

计算机应用编程实验

树结构字符串检索

熊永平@网络技术研究院

ypxiong@bupt.edu.cn

教三楼233

2019.10

实验任务

- 问题
 - ▣ 给定127万个中文字符串作为模式库
 - ▣ 待查找的98万个中文串
 - ▣ 从模式库中查找98万个串看是否能命中
- 挑战
 - ▣ 模式串的规模较大，需要构造一个高效数据结构来处理

版本1：二叉搜索树 (Binary Search Tree)

- 定义

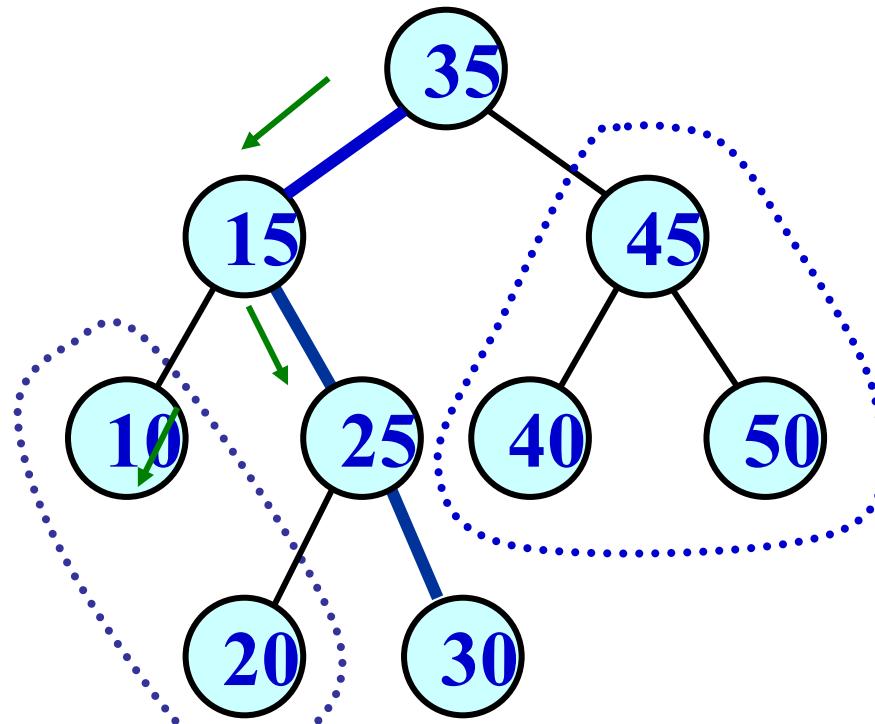
二叉搜索树或者是一棵空树，或者是具有下列性质的二叉树

：

- ✓ 每个结点都有一个作为搜索依据的关键码(key)，所有结点的关键码互不相同。
- ✓ 左子树（如果非空）上所有结点的关键码都小于根结点的关键码。
- ✓ 右子树（如果非空）上所有结点的关键码都大于根结点的关键码。
- ✓ 左子树和右子树也是二叉搜索树。

二叉搜索树例

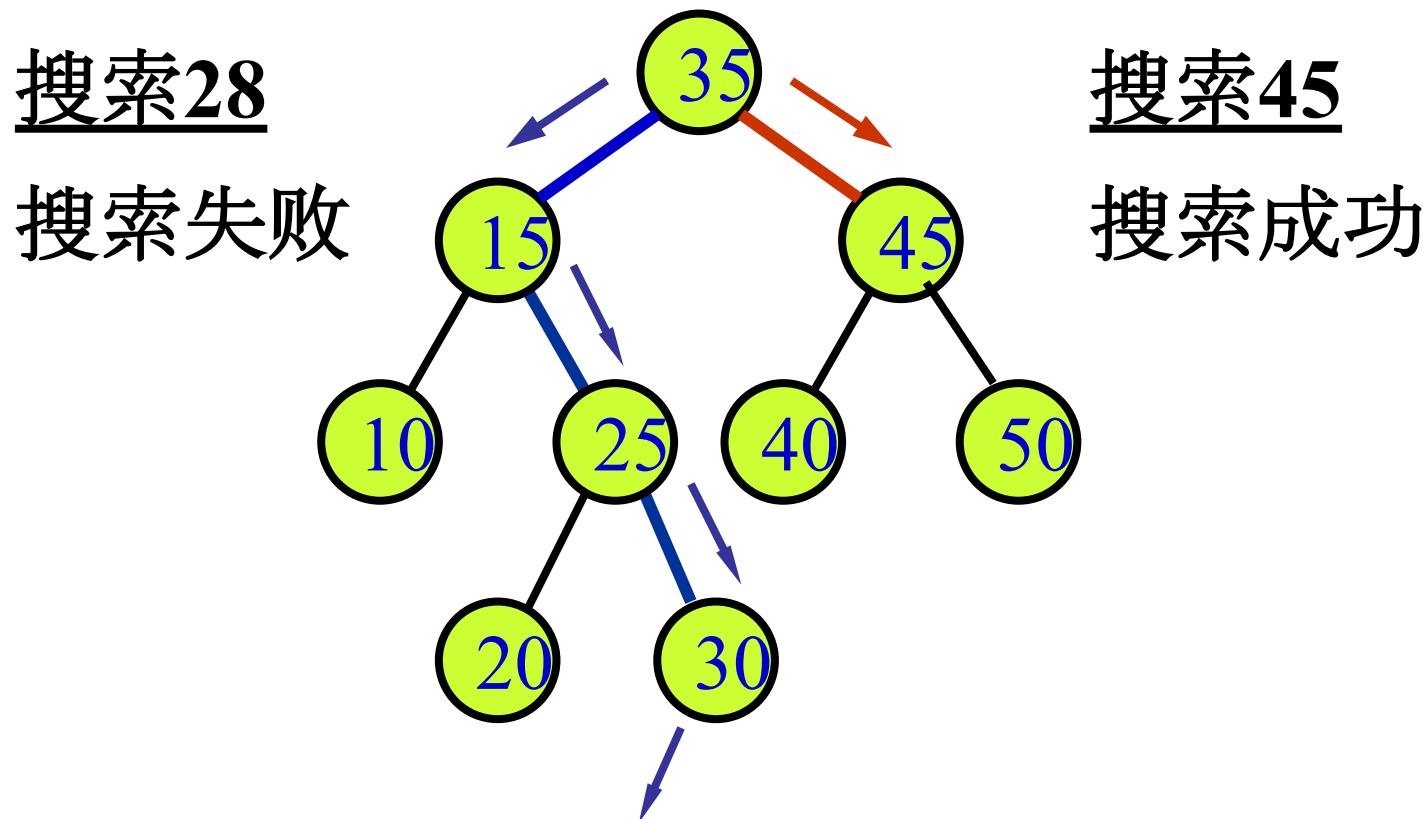
- 结点左子树上所有关键码小于结点关键码；
- 右子树上所有关键码大于结点关键码；



二叉搜索树的搜索算法

- 在二叉搜索树上进行搜索，是一个从根结点开始，沿某一个分支逐层向下进行比较判等的过程。它可以是一个递归的过程。
- 假设想要在二叉搜索树中搜索关键码为 x 的元素，搜索过程从根结点开始。
- 如果根指针为NULL，则搜索不成功；否则用给定值 x 与根结点的关键码进行比较：
 - ✓ 若给定值等于根结点关键码，则搜索成功，返回搜索成功信息并报告搜索到结点地址。

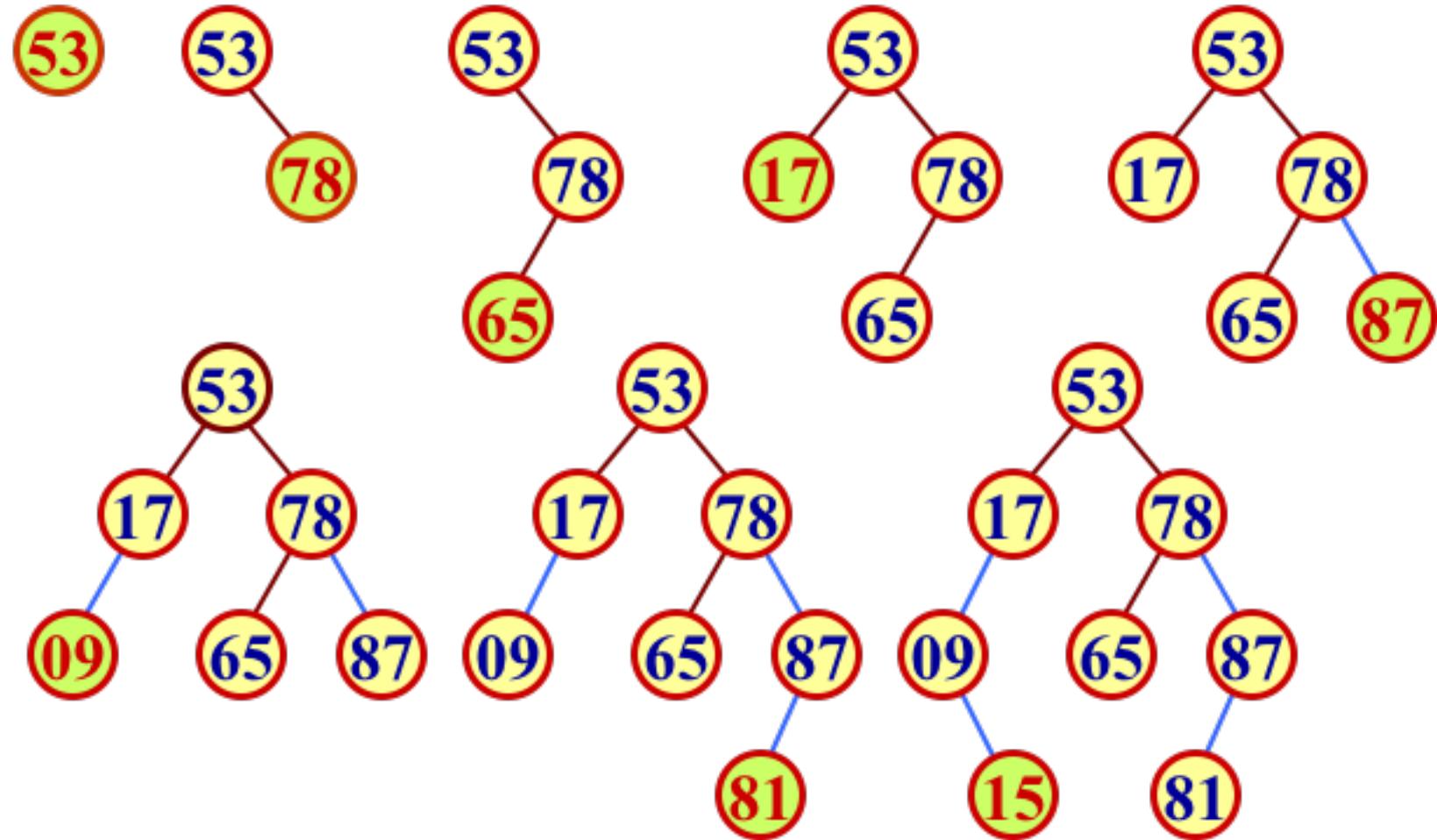
- ✓ 若给定值小于根结点的关键码，则继续递归搜索根结点的左子树；
- ✓ 否则。递归搜索根结点的右子树。



二叉搜索树的插入算法

- 为了向二叉搜索树中插入一个新元素，必须先检查这个元素是否在树中已经存在。
- 在插入之前，先使用搜索算法在树中检查要插入的元素在不在树中。
 - 如果搜索成功，说明树中已经有这个元素，不再插入；
 - 如果搜索不成功，说明树中原来没有关键码等于给定值的结点，把新元素加到搜索操作停止的地方。

