外部查找与排序

朱睿

June 13, 2019

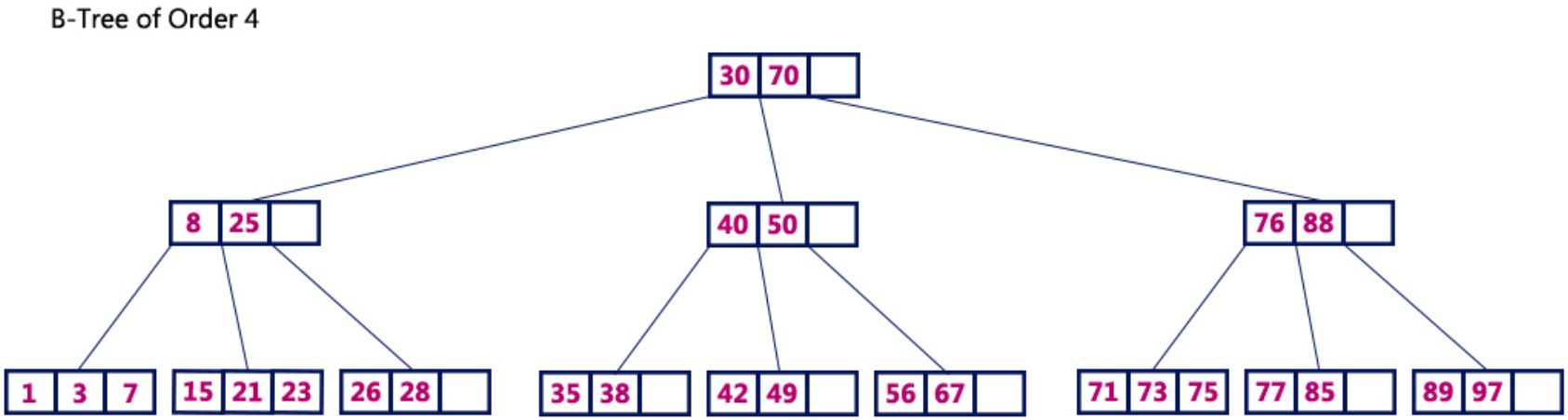
大纲

1. B 树
   - 定义
   - 算法
2. B+ 树
   - 定义
   - 操作
3. 外排序
   - 置换选择
   - 多阶段归并

B 树的定义

一个 m 阶的 B 树是一个有以下属性的树：

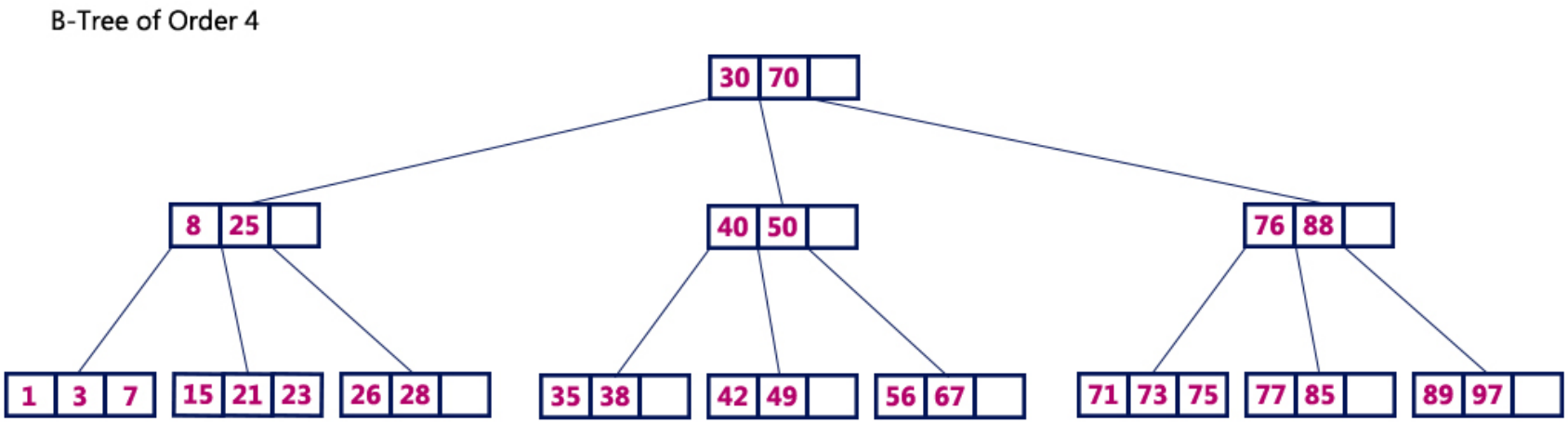
- 每一个节点最多有 m 个子节点
- 每一个非叶子节点（除根节点）最少有 $\lceil m/2 \rceil$ 个子节点
- 如果根节点不是叶子节点，那么它至少有两个子节点
- 有 k 个子节点的非叶子节点拥有 $k-1$ 个键
- 所有的叶子节点都在同一层



Source: *http://btechsmartclass.com/data_structures/b-trees.html*

查找

与二叉查找树的查找类似



Source: *http://btechsmartclass.com/data_structures/b-trees.html*

插入

构造一个三阶 B 树

insert(1)

Since '1' is the first element into the tree that is inserted into a new node. It acts as the root node.

