

数据结构与算法

第十章 索引技术

任课教员：张 铭

<http://db.pku.edu.cn/mzhang/DS/>

mzhang@db.pku.edu.cn

北京大学信息科学与技术学院

网络与信息系统研究所

©版权所有，转载或翻印必究



主要内容

◆ 基本概念

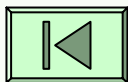
◆ 10.1 线性索引

◆ 10.2 静态索引

◆ 10.3 倒排索引

◆ 10.4 动态索引

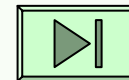
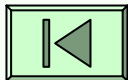
◆ 10.5 动态、静态索引性能比较





基本概念

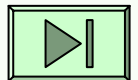
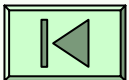
- ◆ 输入顺序文件
- ◆ 主码与辅码
- ◆ 索引与索引文件
- ◆ 稠密索引与稀疏索引





输入顺序文件

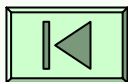
- 输入顺序文件(**entry-sequenced file**)
按照记录进入系统的顺序存储记录
 - 一般说来，输入顺序文件的结构相当于一个磁盘中未排序的线性表
 - 因此不支持高效率的检索





主码

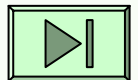
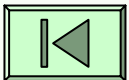
- 主码(**primary key**)是数据库中的每条记录的唯一标识
 - 例如，公司职员信息的记录的主码可以是职员的身份证号码
 - 如果只有主码，不便于各种灵活检索

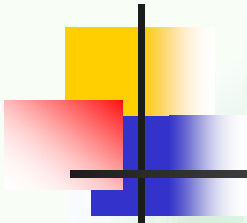




辅码

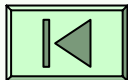
- 辅码(**secondary key**)是数据库中可能出现重复值的码
- 辅码索引把一个辅码值与具有这个辅码值的每一条记录的主码值关联起来
 - 大多数检索都是利用辅码索引来完成的





索引

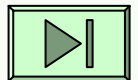
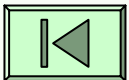
- **索引(indexing)**是把一个关键码与它对应的数据记录的位置相关联的过程
 - (关键码, 指针)对, 即(key, pointer)
 - 指针指向主要数据库文件(也称为“主文件”)中的完整记录
- **索引文件(index file)**是用于记录这种联系的文件组织结构
- **索引技术**是组织大型数据库的一种重要技术
 - 高效率的检索
 - 插入、更新、删除





索引文件

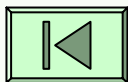
- 一个主文件可能有多个相关索引文件
 - 每个索引文件往往支持一个关键码字段
 - 不需要重新排列重排主文件
- 可以通过该索引文件高效访问记录中该关键码值





稠密索引 vs 稀疏索引

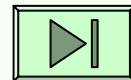
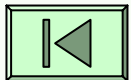
- 稠密索引：对每个记录建立一个索引项
 - 主文件不按照关键码的顺序排列
- 稀疏索引：对一组记录建立一个索引
 - 记录按照关键码的顺序存放
 - 可以把记录分成多个组（块）
 - 索引指针指向的这一组记录在磁盘中的起始位置





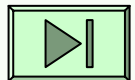
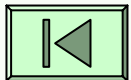
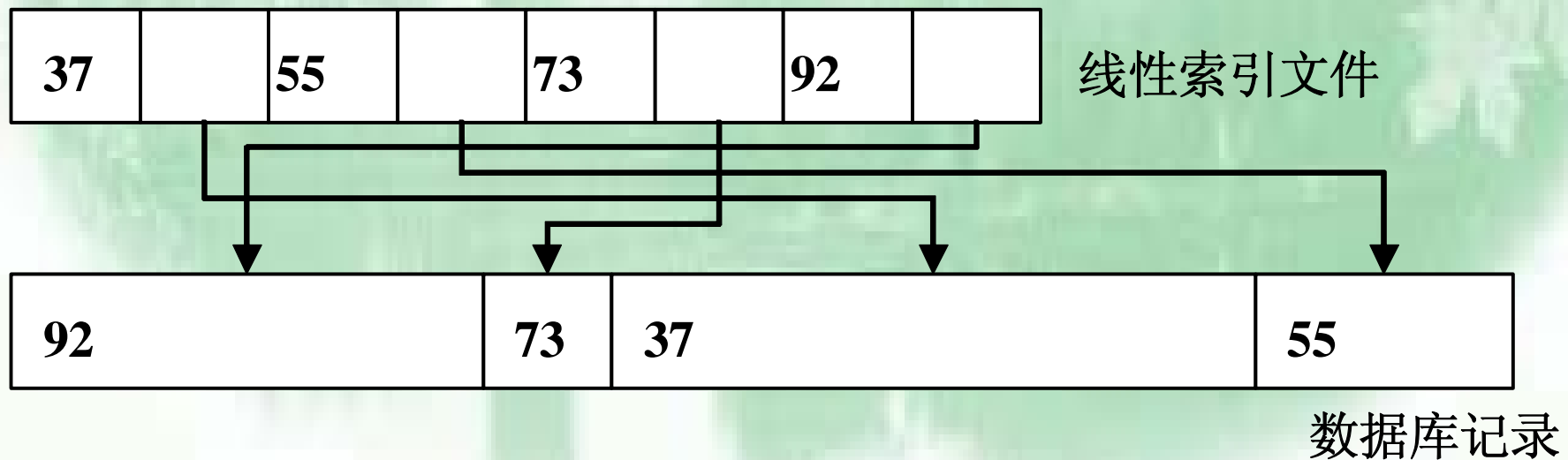
10.1 线性索引

- ◆ 基本概念
- ◆ 线性索引的优点
- ◆ 线性索引的问题
- ◆ 二级线性索引



线性索引文件

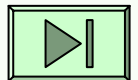
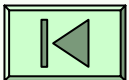
- 按照关键码的顺序进行排序
- 文件中的指针指向存储在磁盘上的文件记录起始位置或者主索引中主码的起始位置





线性索引的优点

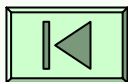
- 对变长的数据库记录的访问
- 可以对数据进行高效检索
 - 二分检索
- 顺序处理
 - 比较操作
 - 批处理的操作
- 节省空间（相对其它索引结构）





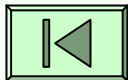
线性索引的问题

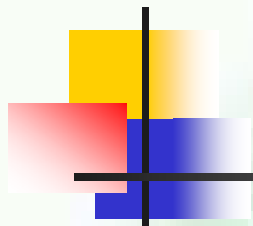
- 线性索引太大，存储在磁盘中
 - 在一次检索过程中可能多次访问磁盘，从而影响检索的效率
 - 使用二级线性索引
- 更新线性索引
 - 在数据库中插入或者删除记录时



二级线性索引

- 例如，磁盘块的大小是**1024**字节，每对(关键码，指针)索引对需要**8**个字节
 - $1024 / 8 = 128$
 - 每磁盘块可以存储**128**条这样的索引对
- 假设数据文件包含**10000**条记录
 - 稠密一级线性索引中包含**10000**条记录
 - $10000/128 = 78.1$
 - 那么一级线性索引占用**79**个磁盘块
 - 相应地，二级线性索引文件中有**79**项索引对
 - 这个二级线性索引文件可以在一个磁盘块





二级线性索引的例子

- 关键码与相应磁盘块中第一条记录的关键码的值相同
- 指针指向相应磁盘块的起始位置

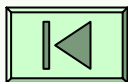
二级索引

1	2003	5744	10723
---	------	------	-------	-------

一级索引

1.....	2002	2003	5583	5744	9297	10723	13293
.....							

磁盘块



例如：检索关键码为**2555**的记录

关键字2555的记录指针

二级索引

1	2003	5744	10723
---	------	------	-------	-------

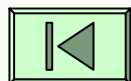
线性索引

1	2002	2003	5583	5744	9297	10723	13293
.....							

磁盘块

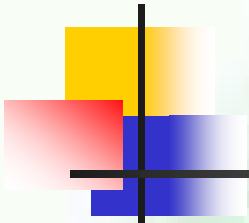
关键码为2555的记录

1. 二级线性索引文件读入内存
2. 二分法找关键码的值小于等于**2555**的最大关键码所在一级索引磁盘块地址——**关键码为2003**的记录
3. 根据记录**2003**中的地址指针找到其对应的一级线性索引文件的磁盘块，并把该块读入内存
4. 按照二分法对该块进行检索，找到所需要的记录在磁盘上的位置



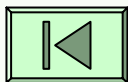
5. 最后把所需记录读入，完成检索操作





10.2 静态索引

- ◆ 基本概念
- ◆ 10.2.1 多分树
- ◆ 10.2.2 ISAM

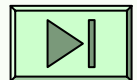
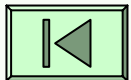


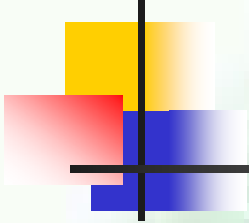


10.2.1 基本概念

静态索引

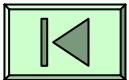
- 索引结构在文件创建、初始装入记录时生成
- 一旦生成就固定下来，在系统运行(例如插入和删除记录)过程中索引结构并不改变
- 只有当文件再组织时才允许改变索引结构





10.2.2 多分树

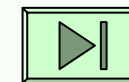
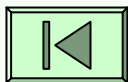
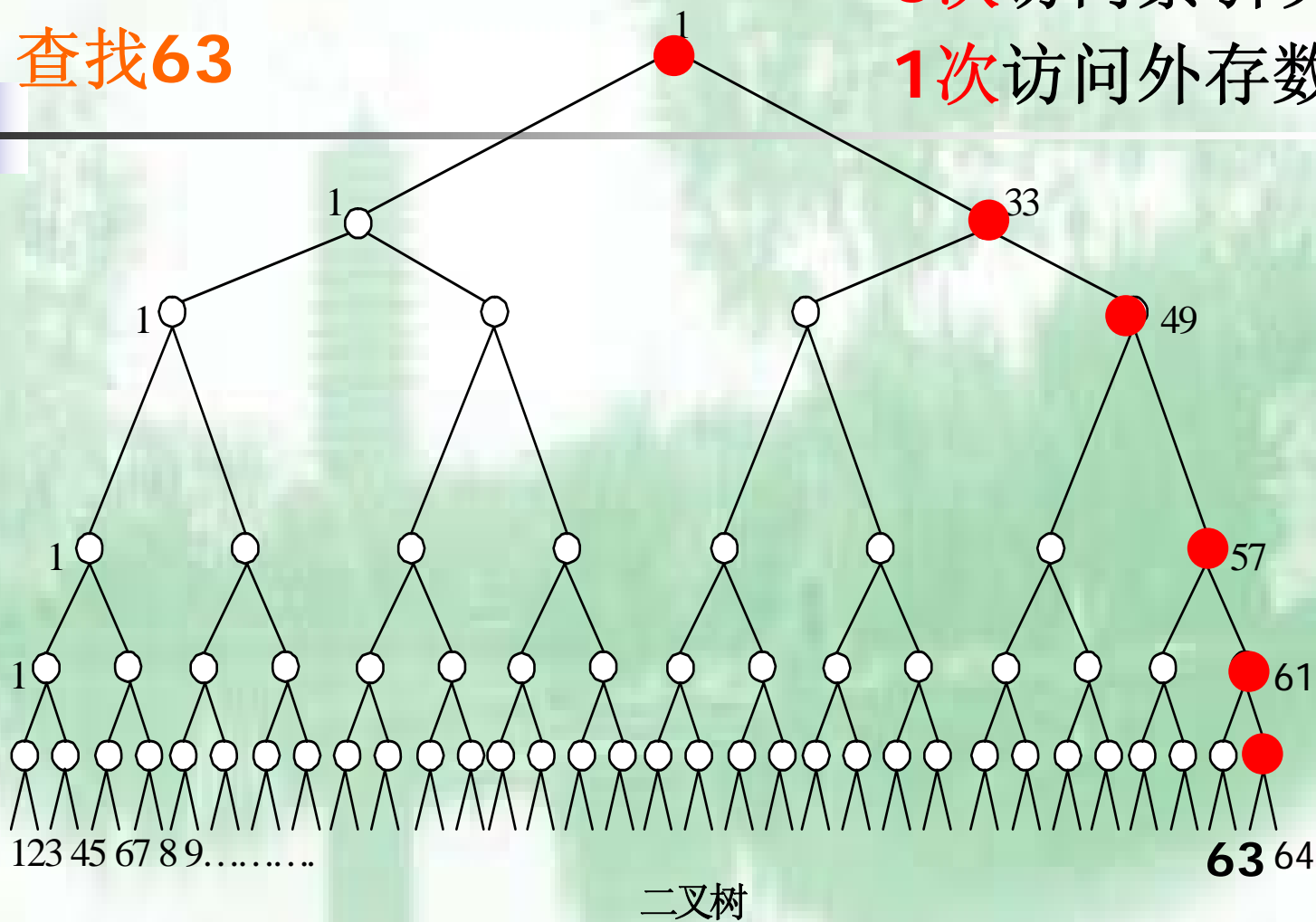
- 组织索引一般不用二叉树而采用多分树
- 大大减少访问外存的次数



数据1,2...63,64的二分索引

查找63

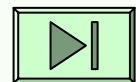
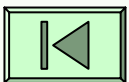
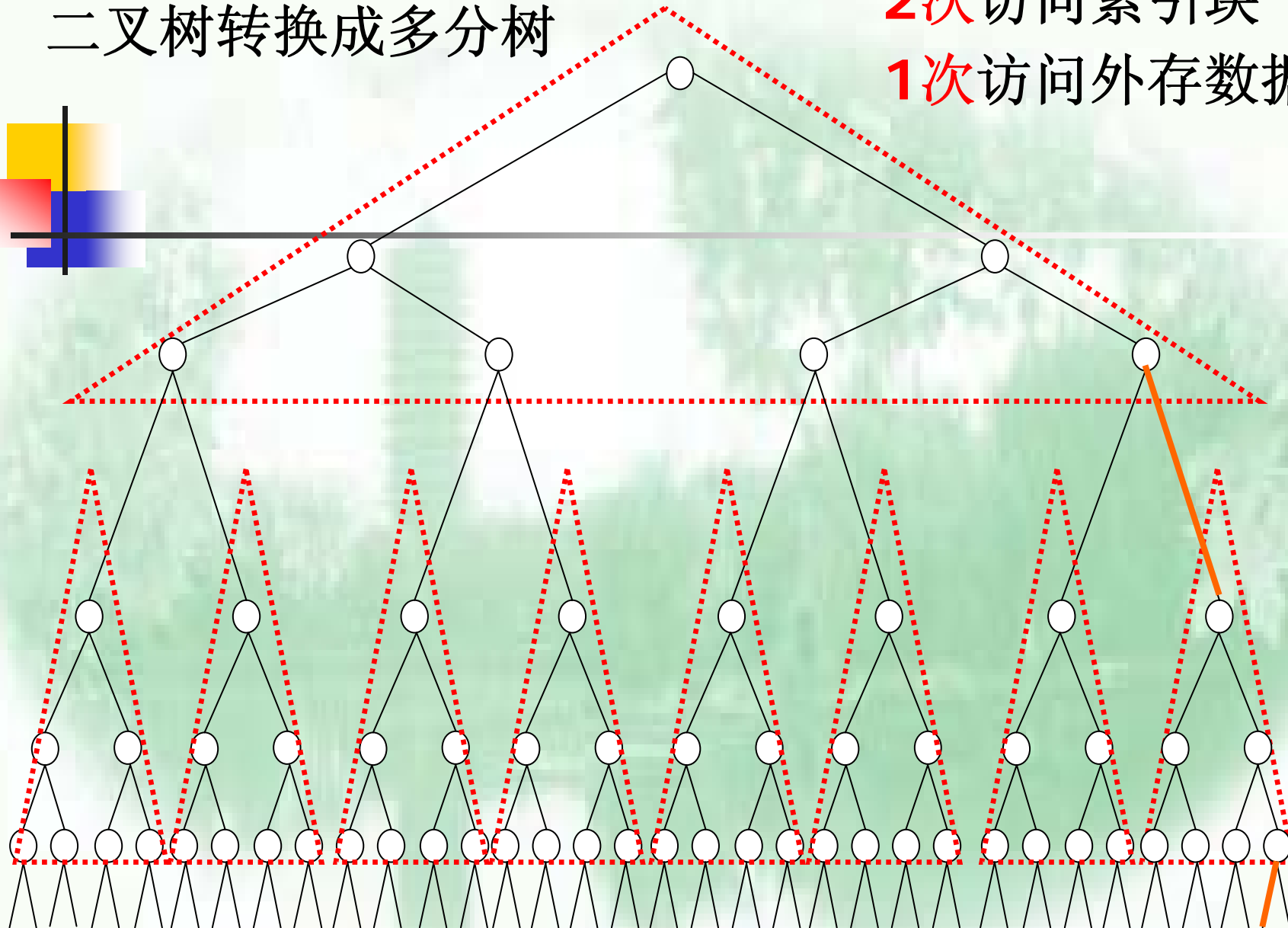
6次访问索引块
1次访问外存数据块

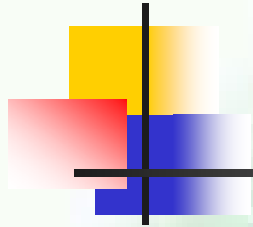


二叉树转换成多分树

2次访问索引块

1次访问外存数据块



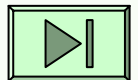
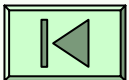


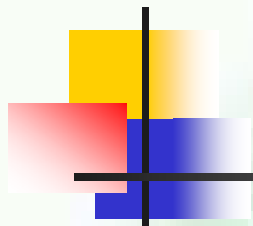
- “数据基本区”

- 多分树的叶结点区域
- 存放数据记录

- “索引区”

- 多分树的非叶结点区域
- 存放各子树结点中的最大(或最小)的关键码



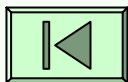


■ 溢出、溢出区

- 新记录要插入的结点已满
- 把溢出的记录存放到另开辟的溢出区
- 不改变索引的结构

■ 记录送入溢出区的两种方式

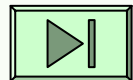
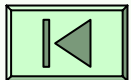
- 保持顺序，把最后一个记录送往溢出区
- 不保持顺序，把新插入的记录送入溢出区

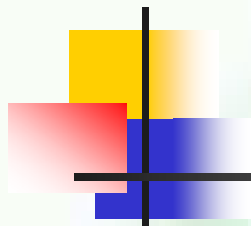




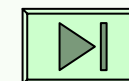
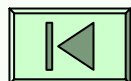
10.2.2 ISAM

