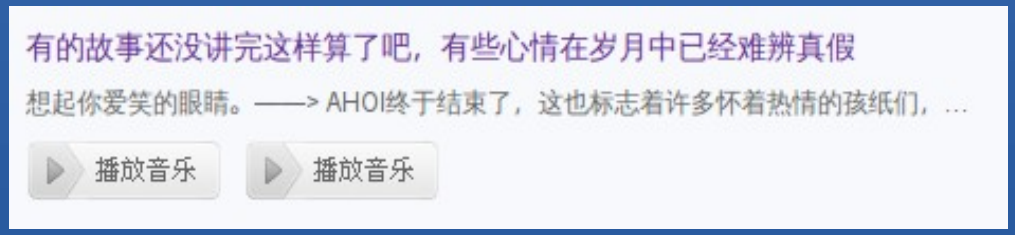
标号法——顺序维护的有力工具

合肥一中 梁泽宇

Mato_No1, RSSC(Real Super Stupid Cross)

关于个人的询问 && 吐槽

- Query0：你为什么叫 Mato No.1 ？难道你每次比赛都是第一？
- Answer0：是的，很多次都是第一，只不过是逆序之后的囧……
- Query1：额……我记得你好像是在一次比赛之后出名的……而且网上对这次比赛的争论好像很多的样子……能说一下这究竟是肿么回事么？



- Answer1：就是一次省选而已，只是它的题目很奇（dou）特（bi），多做这种题目对暴搜能力、乱搞能力以及猜数据能力的提高很有帮助……
- Query2：也就是说，你很弱？那你这次讲的东西会是什么样子？
- Answer2：我是真弱，比一般人想像中的更弱，从我在上一个问题中提到的那次比赛中的表现就可以看出来囧……当然，这次讲的东西也很弱，相信在场的大多数人已经懂了，或者很快就能搞懂……被白天讲的虐得想睡觉的可以开始睡了囧……

引入：序列顺序维护问题

- 先来看一个在某著名世界冠军（吓傻了……）的论文中曾经出现过的一个问题。
- 序列顺序维护问题（List Order Maintenance Problem，LOMP）
- 有一个由若干元素组成的序列，三种操作：
- （0）在某个元素的后面（或序列的开头）插入一个新元素；
- （1）删除某个元素；
- （2）询问某两个元素之间的前后关系（即谁在谁前，保证这两个元素此时必然在序列中存在）。
- 要求在线回答。

序列顺序维护问题的常见解法

- —— 这个问题太容易了囧……一棵平衡树不就搞定了？
- 那么，如果你突然不会写平衡树了，或者比赛时间已经不够写平衡树了，肿么办？

- —— 那就用块状链表！！
- 那么，如果题目不支持根号级别的做法呢？

- ……
- ……
- 应该木有别的办法了囧……

下标是什么？

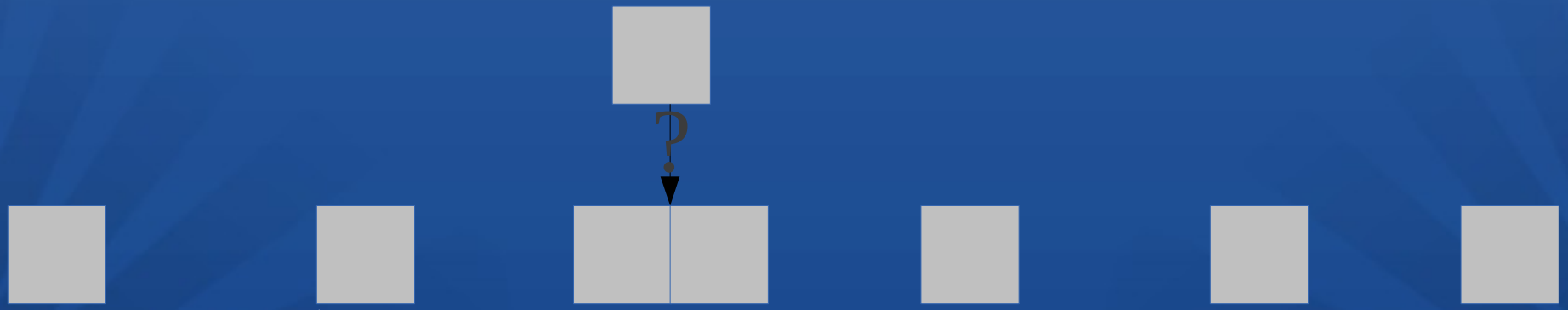
- 如果无视插入和删除操作，也就是这个序列是静态的，则只要一开始将这个序列存入数组，然后在回答询问的时候，调出两个元素在数组中的下标，比较大小，就知道谁在前谁在后了……
- 问题是现在有插入和删除操作，使得仅仅用数组无法支持，所有静态的数据结构（线段树等）好像也不能使用了……
- 然而，所谓的“下标”是什么？
- 是一个表示元素的前后位置的数字，下标越小的，位置越靠前；
- 对于一个大小为 N 的数组，其下标总是 0 ~ N-1；
- 我们考虑一种广义的”数组“，它的下标也满足”表示元素的前后位置的数字，下标越小的，位置越靠前“这个条件，但是它的下标不一定是连续的，也就是大小为 N 的数组，其下标不一定是 0 ~ N-1；
- 它可以是任意递增数列！！

好像这可以处理插入和删除

- 比如，一个大小为 4 的广义数组 S，按顺序存放 A、B、C、D 四个元素，则它可以是 S[0]=A、S[2]=B、S[15]=C、S[37]=D，当然其下标也可以是其它的递增数列。
- 我们发现了一个惊人的事实……
- 这种广义数组好像是可以处理插入和删除操作的！！
- 比如，对于刚才的那个数组，现在要将一个新元素 E 插入到 B 之后，则我们可以给它分配一个大于 2、小于 15 的任意整数（甚至任意实数）作为 E 的下标。
- 如果要删除某个元素，只需要将它的下标从下标集合中去掉就行了囧……对其它元素的下标木有任何影响……
- 其实，这种广义数组的下标已经超越了下标本身的意义，它只表示一个序号而已……所以可以将它称为“标号”。
- 利用这种广义数组，我们似乎已经解决了 LOMP……

标号表示法的问题

- 但是……如何将这种广义数组在内存中存储呢囧……
- 显然用一般的数组是不行的……因为其标号可能非常大，甚至可能是实数……
- 用（标号， 元素）这样的 `pair` 存储？好像可以……
- 但是还有一个更可怕的问题……
- 如果要在两个元素之间插入一个新元素，而这两个元素的标号相差太小，以至于已经插入不了新元素了（如果是整数标号，相差 `1`；如果是实数标号，相差的值已经小于最小能够识别的精度），肿么办？



- 需要一个“重新分配标号”的过程。

如何重新分配标号

- 设想一下这样的局面：桌子上放了一堆物件，摆成一排。
- 现在要将一个新的物件塞到已有的两个物件之间，但这个新物件太大了，以至于塞不进去……
- 将与它相邻的几个物件往两边移开，给新物件留出足够空间……