输入数据 { 53, 78, 65, 17, 87, 09, 81, 15 }



平衡二叉排序树——AV树

定义：

一棵平衡二叉排序树或者是空树，或者是具有下列性质的二叉排序树：

- 1、左子树与右子树的高度之差的**绝对值**小于等于1；
- 2、左子树和右子树也是平衡二叉排序树。

平衡二叉排序树的**平均查找长度**为 $O(\log_2 n)$ 。

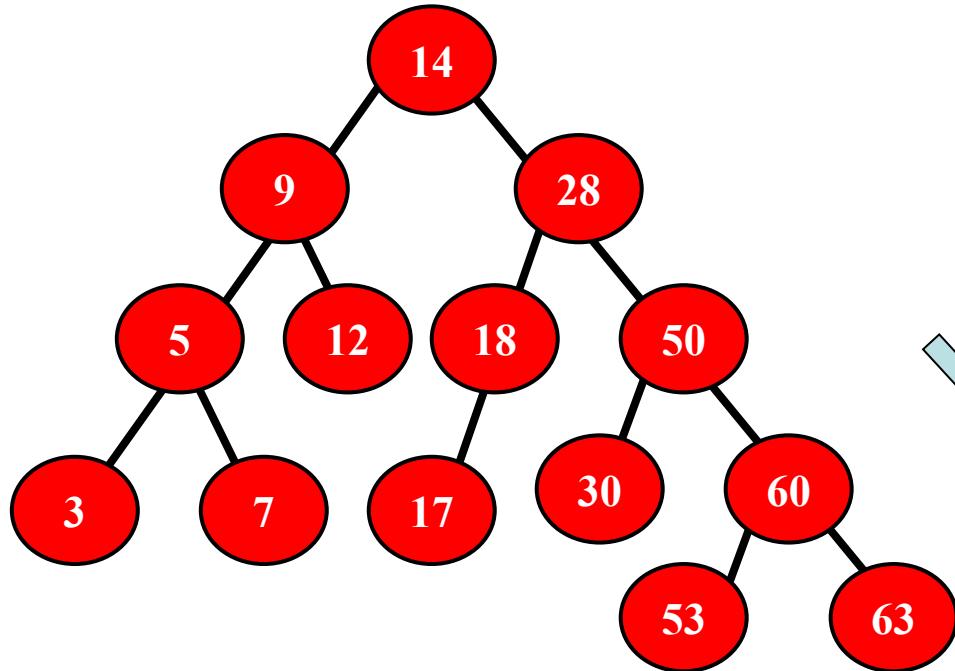
平衡因子：结点的左子树深度与右子树深度之差： -1， 0 , 1。

保持平衡

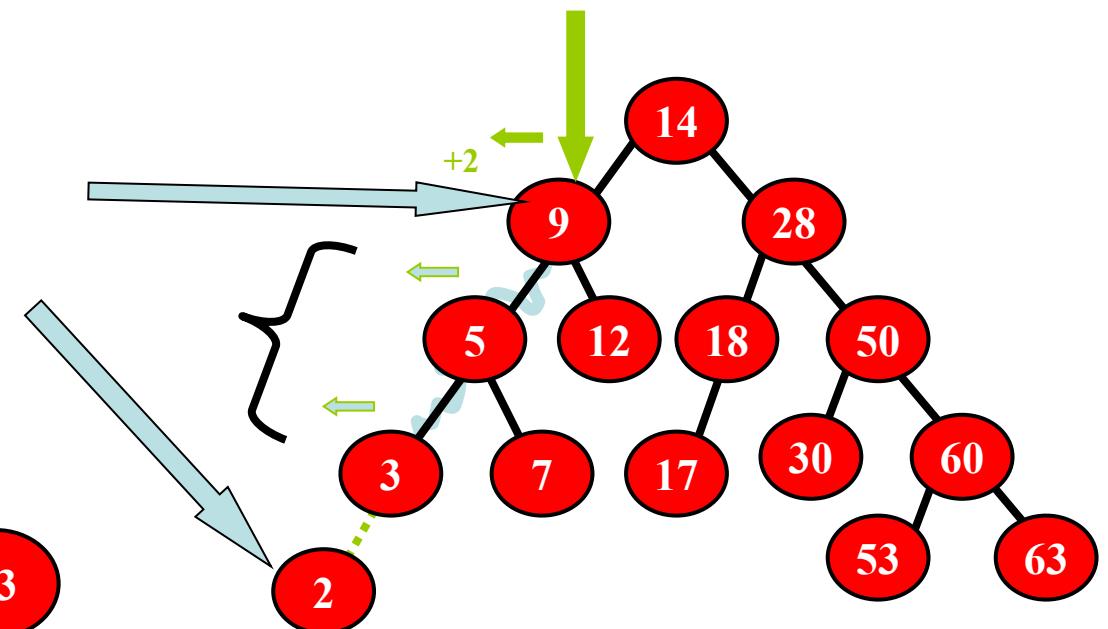
• 以插入为例：

在左图所示的平衡树中插入数据场为 2 的结点。

插入之后仍应保持平衡分类二叉树的性质不变。



平衡树



如何用最简单、最有效的办法保持平衡分类二叉树的性质不变？

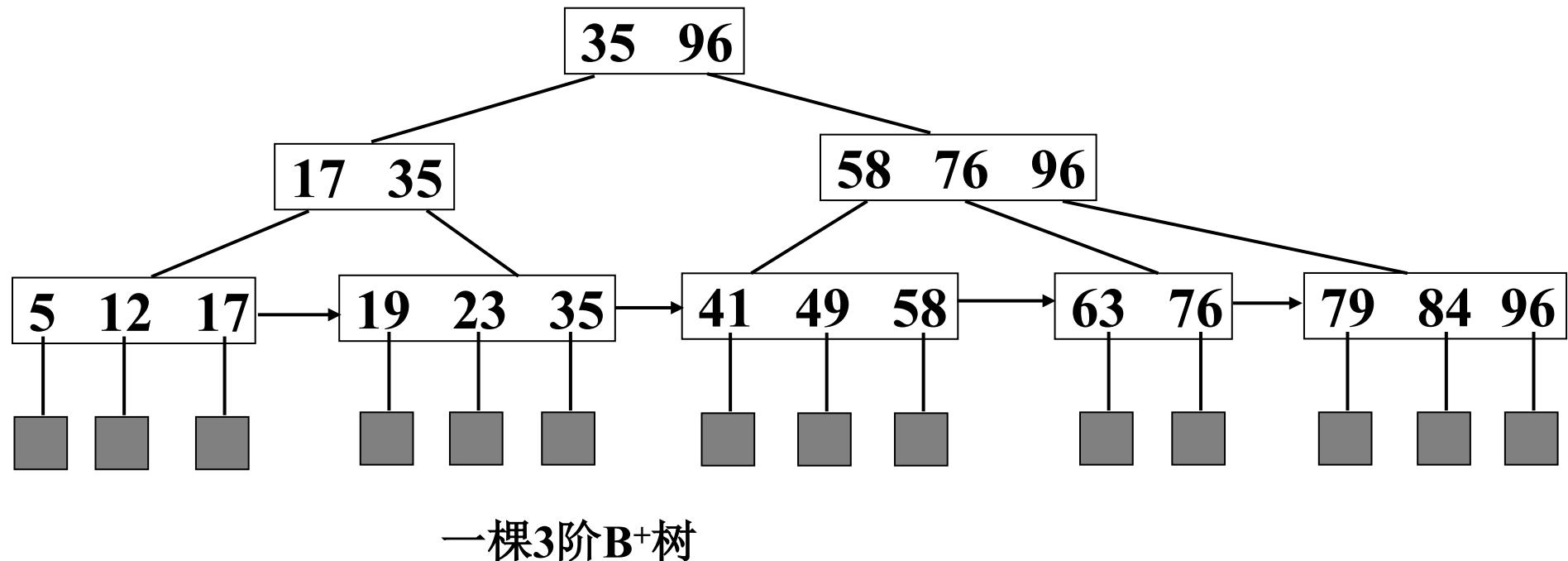
版本2：B⁺树

数据库系统中广泛使用的索引结构为m阶B⁺树。它与B_m树的主要不同是叶子结点中存储记录。在B⁺树中，所有的非叶子结点可以看成是索引，而其中的关键字是作为“分界关键字”，用来界定某一关键字的记录所在的子树。一棵m阶B⁺树与m阶B_m树的主要差异是：

- (1) 若一个结点有n棵子树，则必含有n个关键字；
- (2) 所有叶子结点中包含了全部记录的关键字信息以及这些关键字记录的指针，而且叶子结点按关键字的大小从小到大顺序链接；
- (3) 所有的非叶子结点可以看成是索引的部分，结点中只含有其子树的根结点中的最大(或最小)关键字。

如图是一棵3阶B+树。

由于B⁺树的叶子结点和非叶子结点结构上的显著区别，因此需要一个标志域加以区分，结点结构定义如下：



B+树构建

- B⁺树的查找

对B⁺树可以进行两种查找运算：

1. 从最小关键字起顺序查找；
2. 从根结点开始，进行随机查找。

在查找时，若非终端结点上的剧组机等于给定值，并不终止，而是继续向下直到叶子结点。因此，在B⁺树中，不管查找成功与否，每次查找都是走了一条从根到叶子结点的路径。其余同B-树的查找类似。

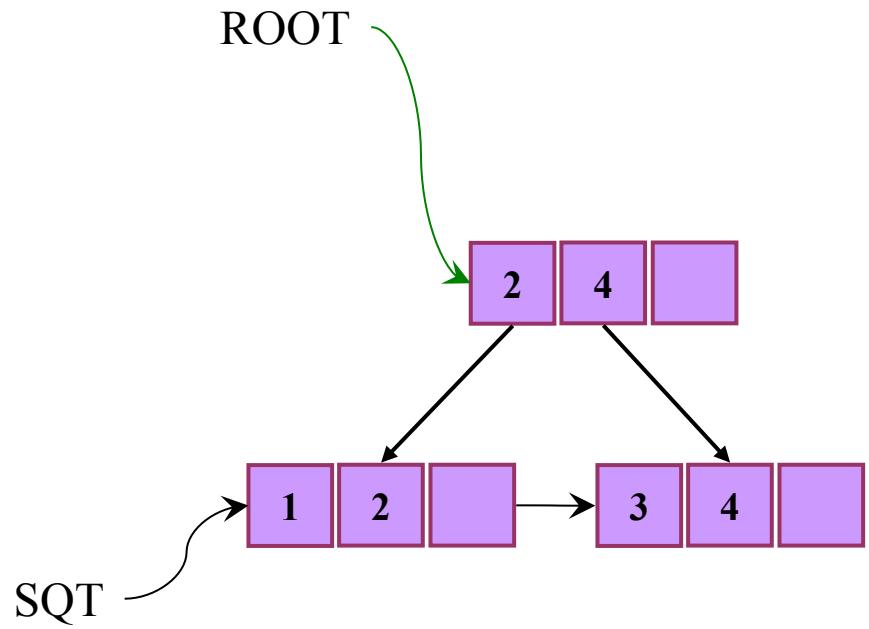
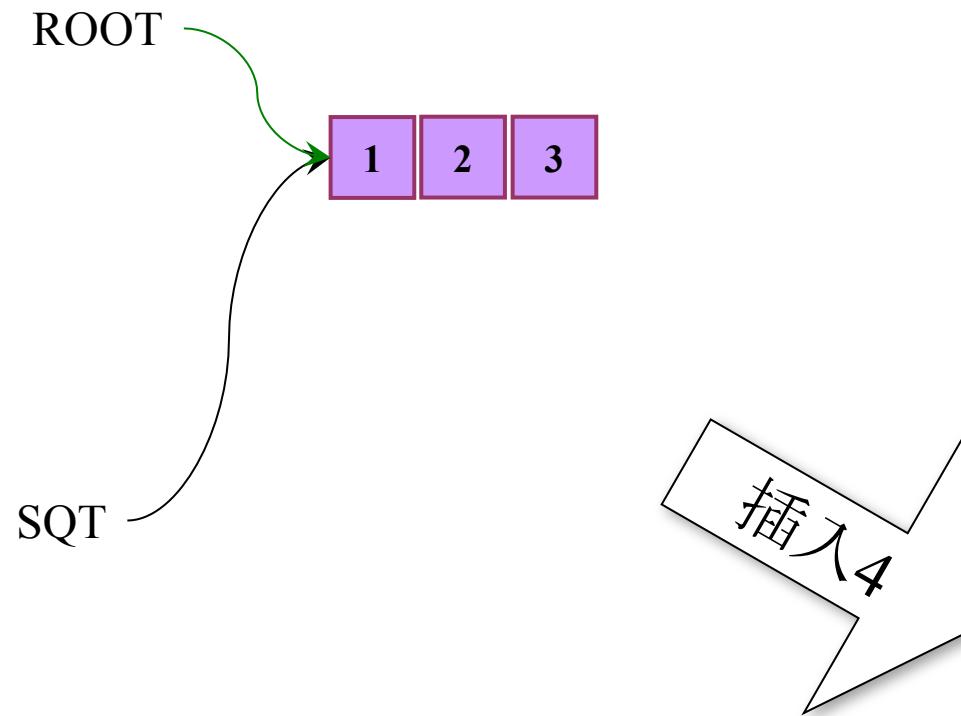
- B⁺树的插入

B⁺树的插入仅在叶子结点上进行，当结点中的关键字个数大于m时要分裂成两个结点，它们所含关键字的个数分别为 $\lceil \frac{m+1}{2} \rceil$ 和 $\lceil \frac{m}{2} \rceil$ 。并且，它们的双亲结点中应同时包含这两个结点中的最大关键字。如果插入的元素是当前节点的最小值或最大值，需要递归向上更新父节点。

