$\log n$ Search

Warm up

> **定义 3.2** 查找问题
>
> - 输入：n 个键值 $\{k_1, k_2, \ldots, k_n\}$ ，键值 KEY
> - 输出：是否有某个键值 $k_i = KEY (1 \leqslant i \leqslant n)$

查找问题的关键

> search 的关键永远不是 search，而是如何**组织数据**使其查找起来高效
>
> 当查找操作频繁，而待查找元素变化不大时，付出较高代价组织数据在长远上看是**合算**的

- BF：基于遍历线性表查找，性能为 $O(n)$ ，在实际应用中不可接受
- $O(\log n)$ ：做到每次查找将查找空间减半
- $O(1)$ ：理想情况，实际无法做到，但可通过 hash 使代价降到 $O(1+\alpha)$

Binary search over sorted sequence

最基本的二分查找，需要查找空间有序（数组，链表因为访存不是常数时间故不适用二分查找）

BINARY-SEARCH (A[first...last], key):

```
if last < first then
    return -1;
int mid := (first + last) / 2;
if key == A[mid] then
    index := mid;
else if key < A[mid] then
    index := BINARY-SEARCH(A[first...mid-1], key);
else
    index := BINARY-SEARCH(A[mid+1...last], key);
return index;
```

在此基础上可产生很多基于二分查找的衍生问题

- find peak number in unimodal array (单峰数组的峰值)
- 在有序的自然数集中寻找最小的不在集合中的自然数
- 在有序整数数组中寻找 $A[i] = i$
- 计算 $\lfloor \sqrt{N} \rfloor$

Balanced Binary Search Tree

Binary Search Tree (BST)

> **定义 9.1** 二叉搜索树：任意结点满足如下性质的二叉树
>
> - 对于结点 x， x 左子树中任意结点 y 和右子树中任意结点 z 均满足
>
> $$y.key < x.key, x.key < z.key$$

二叉树的查找代价与树的平衡度相关，平衡的二叉树查找代价为 $O(\log n)$ ，而不平衡的二叉树代价会退化到 $O(n)$ ，因此如何保持二叉树的平衡性成为了关键

Red-Black Tree (RBT)

定义红黑树之前需要一些准备性定义

> - 染色：红黑树的每个结点必然是红色或黑色
> - 黑色深度：
>   - 根的黑色深度为 0
>   - 非根结点的黑色深度为从根结点到该结点路径上黑色结点个数（不包括根）

红黑树是一颗 BST，且是 2-tree，并且满足如下定义

> **定义 9.2** 红黑树的直接定义
>
> - 根结点为黑色，所有外部结点是黑色
> - 红色结点不能连续出现，即没有任何红色结点有红色子女
> - 以任意结点为根的子树中，所有外部结点的黑色深度相等。这一统一的黑色深度定义为该子树的黑色高度，也即该结点的黑色高度

对于根节点为红色的 BST，若其满足其他红黑树的定义，则其为准红黑树（Almost Red-Black Tree, ARB）

可递归定义红黑树

> **定义 9.3** 红黑树的递归定义
>
> - 唯一一个外部结点（根节点）构成黑色高度为 0 的红黑树 RB_0
> - 对于 $h \geqslant 1$，一棵二叉树为 ARB_h ，如果其根为红色，且左右子树均为 RB_{h-1}
> - 对于 $h \geqslant 1$，一颗二叉树为 RB_h ，如果其根为黑色，且左右子树为 RB_{h-1} 或 ARB_h

RBT 的平衡性

引入红色结点是为了权衡平衡性和维护开销，红色结点允许 RBT 有一定程度的不平衡，但红色结点不能连续出现使得这种不平衡性有一定限制。

> **引理 9.1** 假设 T 为一颗 RB_h
>
> - T 有至少 $2^h - 1$ 个黑色内部结点
> - T 有至多 $4^h - 1$ 个内部结点
> - 任何黑色结点的普通高度至多是其黑色高度的两倍
>
> **引理 9.2** 假设 T 为一颗 ARB_h
>
> - T 有至少 2^h −2 个黑色内部结点
> - T 有至多 $\frac{4^h}{2} - 1$ 个内部结点
> - 任何黑色结点的普通高度至多是其黑色高度的两倍

上述引理均可结合 RBT 的递归定义使用数学归纳法对高度归纳证明

根据上述引理，可得一颗有 n 个结点的 RBT，其黑色高度至多为 $\log(n+1)$ ，即其普通高度至多为 $2\log(n+1)$ ，这保证了红黑树的查找代价

> **定理 9.1** 假设 T 为一个 n 个内部结点的 RBT，其高度不超过 $2\log(n+1)$ ，查找代价为 $O(\log n)$

RBT 的插入与删除

插入原则：不改变黑色高度，故插入结点为红色，通过重新染色与旋转修复被破坏的性质

删除原则：BST 的删除均可递归地转化为删除叶结点的情况，对于红黑树则是转化为删除至少有一个子结点是外部结点的结点，依旧是通过染色和旋转修复被破坏的性质

RBT 插入与删除的代价均为 $O(\log n)$

参见 [Red–black tree]() 与 CLRS 中 RBT 的相关内容