- 重新分配标号的过程，原来如此……
- 当然，为了保证效率，我们希望能找到一种重新分配标号的方案，使得在最坏情况下，每次插入所引发的重新分配标号的平均次数（即重新分配标号的元素平均个数）不要太多……

动态标号法

- 这种用标号来表示各个元素的位置顺序，以及在插入过程中重新分配标号的办法，称为动态标号法（Online Monotonic List Labeling,OMLL）。
- 动态标号，最容易想到的实现方法是使用重量平衡树……
- 比如 Scapegoat Tree……
- 它可以在任何时刻保证绝对平衡，即它的深度永远不会超过 logN 级别……假设某一时刻它的深度为 D，则对于一个结点 x，只要从根结点走到 x，同时让一个整数从 2^{D-1} 开始，在第 i 层（根为第 0 层）进入左子树时这个整数减去 2^{D-2-i}，进入右子树时这个整数加上 2^{D-2-i}，则到达 x 时这个整数的值就是 x 的标号。
- 它维护平衡的方法是将某棵子树整体拆掉，然后建成完全二叉树……由于一棵子树在 DFS 序列中表示连续的一段……所以实际上就是重新分配了连续一段元素的标号……
- 根据 Scapegoat Tree 的性质可得，平均每次插入时重新分配标号的次数是 O(logN) 的。

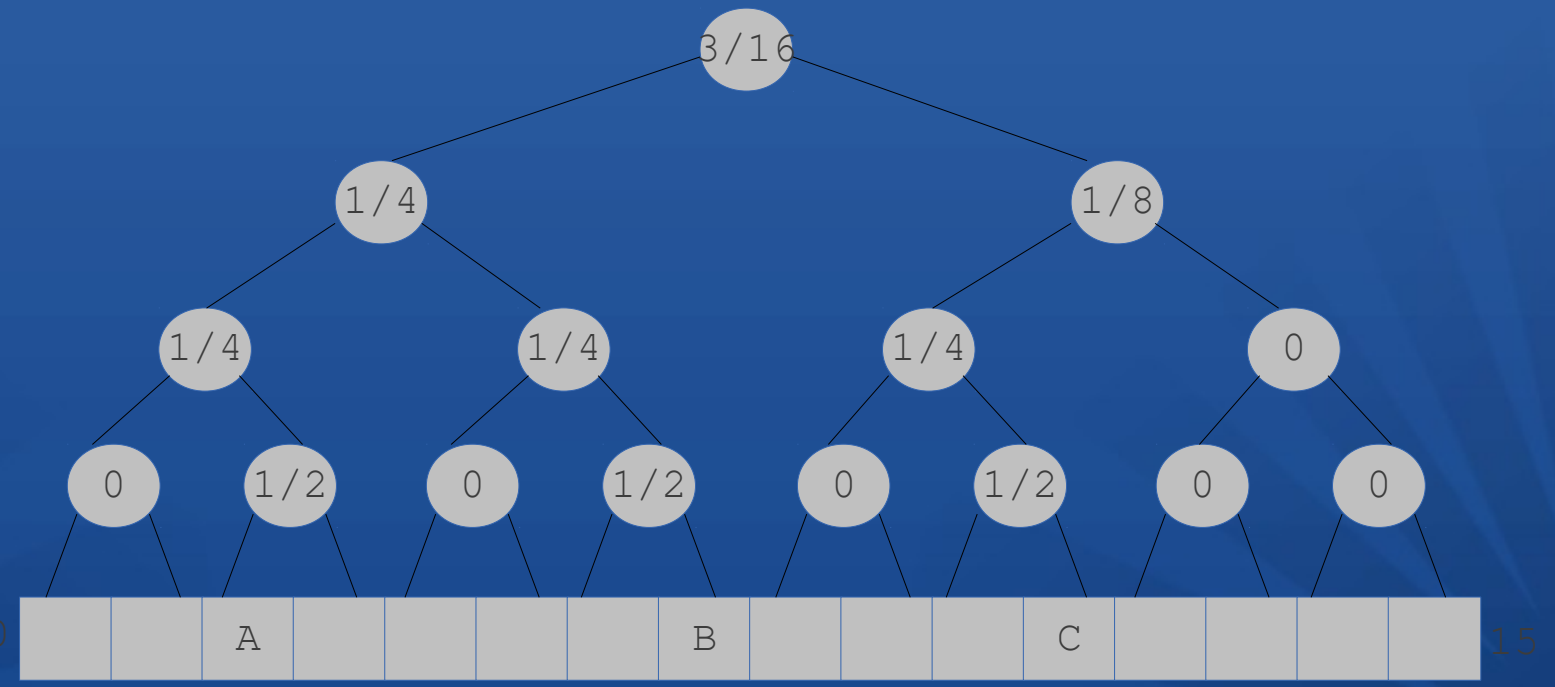
有木有不用平衡树的方法？

- 然而，Scapegoat Tree 并不是很好写囧……
- 有木有好写一些的做法？最好不要使用平衡树。
- 考虑在 0 ~ 2^M-1 的整数范围内给元素分配标号。建立一棵深度为 (M+1) 的满二叉树，同时选定一个临界因子 ε（ $1/2<\varepsilon<1$ ）。
- 在这棵树中，一个结点的管辖范围内，被元素占用的标号个数除以总标号个数的值，称为这个结点的密度（ Density ），记作 $D(x)$ 。
- 对于任意非叶结点 x ，显然有 $D(x)=(D(x.lc)+D(x.rc))/2$ 。
- 下起第 i 层（叶结点为下起第 0 层）的结点的临界密度为 ε^i。在任意时刻，如果某个结点的密度大于其临界密度，则称它是越界的，否则称它是不越界的。

有木有不用平衡树的方法？

- 以下为 M=4 ， ε=3/4 的情况，结点上标记的数字为该结点目前的密度。
- 显然，目前所有结点都是不越界的.....