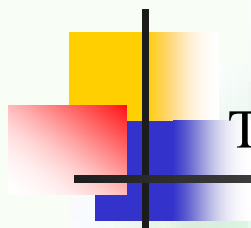
- ISAM是解决需要频繁更新的大型数据库的一个早期尝试
- 在采用基于B⁺树的VSAM技术之前，IBM公司曾经广泛地采用ISAM技术





- 多分树的应用
- 为磁盘存取而设计
- 结构采用多级索引
 - 主索引
 - 柱面索引
 - 磁道索引





C_0

T_0	400	T_1	625	T_2	1000	T_3
-------	-----	-------	-----	-------	------	-------

主索引

T_1	80	C_1T_0	200	C_2T_0	400	C_3T_0
T_2					625	C_6T_0
T_3					1000	C_9T_0
	\vdots					

柱面索引

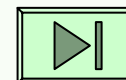
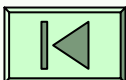
C_1

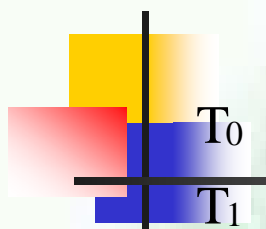
T_0	40 T_1	40 T_1	80 T_2	80 T_2	...
T_1	R_{10}	R_{20}	R_{30}	R_{40}	
T_2	R_{50}	R_{60}	R_{70}	R_{80}	
\vdots			\vdots		
T_7					

磁道索引

基本区

溢出区



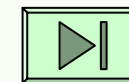
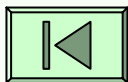


C ₂						
T ₀	150 T ₁	150 T ₁	200 T ₂	200 T ₂	...	磁道索引
T ₁	R ₉₀ R ₁₁₀ R ₁₂₀ R ₁₅₀					基本区
T ₂	R ₁₆₀ R ₁₇₅ R ₁₉₀ R ₂₀₀					
⋮	⋮					溢出区
T ₇						

\vdots

C_9

T_0	890 T_1	890 T_1	1000 T_2	1000 T_2	...	磁道索引
T_1	R_{830} R_{840} R_{880} R_{890}					基本区
T_2	R_{920} R_{930} R_{980} R_{1000}					
\vdots	\vdots					溢出区
T_7						



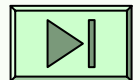
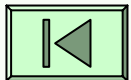


10.3 倒排索引

◆ 基本概念

◆ 10.3.1 基于属性的倒排

◆ 10.3.2 对正文文件的倒排

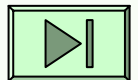
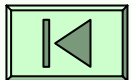




基本概念

基于属性的检索

- 要求检索结构中某个或若干个属性满足一定条件的结点
- 不是按关键码的值检索

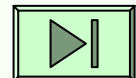
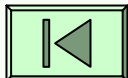


例如，在某百货公司的职工文件中，有如下的记录格式：**(EMP#, NAME, DEPT, AGE, SAL)**

该记录格式中的数据项其含义分别为职工号，姓名，所在部门，年龄，工资。



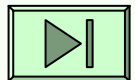
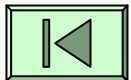
EMP#	NAME	DEPT	AGE	SAL
0100	李宇	玩具部	32	4000
0172	刘阳	食品部	43	5000
0193	赵亮	服装部	39	4000
0197	张伟	服装部	26	3000
0201	王亮	食品部	55	5000
0204	王卓	服装部	39	3500
0221	孙丽	玩具部	47	5000
0375	刘珍	电器部	26	2500
0552	周兵	玩具部	26	2500
0673	何江	电器部	40	3500

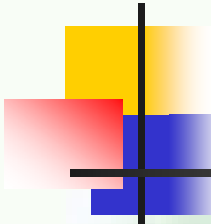




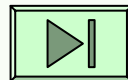
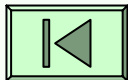
10.3.1 基于属性的倒排

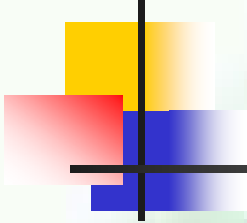
- 对某属性按属性值建立索引表，称倒排表
 - (attr, ptrList)
 - （属性值，具有该属性值的各记录指针）
 - 记录指针可以是关键码，或该记录的主文件地址
- 颠覆主文件的顺序，因而称为倒排索引
- 属性往往是离散型的
 - 对于连续型的索引，往往用**B**树
- 倒排文件：带有倒排索引的文件





DEPT list	EMP#
玩具部	0100, 0221, 0552
食品部	0172, 0201
服装部	0193, 0197, 0204
电器部	0375, 0673
AGE list	EMP#
26	0197, 0375, 0552
32	0100
39	0193, 0204
40	0673
43	0172
47	0221
55	0201
SAL list	EMP#
2500	0375, 0552
3000	0197
3500	0204, 0673
4000	0100, 0193
5000	0172, 0201, 0221

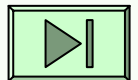
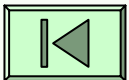


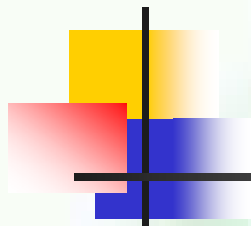


检索实例

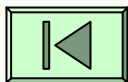
列出玩具部中年龄在**50**岁以上或者工资在**5000**元以上的职工记录
(**DEPT="Toy" AND (AGE \geq 50 OR SAL \geq 5000)**)。

分别找出满足条件**DEPT="Toy"**，**AGE \geq 50**，和**SAL \geq 5000**的指针集合，然后对后两个指针集合求并，并且将结果集合与第一个指针集合求交，最后的结果集合中包含的指针所指的各记录即为所求。





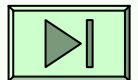
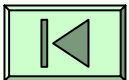
- 优点：能够对于基于属性的检索进行较高效率的处理
- 缺点：
 - 花费了保存倒排表的存储代价
 - 降低了更新运算的效率





10.3.2 对正文文件的倒排

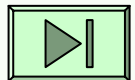
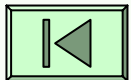
- 正文索引(**Text Indexing**)处理的就是“建立一个数据结构以提供对文本内容的快速检索”。
- 方法
 - 词索引(**word index**)
 - 全文索引(**full-text index**)





词索引

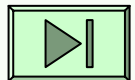
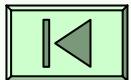
- 基本思想：
 - 把正文看作由符号和词所组成的集合，从正文中抽取出关键词，然后用这些关键词组成一些适合快速检索的数据结构。
- 适用于多种文本类型，特别是那些可以很容易就解析成一组词的集合的文本
 - 适用于英文
 - 中文等东方文字要经过“切词”处理

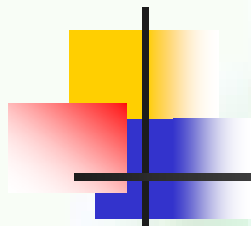




全文索引

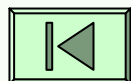
- 基本思想：
 - 把正文看作一个长的字符串
 - 在数据结构中记录的是子字符串的开始位置
 - 查询就可以针对正文中的任何子字符串
- 可以对每一个字符建立索引，从而使查询词不再限于关键词
- 需要更大的空间





词索引使用最广泛

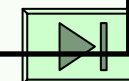
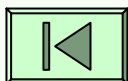
- 一个已经排过序的关键词的列表
 - 其中每个关键词指向一个倒排表
(**posting list**)
 - 指向该关键词出现文档集合
 - 在文档中的位置



文档编号	文本内容
1	Pease porridge hot, please porridge cold,
2	Pease porridge in the pot,
3	Nine days old.
4	Some like it hot, some like it cold,
5	Some like it in the pot,
6	Nine days old.

倒排索引

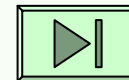
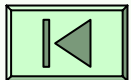
编号	词语	(文档编号, 位置)
1	cold	(1,6) (4,8)
2	days	(3,2) (6,2)
3	hot	(1,3) (4,4)
4	in	(2,3) (5,4)
5	it	(4,3) (4,7) (5,3)
6	like	(4,2) (4,6) (5,2)
7	nine	(3,1) (6,1)
8	old	(3,3) (6,3)
9	pease	(1,1) (1,4)
10	porridge	(1,2) (1,5) (2,2)
11	pot	(2,5) (5,6)
12	some	(4,1) (4,5) (5,1)
13	the	(2,4) (5,5)



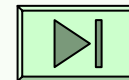
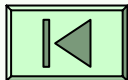


建立正文倒排文件

- 1. 对文档集中的所有文件都进行分割处理，把正文分成多条记录文档
 - 切分正文记录取决于程序的需要
 - 定长的块、段落、章节，甚至一组文档



- 
- 2. 给每条记录赋一组关键词
 - 以人工或者自动的方式从记录中抽取关键词
 - 停用词(**Stopword**)
 - 抽词干(**Stemming**)
 - 切词 (**segmentation**)





教育部部长周济说，今后若干年内，毕业生总量将会持续增加，每年都有数十万的增量，对毕业生就业工作无疑是巨大的挑战。2005年高校毕业生总量大、增幅高，各地和高校工作进展情况差异较大，并且还存在许多深层次的矛盾和问题。在全社会就业形势十分严峻的形势下，2005年高校毕业生就业工作压力十分突出。

操作选项

☒ 词语切分 ☐ 一级标注 ☐ 二级标注

输出格式

☒ 北大标准 ☐ 973标准 ☐ XML

运行

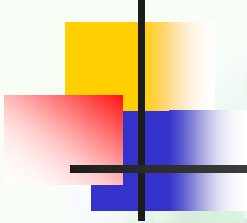
处理文件...

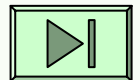
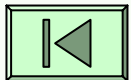
退出

关于...

1 结果 平滑参数: 0 当前结果评分: -101.911148 处理用时: 10 ms

教育部 部长 周济 说 ， 今后 若干 年 内 ， 毕业生 总量 将 会 的 持续 增加 ， 每年
都 有 数十 万 的 增量 ， 对 毕业生 就业 工作 无疑 是 巨大 的 挑战 。 2005年 高
校 毕业生 总量 大 、 增幅 高 ， 各地 和 高校 工作 进展 情况 差异 较 大 ， 并
且 还 存在 许多 深 层次 的 矛盾 和 问题 。 在 全 社会 就业 形势 十分 严峻 的
形势 下 ， 2005年 高校 毕业生 就业 工作 压力 十分 突出 。

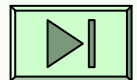
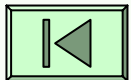
- 
- 3. 建立正文倒排表、倒排文件
 - 得到各个关键词的集合
 - 对于每一个关键词得到其倒排表
 - 然后把所有的倒排表存入文件
 - 记录在每个倒排表在索引文件中开始的位置以及每个表的大小（也可以记录每个关键词的出现次数）





对关键词的检索

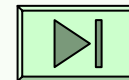
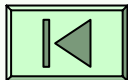
- 第一步，在倒排文件中检索关键词
- 第二步，如果找到了关键词，那么获取文件中的对应的倒排表，并获取倒排表中的记录
- 通常使用另一个索引结构（字典）进一步对关键词表进行有效索引
 - Trie
 - 散列

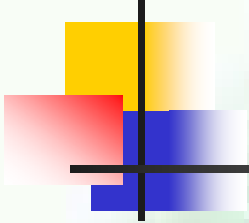




倒排文件优劣

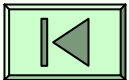
- 高效检索，用于文本数据库系统
- 支持的检索类型有限
 - 检索词有限
 - 只能用索引文件中的关键词
 - 倒排文件中的索引效率可能不高
 - 需要的空间代价往往很高





10.4 动态索引

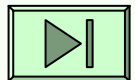
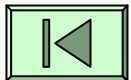
- ◆ 基本概念
- ◆ 10.4.1 B树
- ◆ 10.4.2 B⁺树
- ◆ 10.4.3 VSAM
- ◆ 10.4.4 B树的性能分析





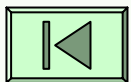
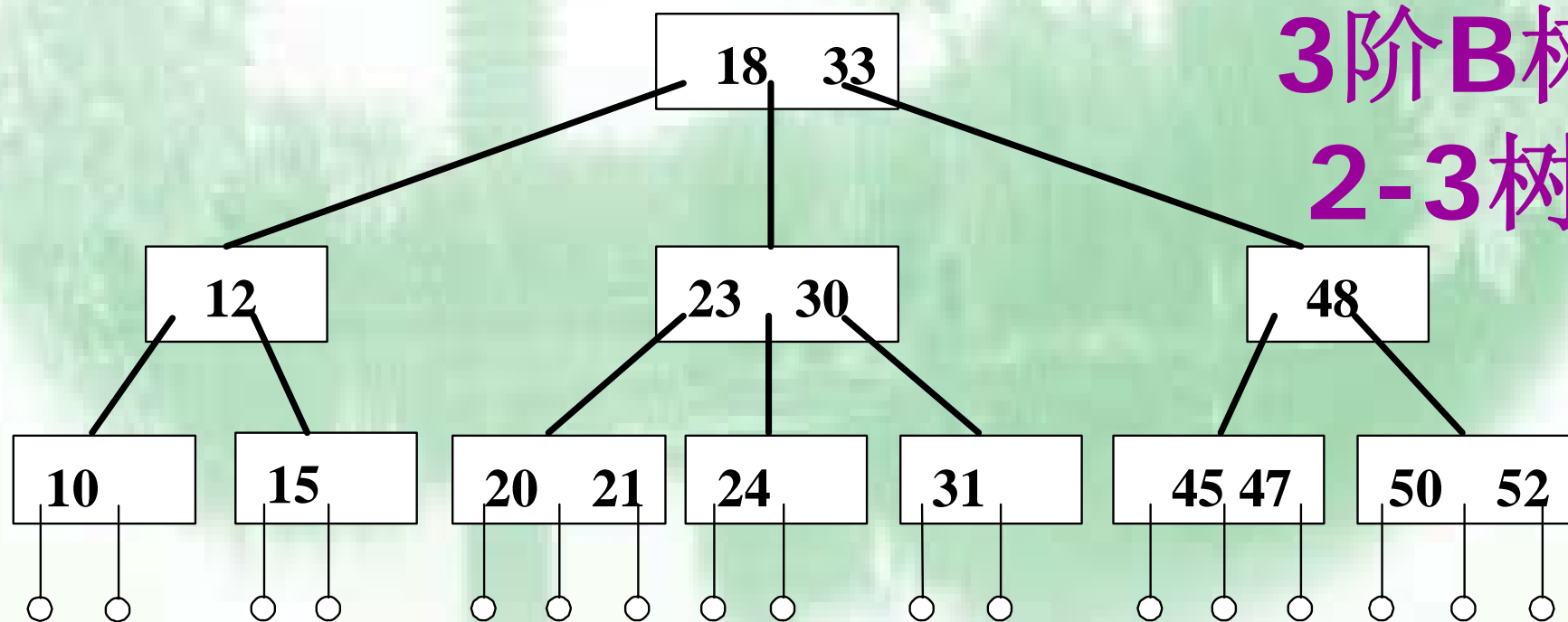
基本概念

- 动态索引结构
 - 索引结构本身也可能发生改变
 - 在系统运行过程中插入或删除记录时
- 目的
 - 保持较好的性能
 - 例如较高的检索效率



10.4.1 B树

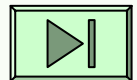
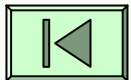
- 一种平衡的多分树 (Balanced Tree)





m阶B树的结构定义

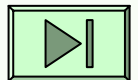
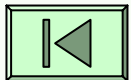
- (1) 每个结点至多有m个子结点;
- (2) 除根结点和叶结点外, 其它每个结点至少有 $\lceil m/2 \rceil$ 个子结点;
- (3) 根结点至少有两个子结点
 - 唯一例外的是根结点就是叶结点时没有子结点
 - 此时B树只包含一个结点
- (4) 所有的叶结点在同一层;
- (5) 有k个子结点的非根结点恰好包含k-1个关键码。





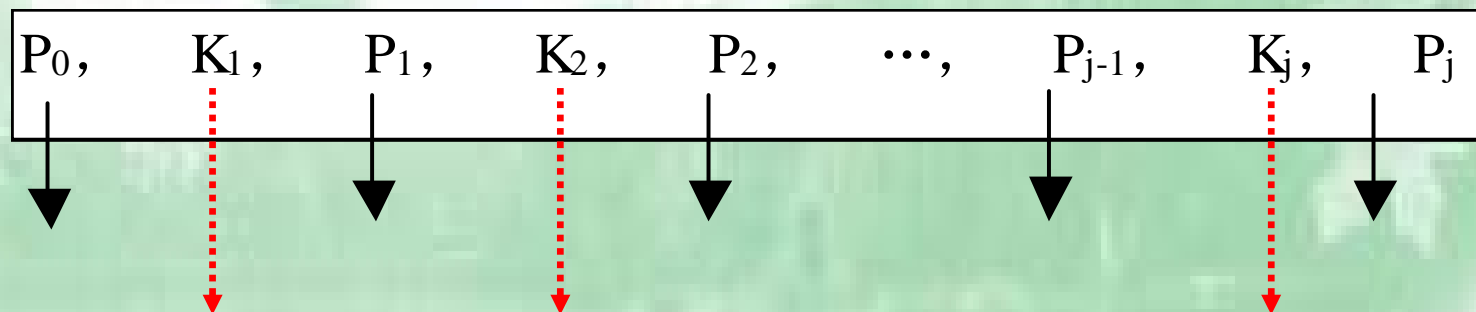
2. B树的性质

- (1) 树高平衡，所有叶结点都在同一层；
- (2) 关键码没有重复，父结点中的关键码是其子结点的分界；
- (4) B树把（值接近）相关记录放在同一个磁盘页中，从而利用了访问局部性原理；
- (5) B树保证树中至少有一定比例的结点是满的
 - 这样能够改进空间的利用率
 - 减少检索和更新操作的磁盘读取数目

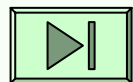
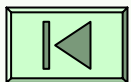


B树的结点结构

B树的一个包含 j 个关键码， $j+1$ 个指针的结点的一般形式为：



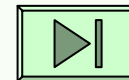
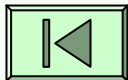
- 其中 K_i 是关键码值， $K_1 < K_2 < \dots < K_j$ ，
- P_i 是指向包括 K_i 到 K_{i+1} 之间的关键码的子树的指针。
- 还有指针吗？





B树的查找

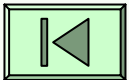
- 交替的两步过程
 - 1. 把根结点读出来，在根结点所包含的关键码 K_1, \dots, K_j 中查找给定的关键码值
 - 找到则检索成功
 - 2. 否则，确定要查的关键码值是在某个 K_i 和 K_{i+1} 之间，于是取 p_i 所指向的结点继续查找
- 如果 p_i 指向外部空结点，表示检索失败