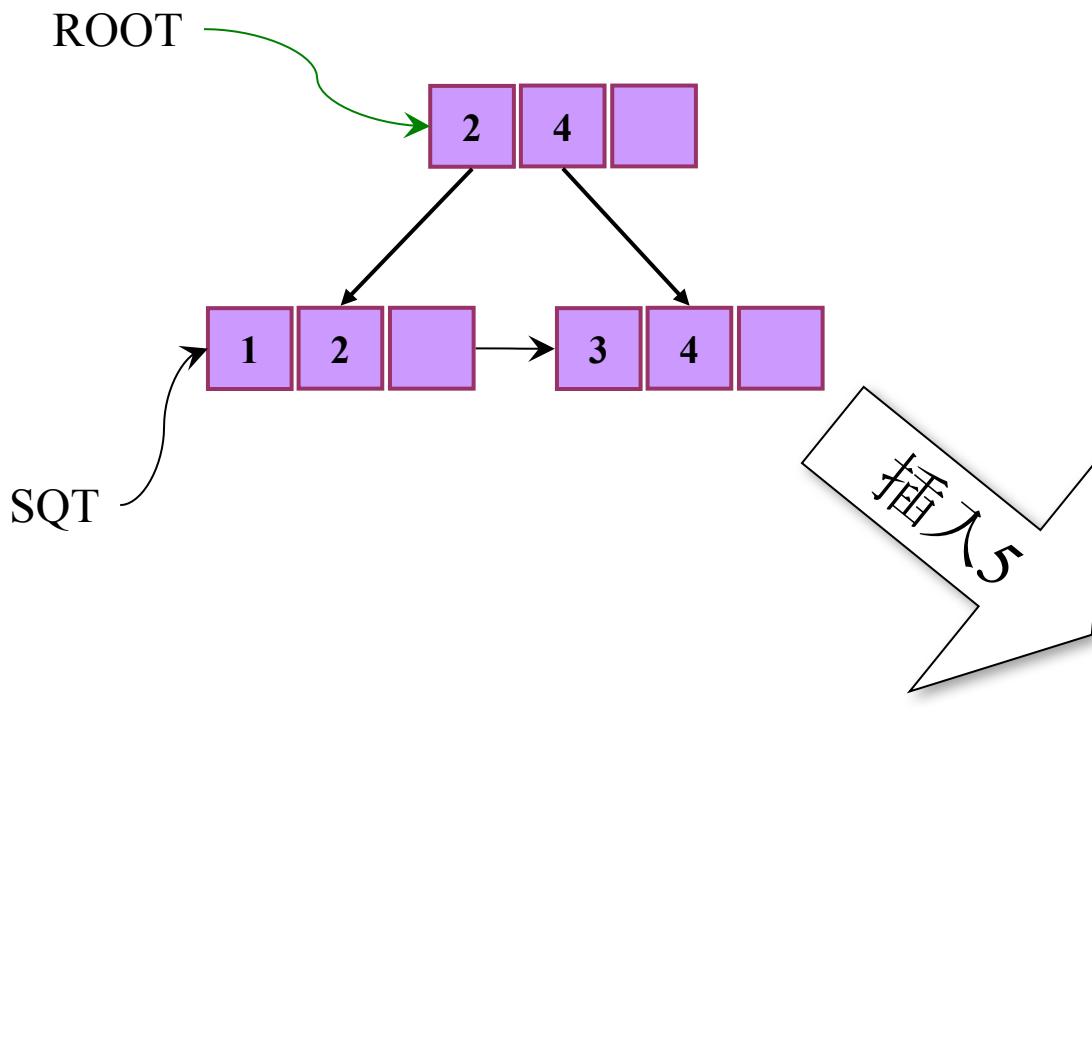
- B⁺树的删除

B⁺树的删除也仅在叶子结点进行，当叶子结点中的最大关键字被删除时，其在非终端结点中的值可以作为一个“分界关键字”存在。若因删除而使结点中关键字的个数少于 $\lceil \frac{m}{2} \rceil$ 时，其和兄弟结点的合并过程亦和B-树类似。

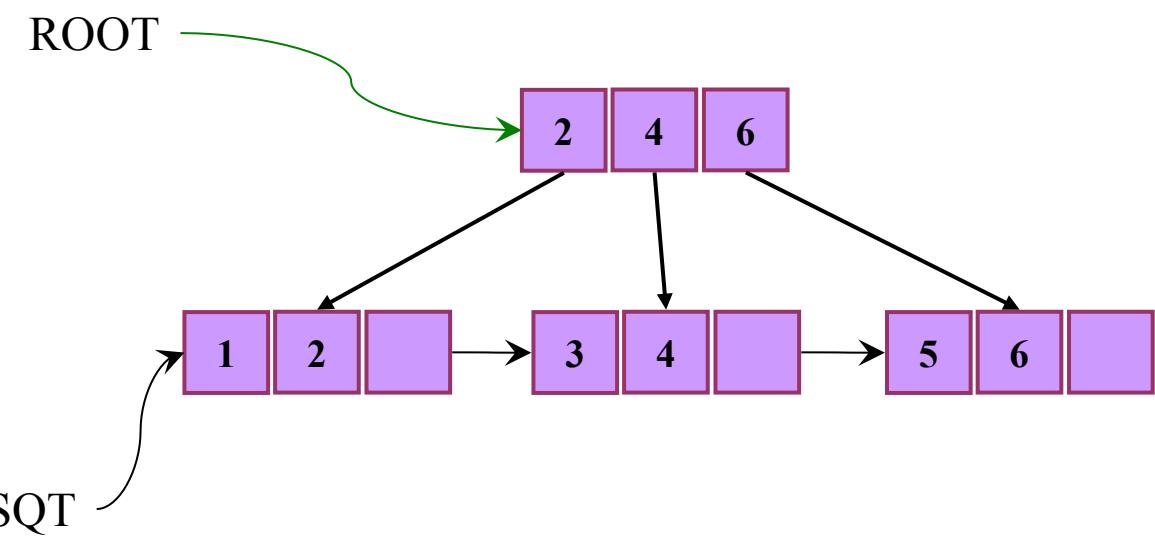
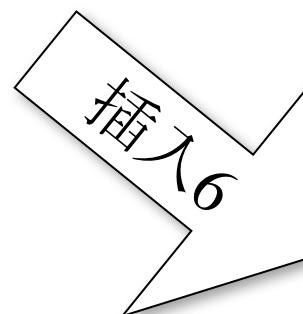
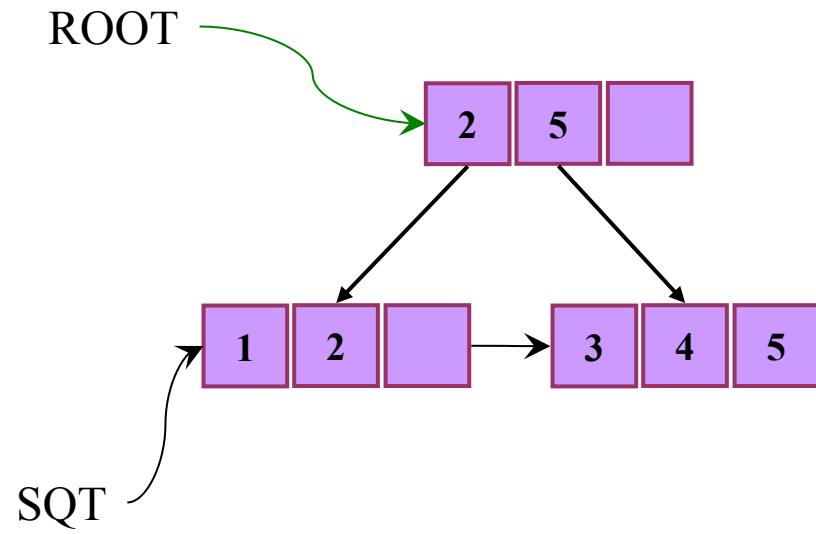
B+树插入