Source: *http://btechsmartclass.com/data_structures/b-trees.html*

插入

构造一个三阶 B 树

insert(2)

Element '2' is added to existing leaf node. Here, we have only one node and that node acts as root and also leaf. This leaf node has an empty position. So, new element (2) can be inserted at that empty position.

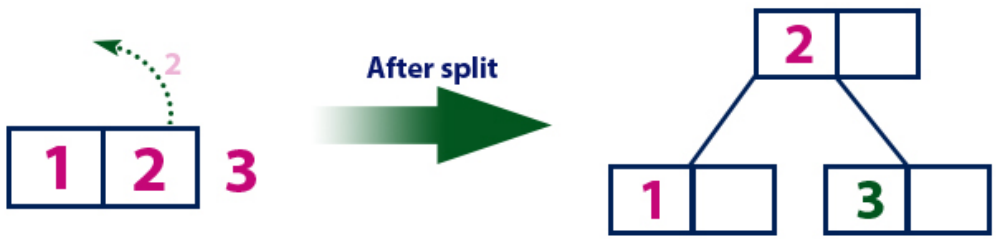
Source: *http://btechsmartclass.com/data_structures/b-trees.html*

插入

构造一个三阶 B 树

insert(3)

Element '3' is added to existing leaf node. Here, we have only one node and that node acts as root and also leaf. This leaf node doesn't has an empty position. So, we split that node by sending middle value (2) to its parent node. But here, this node doesn't has parent. So, this middle value becomes a new root node for the tree.



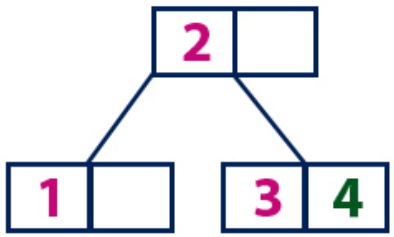
Source: *http://btechsmartclass.com/data_structures/b-trees.html*

插入

构造一个三阶 B 树

insert(4)

Element '4' is larger than root node '2' and it is not a leaf node. So, we move to the right of '2'. We reach to a leaf node with value '3' and it has an empty position. So, new element (4) can be inserted at that empty position.



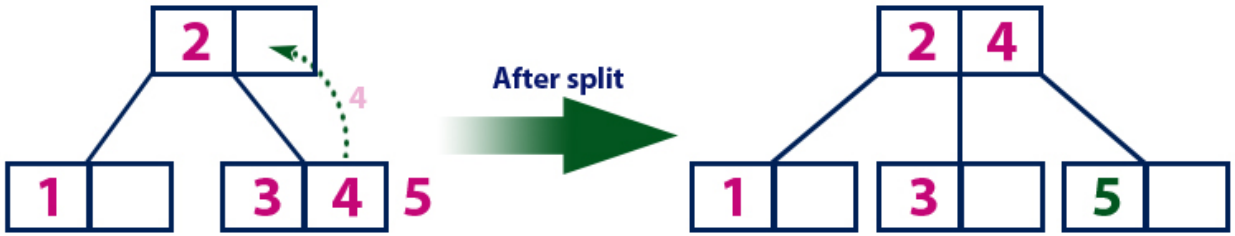
Source: *http://btechsmartclass.com/data_structures/b-trees.html*

插入

构造一个三阶 B 树

insert(5)

Element '5' is larger than root node '2' and it is not a leaf node. So, we move to the right of '2'. We reach to a leaf node and it is already full. So, we split that node by sending middle value (4) to its parent node (2). There is an empty position in its parent node. So, value '4' is added to node with value '2' and new element '5' added as new leaf node.



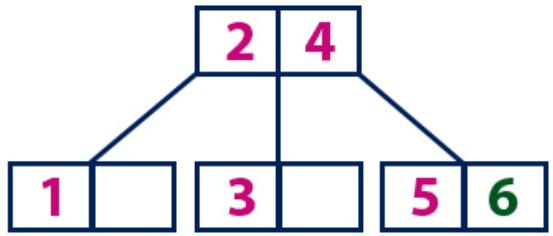
Source: *http://btechsmartclass.com/data_structures/b-trees.html*

插入

构造一个三阶 B 树

insert(6)

Element '6' is larger than root node '2' & '4' and it is not a leaf node. So, we move to the right of '4'. We reach to a leaf node with value '5' and it has an empty position. So, new element (6) can be inserted at that empty position.