B树检索长度

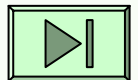
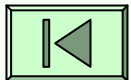
- 设B树的高度为 h
 - 独根树高为1
- 在自顶向下检索到叶结点的过程中可能需要进行 h 次读盘





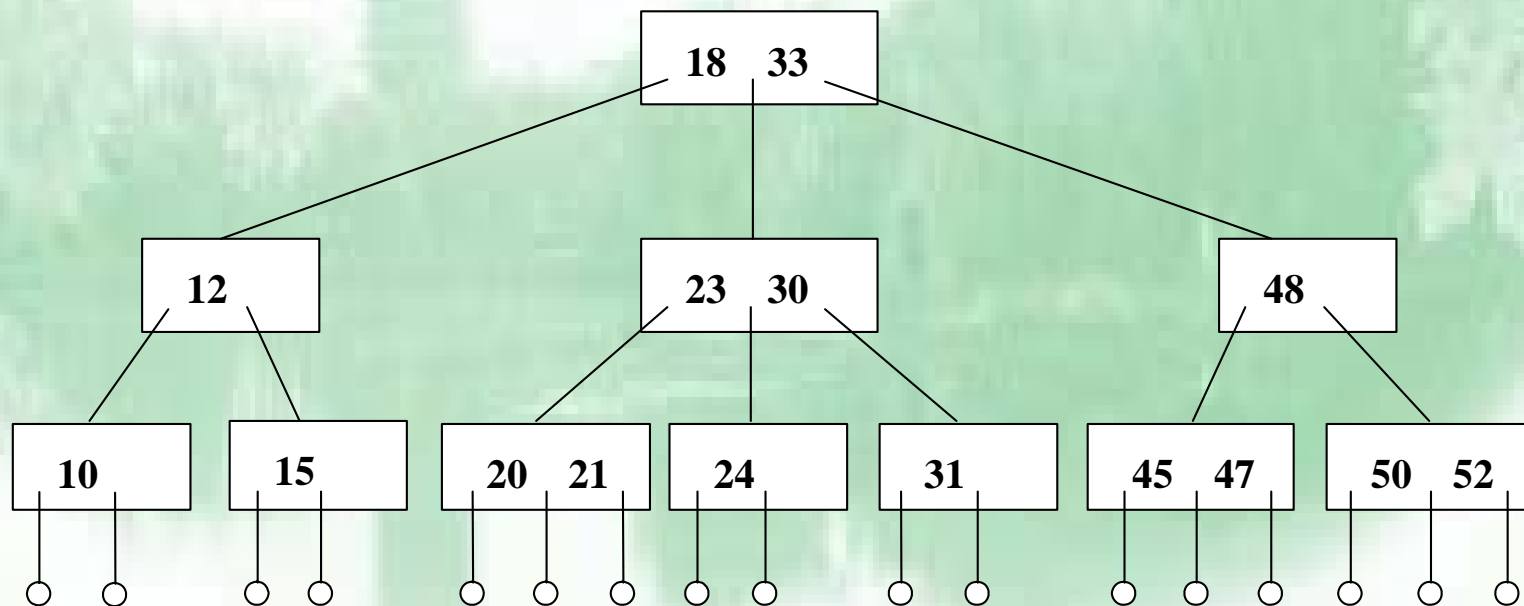
B树插入

- 注意保持性质，特别是等高和阶的限制
 - 1) 找到最底层，插入
 - 2) 若溢出，则结点分裂，中间关键码连同新指针插入父结点
 - 3) 若父结点也溢出，则继续**分裂**
 - 分裂过程可能传达到根结点(则树升高一层)

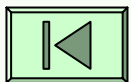


B树的插入

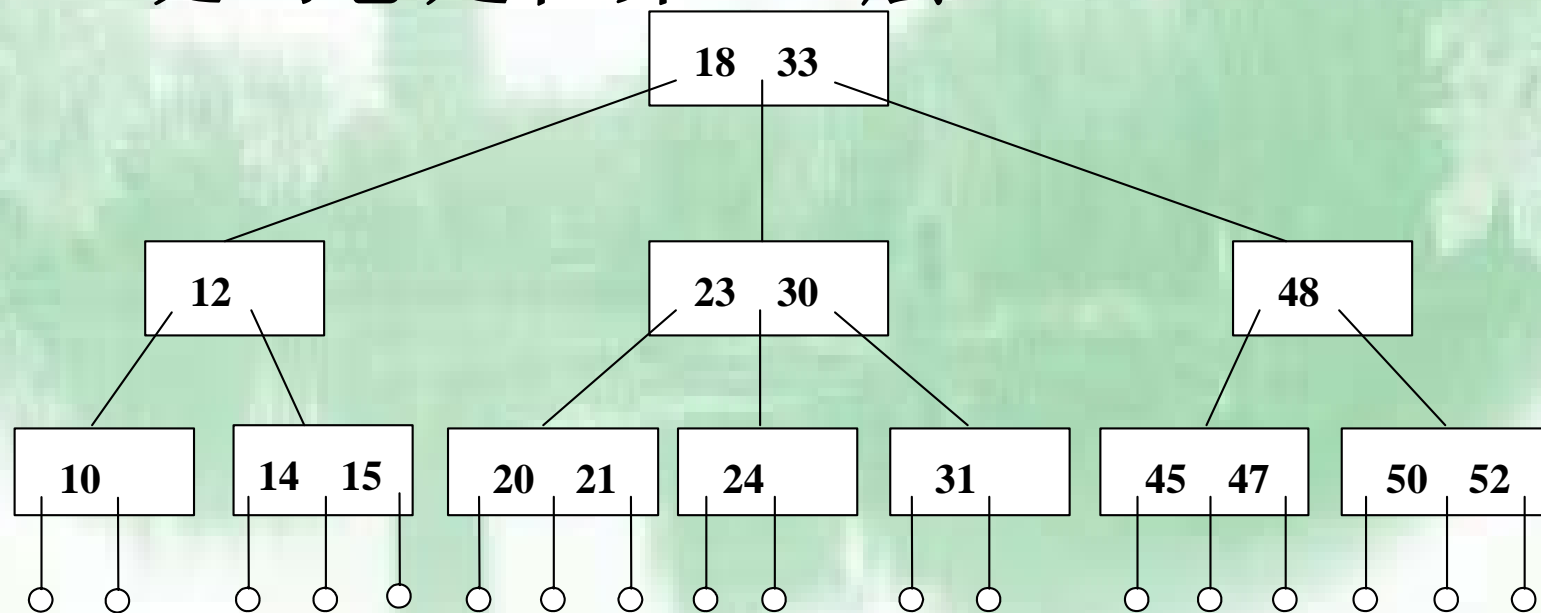
- 叶结点处在第I层的B树，插入的关键码总是在第I-1层



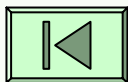
插入14



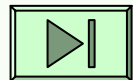
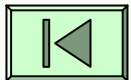
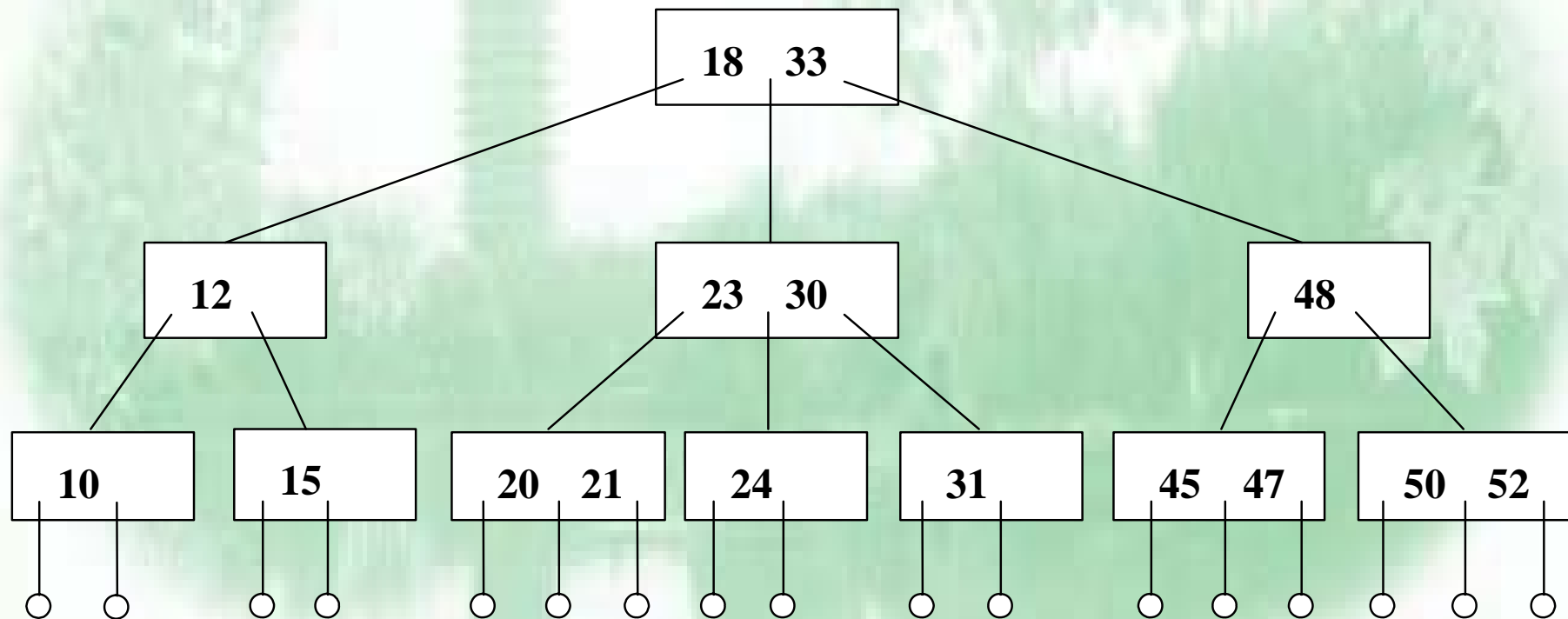
- 
- 叶结点处在第I层的B树，插入的关键码总是在第I-1层



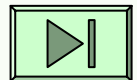
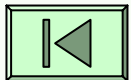
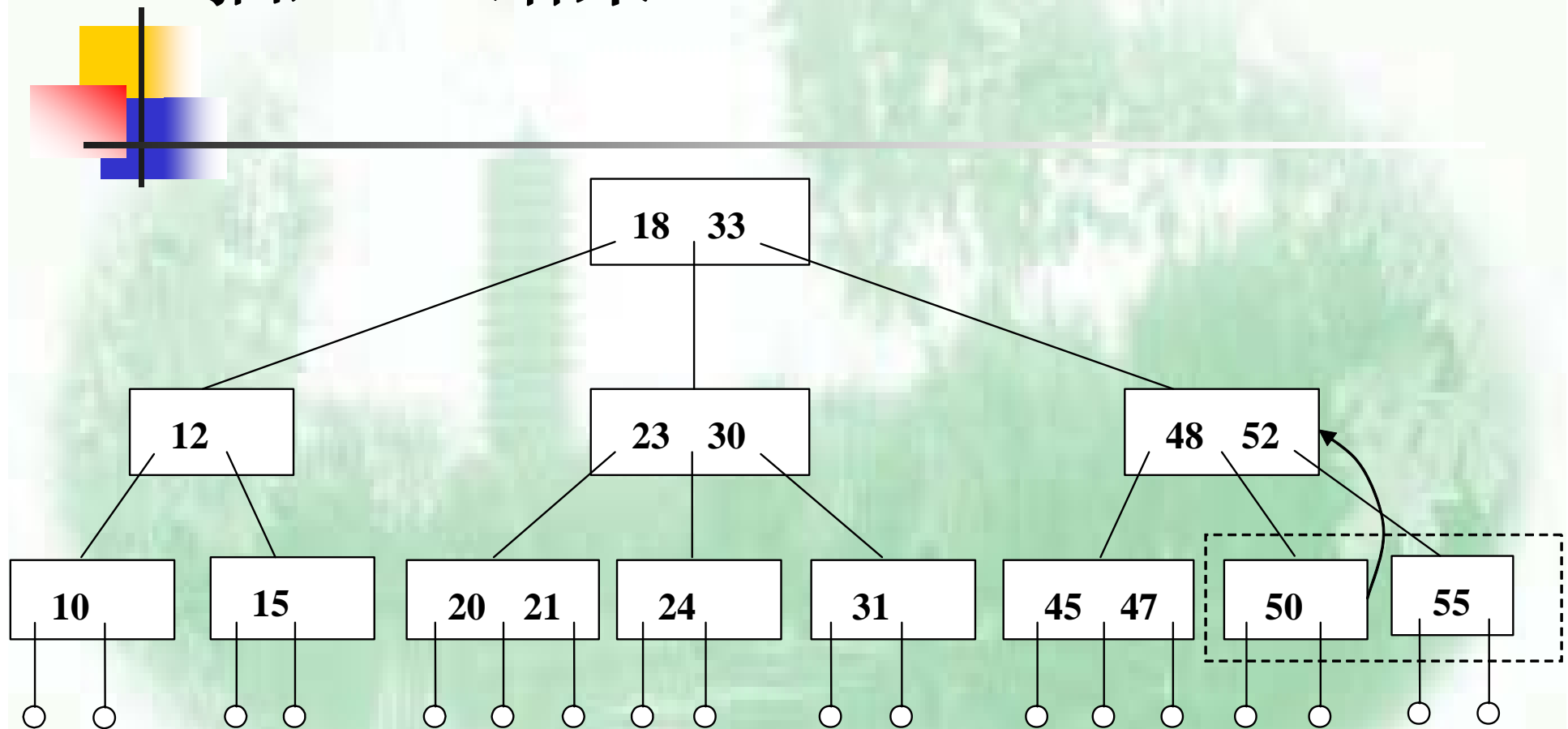
插入14



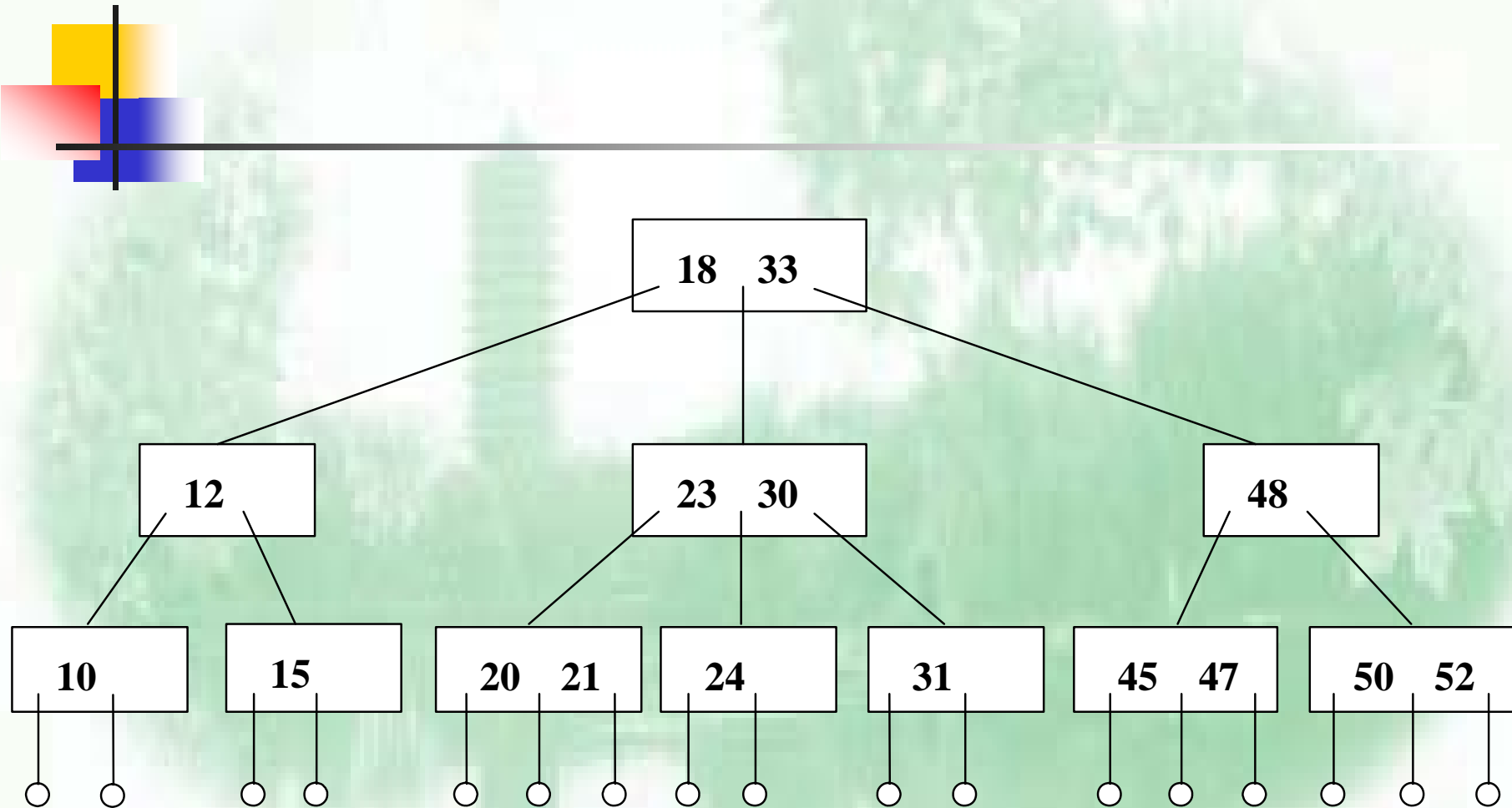
插入**55**,使得包含值**50**和**52**的页分裂, 把值**52**提升到父结点



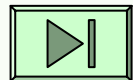
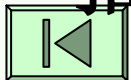
插入55结果



