数据结构与算法

第11章　索引技术

主讲：赵海燕
北京大学信息科学技术学院
“数据结构与算法”教学组

国家精品课“数据结构与算法”
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/

张铭，王腾蛟，赵海燕
高等教育出版社，2008. 6, “十一五”国家级规划教材

主要内容

- 基本概念
- 线性索引
- 静态索引
- 倒排索引
- 动态索引
  - 动态、静态索引性能的比较
- 位索引技术
- 红黑树

索引的目标

- 大数据量的检索
  - 支持大文件
  - 支持多种检索码
  - 支持有效的插入、删除
  - 支持范围查询

文件检索涉及的概念

- 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

- entry-sequenced file
  - 按照记录进入系统的顺序存储记录
  - 结构相当于磁盘中**未排序的线性表**
  - 难以支持高效率的检索

主码（primary key）

- 一个文件（或数据库）中每个记录的唯一标识
  - 例如，每个人的身份证号码或者学生的学号
    - 数据库表的主码定义

```
create table Student (
        SNO  char[10],
        SNAME char[20] not null,
        AGE int,
        DNO char[10],
        ……,
        primary key (SNO),
        foreign key (DNO) references DEPT(DNO));
```

  - 若只有主码，不便于各种灵活的检索

辅码（secondary key）

- 常用于检索，也称次码
- 一般不唯一，即多个记录可能具有相同的辅码，辅码索引把一个辅码值与具有这个辅码值的每条记录的主码值关联起来
  - 例，数据库中建立索引

    create unique cluster index Sname on Student(SNAME)

    - 表上可建立多个索引。索引在提高查询效率的同时也耗费空间，且降低插入、删除、更新等的效率
    - 索引的使用一般由DBMS根据检索语句来决定，不允许用户显式使用
- 大多数检索都是利用**辅码索引**来完成的

索引技术

- 组织大型数据库的一种重要技术
  - 高效率的检索
  - 支持插入、更新、删除
  - 在大型数据库一般采取树型索引结构（多级索引）

- Indexing：把关键码（辅码）与其对应的数据记录的位置**相关联**的过程

索引文件

- 用于记录 **关键码** 与其 **对应的数据记录的位置** 的文件组织结构
- 基本单位：记录
  - （关键码，指针）对，将每个关键码和一个指针关联
  - 指针指向数据库文件（也称为“主文件”）中的完整记录
- 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字段
  - 不需重新排列主文件
- 通过索引文件可高效访问记录中关键码值

索引的分类

- 根据索引量
  - 稠密索引
  - 稀疏索引
- 根据结构
  - 线性索引
  - 树索引
- 根据变化性
  - 静态索引
  - 动态索引

索引的分类

- 根据索引量
  - 稠密索引
  - 稀疏索引
- 根据结构
  - 线性索引
  - 树索引
- 根据变化性
  - 静态索引
  - 动态索引

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
  - 记录按照关键码顺序存放
  - 可把记录分成多个组（块）
    - 索引指针指向的这一组记录在磁盘中的起始位置

