

王道考研——数据结构

WWW.CSKAOYAN.COM

第六章 查找

本章总览



王道考研/CSKAOYAN.COM

本节内容

查找

基本概念

王道考研/CSKAOYAN.COM

查找

查找

在数据集中寻找满足某种条件的数据元素的过程。

查找结果分为查找成功和查找失败。

查找表

用于查找的数据集合，由同一种数据类型(或记录)的组成，可以是一个数组或链表等数据类型

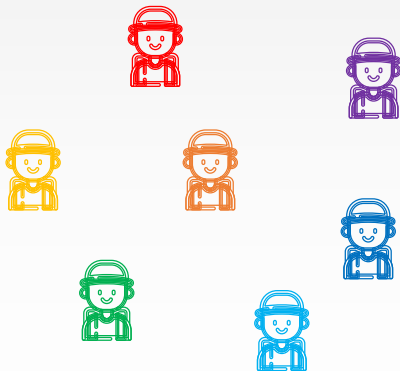
操作：

- 查询某个特定的数据元素是否在查找表中
- 检索满足条件的某个特定的数据元素的各種属性
- 在查找表中插入一个数据元素
- 从查找表中删除一个数据元素

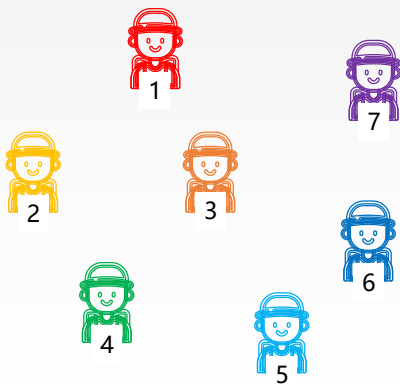
静态查找表

动态查找表

王道考研/CSKAOYAN.COM



查找



关键字

数据元素中唯一标识该元素的某个数据项的值，使用基于关键字的查找，查找结果应该是唯一的。

平均查找长度

查找时，关键字比较次数的平均值：

$$ASL = \sum_{i=1}^n P_i C_i$$

王道考研/CSKAOYAN.COM

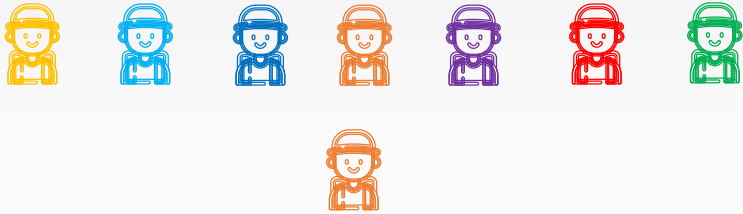
本节内容

查找

顺序查找

王道考研/CSKAOYAN.COM

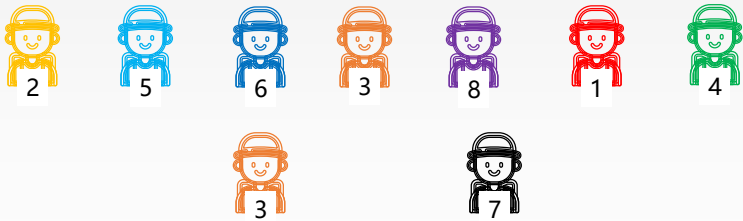
顺序查找



顺序查找 又称线性查找，主要用于在**线性表**中进行查找。

王道考研/CSKAOYAN.COM

顺序查找



★ 对无序线性表进行顺序查找，查找失败时要遍历整个线性表

王道考研/CSKAOYAN.COM

顺序查找

```
typedef struct{
    ElemType *elem;
    int TableLen;
}SSTable;

int Search_Seq(SSTable ST, ElemType key){
    ST.elem[0]=key;
    for(int i=ST.TableLen; ST.elem[i]!=key; i--);
    return i;
}
```

$$ASL_{成功} = \sum_{i=1}^n P_i C_i = \sum_{i=1}^n P_i (n-i+1) = \sum_{i=1}^n \frac{1}{n} * (n-i+1) = \frac{n+1}{2}$$

$$ASL_{失败} = \sum_{i=1}^n P_i C_i = \sum_{i=1}^n \frac{1}{n} * (n+1) = n+1$$

王道考研/CSKAOYAN.COM

顺序查找



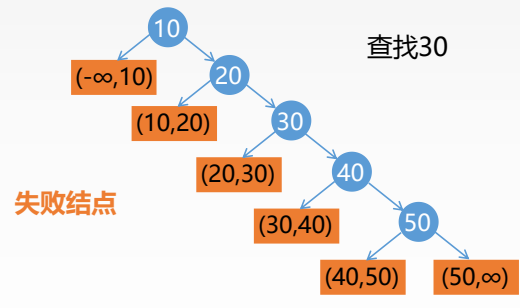
★ 对关键字有序线性表进行顺序查找，查找失败时不一定要遍历整个线性表

王道考研/CSKAOYAN.COM

顺序查找

判定树 描述查找过程的二叉排序树

(10, 20, 30, 40, 50)

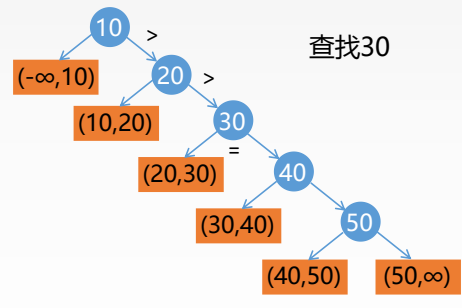


王道考研/CSKAOYAN.COM

顺序查找

判定树 描述查找过程的二叉排序树

(10, 20, 30, 40, 50)

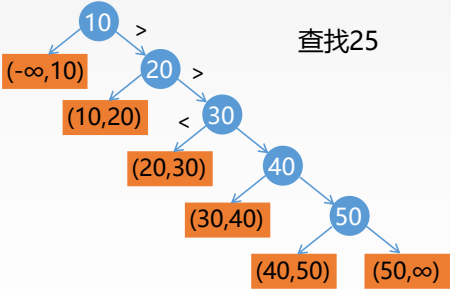


王道考研/CSKAOYAN.COM

顺序查找

判定树 描述查找过程的二叉排序树

(10, 20, 30, 40, 50)



$$ASL_{失败} = \sum_{i=1}^n P_j C_j = \sum_{j=1}^n \frac{1}{n+1} * (l_j - 1) = \frac{1+2+...+n+n}{n+1} = \frac{n}{2} + \frac{n}{n+1}$$

王道考研/CSKAOYAN.COM

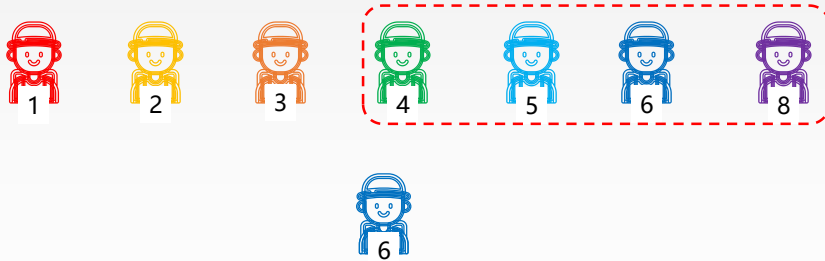
本节内容

查找

折半查找

王道考研/CSKAOYAN.COM

折半查找



王道考研/CSKAOYAN.COM

折半查找

折半查找 又称二分查找，仅适用于有序的顺序表

算法思想

首先将给定值key与表中中间位置元素的关键字比较，
若相等，则返回该元素的位置；
若不等，则在前半部分或者是后半部分进行查找。

查找序列升序时，
若key小于中间元素，则查找前半部分；
若key大于中间元素，则查找后半部分。


重复该过程，直到找到查找的元素为止；或查找失败。

王道考研/CSKAOYAN.COM

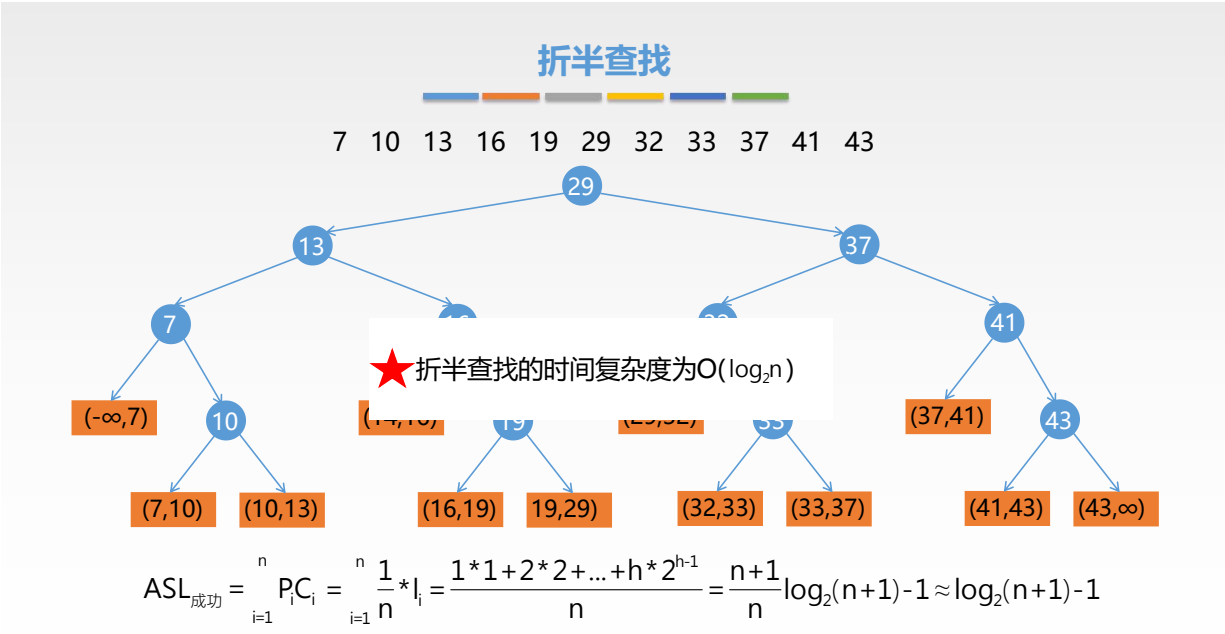
查找16



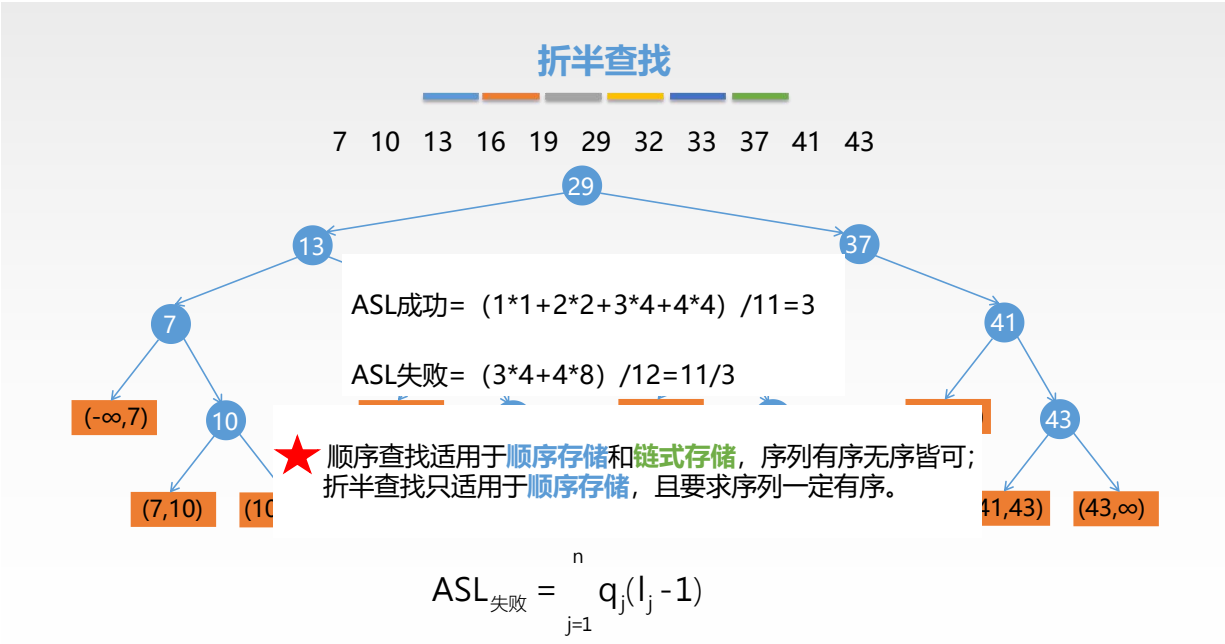
折半查找

查找35 





王道考研/CSKAOYAN.COM



王道考研/CSKAOYAN.COM

本节内容

查找

分块查找

王道考研/CSKAOYAN.COM

分块查找

3

5

7



王道考研/CSKAOYAN.COM

分块查找

分块查找 又称索引顺序查找，它吸取了顺序查找和折半查找各自的优点，既有动态结构，又适于快速查找。

? 如何分块

- 将查找表分为若干子块。块内的元素可以无序，但块间是有序的，即对于所有块有第*i*块的最大关键字小于第*i*+1块的所有记录的关键字。
- 建立索引表，索引表中的每个元素含有各块的最大关键字和各块中的第一个元素的地址，索引表按关键字有序排列。