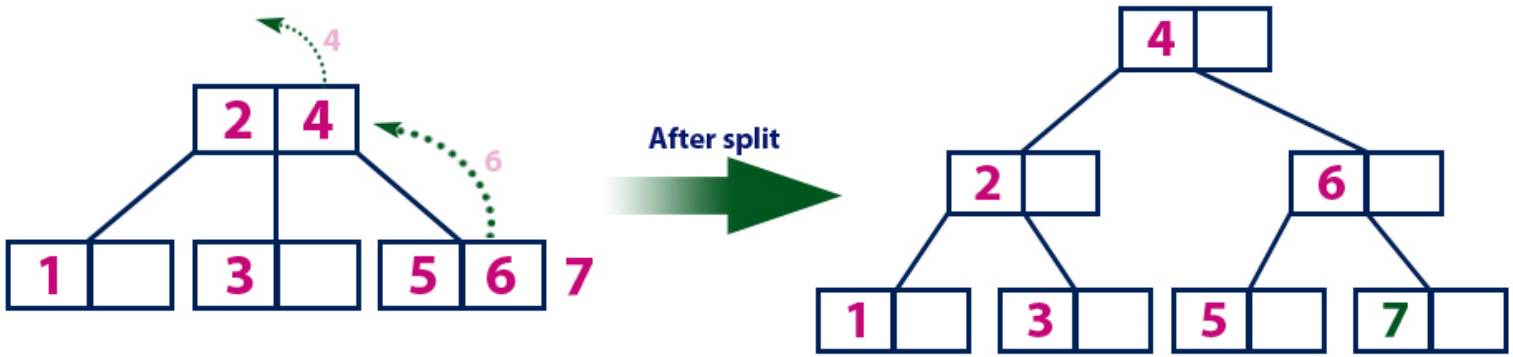
Source: *http://btechsmartclass.com/data_structures/b-trees.html*

插入

构造一个三阶 B 树

insert(7)

Element '7' is larger than root node '2' & '4' and it is not a leaf node. So, we move to the right of '4'. We reach to a leaf node and it is already full. So, we split that node by sending middle value (6) to its parent node (2&4). But the parent (2&4) is also full. So, again we split the node (2&4) by sending middle value '4' to its parent but this node doesn't have parent. So, the element '4' becomes new root node for the tree.



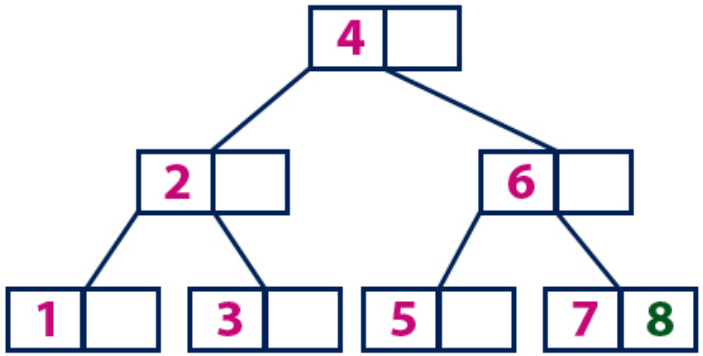
Source: *http://btechsmartclass.com/data_structures/b-trees.html*

插入

构造一个三阶 B 树

insert(8)

Element '8' is larger than root node '4' and it is not a leaf node. So, we move to the right of '4'. We reach to a node with value '6'. '8' is larger than '6' and it is also not a leaf node. So, we move to the right of '6'. We reach to a leaf node (7) and it has an empty position. So, new element (8) can be inserted at that empty position.



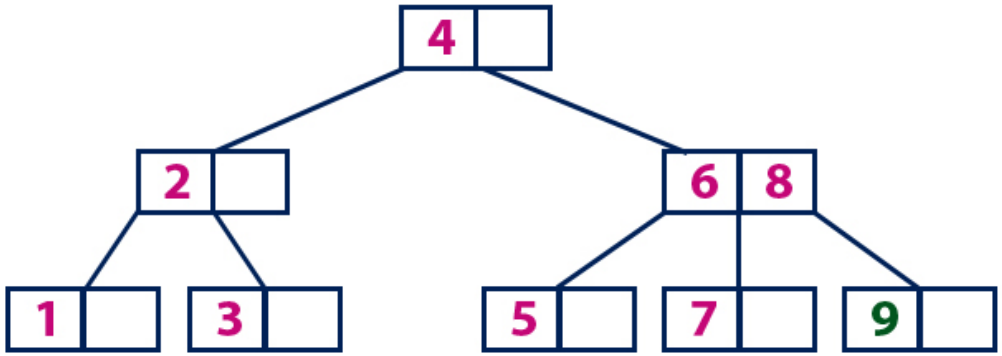
Source: *http://btechsmartclass.com/data_structures/b-trees.html*

插入

构造一个三阶 B 树

insert(9)

Element '9' is larger than root node '4' and it is not a leaf node. So, we move to the right of '4'. We reach to a node with value '6'. '9' is larger than '6' and it is also not a leaf node. So, we move to the right of '6'. We reach to a leaf node (7 & 8). This leaf node is already full. So, we split this node by sending middle value (8) to its parent node. The parent node (6) has an empty position. So, '8' is added at that position. And new element is added as a new leaf node.



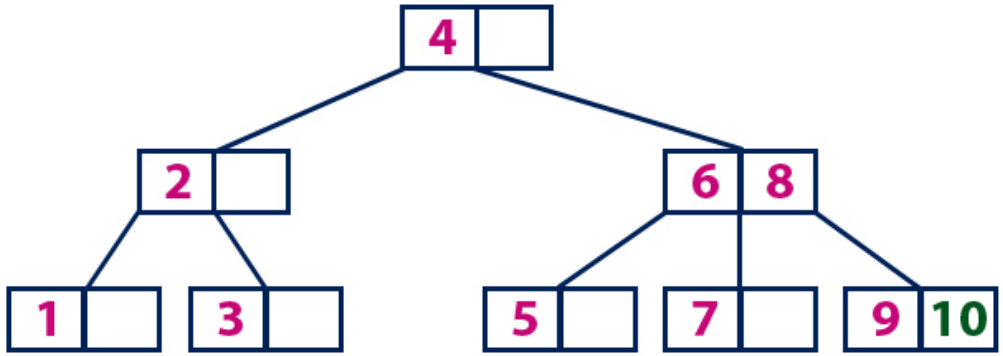
Source: *http://btechsmartclass.com/data_structures/b-trees.html*

