IV Leftist/Skew Heap, Binomial Queue

Review of Heap

堆（heap）也称作优先队列（priority queue），支持插入值、找最小 / 大值，删除最小 / 大值

二叉堆（binary heap）中的 **任意节点的值都不大于（或不小于）其父节点的值（即最大堆或最小堆）**

- 插入，先放到最后一个位置，然后和父节点比较，不满足条件则和父节点交换
- 删除，将最后的叶节点放到根节点，然后向下比较，不满足则和最大的儿子交换
- 建堆：Floyd建堆算法，从最后一个非叶节点开始，向前遍历到根节点

其中，插入删除的时间复杂度都是 $O(\log n)$, 而建堆的时间复杂度则是 $O(n)$, 但是当我们尝试对两个堆进行 merge 操作时，其实等效为插入 n 个节点，所以需要

$O(n\log n)$ 量级，而此时问题的输入可以理解为将 n 个数复制到另一个队列里，这样的操作其实会有浪费。

Leftist Heap (C.A. Crane 1972)

> - Order Property – the same
> - Structure Property – binary tree, but unbalanced

为了使得左倾堆在 merge 的操作上能有更快的速度, C.A. Crane 希望能只处理堆中的一条 path，并且这条 path 的长度应该尽可能的短，所以他尝试构造并维护一个不平衡的二叉树：左倾堆

Leftist Heap Property

Definition: The null path length, $Npl(X)$, of any node X is the length of the shortest path from X to a node without two children. Define $Npl(NULL) = -1$. *The leftist heap property* is that for every node X in the heap, the null path length of the left child is at least as large as that of the right child.

> $Npl(X) = \min\{Npl(C) + 1 \text{ for all C as children of X}\}$

Theorem: A leftist tree with r nodes on the right path must have at least 2^r-1 nodes.

> 这个定理可以考虑使用下面的引理来归纳证明
> Lemma: A leftist tree with r nodes on the right path must has $Npl(X) = r-1$

Operations

由于左偏堆不再是一个完全二叉树，所以我们不能使用数组来维护它了。

```
struct LeftistHeapNode {
    ElementType val;
    int npl;
    LeftistHeapNode * ls, * rs;
};
```

另外，如果我们很好的解决了 merge 的操作，那么 insert 其实就是 merge 一个 node 而已

Merge

Merge 的基本思路是利用左倾堆的最右路径较短的性质，排列该路径上的节点(保证 heap 的特性)，然后再决定是否需要 swap 两个孩子节点来满足 leftist 的特性，据此有两种思路

- Recursion: 在每次迭代之后都检查是否不满足迭代的特性
- Iterative: 有点类似分治算法最后的 Merge 处理，先排列该路径上的节点，然后从上到下 swap. 虽然代码可能会复杂一些，但是在限制递归深度和实际模拟操作时更加自然

Delete

先自上而下找到目标节点，然后 merge 他的两个根节点，最后自下而上的 swap 和更新 npl 即可

Skew Heap (Sleator and Tarjan, 1986)

由于不想维护 npl 这个属性，Sleator and Tarjan 设计出了 Skew Heap 来保证其 Amortized Time 是 $O(n)$ 的

Merge

Always swap the left and right children *except that the largest of all the nodes on the right paths does not have its children swapped*. 在模拟时可以理解为 **无条件交换左右子树**

Amortized Analysis

类比左倾堆的操作，分析 skew heap 的均摊复杂度，主要就是分析合并操作的复杂度，因为其他操作都可以转化为合并操作。

先定义势能函数，$\Phi(D_i) = \text{number of heavy nodes}$, 其中

$D_i = \text{the root of the resulting tree}$

> Definition: A node p is heavy if the number of descendants of p's right subtree is at least half of the number of descendants of p, and light otherwise. Note that the number of descendants of a node includes the node itself.

