FSE598 前沿计算技术

模块 3　算法设计与分析
单元 3　高级数据结构
第 4 讲　B-树与 B+ 树

本课程的B-树部分内容是基于 Thomas H.Cormen、Charles E.Leiserson 等人的“算法简介”教材

课程概要

学习内容

- B树的定义
- B树的基本操作
- 谷歌大表（BigTable）
- B+树

B-树：定义

B-树是具有以下属性的有根树：

1. 任意节点 x 都有以下字段
   a) $x.m$: int, 存储在节点 x 中的键数。
   b) m 个键按排序顺序存储：
   $$x.key_1 \leqslant x.key_2 \leqslant x.key_3 \leqslant \ldots \quad x.key_m$$
   c) $x.leaf$:*布尔值*，如果 x 为叶节点，则为 true，反之则为 false
2. 如果 x 是内部节点，则有 m+1 个指向其子节点的指针
   $$x.c_1,\ x.c_2,\ x.c_3,\ \ldots,\ x.c_m,\ x.c_{m+1}$$
3. 节点 x 中的键会分隔开存储在其子节点中的键的范围：
   $$k_1 \leqslant x.key_1 \leqslant k_2 \leqslant x.key_2 \leqslant k_3 \leqslant \ldots \leqslant x.key_m \leqslant k_{m+1}$$
   其中 k_i 是 x 子节点 c_i 中的任意键。
4. 每个叶节点都有相同的深度，也就是树的高度 h。B-树是完全平衡的。

B-树：定义（接上页）

5. 节点可持有的键数量存在下界和上界。对于已知的整数 $t \geq 2$，称其为“B-树的**最小度**”

   a) 下界

   如果该树非空，则根节点的键数量一定会 ≥ 1

   非根节点的键数量一定 $\geq t-1$

   非根/非叶节点的子节点数量一定 $\geq t$

   b) 上界

   任意节点都有 $\leq 2t-1$ 的键数

   任意节点都有 $\leq 2t$ 的子节点数

   如果一个节点正好有 2t-1 个键，我们就称其为“**全节点**”。

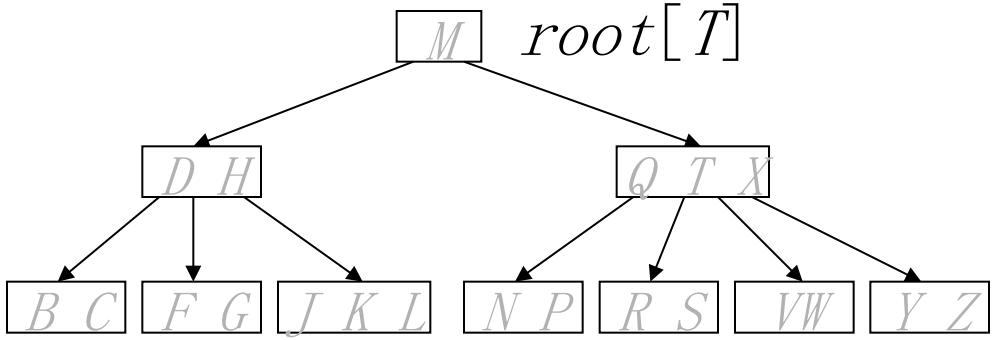
B-树：小型示例

最简单的 B-树是最小度 t=2 的树。

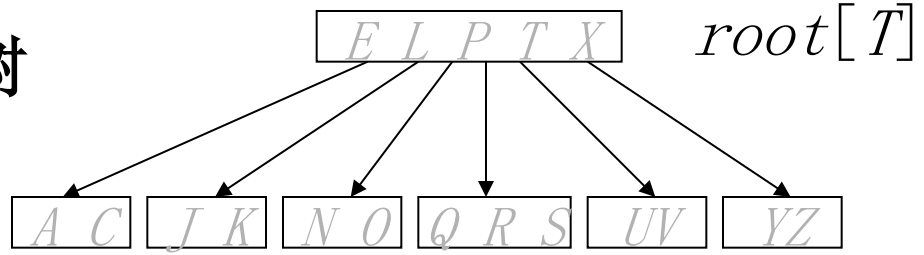
那么，如果内部节点的子节点数均为 2 个、3 个或 4 个，则称之为“**2-3-4 树**”。在实践中，t 的数值可能会大得多。