插入

构造一个三阶 B 树

insert(10)

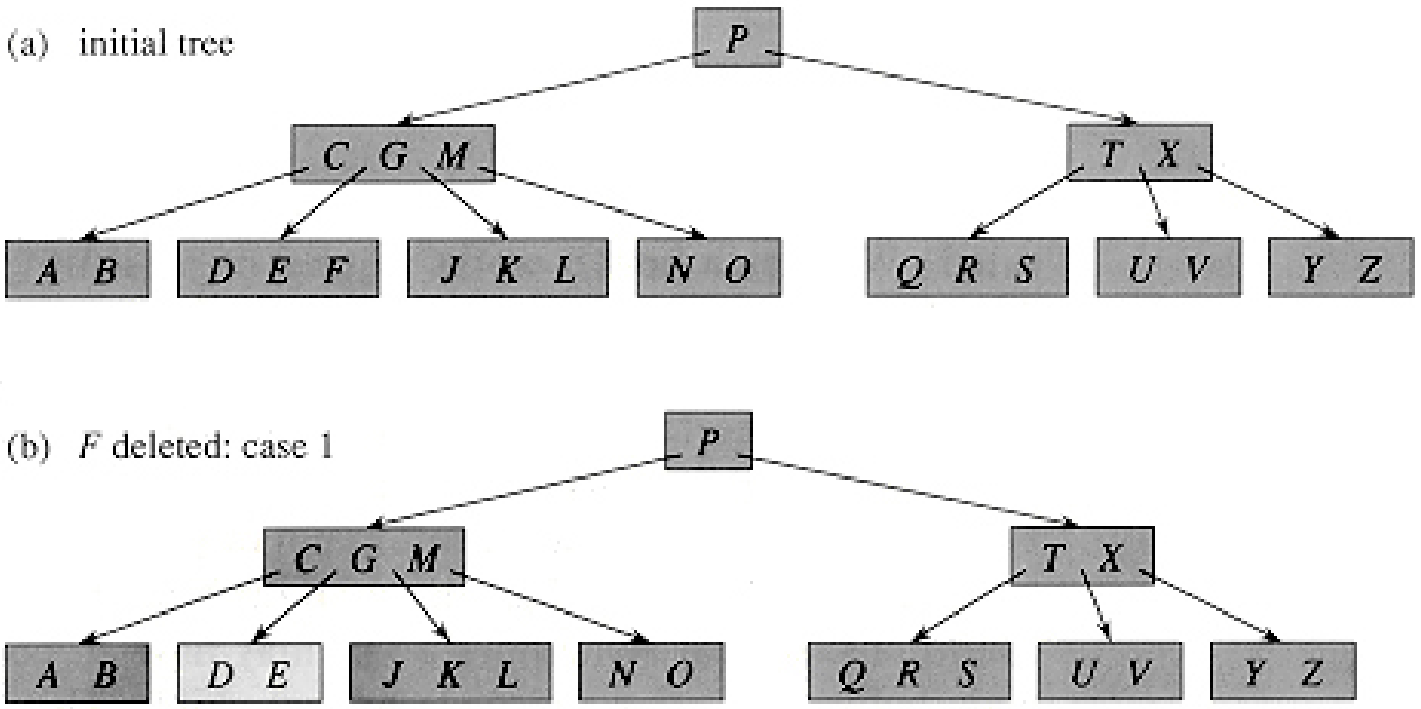
Element '10' is larger than root node '4' and it is not a leaf node. So, we move to the right of '4'. We reach to a node with values '6 & 8 '. '10' is larger than '6 & 8' and it is also not a leaf node. So, we move to the right of '8'. We reach to a leaf node (9). This leaf node has an empty position. So, new element '10' is added at that empty position.



