ADS 期中复习

AVL 树

- $n_h = F_{h+3} - 1$. n_h 是最小结点数。这里空树的高度定义为 -1.
- In an AVL tree, it is possible to have this situation that the balance factors of a node and both of its children are all +1. (T) 比较容易构造。
- For an AVL tree, the balance factors of all the non-leaf nodes are 0 iff the tree is a complete binary tree. （F） 当且仅当它是完美二叉树。

Splay 树

- How to delete?

红黑树

- For any node x, $sizeof(x) \geq 2^{bh(x)} - 1$. A red-black tree with N internal nodes has height at most $2\ln(N+1)$.
- 插入和删除操作
- In a Red-Black tree, the path from the root to the nearest leaf is no more than half as long as the path from the root to the farthest leaf. (F) 根据黑高相等以及一条路径上红节点一定小于等于黑节点个数可以得到，不等号反了。
- In a red-black tree, if an internal black node is of degree 1, then it must have only 1 descendent node. (T) Why???
- In a red-black tree, the number of rotations in the EDLETE operation is $O(1)$. Why??
- Is it true that the DELETE operation in a RED-BLACK tree of n nodes requires $\Omega(\log n)$ rotations in the worst case? (F)

B+ 树

- A 2-3 tree with 12 leaves may have at most 10 nonleaf nodes. （T）

熟悉 M 阶 B+ 树的概念，以及插入的过程是如何分支的，就可以画出来。

- the root always has between 2 and M children.(F) 注意定义中 root 也可能是 Leaf.

倒排索引

- Inverted file contains a list of pointers to all occurrences of that term in the text.
- Precision $P = R_R/(R_R + I_R)$.
- Recall $R = R_R/(R_R + R_N)$.
- While accessing a term stored in a B+ tree in an inverted file index, range searched are expensive(F). Hash 搜索范围 expensive, B+ 不对(为什么？)
- In distributed indexing, document-partitioned strategy is to store on each node all the documents that contain the terms in a certain range. (F) 不是所有的
- Stemming increases recall while harming precision.

Leftiest Tree and Skew Tree

- 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)

如果个数满足一个完全二叉树，那么用skew插入是可以插出完全二叉树的。

- The right path of a skew heap can be arbitrarily long.(T) 注意是斜堆，没有 NPL 的约束。
- A skew heap is a heap data structure implemented as a binary tree. Skew heaps are advantageous because of their ability to merge more quickly than balanced binary heaps. The worst case time complexities for Merge, Insert, and DeleteMin are all $O(N)$, while the amorited time complexities for Merge, Insert, and DeleteMin are all $O(\log N)$. (注意到 skew tree 的 Insert, delete 操作实际上就是 Merge)。
- 对于左式堆的 merge, 不要忘记计算 Npl 决定是否 swap。
- The number of light nodes along the right path of a skew heap is $O(\log N)$.
- the amortized time complexity of insertions in a heap is $O(\log N)$, that of deletions is $O(1)$.(??)
- A leftist tree with r nodes on the right path must have at least $2^r - 1$ nodes.

Binomial Heap

- To implement a binomial queue, the subtrees of a binomial tree are linked in increasing sizes.(F)

横向没有大小限制。

- For a binomial queue, insertion takes a constant time on average. (An insertion that costs C units results in a net increase of $2 - c$ trees in the forest, Φ_i = number of trees after insertion).
- Performing N inserts on an initially empty binomial queue will take $O(N)$ worst-case time.
- To implement a binomial queue, the subtrees are linked in increasing size(F)

数据结构部分复杂度

- Splay: Any M consecutive tree operations starting from an empty tree take at most $O(M \log N)$.
- B+ Tree order M: $T(M, N) = O((M/\log M) \log N)$.

BackTracking

- In backtracking, if different solution spaces have different sizes, start testing from the partial solution with the smallest space size would have a better chance to reduce the time cost.

Divide and Conquer

Master method(another form)

$T(N) = aT(N/b) + f(N)$:

- If $af(N/b) = \kappa f(N)$ for some constant $\kappa < 1$, then $T(N) = \Theta(f(N))$.
- If $af(N/b) = Kf(N)$ for some constant $K > 1$, then $T(N) = \Theta(N^{\log_b a})$.
- If $af(N/b) = f(N)$, then $T(N) = \Theta(f(N) \log_b N)$.

$T(n) = T(n-1) + \log n = \Theta(n \log n)$. $(\log n!) = \Theta(n \log n)$

$T(n) = T(n/6) + T(7n/9) + O(n)$ is $O(n)$. 可以用归纳法证明。

DP

In dynamic programming, we derive a recurrence relation for the solution to one subproblem in terms of solutions to other subproblems. To turn this relation into a bottom up dynamic programming algorithm, we need an order to fill in the solution cells in a table, such that all needed subproblems are solved before solving a subproblem. Among the following relations, which one is impossible to be computed?

- ○ A. $A(i,j) = min(A(i-1,j), A(i,j-1), A(i-1,j-1))$
- ○ B. $A(i,j) = F(A(min\{i,j\}-1, min\{i,j\}-1), A(max\{i,j\}-1, max\{i,j\}-1))$
- ○ C. $A(i,j) = F(A(i,j-1), A(i-1,j-1), A(i-1,j+1))$
- ◉ D. $A(i,j) = F(A(i-2,j-2), A(i+2,j+2))$

B?