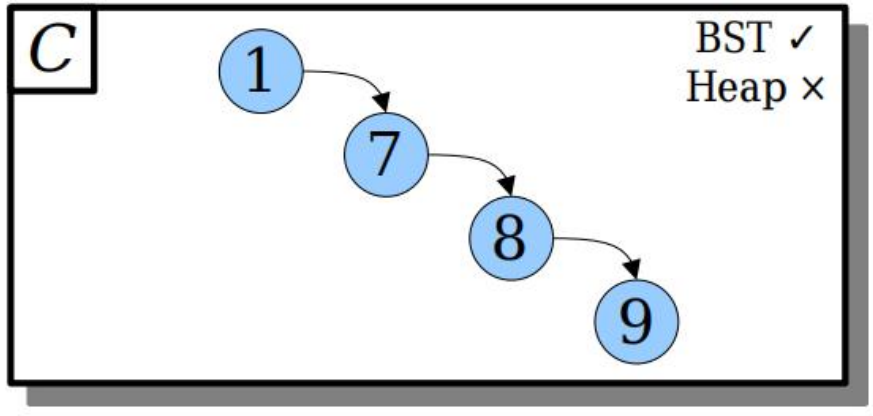

B
1
2
4
3
7
5
6

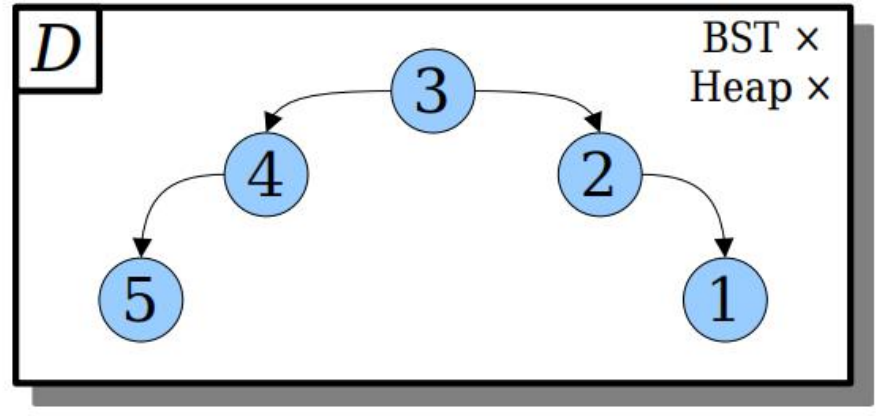

C
1
7
8
9


D
3
4
2
5
1


A
BST ✓
Heap ×
4
2
6
1
3
5
7


B
BST ×
Heap ✓
1
2
4
3
7
5
6


C
BST ✓
Heap ×
1
7
8
9


D
BST ×
Heap ×
3
4
2
5
1

BST 节点

```
struct Node {
    string key;
    Node *left;
    Node *right;
};
```

常见操作

- 查找节点
- 插入节点
- 遍历树
- 删除树

查找节点

```
Node *findNode(Node *root, const string &key) {
    // edge case: empty link
    if (root == nullptr)
        return nullptr;
    // base case: find key
    if (key == root->key)
        return root;
    // recursive case
    if (key < root->key) {
        return findNode(root->left, key);
    } else {
        return findNode(root->right, key);
    }
}
```

插入节点

```
void insertNode(Node *&root, const string &key) {
    if (root == nullptr) {
        root = new Node{key, nullptr, nullptr};
    } else {
        // key should be unique
        if (key < root->key) {
            insertNode(root->left, key);
        } else if (key > root->key) {
            insertNode(root->right, key);
        }
    }
}
```

遍历树

```
void displayTree(Node *root) {
    if (root != nullptr) {
        displayTree(root->left);
        displayTree(root->right);
    }
}
```