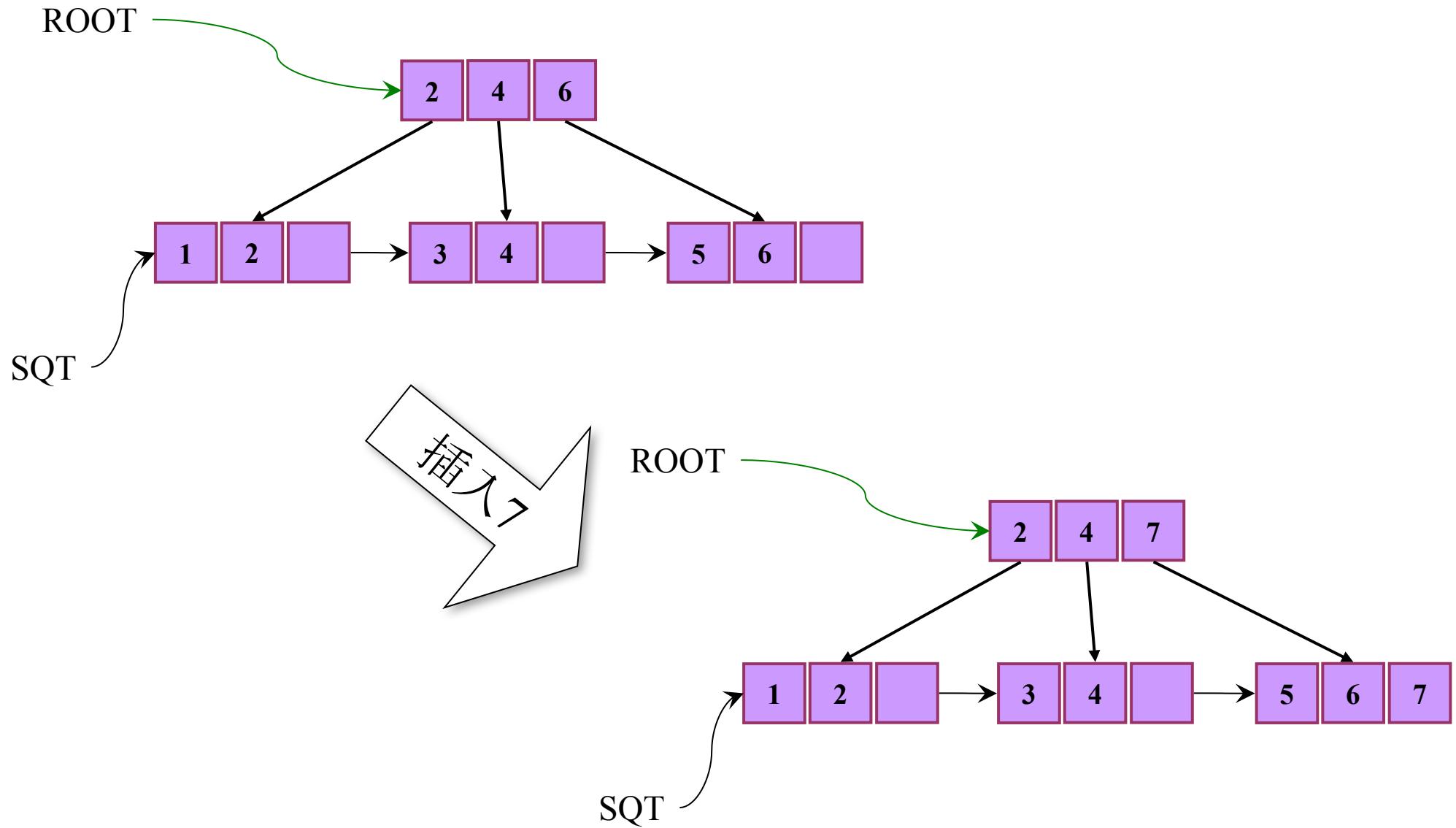
B+树插入



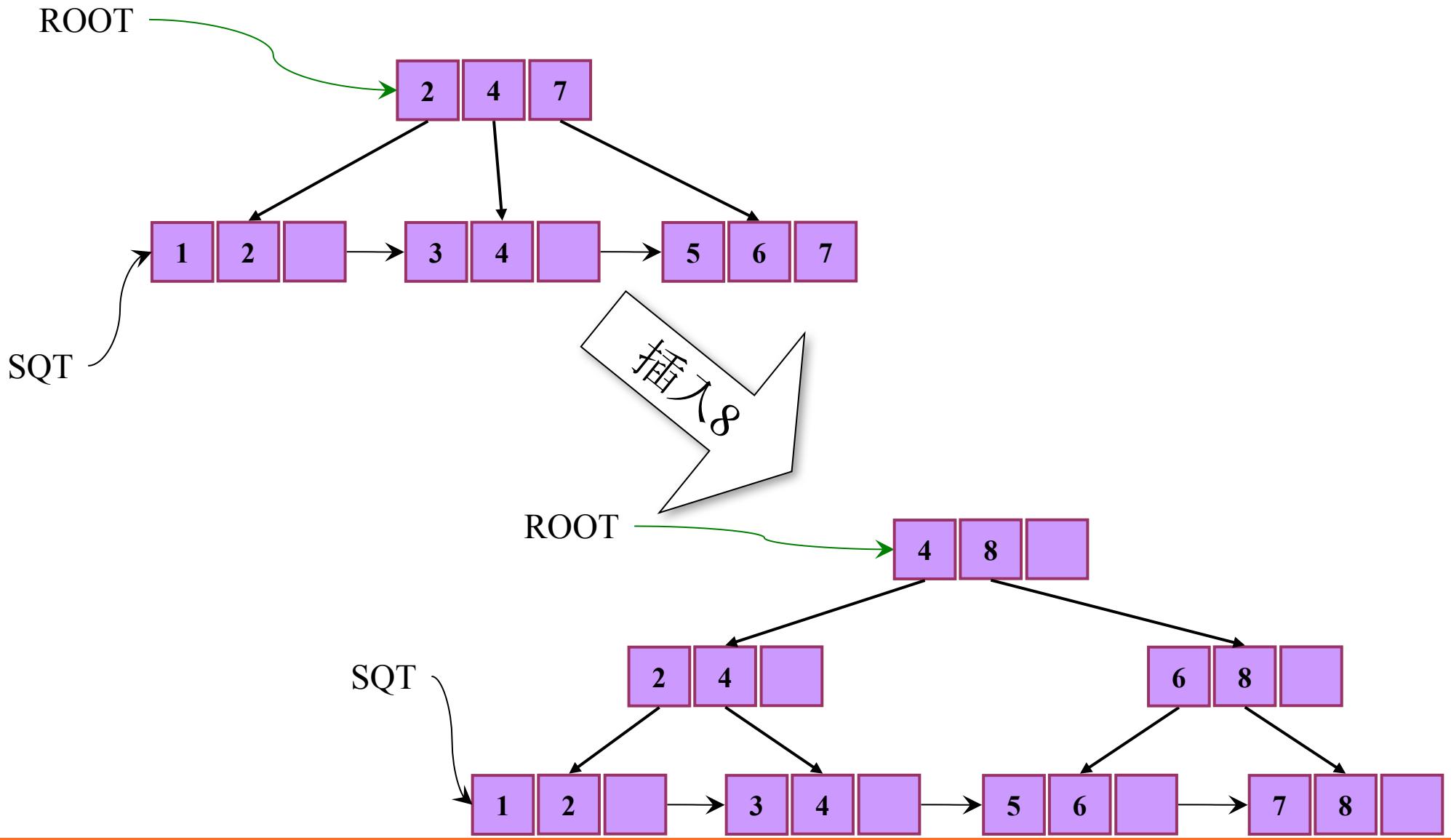
B+树插入



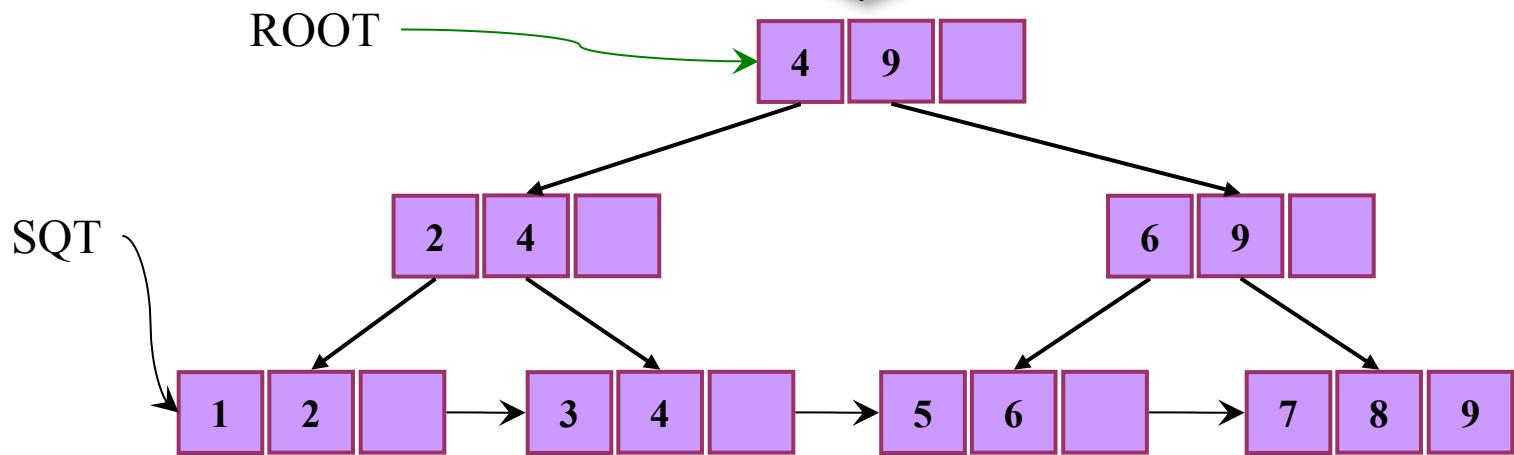
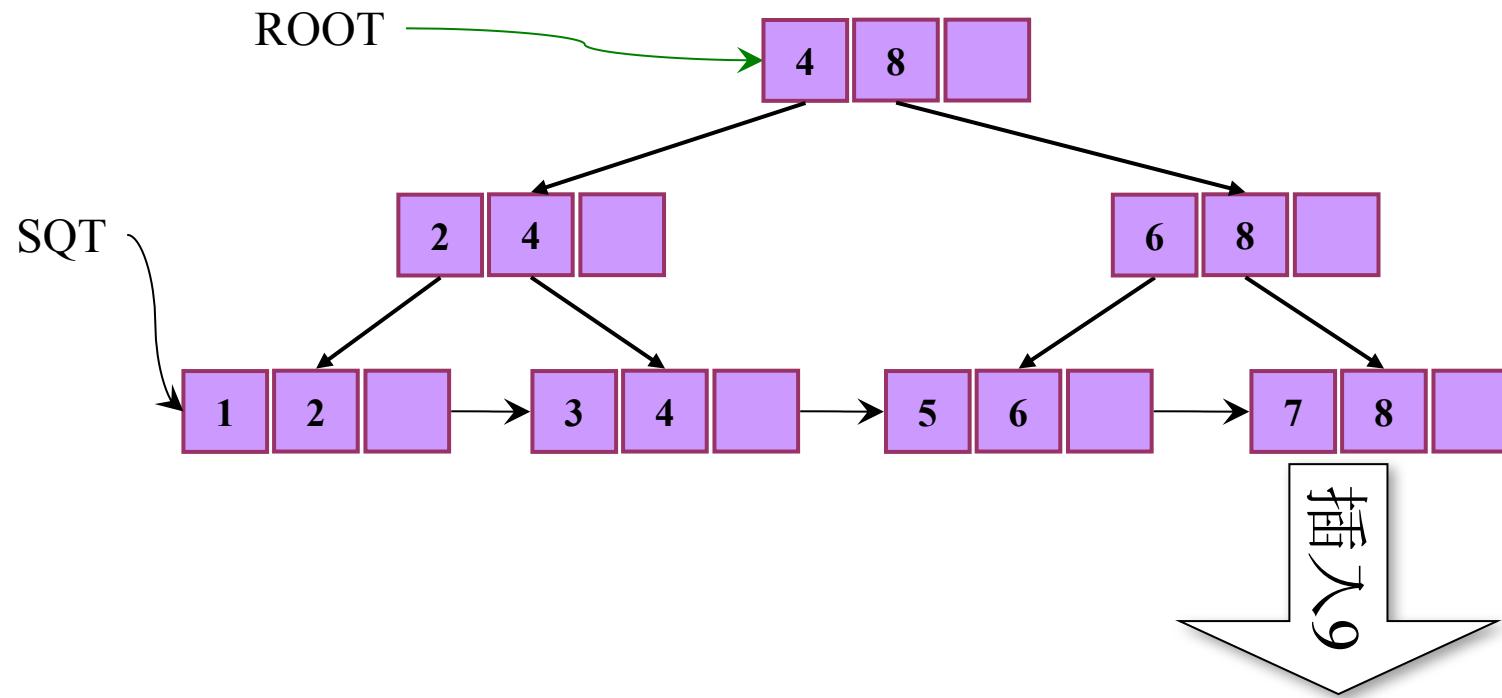
B+树插入



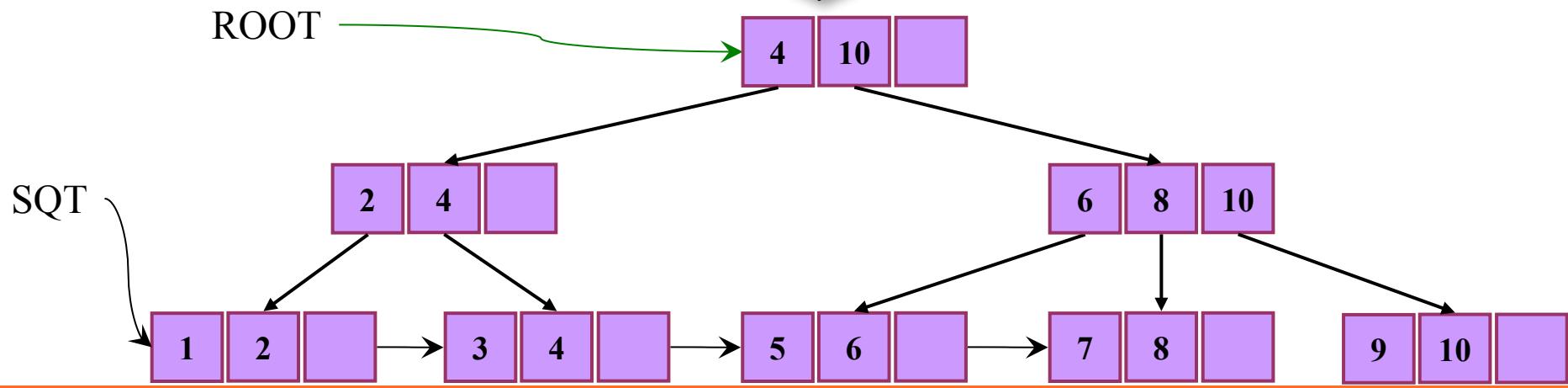
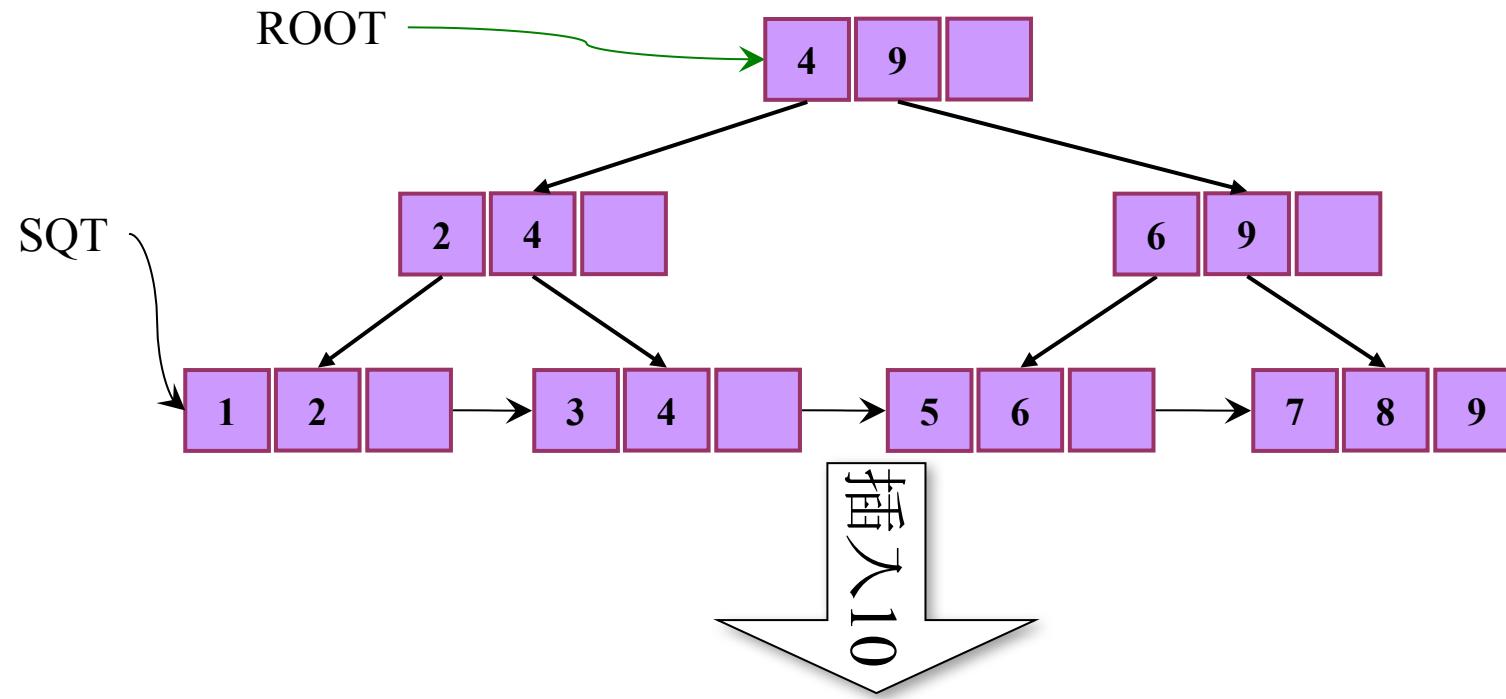
B+树插入