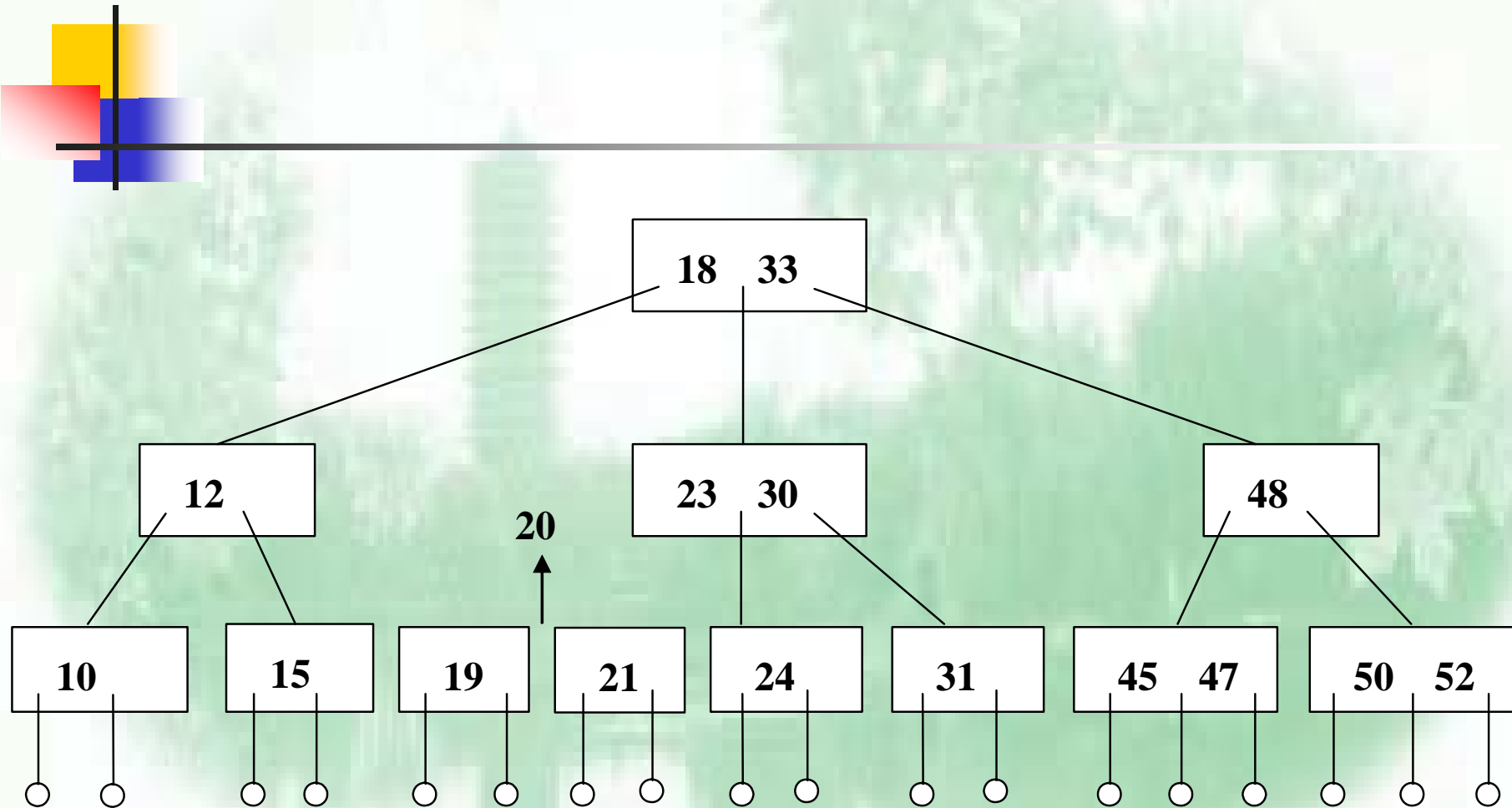
插入引起3阶B树根结点分裂的例子



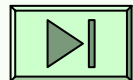
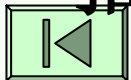
插入19

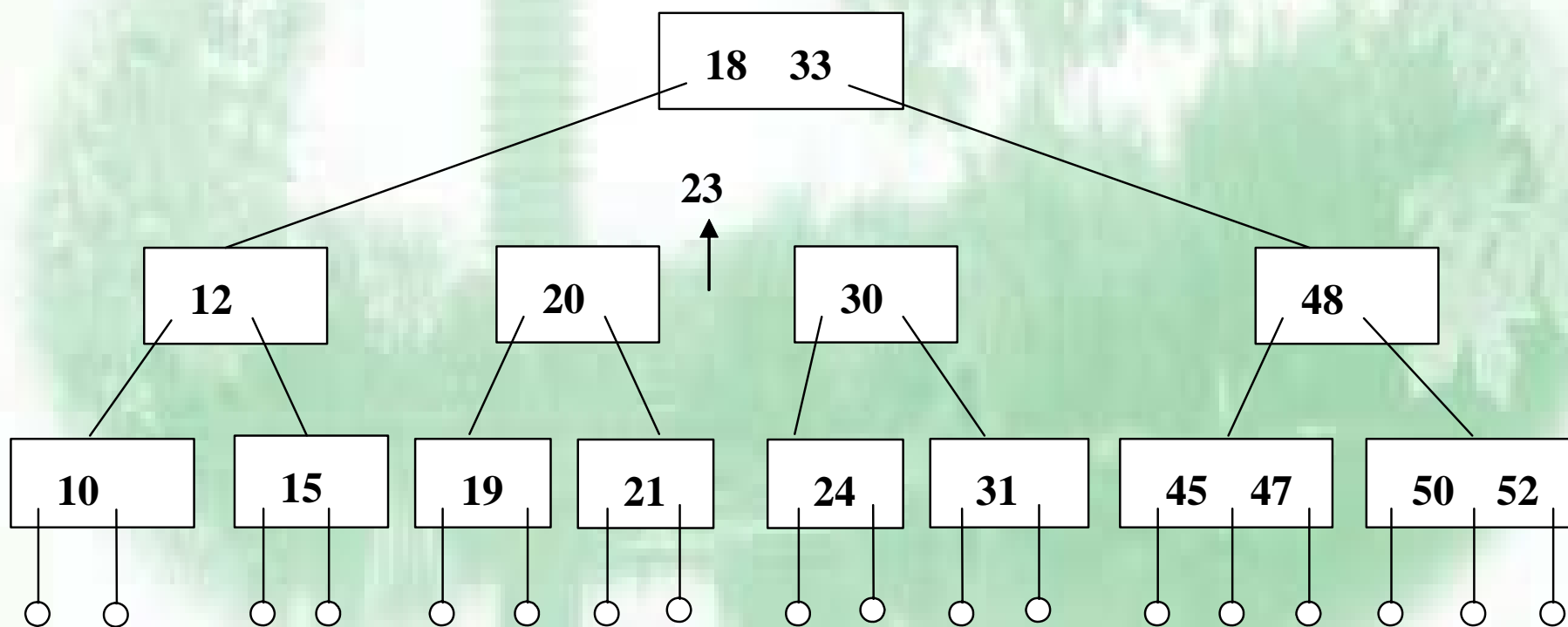
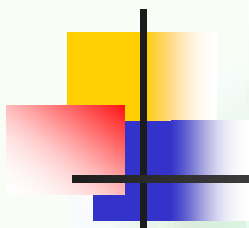


插入引起3阶B树根结点分裂的例子

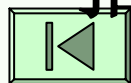


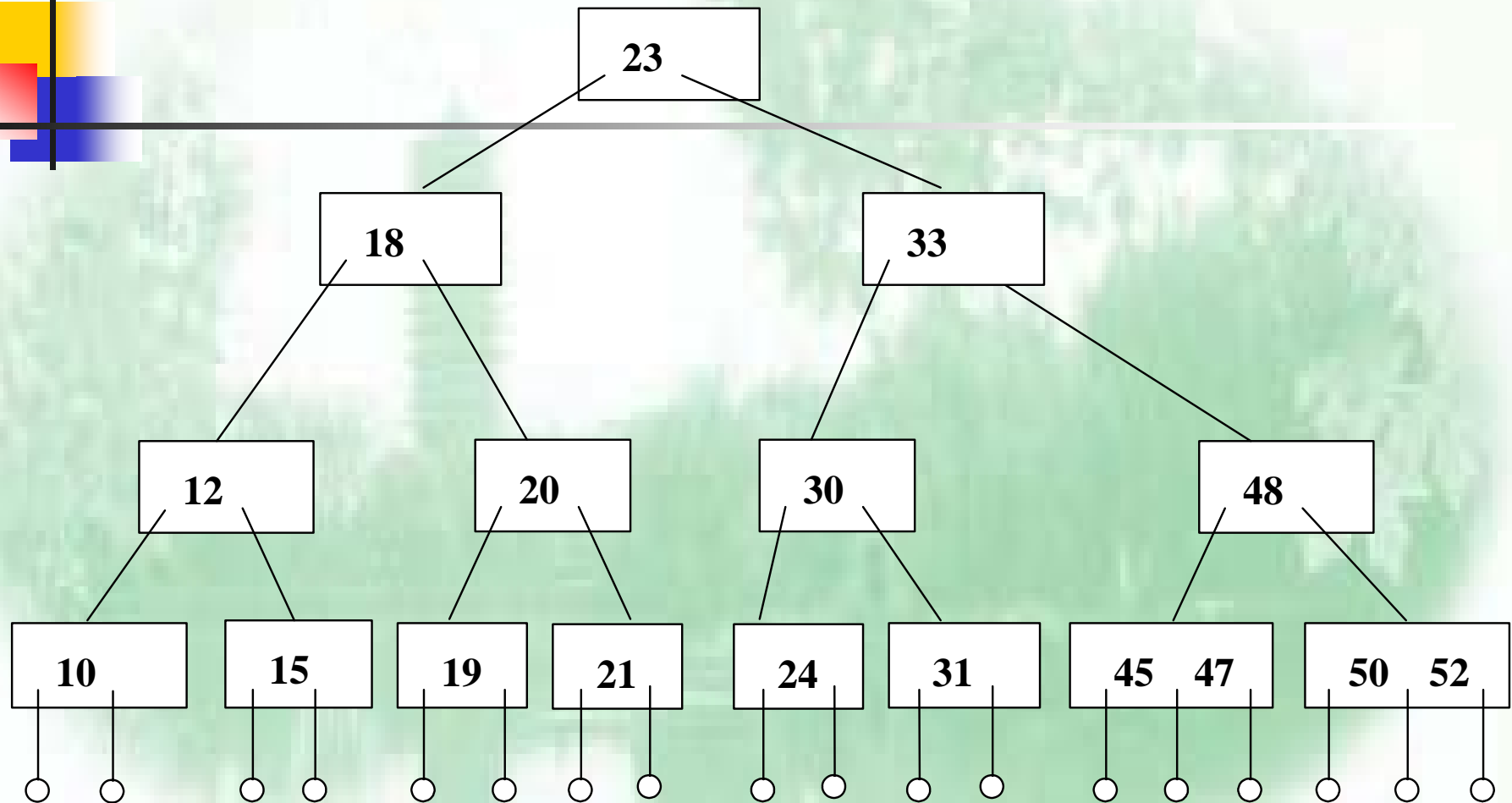
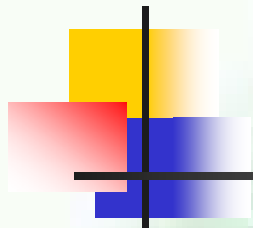
插入19



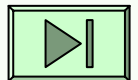
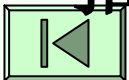


插入19





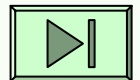
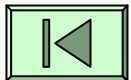
插入19





B树操作的访外次数

- 假设内存工作区足够大，使得在向下检索时读入的结点在插入后向上分裂时不必再从磁盘读入
- 读盘次数与查找相同
- 最少写盘次数：一次
 - 不分裂，写出这个关键码所插入到的结点

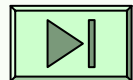
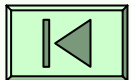


一次插入操作最多读写次数

- 总共 h 层，每层都需要分裂（包括根）
- 分裂一个非根结点要向磁盘写出 2 个结点，分裂根结点（最后一次）要写出 3 个结点

= 找插入结点向下读盘次数 +
+ 分裂非根结点时写盘次数 +
+ 分裂根结点时写盘次数

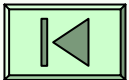
$$= h + 2(h-1) + 3 = 3h + 1$$





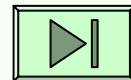
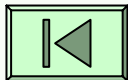
B树的删除

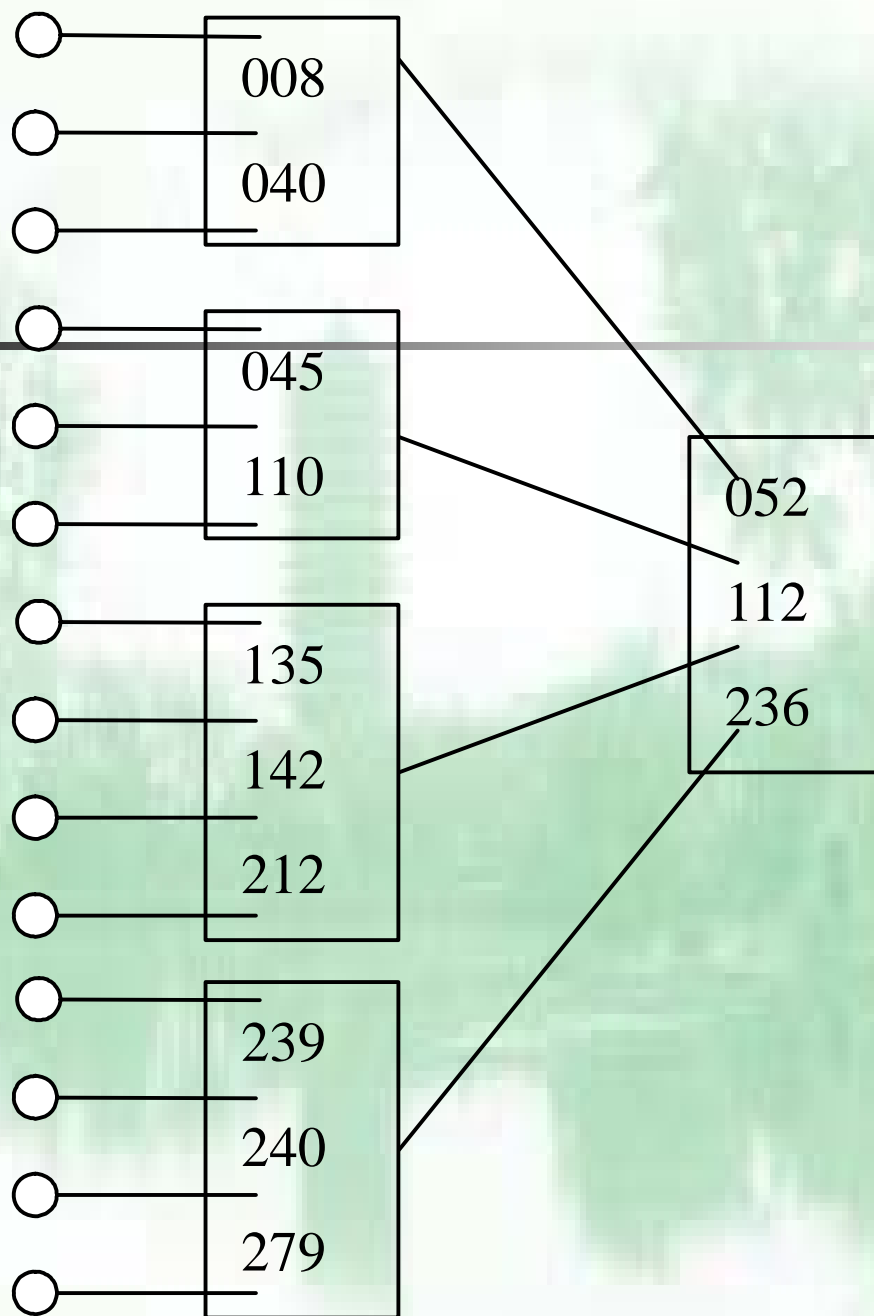
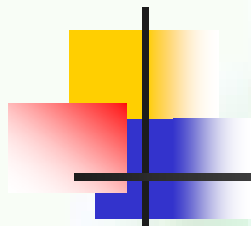
- 删除的关键码不在叶结点层
 - 先把此关键码与它在B树里的后继对换位置，然后再删除该关键码



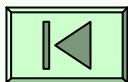
B树的删除(续)

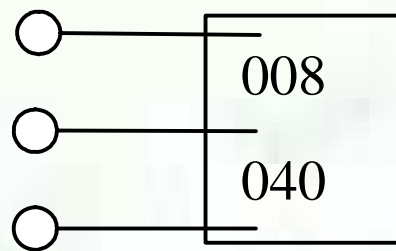
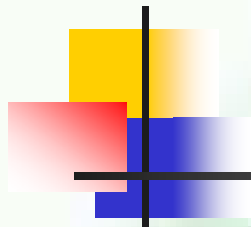
- 删除的关键码在叶结点层
 - 删除后关键码个数不小于 $\lceil m/2 \rceil - 1$
 - 直接删除
 - 关键码个数小于 $\lceil m/2 \rceil - 1$
 - 如果兄弟结点关键码个数不等于 $\lceil m/2 \rceil - 1$
 - 从兄弟结点移若干个关键码到该结点中来
(父结点中的一个关键码要做相应变化)
 - 如果兄弟结点关键码个数等于 $\lceil m/2 \rceil - 1$
 - 合并



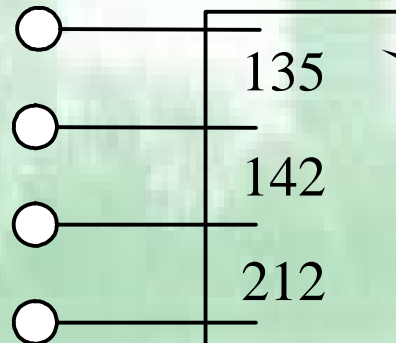
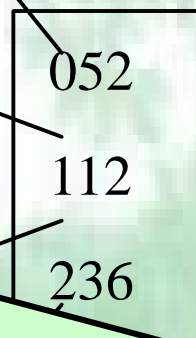
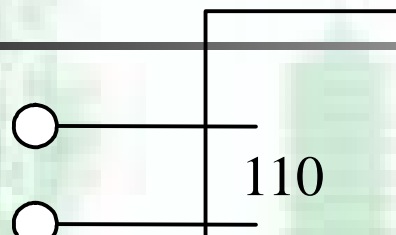


6阶B树
删除045

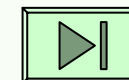
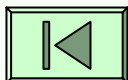
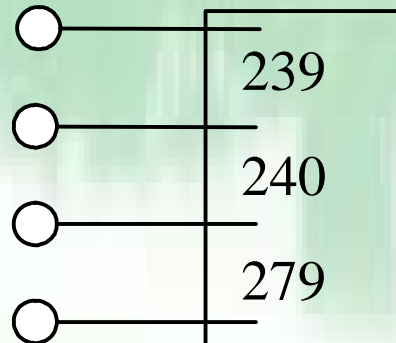