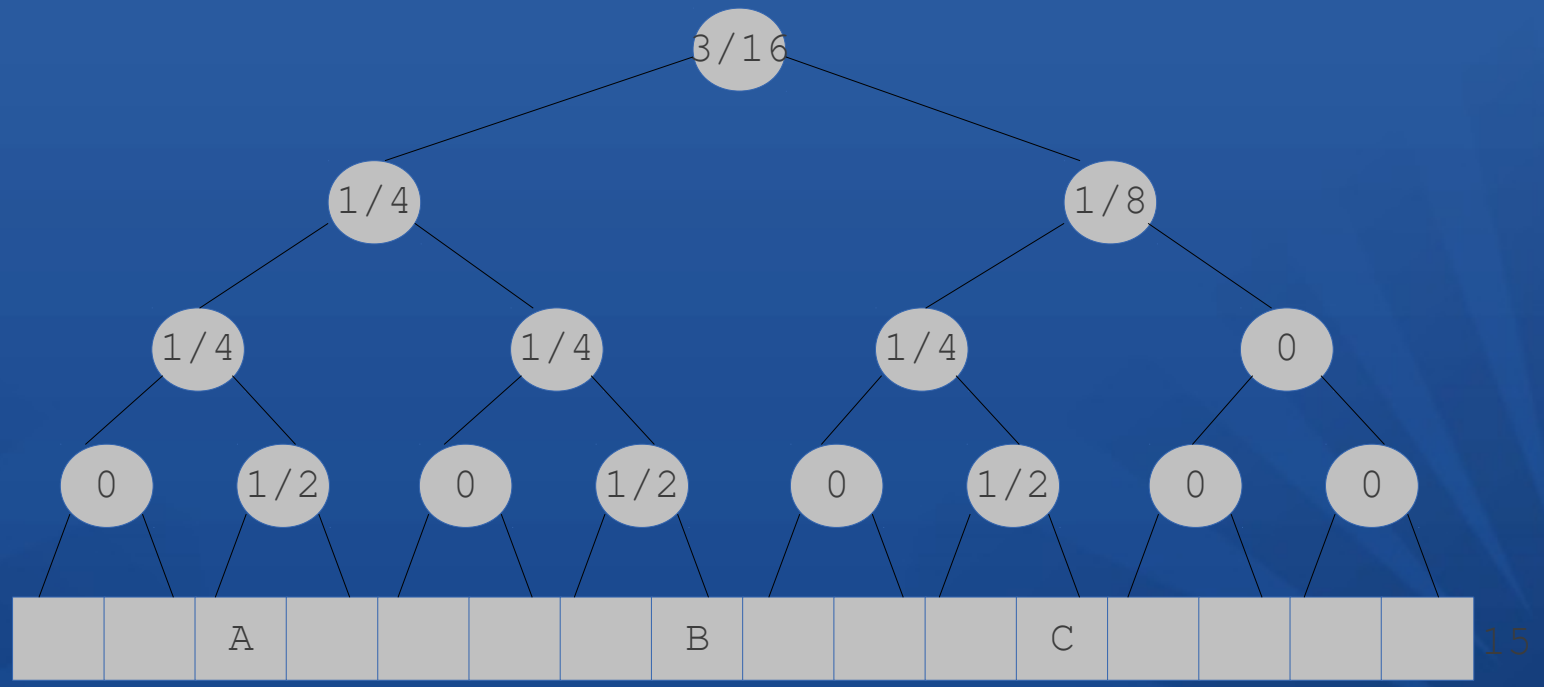
层数	临界密度
4	81/256
3	27/64
2	9/16
1	3/4
0	1



插入及重标号过程

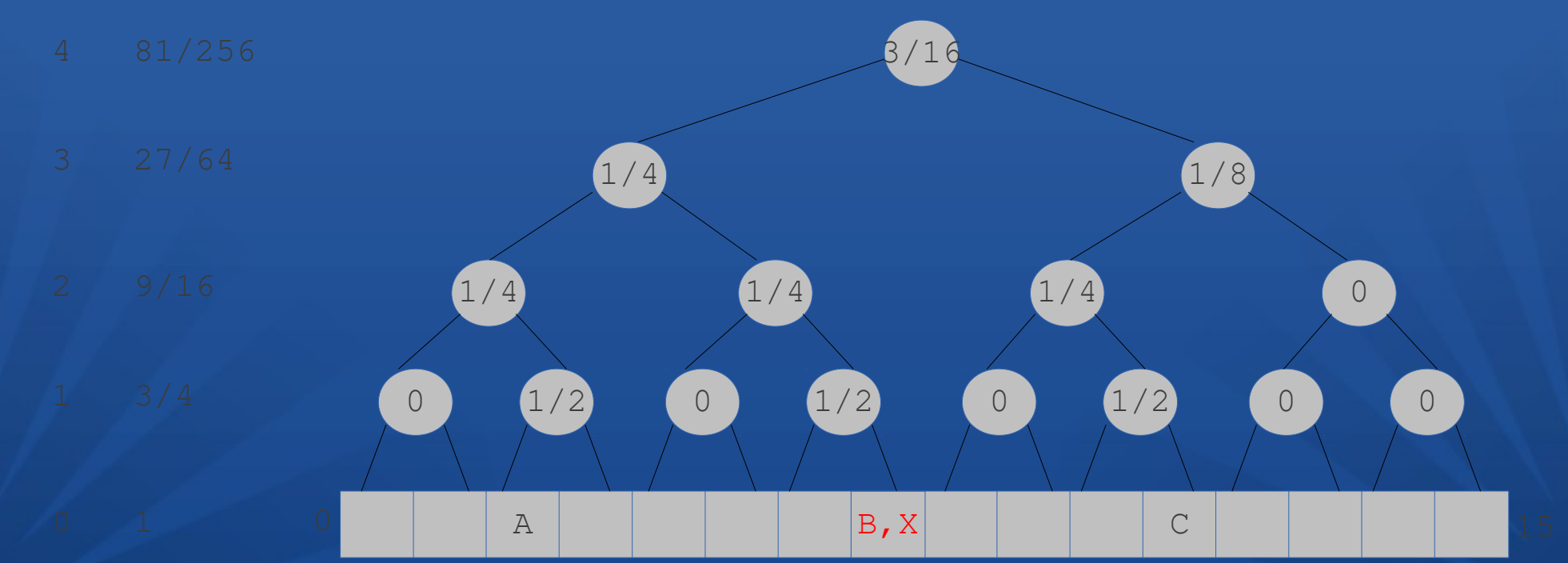
- 现在是如何插入一个元素，以及进行重新分配标号的过程。
- 假设待插入的元素为 X ，要插在 B 的后面。

