10 Indexing

Indexing

Goals:

- Store large files
- Support multiple search keys
- Support efficient insert, delete, and range queries

Terms(1)

Entry sequenced file: 入口顺序文件Order records by time of insertion.

– Search with sequential search

Index file: 索引文件consists of key/pointer pairs, where each key is associated with a pointer to an actual record in the main file.

– Could be organized with a tree or other data structure.

Terms(2)

Primary Key: 主键A unique identifier for records.

May be inconvenient for search.

Secondary Key: An alternate search key, often not unique for each record. Often used for search key.

Contents

10.1 Linear Indexing

10.2 ISAM (read by yourself)

10.3 Tree-based Indexing

10.4 2-3Trees

10.5 B-Trees

Contents

10.1 Linear Indexing

10.2 ISAM (read by yourself)

10.3 Tree-based Indexing

10.4 2-3Trees

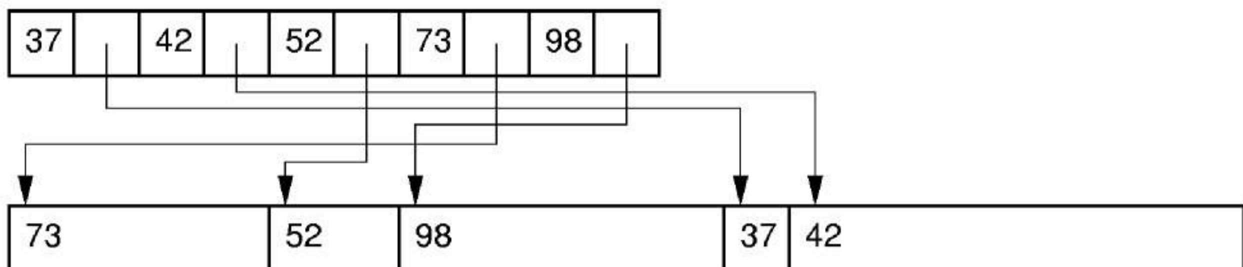
10.5 B-Trees

10.1 Linear Indexing
线性索引

Linear Indexing

Linear index: Index file organized as a simple sequence of key/record pointer pairs with key values are in sorted order.

Linear indexing is good for searching variable-length records, and good for indexing an entry sequenced file.



Linear Indexing (2)

If the index is too large to fit in main memory, a second-level index might be used.

1	2003	5894	10528

Second Level Index

1 2001	2003 5688	5894 9942	10528 10984

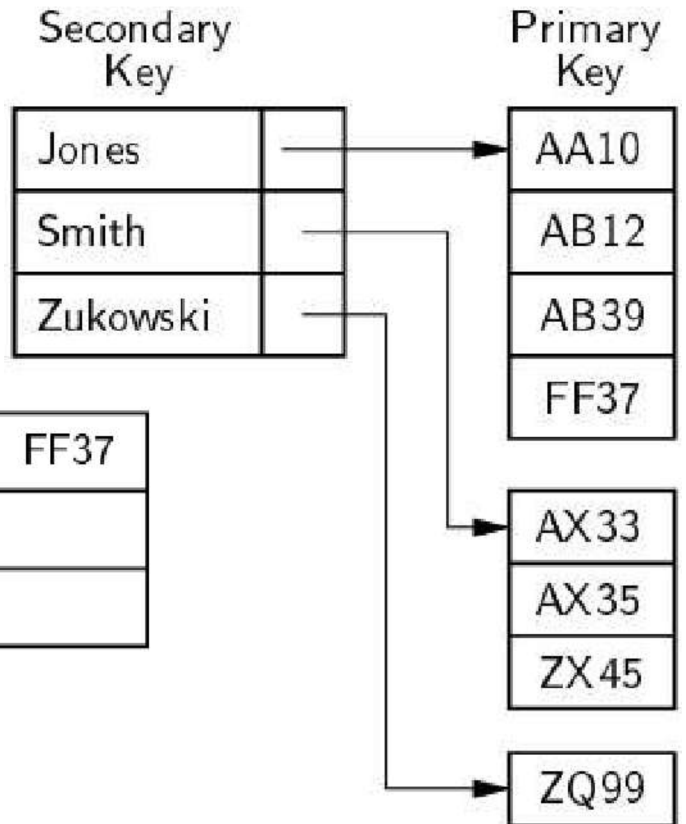
Linear Index: Disk Pages

Inverted List(1)

- 倒排表
- Inverted list is another term for a secondary index. A secondary key is associated with a list of primary keys, which in turn locate the records.
- It is inverted in that searches work backwards from the secondary key to the primary key, then to the actual data record.

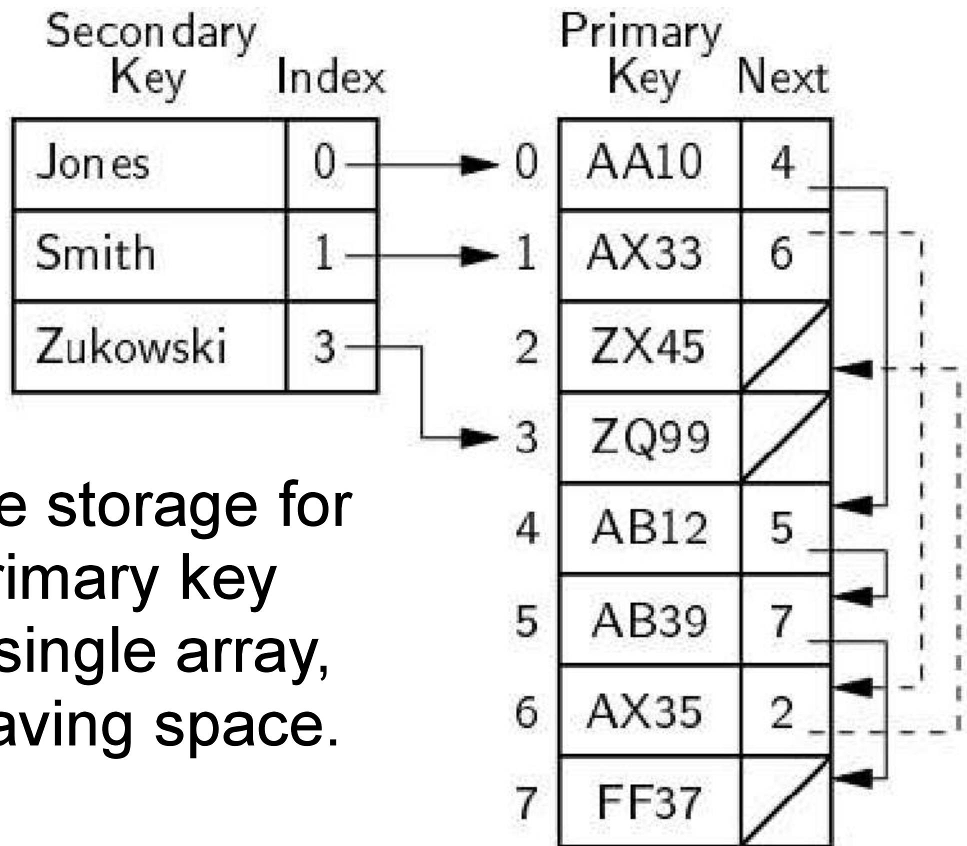
Inverted List(2)

Each secondary key value has a list of primary keys associated with it.



Jones	AA10	AB12	AB39	FF37
Smith	AX33	AX35	ZX45	
Zukowski	ZQ99			

Inverted List(3)



Combines the storage for all of the primary key lists into a single array, probably saving space.