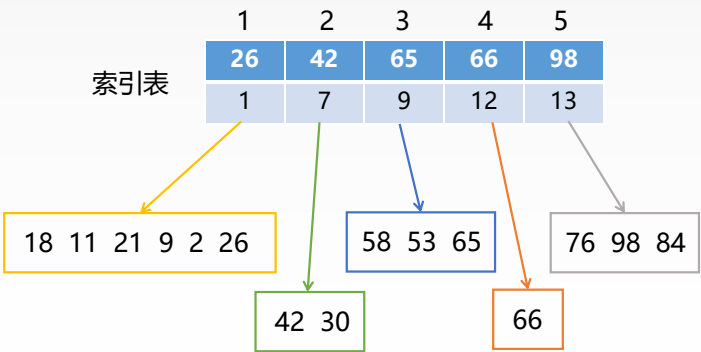
块内无序块间有序

王道考研/CSKAOYAN.COM

分块查找

分块查找 又称索引顺序查找，它吸取了顺序查找和折半查找各自的优点，既有动态结构，又适于快速查找。

18 11 21 9 2 26 42 30 58 53 65 66 76 98 84

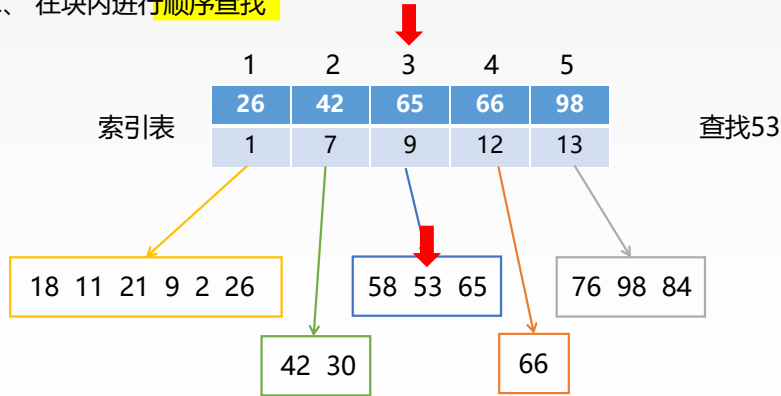


王道考研/CSKAOYAN.COM

分块查找

? 如何查找

- 1、在索引表中确定待查记录所在的块，可以顺序查找或折半查找索引表。
- 2、在块内进行顺序查找



王道考研/CSKAOYAN.COM

分块查找

分块查找的平均查找长度为索引查找(L_I)和块内查找(L_S)之和。
设长度为n的查找表均匀分为b块，每块有s个记录。

$$ASL_{成功} = L_I + L_S$$

· 若块内和块间均采用顺序查找。

$$ASL_{成功} = L_I + L_S = \frac{b+1}{2} + \frac{s+1}{2} = \frac{s^2 + 2s + n}{2s} \xrightarrow[\min]{s=\sqrt{n}} \sqrt{n} + 1$$

· 若块内采用顺序查找，块间采用折半查找。

$$ASL_{成功} = L_I + L_S = \lceil \log_2(b+1) \rceil + \frac{s+1}{2}$$

王道考研/CSKAOYAN.COM

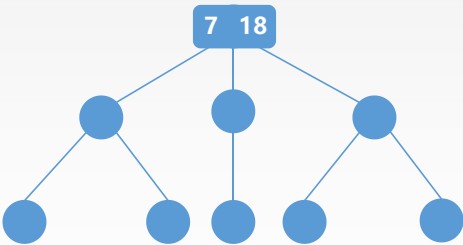
本节内容

查找

B树

王道考研/CSKAOYAN.COM

B-树



王道考研/CSKAOYAN.COM

B-树

B树 又称多路平衡查找树，B树中所有结点的孩子结点数的最大值称为B树的阶。

一棵m阶B树或为空树，或为满足如下特性的m叉树：

- 1) 树中每个结点至多有m棵子树（即至多含有m-1个关键字）
- 2) 若根结点不是终端结点，则至少有两棵子树
- 3) 除根结点外的所有非叶结点至少有 $\lceil m/2 \rceil$ 棵子树（即 $\lceil m/2 \rceil - 1$ 个关键字）
- 4) 非叶结点的结构：

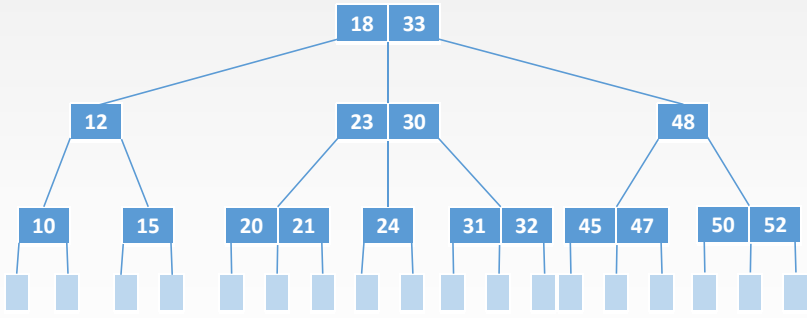
| | | | | | | | | |
|---|----|----|----|----|----|-----|----|----|
| n | P0 | K1 | P1 | K2 | P2 | ... | Kn | Pn |
|---|----|----|----|----|----|-----|----|----|

$K_i (i=1,2,...,n)$ 为结点的关键字, $K_1 < K_2 < ... < K_n$,
 $P_i (i=0,1,...,n)$ 为子树根结点的指针, P_{i-1} 所指子树的关键字均小于 K_i
 P_i 所指子树的关键字均大于 K_i

- 5) 所有的叶结点都出现在同一层次上，并不带任何信息

王道考研/CSKAOYAN.COM

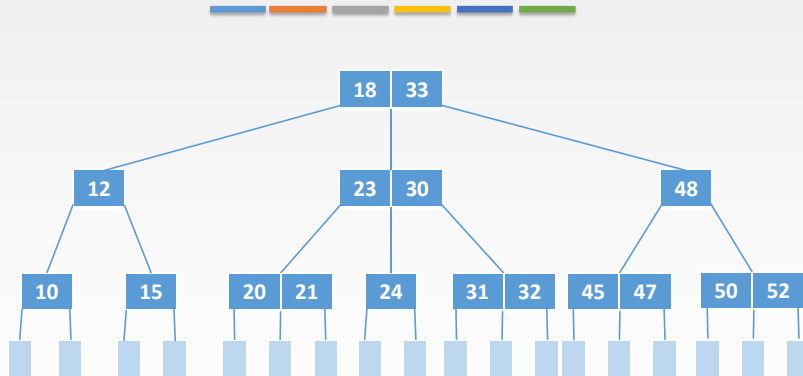
B-树



- 1) 树中每个结点至多有m棵子树（即至多含有m-1个关键字）

王道考研/CSKAOYAN.COM

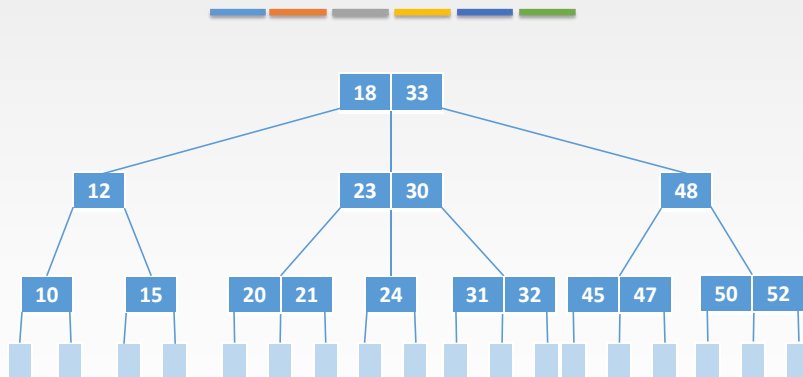
B-树



2) 若根结点不是终端结点，则至少有两棵子树

王道考研/CSKAOYAN.COM

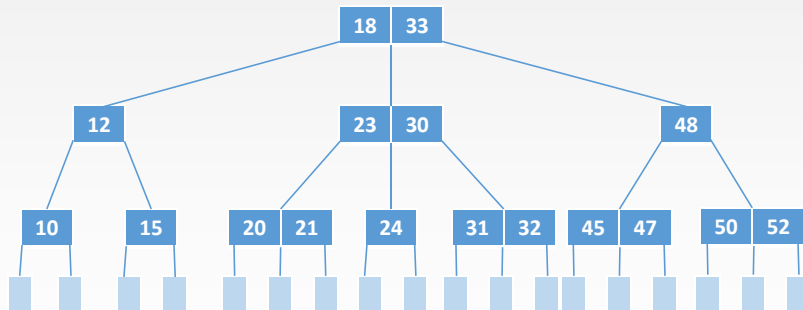
B-树



3) 除根结点外的所有非叶结点至少有 $\lceil m/2 \rceil$ 棵子树 (即 $\lceil m/2 \rceil - 1$ 个关键字)

王道考研/CSKAOYAN.COM

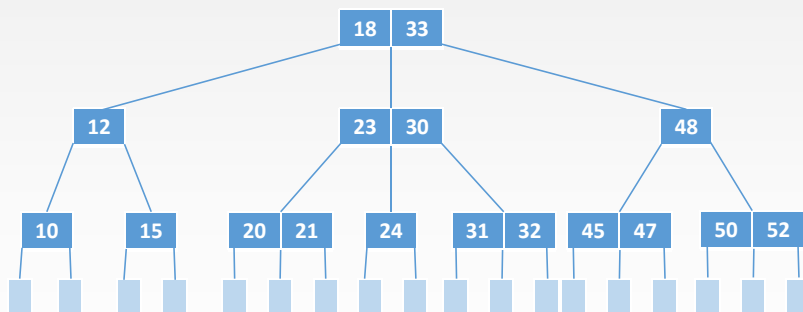
B-树



$K_i(i=1,2,\dots,n)$ 为结点的关键字, $K_1 < K_2 < \dots < K_n$,
 $P_i(i=0,1,\dots,n)$ 为子树根结点的指针, P_{i-1} 所指子树的关键字均小于 K_i
 P_i 所指子树的关键字均大于 K_i

王道考研/CSKAOYAN.COM

B-树



5) 所有的叶结点都出现在同一层次上，并不带任何信息

王道考研/CSKAOYAN.COM

B-树

? n个关键字, 阶数为m, 高度为h的B树

$$\log_m(n+1) \leq h \leq$$

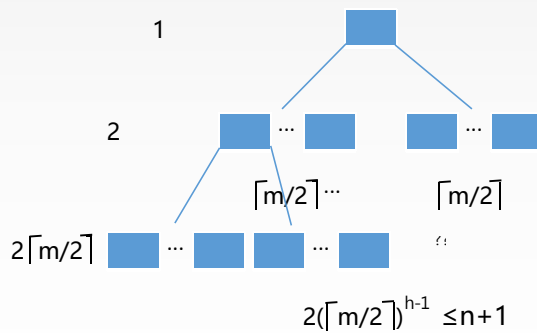
$$n \leq (m-1)(1+m+m^2+\dots+m^{h-1}) = m^h - 1$$

王道考研/CSKAOYAN.COM

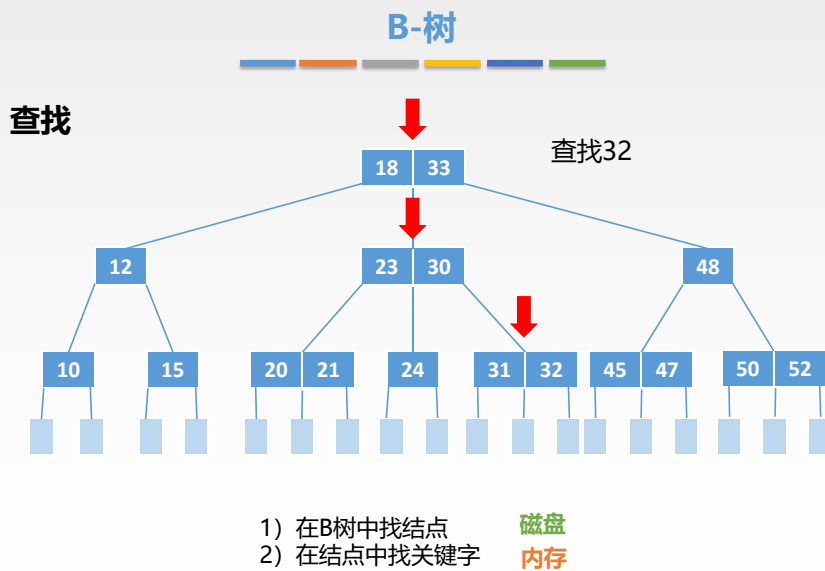
B-树

? n个关键字, 阶数为m, 高度为h的B

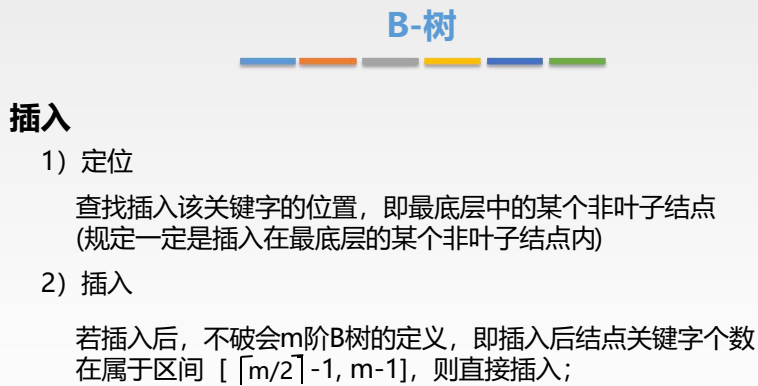
$$\log_m(n+1) \leq h \leq \log_{\lceil m/2 \rceil}((n+1)/2) + 1$$



