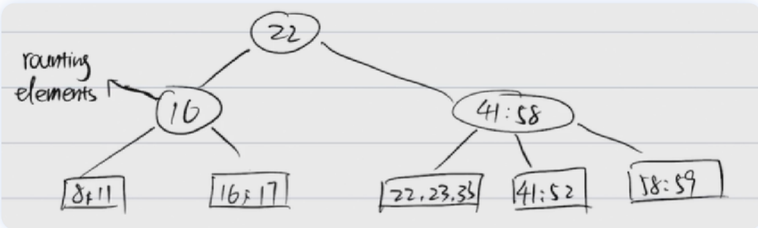
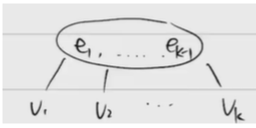
B+ Tree

定义



B+ tree of order B

1. leaves at the same level
2. data elements stored in leaves （其他node是导航的）
3. 导航取值（routing elements）的确定：  e_i = min element stored in leaves of $T_{v_{i+1}}$ 且 $e_i >$ every element om leaves of T_{v_i}
4. $\lceil B/2 \rceil$ ≤ fanout（children的数量） of an internal node ≤ B，所以三阶的B+树又叫2-3树，不过root的fanout的限制是[2,B]
5. $\lceil B/2 \rceil$ ≤ # elements in a leaf ≤ B, if the leaf is the root, 1 ≤ #elements ≤ B

性质

定义N: # data elements (Assume $N > B$)，则 # leaves ≤ $N/\lceil B/2 \rceil$

space： 总节点数不会超过叶子数的两倍，且每个节点最多B个信息，故space = $2N/\lceil B/2 \rceil \cdot B = 4N$

导航取值会重复吗？ - 不会，考虑一个node的导航取值，使用排除法，它既不会出现在节点上面，也不会出现在子树里面，更不会出现在别的子树，因此只出现一次

height ≤ $\log_{\lceil B/2 \rceil} N/\lceil B/2 \rceil + 1 = \log_{\lceil B/2 \rceil} N = \log_2 N/(\log_2 B - 1) = O(\log_2 N/\log_2 B) = O(\log_B N)$ *每次向下走一层node数量以大于等于$\lceil B/2 \rceil$倍增*

操作

Findkey

步骤

和导航取值比较，比它或相等大则向右边的导航取值找，比它小则向左子树找

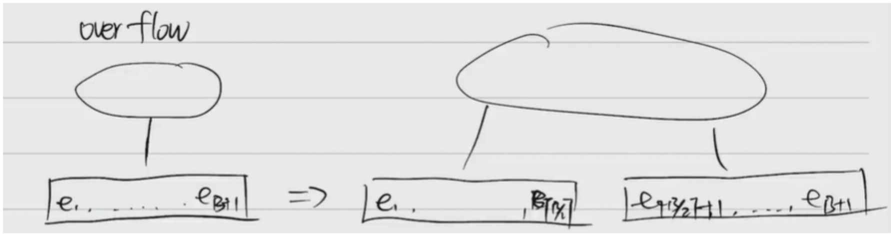
时间复杂度

单次查找（找到导航取值左小右大的临界位置 - 二分查找）的时间复杂度为$\log B$，总共时间复杂度为$O(\log_B N) \cdot O(\log B) = O(\log N)$

Insertion

步骤

寻找位置的方式和findkey一致，但是当一个leaf容纳的data超过B的时候会产生overflow，此时我们把一个叶子节点均分成两个：



，此时如果parent发生overflow了那么继续均分

时间复杂度

- 找位置 - $O(\log_B N) \cdot O(\log B)$
- 处理overflow均分 - $O(\log_B N) \cdot O(B)$（使用数组实现），但是可以降到$O(\log_B N) \cdot O(\log B)$

对内部的每个节点再用一个2-3tree就可以降到$O(\log B)$

- 总时间复杂度 - 二者相加 = $O(\log N)$

Deletion

步骤

先找到要删除的数的位置，然后删除，可能会出现节点信息数小于下限（$\lceil B/2 \rceil$）的问题(underflow)，处理方法如下：

- If some sibling has ≥ $\lceil B/2 \rceil + 1$ children, take one from it
- else merge （在这种情况下合并完之后不会超标的）

最后需要update routing elements

时间复杂度

$O(\log N)$

优势

当data被放在磁盘里的时候，B+树可以减少读写磁盘的次数，因为读取磁盘的时候是以block为单位读取的，通过增大B，可以降低I/O的次数$O(\log_B N)$