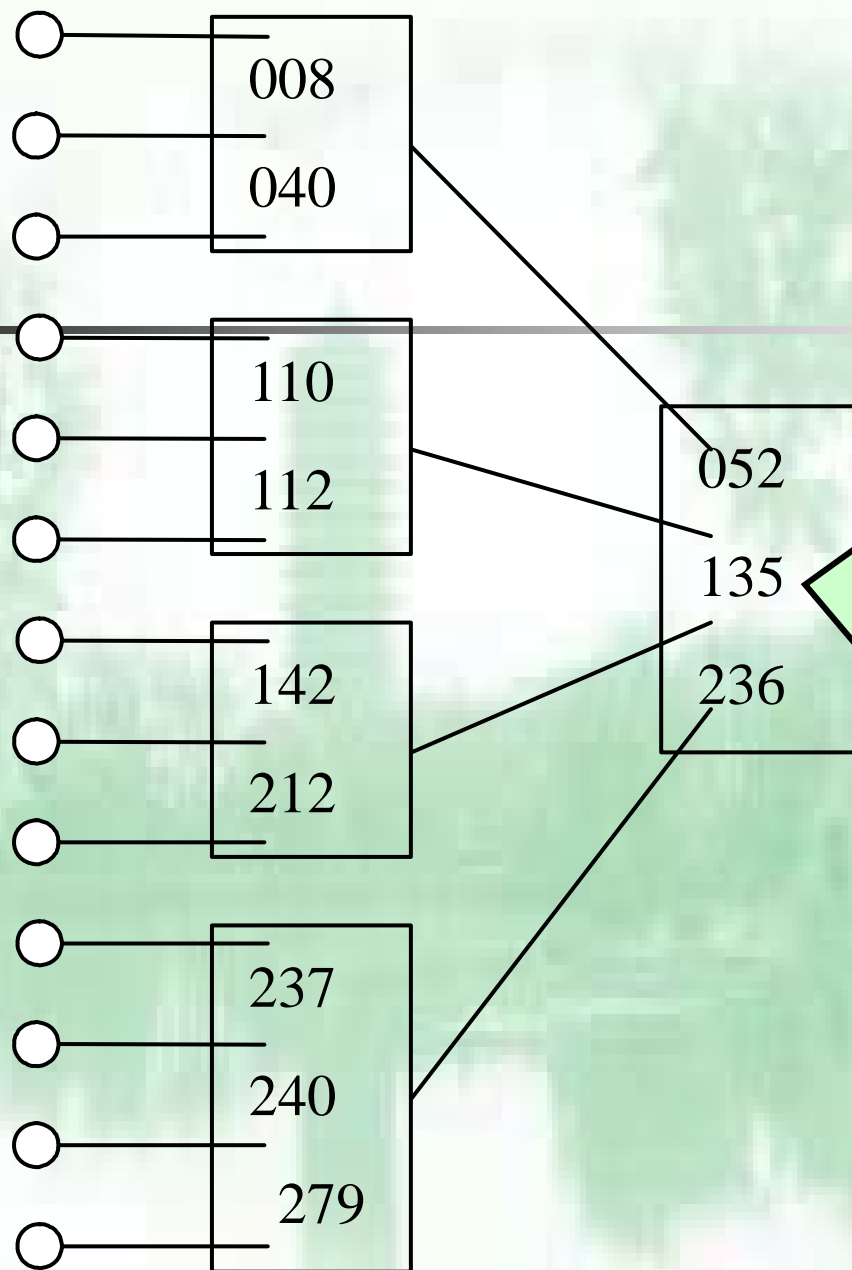
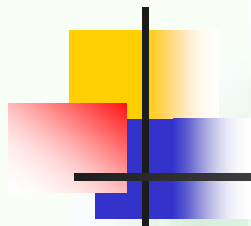


关键码个数不足

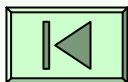


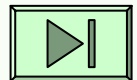
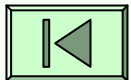
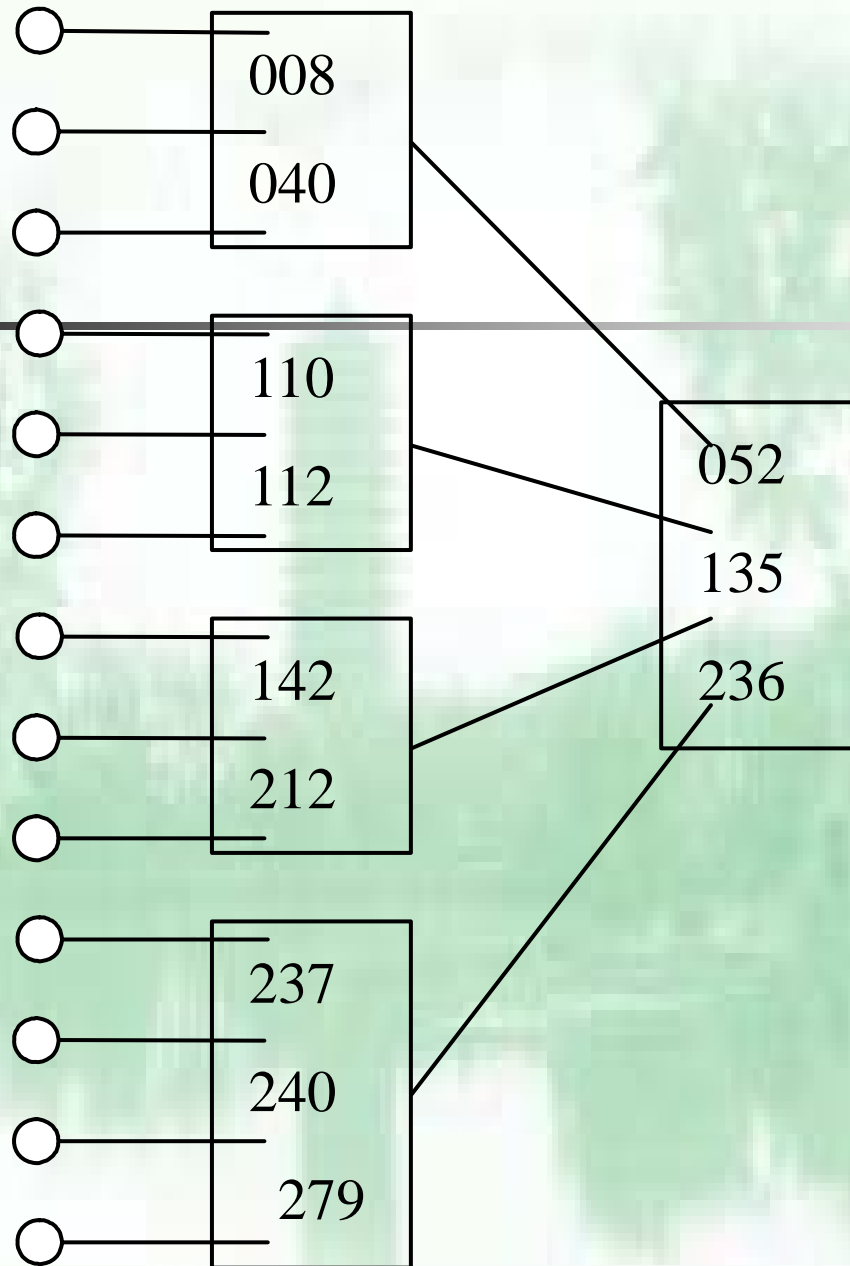
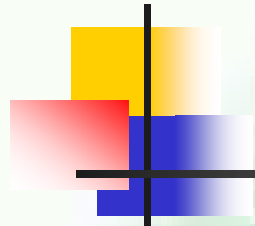
从兄弟结点移动
135

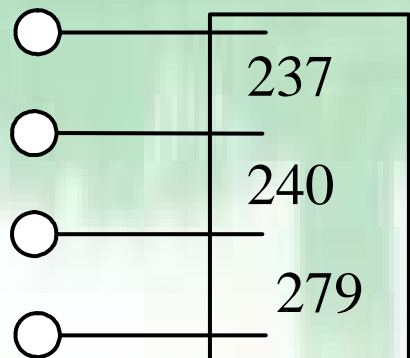
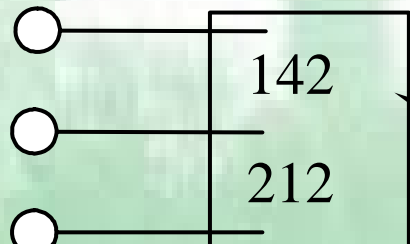
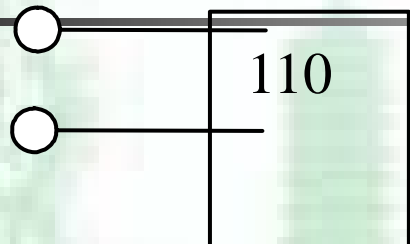
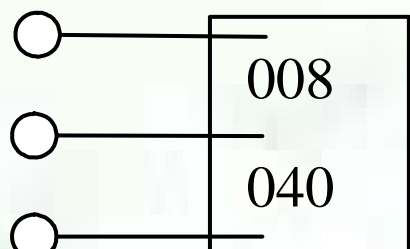
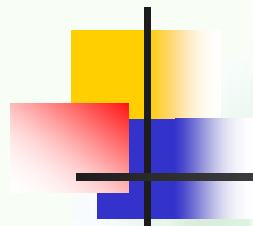




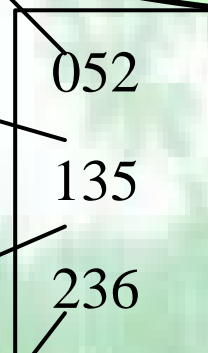
注意：兄弟节点中被父节点移动到中的点，移动到父节点中的135的结点是112



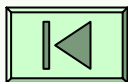


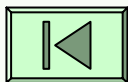
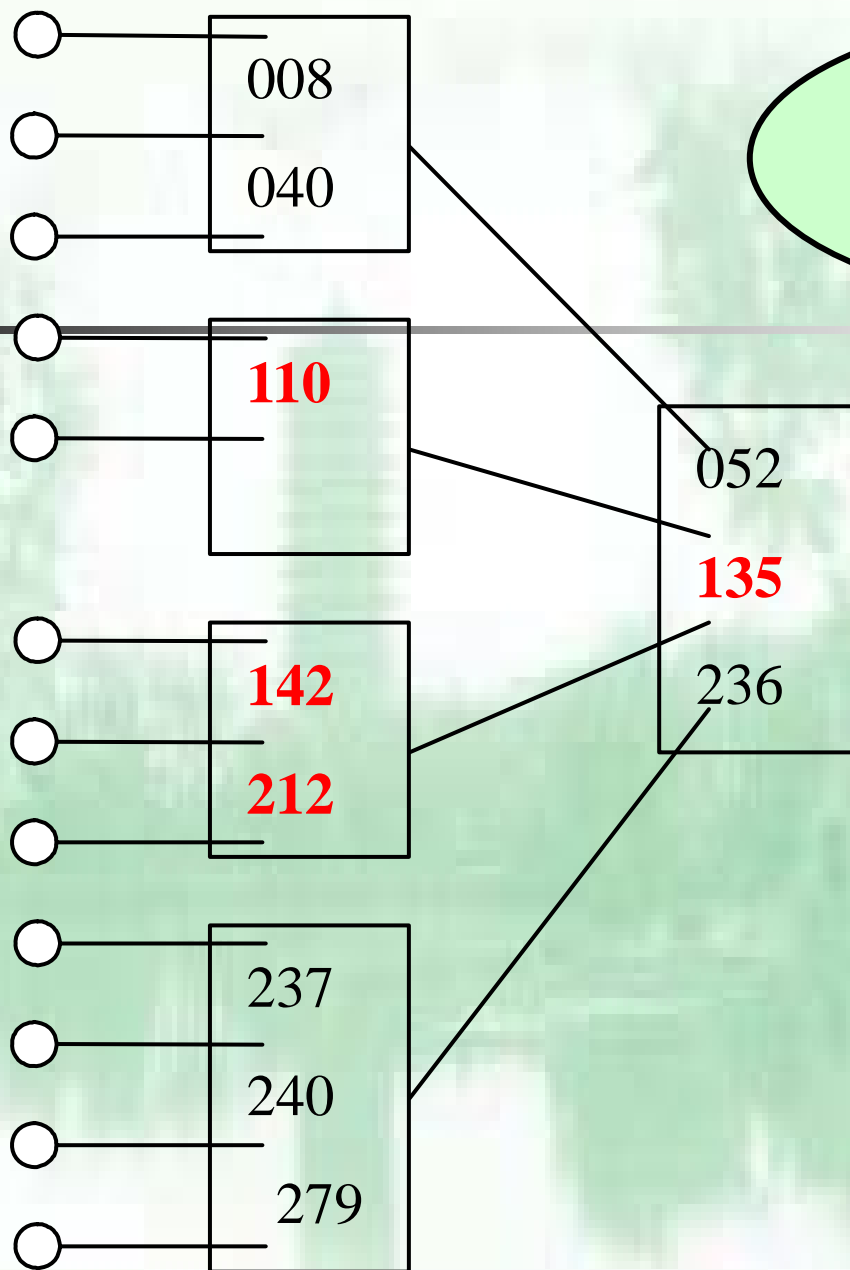
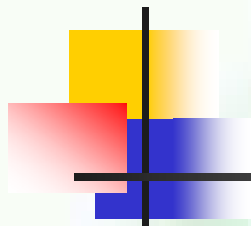


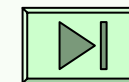
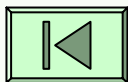
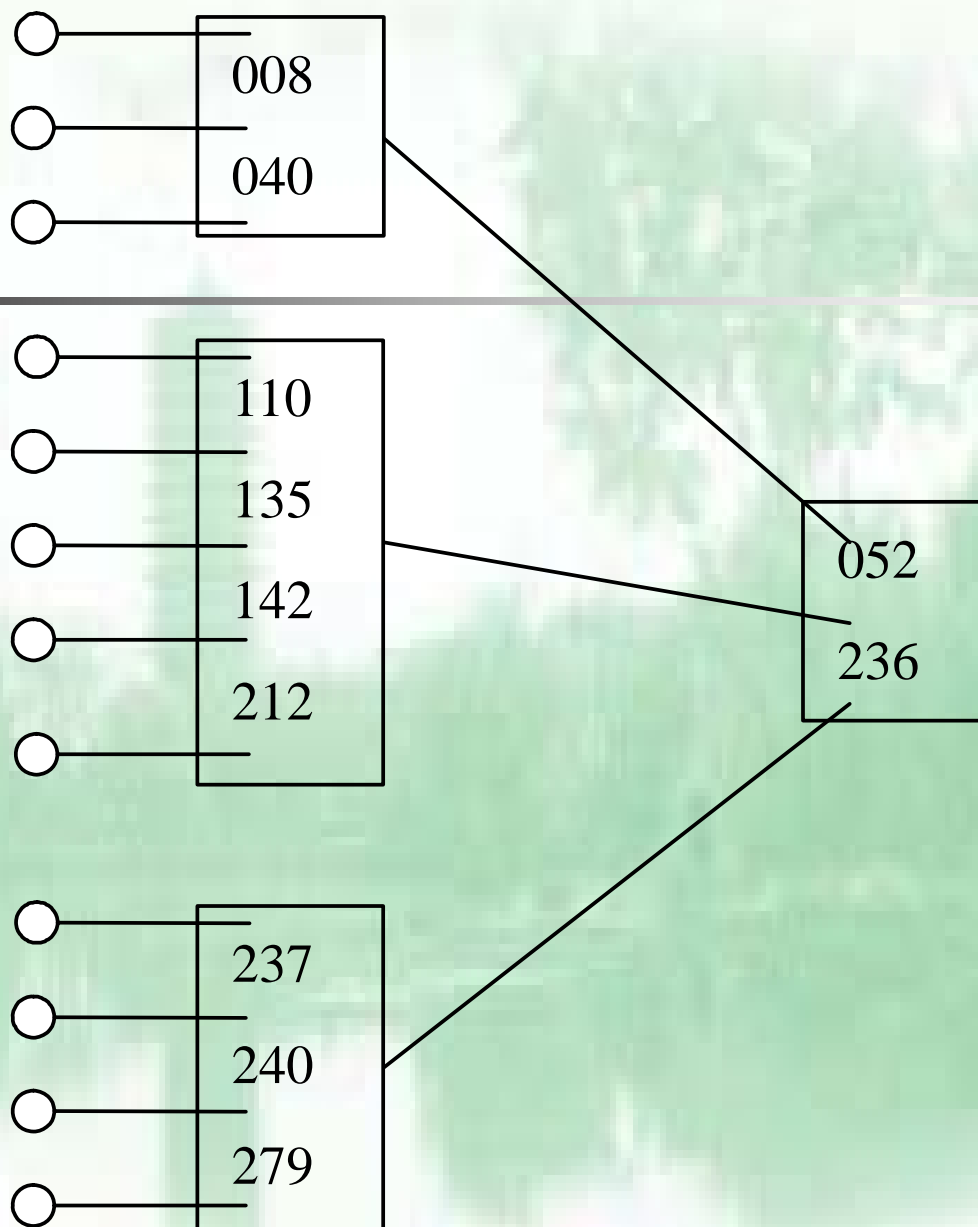
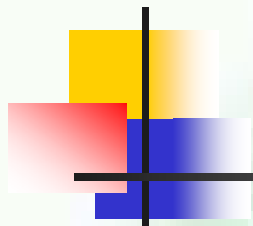
关键码个数不足



左右兄弟结点
也不富余

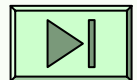
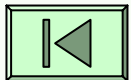






B树删除

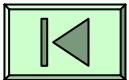
- 保持性质：等高、阶（上下界）
- 交换——删除——借关键码——合并
 - 1) 保证所删除的位置在最底层(否则，与其后继关键码交换)
 - 2) 若删除后，结点中关键码数目不够最低限(下溢出)，则先看左右兄弟有无多余的关键码可借，若有则该结点与它的一个兄弟结点的所有关键码，再加上父结点中的分界码，当成一个结点，类似于插入算法，再分为三部分
 - 3) 若其左右兄弟中关键码数目都处在最低限，则把该结点和它的一个兄弟，以及父结点中它俩的分界关键码一起合并成一个结点；(父结点中关键码数目减少一个)
 - 4) 此合并过程可能传递到根(则树降低一层)





10.4.2 B⁺树

- 是**B**树的一种变形
- 在叶结点上存储信息的树
 - 所有的关键码均出现在叶结点上
 - 各层结点中的关键码均是下一层相应结点中最大关键码（或最小关键码）的复写