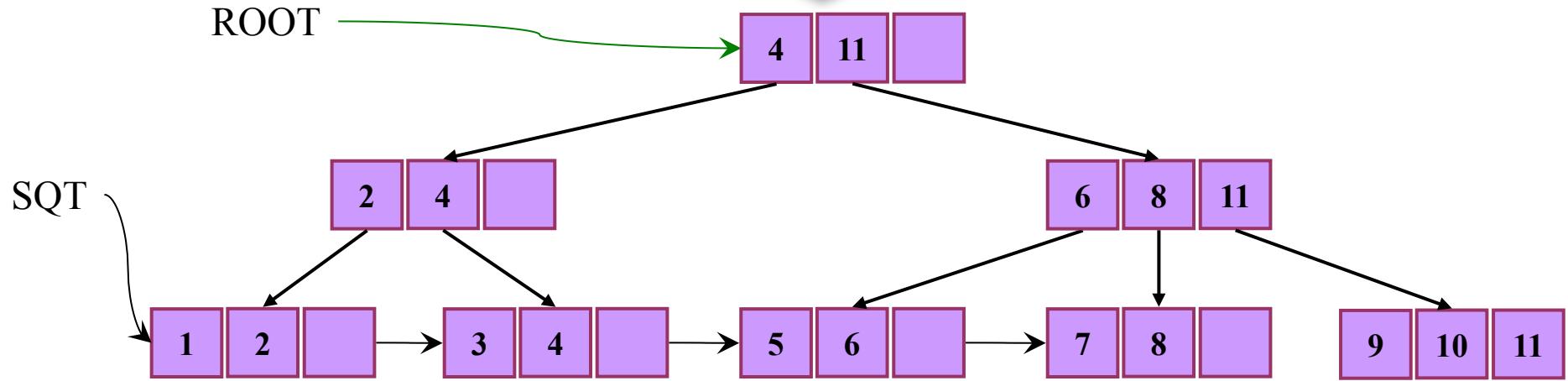
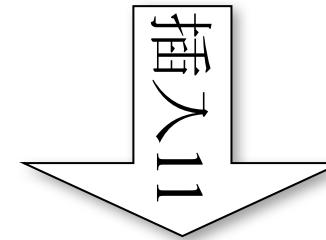
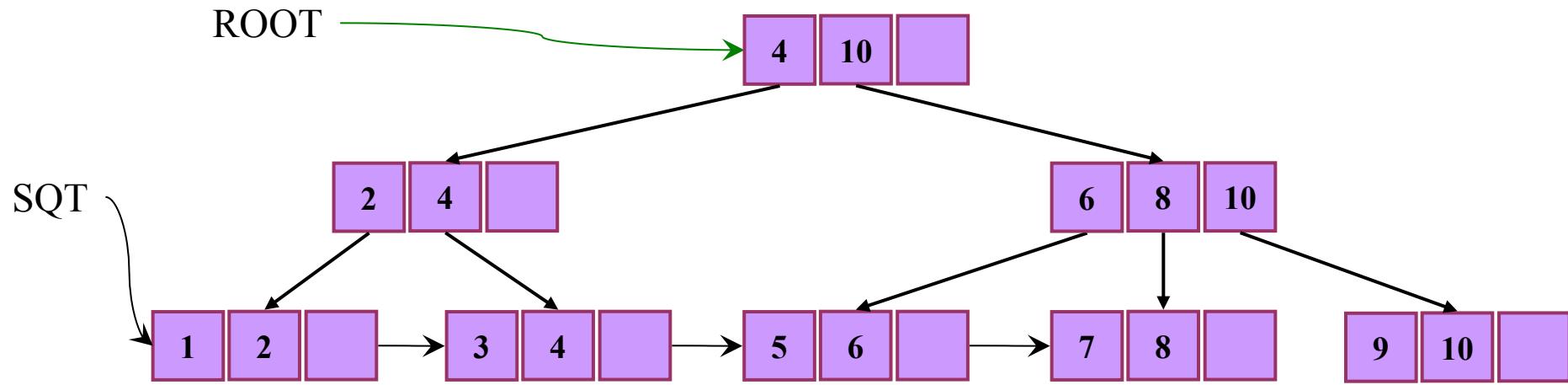
B+树插入



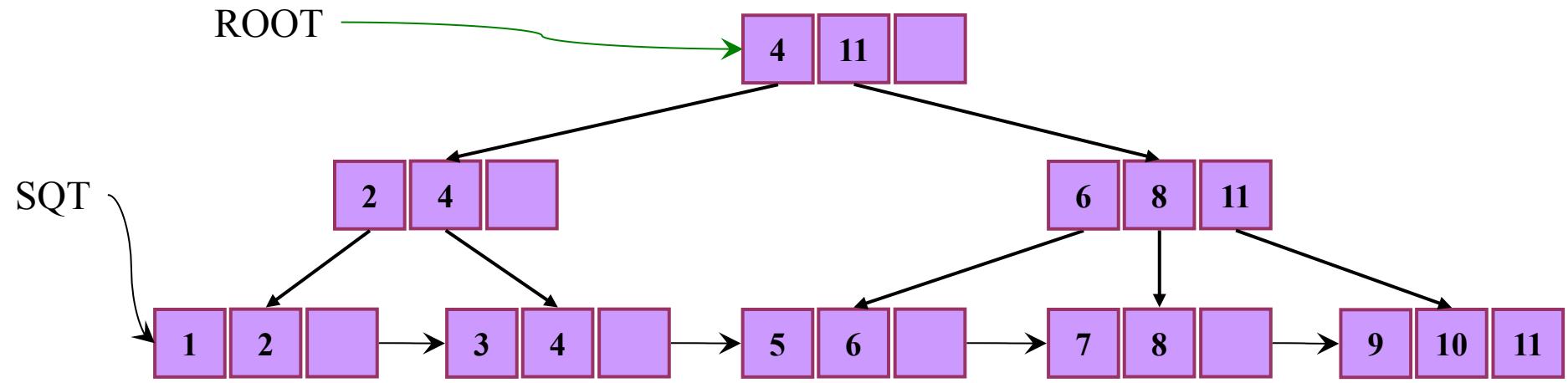
B+树插入



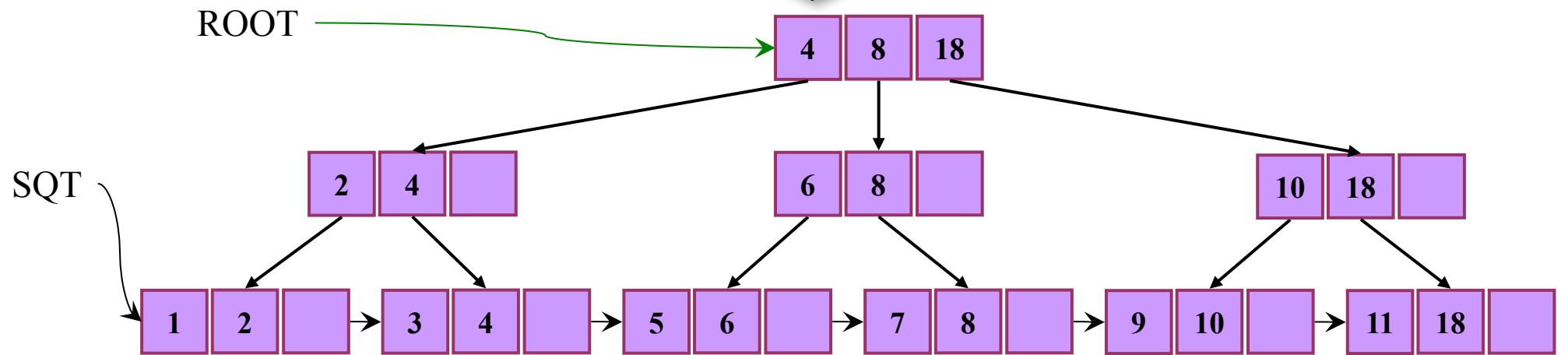
B+树插入



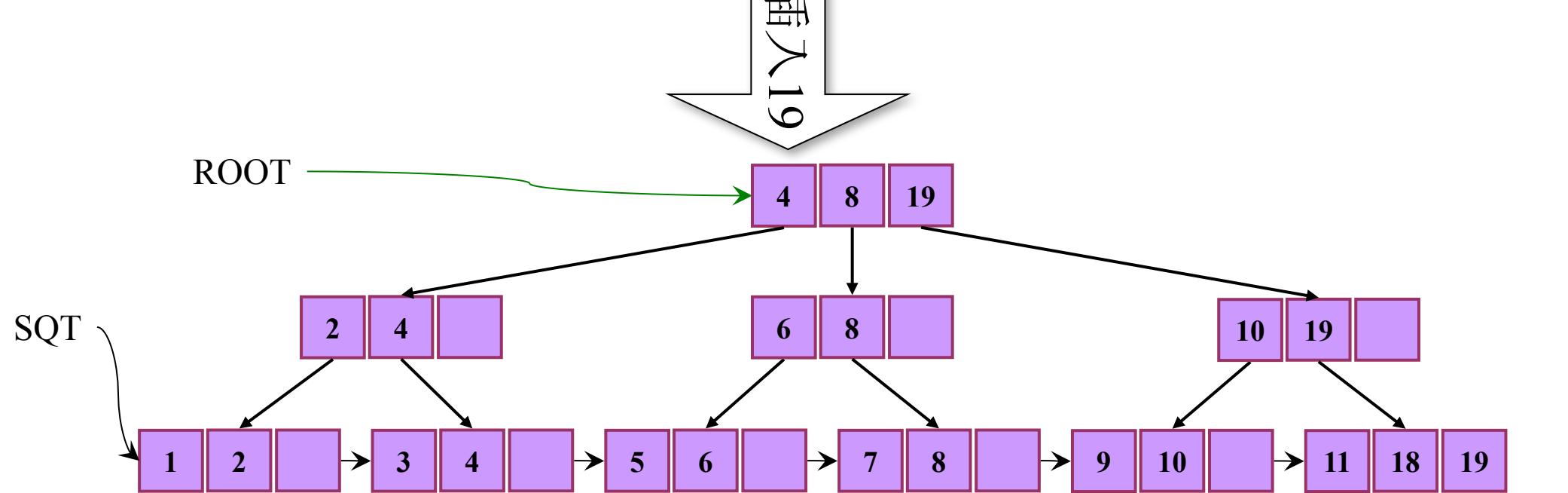
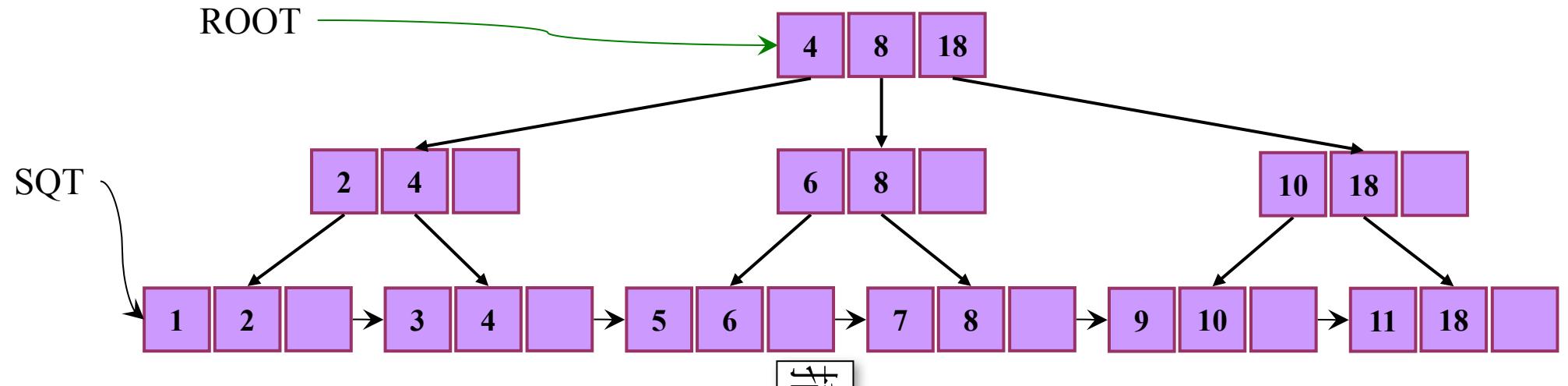
B+树插入