删除树

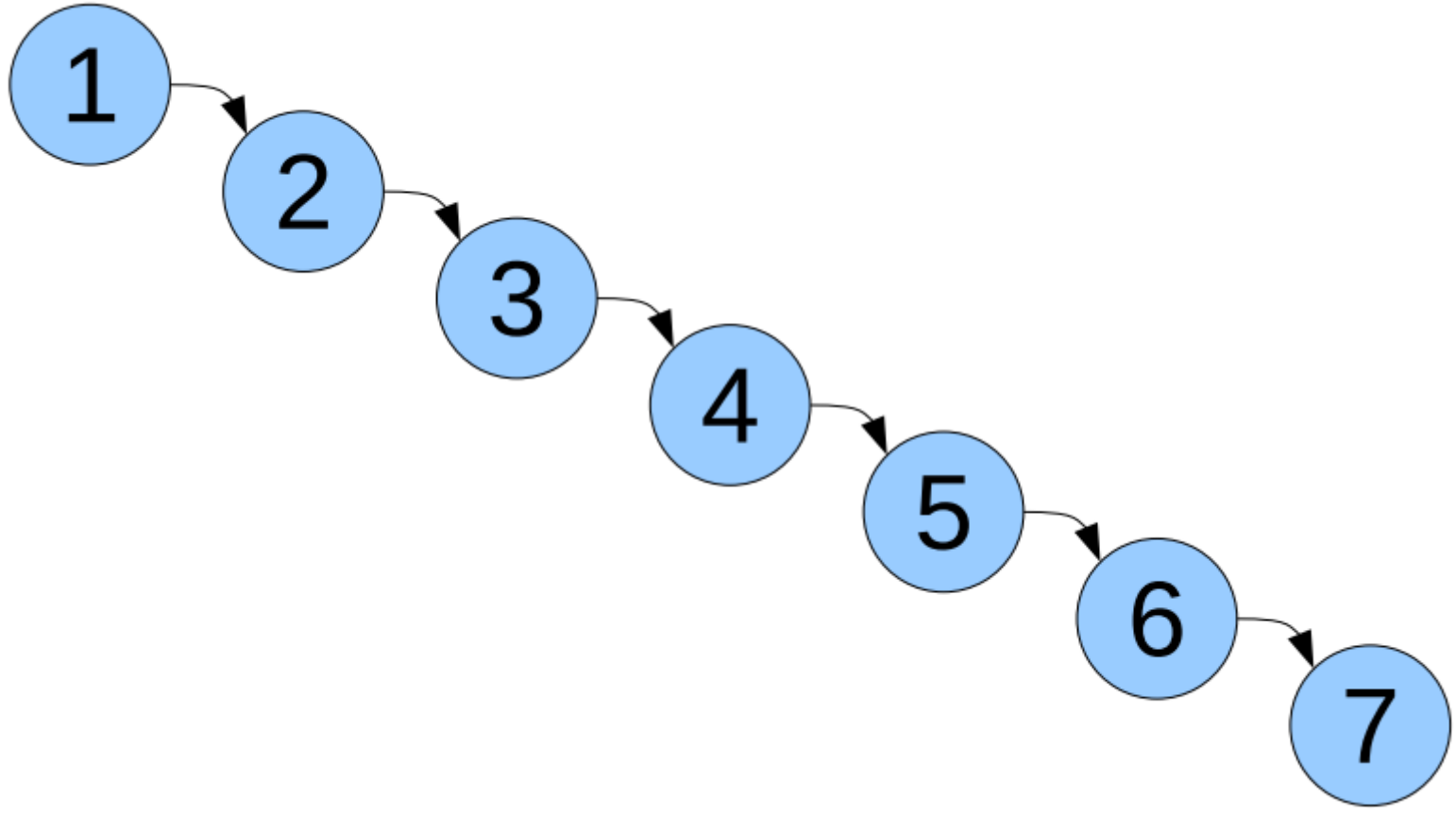
```
void deleteTree(Node *root) {
    // base case
    if (root == nullptr)
        return;
    // recursive case
    deleteTree(root->right);
    deleteTree(root->left);
    delete root;
}
```

算法复杂度

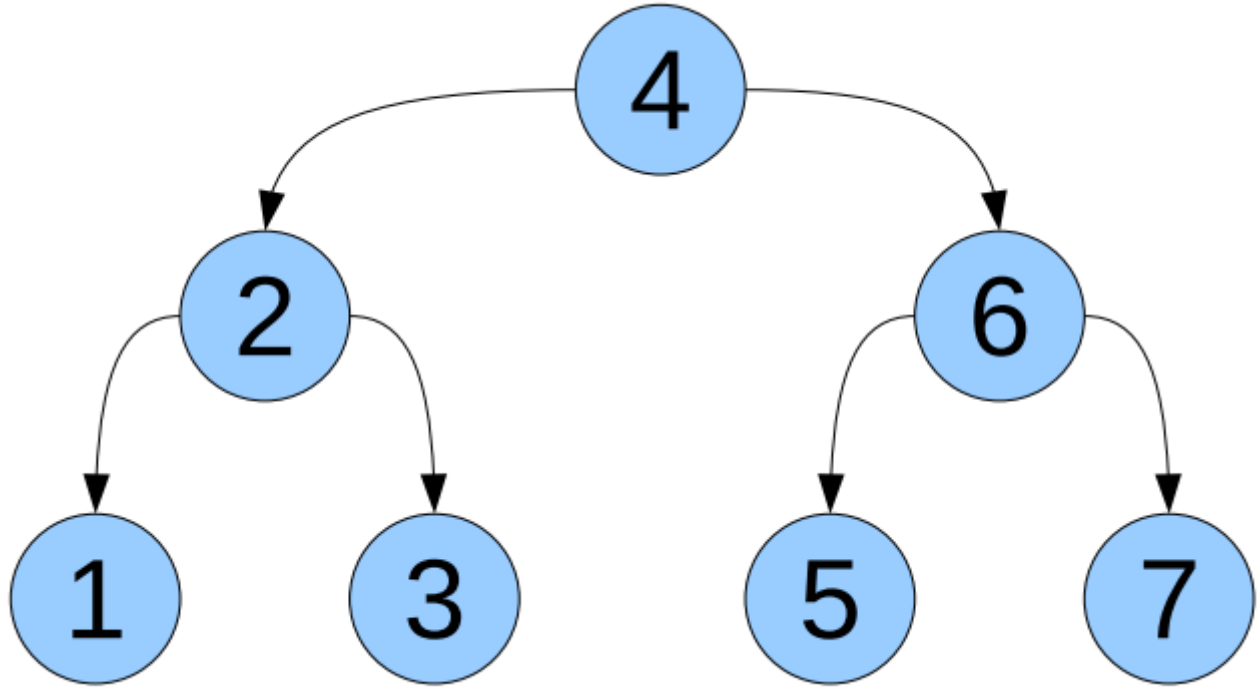
HEIGHT

树的高度是从根节点到叶子之间，最长的链接数

最大高度



最小高度



平衡二叉树

最大高度为 O(log N) 的树

- 查找 O(log N)
- 插入 O(log N)
- 删除 O(log N)

平衡策略

在插入或删除节点时，可以修正自身的形状，以重回平衡状态。

常见策略

- AVL 树（课本）
- 红黑树（CS161）
- …

小试牛刀

基于 BST 创建一个简单版 Set 类

今日话题

- ~~二叉搜索树~~

下一次课

- 哈夫曼编码

THE END