索引的分类

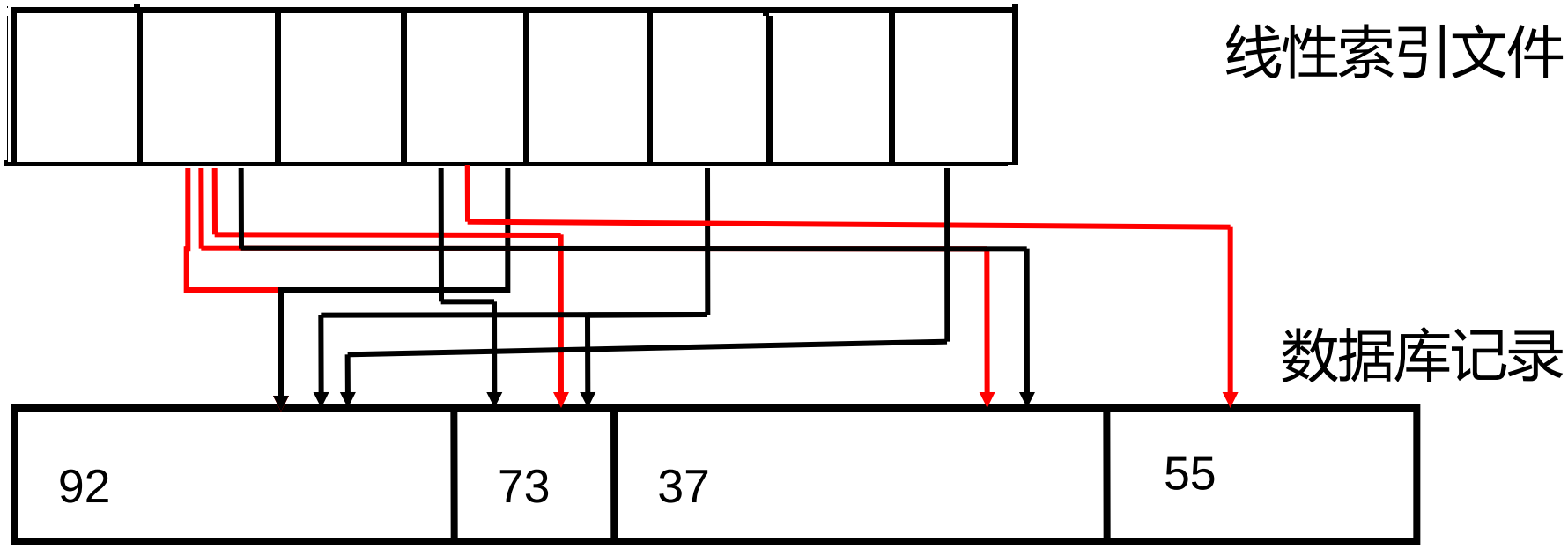
- 根据索引量
  - 稠密索引
  - 稀疏索引
- 根据结构
  - 线性索引
  - 树索引
- 根据变化性
  - 静态索引
  - 动态索引

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

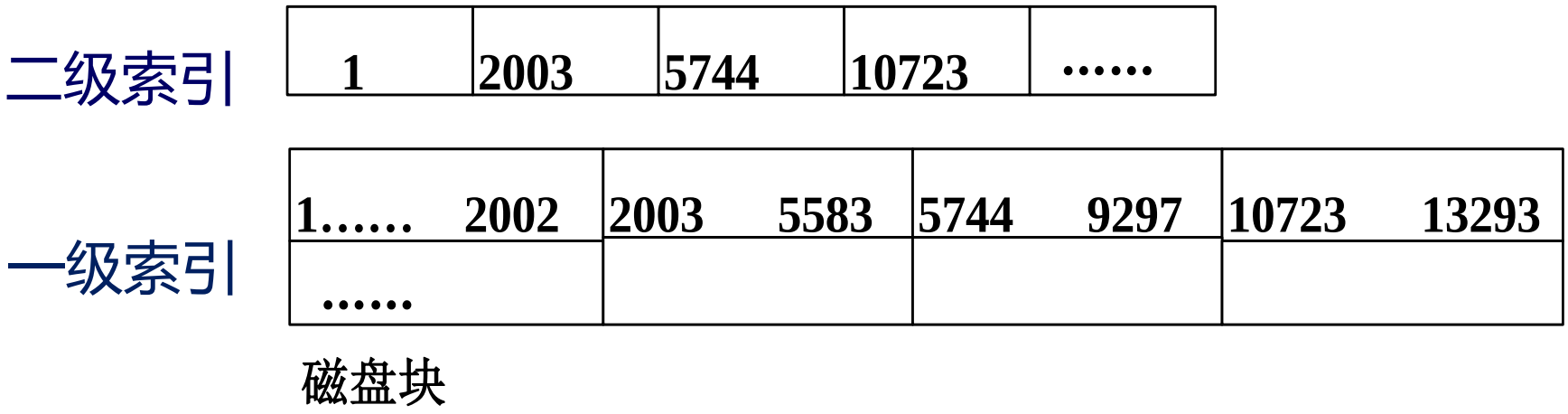
- 便于对变长的数据库记录的访问
- 可对数据进行高效检索
  - 二分检索
- 便于顺序处理
  - 比较操作
  - 批处理的操作
- 节省空间（相对其它索引结构）