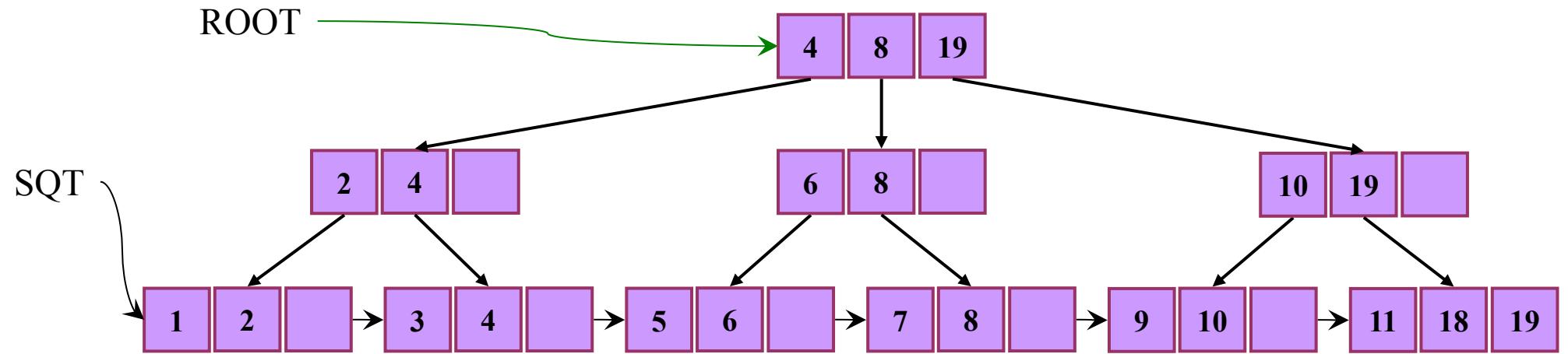
插入 18



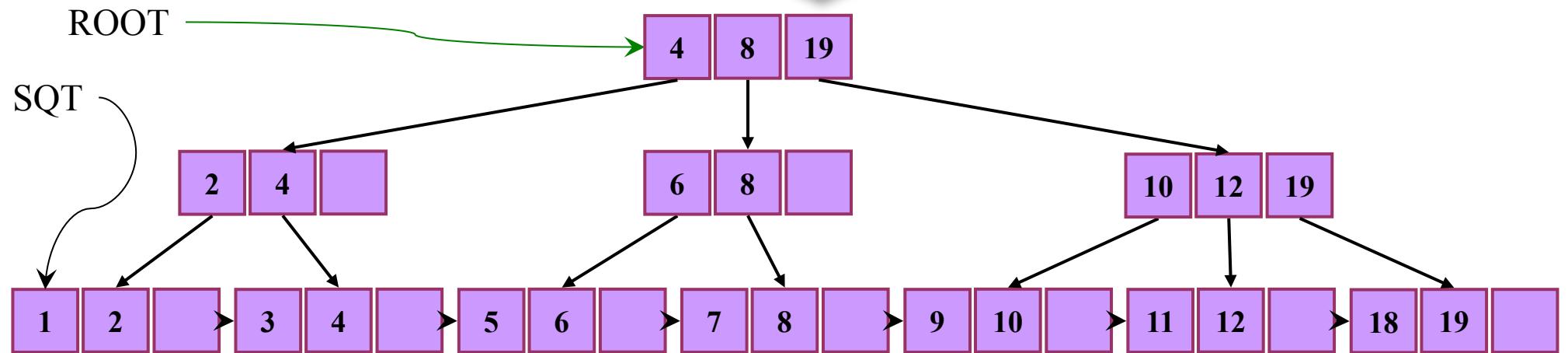
B+树插入



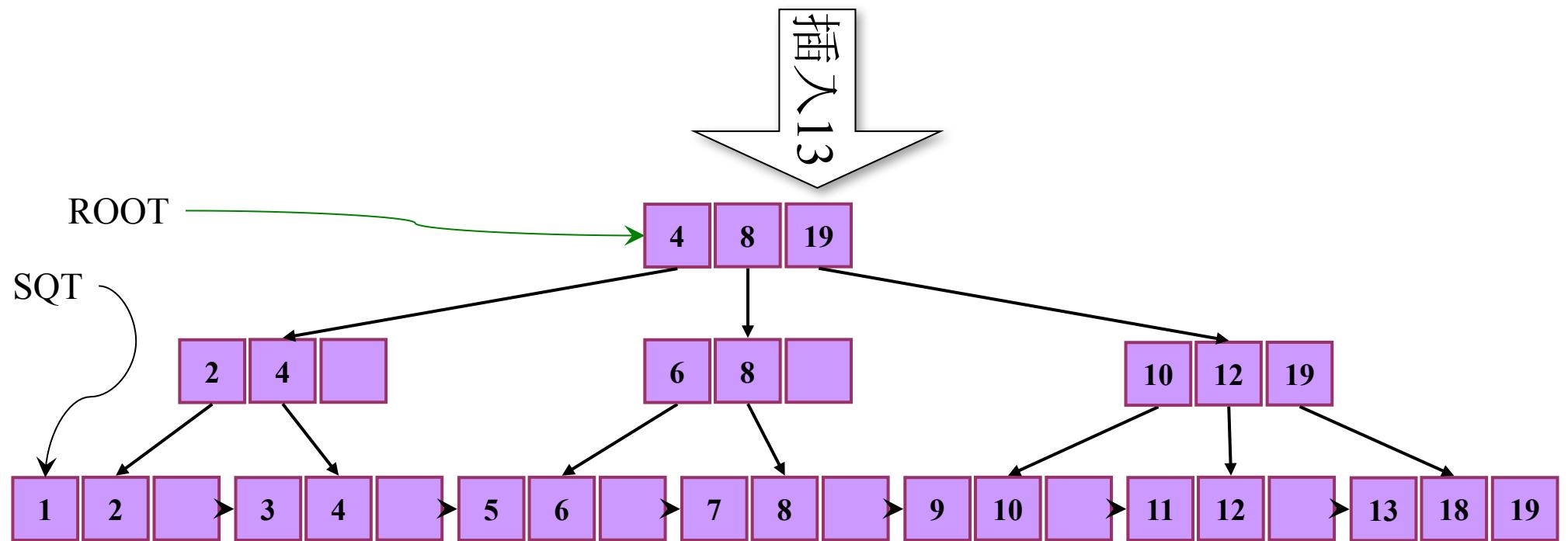
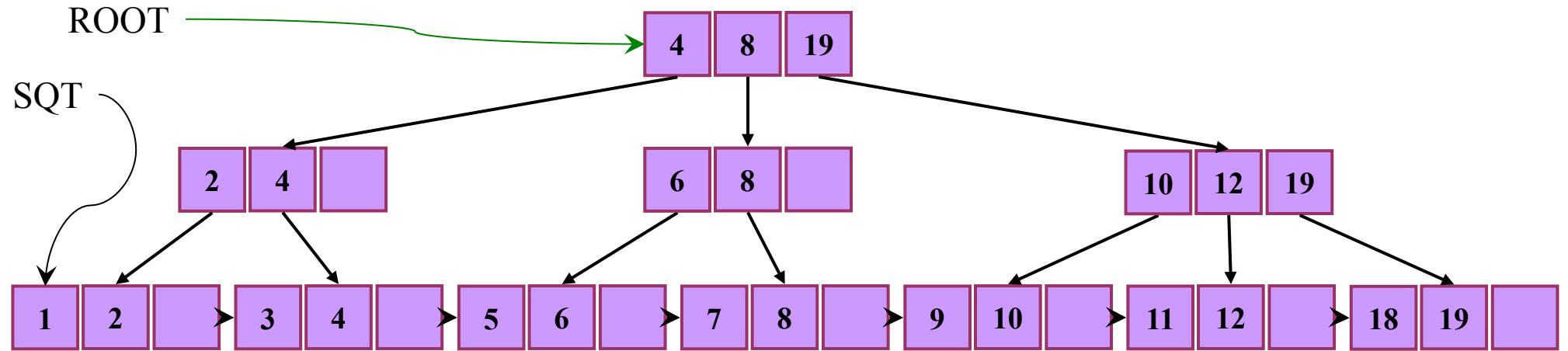
B+树插入



插入 12



B+树插入



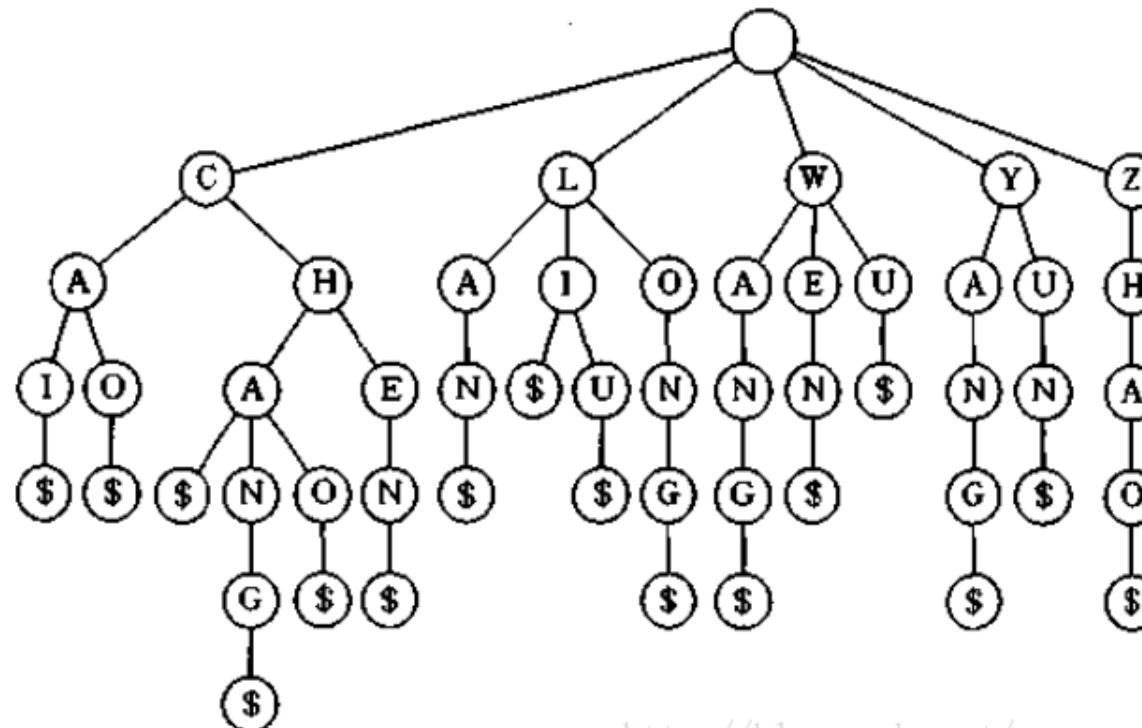
版本3： Trie树

- 基于关键码分解的数据结构， 叫作Trie结构（Trie树）
- 基于两个原则
 - ▣ 有一个固定的关键码集合
 - ▣ 对于结点的分层标记
- **Trie树**
 - ▣ 又称单词查找树、字典树， 是一种树形结构， 是一种用于快速检索的多叉树结构
- **典型应用**
 - ▣ 统计和排序大量的字符串
 - ▣ 文本词频统计和文本检索
 - ▣ 优点：最大限度地减少无谓的字符串比较， 查询效率比哈希表高。

Trie结构示例

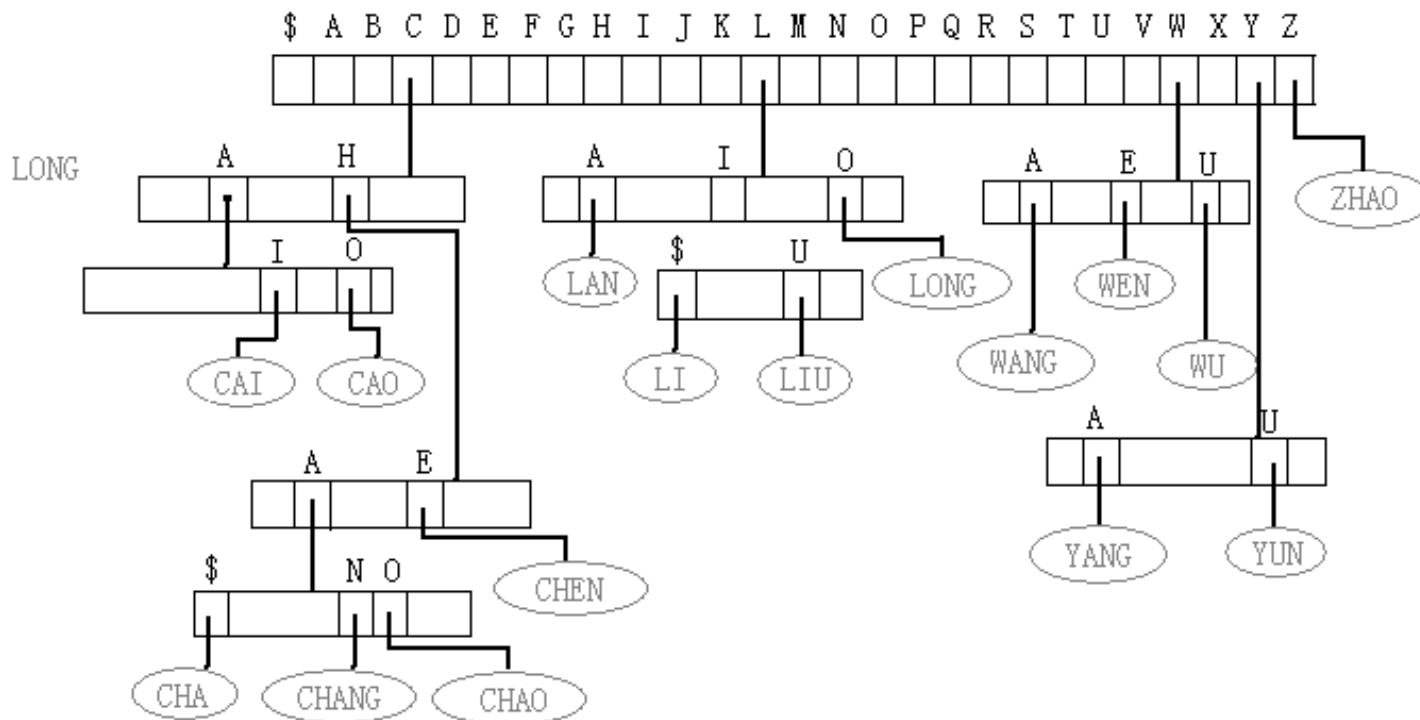
- 存储词典

- { CAI、CAO、LI、LAN、CHA、CHANG、WEN、CHAO、YUN、YANG、
LONG、WANG、ZHAO、LIU、WU、CHEN }
- 树的高度为最长字符串长度



Trie数据结构示例

- 数据结构实现
 - 分支结点：含有d个指针域和一个指示该结点中非空指针域的个数的整数域。
 - 分支结点所表示的字符是由其指向子树指针的索引位置决定的叶子结点：含有关键字域和指向记录的指针域。



```
Typedef Struct TrieNode{  
    NodeKind kind ;  
    union {  
        struct {KeyType K; Record  
        *infoptr} lf; //叶子结点  
        struct {TrieNode *ptr[27]; int num}  
        bh; //分支结点};  
    } TrieNode,*TrieTree ;
```