层数	临界密度
4	81/256
3	27/64
2	9/16
1	3/4
0	1



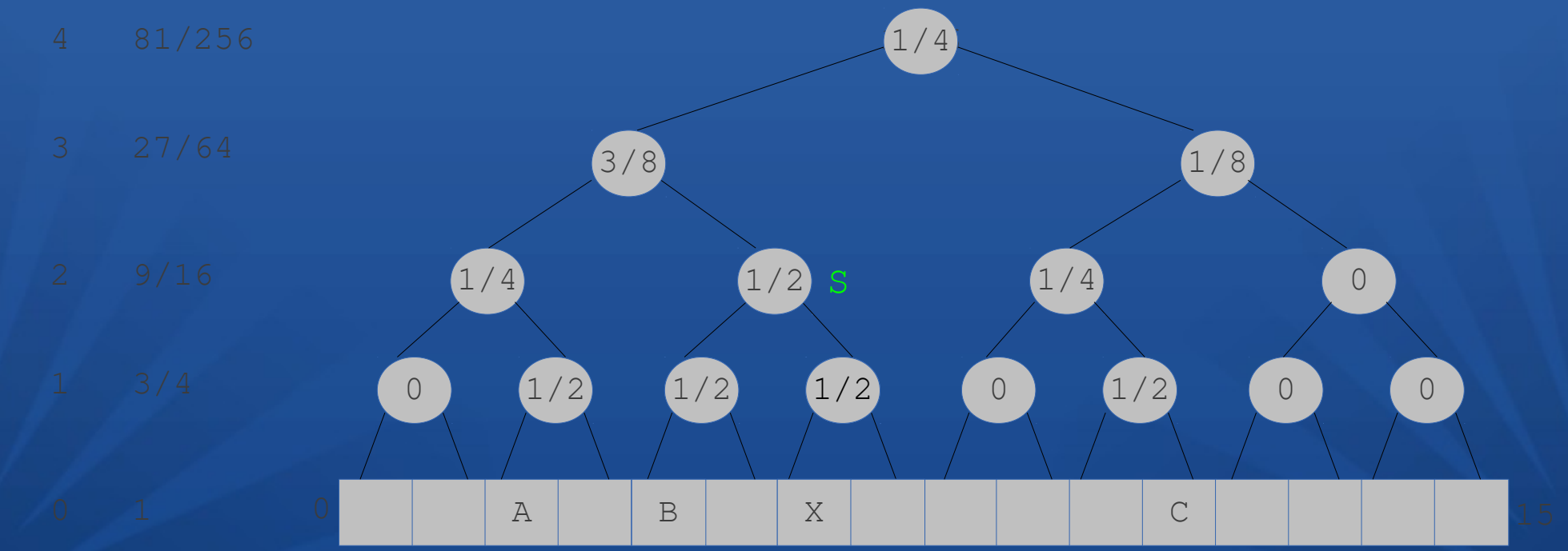
插入及重标号过程

- 假设待插入的元素为 X ，要插在 B 的后面。
- 先将 X 插入到和 B 相同的位置 ， 即 B 和 X 标号相同……
- —— 这……难道还可以有两个不同的元素标号相同？
- 这只是开始，后面会调整的囧……



插入及重标号过程

- 插入 X 之后，其所有祖先的密度都增加了，而这可能导致越界（对于 X 所在的叶结点来说，由于其此时的密度为 2 ，所以必然越界）。
- 找到 X 的祖先中，从下到上第一个不越界的结点 S 。
- 将子树 S 内的所有元素等距离重新标号。



M 的范围分析

- 显然，树的深度 M 不能太小，它必须满足在任意时刻，整棵树的根是不越界的。因此，若序列中最多有 N 个元素，则有

$$(2\varepsilon)^M \geq N$$

- 解得

$$2^M \geq N^{\log_{2\varepsilon} 2}$$

- 当 ε 取 3/4 时，标号范围约为 $N^{1.7095}$。当 ε 取 2/3 时，标号范围约为 $N^{2.4094}$，可见，一般情况下，标号范围在 N^2 左右。

时间复杂度分析

- 接下来就是时间复杂度分析了囧……显然主要的问题是，最坏情况下插入所引起的重新分配标号的平均次数是多少？
- 为了方便描述，以下设 $T(x)$ 为以 x 为根的子树，$|T(x)|$ 为以 x 为根的子树管辖范围中的占用标号个数（即元素个数）。
- 对树中的每个结点 x 定义势能函数 $\Delta(x)$：当 x 是叶结点时 $\Delta(x)=0$，否则 $\Delta(x)=\max\{||T(x.lc)|-|T(x.rc)||, 1\}-1$。
- 一棵子树的势能函数值为子树内所有结点的势能函数值之和，即

$$\Delta(T(x))=\sum_{y\in T(x)}\Delta(y)$$

- （其实这就是 Scapegoat Tree 的势能函数……）
- 由于重新分配标号后，所有元素的标号是均匀（等距离）分布的，所以对 $T(x)$ 重新分配标号后显然有 $\Delta(T(x))=0$。

时间复杂度分析

- 在上面的重新分配过程中， S 是不越界的，设 S 的层数为 i ，则有

$$|T(S)|\leq(2\varepsilon)^i$$

- 不妨设新结点插入到 S 的左子树里，则插入后 S 的左子结点越界，而 S 的右子结点必然不越界（否则 S 越界，矛盾）。进一步，根据此时 S 不越界，可以得出：

$$D(S.lc)-D(S)>\varepsilon^{i-1}-\varepsilon^i$$

- 又因为 $2D(S)=D(S.lc)+D(S.rc)$ ，并且此时显然有 $|T(S.lc)|\neq|T(S.rc)|$ 。易得：

$$\begin{aligned}\Delta(S)&=\|T(S.lc)|-|T(S.rc)\|\\&=2^{i-1}|D(S.lc)-D(S.rc)|\\&>2^{i-1}\cdot 2(\varepsilon^{i-1}-\varepsilon^i)\\&=(2\varepsilon)^i(\frac{1}{\varepsilon}-1)\end{aligned}$$

$$\Delta(S)>(2\varepsilon)^i(\frac{1}{\varepsilon}-1)$$

- 因此，在对 T(S) 重新分配标号之前，有

$$\geq|T(S)|(\frac{1}{\varepsilon}-1)$$

时间复杂度分析

- 在重新分配 $T(S)$ 内的标号过程中，更改了 $|T(S)|$ 个元素的标号。
- 分配完后，整棵树释放的势能至少为 $|T(S)|(1/\varepsilon-1)$ 。
- 而每次插入操作（不计重新分配标号），最多能使新结点所在位置的每个祖先的势能都增加 1，所以整棵树的势能增加了 $O(M)=O(\log N)$ 。
- 这样，可得每次插入后重新分配标号的平均次数为：