Contents

10.1 Linear Indexing

10.2 ISAM (read by yourself)

10.3 Tree-based Indexing

10.4 2-3Trees

10.5 B-Trees

10.3 Tree-based Indexing

Tree Indexing (1)

Linear index is poor for insertion/deletion.

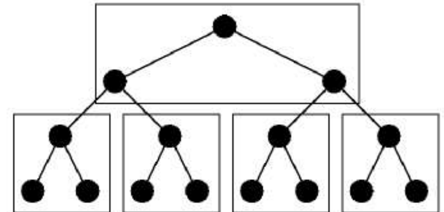
Tree index can efficiently support all desired operations:

- Insert/delete
- Multiple search keys (multiple indices)
- Key range search

Tree Indexing (2)

Difficulties when storing tree index on disk:

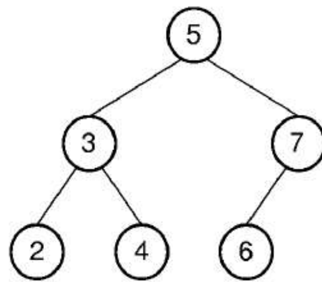
- Tree must be balanced.
- Each path from root to leaf should cover few disk pages.



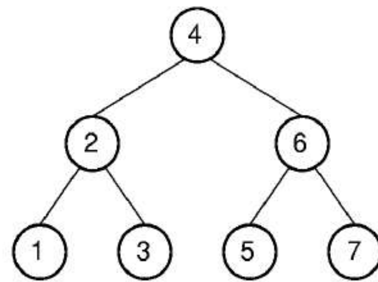
Example:

The path from the root to any leaf is contained on two blocks.

Tree Indexing (3)



(a)



(b)

Insert 1 to the BST and keep it balanced.

It is difficult to keep a tree balanced.

In this case, rebalancing the BST requires that all nodes be moved. It is too expensive.

Solution: Use new tree structure.

Contents

10.1 Linear Indexing

10.2 ISAM (read by yourself)

10.3 Tree-based Indexing

10.4 2-3Trees

10.5 B-Trees

10.4 2-3 Tree

2-3 Tree (1)

A 2-3 Tree has the following properties:

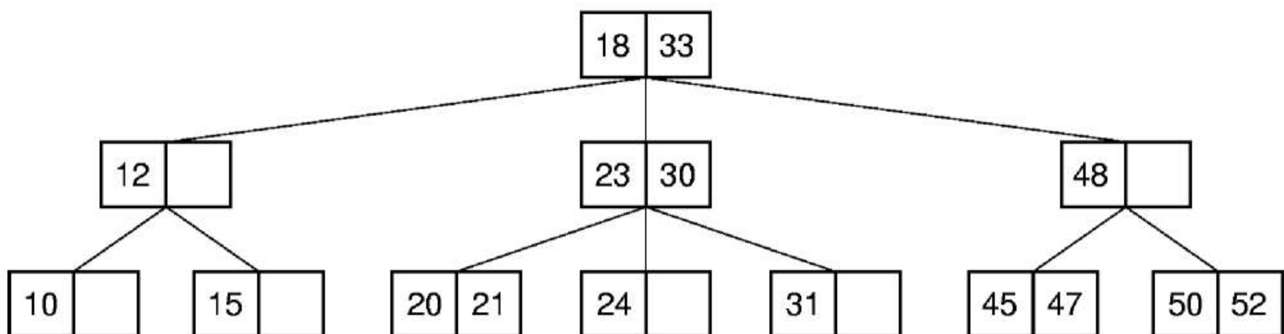
1. A node contains one or two keys
2. Every internal node has either two children (if it contains one key) or three children (if it contains two keys).
3. All leaves are at the same level in the tree, so the tree is always height balanced.

The 2-3 Tree has a search tree property analogous to the BST.

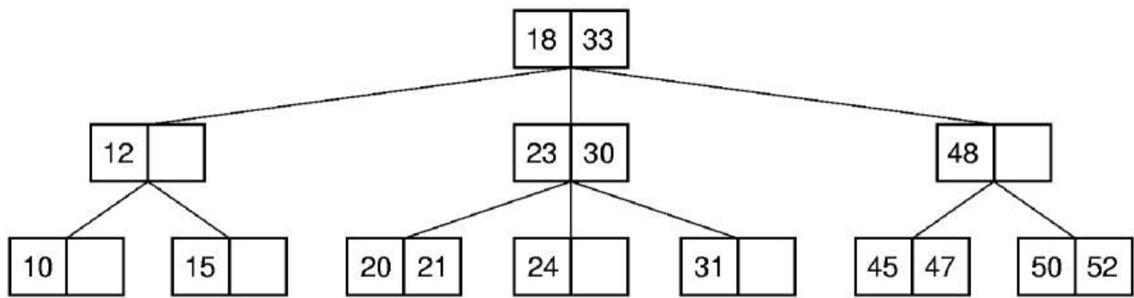
2-3 Tree (2)

The advantage of the 2-3 Tree over the BST is that it can be updated at low cost.

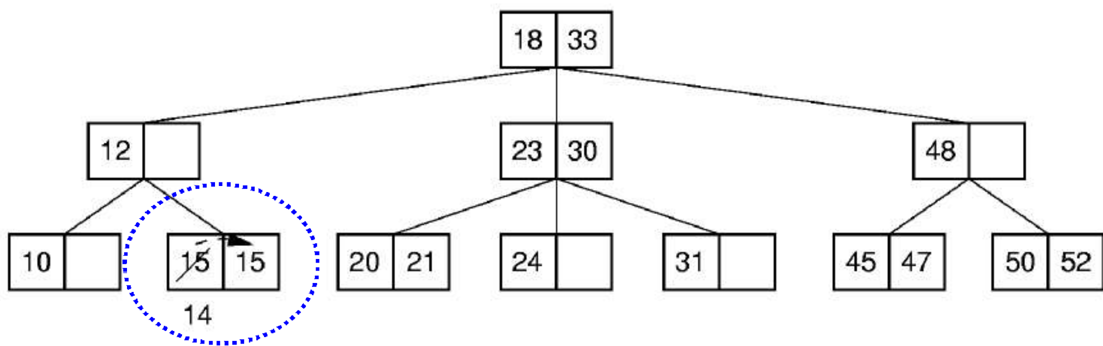
Example:



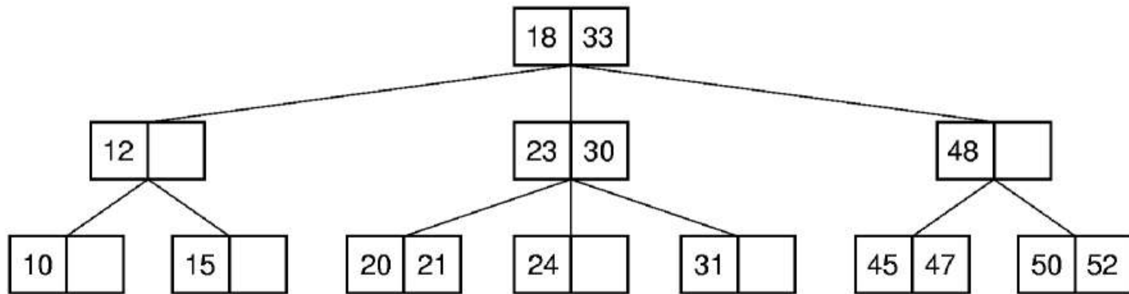
2-3 Tree Insertion (1)