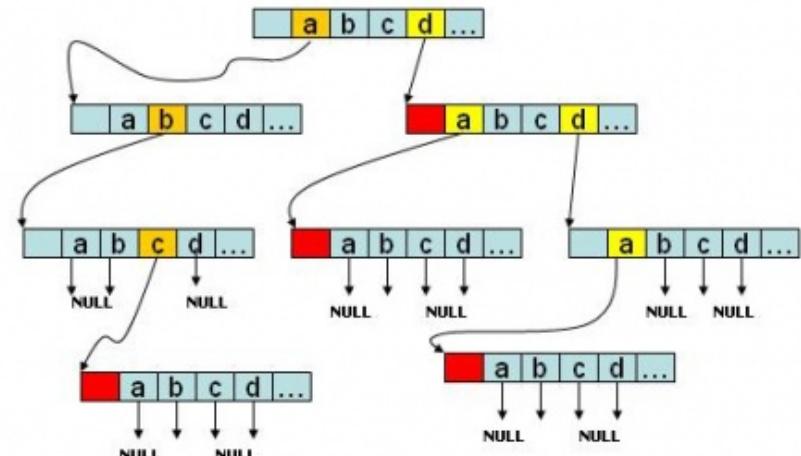
Trie查找

- **查找**

- 在Trie树上进行检索总是始于根结点。
- 取得要查找关键词的第一个字母，并根据该字母选择对应的子树并转到该子树继续进行检索。
- 在某个结点处相应的子树上，取得要查找关键词的第二个字母，并进一步选择对应的子树进行检索。
- 关键词的所有字母已被取出，则读取附在该结点上的信息，即完成查找

- **示例**

- trie树中保存了abc、d、da、ddaa四个单词



Trie树的插入

- 插入
 - ▣ 首先根据插入纪录的关键码找到需要插入的结点位置
 - ▣ 如果该结点是叶结点，那么就将为其分裂出两个子结点，分别存储这个纪录和以前的那个纪录
 - ▣ 如果是内部结点，则在那个分支上应该是空的，所以直接为该分支建立一个新的叶结点即可

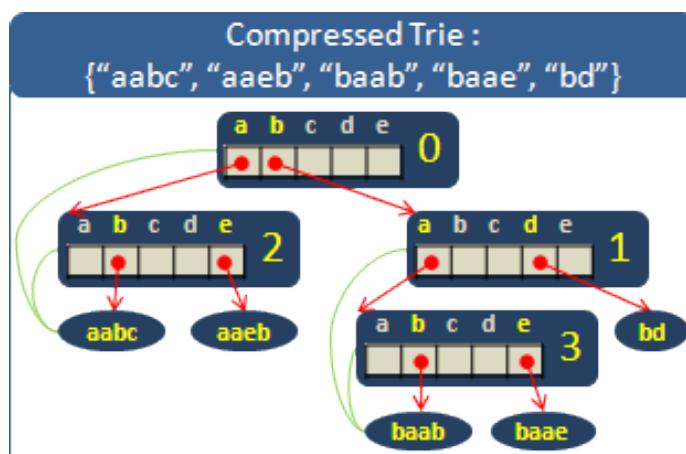
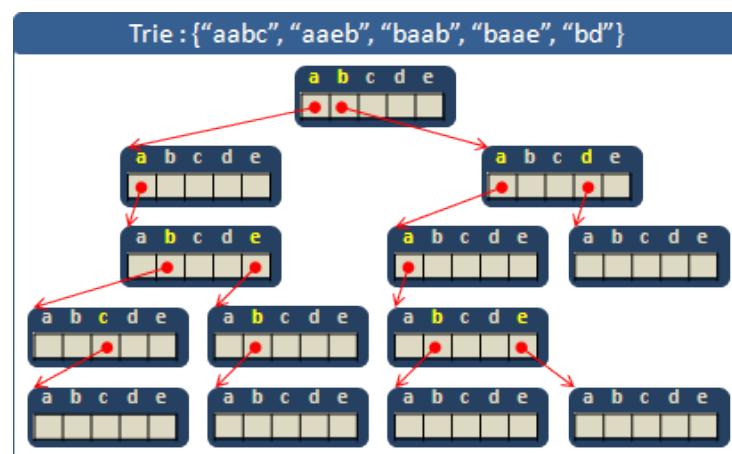
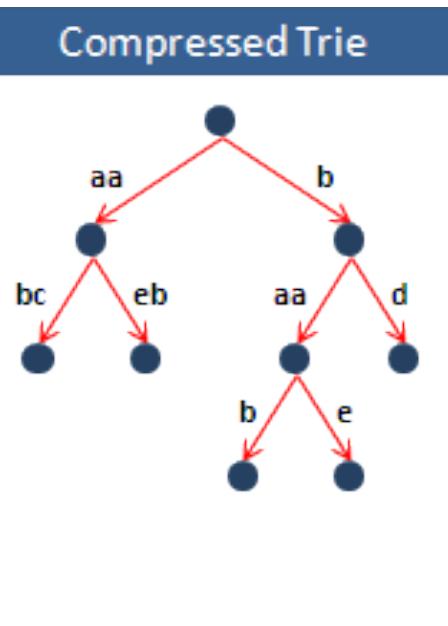
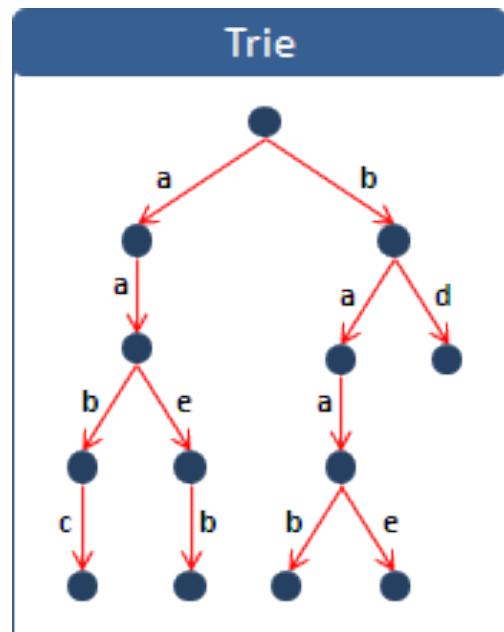
Trie查找效率分析

- 在trie树中查找一个关键字的时间和树中包含的结点数无关，而取决于组成关键字的字符数。
- 对比：二叉查找树的查找时间和树中的结点数有关 $O(\log_2 n)$ 。
- 如果要查找的关键字可以分解成字符序列且不是很长，利用trie树查找速度优于二叉查找树。
 - 若关键字长度最大是5，则利用trie树，利用5次比较可以从 $26^5 = 11881376$ 个可能的关键字中检索出指定的关键字。而利用二叉查找树至少要进行 $\log_2 26^5 = 23.5$ 次比较。

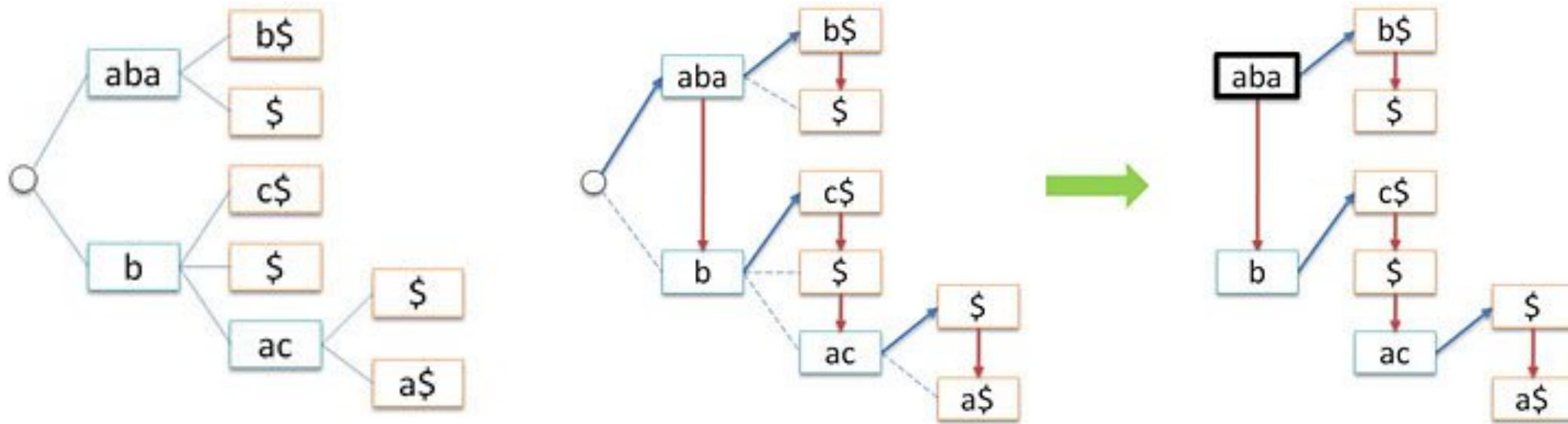
Trie结构总结

- 核心思想
 - 空间换时间
 - 利用字符串的公共前缀来降低查询时间的开销以达到提高效率的目的
- 优点
 - 查找效率高，与词表长度无关
 - Trie树的查找效率只与关键词长度有关
 - 索引的插入，合并速度快
- 缺点
 - 内存空间消耗大
 - 如果是完全m叉树，节点数指数级增长
 - 不可达上限： 词数 \times 字符序列长度 \times 字符集大小 \times 指针长度
 - 例如： $20000 \times 6 \times 256 \times 4 = 120M$
 - 实现较复杂

Trie优化思想（1）：压缩



Trie优化思想 (2) : 二叉树



- 优点: 没有任何空余指针
- 缺点: 原本寻找子树的复杂度为 $O(1)$,
现在需遍历兄弟节点。

高级Trie: Radix Trie (基数树)

- 定义

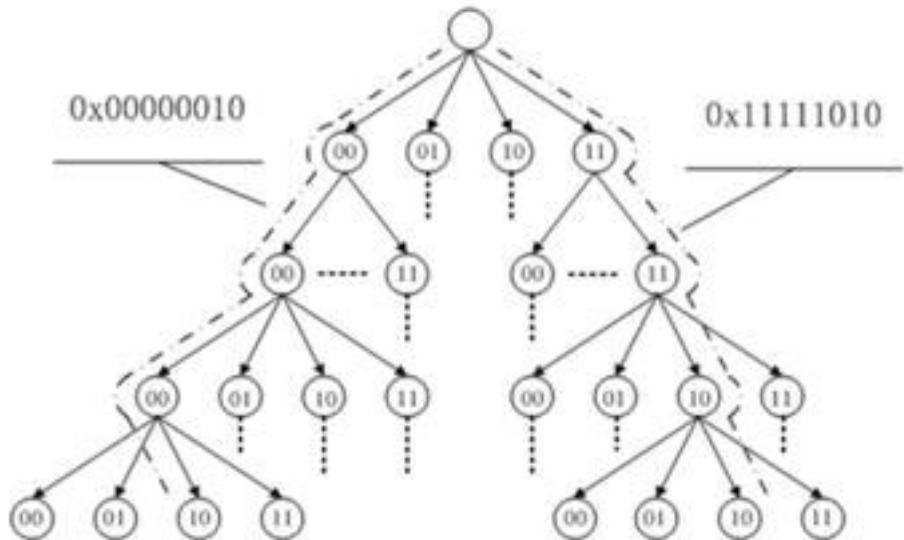
- In computer science, a radix tree (also radix trie or compact prefix tree) is a data structure that represents a space-optimized trie (prefix tree) in which each node that is the **only child is merged with its parent**. The result is that the number of children of every internal node is at most the radix r of the radix tree, where r is a positive integer and a power x of 2, having $x \geq 1$. Unlike in regular tries, **edges can be labeled with sequences of elements as well as single elements**. This makes radix trees much more efficient for small sets (especially if the strings are long) and for sets of strings that share long prefixes.

- 英文词典树

- r 是64 (大小写字母加数字)
 - 如果以二进制存储?