Source: *http://btechsmartclass.com/data_structures/b-trees.html*

删除

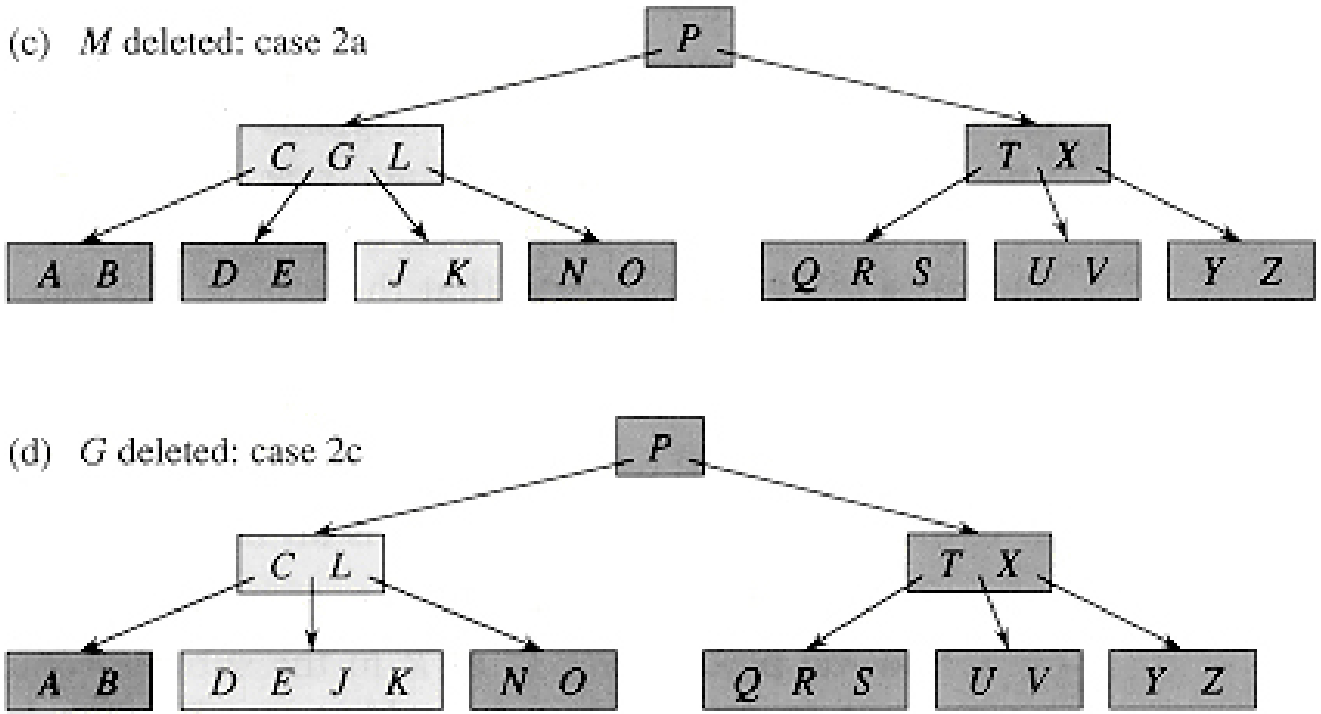
删除一个四阶 B 树



Source: *http://staff.ustc.edu.cn/ csli/graduate/algorithms/book6/chap19.htm*

删除

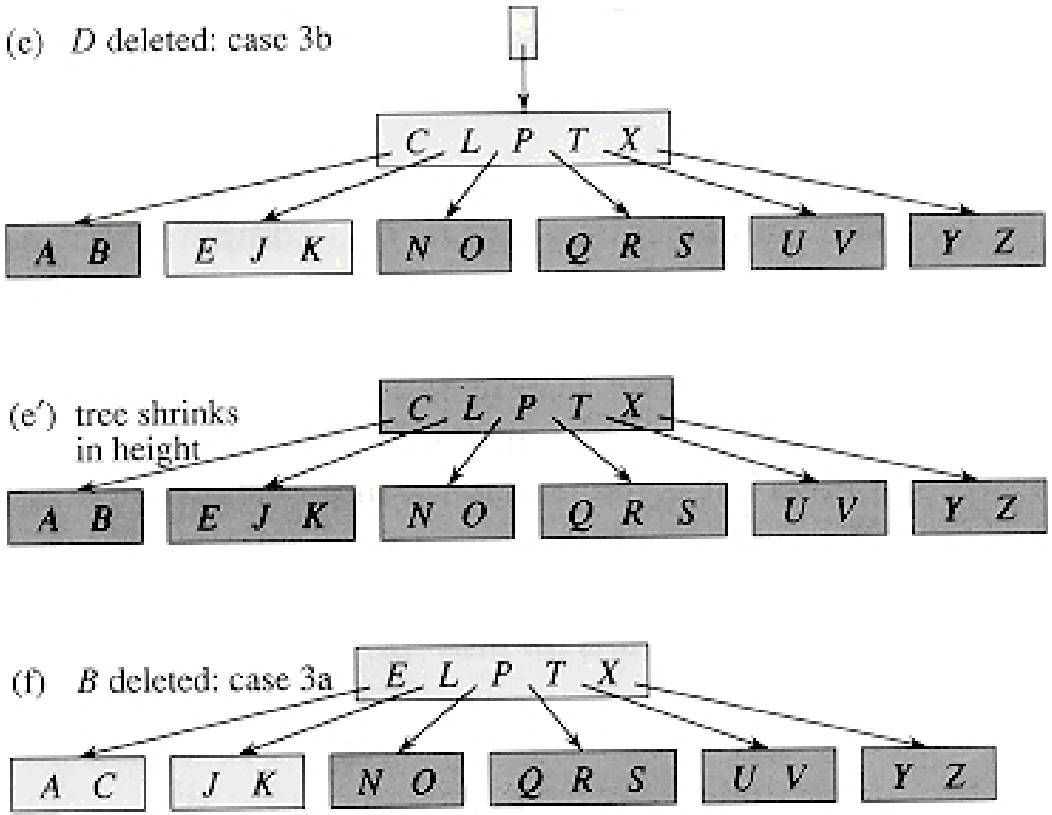
删除一个四阶 B 树



Source: *http://staff.ustc.edu.cn/ csli/graduate/algorithms/book6/chap19.htm*

删除

删除一个四阶 B 树