示例：2-3-4 树和 t=3 的 B-树。

2-3-4 树

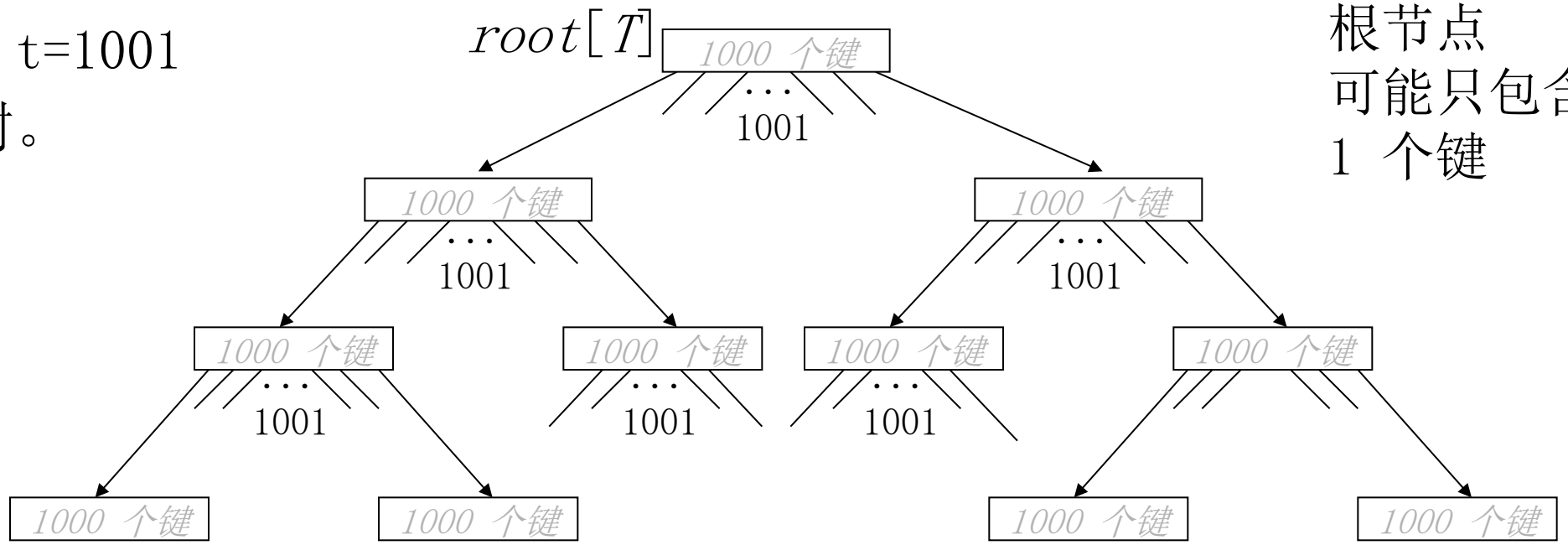


t = 3 的 B-树



B-树：大型示例

最小度 t=1001
的 B-树。



根节点
可能只包含
1 个键

B-树的属性：

- 处于平衡状态，且树的高度为 $h \le \log_t(^{n+1}/_2)$
- 存储的键总数为 $n \ge 2t^h - 1$
- （证明过程省略）

h=3, t=1000

$n \approx 2*10^9$

B-树的应用

B-树是红黑树的自然延伸：它更加平衡，允许有两个和两个以上的子节点。

在可以使用红黑树的情况下，也可以使用 B-树。

B-树的一个特殊别的应用是在辅助内存（磁盘）中

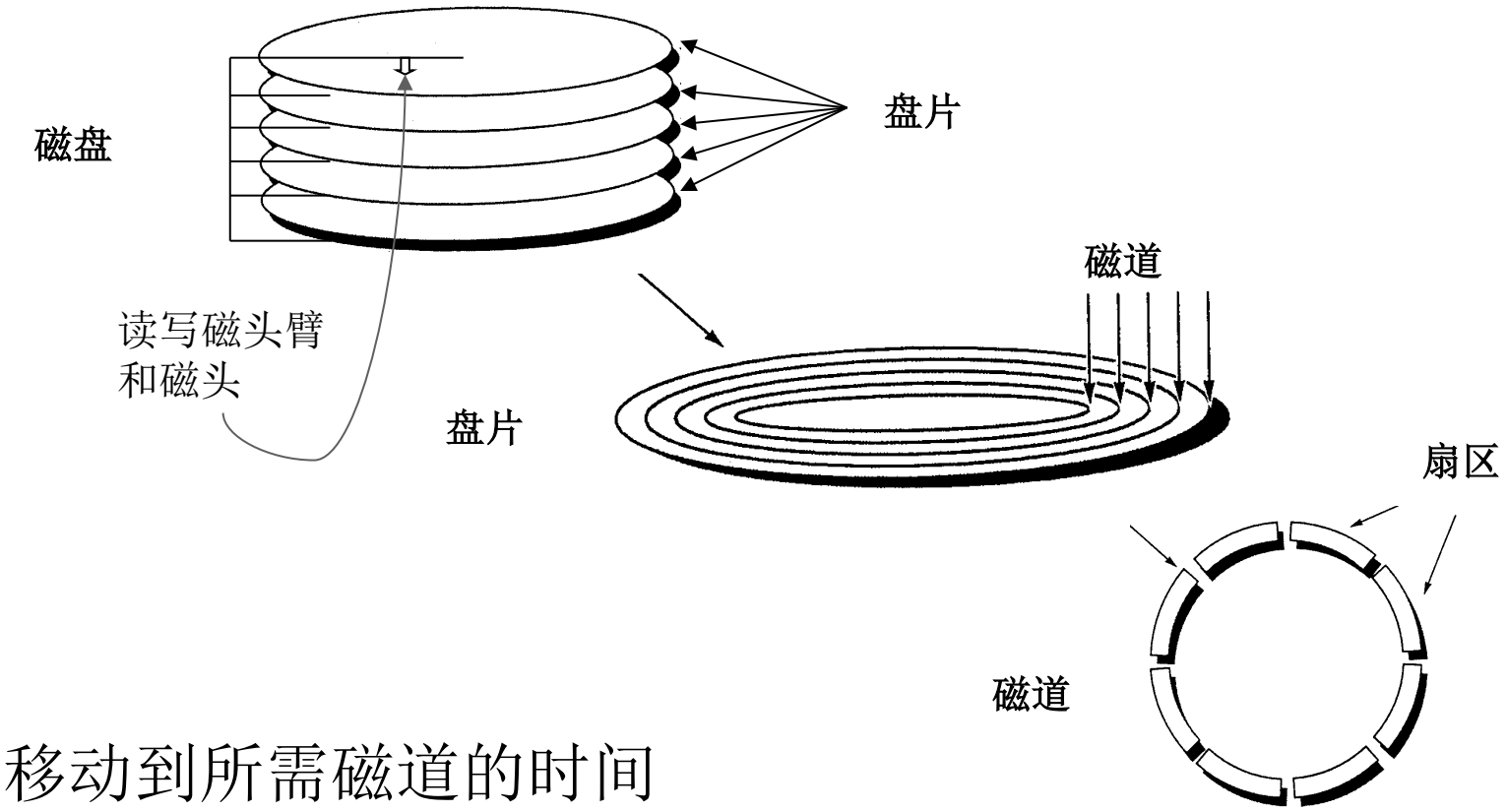
- 计算机的主内存在大小和成本上都会受到限制
- 数据库总是太大，而无法放入内存中
- 虚拟内存可以自动管理内存和磁盘之间的数据互传，但不了解数据结构的逻辑组织 - 以固定页面大小为互传单位。
- **B-树可以提供一种有效的方法来管理内存和磁盘之间的数据互传。**

内存的层级和访问时间

层级	内存技术	一般大小	一般访问时间	每 GB 的成本
寄存器	D 触发器	32 位寄存器	50 - 200 ps	不适用
Cache	SRAM	0.5 GB	0.5 - 2.5ns	200-500 美元/GB
主存储器	DRAM	16 GB	50 - 70 ns	20-75 美元/GB
辅助存储	磁盘	2 TB	5 - 20 ms	0.20-2 美元/GB

ps：皮秒：10^{-12} 秒

磁盘和磁盘访问时间



寻道时间：将磁头移动到所需磁道的时间