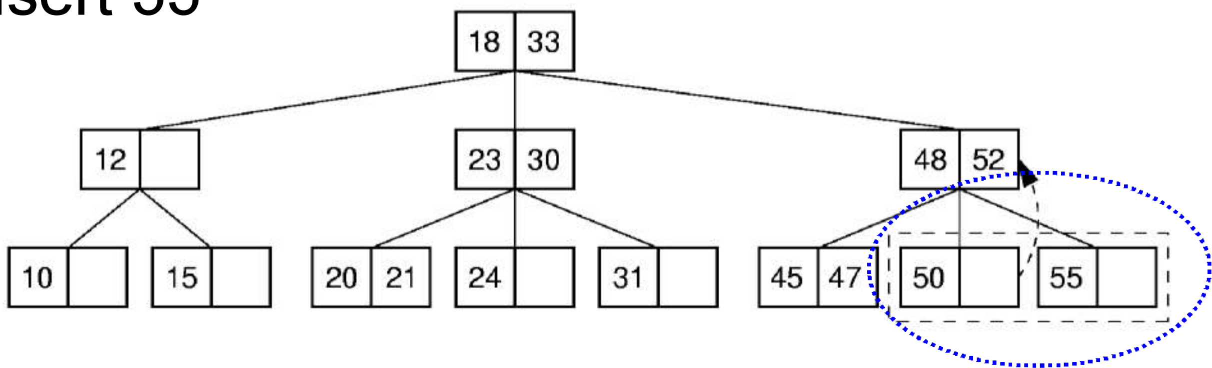
Insert 14



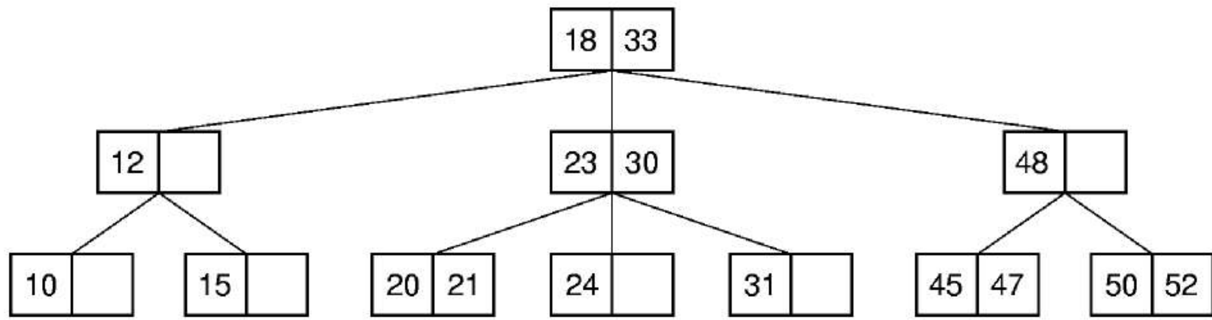
2-3 Tree Insertion (2)



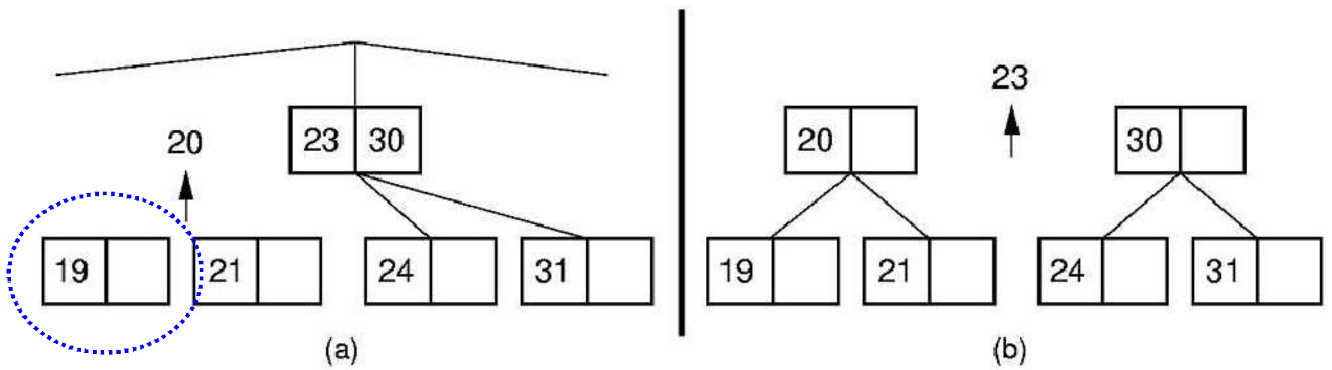
Insert 55



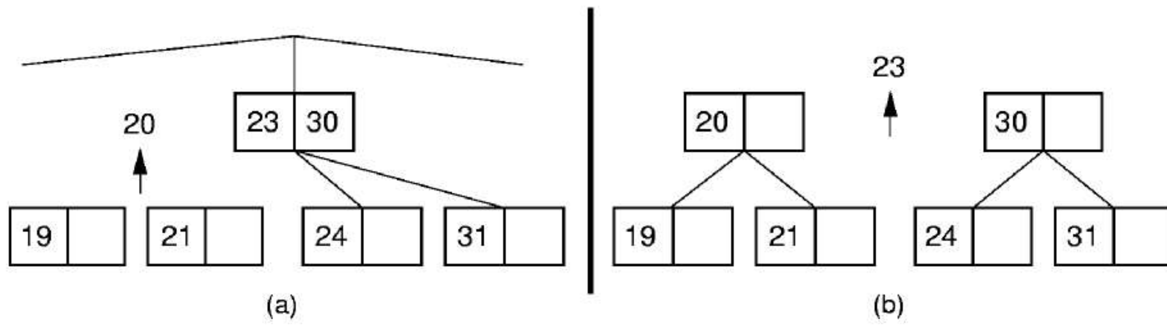
2-3 Tree Insertion (3)

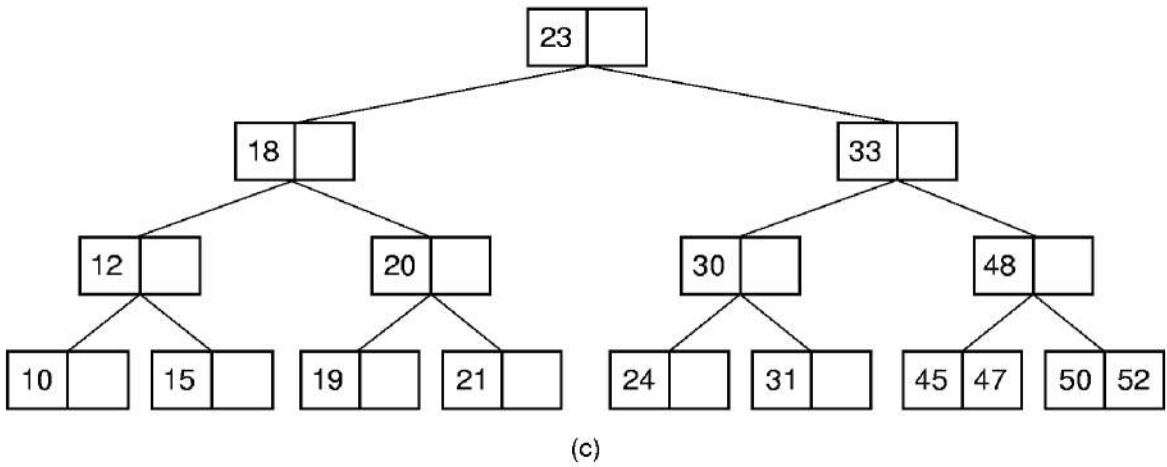


Insert 19



2-3 Tree Insertion (4)





Contents

10.1 Linear Indexing

10.2 ISAM (read by yourself)

10.3 Tree-based Indexing

10.4 2-3Trees

10.5 B-Trees

10.5 B-Trees

B-Trees (1)

The B-Tree is an extension of the 2-3 Tree.