线性索引的问题

- 检索效率较低，尤其对于大数据
  - 线性索引过大，一次检索过程可能多次访问磁盘，从而影响检索的效率
  - 使用二级线性索引
- 不易更新
  - 在数据库中插入或删除记录

二级线性索引

- （关键码，指针）
  - 关键码与相应磁盘块中第1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



二级线性索引示例

- 例如，磁盘块大小是 1024 字节，每个索引项(关键码，指针）需要 8 个字节
  - 1024 / 8 = 128
  - 每磁盘块可存储128条索引项
- 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
    - 一级线性索引需占用 79 个磁盘块
  - 二级线性索引文件中只需 79 项索引项
  - 二级线性索引文件可容纳在 1 个磁盘块 (79<128)

检索关键码为2555的记录

1. 二级线性索引文件读入内存
2. 二分法在二级索引中查找关键码的值小于等于2555的最大关键码所在一级索引盘块地址 —— 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

倒排索引（Inverted Index）

- 基于属性的倒排
  - 检索结构中某个或若干个属性满足一定条件的结点，也即非码属性的检索
  - 按照属性的值来查找记录的方法，也即由属性值来确定记录的位置
- 对正文文件的倒排
  - 提供对文本内容的快速检索，以文中的词 (word) 为索引项建立的索引

基于属性的检索

- 例，有关学生的成绩单

  (sNo, sName, dept, age, sex, homeTown, mathScore, …)

  常见的查询：

  - 简单查询： 列出计算机的所有学生记录

    (dept = “CS”)

  - 范围查询： 列出数学成绩在80-90之间的所有学生记录

    (80 ≤ mathScore ≤ 90)

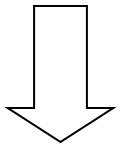
  - 逻辑查询：列出计算机系数学成绩在80-90间的所有学生记录

    (dept = “CS”) && (80 ≤ mathScore ≤ 90)

基于属性的检索

- 最朴素的方法是顺序扫描数据集合（文件）
  - 问题：对大数据集而言，速度太慢，响应时间太长

有无有效的方法?



Yes

基于属性的检索

- 结果为一个集合
  - 数据（包括索引）的组织方式需有所改变
- 基于关键码的检索
  - 因结果唯一，可采用主索引：用主关键码建立的索引（主关键码可唯一标示一个记录）

记录关键码 key	记录地址 addr

基于属性的索引该如何组织？

基于属性的倒排

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

倒排示例：教师数据库

EMP#	NAME	Department	Profession	Specialty	Address
0155	李宇	数学	教授	代数	C105
0421	刘阳	外语	助教	英语	E310
0208	赵亮	物理	助教	力学	C211
0211	张伟	物理	讲师	原子物理	D508
0132	王亮	数学	助教	几何	E220
0119	王卓	数学	讲师	代数	B102
0330	孙丽	计算机	教授	软件	A108
0455	刘珍	外语	讲师	法语	A225
0310	周兵	计算机	讲师	英语	B423
0341	何江	计算机	助教	计算机	F406
……					

倒排示例：教师数据库

Department list	EMP#
数学 物理 计算机 外语	0155, 0132, 0119 0208, 0211 0330, 0310, 0341 0421, 0455
Profession list	**EMP#**
教授 讲师 助教	0155, 0330 0211, 0119, 0455, 0310 0421, 0208, 0132, 0341
Specialty list	**EMP#**
代数 几何 力学 原子物理 软件 英语 法语	0155, 0119 0132 0208 0211 0330, 0341 0421, 0310 0455

倒排索引优缺点

- 优点
  - 能够对基于属性的检索进行较高效率的处理
- 缺点
  - 付出了保存倒排表的存储代价
  - 降低了更新效率

对正文文件的倒排

- 正文索引(Text Indexing)的目标
  - 建立一个数据结构以提供对文本内容的快速检索
- 方法
  - 全文索引(full-text index)
  - 词索引(word index)

