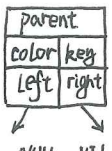
4. Red-Black Tree

平衡二叉搜索树



black-height of node X，标记为 bh(x)，指的是从X点到叶子的路上黑结点数（X不包含在内）

Lemma：一个有N个internal node的红黑树 bh(tree) ≤ 2ln(N+1)

特征：(1) 结点不是红的就是黑的；

(2) 根是黑的

(3) leaf (Nil)是黑的

(4) 如果一个节点是红的，那么其两个孩子都是黑的． T=O(h)=O(lnN)

(5) 所有从根到叶子的path上，黑结点数相同．

5. Leftist Heaps

目标：让合并加速为O(N) Heap：结构属性+order属性

左堆：顺序属性：与堆相同

结构属性：二叉树但不平衡

Npl(X) = min{Npl(C)+1 for all C as children of X} null path length, Npl 指的是从点X到一无两孩子点的最短路径长．

叶子最左堆的所有点，其左子结点的Npl必不小于右子结点的Npl

insert 当作特殊的 merge．

设 H_1 的根小于 H_2 的根

递归版：Step 1：Merge (H_1->Right, H_2)

Step 2：Attach (H_2, H_1->Right)

Step 3：Swap (H_1->Right, H_1->Left) if necessary

```
Merge1(H1, H2) {
  if(H1->Left == NULL)
      H1->Left=H2;
  else{
      H1->Right = Merge(H1->Right,H2);
      if(H1->Left->Npl< H1->Right->Npl)
          Swap(H1)
      H1->Npl=H1->Right->Npl+1;
  }
  return H1;
}
```

6. Skew Heaps (最左堆简化版)

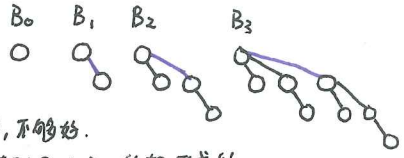
目标：任何连续的M次操作最多花O(M logN)

Merge 时算步永远交换左右孩子．

Skew Heap 有以下优点：①无需多余空间存NPL

②不用测试是否要交换左右孩子．

Tamortized = O(logN)



7. Binomial Queues

最左堆与斜堆 insert 为O(logN)，不够好．

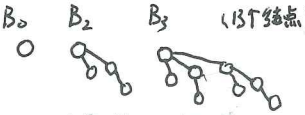
B_0 为单点树，B_k 是由一个 B_{k-1} 连到另一个 B_{k-1} 的根形成的

B_k 的根有k个子节点，从 B_0, B_1, … B_{k-1}

B_k 有 2^k 个节点，在depth d的点数有 $\binom{k}{d} = \frac{k\cdot(k-1)\cdots(k-d+1)}{d!}$

用途：例：用 Binomial Queue 表示一个 Priority Queue，size=13

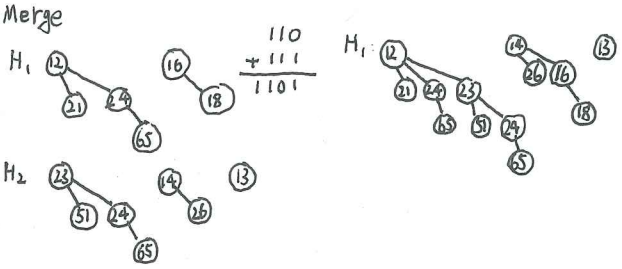
$(13)_{10} = (1101)_2$



操作：(1) Find Min

由于最小点必在根部，因此只需找root即可．Tp = O(logN)．如果用一个 minimum 记忆只需 O(1)

(2) Merge



(3) insert

在空的初始 queue 上插入会遇于最坏情况，总计 O(N)，因此平均为 O(1)

(4) Delete Min

Step 1：FindMin in B_k // O(logN)

Step 2：Remove B_k from H // O(1) B_k 以外为 H'

Step 3：Remove root from B_k // O(logN) B_k 删去根后产物为 H''

Step4：Merge (H', H'') // O(logN)