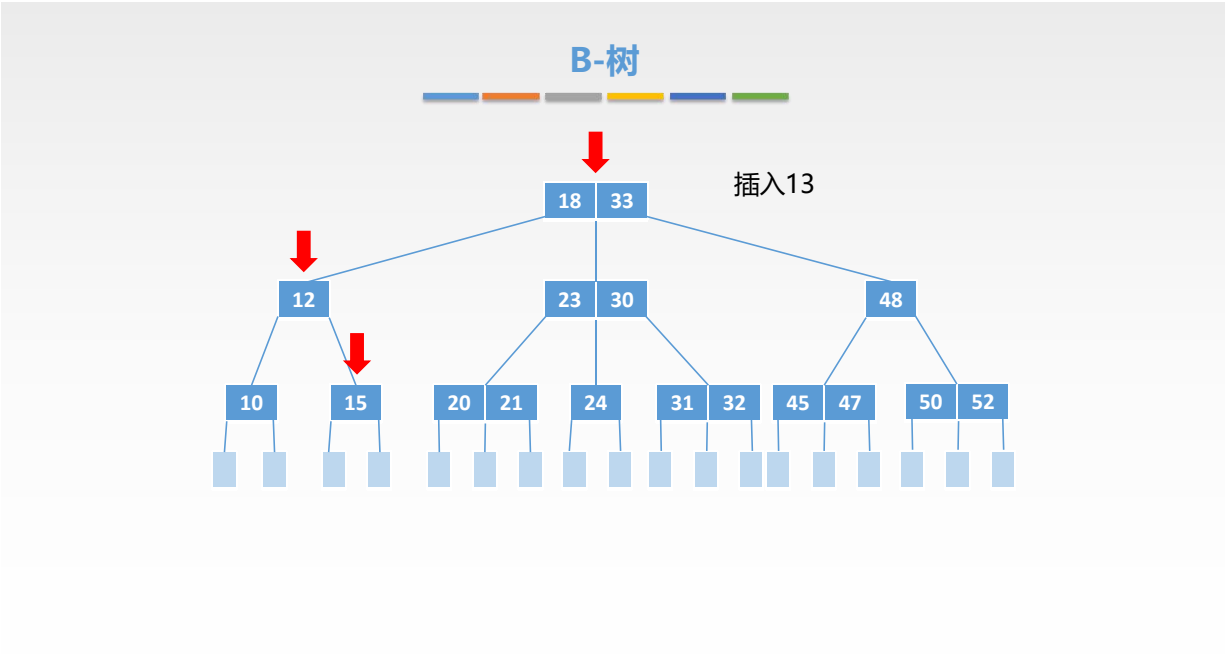
王道考研/CSKAOYAN.COM



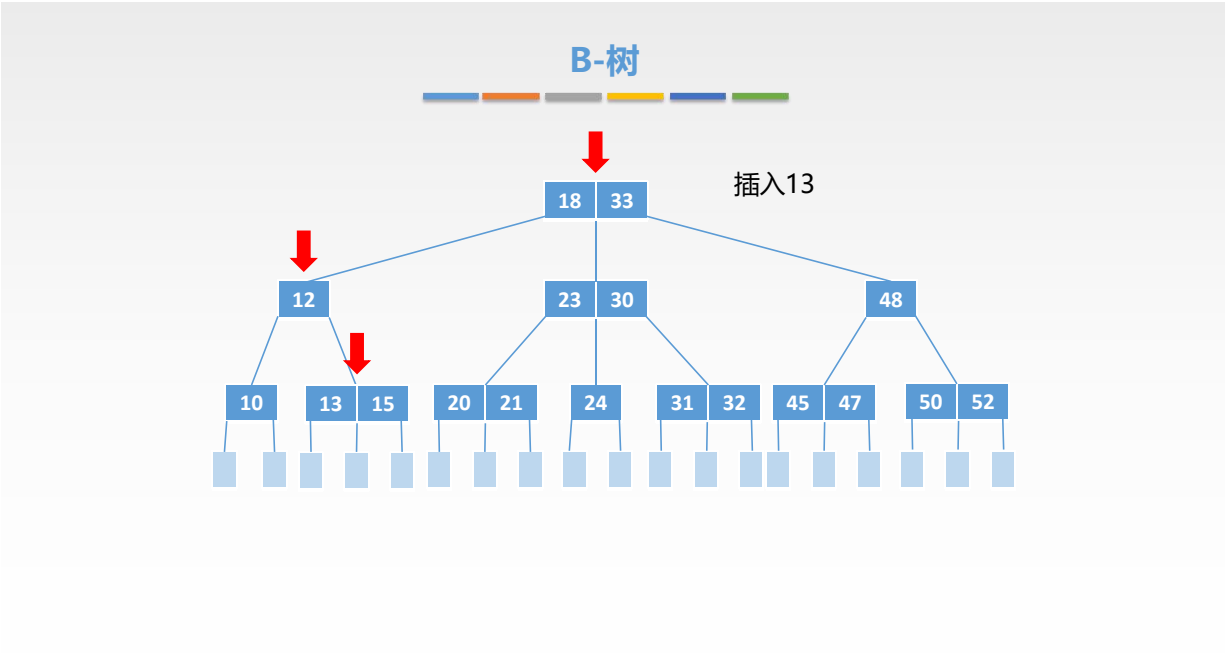
王道考研/CSKAOYAN.COM



王道考研/CSKAOYAN.COM



王道考研/CSKAOYAN.COM



王道考研/CSKAOYAN.COM

B-树

插入

1) 定位

查找插入该关键字的位置，即最底层中的某个非叶子结点
(规定一定是插入在最底层的某个非叶子结点内)

2) 插入

若插入后，不破坏m阶B树的定义，即插入后结点关键字个数
在属于区间 $[\lceil m/2 \rceil - 1, m-1]$ ，则直接插入；

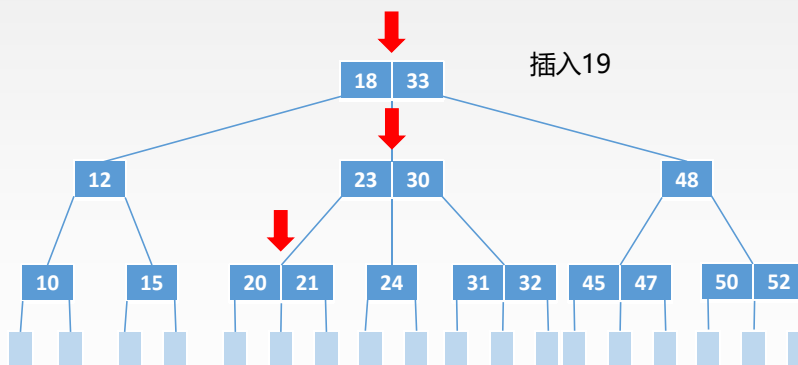
若插入后，关键字数量大于m-1，则对插入后的结点进行分裂操作；

分裂：

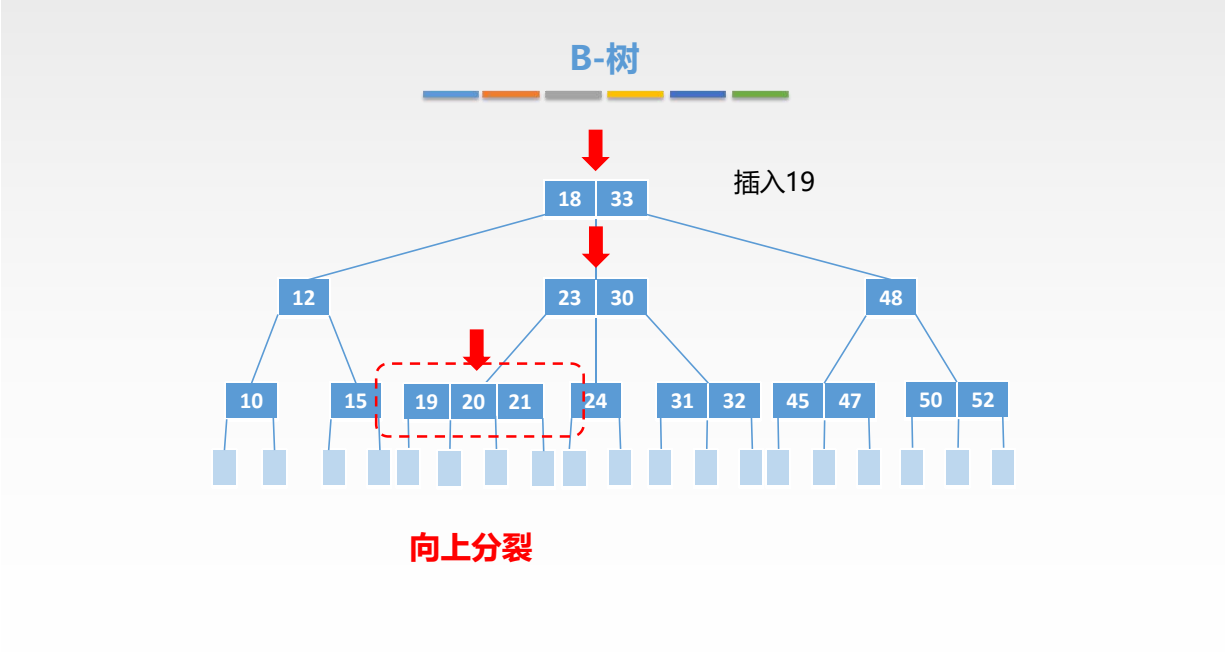
插入后的结点中间位置($\lceil m/2 \rceil$)关键字并入父结点中，
中间结点左侧结点留在原先的结点中，右侧结点放入新的节点中，
若并入父节点后，父结点关键字数量超出范围，继续想上分裂，直
到符合要求为止。