全文索引

- 对文中的每一字符建立索引，使查询词不再限于关键词
- 基本思想
  - 将 正文 看作 一个长字符串
  - 索引结构中记录的是子字符串的开始位置
  - 查询 可以针对正文中的 任何子字符串
- 需要更大的索引空间

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，特别是那些较容解析成 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

词索引

- 使用最广泛的倒排文件
- 实质为一个已排序的关键词列表，其中
  - 每个关键词指向一个倒排表(posting list)，包括：
    - 关键词出现的文档集合
    - 关键词在相应文档中的位置

词索引示例

- 正文文件：由6个文档组成，每个文档都是长字符串

文档编号	文本内容
1	Pease porridge hot, pease 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.

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

类似地处理1中的其他词语

同理，处理2中的词语

依次处理所有文档

建立正文倒排文件

1. 对文档集中的所有文件进行分割处理，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
   - 以人工或自动方式从记录中抽取关键词，停用词(Stopword)、抽词干(Stemming)、切词（segmentation）
3. 建立正文倒排表、倒排文件
   - 得到各个关键词的集合，对每一关键词得到其倒排表，再把所有的倒排表存入文件
     - 记录每个倒排表在索引文件中的开始位置及大小（也可记录每个关键词的出现次数）

中文切词 (Chinese Segmentation)

- 我知道你不知道我知道你不知道我知道你不知道
  1. 我知道，你不知道。我知道，你不知道我知道，你不知道
  2. 我知道你，不知道我。知道你不知道我，知道你不知道
  3. 我，知道你不知道我知道。你，不知道我知道你不知道

对关键词的检索

- 分两步：
  1. 在倒排文件中检索关键词
  2. 若找到了关键词，那么获取文件中的对应倒排表，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，以便快速找到关键词
  - 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
  - 某些应用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

思考

- 怎样有效地组织属性倒排索引表?
- 一个关键词如果在同一个文本中多次出现，它在倒排文件中的索引项可否进行合并?

索引的分类

- 根据索引量
  - 稠密索引
  - 稀疏索引
- 根据结构
  - 线性索引
  - 树索引
- 根据变化性
  - 静态索引
  - 动态索引

静态索引

- 索引结构在文件创建、初始装入记录时一次性生成；系统运行过程中，即使有插入、删除记录的操作，索引结构也保持不变
  - 例如，专门为磁盘存取而设计的ISAM（Indexed Sequential Access Method）结构
  - 必要时可通过文件重组来改变索引结构
- 常用的静态索引
  - 线性索引（不易改变）
  - 多分树、ISAM（索引顺序存取方法）