Source: *http://staff.ustc.edu.cn/ csli/graduate/algorithms/book6/chap19.htm*

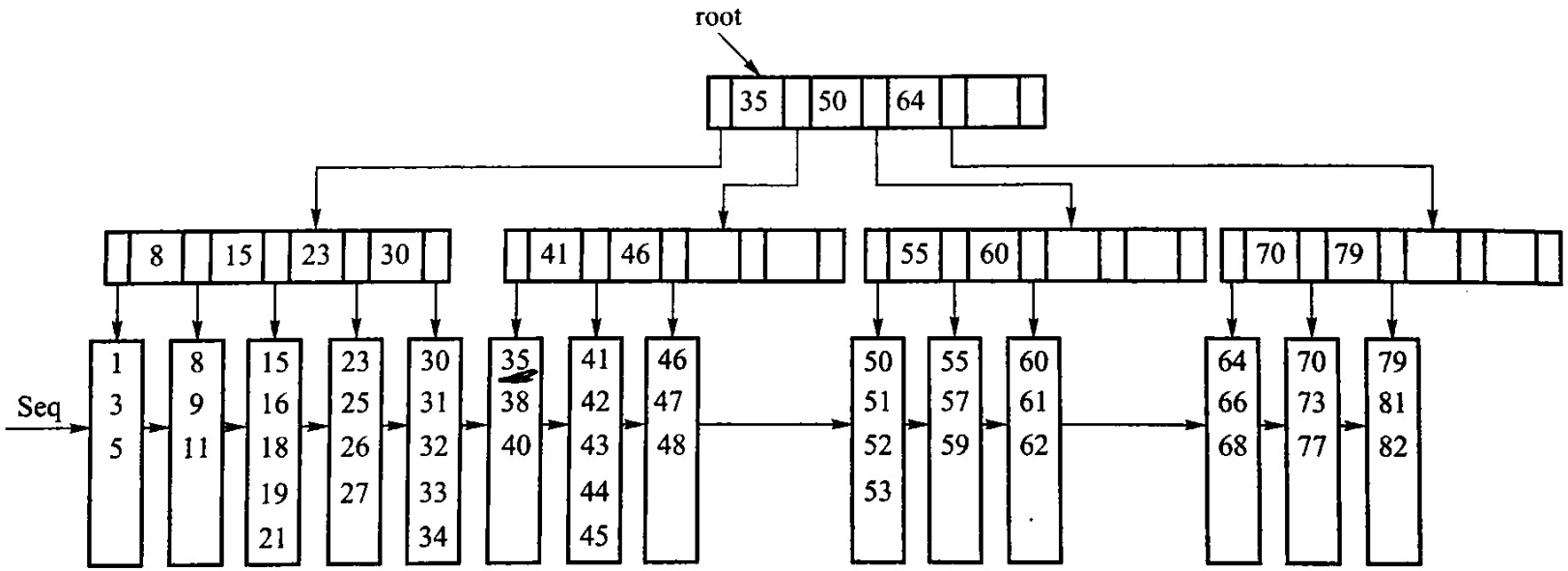
B+ 树的定义

一个 m 阶的 B+ 树是一个有以下属性的树：

- 每一个节点最多有 m 个子节点
- 每一个非叶子节点（除根节点）最少有 $\lceil m/2 \rceil$ 个子节点
- 如果根节点不是叶子节点，那么它至少有两个子节点
- 有 k 个子节点的非叶子节点拥有 $k-1$ 个键，第 i 键代表第 $i+1$ 子树中键的最小值
- 叶节点中的孩子指针指向存储记录的数据块的地址
- 每个数据块记录数范围为 $[\lceil l/2 \rceil, l]$
- 所有的叶子节点都在同一层且按序练成单链表

