数据结构与算法 A

周书予
2000013060@stu.pku.edu.cn

2021 年 12 月 28 日

1 图论

好像没有什么阿拉丁玩意儿。
拓扑排序、最短路、最小生成树总还是会的吧

2 内排序

2.1 直接插入排序

就是维护一个已排好序的前缀，每轮把这段前缀的后面一个数加进去。注意一开始默认第一个数是有序的，因此 k 轮后已排序前缀的长度是 $k+1$。

2.2 Shell 排序

取 $D=\{d_1,d_2,\cdots,d_m\}$，第 k 轮以 d_k 的间隔把整个序列分成 d_k 个子序列，每个长度 (差不多) 为 n/d_k 的子序列内部做插入排序排成有序。

2.3 直接选择排序

仍然是维护一个已排好序的前缀，每轮在未排序的部分找到一个最小的数放在这段前缀的后面。
直接选择排序的交换次数不超过 $n-1$。同时比较次数不依赖于序列的初始状态。

2.4 堆排序

注意建堆的时间复杂度是 $\Theta(n)$。

2.5 冒泡排序

每轮冒泡排序结束后，一个数后面比自己小的数一定会恰好减少一个。所以冒泡排序需要的轮数就是这个东西的最大值 +1(最后一轮发现不需要再排了)。
冒泡排序也维护了已排好的前缀，而且也是每次长度加一。

2.6 快速排序

先把轴值 (pivot) 交换到最后边，然后取出来 (相当于把最右边当成空位)，l,r 指针每次把左边大于轴值的和右边小于轴值的元素放到空位上 (并产生新的空位)。

2.7 归并排序

2.8 桶排序与基数排序

2.9 索引排序

索引 1 下标 IndexArray1 就是 Rank，索引 2 下标 IndexArray2 就是 SA。两者互为逆置换。

2.10 排序算法的稳定性

直接插入排序、冒泡排序、归并排序、桶排序和基数排序是稳定的。

直接选择排序、Shell 排序、快速排序和堆排序是不稳定的。

(上述) 只有直接插入排序和冒泡排序在最好情况下有更低量级的复杂度，其他算法的复杂度即使在最好情况下也不变。

Shell 排序和堆排序不适合链表，因为访问不连续。其他排序算法都是可以访问连续的。

3 文件管理与外排序

3.1 置换选择排序

维护一个内存大小为 M 的堆，需要持续从输入 buffer 中读数据，并往输出 buffer 中写一个递增的序列。如果某次读到一个数比已经输出过的数要小，那么这个数就寄了 (不可能再被输出了)，也就相当于堆大小减一。

平均意义下可以输出长度为 $2M$ 的递增序列，称为**顺串**。

3.2 多路归并

如果要归并 k 个顺串，那就建一棵大小为 k 的线段树，在每个非叶子节点上写子树里最优的结果来自于那个顺串 ($\in [1,k]$)。这棵树叫做赢者树。

把线段树节点上记录的最优改成记录“哪个串在这个节点发生的比较上寄了”，这样每个 $[1,k]$ 都会在树上出现恰好一次——冠军的话，会在线段树的根上面再长出一个节点来记录。

3.3 读盘和写盘次数

就是每次参与归并的总长度。

4 检索

4.1 平均检索长度

需要注意检索失败时检索长度被认为是 $n+1$。以 p 的概率成功顺序检索的平均检索长度是 $p(\frac{n+1}{2})+(1-p)(n+1)=\frac{n+1}{2}(1-\frac{p}{2})$。

4.2 二分查找某个节点的比较次数

就是二叉搜索树上节点的深度。

4.3 散列冲突解决方法

有开散列方法和闭散列方法。

开散列方法就是 Hash 挂链。

闭散列就是冲突/碰撞时找别的位置。具体的，对于每个关键码 K，构造一列散列地址序列 $d_0,d_1,d_2,\cdots$，其中 $d_0=h(K), d_i=(d_0+p(K,i)) \bmod M$。

- 线性探测：$p(K,i)=i$，可改进为 $p(K,i)=ci$
- 二次探测：$p(K,2i-i)=i^2, p(K,2i)=-i^2$
- 随机探测：$p(K,i)=\sigma_i$ 其中 σ 是一个 $[1,M-1]$ 的排列
- 双散列探测：$p(K,i)=ih_2(K)$，其中 h_2 是另一个散列函数

4.4 散列表的删除

注意，散列表删除是不能直接删掉元素的。这是因为在闭散列表中，直接删除一个元素会导致其他元素的探测路径上出现空位，导致查询时出错。正确的做法是留下一个墓碑表示被删除，但仍然占据这个地址。

插入的时候可以插在墓碑上，但需要扫描整个探测序列直到发现空地 (为了避免重复插入)

5 索引技术

5.1 倒排表

就是把某一项特征写在第一列。

5.2 静态索引和动态索引

就是支不支持动态维护。接下来的 B 树和 B+ 树都属于动态索引。

5.3 B 树

m 阶 B 树满足

- 每个节点不超过 m 个子节点
- 非根非叶子节点至少有 $\lceil m/2 \rceil$ 个子节点
- 根至少有两个子节点，除非平凡的情况
- 所有叶子节点在同一层，有 $\lceil m/2 \rceil - 1$ 到 $m-1$ 个关键码

一个节点有 k 个子节点，说明有 $k-1$ 个关键码

一棵 m 阶深度为 k(第 0 层到第 $k-1$ 层) 的 B 树, 键值 (key) 数 N 至少有?