The B-Tree is now **the** standard file organization for applications requiring insertion, deletion, and key range searches.

B-Trees (2)

B-Trees address all of the major problems encountered when implementing disk based search trees:

- B-Trees are always balanced.
- B-Trees keep similar-valued records together on a disk page, which takes advantage of locality of reference.
- B-Trees guarantee that every node in the tree will be full at least to a certain minimum percentage.

B-Tree Definition

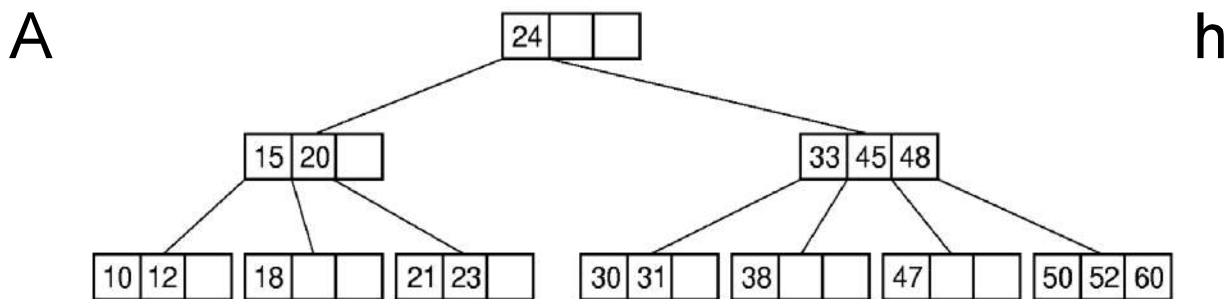
A B-Tree of order *m* has these properties:

– The root is either a leaf or has at least two children.

– Each node, except for the root and the leaves, has between $\lceil m/2 \rceil$ and *m* children.

– All leaves are at the same level in the tree, so the tree is always height balanced.

A

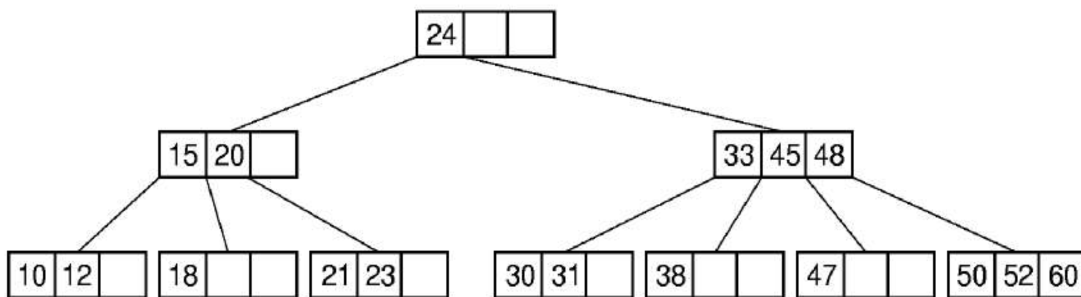


h

B-Tree Search

Search in a B-Tree is a generalization of search in a 2-3 Tree.

1. Do binary search in current node. If found, then return record. If current node is a leaf node and key is not found, then report an unsuccess.
2. Otherwise, follow the proper branch and repeat the process.



A B-tree of order four

B^+-Trees

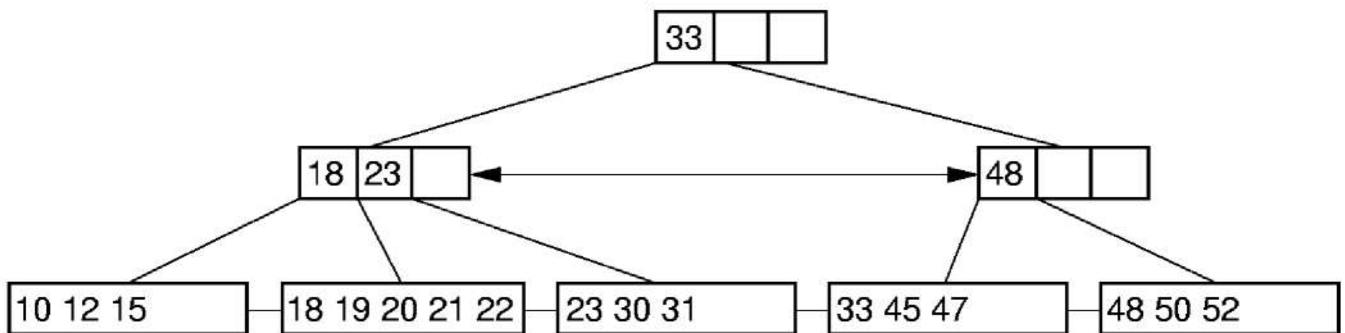
The most commonly implemented form of the B-Tree is the B^+-Tree.

Internal nodes of the B^+-Tree do not store record -- only key values to guide the search.

Leaf nodes store records or pointers to records.

A leaf node may store more or less records than an internal node stores keys.

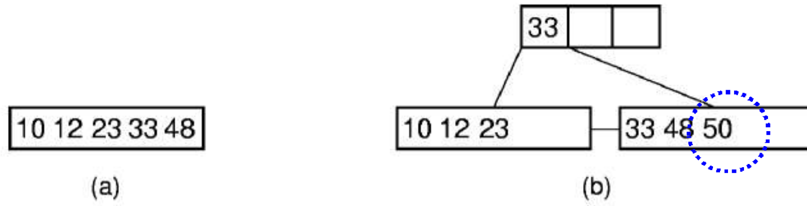
B^+-Tree Example



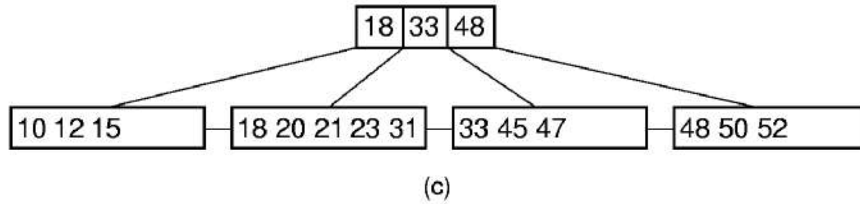
An example of B^+-Tree of order four

The leaf nodes of a B^+-Tree are normally linked together to form a doubly linked list.