$$\frac{|T(S)|\cdot O(\log N)}{|T(S)|(\frac{1}{\varepsilon}-1)}=O(\log N)$$

更好的做法

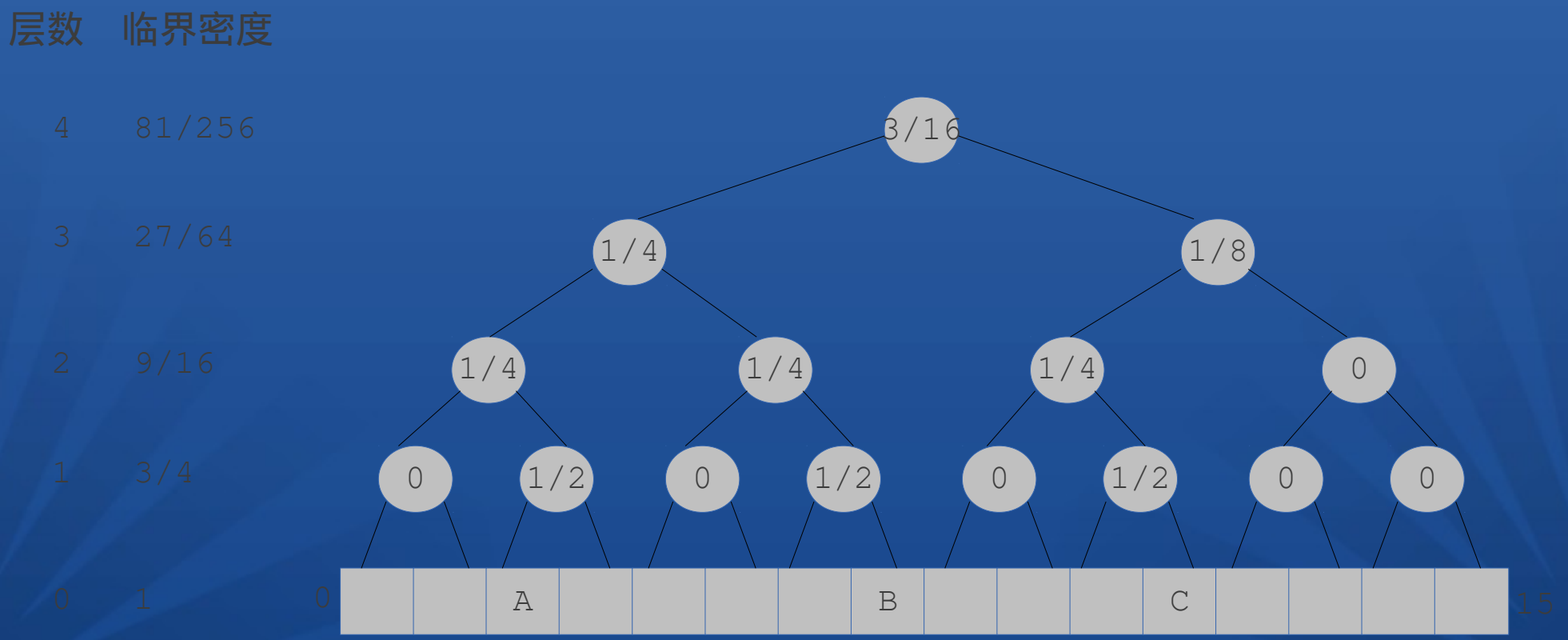
- 以上方法的均摊重新分配次数都是 O(logN) 的，但是，会不会存在均摊重新分配次数 O(1) 的做法？
- 很悲剧的是， P.F.Dietz 、 J.I.Seiferas 和 J.Zhang 在 1994 年的一篇论文中证明了，动态标号中，如果采用整数标号，且标号范围为 N 的多项式级别，则插入操作的均摊重新分配次数的下界为 logN 。具体的证明很复杂，这里由于时间有限就不说了囧……
- 但是，我们没必要采用整数标号！！实数标号也可以！！进一步，我们甚至木有必要用一个数字作为标号，可以用一个 pair 作为标号。
- （这也可以？其实只要这东西能比较大小就行）
- 使用 pair 标号，借助分块思想，可以很容易设计出一个均摊重新分配次数 O(1) 的做法。

更好的做法

- 设这个序列中的元素个数为 N，将这个序列分成 N/logN 块，每块 logN 个元素。每个元素的位置用一个 pair 标号 (first, second) 表示，其中 first 为它所在的块的链接，second 为它在块内的标号（为整数）。同时，每个块也有一个整数标号。
- 插入一个元素时，直接插到对应块里，并取其两个相邻元素块内标号 second 的算术平均数作为新元素的块内标号 second。
- 如果某个块的大小达到 2logN，则分成两个块。由于增加了一个新块，相当于在块序列中插入一个新元素，按照前面所说的方法对这个新块重新分配标号，同时修改原来的块中部分（不超过 logN 个）元素的块链接 first。
- 由于一个块的大小始终不会超过 2logN，所以 second 的范围只需要取到 $4^{\log N}$ 即 N^2，就可以使得块内插入不会重新分配标号。
- 时间复杂度分析：插入新元素后，如果木有增加新块，则本次插入的时间为 O(1)，否则（增加了新块），需要用 O(logN) 的时间重新分配块标号，并用 O(logN) 的时间更改元素的 first。然而，至少每 logN 次插入才会出现一次增加新块的操作，所以总的均摊时间复杂度是 O(1) 的。

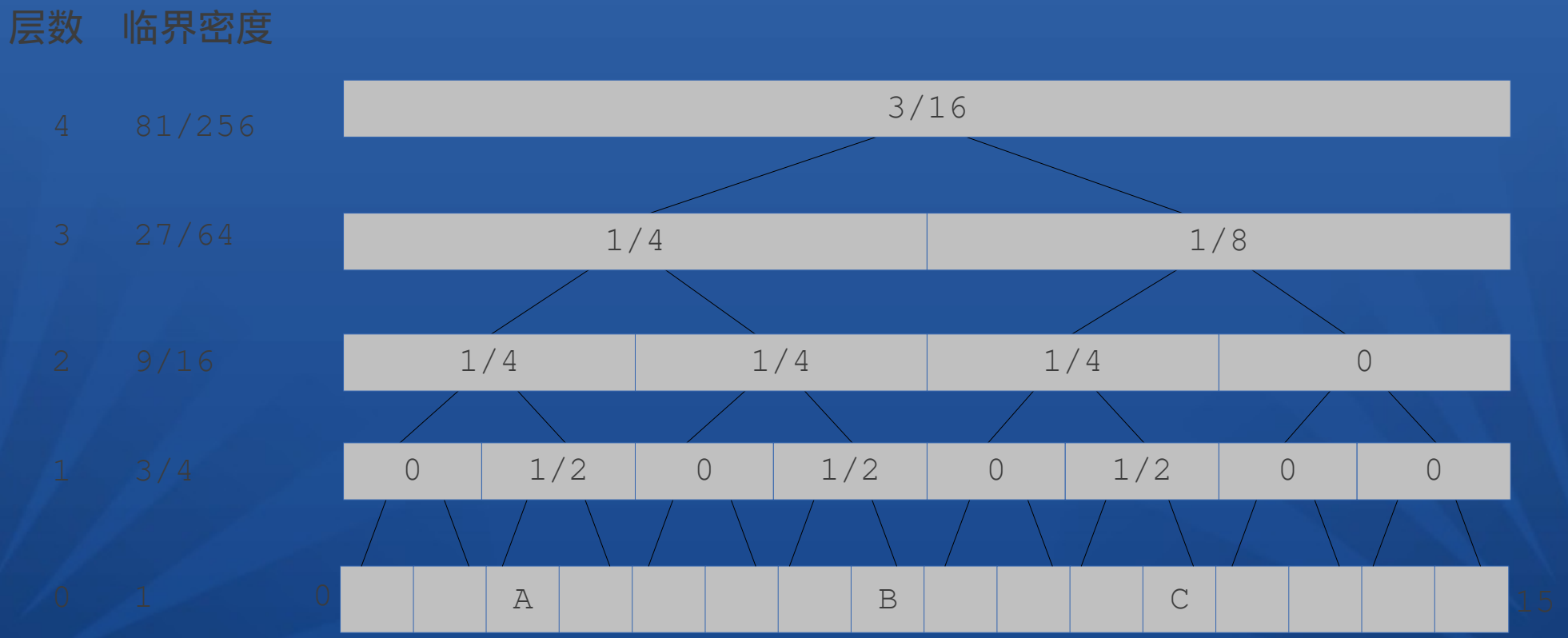
如何存储标号

- 最后还有一个问题，之前所说的这种动态标号方法，如何存储这些标号？
- 直接用数组或 `set` 存储是不行的，因为它不能快速得到树上一个结点管辖范围内的元素个数，进而得到它的密度。
- 我们再来审视一下这棵树：