考虑键值数会恰好等于最后一层连出去的空指针数量减一，也就是“理论上的第 k 层”的节点数，是 $2\lceil \frac{m}{2} \rceil^{k-1}$，所以

$$N+1 \geq 2\lceil \frac{m}{2} \rceil^{k-1}, k \leq 1 + \log_{\lceil m/2 \rceil} \frac{N+1}{2}$$

5.3.1 插入

插入时一定会找到一个对应的叶子节点，插入后该叶子节点关键码数量小于 m 则直接结束；

否则产生上溢出，需要分裂成两个分别有 $\lceil (m-1)/2 \rceil$ 和 $\lfloor (m-1)/2 \rfloor$ 个关键码的叶子，多出来一个关键码丢给父亲。

父亲可能会继续上溢出，持续这个过程，如果分裂根，则将导致树的深度加一。

访外次数：向下找的读盘次数 (= 深度) + 1 + 2 × 分裂次数

5.3.2 删除

如果待删除的节点不在叶子层，就与后继交换。后继一定在叶子层。

直接删除后叶子节点关键码数量可能少于 $\lceil m/2 \rceil - 1$, 产生下溢出。先找兄弟借, 如果借不了说明邻居都恰好是 $\lceil m/2 \rceil - 1$ 个关键码，此时考虑和兄弟以及父亲处的分界关键码合并。

合并后相当于吃掉了父亲节点的一个关键码，于是父亲节点重复这个过程。如果根仅有的两个子节点合并了，那么树的深度减一。

最多访外次数：向下找的读盘次数 (= 深度) + 访问左右兄弟并在失败后合并 × (深度 - 1) + 根节点重写 $= 4h - 2$

5.4 B+ 树

相比于 B 树中父节点维护的是子节点分界的关键码，B+ 树中维护子节点的最大值，也即 B+ 树叶子节点层包含了维护的所有关键码。

不过注意这时候一个有 k 个关键码的节点对应就有 k 个子节点，因此每个节点的关键码数量为 $\lceil m/2 \rceil$ 到 m。

插入和删除是差不多的

5.5 红黑树

- 红黑树每个节点要么是红的，要么是黑的。根节点一定是黑的。
- 外部节点被认为是黑的。
- 根到所有外部节点路径上的黑点数量相同。
- 不存在相邻的红色节点。

一个节点的 rank 是从这个节点出发到子树内一个外部节点路径上的黑色节点数量，不算自身。整棵树的 rank 是根节点的 rank。

阶为 n 的红黑树，树高最小是 $n+1$，最大是 $2n+1$；内部节点最少时是一棵完全满二叉树，内部节点数为 2^n-1

含有 m 个内部节点的红黑树树高最大是 $2\log_2(m+1)+1$

5.5.1 插入

按照正常 BST 插入，插入后默认红色，如果父亲是黑色则直接结束。如果父亲是红色，那么祖父一定是黑色

此时讨论叔叔的颜色：如果叔叔是黑色，那么就转一下，就能结束了；

如果叔叔的红色，那么就把父亲、叔叔和祖父都反色，相当于把祖父的黑色下放。此时祖父可能产生新的红红冲突，需要继续向上调整。特别地，如果把根的颜色改成了红色，那就再改成黑色，此时树的 rank 加一

5.5.2 删除

如果两个子节点都是内部节点，就和自己的后继交换。此时待删除节点一定有至少一个外部子节点

如果有恰好一个外部子节点，那么待删除节点一定是黑色，子节点一定是红色，此时只需要把子节点提上来改成黑色即可。

如果两个都是外部子节点，那么红色就直接删，黑色的话就留一个“双黑”标记。

接下来考虑怎么解决“双黑”。讨论“双黑”的兄弟节点及其子节点

- 兄弟黑，且有红色子节点——把父亲、兄弟、兄弟的红儿子三个点摊平，染上恰当的颜色后可以直接结束
- 兄弟黑，且儿子全黑——把“双黑”和兄弟的一个黑色上传给父亲，此时父亲可能变成新的“双黑”，递归处理
- 兄弟红——此时父亲一定黑，对兄弟做一次单旋，这样“双黑”的父亲就变成红色，回到前两种情况

6 高级数据结构

6.1 广义表

表就是直观意义上的表，类似于文件系统

$$\text{线性表} \subseteq \text{纯表 (树)} \subseteq \text{可重入表 (DAG)} \subseteq \text{循环表 (图)}$$

$C=(c,(d,A),B,c)$ 是一张广义表，则 $head(C)$ 表示第一个元素，$tail(C)$ 表示去掉第一个元素后得到的子表。

6.2 Trie & Patricia

Patricia 就是压缩后的二进制 Trie

6.3 最佳二叉搜索树

内部节点频率 $\{p_1,p_2,\cdots,p_n\}$，外部节点频率 $\{q_0,q_1,\cdots,q_n\}$。

动态规划。$f(i,j)$ 表示 $\{q_i,p_{i+1},q_{i+1},\cdot,p_j,q_j\}$ 这些节点安排好的最小代价，转移

$$f_{i,j}=W(i,j)+\min_{k=i+1^j}(f(i,k-1)+f(k,j))$$

ASL = Average Search Length 平均检索长度，就是 $\sum p_ic_i$，p 是概率，c 个某个元素的检索长度。

6.4 AVL 树

任意一个节点，其左右子树高度差不超过 1

插入可能导致不平衡，从下往上处理：LL 不平衡或者 RR 不平衡就做一次单旋，LR，RL 需要做两次 (或者说需要把三个点摊平)

6.5 splay