#ADS04 Leftist Heaps and Skew Heaps

1. Leftist Heaps

Skew Heaps

1. Leftist Heaps

左式堆

Target: Speed up merging in O(N)

左式堆保留了堆序性质，与二叉堆唯一的区别在于左式堆不是理想平衡的，而是趋于**不平衡**的

Heap : Structure Property + Order Property

Structure Property： the same

Order Property : binary tree , but unbalanced

【Definition】

1. NPL

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**.

We define **NPL(NULL) = −1**

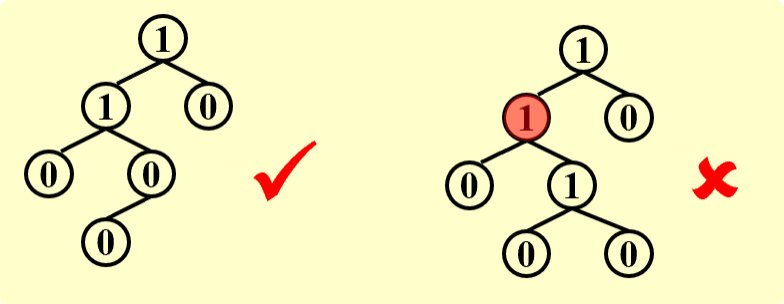
> NPL(X) = min{NPL(C) + 1 for all C as children of X}

2. leftest heap property

The **leftest 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.

The tree is biased to get deep toward the left.

> Npl left >= Npl right



【**Theorem**】 A leftist tree with **r** nodes **on the right path** must have at least $\mathbf{2^r - 1}$ nodes.

在右路径上有r个节点的左式树必然至少有$\mathbf{2^r - 1}$ 个节点。

数学归纳。

由这个定理立刻得到，N个节点的左式树的一条右路径最多含有 $\lfloor \log(N+1) \rfloor$ 个节点。对左式堆的操作一般思路都是将所有的工作放到右路径上进行，保证树深短。

唯一棘手的在于insert and merge 可能会破坏左式堆的性质。

Declaration

Merge:

1. Merge(H1->Right,H2) while H1->element < H2->element
2. Attach(H2,H1->Right)
3. Swap(H1->Right,H1->Left) if necessary

Merge C 复制代码

```c
PrirityQueue Merge(PrirityQueue h1,PrirityQueue h2)
{
    if(H1 == NULL) return H2;
    if(H2 == NULL) return H1;
    if(H1->Element < H2->Element)
        return Merge1(H1,H2);
    else rethrn Merge1(H2,H1);
}
```

Merge1 C 复制代码

```c
static PrirityQueue
Merge1(PrirityQueue h1,PrirityQueue h2)
{
    if(H1->Left == NULL)
        H1->Left = H2;
    else{
        H1->Right = Merge(H1->Right,H2);
        if(H1->Left->Npl < H1->Right->Npl)
            Swap(H1->left,H2->right);
        H1->Npl = H1->Right->Npl + 1;
    }
    return H1;
}
```

Skew Heaps

a simple version of the leftist heaps

Target : Any M consecutive operations take at most **O(MlogN)** time

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.

No Npl.

skew heap 是 leftest heap的子调节形式，斜堆与左式树的关系类似与伸展树与AVL树间的关系。不同于左式堆，关于任意节点的零路径长的任何信息都不保留。右路径可以任意长，所以所有操作的最坏运行时间为O(N).

但可以证明，任意M次连续的操作，**总的最坏运行时间为 O(MlogN)**

对于Merge, 我们执行与以前完全相同的操作，除了一个例外。

在左式堆中，我们查看堆序性质并交换不满足条件者，但对于斜堆，除了这些右路径上所有节点的最大者不交换它们的左右儿子，交换是无条件的。

Skew heaps have the advantage that **no extra space is required to maintain path lengths** and **no tests are required** to determine when to swap children.

1-1 分数 1 作者 陈越 单位 浙江大学

The result of inserting keys 1 to $2^k - 1$ for any $k > 4$ in order into an initially empty skew heap is always a full binary tree.

◉ T ○ F

答案正确：1 分 创建提问

可以使用数学归纳法进行证明。

1-2 分数 1 作者 陈越 单位 浙江大学

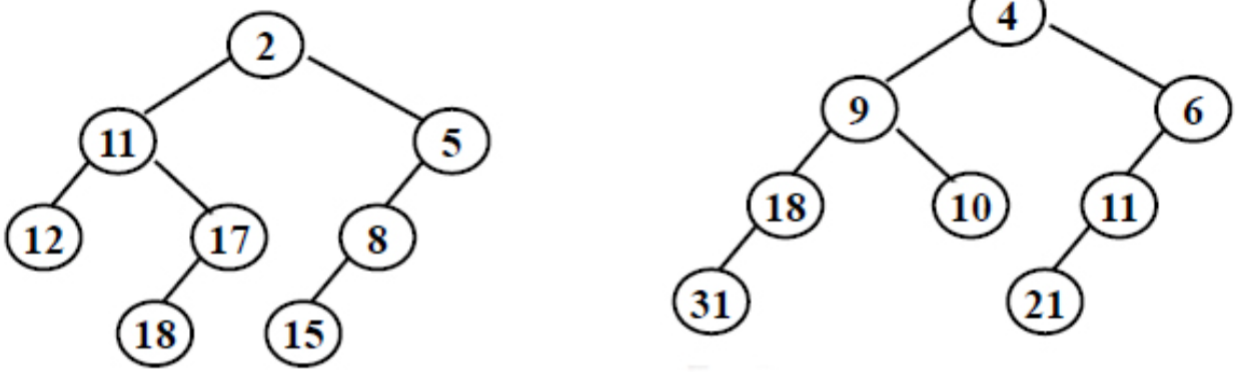
The right path of a skew heap can be arbitrarily long.