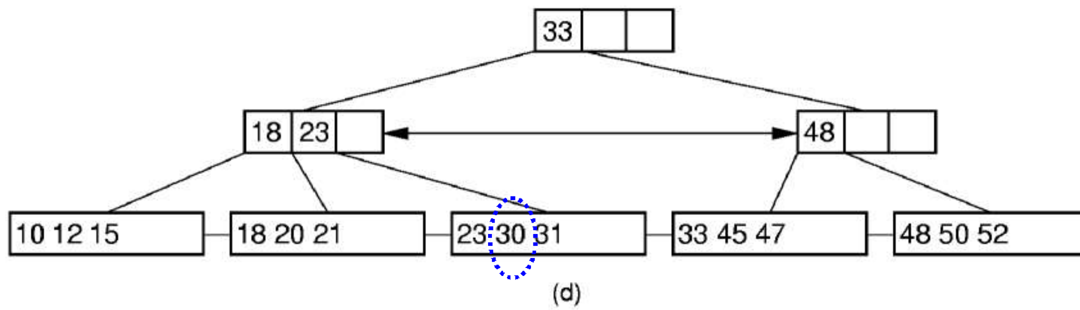
B+-Tree Insertion



(a) (b)

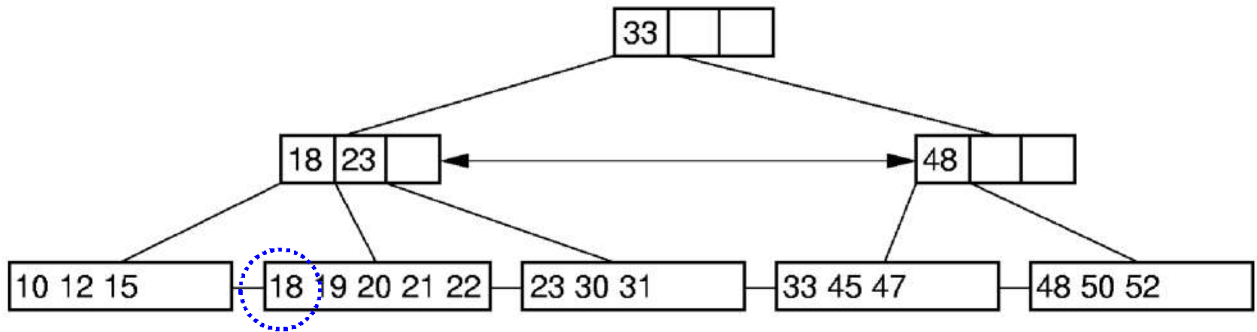


(c)

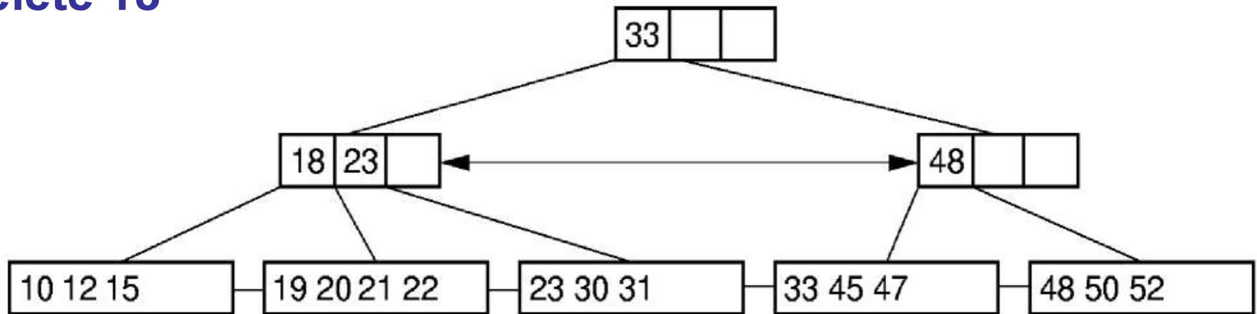


(d)

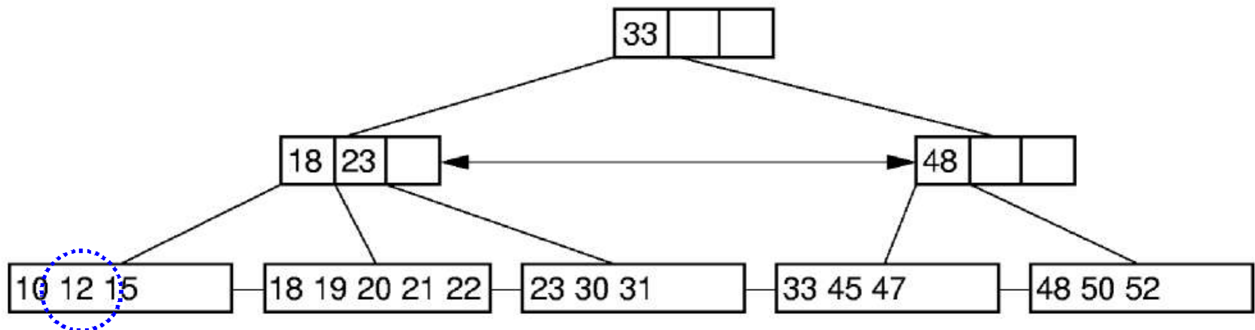
B+-Tree Deletion (1)



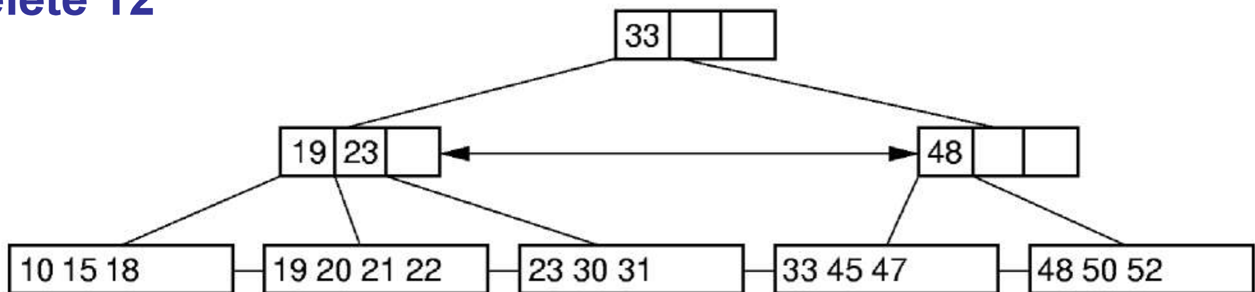
Delete 18



B^+-Tree Deletion (2)

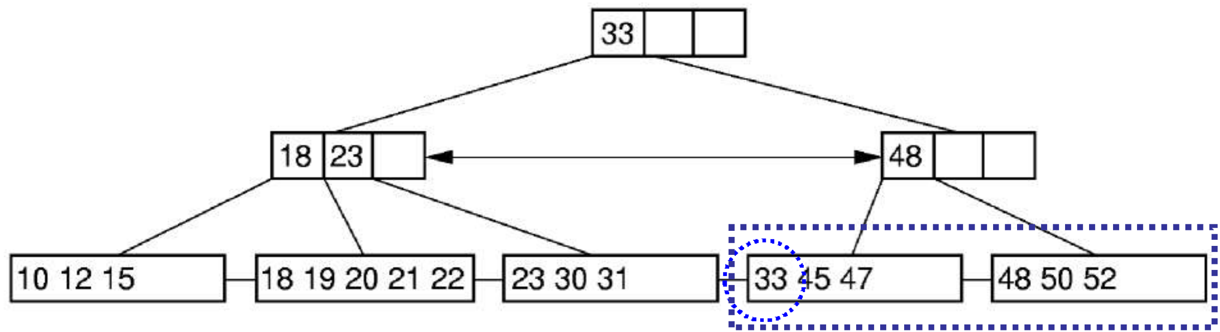


Delete 12

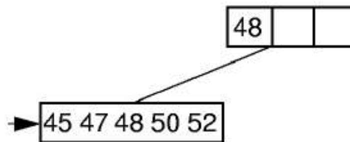


Borrow 18 from the second child

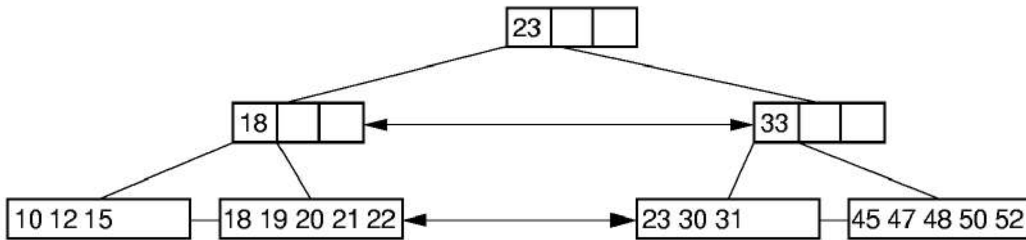
B$^+$-Tree Deletion (3)



Delete 33



(a)



(b)

B-Tree Space Analysis (1)

B^+-Trees nodes are always at least half full.