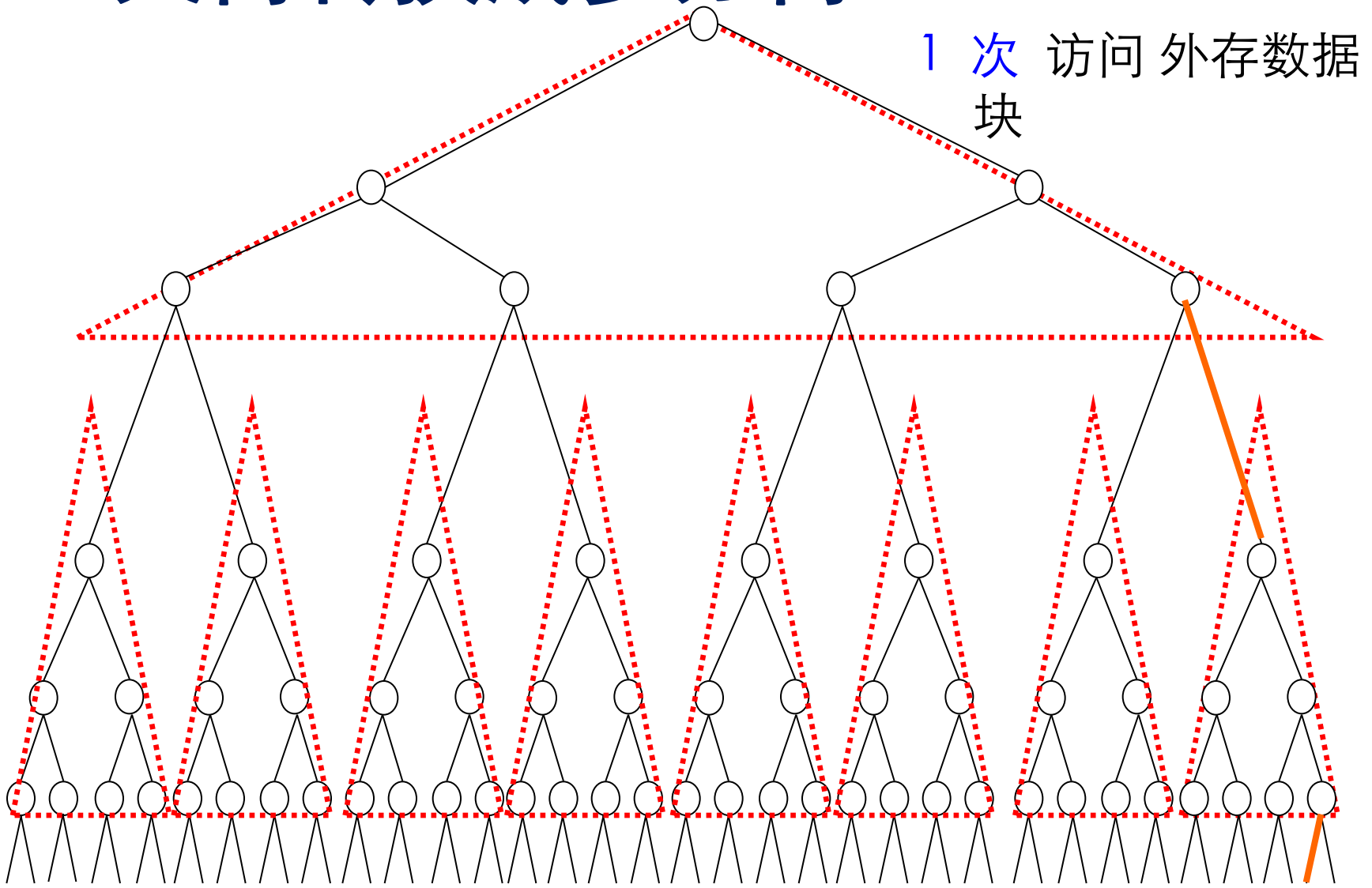
多分树

- 为大幅减少访问外存的次数，组织索引一般采用多分树而非二叉树
- 大大减少访问外存的次数
  - 结点更大
    - 以更少的外存访问次数来完成查找
    - 需要较大的缓冲区
    - 读入一个结点也需较多时间
  - 结点的大小：以放在 1 个磁盘块中为宜