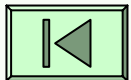


B⁺树的结构定义

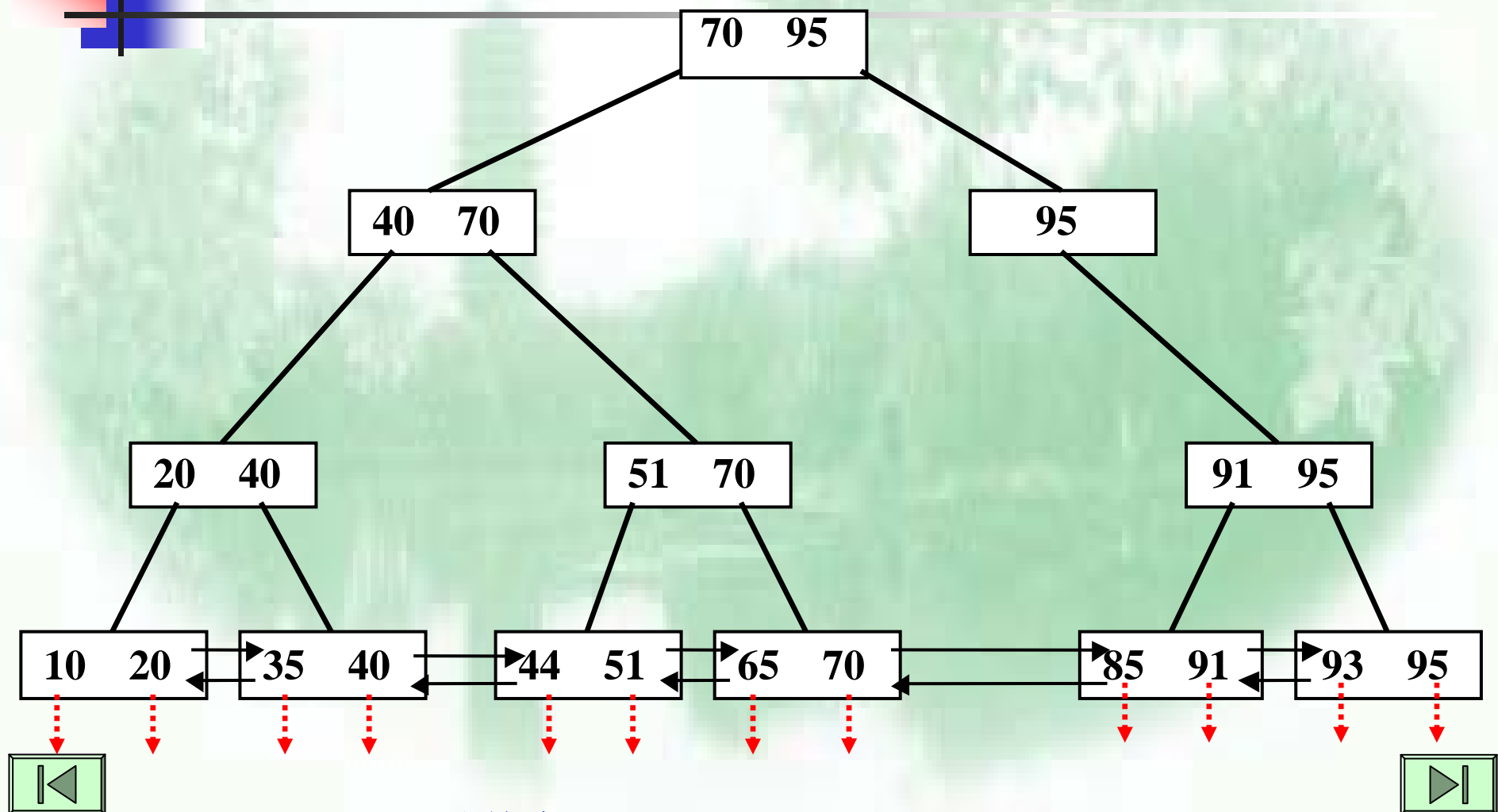
m阶B⁺树的结构定义如下：

- (1) 每个结点至多有m个子结点；
- (2) 每个结点(除根外)至少有 $\lceil m/2 \rceil$ 个子结点；
- (3) 根结点至少有两个子结点；
- (4) 有k个子结点的结点必有k个关键码。

其实，根可以为空，或者独根



2阶B+树的例子

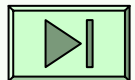
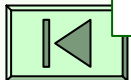


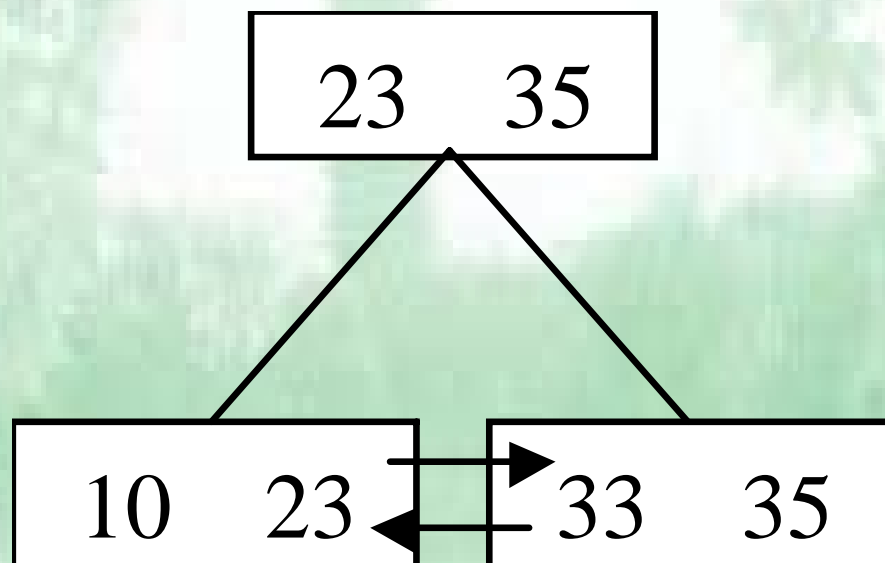
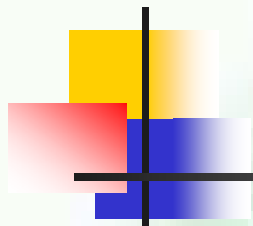


B⁺树的查找

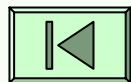
- 查找应该到叶结点层
 - 在上层已找到待查的关键码，并不停止
 - 而是继续沿指针向下一直查到叶结点层的这个关键码
- B⁺树的叶结点一般链接起来，形成一个双链表
 - 适合顺序检索（范围检索）
 - 实际应用更广

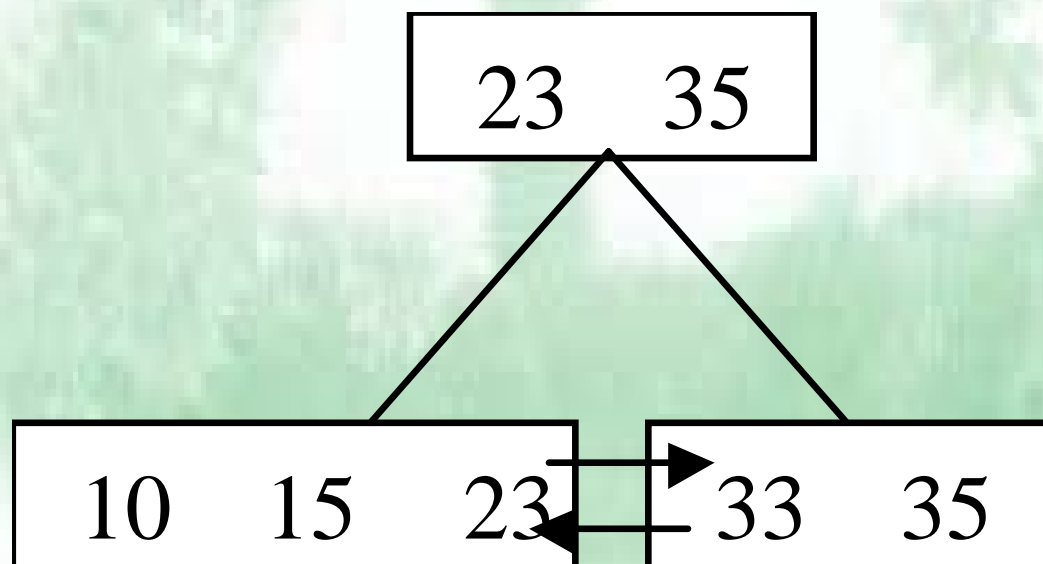
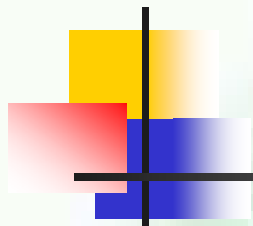
需要的话，每一层结点也可以顺序链接



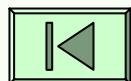


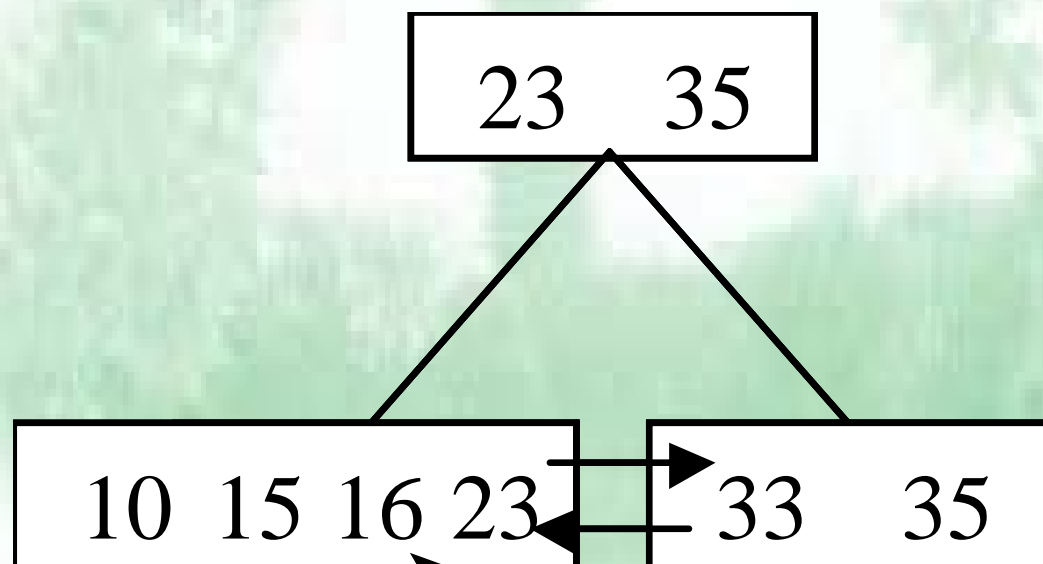
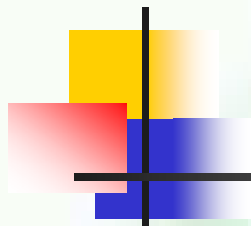
插入15





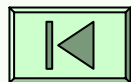
插入15后

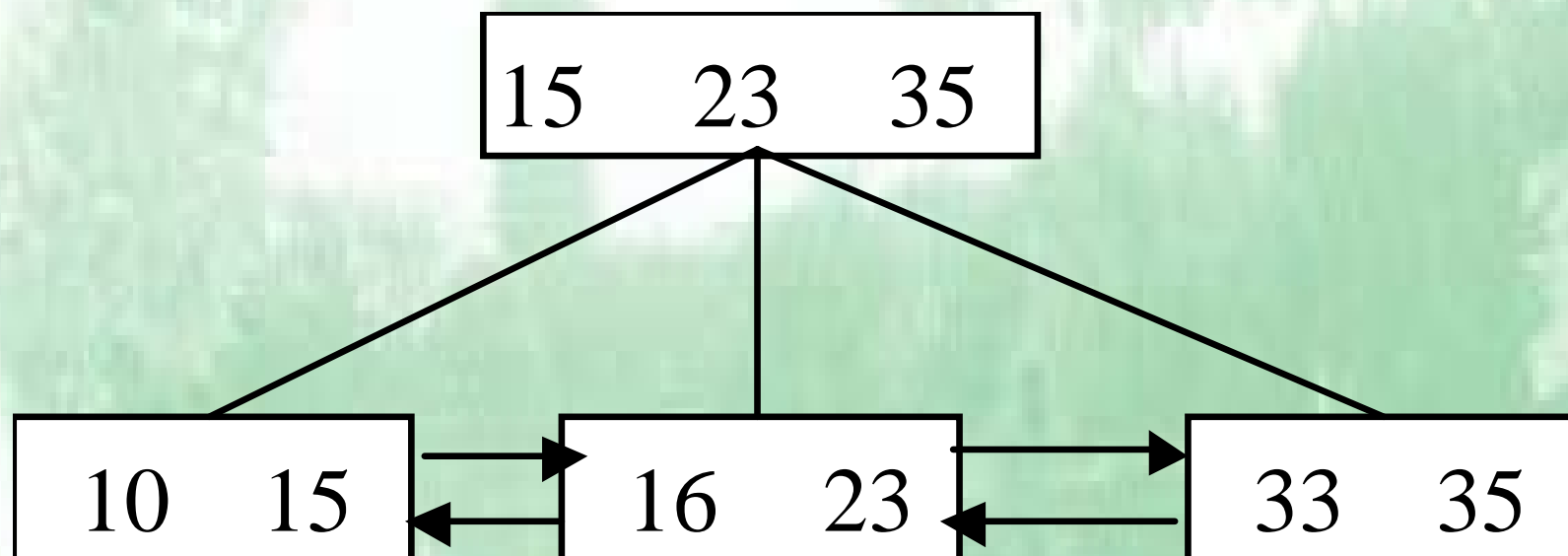
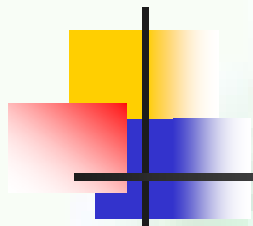




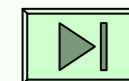
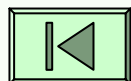
插入16

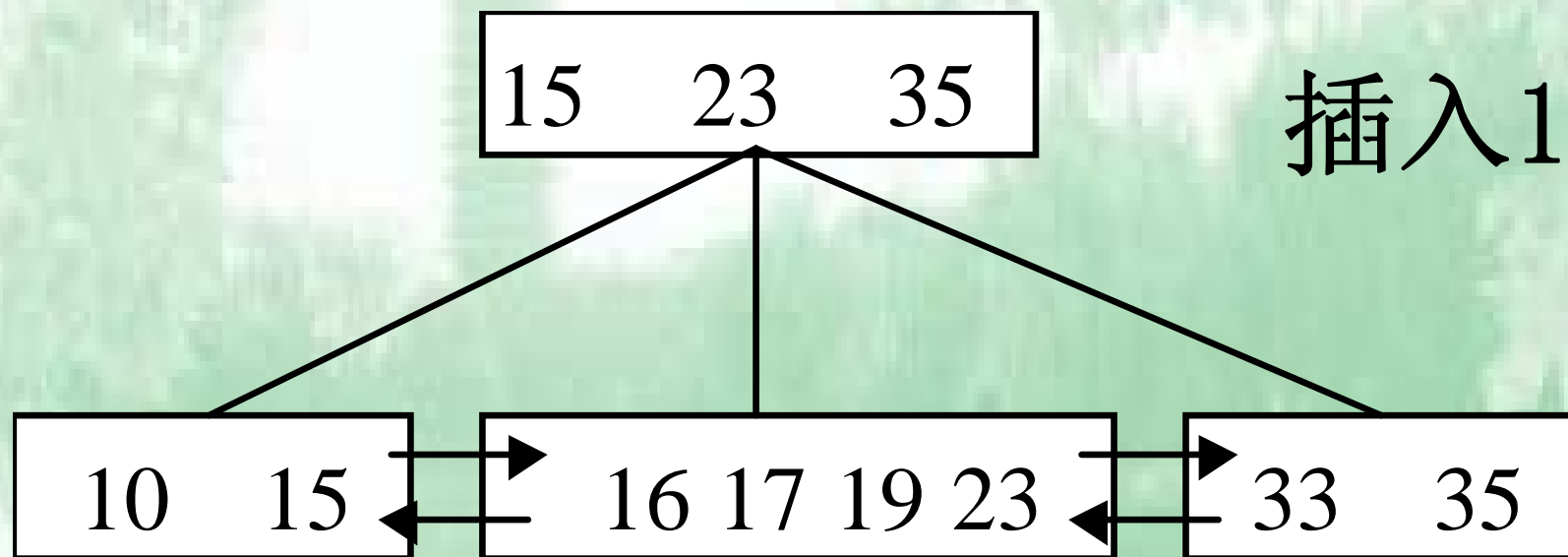
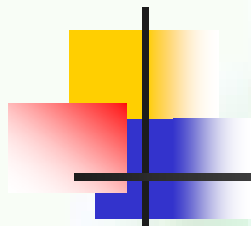
结点满，需要分裂





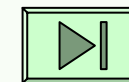
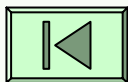
插入17