王道考研/CSKAOYAN.COM

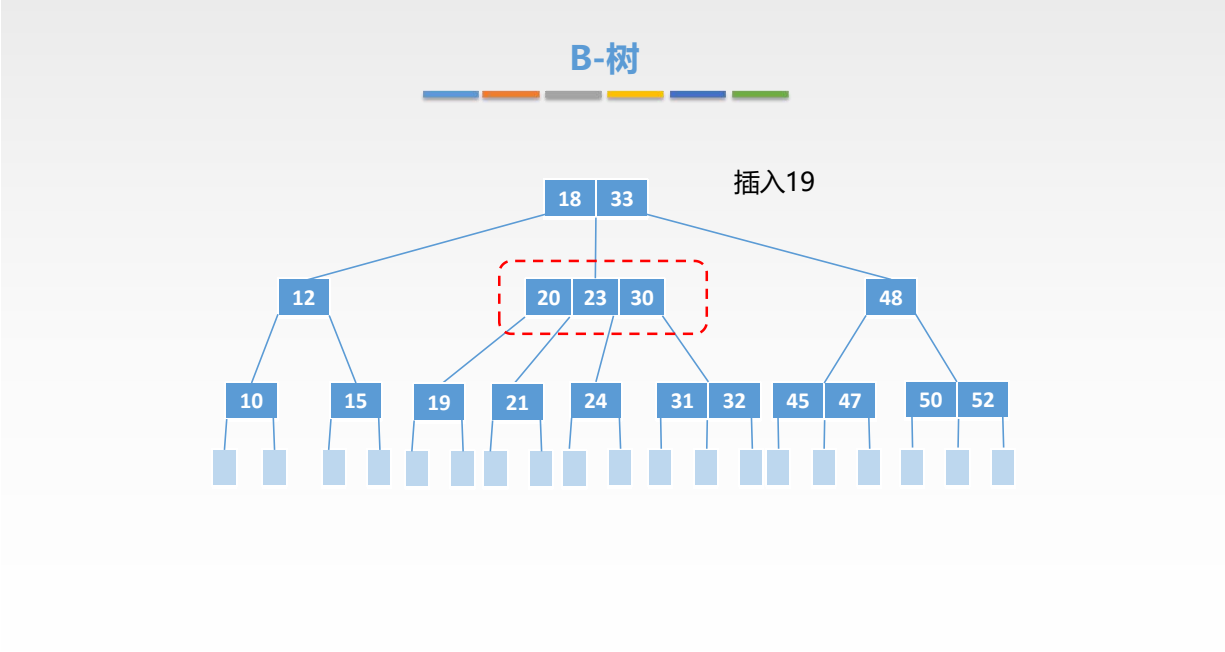
B-树



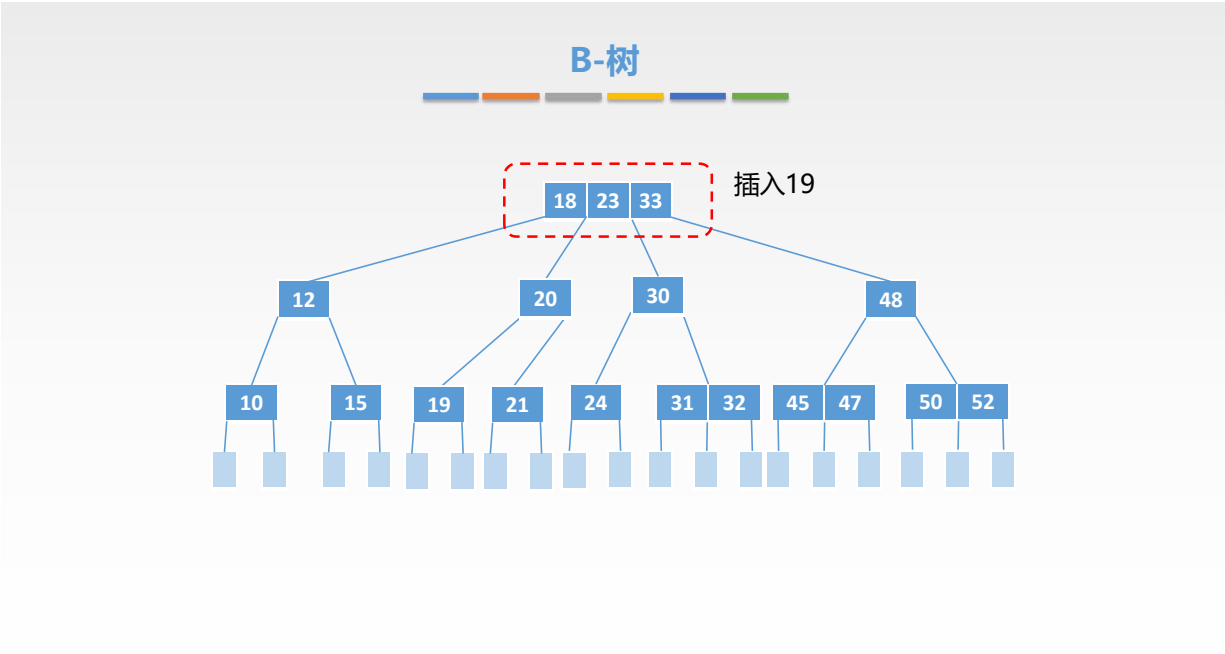
王道考研/CSKAOYAN.COM



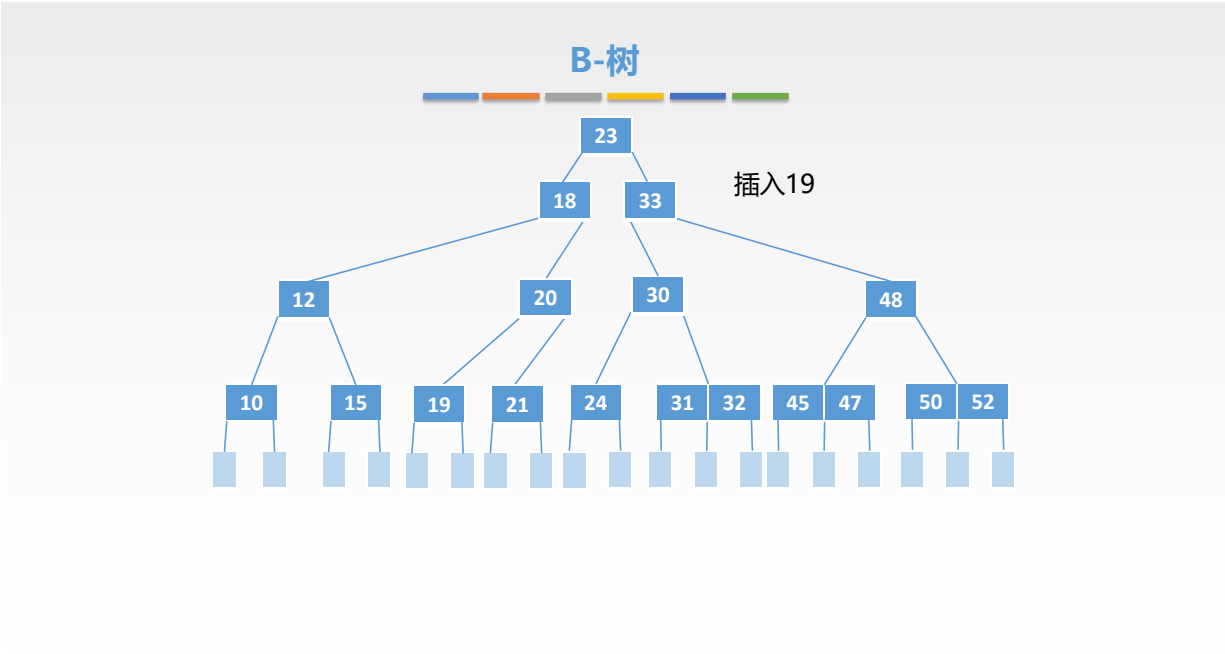
王道考研/CSKAOYAN.COM



王道考研/CSKAOYAN.COM



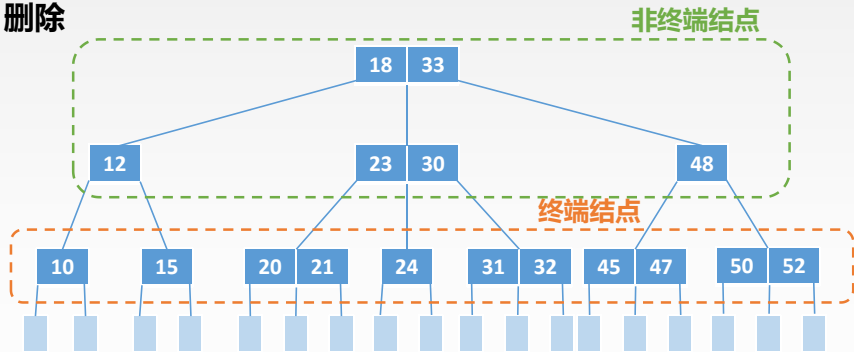
王道考研/CSKAOYAN.COM



王道考研/CSKAOYAN.COM

B-树

删除



王道考研/CSKAOYAN.COM

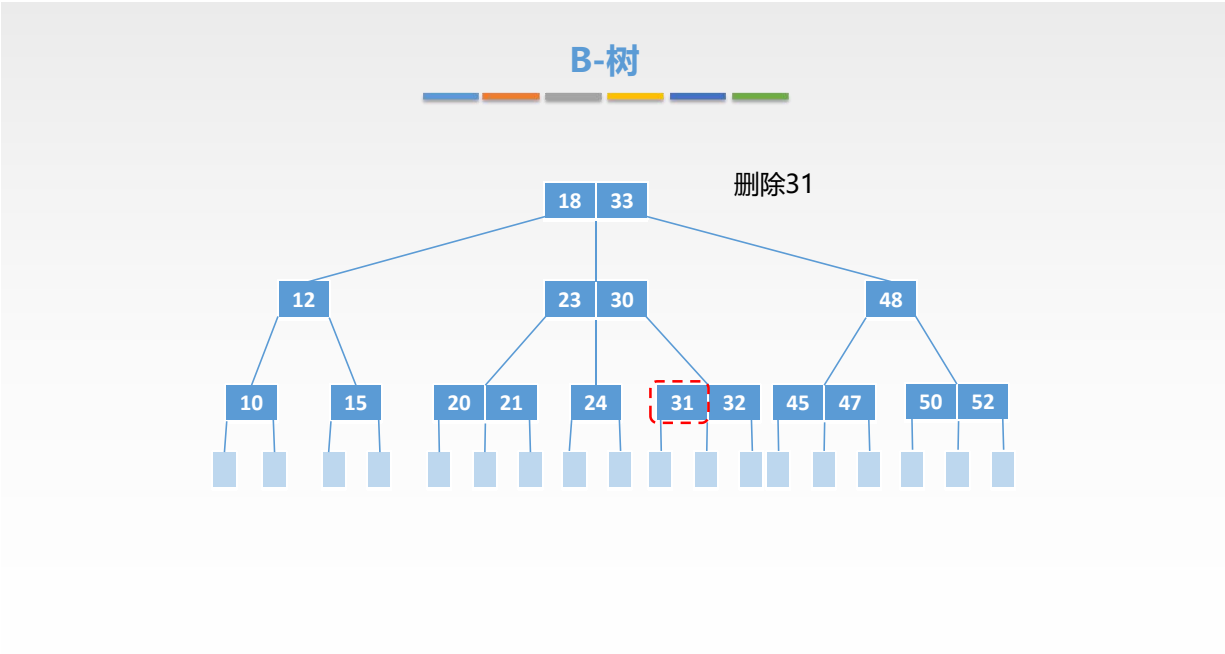
B-树

删除

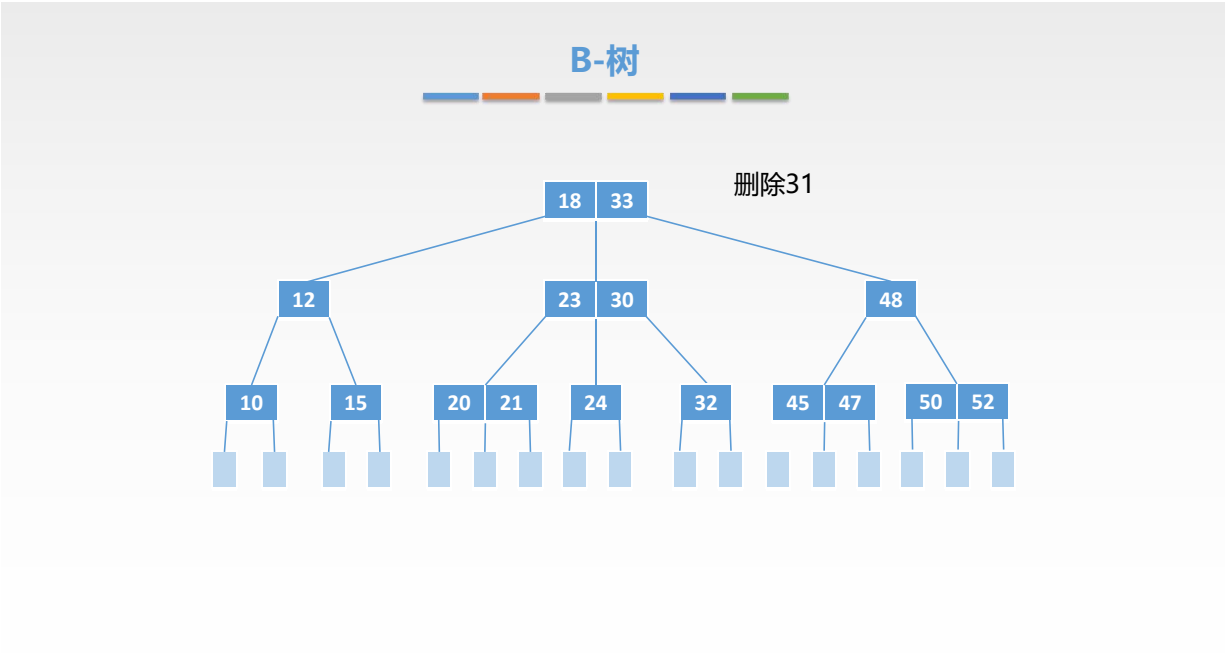
1) 直接删除

若被删除关键字所在结点关键字个数 $> \lceil m/2 \rceil - 1$ ，表明删除后仍满足B树定义，直接删除

王道考研/CSKAOYAN.COM



王道考研/CSKAOYAN.COM



王道考研/CSKAOYAN.COM

B-树

删除

1) 直接删除

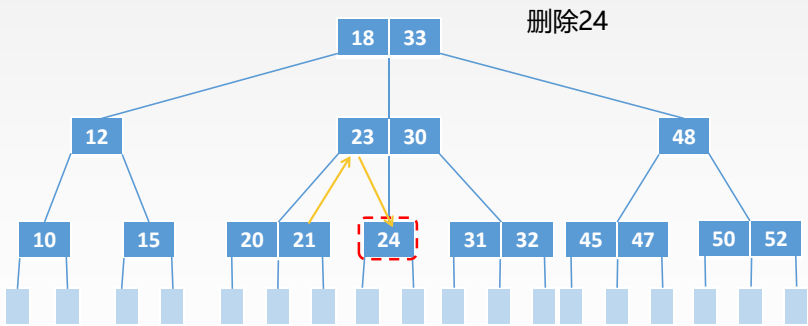
若被删除关键字所在结点关键字总数 $> \lceil m/2 \rceil - 1$ ，表明删除后仍满足B树定义，直接删除

2) 兄弟够借

若被删除关键字所在结点关键字总数 $= \lceil m/2 \rceil - 1$ ，且与此结点邻近的兄弟结点的关键字个数 $\geq \lceil m/2 \rceil$ ，则需要从兄弟结点借一个关键字，此过程需要调整该结点、双亲结点和兄弟结点的关键字

