高级数据结构与算法分析

Chap 1 AVL Trees, Splay Trees and Amortized Analysis

1. Adelson-Velskii-Landis (AVL) Tree

平衡因子 (Balance factor)
$BF(node) = h_L - h_R$
在AVL树中，BF(node)只能为 -1, 0, 1

① 规定空树规定高度=-1是 height balanced.

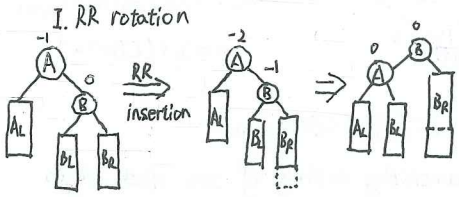
② 当满足以下条件，称T为 height balanced.

I. 左子树 T_L 与右子树 T_R height balanced

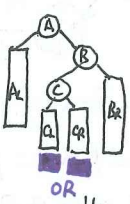
II. $|h_L - h_R| \le 1$ (h_L 与 h_R 为左、右子树高)

③ 各类操作：

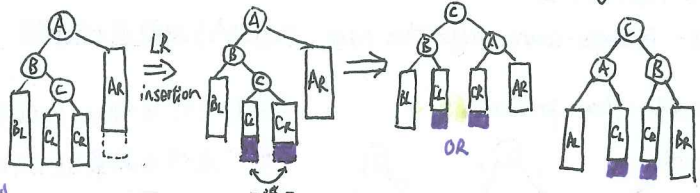
I. RR rotation



III. RL Rotation



II. LR rotation



队列

BF(node)=±2的点作为变换节点的根进行操作

$\therefore h = O(\ln n)$

$T_P = O(h)$

令n_h为高度平衡树高度为h时，最少点数

$n_h = n_{h-1} + n_{h-2} + 1$

$n_h = F_{h+2} - 1$, for $h \ge 0$ $\quad F_i \approx \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^i$

2. Splay Tree

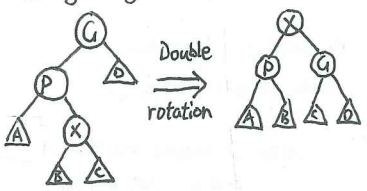
访问一个点后就将该点通过AVL旋转推到root.

设某非根点X，父结点为P，祖父结点为G

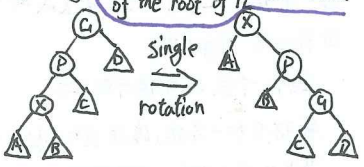
Case 1: P为根 ⇒ 旋转X与P

Case 2: P非根

I. Zig-Zag



II. Zig-Zag



Deletion: Step 1: Find X ← X 到 root

Step 2: Remove X ← 两个子数 T_L 与 T_R

Step 3: FindMax(T_L) ← 会成为根

Step 4: Make T_R the right child of the root of T_L

3. B+ Tree

特征：order为M的B+树：

(1) 根要么是叶子 要么有2~M个孩子

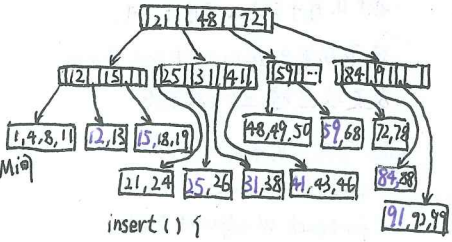
(2) 除根的非叶子节点孩子在⌈M/2⌉~M间

(3) 所有叶子深度相关，即在同一层

(4) 所有实际数据在叶子中

(5) 每个interior node 都包含M个指向子结点指针

(6) key值都是按下来孩子最小值



insert ( ) {

Search from root to leaf and find the proper leaf node;

Insert X;

while (this node has M+1 keys) {

split it into 2 nodes with ⌈(M+1)/2⌉ and ⌊(M+1)/2⌋ keys respectively;

if (this node is the root)

Create a new root with two children

check its parent;

}

}

$T(M,N) = O((M/\log M)\log N)$

$Depth(M,N) = O(\lceil \log_{\lceil M/2 \rceil} N \rceil)$

$T_{Find}(M,N) = O(\log N)$

---

6. sorting large structure

交换真实大结构数值成本过高，替代法为仅交换指针

Table sort

list	[0]	[1]	[2]	[3]	[4]	[5]
key	d	b	f	c	a	e
table	4	1	3	0	5	2

The sorted list is

list[table[0]], list[table[1]],

… list[table[n-1]]

7. 搜索算法下限

Theorem: 任何基于比较的搜索算法，最坏复杂度 $\Omega(N \log N)$

Proof: when sorting N distinct elements, there are N! different results.

Any decision tree must have at least N! leaves

If the height of the tree is k, then $N! \le 2^{k-1}$

$k \ge \log_2(N!) + 1$

$\because N! \ge (N/2)^{N/2} \quad \therefore \log_2(N!) \ge (N/2)\log_2(N/2) = \Theta(N \log_2 N)$

$\therefore T(N) = k \ge C \cdot N \log_2 N$

8. Bucket Sort and Radix Sort

桶排：$T(N,M) = O(M+N)$

```
initialize count[];
while (read in a student's record)
    insert to list count[Stdnt.grade];
for(i=0; i<M; i++)
    if (count[i])
        output list count[i];
```

MSD (Most Significant Digit)

LSD (Least Significant Digit)

Radix Sort: 数列中最高位数即为执行轮次

原始数组: 72, 99, 6, 25, 17, 22, 35, 64, 98, 12

	0	1	2	3	4	5	6	7	8	9
round 1	72	22	12	64	25	35	06	17	98	99
round 2	06	12	17	22	25	35	64	72	98	99

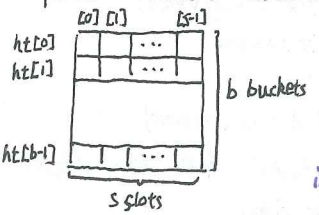
适用于 bucket 远多于数的情况

Chap 7 Hashing

1. Symbol Table ADT

A set of name-attribute pairs, where the names are unique

Operation: Create() isInTable() Find() insert() delete()



f(x) = 该单词进入几号 bucket

T = x 可能值总数

n = 表内已含元素数量

identifier density ::= n/T

loading density λ ::= n/(sb)

collision: $f(i_1) = f(i_2)$ when $i_1 \ne i_2$

without overflow: $T_{search} = T_{insert} = T_{delete} = O(1)$

hash function 应易于计算并且能减少冲突到最低，此外应为 unbiased 的，即分布概率均等.

3. Separate Chaining

让 TableSize 与 key 数量大致近似，$\lambda \approx 1$

4. Open Address

找到另一个空槽放数据以解决冲突.

① naive 法：先 hash，发现冲突后顺延至下一格，直到发现空位 $f(i) = i$

Cause primary clustering: any key that hashes into the cluster will add to the cluster after several attempts to resolve the collision.

② Quadratic Probing $f(i) = i^2$

当 table size 为奇数时，表内已有元素不超过 ⌊TableSize/2⌋ 时，总能用 $f(i) = i^2$ 不冲突找到空位

问题：① 当删除i段台插入会严重影响效率

② secondary clustering：key会被hash到相同的替补位

③ double hashing

$f(i) = i * hash_2(x)$ ($hash_2(x)$ 为第二层 hash 函数)

$hash_2(x) \ne 0$ 经验法则而言，$hash_2(x) = R - (x \% R)$

(R为质数，小于 TableSize) 效果较好

5. rehashing

在特定时间要新建一个原表大小两倍的表，然后把原表内的元素 rehash 至新表.