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



多分树

- 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，作为检索判断的依据

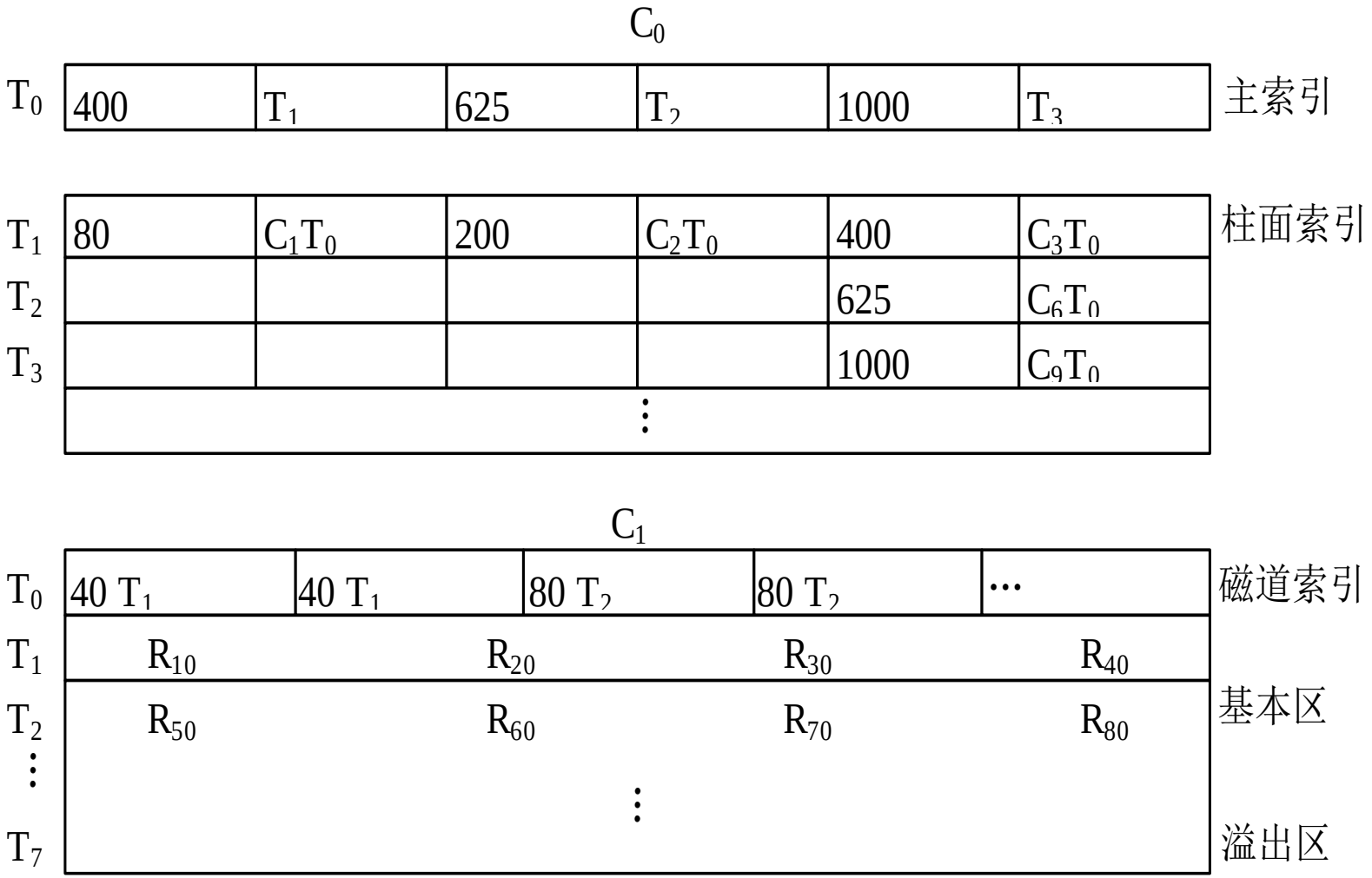
多分树

- 溢出
  - 当新记录要插入的结点已满时，发生称为溢出的现象
- 溢出区
  - 不改变索引的结构（静态索引），把溢出的记录存放到另开辟的**溢出区**
- 记录送入**溢出区**的两种方式
  - 数据基本区的记录和溢出区的记录保持顺序，把数据结点的**最后一个记录**（不一定为新插入的记录）送往溢出区（诸如ISAM）
  - 数据基本区的记录和溢出区的记录不要求保持顺序，把**新插入**的记录送入溢出区