王道考研/CSKAOYAN.COM

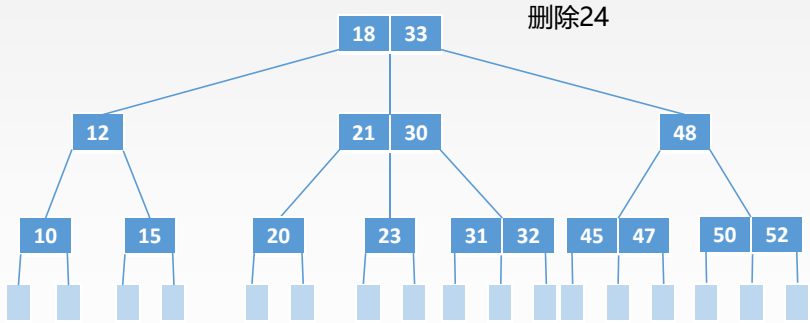
B-树



从左兄弟结点借一个

王道考研/CSKAOYAN.COM

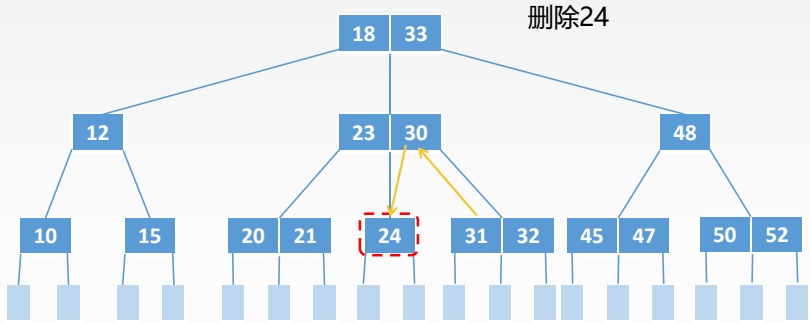
B-树



从左兄弟结点借一个

王道考研/CSKAOYAN.COM

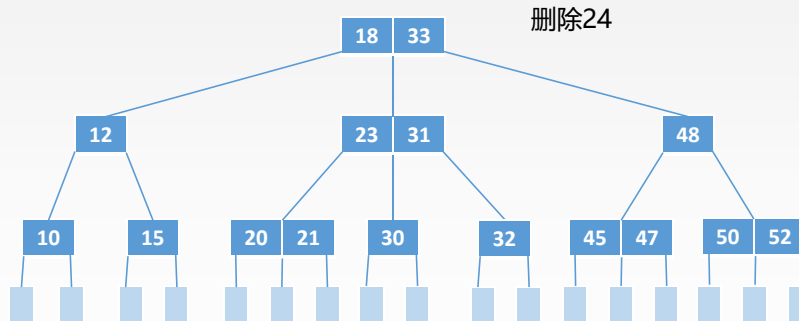
B-树



从右兄弟结点借一个

王道考研/CSKAOYAN.COM

B-树



从右兄弟结点借一个

王道考研/CSKAOYAN.COM

B-树

删除

1) 直接删除

若被删除关键字所在结点关键字总数 $> \lceil m/2 \rceil - 1$ ，表明删除后仍满足B树定义，直接删除

2) 兄弟够借

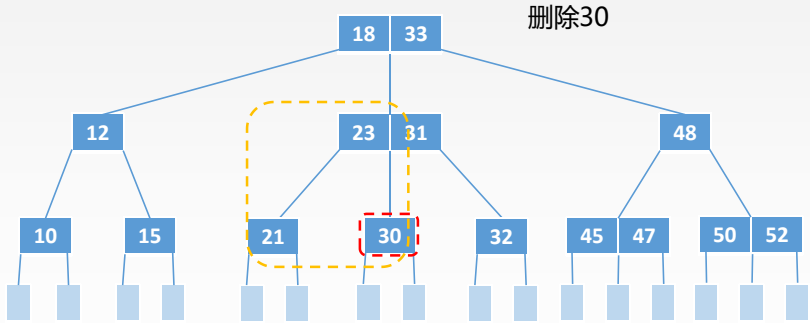
若被删除关键字所在结点关键字总数 $= \lceil m/2 \rceil - 1$ ，且与此结点邻近的兄弟结点的关键字个数 $> \lceil m/2 \rceil$ ，则需要从兄弟结点借一个关键字，此过程需要调整该结点、双亲结点和兄弟结点的关键字

3) 兄弟不够借

若被删除关键字所在结点关键字总数 $= \lceil m/2 \rceil - 1$ ，且与此结点邻近的兄弟结点的关键字个数 $= \lceil m/2 \rceil - 1$ ，则删除关键字，并与一个不够借的兄弟结点和双亲结点中两兄弟子树中间的关键字合并。
合并后若双亲结点因减少一个结点导致不符合定义，则继续执行2、3步骤。

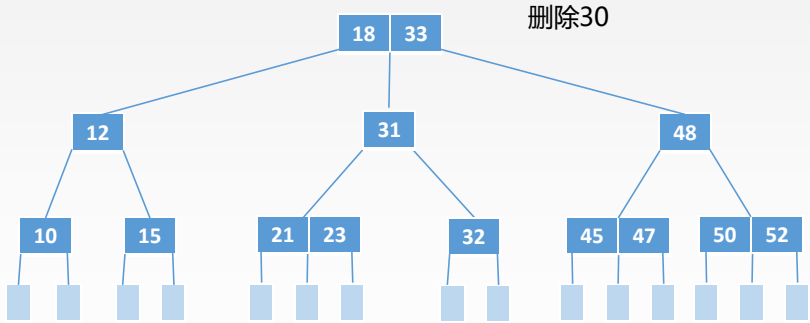
王道考研/CSKAOYAN.COM

B-树



王道考研/CSKAOYAN.COM

B-树



王道考研/CSKAOYAN.COM

B-树

删除

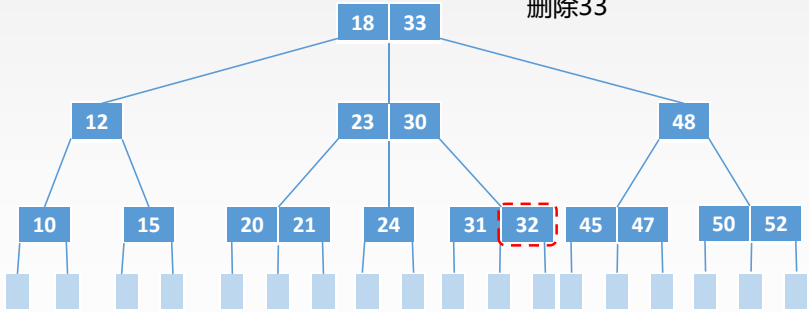
- 1) 若小于k的子树中关键字个数 $> \lceil m/2 \rceil - 1$, 则找出k的前驱值k', 并用k' 来取代k, 再递归地删除k'即可。

王道考研/CSKAOYAN.COM

B-树

删除

删除33

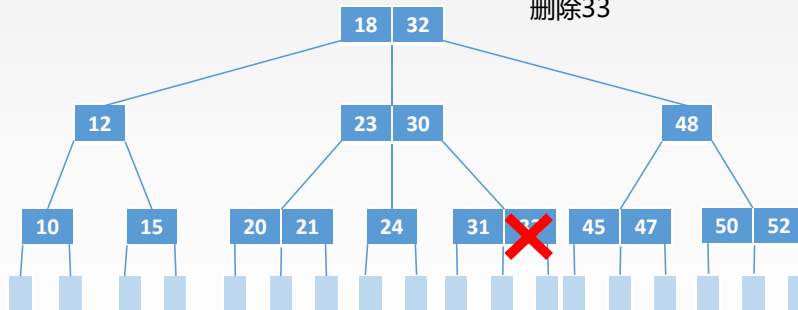


王道考研/CSKAOYAN.COM

B-树

删除

删除33



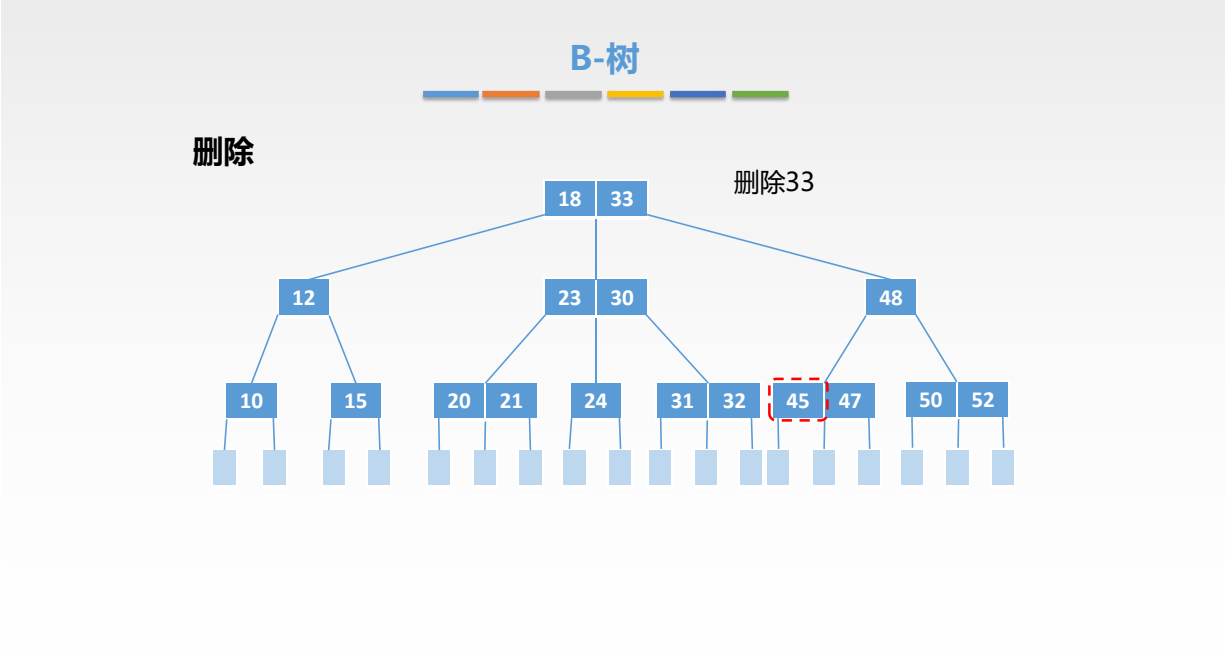
王道考研/CSKAOYAN.COM

B-树

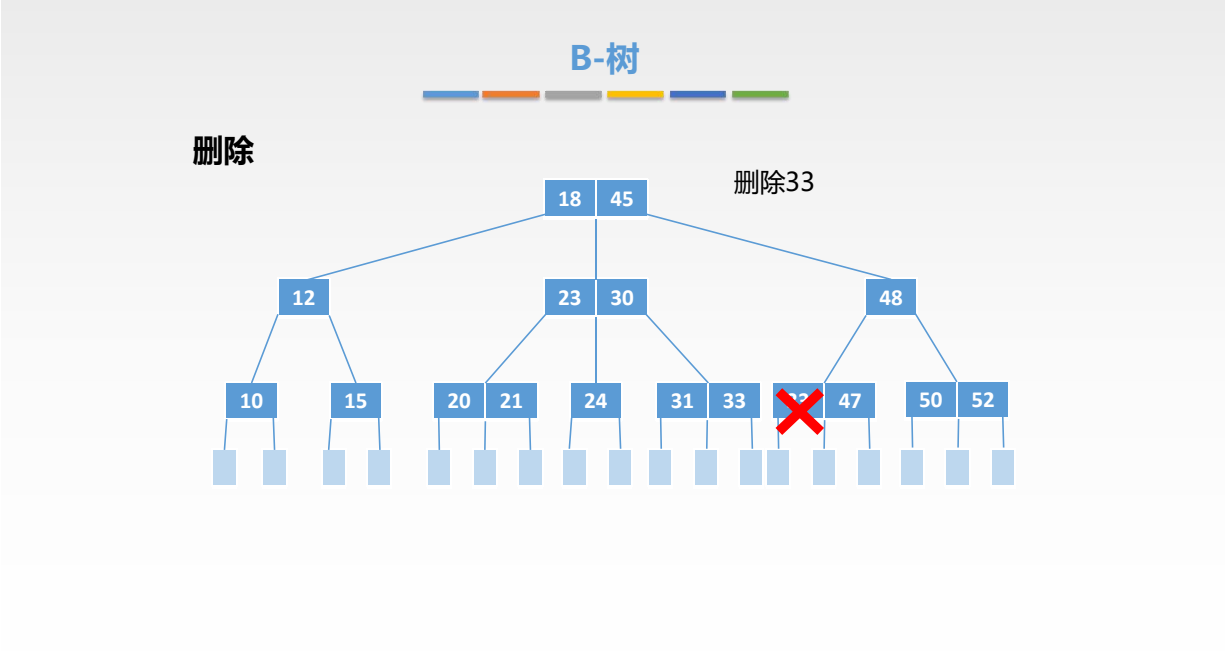
删除

- 1) 若小于k的子树中关键字个数 $> \lceil m/2 \rceil - 1$, 则找出k的前驱值k', 并用k' 来取代k, 再递归地删除k'即可。
- 2) 若大于k的子树中关键字个数 $> \lceil m/2 \rceil - 1$, 则找出k的后继值k', 并用k' 来取代k, 再递归地删除k'即可。

王道考研/CSKAOYAN.COM



王道考研/CSKAOYAN.COM



王道考研/CSKAOYAN.COM

B-树

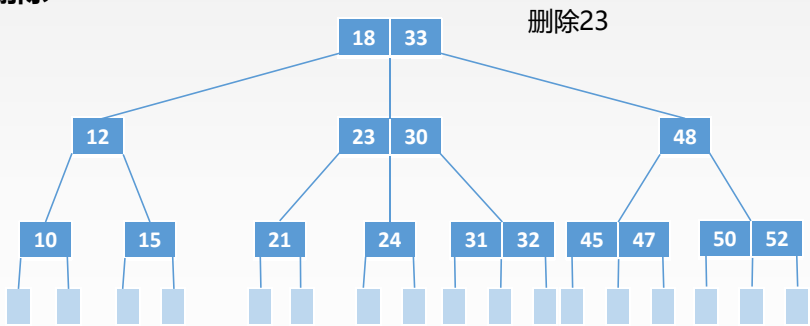
删除

- 1) 若小于k的子树中关键字个数 $>\lceil m/2 \rceil - 1$,则找出k的前驱值k', 并用k' 来取代k, 再递归地删除k'即可。
- 2) 若大于k的子树中关键字个数 $>\lceil m/2 \rceil - 1$,则找出k的后继值k', 并用k' 来取代k, 再递归地删除k'即可。
- 3) 若前后两子树关键字个数均为 $\lceil m/2 \rceil - 1$,则直接两个子结点合并, 然后删除k即可。