◉ T ○ F

答案正确：1 分 创建提问

2-1 分数 2 作者 陈越 单位 浙江大学

Merge the two leftist heaps in the following figure. Which one of the following statements is FALSE?



○ A. 2 is the root with 11 being its right child

○ B. the depths of 9 and 12 are the same

○ C. 21 is the deepest node with 11 being its parent

◉ D. the null path length of 4 is less than that of 2

答案正确：2 分 创建提问

2-2 分数 3 作者 陈越 单位 浙江大学

We can perform BuildHeap for leftist heaps by considering each element as a one-node leftist heap, placing all these heaps on a queue, and performing the following step: Until only one heap is on the queue, dequeue two heaps, merge them, and enqueue the result. Which one of the following statements is FALSE?

- A. in the k-th run, $\lceil N/2^k \rceil$ leftist heaps are formed, each contains 2^k nodes
- B. the worst case is when $N = 2^K$ for some integer K
- C. the time complexity $T(N) = O(\frac{N}{2}log2^0 + \frac{N}{2^2}log2^1 + \frac{N}{2^3}log2^2 + \cdots + \frac{N}{2^K}log2^{K-1})$ for some integer K so that $N = 2^K$
- ◉ D. the worst case time complexity of this algorithm is $\Theta(NlogN)$

答案正确：3 分 创建提问

使用分治的方法建立左式堆、

T(N) = 2 * T(N/2) + O(logN)

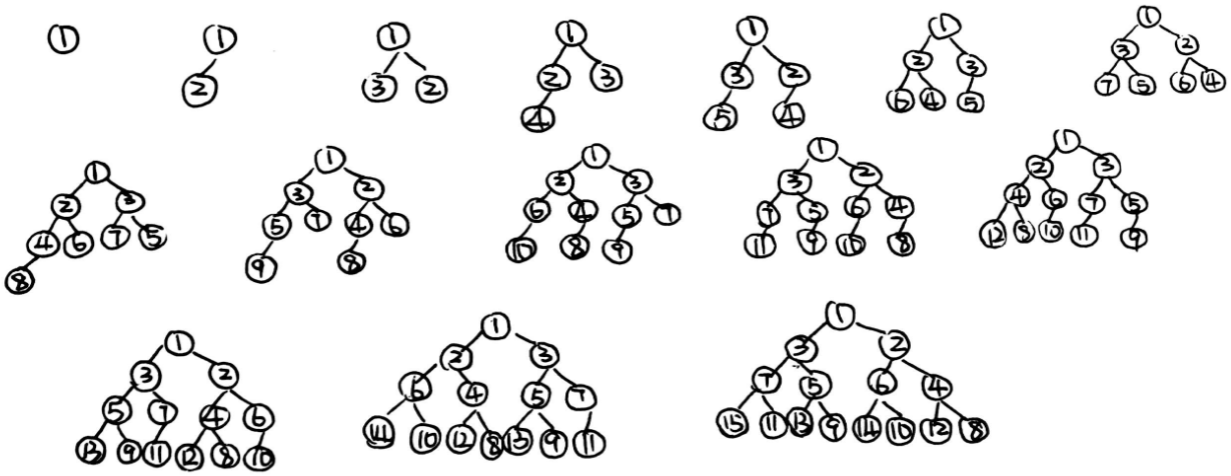
O(N)

2-3 分数 2 作者 陈越 单位 浙江大学

Insert keys 1 to 15 in order into an initially empty skew heap. Which one of the following statements is FALSE?

- A. the resulting tree is a complete binary tree
- B. there are 6 leaf nodes
- C. 6 is the left child of 2
- D. 11 is the right child of 7

答案正确：2 分 创建提问



斜堆（skew heap）是一种简易的可并堆。

对于小根堆，合并时，比较两堆的根，将权值小的根作为新堆的根，将另一个堆与新堆的右子树合并，然后交换新堆的左右子树。

定义：

若一个节点的右子树大小不小于以该节点为根的子树大小的一半，则称这个节点为重节点，否则为轻节点。定义斜堆的势为该斜堆中重节点的个数。

容易发现，合并两个斜堆相当于把两个堆从根到最右节点的路径合并后作为新堆的从根到最左节点的路径。那么时间复杂度为两个堆极右路径的和。

设 h_1, l_1, h_2, l_2 分别为第一个堆的极右路径上重节点的个数，第一个堆的极右路径上轻节点的个数，第二个堆的极右路径上重节点的个数，第二个堆的极右路径上轻节点的个数。$c = h_1 + l_1 + h_2 + l_2$。

发现，一个重节点合并后必然变成轻节点。因为我们要分析最坏情况下的摊还时间复杂度，所以我们令势的变化最大，即所有轻节点变为重节点，也即 $t_i - t_{i-1} = -h_1 + l_1 - h_2 + l_2$，那么 $k_i = c + (t_i - t_{i-1}) = 2(l_1 + l_2)$。

我们将证明 $l_1 + l_2 = O(\log n)$。显然有每个轻节点的右子树小于以该节点为根的子树大小的一半，所以每次子树规模至少缩减一半，故而为 $O(\log n)$。

代入势能分析的式子可得

$$\sum_{i=1}^{n} c_i = n \log n - t_n + t_0$$

根据定义，我们知道 $t_n = O(n)$，$t_0 = 0$，所以总时间摊还复杂度为 $O(n \log n)$。