Linux内核的Radix Tree

- Linux内核的应用
 - 管理内存分配
 - 缓存区映射
 - R=4

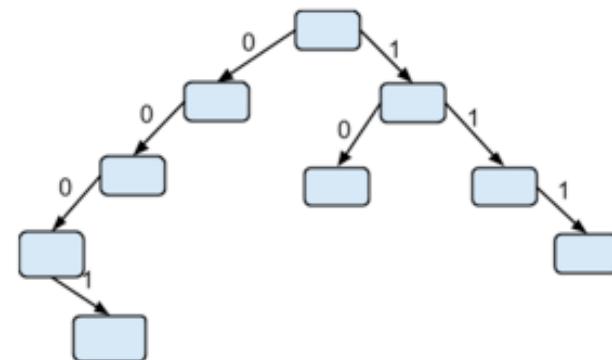


```
1 struct radix_tree_node {  
2     unsigned int    path;  
3     unsigned int    count;  
4     union {  
5         struct {  
6             struct radix_tree_node *parent;  
7             void *private_data;  
8         };  
9         struct rcu_head rCU_head;  
10    };  
11    /* For tree user */  
12    struct list_head private_list;  
13    void __rcu    *slots[RADIX_TREE_MAP_SIZE];  
14    unsigned long  tags[RADIX_TREE_MAX_TAGS][RADIX_TREE_TAG_LONGS];  
15};
```

2基数的Radix Trie: Patricia Trie

- Trie结构缺点
 - Trie结构显然也不是平衡的
 - 存取英文单词时，显然t子树下的分支比z子树下的分支多很多
 - 26个分支因子使得树的结构过于庞大，检索不便
- **PATRICIA Trie** (“Practical Algorithm To Retrieve Information Coded In Alphanumeric”)
 - 关键码二进制形式存储
 - 根据关键码每个二进制位的编码来划分
 - 是对整个关键码大小范围的划分

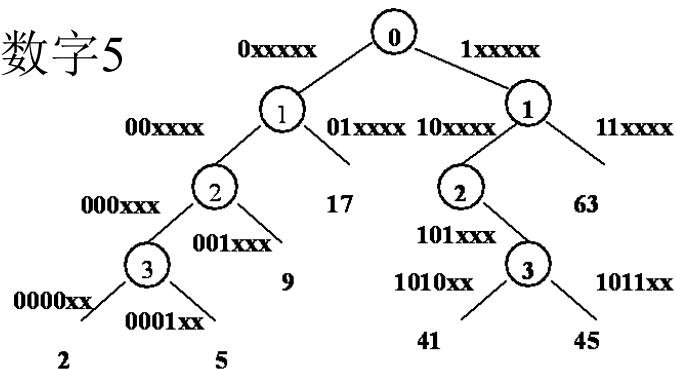
每个内部结点都代表一个位的比较，必然产生两个子结点，所以它是一个满二叉树，进行一次检索，最多只需要关键码位数次的比较即可。



PATRICIA原理

- 举例 (2、5、9、17、41、45、63)

- 因为最大的数是63，用6位二进制表示即可
- 每个结点都有一个标号，表示它是比较第几位，然后根据那一位是0还是1来划分左右两个子树
- 标号为2的结点的右子树一定是编码形式为xx1xxx，（x表示该位或0或1，标号为2说明比较第2位）
- 在图中检索5的话，5的编码为000101
- 首先我们比较第0位，从而进入左子树，然后在第1位仍然是0，还是进入左子树，在第2位还是0，仍进入左子树，第3位变成了1，从而进入右子树，就找到了位于叶结点的数字5



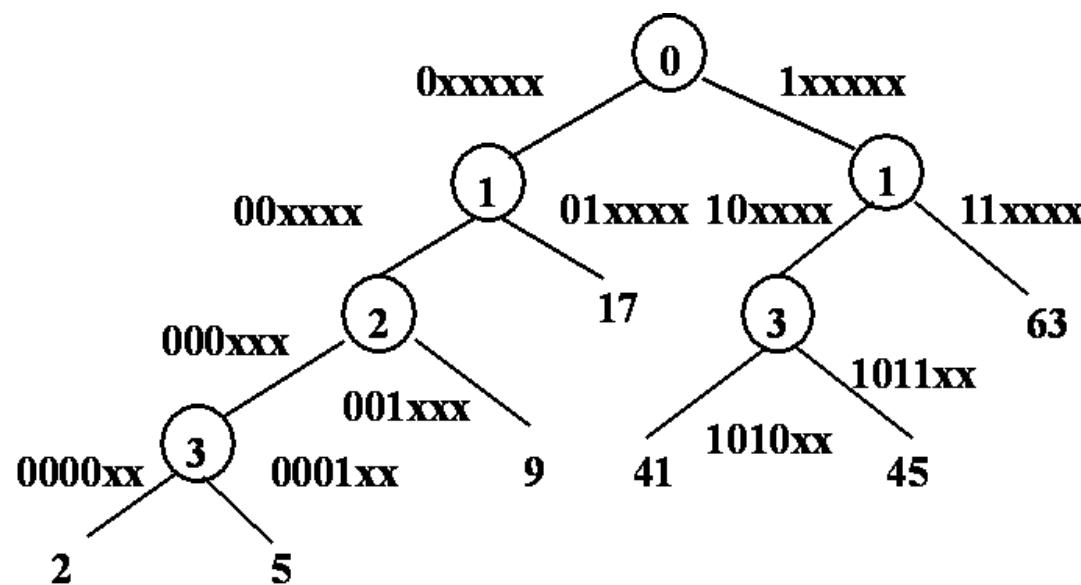
编码: 2: 000010 5: 000101 9: 001001

17: 010001 41: 101001 45: 101101 63: 111111

PATRICIA压缩优化

- 优化

- 在区分2和5、41和45时，第3个二进制位的比较不能区别它们，可以将它省略，得到一棵更为简洁的树。

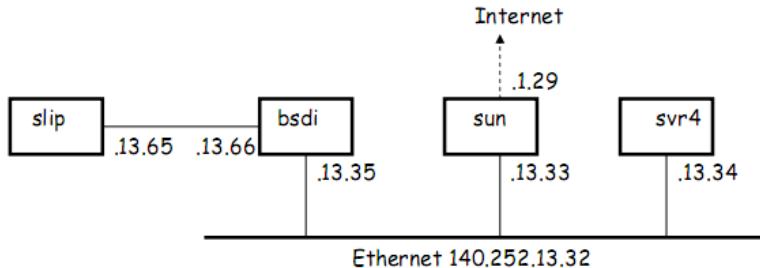


编码: 2: 000010 5: 000101 9: 001001

17: 010001 41: 101001 45: 101101 63: 111111

PATRICIA 应用：路由表查找

- Example Net



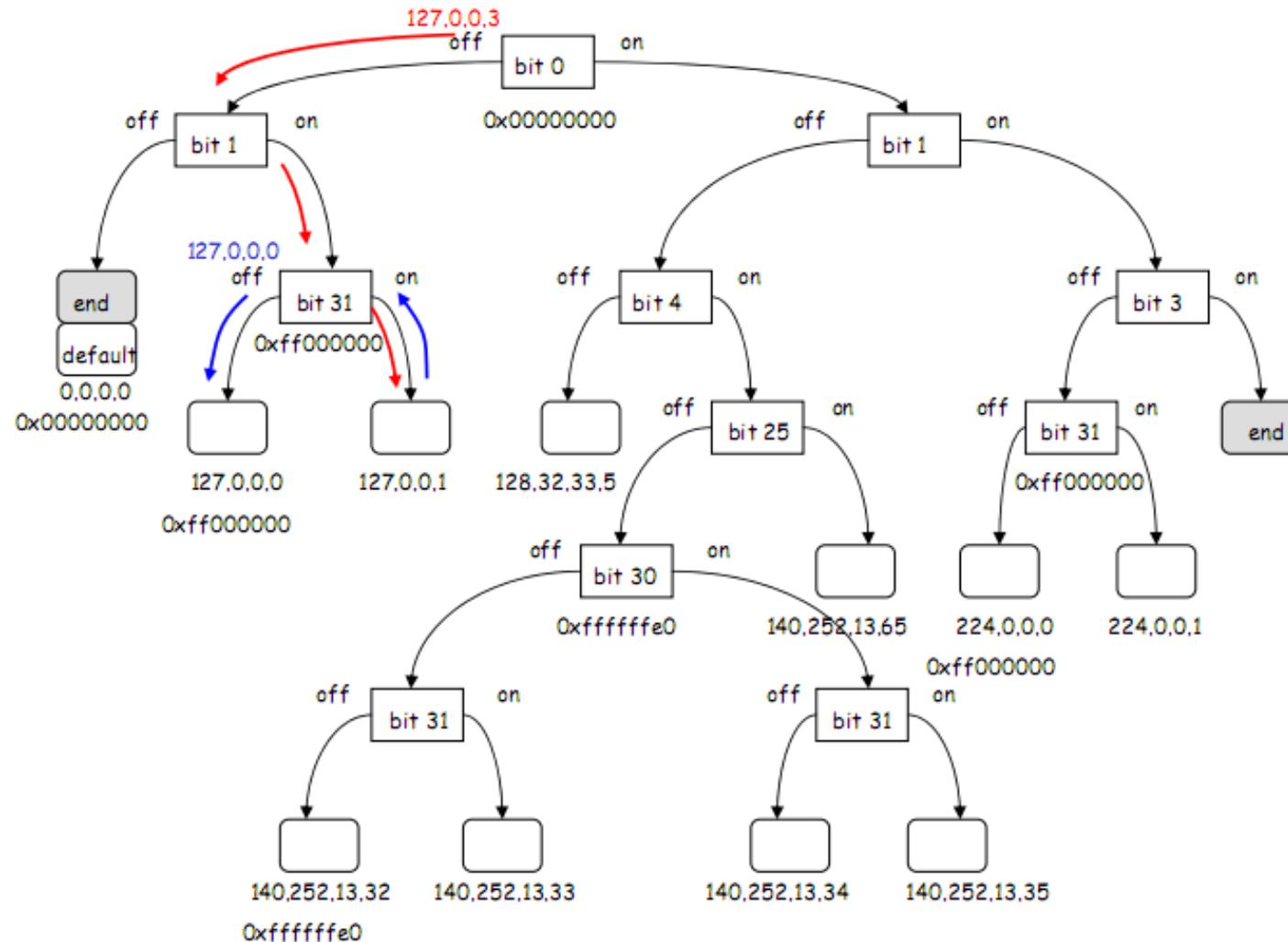
➤ Routing Table

Destination	Gateway	Flags	Ref	Use	Interface
default	140.252.13.33	UGS	0	3	1e0
127	127.0.0.1	UGSR	0	2	1o0
127.0.0.1	127.0.0.1	UH	1	55	1o0
128.32.33.5	140.252.13.33	UGHS	2	16	1e0
140.252.13.32	link#1	UC	0	0	1e0
140.252.13.33	8:0:20:3:f6:42	UHL	11	55146	1e0
140.252.13.34	0:0:c0:c2:9b:26	UHL	0	3	1e0
140.252.13.35	0:0:c0:6f:2d:40	UHL	1	12	1o0
140.252.13.65	140.252.13.66	UH	0	41	s10
224	link#1	UC	0	0	1e0
224.0.0.1	link#1	UHL	0	5	1e0

Bit:	32-bit IP address (bits 32-63)	
	0123 4567 8911 1111 1111 2222 2222 2233 01 2345 6789 0123 4 567 8901	
	0000 0000 0000 0000 0000 0000 0000 0000	0.0.0.0
	0111 1111 0000 0000 0000 0000 0000 0000	127.0.0.0
	0111 1111 0000 0000 0000 0000 0000 0001	127.0.0.1
	1000 0000 0010 0000 0010 0001 0000 0101	128.32.33.5
	1000 1100 1111 1100 0000 1101 0010 0000	140.252.13.32
	1000 1100 1111 1100 0000 1101 0010 0001	140.252.13.33
	1000 1100 1111 1100 0000 1101 0010 0010	140.252.13.34
	1000 1100 1111 1100 0000 1101 0010 0011	140.252.13.35
	1000 1100 1111 1100 0000 1101 0100 0001	140.252.13.65
	1110 0000 0000 0000 0000 0000 0000 0000	224.0.0.0
	1110 0000 0000 0000 0000 0000 0000 0001	224.0.0.1

PATRICIA应用：路由表查找

- Example: 127.0.0.3



程序要求

- 分别实现四个版本，程序分别命名为
 - btree_search (2叉树)： 2叉平衡查找树
 - bplus_search (自定义)： m阶B+树
 - radix4_search(4叉树)： 4叉01二进制树
 - patricia_search (2叉树)： 2叉二进制树

程序要求

- 输入数据
 - 词典串pattern.txt: 127万个
 - 待匹配的98万个字符串: words.txt
- 实验结果result.txt
 - 在模式串中的输出yes, 不在就输出no
 - Keyword1 yes
 - Keyword2 no
 - 最后一行输出四个数字, 用空格分割:
 - 树节点个数
 - 树结构占用内存量 (**KB**)
 - 字符比较次数 (**K**)
 - words总个数
 - 成功检索的word总个数

报告要求

- 实验报告
 - 主要数据结构和流程
 - 实验过程
 - 遇到的问题
 - 结果指标: cpu 内存 准确率等
 - 结论和总结

进度要求

- 实验进度
 - W1: 实现平衡二叉树
 - W2: 实现B+树
 - W3: 实现Radix树

THE END