如何存储标号

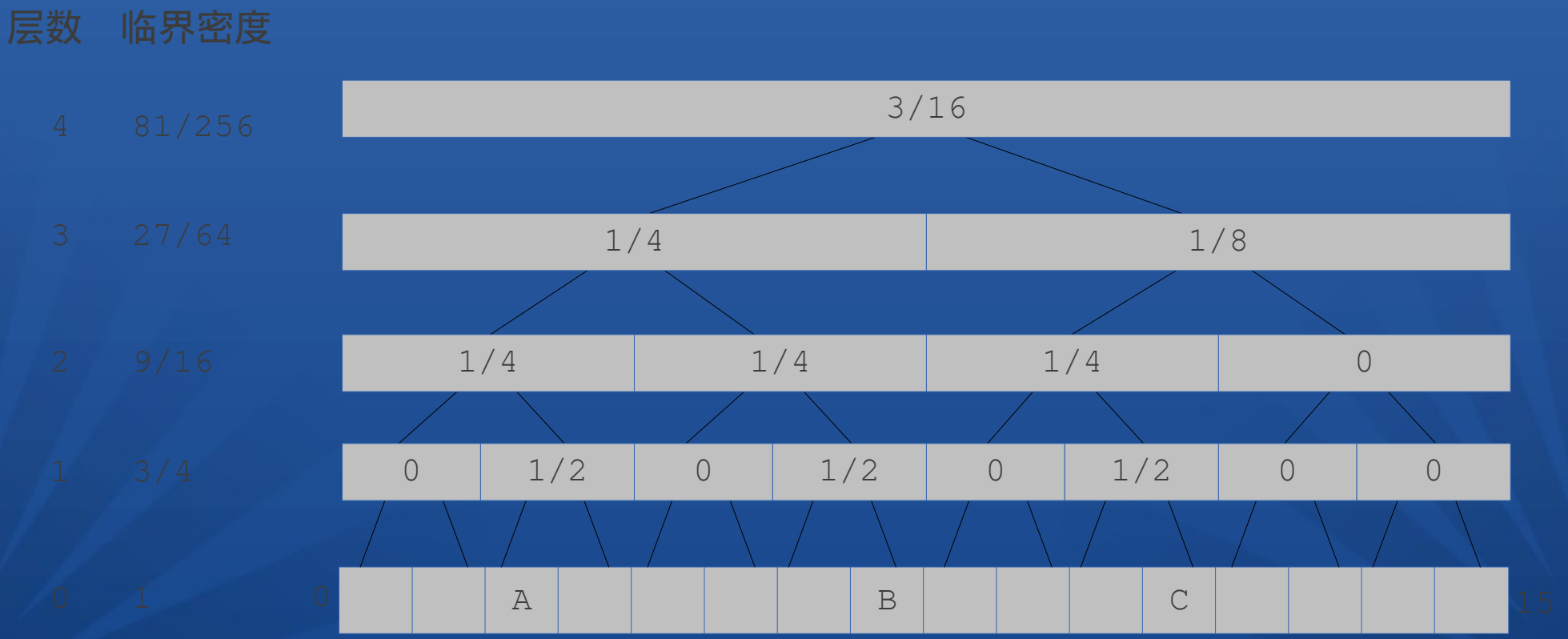
- 最后还有一个问题，之前所说的这种动态标号方法，如何存储这些标号？
- 直接用数组或 `set` 存储是不行的，因为它不能快速得到树上一个结点管辖范围内的元素个数，进而得到它的密度。
- 我们再来审视一下这棵树：



如何存储标号

- 线段树！！！
- 进一步，将其中所有密度为 0 的结点都去掉不看……

- 动态开结点的线段树！！！



如何存储标号

- 可以发现，这棵表示一段内密度的树就是一棵现成的线段树，而且可以动态开结点。
- 所以，我们就用动态开结点的线段树来存储标号，以及计算密度。
- 木有标记的动态开结点线段树是很好写的，至少比平衡树好写很多……

- 至此，我们终于完美解决了传说中的序列顺序维护问题……

静态（离线）标号

- 有了动态标号，自然也应该有静态标号……
- 静态标号就是已知各个元素之间的前后顺序，并且不再有新的元素插入，或者虽然有新的元素插入但插入位置可以预先知道（一般出现在可以离线的问题中，所以静态标号又叫离线标号）。这时，可以将所有元素的标号一次算出来，而不用重新分配标号。

- —— 那样直接排序然后用下标不就行了，还要标号么囧……
- 确实，静态标号很多时候就是直接用下标，但是……可能有无法直接排序的时候……
- —— 那是肿么回事？
- 后面自然就知道了囧……

标号法在实际问题中的应用

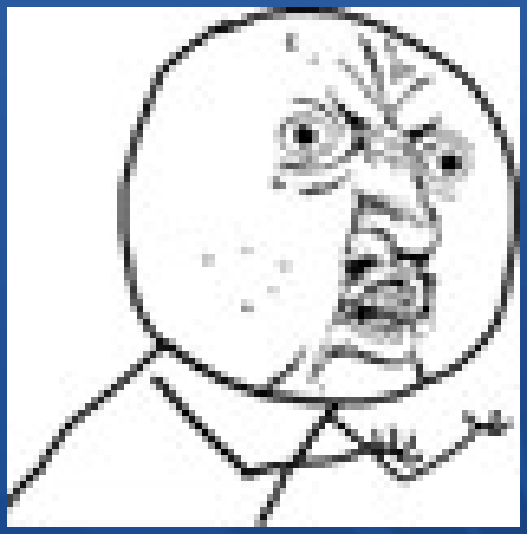
- —— 标号法有什么用呢囧……难道只能来解决 LOMP ？
- 当然不是，不然本沙茶也不敢来这里讲这个了……否则太侮辱神犇们的智商了囧……
- 各位神犇不要鄙视啊……

- 实际上，标号法在实际问题中是有大用的。
- 主要体现在以下几个方面：

- 数据结构
- 递推 /DP 中的字典序问题
- 复杂元素的有序化
- 动态离散，离线转在线

数据结构

- 显然，标号法最直接的应用就是在数据结构问题中了……
- 有一类这样的数据结构问题——
- 在无视插入删除操作的情况下，很容易解决（秒过的节奏……）
- 在有插入删除操作，但可以离线的情况下，也很容易解决（继续秒过……）
- 但是题目中有插入和删除操作，而且要求在线！！！！



数据结构

- 显然，标号法最直接的应用就是在数据结构问题中了……
- 有一类这样的数据结构问题——
- 在无视插入删除操作的情况下，很容易解决（秒过的节奏……）
- 在有插入删除操作，但可以离线的情况下，也很容易解决（继续秒过……）
- 但是题目中有插入和删除操作，而且要求在线！！！！
- 对于这类问题的常见解法展示——