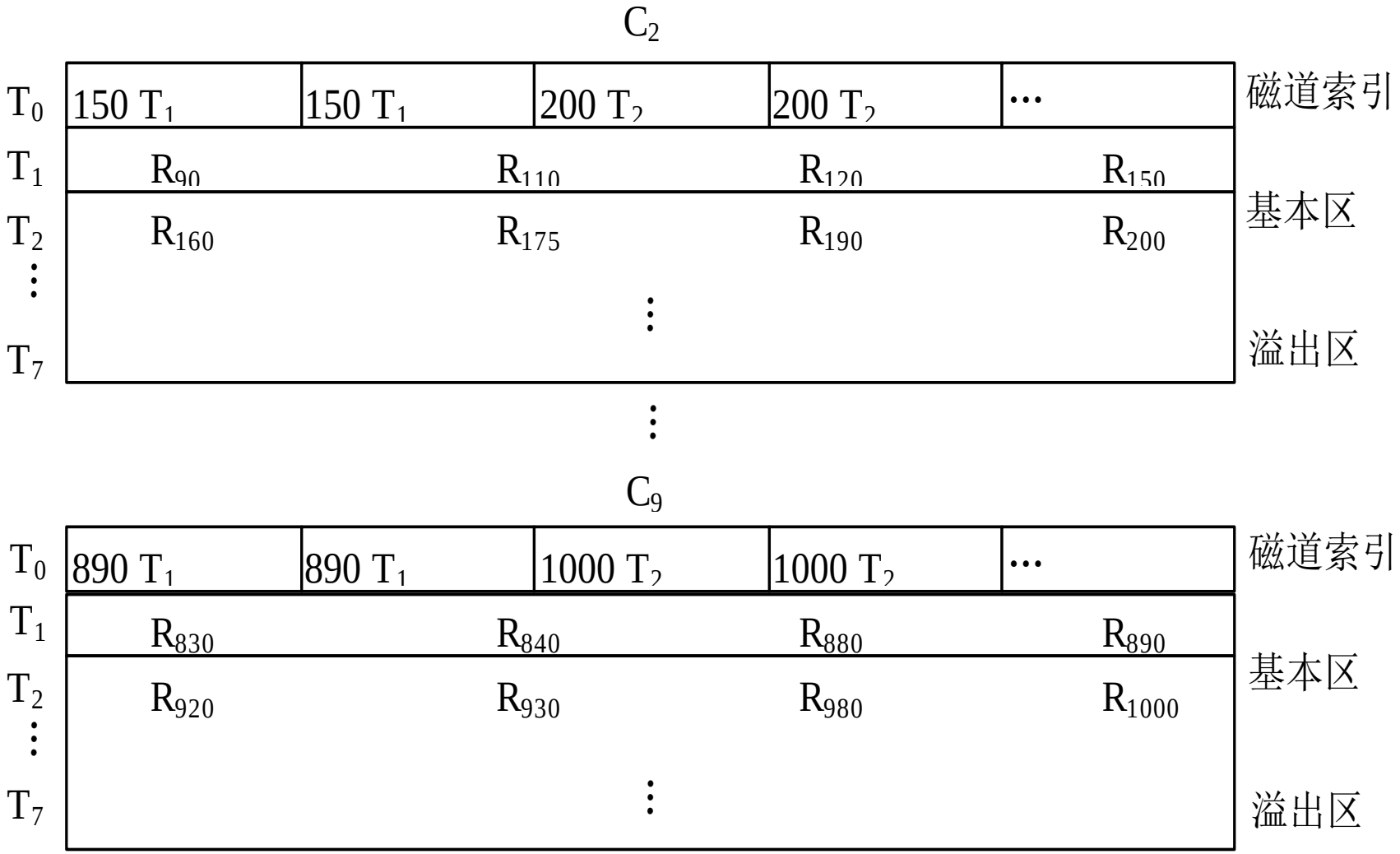
ISAM

- 多分树的应用
  - 为磁盘存取而设计
  - 采用多级索引结构
    - 主索引
    - 柱面索引
    - 磁道索引
- 解决需要频繁更新的大型数据库的一个早期尝试
- 采用基于B^+树的VSAM技术之前，IBM公司曾经广泛采用ISAM技术

ISAM结构示意图


C0
T0
400
T1
625
T2
1000
T3
主索引
T1
80
C1T0
200
C2T0
400
C3T0
柱面索引
T2
625
C6T0
T3
1000
C9T0
C1
T0
40 T1
40 T1
80 T2
80 T2
…
磁道索引
T1
R10
R20
R30
R40
T2
R50
R60
R70
R80
基本区
T7
溢出区

ISAM结构示意图


C2
T0
150 T1
150 T1
200 T2
200 T2
…
磁道索引
T1
R90
R110
R120
R150
T2
R160
R175
R190
R200
基本区
T7
溢出区
C9
T0
890 T1
890 T1
1000 T2
1000 T2
…
磁道索引
T1
R830
R840
R880
R890
T2
R920
R930
R980
R1000
基本区
T7
溢出区

思考

- 在什么情况下需要组织二级线性索引?
- 多分树的阶（子结点的个数）应该怎么确定?