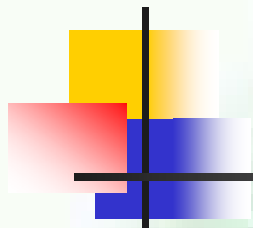




插入19

结点满，需要分裂





结点满，需要分裂

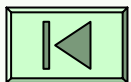
15 17 23 35

10 15

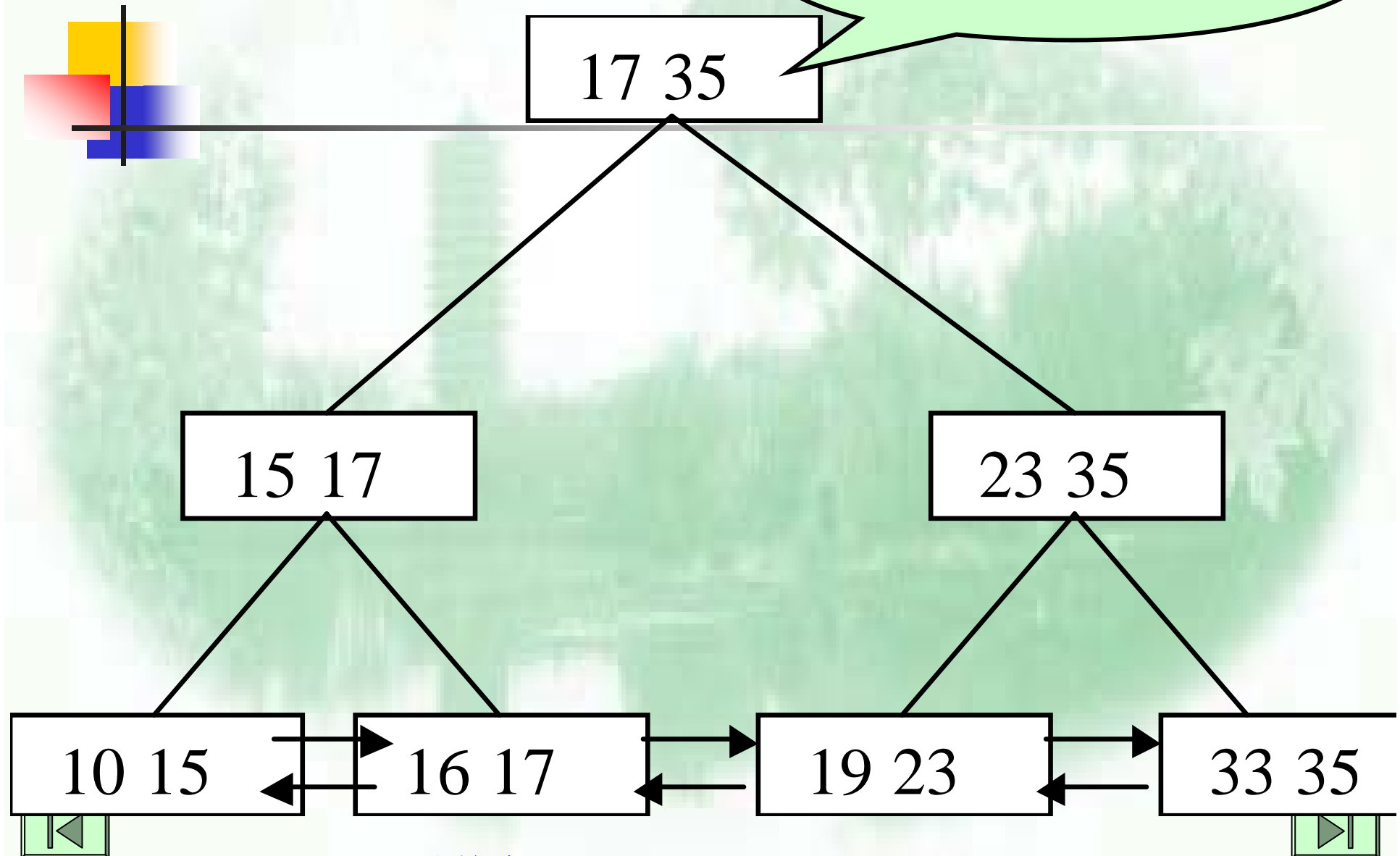
16 17

19 23

33 35



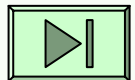
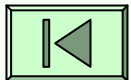
树增加一层



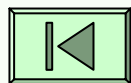
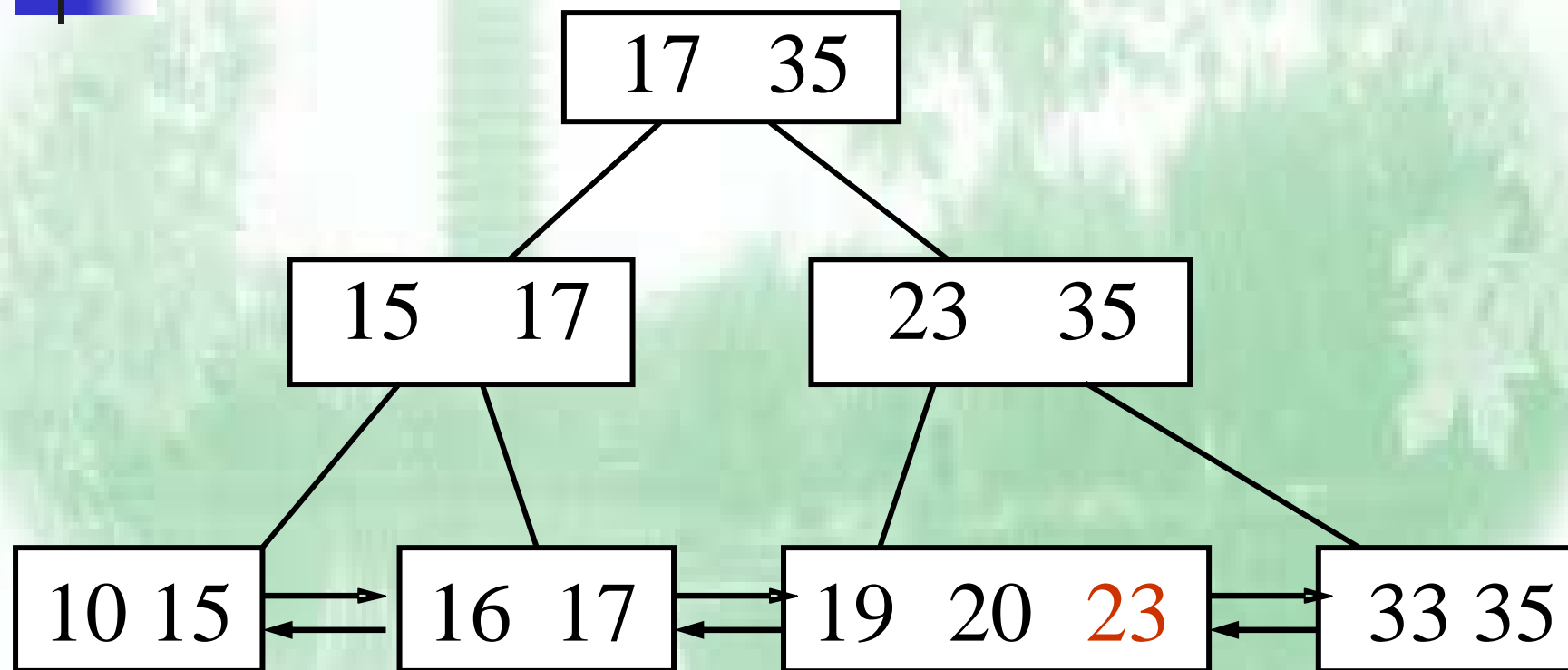


B⁺树的删除

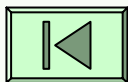
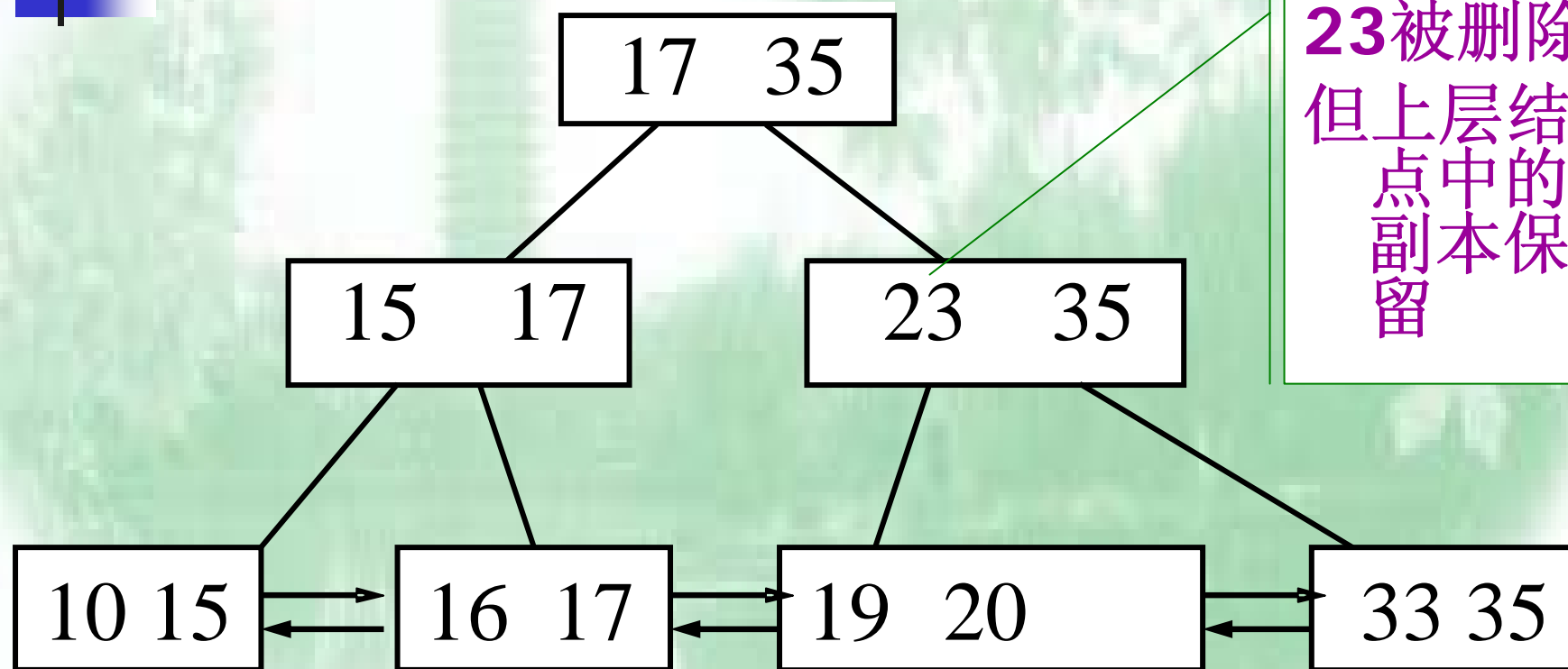
- 当关键码不满时，与左右兄弟进行调整、合并的处理和B树类似
- 关键码在叶结点层删除后，其在上层的复本**可以保留**，做为一个“分界关键码”存在
 - 也可以替换为新的最大关键码（或最小关键码）



m=3, 删除23

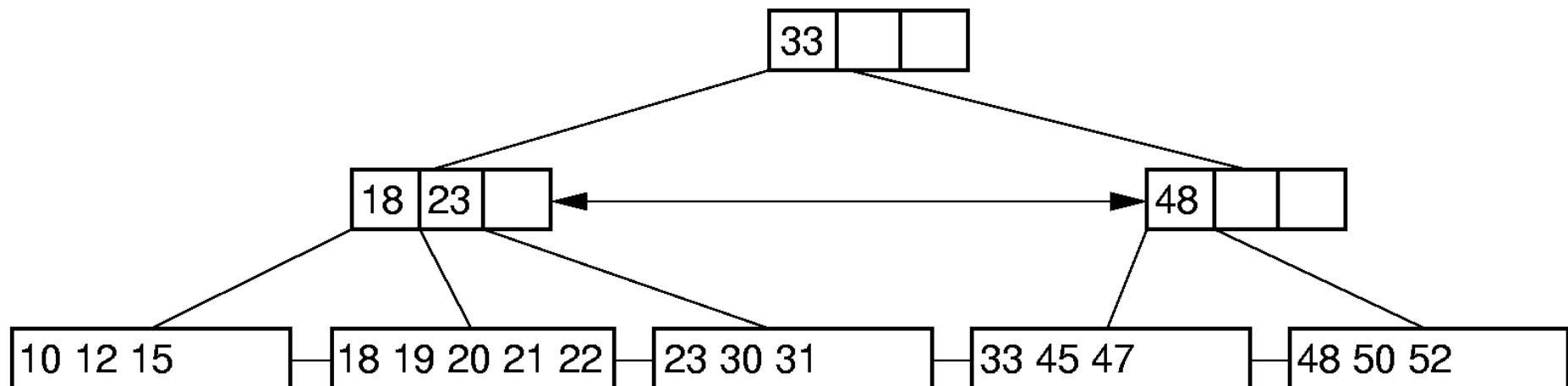


m=3, 删除23

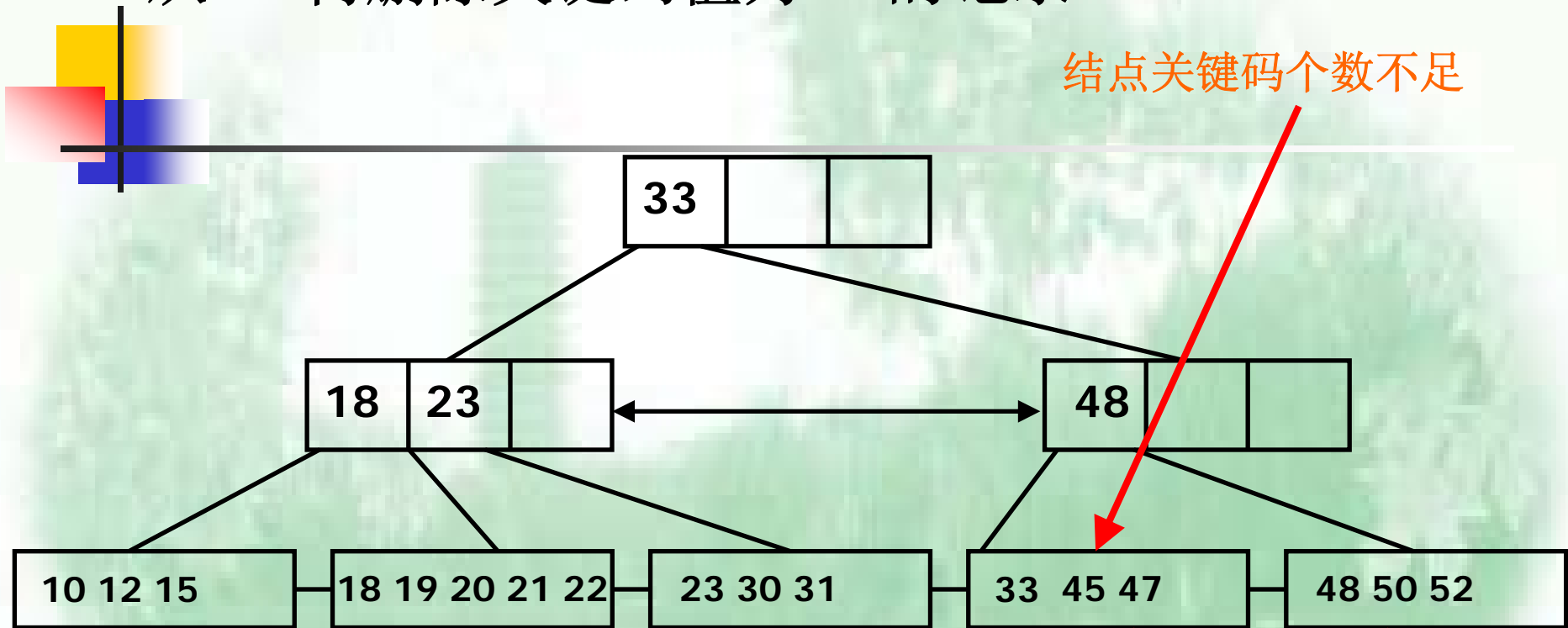


另一种B+树

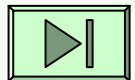
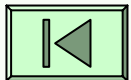
- 叶结点中关键码数目与非叶的不同
 - 内部非叶结点构成**B**树
 - 叶的阶与**B+**树一致
 - 例如，叶结点阶**5**，内部阶**4**



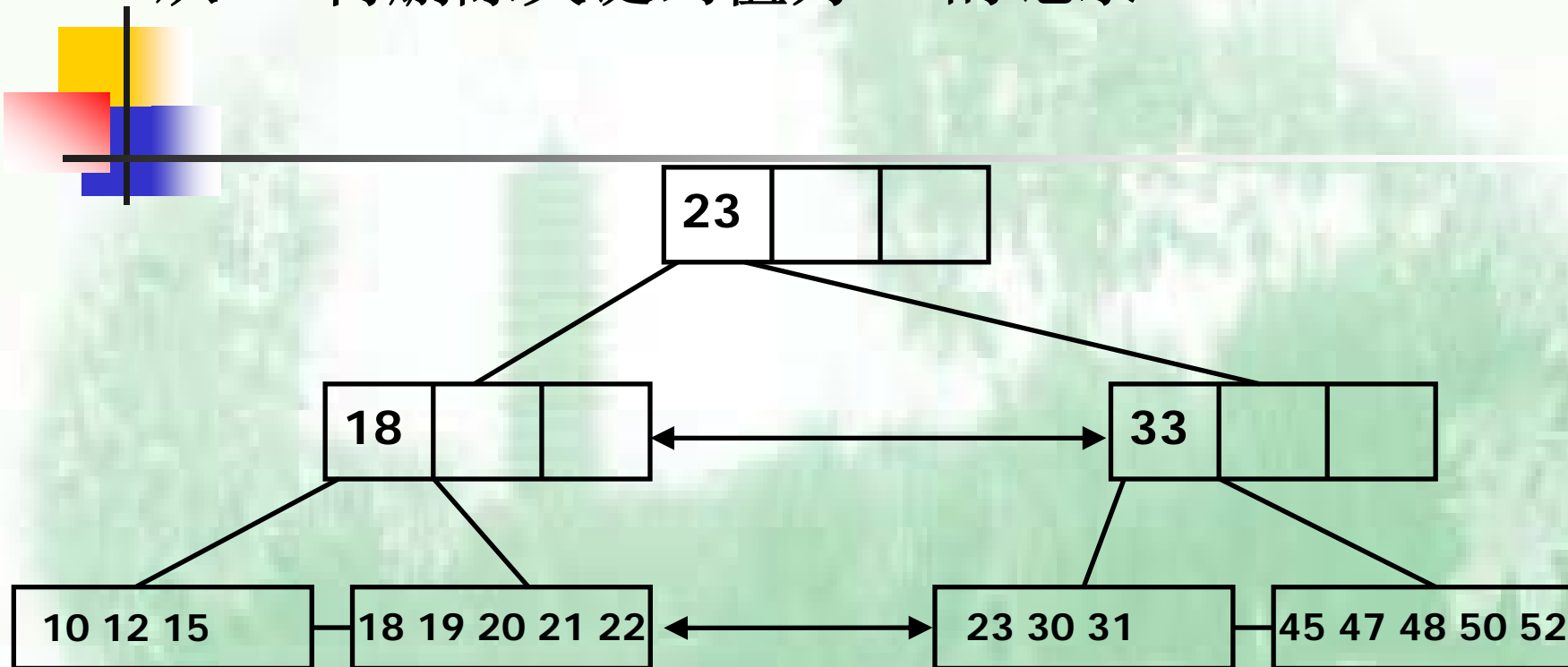
从B+树删除关键码值为33的记录



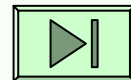
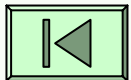
■ 叶结点阶5，内部阶4



从B+树删除关键码值为33的记录



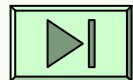
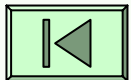
■ 叶结点阶5，内部阶4