题解之一——替罪羊套函数式线段树

题解之二——Treap套函数式线段树

题解之三——划分树崛起

题解之四——根号大军

【块状链表套线段树】

【朝鲜树套函数式线段树】

- 这些解法的共同特点为，需要使用很繁琐的数据结构。
- 从它们的代码量也可以看出来囧——

3065_seg_splay	3065	Time_Limit_Exceed	50508 kb	61223 ms	C++	6831 B
albus	3065	Time_Limit_Exceed	60352 kb	61011 ms	C++	6891 B
albus	3065	Time_Limit_Exceed	60352 kb	61003 ms	C++	6793 B
3065_seg_treap	3065	Accepted	60352 kb	43416 ms	C++	7093 B
3065_seg_splay	3065	Time_Limit_Exceed	395048 kb	61879 ms	C++	6755 B
3065_seg_splay	3065	Wrong_Answer	395044 kb	448 ms	C++	6714 B
3065_treap_seg	3065	Accepted	199052 kb	44860 ms	C++	8007 B
3065_scap_seg	3065	Accepted	179260 kb	31496 ms	C++	8608 B
3065_scap_seg	3065	Runtime_Error	179260 kb	204 ms	C++	8717 B

数据结构

- 显然，标号法最直接的应用就是在数据结构问题中了……
- 有一类这样的数据结构问题——
- 在无视插入删除操作的情况下，很容易解决（秒过的节奏……）
- 在有插入删除操作，但可以离线的情况下，也很容易解决（继续秒过……）
- 但是题目中有插入和删除操作，而且要求在线！！！！

题解之五——动态标号

3065_OMLL_seg	3065	Accepted	490628 kb	31796 ms	C++	4968 B

- 在这种问题中应用标号法（主要是动态标号）会取得很好的效果，省去了很多代码量。

ALOEXT

- 本沙茶至今在 BZOJ 上的唯一一道原创题……
- 有一个整数序列，所有数都在 0 ~ $2^{20}-1$ 之间，四种操作：
- （0）在某个位置插入一个新的整数（当然也在 0 ~ $2^{20}-1$ 之间）；
- （1）删除某个数；
- （2）修改某个数；
- （3）给出一个区间 [l，r]，设序列中第 l 个到第 r 个数中的次大值为 v，询问第 l 个到第 r 个数中与 v 作按位异或（xor）结果的最大值。
- 要求在线。
- 相信许多神犇已经用各种“ XX 树套平衡树”或者“平衡树套 XX 树”等的办法搞定了囧……
- 但有木有不用平衡树的办法呢？平衡树有时候很讨厌的样子……

ALOEXT

- 考虑用标号来维护这个序列中各个元素的位置顺序。
- 建立一棵动态开结点的标号线段树，内层再套一棵表示标号在此范围内的所有元素的值的线段树（同样是动态开结点的），值线段树的每个结点存储一个 sum 表示值在此范围内的元素个数.....
- 插入时，按照前面的方法重新分配标号，注意先将外层标号线段树中的 T(S) 整体销毁，得到新的标号后再重建 T(S)（得到每个结点的内层的值线段树的时候需要使用线段树的合并技巧，参见去年主席的讲稿）；
- 删除、修改直接搞就行了；
- 询问时，先通过整体二分在 $O(\log^2 N)$ 时间内找到次大值，然后可以发现，内层的值线段树其实具备了 Trie 的功能，所以可以直接在里面找到询问的答案，同样使用整体二分。

ALOEXT

- —— 这样做好像仍然很繁琐的样子……有木有代码量更小的做法？
- “我们要充分发挥人类智慧，探索、测试、改进解决方案的能力。”—— lemon_workshop, IOI2013 #3。
- 我们还可以将内层的值线段树换成 map，它的每个元素 <first, second> 表示该结点值线段树中的编号为 first 的结点目前的 sum 为 second。
- 合并的时候扫一遍就行了囧……
- 只是这种方法由于 map 自身的速度问题，很可能会 T 掉。

递推 /DP 中的字典序问题

- 在某些递推 /DP 问题中，需要输出方案……
- 如果有多个可行或最优方案，有的题目要求输出字典序（或者按照其它有阶段性的序）最小（或最大）的那个（或前若干个）。
- 对于这种问题，一般有两种方法：
- —— 预存具体方案；
- —— 找到决策之间本身的决定字典序的顺序关系；
- 比较囧的是，有时候决策本身木有什么特点，而预存具体方案在时间或空间上无法承受。
- 这时，可以将可行解之间的字典序（或其它顺序）转化为标号……

Undecodable Codes

- Source： ACM World Final 2002
- 有 N 个已知的 01 串片段，求出最短的 01 串使得它可以用这些片段以至少两种不同方式拼成（每个片段可以使用任意次，当然也可以是 0 次）。
- 如果有多解，输出字典序最小的。
- 1<=N<=64， 1<= 片段长度 len<=64 。

（原题为 20，这里加强）

（其实把 01 串换成任意字符串也可以囧……）

Undecodable Codes

- 先无视字典序这个限制。
- 设 `F[i1][j1][i2][j2]` 表示目前两种方式分别到达第 `i1` 个片段的第 `j1` 个字符、第 `i2` 个片段的第 `j2` 个字符，经过的最短长度。
- 若两种方式同时到达片段末尾，则已找到最优解。
- 若一种方式到达末尾，另一种未到达，枚举一个新的片段替换上它。
- 若两种都没到末尾，继续推到下一个字符。
- `BFS` 即可实现。
- 时间复杂度 $O(N^2len^2)$ 。

Undecodable Codes

- 有了字典序最小的限制肿么办？
- 显然不能预存具体方案，这样在时间、空间上都无法承受……
- BFS 具有自然的阶段性！！它总是把第 i 层（值为 i ）的状态全部扩展出来后，才会开始扩展第（i+1）层（值为（i+1））的状态。
- 因此，我们可以每扩展出一层状态，就将这层状态按照字典序递增排序，然后在扩展下一层状态时，遇到前趋状态字典序更小的解，显然更优。
- 问题是怎么排序？标号再一次派上用场了，只不过这次用静态标号。
- 扩展完一层状态后，就将其按照（前趋状态标号，本状态最后一个字符）这个 pair 递增排序，排序后的下标作为其标号。
- 注意，字典序完全相等的，也要使用相等的标号。

Ideal Path(EXT)

- Source： NEERC 2010，有加强。
- 给出一个图，每条边有一个权值和一个颜色（用数字表示）。
- 要求找出从一个给定的起点 S 到一个给定的终点 T 的最短路径，如果有多条，找到那条经过的边数最少的，如果还有多条，找到从 S 到 T 经过的所有边颜色字典序最小的。
- 1<= 点数 N<=100000， 1<= 边数 M<=200000，颜色用 1 ~ 10^9 的正整数表示。

Ideal Path(EXT)