B+ 树的定义

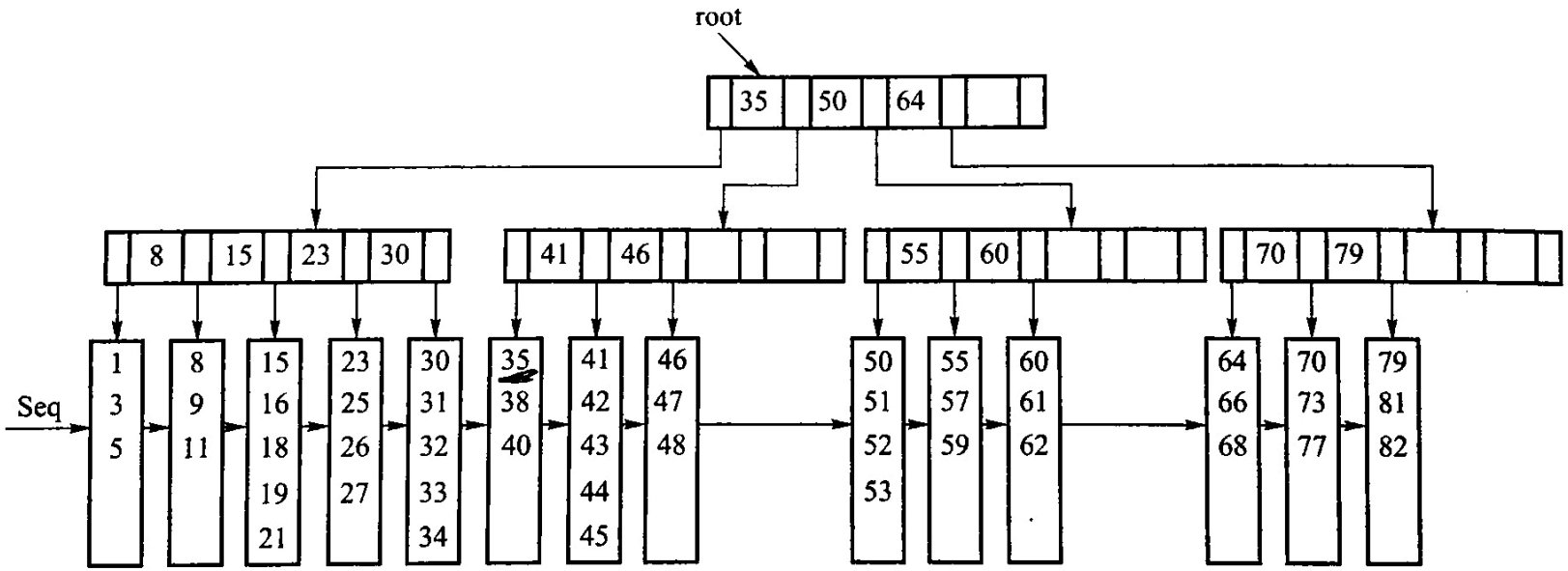
一颗 5 阶的 B+ 树



Source: 翁惠玉，俞勇《数据结构：思想与实现》

查找

与二叉查找树的查找类似



Source: 翁惠玉，俞勇《数据结构：思想与实现》

插入、删除

我突然找到了一个贼牛逼的网站，咱们去看动图：
https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html

置换选择

- 内存中使用优先级队列排序
- 产生若干已排序片段（然后再多阶段归并）
- 大小为 p 的内存平均能生成长度为 $2p$ 的已排序片段
- *E.g.* 一个例子：内存大小为 3，排序 {1, 4, 10, 2, 0, 5, 7, 3, 6, 9, 12}

a[0]	a[1]	a[2]	Out	In
1	4	10	1	2

置换选择

- 内存中使用优先级队列排序
- 产生若干已排序片段（然后再多阶段归并）
- 大小为 p 的内存平均能生成长度为 $2p$ 的已排序片段
- *E.g.* 一个例子：内存大小为 3，排序 {1, 4, 10, 2, 0, 5, 7, 3, 6, 9, 12}

a[0]	a[1]	a[2]	Out	In
1	4	10	1	2
2	4	10	2	0

置换选择

- 内存中使用优先级队列排序
- 产生若干已排序片段（然后再多阶段归并）
- 大小为 p 的内存平均能生成长度为 $2p$ 的已排序片段
- *E.g.* 一个例子：内存大小为 3，排序 {1, 4, 10, 2, 0, 5, 7, 3, 6, 9, 12}

a[0]	a[1]	a[2]	Out	In
1	4	10	1	2
2	4	10	2	0
4	10	0	4	5

置换选择

- 内存中使用优先级队列排序
- 产生若干已排序片段（然后再多阶段归并）
- 大小为 p 的内存平均能生成长度为 $2p$ 的已排序片段
- *E.g.* 一个例子：内存大小为 3，排序 {1, 4, 10, 2, 0, 5, 7, 3, 6, 9, 12}

a[0]	a[1]	a[2]	Out	In
1	4	10	1	2
2	4	10	2	0
4	10	0	4	5
5	10	0	5	7

置换选择

- 内存中使用优先级队列排序
- 产生若干已排序片段（然后再多阶段归并）
- 大小为 p 的内存平均能生成长度为 $2p$ 的已排序片段
- *E.g.* 一个例子：内存大小为 3，排序 {1, 4, 10, 2, 0, 5, 7, 3, 6, 9, 12}

a[0]	a[1]	a[2]	Out	In
1	4	10	1	2
2	4	10	2	0
4	10	0	4	5
5	10	0	5	7
7	10	0	7	6

置换选择

- 内存中使用优先级队列排序
- 产生若干已排序片段（然后再多阶段归并）
- 大小为 p 的内存平均能生成长度为 $2p$ 的已排序片段
- *E.g.* 一个例子：内存大小为 3，排序 {1, 4, 10, 2, 0, 5, 7, 3, 6, 9, 12}

a[0]	a[1]	a[2]	Out	In
1	4	10	1	2
2	4	10	2	0
4	10	0	4	5
5	10	0	5	7
7	10	0	7	6
10	0	6	10	3

置换选择

- 内存中使用优先级队列排序
- 产生若干已排序片段（然后再多阶段归并）
- 大小为 p 的内存平均能生成长度为 $2p$ 的已排序片段
- *E.g.* 一个例子：内存大小为 3，排序 {1, 4, 10, 2, 0, 5, 7, 3, 6, 9, 12}

a[0]	a[1]	a[2]	Out	In
1	4	10	1	2
2	4	10	2	0
4	10	0	4	5
5	10	0	5	7
7	10	0	7	6
10	0	6	10	3
0	6	3	Over	

置换选择

- 内存中使用优先级队列排序
- 产生若干已排序片段（然后再多阶段归并）
- 大小为 p 的内存平均能生成长度为 $2p$ 的已排序片段
- *E.g.* 一个例子：内存大小为 3，排序 {1, 4, 10, 2, 0, 5, 7, 3, 6, 9, 12}

a[0]	a[1]	a[2]	Out	In
1	4	10	1	2
2	4	10	2	0
4	10	0	4	5
5	10	0	5	7
7	10	0	7	6
10	0	6	10	3
0	6	3	Over	
0	3	6	0	9