Asymptotic cost of search, insertion, and deletion of nodes from B-Trees is $\Theta(\log n)$.

- Base of the log is the (average) branching factor of the tree.

B-Tree Space Analysis (2)

Minimum and maximum number of records that can be stored by a B+-Tree.

Example: Consider a B+-Tree of order 100 with leaf nodes containing 100 records.

- 1 level B+-tree: Min 0, Max 100
- 2 level B+-tree:
  - Min: 2 leaves of 50 for 100 records.
  - Max: 100 leaves with 100 for 10,000 records
- 3 level B+-tree:
  - Min: 2 x 50 nodes of leaves, for 5000 records.
  - Max: 100^3 = 1,000,000 records
- 4 level B+-tree:
  - Min: 250,00 records (2 * 50 * 50 * 50).
  - Max: 100^4 = 100 million records

Homework

10.9

10.11

10.12

10.13

补充材料：

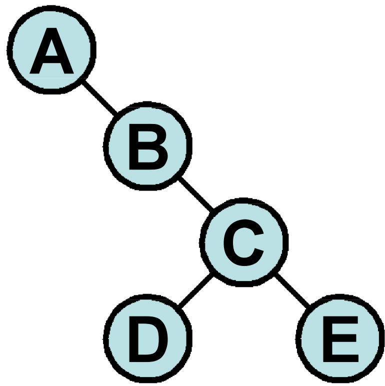
二叉平衡树

（AVL树）

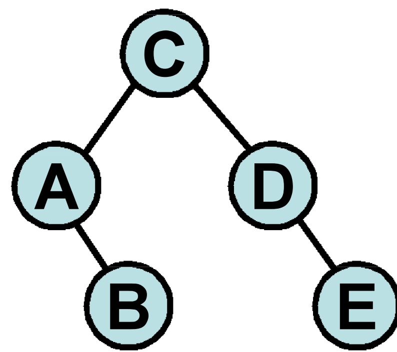
Named for its inventors

Adelson-Velskii and Landis

一棵AVL树或者是空树，或者是具有下列性质的BST：它的左子树和右子树都是AVL树，且左子树和右子树的高度之差的绝对值不超过1。



不平衡



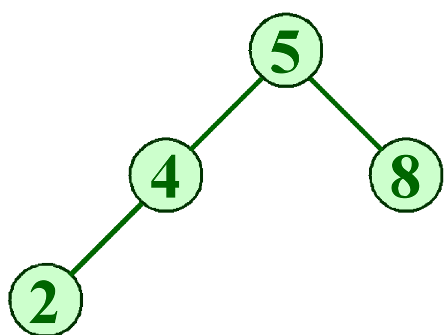
平衡

结点的平衡因子 (balance factor)

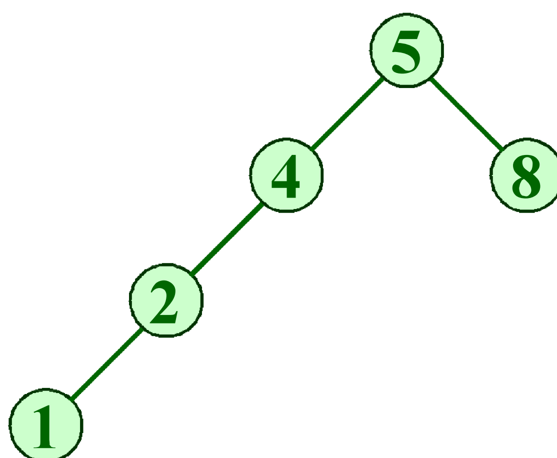
每个结点附加一个数字，给出该结点右子树的高度减去左子树的高度所得的高度差，这个数字即为结点的平衡因子。

AVL树任一结点平衡因子只能取

−1, 0, 1



是平衡树



不是平衡树

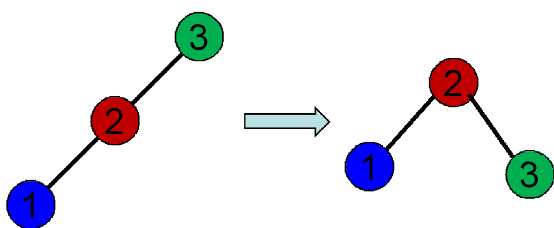
平衡化旋转

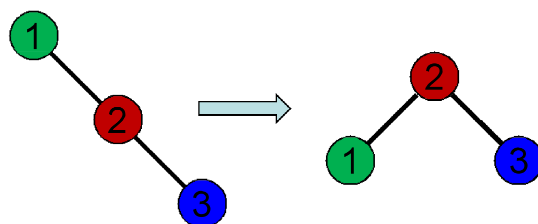
- 如果在一棵AVL树中插入一个新结点，造成了不平衡。此时必须调整树的结构，使之平衡化。
- 平衡化旋转有两类：
  - 单旋转（左旋和右旋）
  - 双旋转（左旋加右旋和右旋加左旋）