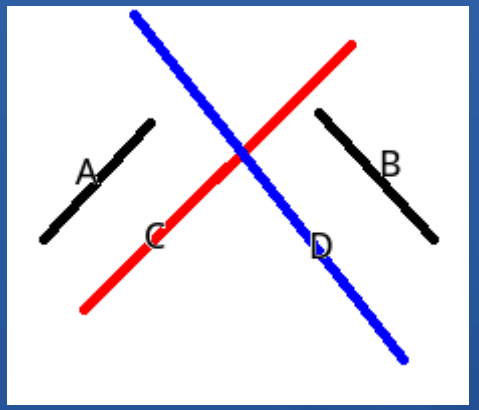
- 本题的字典序……还是不好直接搞啊囧……继续上标号……
- 然而最可怕之处在于……求最短路径的算法木有明显的阶段性，因此上面那题的“分阶段排序”显然不适用了。
- 静态标号不适用了，用动态标号！！
- SPFA。对每个点，按照其（最短路径长度，经过的边数，前趋状态的标号，上一条边的颜色）这个四元组进行递增排序，同时用标号维护顺序……
- 用一个 map 来存储这些四元组和标号。
- 每当一个点被更新时，就在 map 中删去其原有信息，在标号线段树中删去其原有标号，然后根据新的四元组找到它的前趋和后继，再将这个新的四元组插入 map，同时在标号线段树中重新分配标号……
- 时间复杂度 O(MlogM)。

平面线段排序

- 下面来看一个喜闻乐见的问题：平面线段排序。
- 给出在二维平面上的 N 条线段，所有线段端点的 X 坐标互不相同，且任意两条线段不内交（即任意两条线段的交点只可能是它们的端点）。
- 现在要求出这 N 条线段的一个顺序，使得对于任意两条线段 i 和 j，如果存在实数 x0，使得在 x=x0 处 i 在 j 的下方（ Y 坐标比 j 小），则 i 要排在 j 的前面 。
- 1<=N<=100000 。

平面线段排序

- 对于这种问题，首先容易想到的是定义一个 < 关系，能直接判断两条线段的前后关系。
- 如果两条线段在 X 坐标跨度上有交集，显然可以判断出来。
- 如果没交集？
- 或许……或许……或许也可以判断出来吧囧……
- 可是残酷的事实告诉我们……
- 如果 C 存在，则 B<C ， C<A ，得出 B<A……
- 如果 D 存在，则 A<D ， D<B ，得出 A<B……



- 也就是， A 和 B 两条线段的前后关系与其它线段相关，不可直接比较，故所谓的 < 关系是构造不出来的囧……
- 幸运的是 C 和 D 不可能都存在，因为题目说了任意两条线段不内交。事实上，满足题目要求的顺序是一定存在的囧……

平面线段排序

- 扫描线？
- 题目中说了所有线段端点的 X 坐标互不相同，因此木有与 Y 轴平行的线段，可以按照 X 坐标扫描。
- 维护一个 set ，存储目前扫描线经过的线段。
- 每插入一条线段，就在 set 中找到其前趋和后继。
- 扫描完后，就得到了 O(N) 个前趋和后继关系。
- 再将它们转化为标号即可得到顺序（因为可以离线，可用静态标号）。

- —— 求出前趋和后继，直接建出图，拓扑一下不就行了么囧……
- 好像是可以……
- 但是……这题还可以加强一下……

平面线段排序（动态版）

- 仍然是二维平面上的线段。
- 仍然木有两个端点的 X 坐标相同。
- 仍然木有两条线段内交。
- 但是，可以在过程中实时增加新的线段！！
- 也可以实时删除线段！！
- 要求实时维护顺序。

- 先降低一点难度，一开始告诉你一些点（其 X 坐标互不相同），然后所有出现的线段的端点都在这些点中。

平面线段排序（动态版）

- 由于一开始已知端点，先将这些端点的 X 坐标离散化。
- 注意到两个区间 (l1, r1) 和 (l2, r2) 相交当且仅当 l2<r1 && r2>l1 。
- 因此，可以用动态标号维护这些线段的顺序。建立动态开结点的标号线段树，每个结点中套两棵动态开结点的值线段树，分别存储标号在这个区间内的所有线段的左、右端点 X 坐标的分布情况。
- 这样对于一条已知的线段，设其 X 坐标跨度为 (l1, r1)，则外层标号线段树的某个结点范围内，和这条线段相交的线段条数为（“结点内线段总条数”-“左端点 >=r1 的线段条数”-“右端点 <=l1 的线段条数”），根据这个值是否为 0，可以对标号范围进行二分。
- 每次插入一条线段的时候，其前趋和后继就是和它的 X 坐标跨度有交集的线段标号的前趋和后继，这个可以用上述办法在 $O(\log^2 N)$ 时间内找到。
- 找到前趋和后继之后，先将新线段插入标号线段树，重新分配标号，再将新线段引起的左右端点分布的变化，在内层值线段树上处理。

平面线段排序（动态版 EXT）

- 其它条件与动态版相同，只是去掉为了降低难度的“预先给出所有端点”的条件。
- 由于连端点的 X 坐标也不知道了，所以在插入线段的过程中可能会插入新的端点（及其 X 坐标），一种很自然的思路是，对端点序列（对应内层线段树）也用动态标号维护。
- 但是直接这样做会出现很可怕的问题.....在重新分配标号的时候，如果某个端点关联的线段过多，而它又被重新分配了标号，则它关联的这些线段都要修改标号，时间上不能承受。
- 所以，应该将每条线段的两个端点都视为“独立端点”，即如果两条不同的线段有公共端点，这个公共端点在两条线段中仍被视为不同的独立端点。用动态标号维护独立端点序列，修改一个独立端点的标号时只要同时处理它属于的那条线段即可，和别的线段无关。
- 这样会导致端点序列（对应内层线段树）中不同的元素的 x 坐标相同，因此在找与（l1, r1）相交的线段时，也要将“左端点 >=r1” 和”右端点 <=l1” 改一下。

总结

- 标号的本质：顺序的数量化表示。
- 在数据结构题中，标号法可以来解决一系列静态（无插入删除）较容易解决，动态（有插入删除）较难解决的题目。
- 利用标号，可以将难以排序的量变得容易排序，将难以比较的量变得容易比较，因此标号法可以用来解决一些有关字典序的问题。
- 此外，动态标号还可以实现动态离散化或其它一些离线转在线的问题。

参考文献

- [0] 陈立杰，《重量平衡树和后缀平衡树在信息学奥赛中的应用》，2013 年候选队论文。
- [1] List Order Maintenance && Monotonic List Labeling Density Maintenance.
- [2] P.Dietz and D.Sleator. Two algorithms for maintaining order in a list. In STOC, 1987.
- [3] P.F.Dietz, J.I.Seiferas, and J.Zhang. A tight lower bound for on-line monotonic list labeling. In SWAT, 1994.
- [4] D.Willard. A density control algorithm for doing insertions and deletions in a sequentially ordered file in good worst-case time. *Information and Computation*, 97(2):150-204, 1992.
- [5] 吕凯风，《对无旋转操作的平衡树的一些探究》。
- [6] 黄嘉泰，《线段树的合并》，2013 年冬令营营员交流讲稿。

Thanks

- 感谢 CCF 给我提供了这次培训和交流的机会。
- 感谢主席（黄嘉泰）、 vfleaking（吕凯风）、 taorunz（陶润洲）等神犇在我完成有关标号法的研究的过程中对我的帮助。
- 感谢董宏华神犇帮助审稿。
- 感谢在场的各位神犇不鄙视我这样的沙茶，听到了这里……

Time for further queries...