置换选择

- 内存中使用优先级队列排序
- 产生若干已排序片段（然后再多阶段归并）
- 大小为 p 的内存平均能生成长度为 $2p$ 的已排序片段
- *E.g.* 一个例子：内存大小为 3，排序 {1, 4, 10, 2, 0, 5, 7, 3, 6, 9, 12}

a[0]	a[1]	a[2]	Out	In
1	4	10	1	2
2	4	10	2	0
4	10	0	4	5
5	10	0	5	7
7	10	0	7	6
10	0	6	10	3
0	6	3	Over	
0	3	6	0	9
3	6	9	3	12

置换选择

- 内存中使用优先级队列排序
- 产生若干已排序片段（然后再多阶段归并）
- 大小为 p 的内存平均能生成长度为 $2p$ 的已排序片段
- *E.g.* 一个例子：内存大小为 3，排序 {1, 4, 10, 2, 0, 5, 7, 3, 6, 9, 12}

a[0]	a[1]	a[2]	Out	In
1	4	10	1	2
2	4	10	2	0
4	10	0	4	5
5	10	0	5	7
7	10	0	7	6
10	0	6	10	3
0	6	3	Over	
0	3	6	0	9
3	6	9	3	12
6	9	12	6	

置换选择

- 内存中使用优先级队列排序
- 产生若干已排序片段（然后再多阶段归并）
- 大小为 p 的内存平均能生成长度为 $2p$ 的已排序片段
- *E.g.* 一个例子：内存大小为 3，排序 {1, 4, 10, 2, 0, 5, 7, 3, 6, 9, 12}

a[0]	a[1]	a[2]	Out	In
1	4	10	1	2
2	4	10	2	0
4	10	0	4	5
5	10	0	5	7
7	10	0	7	6
10	0	6	10	3
0	6	3	Over	
0	3	6	0	9
3	6	9	3	12
6	9	12	6	
9	12		9	

置换选择

- 内存中使用优先级队列排序
- 产生若干已排序片段（然后再多阶段归并）
- 大小为 p 的内存平均能生成长度为 $2p$ 的已排序片段
- *E.g.* 一个例子：内存大小为 3，排序 {1, 4, 10, 2, 0, 5, 7, 3, 6, 9, 12}

a[0]	a[1]	a[2]	Out	In
1	4	10	1	2
2	4	10	2	0
4	10	0	4	5
5	10	0	5	7
7	10	0	7	6
10	0	6	10	3
0	6	3	Over	
0	3	6	0	9
3	6	9	3	12
6	9	12	6	
9	12		9	
12			12	

置换选择

- 内存中使用优先级队列排序
- 产生若干已排序片段（然后再多阶段归并）
- 大小为 p 的内存平均能生成长度为 $2p$ 的已排序片段
- *E.g.* 一个例子：内存大小为 3，排序 {1, 4, 10, 2, 0, 5, 7, 3, 6, 9, 12}

a[0]	a[1]	a[2]	Out	In
1	4	10	1	2
2	4	10	2	0
4	10	0	4	5
5	10	0	5	7
7	10	0	7	6
10	0	6	10	3
0	6	3	Over	
0	3	6	0	9
3	6	9	3	12
6	9	12	6	
9	12		9	
12			12	
			Over	

k 路归并

- 使用 k 条磁带输入，使用 k 条磁带输出
- 初始数据存在 A_1 上，首先将初始片段轮流存放到 $\mathsf{B}_1 \sim \mathsf{B}_k$ 上
- 取每个磁带上的首个归并片段进行归并写至 A_1，重复并轮流写到 $\mathsf{A}_1 \sim \mathsf{A}_k$ 上
- 将 A 磁带作为输入，B 磁带作为输出，重复以上过程，直到归并结束
- 缺点：需要使用 $2k$ 条磁带，穷人用不起
- 改进：用 $k+1$ 条磁带同样也行，称为多阶段归并

多阶段归并

将初始片段非均匀的放在磁带上。

E.g. 三条磁带，34 个初始片段，按照 0，13，21 分配。

	已排序片段数	执行的操作						
		$T_2\|T_3$	$T_1\|T_2$	$T_1\|T_3$	$T_2\|T_3$	$T_1\|T_2$	$T_1\|T_3$	$T_2\|T_3$
T_1	0	13	5	0	3	1	0	1
T_2	21	8	0	5	2	0	1	0
T_1	13	0	8	3	0	2	1	0

多阶段归并

将初始片段非均匀的放在磁带上。

E.g. 三条磁带，34 个初始片段，按照 0，13，21 分配。

	已排序片段数	执行的操作						
		$T_2 \vert T_3$	$T_1 \vert T_2$	$T_1 \vert T_3$	$T_2 \vert T_3$	$T_1 \vert T_2$	$T_1 \vert T_3$	$T_2 \vert T_3$
T_1	0	13	5	0	3	1	0	1
T_2	21	8	0	5	2	0	1	0
T_1	13	0	8	3	0	2	1	0

最优分配方式：斐波那契数列