- 每插入一个新结点时，AVL树中相关结点的平衡状态会发生改变。因此，在插入一个新结点后，需要从插入位置沿通向根的路径回溯，检查各结点的平衡因子。

- 如果在某一结点发现高度不平衡，停止回溯。从发生不平衡的结点起，沿回溯的路径取直接下两层的结点。

- **如果这三个结点处于一条直线上，则采用单旋转进行平衡化**　　单旋转可按其方向分为左单旋转和右单旋转，其中一个是另一个的镜像，其方向与不平衡的形状相关；

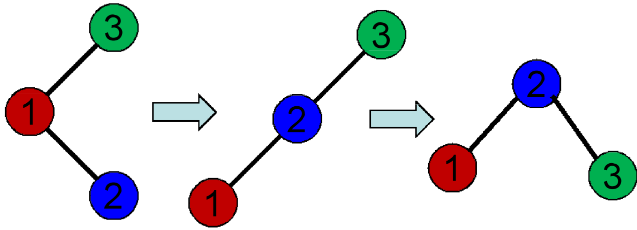


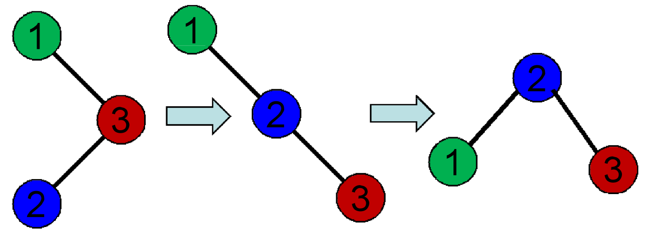


右单旋转　　　　　　　　**左单旋转**

LL型　　　　　　　　　**RR型**

- <u>如果这三个结点处于一条折线上，则采用双旋转进行平衡化</u>　　双旋转分为先左后右和先右后左两类。





左右双旋转

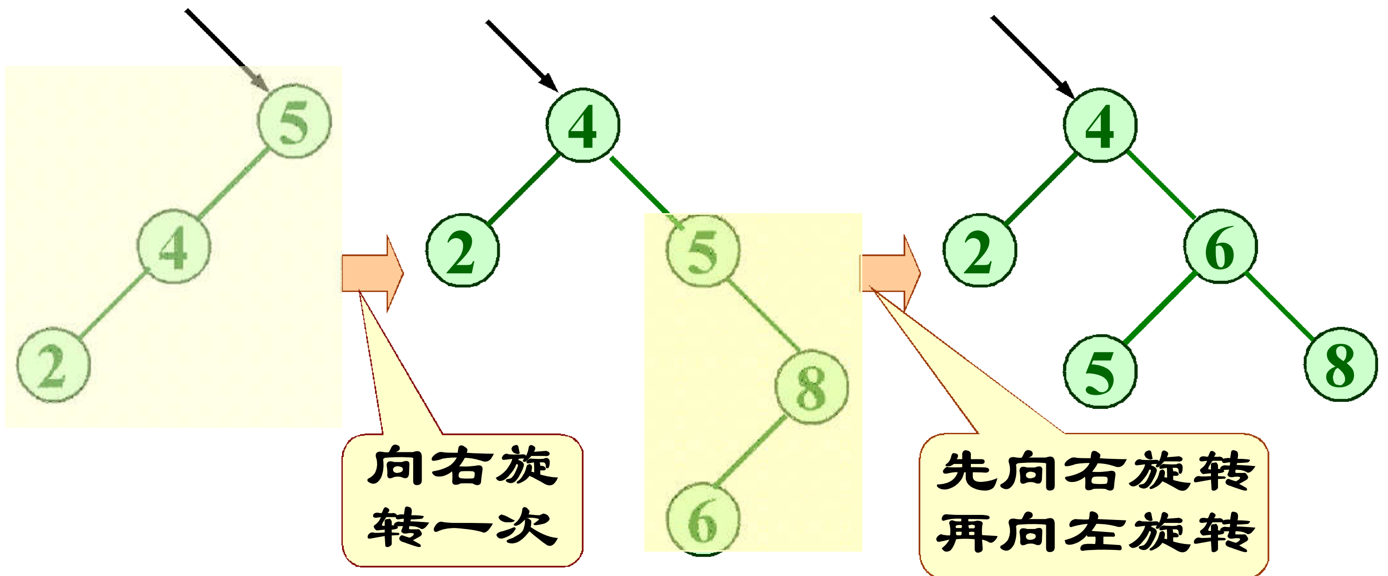
LR型

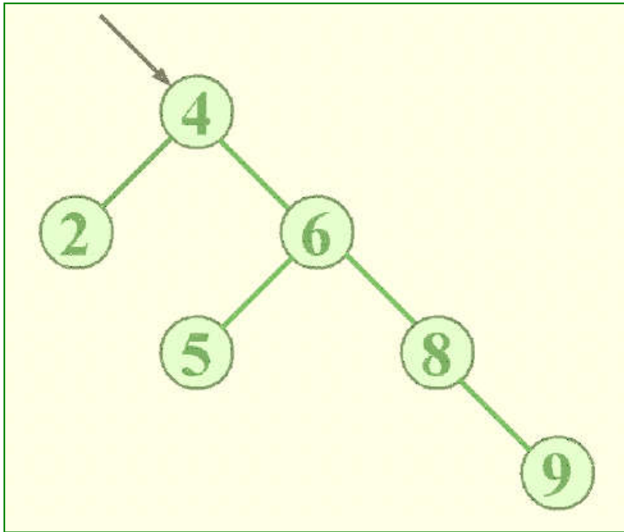
右左双旋转

RL型

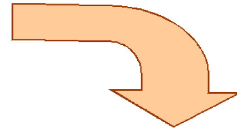