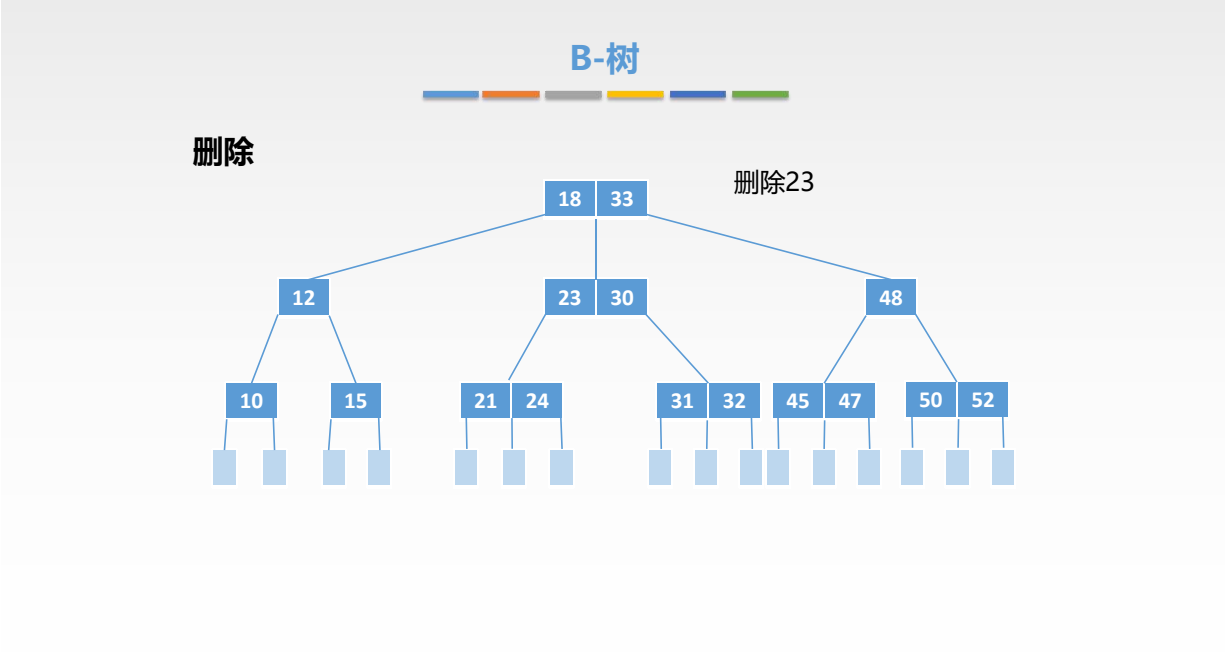
王道考研/CSKAOYAN.COM

B-树

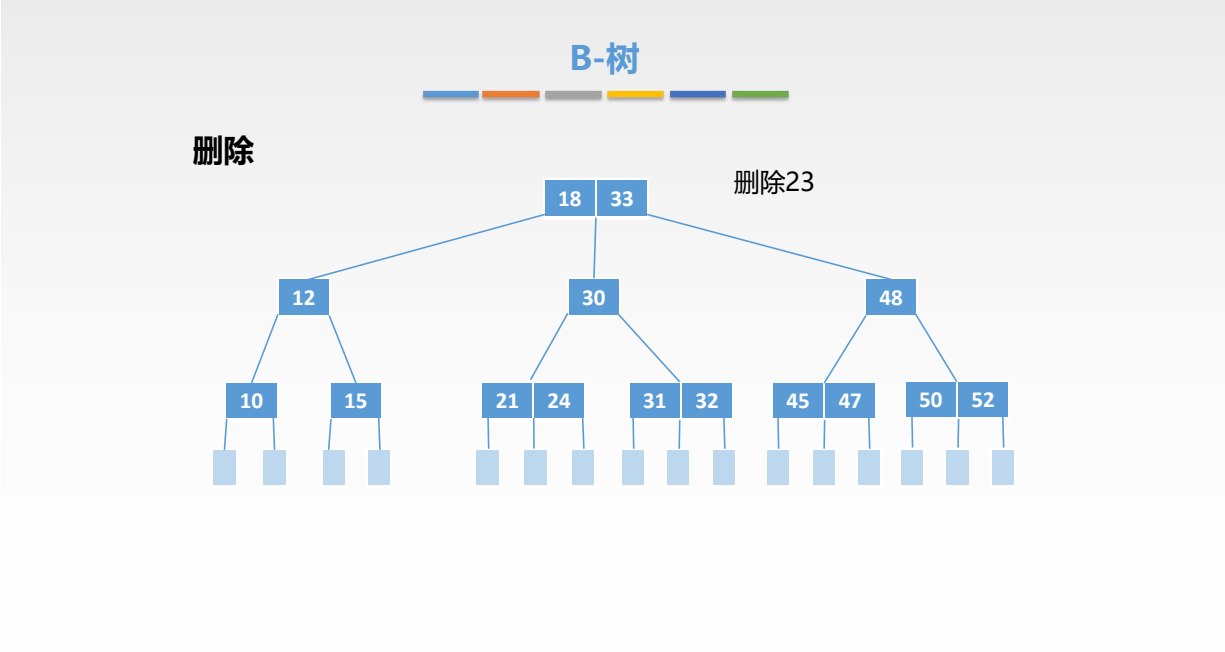
删除



王道考研/CSKAOYAN.COM



王道考研/CSKAOYAN.COM



王道考研/CSKAOYAN.COM

本节内容

查找

B+树

王道考研/CSKAOYAN.COM

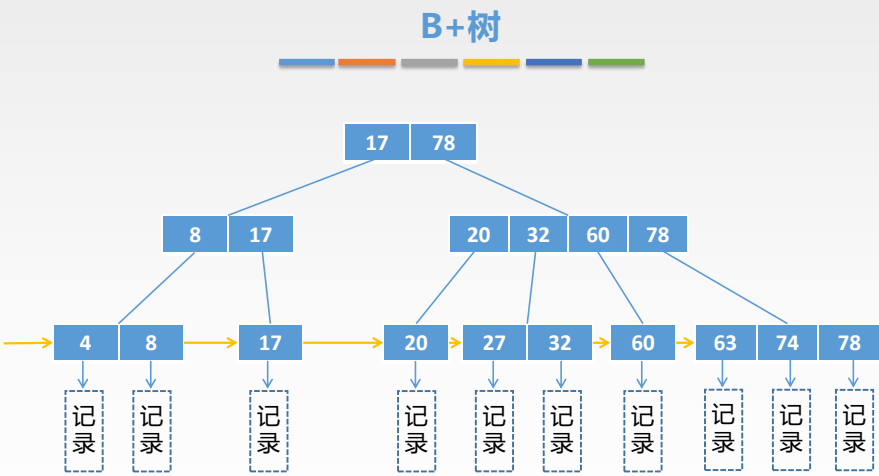
B+树

B+树

一棵m阶B+树满足如下特性：

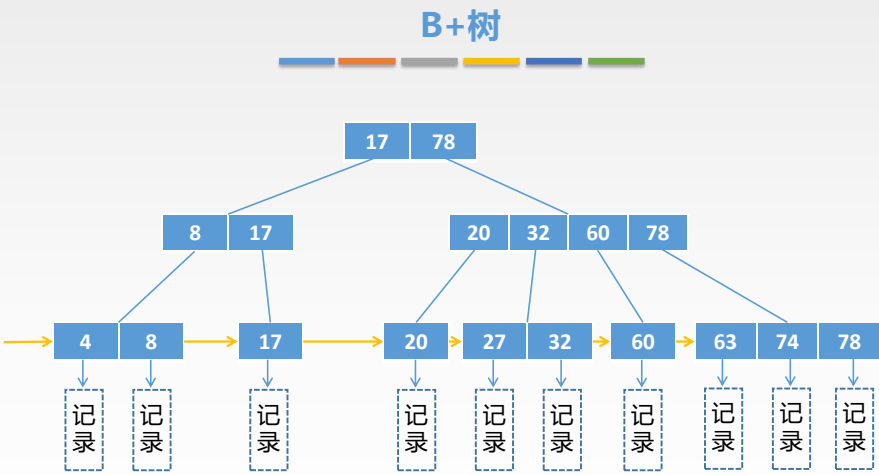
- 1) 每个分支结点最多有m棵子树(子结点)
- 2) 若根结点不是终端结点，则至少有两棵子树
- 3) 除根结点外的所有非叶结点至少有 $\lceil m/2 \rceil$ 棵子树，子树和关键字个数相等
- 4) 所有叶结点包含全部关键字及指向相应记录的指针，叶结点中将关键字按大小顺序排列，并且相邻结点按大小顺序连接起来
- 5) 所有分支结点(可视为索引的索引)中仅包含他的各个子结点（下一级索引块）中关键字的最大值及指向其子结点的指针

王道考研/CSKAOYAN.COM



1) 每个分支结点最多有m棵子树(子结点)

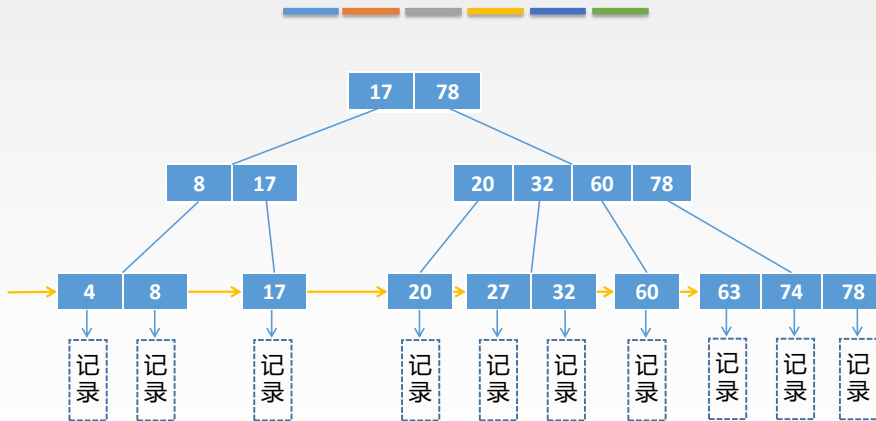
王道考研/CSKAOYAN.COM



2) 若根结点不是终端结点，则至少有两棵子树

王道考研/CSKAOYAN.COM

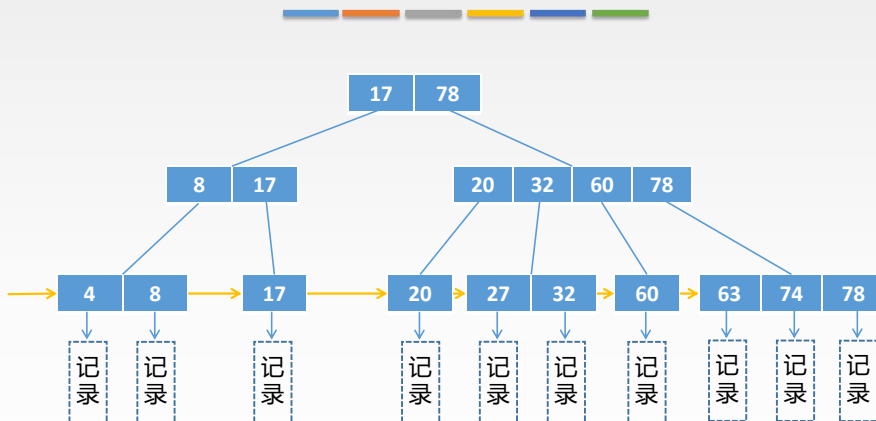
B+树



3) 除根结点外的所有非叶结点至少有 $\lceil m/2 \rceil$ 棵子树，子树和关键字个数相等

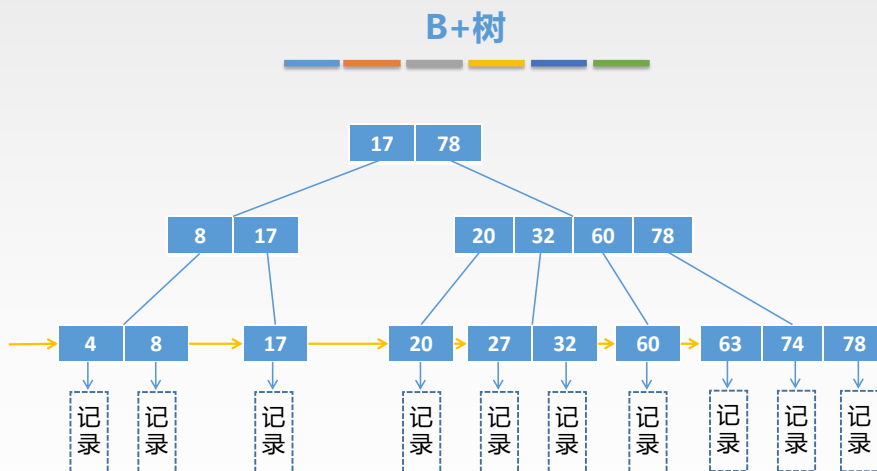
王道考研/CSKAOYAN.COM

B+树



4) 所有叶结点包含全部关键字及指向相应记录的指针，叶结点中将关键字按大小顺序排列，并且相邻结点按大小顺序连接起来

王道考研/CSKAOYAN.COM



5) 所有分支结点(可视为索引的索引)中仅包含他的各个子结点 (下一级索引块) 中关键字的最大值及指向其子结点的指针

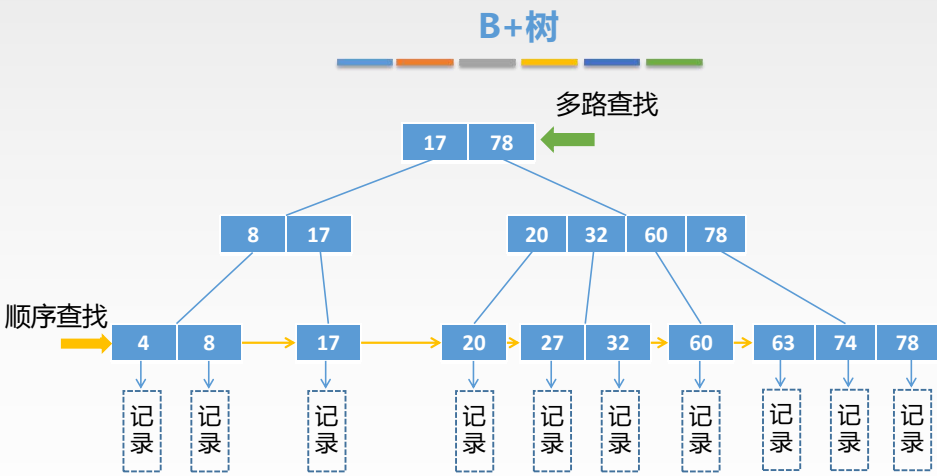
王道考研/CSKAOYAN.COM

B+树

B+树 vs B树

- 1) 在B+树中, 具有n个关键字的结点值含有n棵子树, 即每个关键字对应一棵子树;
在B树中, 具有n个关键字的结点含有n+1棵子树
- 2) 在B+树中, 叶结点包含信息, 所有非叶结点仅起索引作用, 非叶结点中的每个索引项只含有对应子树的最大关键字和指向该子树关键字的指针, 不含有该关键字对应记录的存储地址
- 3) 在B+树中, 叶结点包含全部关键字, 即在非叶结点中出现的关键字也会出现在叶结点中
在B树中, 叶结点包含的关键字和其他结点包含的关键字是不重复的