很容易验证这个势能函数满足定义，则操作的摊还开销可以写为

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_x) - \Phi(D_y)$$

并且观察后不难得到以下性质:

- 如果一个节点是 heavy node，并且在其右子树发生了合并，那么它一定变为 light node;
- 如果一个节点是 light node，并且在其右子树发生了合并，那么它可能变为 heavy node;
- 只有最右侧路径上的点会有状态变化

进一步定义 l_x 为 D_x 最右侧路径上的 light node (heavy node 记为 h_x), 那么上式可以展开:

$$c_i = l_x + h_x + h_x + h_y$$
$$\Phi(D_i) - \Phi(D_x) - \Phi(D_y) \leq l_x + l_y - h_x - h_y$$

那么摊还开销 $\hat{c}_i$, 应该满足:

$$\hat{c}_i \leq 2 \cdot (l_x + l_y)$$

可以通过归纳得出 (是我们课上的讨论题) 最右侧路径上的轻点个数是 $O(\log N)$，综上

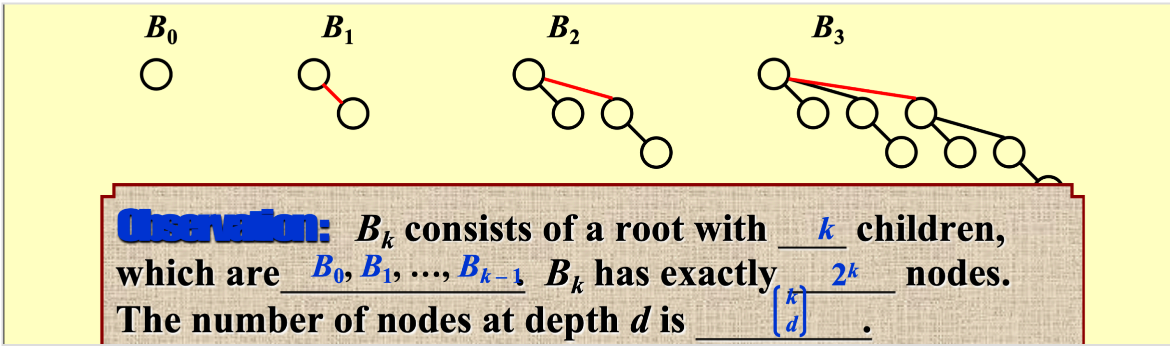
$$\hat{c}_i = O(\log N) \quad \blacksquare$$

Binomial Queue

> A binomial queue is not a heap-ordered tree, but rather a collection of heap-ordered trees, known as a forest. Each heap-ordered tree is a binomial tree (J. Vuillemin, 1978). 其主要是为了优化在用左倾堆和斜堆创建一个新元素仍然需要 $O(\log N)$

Property

A binomial tree of height 0 is a one-node tree. A binomial tree, B_k, of height k is formed by attaching a binomial tree, $B_k - 1$, to the root of another binomial tree, $B_k - 1$.



Operation

```
typedef struct BinNode *Position;
typedef struct Collection *BinQueue;
typedef struct BinNode *BinTree;  /* missing from p.176 */

struct BinNode
{
        ElementType Element;
        Position LeftChild;
        Position NextSibling;
} ;

struct Collection
{
        int     CurrentSize;  /* total number of nodes */
        BinTree TheTrees[MaxTrees];
} ;
```

Merge and Add

可以通过类比为一个 1 bit 全加器来理解

合并规则真值表

T1.B[k]	T2.B[k]	T.carry[k-1]	T.result[k]	T.carry[k]	说明
0	0	0	0	0	无输入，无输出
0	0	1	1	0	只有进位，保留k阶树
0	1	0	1	0	只有一个输入，保留k阶树
0	1	1	0	1	进位+输入，产生k+1阶树
1	0	0	1	0	只有一个输入，保留k阶树
1	0	1	0	1	进位+输入，产生k+1阶树
1	1	0	0	1	两个输入，产生k+1阶树
1	1	1	1	1	三个输入，保留k阶树并产生进位

插入则类比加一个 1 的二项队列

Dequeue

我们只要找到队首 ($O(\log n)$), 然后用去掉队首元素的树构造一个新的二项队列，最后实现$merge(T - B_k, B_k.root.children)$ 即可