举例：构造AVL树

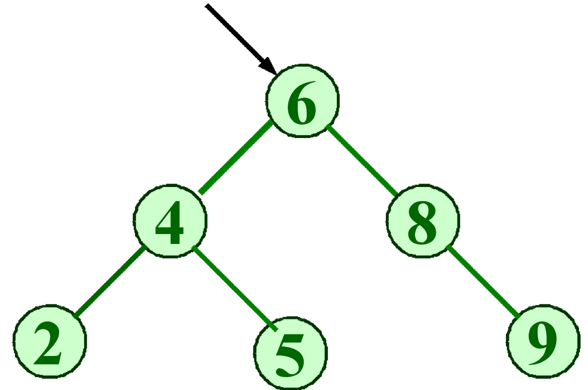
依次插入关键字：5, 4, 2, 8, 6, 9





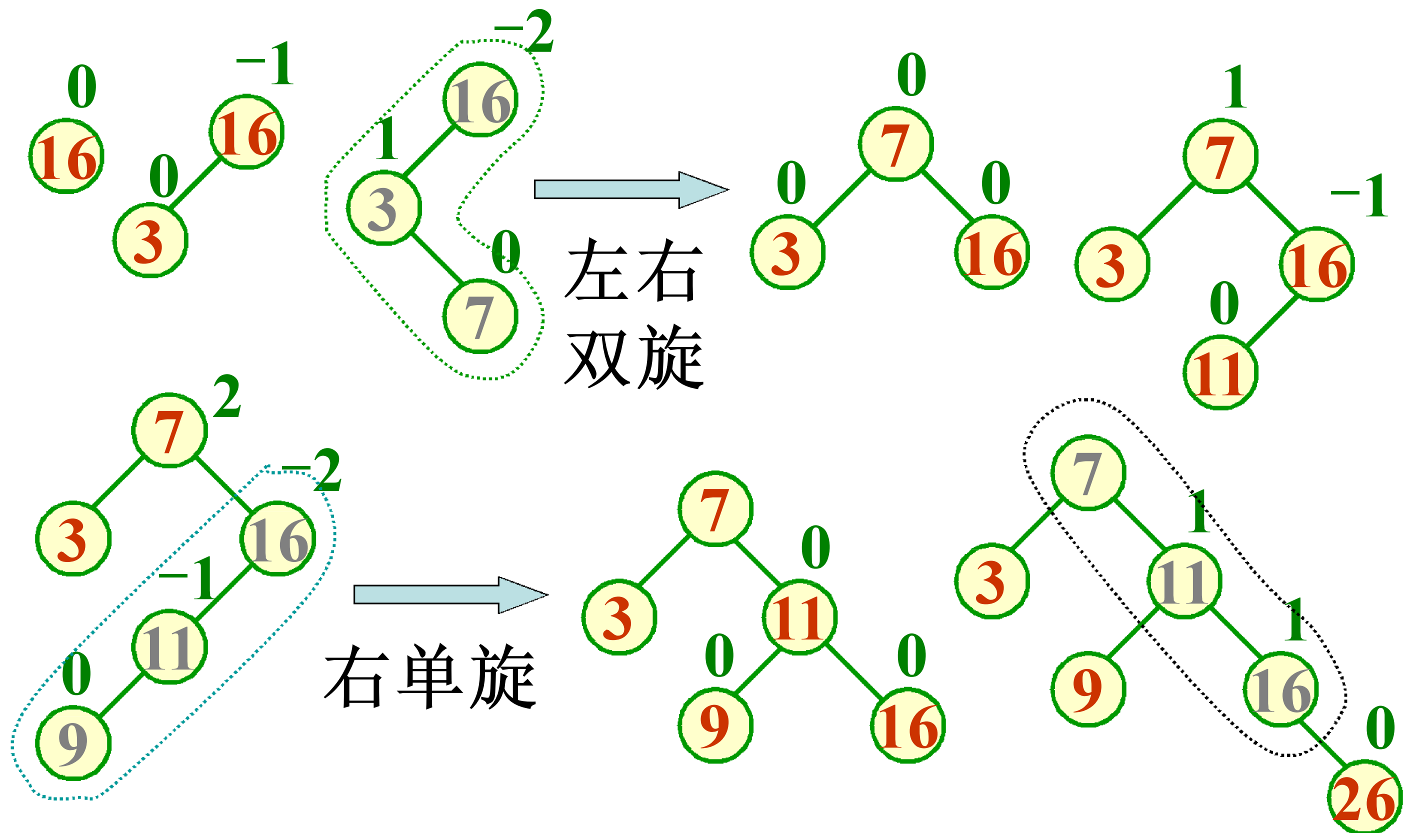
向左旋转一次



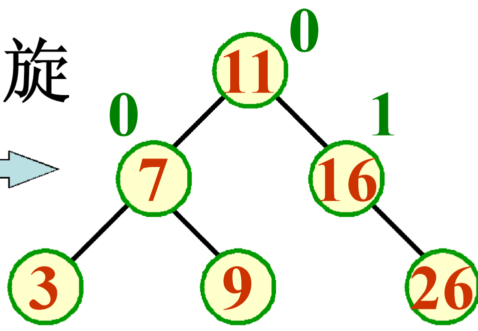


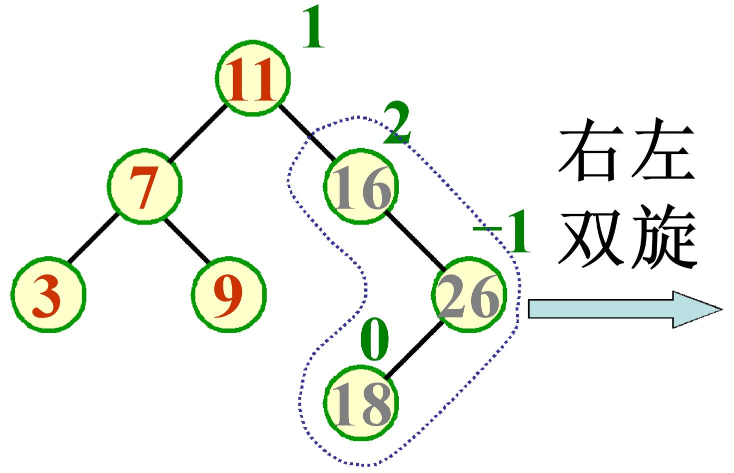
继续插入关键字 9

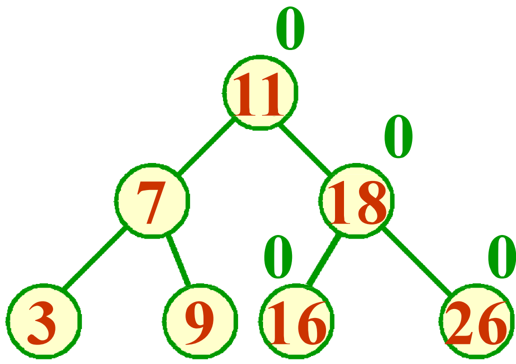
输入关键码序列为 { 16, 3, 7, 11, 9, 26, 18, 14, 15 } 插入和调整过程如下

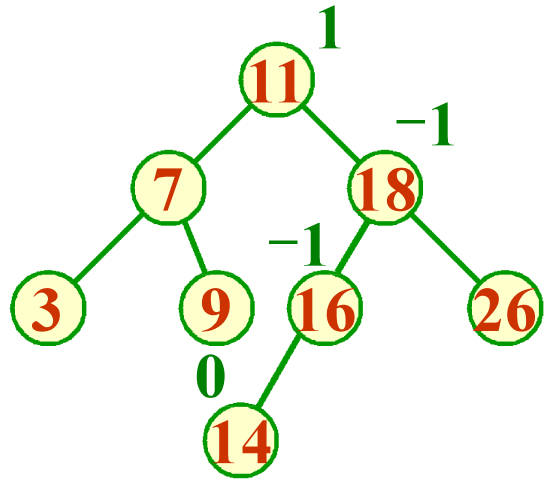


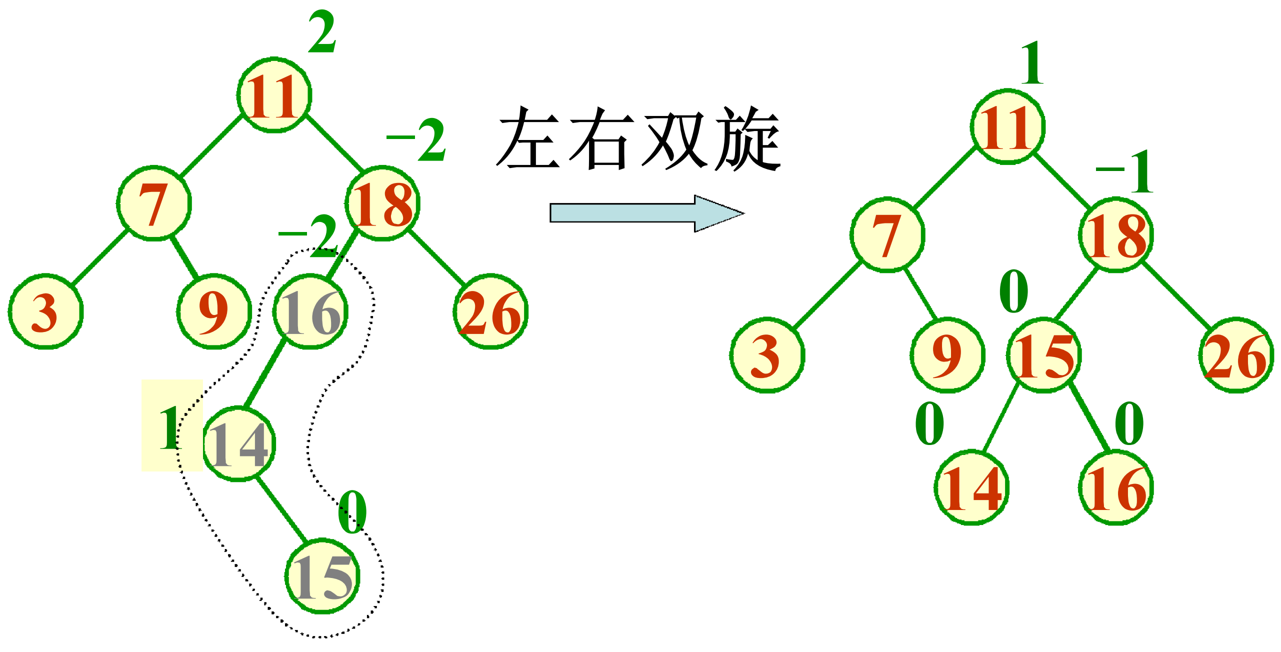
左单旋










2
11
−2
7
18
−2
3
9
16
26
1
14
0
15
左右双旋
1
11
−1
7
18
0
3
9
15
26
0
0
14
16

The End

Question?