王道考研/CSKAOYAN.COM



★ 在B+树中查找时，无论查找成功还是失败一定是查找到叶结点当中的值为止

王道考研/CSKAOYAN.COM

本节内容

查找

散列表
基本概念

王道考研/CSKAOYAN.COM

散列表



王道考研/CSKAOYAN.COM

散列表

散列函数 一个把查找表中的关键字映射成该关键字对应的地址的函数。

| | | |
|------|-----|-----------------|
| Addr | key | Hash(key)=Addr |
| 0 | 6 | Hash(key)=key%3 |
| 1 | 13 | {6, 13, 26 } |
| 2 | 26 | |

散列表 根据关键字而直接进行访问的数据结构。他建立了关键字与存储地址之间的一种直接映射关系。

冲突 散列函数可能会把多个不同的关键字映射到同一地址下的情况。

王道考研/CSKAOYAN.COM

本节内容

查找

散列表
散列函数&
冲突处理

王道考研/CSKAOYAN.COM

散列表

散列函数 一个把查找表中的关键字映射成该关键字对应的地址的函数。

$$\text{Hash}(\text{key}) = \text{Addr}$$

要求：

- 1) 散列函数的定义域必须包含全部需要存储的关键字，而值域的范围则依赖于散列表的大小或地址范围。
- 2) 散列函数计算出来的地址应该能等概率、均匀分布在整个地址空间中，从而减少冲突的发生。
- 3) 散列函数应尽量简单，能够在较短时间内计算出任一关键字对应的散列地址。

王道考研/CSKAOYAN.COM

散列表

直接定址法 直接取关键字的某个线性函数值为散列地址。

$$\text{Hash}(\text{key}) = a * \text{key} + b,$$

其中a, b为常数

方法简单，不会产生冲突，若关键字分布不连续，则会浪费空间。

| | |
|---|----|
| 0 | -1 |
| 1 | |
| 2 | |
| 3 | 2 |
| 4 | |
| 5 | 4 |

$$\text{Hash}(\text{key}) = \text{key} + 1,$$

{-1, 2, 4}

王道考研/CSKAOYAN.COM

散列表

除留取余法

$$\text{Hash}(\text{key}) = \text{key} \% p,$$

假定散列表表长为m，取一个不大于m但最接近或等于m的质数p

选好p是关键，可以减少冲突的可能

| | |
|---|----|
| 0 | 30 |
| 1 | 16 |
| 2 | 12 |
| 3 | 23 |
| 4 | 4 |
| 5 | |

$$\text{Hash}(\text{key}) = \text{key} \% 5,$$

{16, 23, 30, 4, 12}

王道考研/CSKAOYAN.COM

散列表

数字分析法

| | | |
|------|------|------|
| 1011 | 1111 | 1100 |
| 1011 | 1111 | 1001 |
| 1011 | 1111 | 0001 |
| 1011 | 1111 | 0011 |
| 0000 | 0000 | 1111 |
| 0000 | 0000 | 1101 |
| 0000 | 0000 | 0000 |
| 0000 | 0000 | 1000 |

适用于关键字已知的集合，若更换关键字则需要重新构造散列函数。

王道考研/CSKAOYAN.COM

散列表

平方取中法 这种方法取关键字的平方值的中间几位作为散列地址

521
271441

适用于关键字的每位取值不均匀或均小于散列地址所需要的位数。

折叠法 将关键字分割成位数相同的几部分，然后取这几部分的叠加和作为散列地址

5211252
 $521+125+2=648$

适用于关键字的位数多，而且关键字中的每位上数字分布大致均匀

王道考研/CSKAOYAN.COM

散列表

| | |
|---|----|
| 0 | 30 |
| 1 | 16 |
| 2 | 12 |
| 3 | 23 |
| 4 | 4 |
| 5 | |

Hash(key)=key % 5,
{16, 23, 30, 4, 12}

24

? 冲突不可能绝对避免，那如何处理冲突

为产生冲突的关键字寻找下一个“空”的Hash地址

开放定址法

拉链法

王道考研/CSKAOYAN.COM

散列表

开放定址法 是指可存放新表项的空闲地址既向它的同义词表项开放，又向它的非同义词表项开放。

$H_i = (H(key) + d_i) \% m, i = 0, 1, 2, \dots, k (k \leq m - 1); m$ 为散列表表长, d_i 为增量序列

| | |
|----------------|-------|
| H(key) | |
| | |
| H ₀ | |
| | |
| H ₁ | key |

d₀

d₁

? 如何计算增量序列

- 线性探查法
- 平方探测法
- 再散列法
- 伪随机序列法

王道考研/CSKAOYAN.COM

散列表

线性探测法 即 $d_i = 0, 1, 2, 3, \dots, m-1$

| | 散列表 |
|---|-----|
| 0 | |
| 1 | 1 |
| 2 | 10 |
| 3 | 21 |
| 4 | |
| 5 | 14 |
| 6 | 15 |
| 7 | 23 |
| 8 | |

Hash(key)=key % 9,
{1, 10, 21, 15, 14, 23}

王道考研/CSKAOYAN.COM

散列表

线性探测法 即 $d_i = 0, 1, 2, 3, \dots, m-1$

| | 散列表 |
|---|-----|
| 0 | |
| 1 | 1 |
| 2 | 10 |
| 3 | 11 |
| 4 | 12 |
| 5 | |
| 6 | |
| 7 | |
| 8 | |

Hash(key)=key % 9,
{1, 10, 11, 12}

! 堆积现象

堆积现象会大大降低查找效率

王道考研/CSKAOYAN.COM

散列表

平方探测法 即 $d_i = 0^2, 1^2, -1^2, 2^2, -2^2, \dots, k^2, -k^2$, 其中 $k \leq m/2$

避免堆积问题，缺点是不能探测到散列表上的所有单元
(至少可以探测到一般单元)

再散列法 即 $d_i = i * \text{Hash2}(\text{key})$

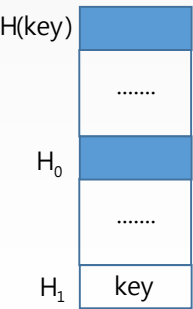
伪随机法 即 $d_i = \text{伪随机序列}$

王道考研/CSKAOYAN.COM

散列表

开放定址法 是指可存放新表项的空闲地址既向它的同义词表项开放，又向它的非同义词表项开放。

$H_i = (H(\text{key}) + d_i) \% m, i = 0, 1, 2, \dots, k (k \leq m-1); m$ 为散列表表长, d_i 为增量序列

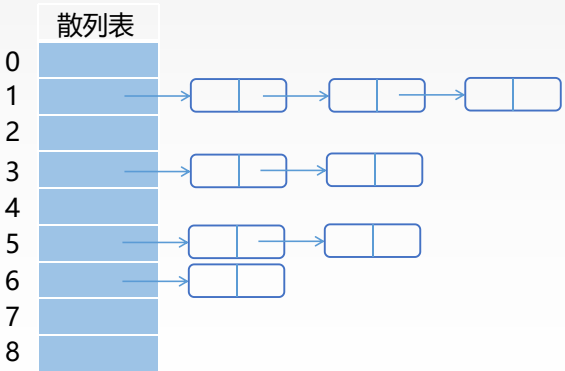


★ 在开放定址法中不能随便删除某个元素

王道考研/CSKAOYAN.COM

散列表

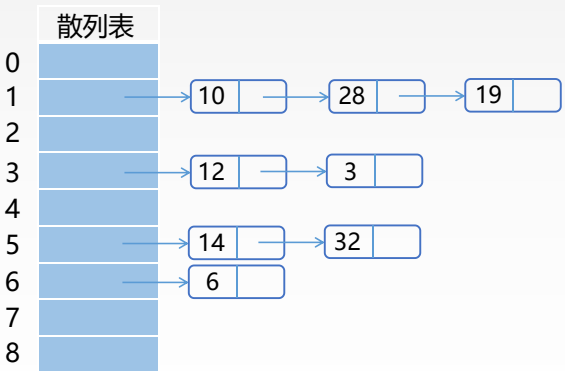
拉链法 是指把所有同义词存放在一个线性链表中，这个线性链表由地址唯一标识，即散列表中每个单元存放该链表头指针。



王道考研/CSKAOYAN.COM

散列表

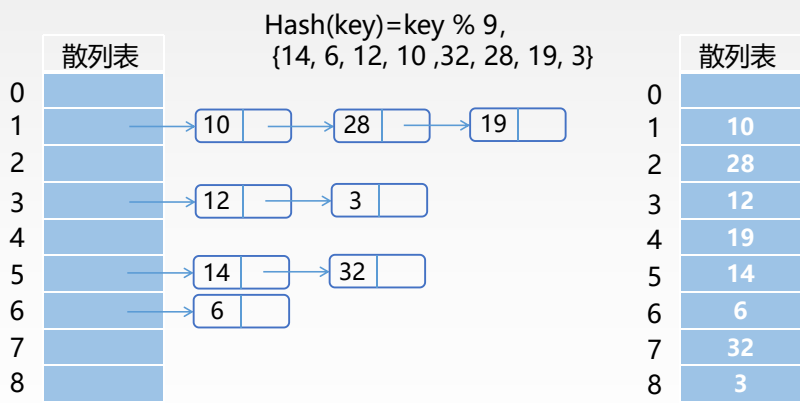
$\text{Hash}(\text{key}) = \text{key} \% 9$,
{14, 6, 12, 10, 32, 28, 19, 3}



★ 拉链法适用于经常进行插入和删除的情况

王道考研/CSKAOYAN.COM

散列表



- 初始化: Addr=Hash(key);
- ① 检测查找表中地址为Addr的位置上是否有记录, 若无记录, 则返回查找失败; 若有记录, 则比较它与key值, 若相等则返回成功, 否则执行步骤②
 - ② 用给定的处理冲突方法计算 “下一散列地址”, 把Addr置为此地址, 转入步骤①

王道考研/CSKAOYAN.COM

散列表

查找效率

散列函数、处理冲突的方法和填装因子

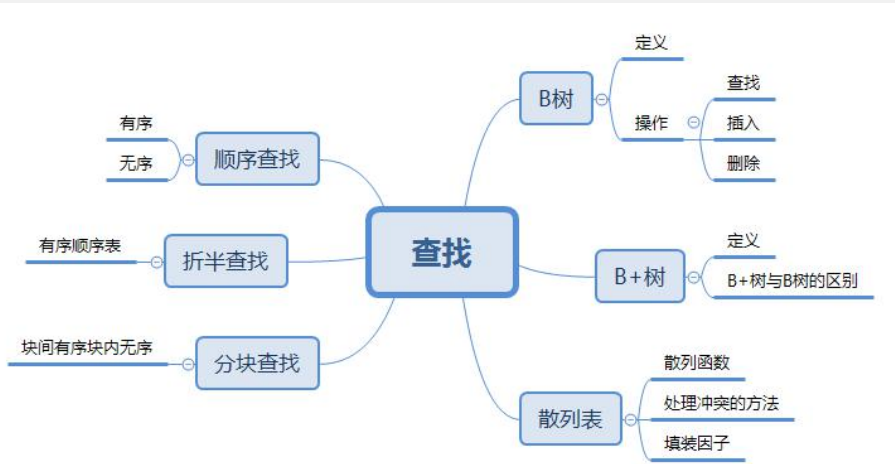
填装因子 一般记为 α , 表示表的装满程度

$$\alpha = \frac{\text{表中记录数}n}{\text{散列表长度}m}$$

★ 散列表的平均查找长度依赖于散列表的填装因子

王道考研/CSKAOYAN.COM

散列表



王道考研/CSKAOYAN.COM

本节内容

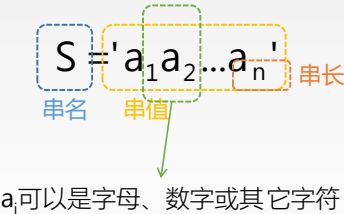
查找

串
基本概念

王道考研/CSKAOYAN.COM

串的定义

串 (String) 是由零个或多个字符组成的有限序列。一般记为



1 2 3 4 5 6 7 8 9 10 11 12
S='Hello World!'