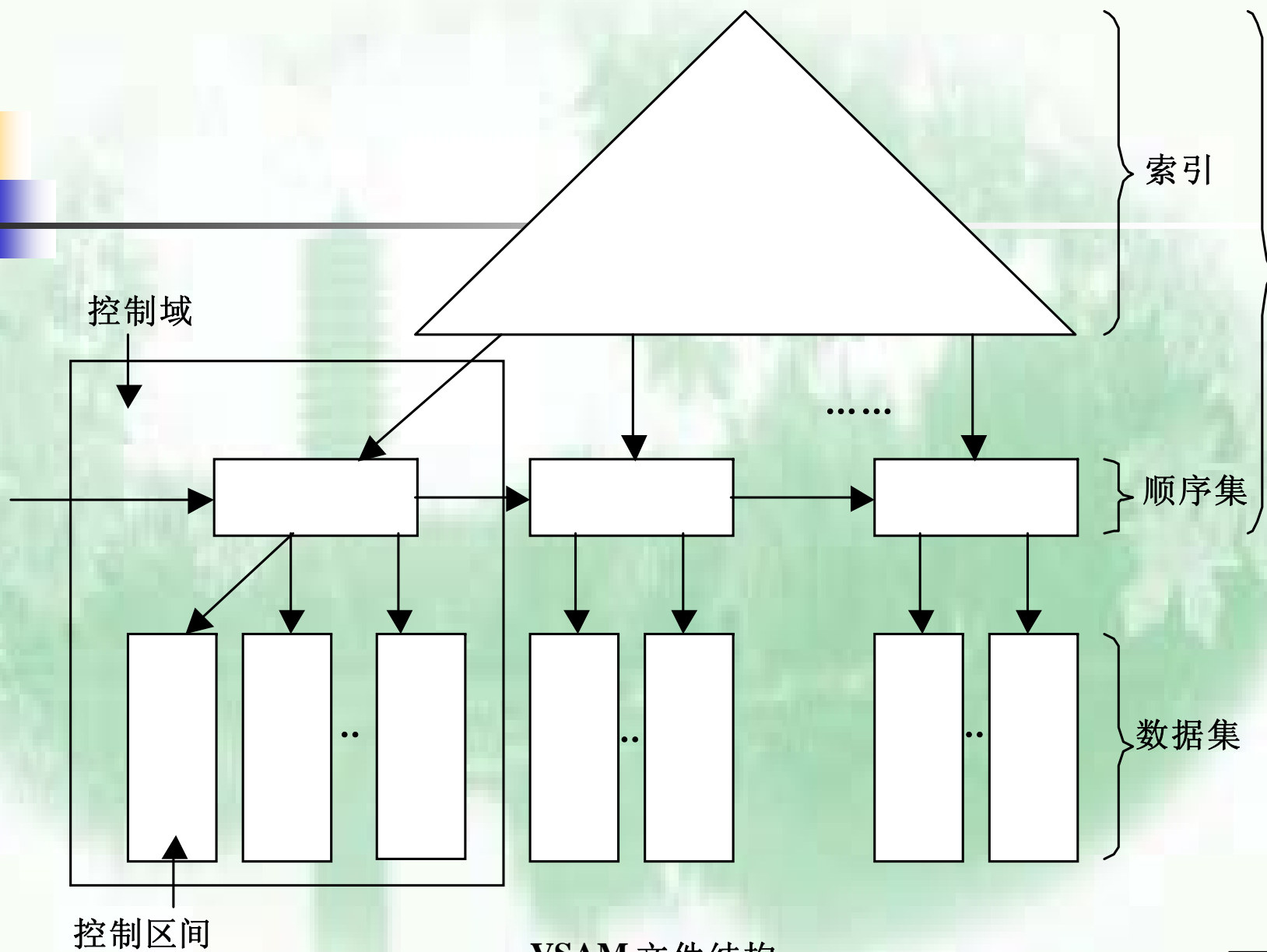
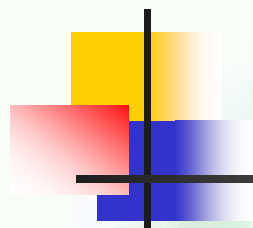




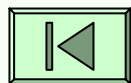
10.5.4 VSAM

- **VSAM(Virtual Storage Access Method)—虚拟存储存取方法**
 - **B⁺树的应用**
 - 一种索引顺序文件的组织方式
 - 与存储设备无关，存储单位是“逻辑”的





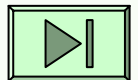
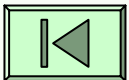
VSAM 文件结构



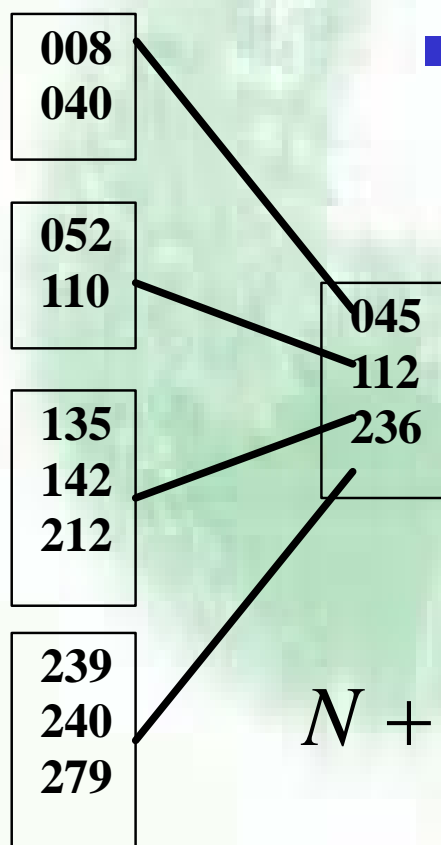


VSAM的组成

- 索引集
- 顺序集（顺序集索引）
 - 和索引集共同形成了B⁺树结构的文件索引
- 数据集
 - 存放文件记录
 - 记录可以是变长的

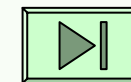
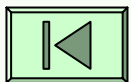


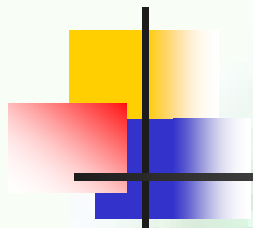
10.5.5 B树的性能分析



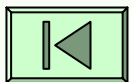
- 包含N个关键码的B树，有N+1个外部空指针，假设外部指针在第k层。
 - 第0层为根，第一层至少两个结点，
 - 第二层至少 $2 \cdot \lceil m/2 \rceil$ 个结点，
 - 第k层至少 $2 \cdot \lceil m/2 \rceil^{k-1}$ 个结点，

$$N + 1 \geq 2 \cdot \lceil m/2 \rceil^{k-1}, \quad k \leq 1 + \log_{\lceil m/2 \rceil} \left(\frac{N+1}{2} \right)$$

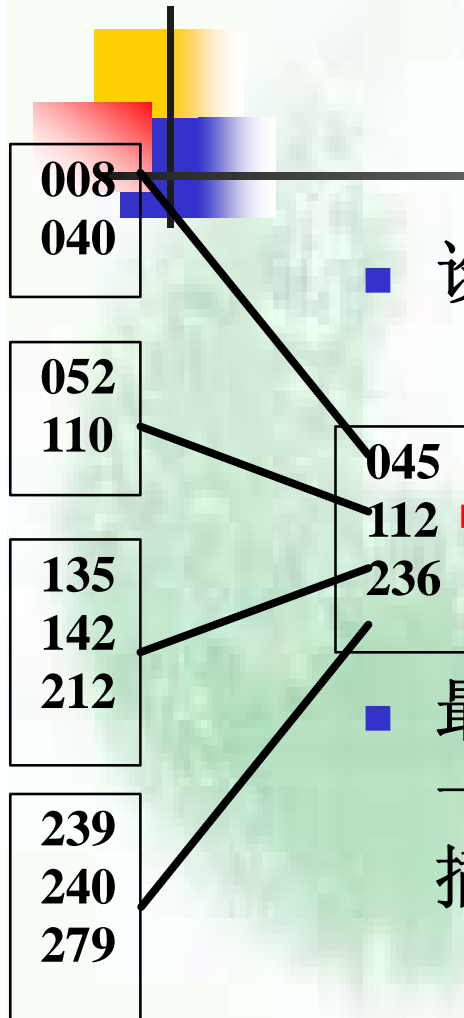




- $N=1,999,998$, $m=199$ 时
 - $k=4$
 - 一次检索最多4层



结点分裂次数



- 设关键码数为 N (空指针 $N+1$)，内部结点数为 p

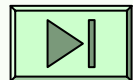
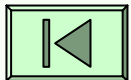
$$N \geq 1 + (\lceil m/2 \rceil - 1)(p - 1)$$

■ 即

$$p - 1 \leq \frac{N - 1}{\lceil m/2 \rceil - 1}$$

- 最差情况下每插入一个结点都经过分裂(除第一个)，即 $p-1$ 个结点都是分裂而来的，则每插入一个关键码平均分裂结点个数为

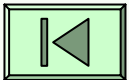
$$s = \frac{p - 1}{N} \leq \frac{N - 1}{(\lceil m/2 \rceil - 1) \cdot N} \leq \frac{1}{\lceil m/2 \rceil - 1}$$



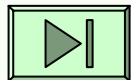
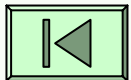
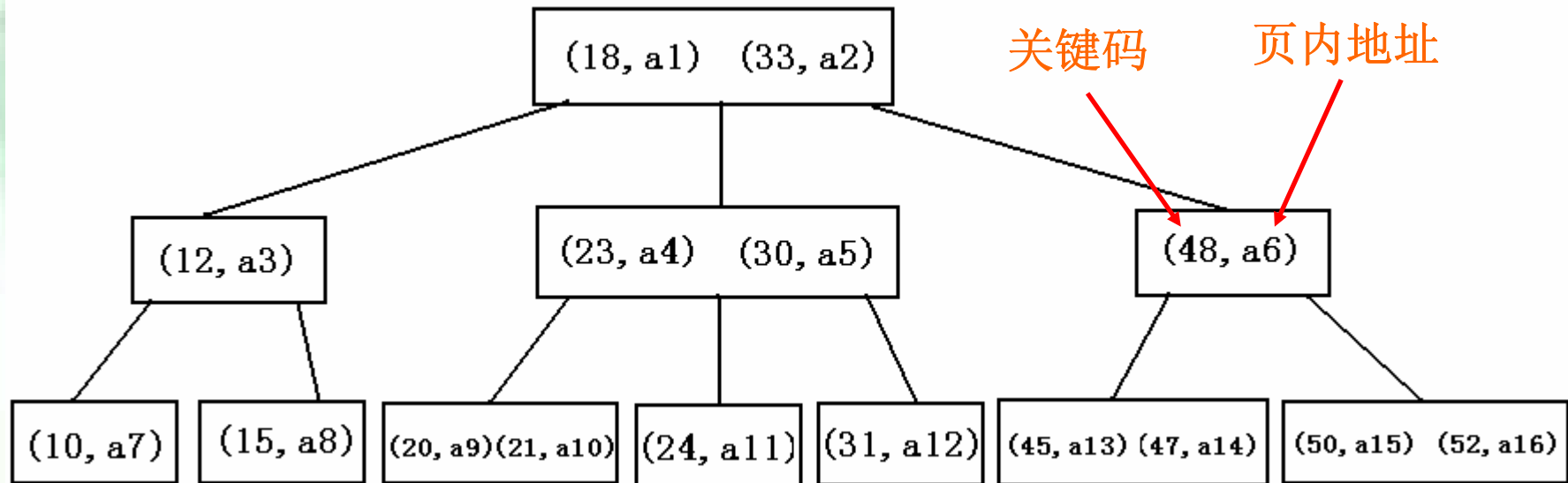
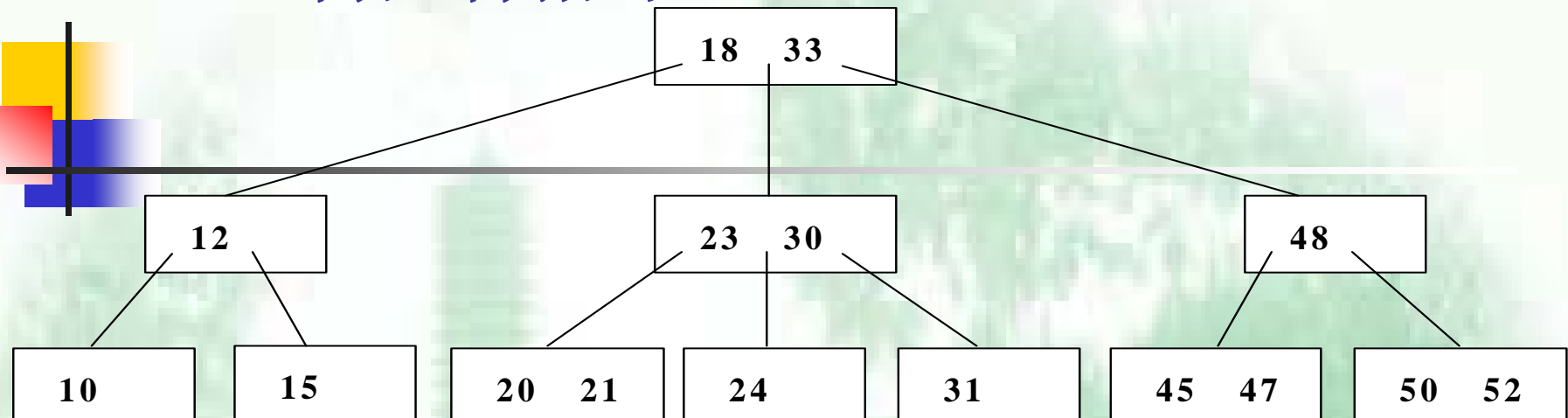


存储效率分析

- B树整个树结构关键码不重写，
 - 每个关键码实质上是（关键码，页块地址）的二元组，
 - 其中的记录地址也称为“隐含指针”



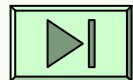
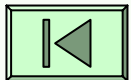
2-3树隐含指针

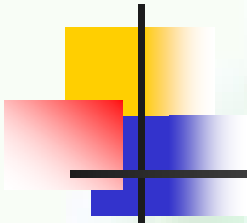


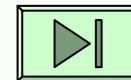
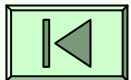


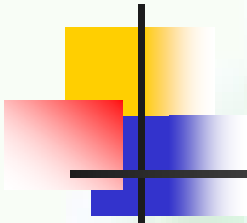
B+树的存储效率实际上更高

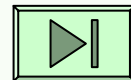
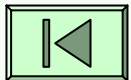
- B+树其实是多级索引形式
 - 最下一层是所有关键码的全集，
 - 因此可以把此层形成顺序链表
 - 非叶层结点中的关键码不需要带隐含指针

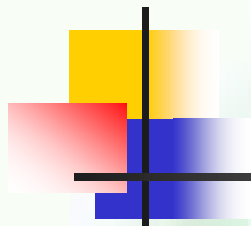


- 
- 假设一个主文件有 N 个记录，假设一个页块可以存 m 个(关键码，子结点页块地址)二元对
 - 假设B+树平均每个结点有 $0.75m$ 个子结点
 - 充盈度为 $(1+0.5)/2 = 75\%$
 - 因此B+树的高度为 $\lceil \log_{0.75m} N \rceil$



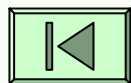
- 
- 可以容纳 m 个(关键码, 子结点页块指针), 假设关键码所占字节数与指针相同
 - 可以容纳B树的(关键码, 隐含指针, 子结点页块指针)最多为 $2m/3$ (B树为 $0.67m$ 阶)。
 - 假设B树充盈度也是75%, 则B树结点有 $0.5m$ 个子结点
 - B树的高度为 $\lceil \log_{0.5m} N \rceil$





B+树应用得更为广泛

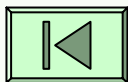
- B+树的存储效率更高
 - 检索层次更少（树较矮）
- 操作更方便
 - 插入删除操作更方便
 - 树型结构随机存取
 - 叶层的拉链顺序访问
- 因此，B+树应用得更为广泛



10.5 动态索引和静态索引 性能的比较

比较：

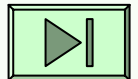
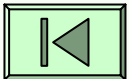
- 基于多分树的静态索引
- 基于**B**树和**B⁺**树的动态索引





动态索引的优点

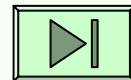
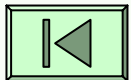
- 能保持较高的查找效率
- 动态地分配和释放存储
- 动态索引结构不需要进行文件再组织





静态索引的缺点

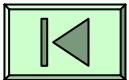
- 多次插入、删除后
 - 溢出区中记录越来越多，溢出区拉链越来越长，大大降低查找效率
 - 有的数据区却有很多空单元处于无用状态
- 严重影响空间利用率，需要进行文件再组织





动态索引的问题

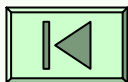
- 要考虑并行策略
- 辅助索引维护困难
- 索引层数多

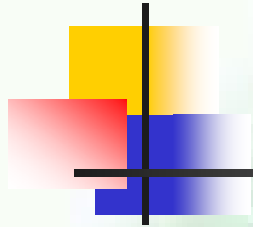




补充：位图索引

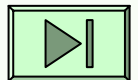
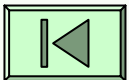
- **B**树适合于查找并取回少量记录的情况
- 对于数据仓库的复杂交互式查询，**B**树有三个缺点：
 - **B**树对唯一值极少的（低基数）数据字段几乎毫无价值
 - 在数据仓库中构造和维护索引的代价高
 - 对于带有分组及聚合条件的复杂查询无能为力





Bit-Map（位图）索引

- 字段**F**的一个位图索引是一个长度为**n**的位向量的集合（**n**为文件的记录数）

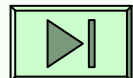
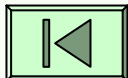


对于数据库表的位图索引

date store state class sales **State=NY** Class=A

3/1	32	NY	A	6	1	1
3/1	36	AL	A	9	0	1
3/1	38	NY	B	5	1	0
3/1	41	AK	A	11	0	1
3/1	43	NY	A	9	1	1
3/1	46	AK	B	3	0	0

state=AK	state=AL	...	state=NY
0	0		1
0	1		0
0	0		1
1	0		0
0	0		1
1	0		0



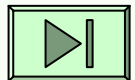
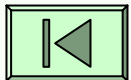


特征文件

- **Signature file** （也译为“签名文件”）
- 倒排表(30,foo),(30,bar),(30,baz)
(40,baz),(40,bar),(50,foo)

记录	bar, baz, foo		
30	1	1	1
40	0	1	1
50	0	0	1

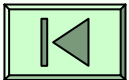
1表示在那个记录中出现了相应的字段值





位图索引特点

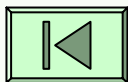
- 按“列”为单位存储数据
- 列数据比行数据更易进行压缩，可节省**50%**的磁盘空间
- 索引空间比**B**树小





总结

- 基本概念
- 10.1 线性索引
- 10.2 静态索引
- 10.3 倒排索引
- 10.4 动态索引
- 10.5 动态、静态索引性能比较
- 讨论和补充





The End!