★ 若两个串长度相等且每个对应位置的字符都相等时，称这两个串是相等的

王道考研/CSKAOYAN.COM

串的定义

子串 串中任意个连续的字符组成的子序列称为该串的**子串**，包含子串的串为**主串**

通常称字符在序列中的序号为该字符在串中的位置。
子串在主串中的位置是以子串的第一个字符在主串的位置来表示的

1 2 3 4 5 6 7 8 9 10 11 12
S='Hello World! '

S1='Hello'

位置为1

S2='World'

位置为7

王道考研/CSKAOYAN.COM

串的存储结构

1、定长顺序存储

```
#define MAXLEN 255
typedef struct{
    char ch[MAXLEN];
    int length;
}SString;
```

'Hello World!'

H e l l o sp W o r l d ! \0

2、堆分配存储表示

```
typedef struct{
    char *ch;
    int length;
}HString;
```

malloc()/free()

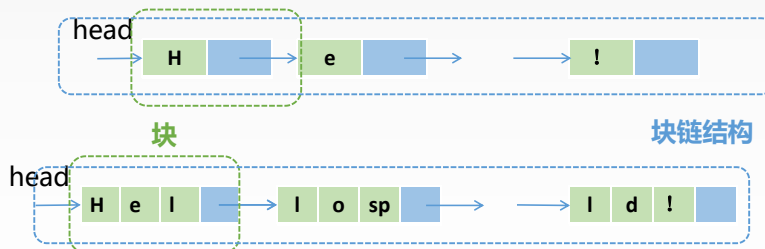
王道考研/CSKAOYAN.COM

串的存储结构

3、块链存储表示

利用链表来存放字符串

'Hello World!'



王道考研/CSKAOYAN.COM

串的基本操作

串的基本操作

StrAssign(&T, chars): 赋值操作。把串T赋值为chars。

StrCompare(S, T): 比较操作。若 $S > T$ ，则返回值 > 0 ；若 $S = T$ ，则返回值 $= 0$ ；若 $S < T$ 返回值 < 0 。

StrLength(S): 求串长。返回串S的元素个数。

SubString(&Sub, S, pos, len): 求子串。用Sub返回串S的第pos个字符起长度为len的子串

Concat(&T, S1, S2): 串联接。用T返回由S1和S2联接而成的新串

最小操作子集

王道考研/CSKAOYAN.COM

串的基本操作

串的基本操作

Index(S, T, pos): 定位操作。若主串S中存在与串T值相同的子串，则返回它在主串S中第pos个字符后第一次出现的位置；否则函数值为0。

StrCopy(&T, S): 复制操作。由串S复制得到串T。

StrEmpty(S): 判空操作。若S为空串，返回TRUE；若非空，返回FALSE。

Replace(&S, T, V): 替换子串作。用V替换主串S中出现的所有与T相等的非重叠的子串。

StrInsert(&S, pos, T): 插入操作。在串S的第pos个字符之前插入串T。

StrDelete(&S, pos, len): 删除子串。从串S中删除第pos个字符起长度为len的子串。

ClearString(&S): 清空操作。将S清为空串。

DestroyString(&S): 销毁串。将串S销毁。

王道考研/CSKAOYAN.COM

串的基本操作

Index(S, T, pos): 定位操作。若主串S中存在与串T值相同的子串，则返回它在主串S中第pos个字符后第一次出现的位置；否则函数值为0。

```
int Index(String S, String T, int pos){
    if(pos>0){
        int n = StrLength(S);
        int m = StrLength(T);
        int i = pos;
        String sub=NULL;
        while(i<=n-m+1){
            SubString(sub,S,i,m);
            if(StrCompare(sub, T)!=0)
                i++;
            else
                return i;
        }
    }
    return 0;
}
```

主串S: absaaasdcbvas
串T: asd

int position=Index(S, T, 2);

王道考研/CSKAOYAN.COM

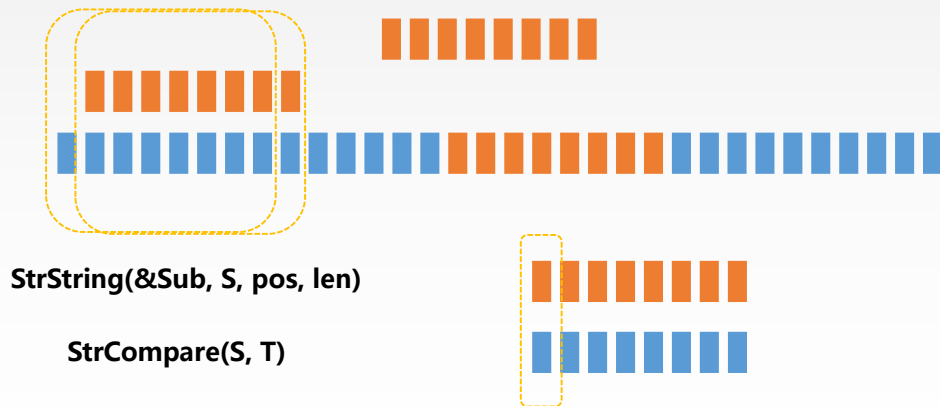
本节内容

查找

串
模式匹配

王道考研/CSKAOYAN.COM

串的模式匹配



王道考研/CSKAOYAN.COM

串的模式匹配

```
int Index(SString S, SString T, int pos){
    int i=pos, j=1;
    while(i<=S.length && j<=T.length){
        if(S.ch[i]==T.ch[j]){
            i++;
            j++;
        }
        else{
            i=i-j+2;
            j=1;
        }
        if(j>T.length)
            return i-T.length;
        else
            return 0;
    }
}
```

O(nm)

王道考研/CSKAOYAN.COM

串的模式匹配

模式 `baac`

主串 `acbbabaaacbcacccabca`

| | | | |
|-----|--------------------------------------|-----|--------------------------------------|
| 第一趟 | 主串 <code>acbbabaaacbcacccabca</code> | 第五趟 | 主串 <code>acbbabaaacbcacccabca</code> |
| | <code>baac</code> | | <code>baac</code> |
| 第二趟 | 主串 <code>acbbabaaacbcacccabca</code> | 第六趟 | 主串 <code>acbbabaaacbcacccabca</code> |
| | <code>baac</code> | | <code>baac</code> |
| 第三趟 | 主串 <code>acbbabaaacbcacccabca</code> | | |
| | <code>baac</code> | | |
| 第四趟 | 主串 <code>acbbabaaacbcacccabca</code> | | |
| | <code>baac</code> | | |

王道考研/CSKAOYAN.COM

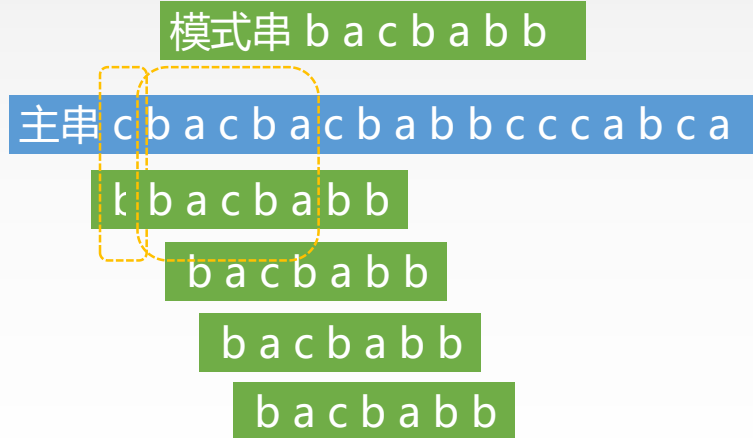
本节内容

查找

串
KMP算法

王道考研/CSKAOYAN.COM

KMP算法



王道考研/CSKAOYAN.COM

KMP算法

前缀 除最后一个字符外，字符串的所有头部子串

后缀 除第一个字符外，字符串的所有尾部子串

部分匹配值 字符串前缀和后缀最长相等前后缀的长度

'a b a b a'

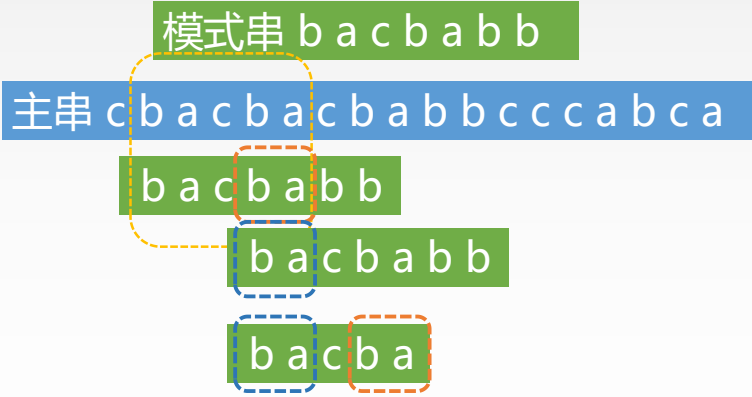
前缀 {a, ab, aba, abab}

后缀 {a, ba, aba, baba}

部分匹配值 3

王道考研/CSKAOYAN.COM

KMP算法



$$\text{Move} = (j-1) - \text{next}[j-1]$$

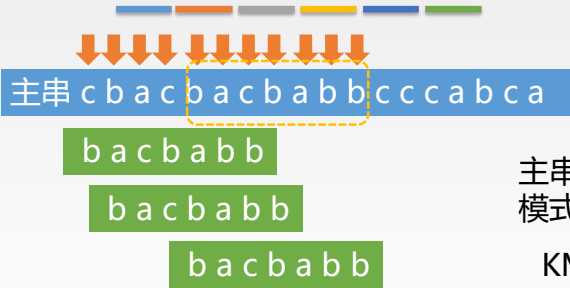
王道考研/CSKAOYAN.COM

KMP算法

| 模式串 b a c b a b b | next[] |
|-------------------|-----------|
| b | next[1]=0 |
| b a | next[2]=0 |
| b a c | next[3]=0 |
| b a c b | next[4]=1 |
| b a c b a | next[5]=2 |
| b a c b a b | next[6]=1 |
| b a c b a b b | next[7]=1 |

王道考研/CSKAOYAN.COM

KMP算法



主串标记 i 不回退
模式串标记 j 回退

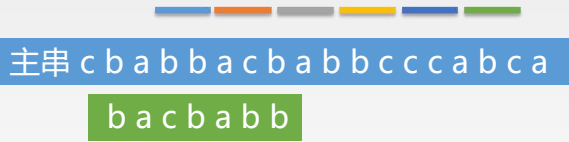
KMP: $O(m+n)$

$Move = (j-1) - next[j-1]$

| 编号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|------|---|---|---|---|---|---|---|
| S | b | a | c | b | a | b | b |
| next | 0 | 0 | 0 | 1 | 2 | 1 | 1 |

王道考研/CSKAOYAN.COM

KMP算法



| 编号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|------|---|---|---|---|---|---|---|
| S | b | a | c | b | a | b | b |
| next | 0 | 0 | 0 | 1 | 2 | 1 | 1 |

- 右移: 1) 第一个元素右移后变为-1
2) 最后一个元素右移后直接舍去

| 编号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|------|----|---|---|---|---|---|---|
| S | b | a | c | b | a | b | b |
| next | -1 | 0 | 0 | 0 | 1 | 2 | 1 |

$Move = (j-1) - next[j]$

王道考研/CSKAOYAN.COM

KMP算法

主串 c b a b b a c b a b b c c c a b c a
子串 b a c b a b b

| 编号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|------|----|---|---|---|---|---|---|
| S | b | a | c | b | a | b | b |
| next | -1 | 0 | 0 | 0 | 1 | 2 | 1 |

$$\text{Move} = (j-1) - \text{next}[j]$$
$$j = j - \text{Move} = j - ((j-1) - \text{next}[j]) = \text{next}[j] + 1$$

| 编号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|------|---|---|---|---|---|---|---|
| S | b | a | c | b | a | b | b |
| next | 0 | 1 | 1 | 1 | 2 | 3 | 2 |

$$j = \text{next}[j]$$

王道考研/CSKAOYAN.COM

KMP算法

计算机中求解next更高效的算法

初始化移动标记 $i=1$ ，当前部分匹配值 $j=0$ ， $\text{next}[1]=0$

当 $\text{next}[j]$ 已经求得, 则 $\text{next}[j+1]$ 为:

- 1) 若子串中 $t_j = t_i$, $\text{next}[i+1]=j+1$
- 2) 若子串中 $t_j \neq t_i$, $j=\text{next}[j]$, 继续进行1) 比较



王道考研/CSKAOYAN.COM

KMP算法

```
void get_next(String T,int next[]){
    int i=1, j=0;
    next[1]=0;
    while(i<T.length){
        if(j==0||T.ch[i]==T.ch[j]){
            ++i;++j;next[i]=j;
        }else{
            j=next[j];
        }
    }
}
```

b a c b a b b

| 编号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|------|---|---|---|---|---|---|---|
| S | b | a | c | b | a | b | b |
| next | 0 | 1 | 1 | 1 | 2 | 3 | 2 |

王道考研/CSKAOYAN.COM

KMP算法

```
int Index_KMP(String S, String T, int next[], int pos){
    int i=pos, j=1;
    while(i<=S.length && j<=T.length){
        if(j==0||S.ch[i]==T.ch[j]){
            i++;
            j++;
        }
        else{
            j=next[j];
        }
    }
    if(j>T.length)
        return i-T.length;
    else
        return 0;
}
```

王道考研/CSKAOYAN.COM