旋转延迟：将扇区移动到磁头下的时间

大数据块中的数据传输，例如，扇区 = 节点

B-树的基本操作

目标：尽量缩短

- 执行步骤中的运行时间，
- 磁盘访问的次数

```
B-Tree-Search(x, k)
1.   i := 1
2.   while i ≤ x.m and k > x.key_i do   // find position
3.       i := i+1                       // within a node
4.   if i ≤ x.m and k == x.key_i
5.       then return (x, i)
6.   if x.leaf == true
7.       then return nil
8.       else  Disk-Read(x.c_i)   // have to go into a child
9.             return B-Tree-Search(x.c_i, k)
```

B-树操作的复杂度

搜索 n 个节点的 B-树

- 执行步骤的运行时间 = $O(t \cdot h) = O(t \cdot \log_t n)$
- 磁盘访问的次数 = $O(h) = O(\log_t n)$

*“插入”*和*“删除”* 很烦杂。

- 插入：如果节点为全节点，我们则需要拆分节点
- 删除：如果节点具有最小度键数，我们则需要重新构建该树。

但从 n 个节点的 B-树上 *插入/删除*一个节点的复杂度与“搜索”操作的相同：

- 执行步骤的运行时间 = $O(t \cdot h) = O(t \cdot \log_t n)$
- 磁盘访问的次数 = $O(h) = O(\log_t n)$

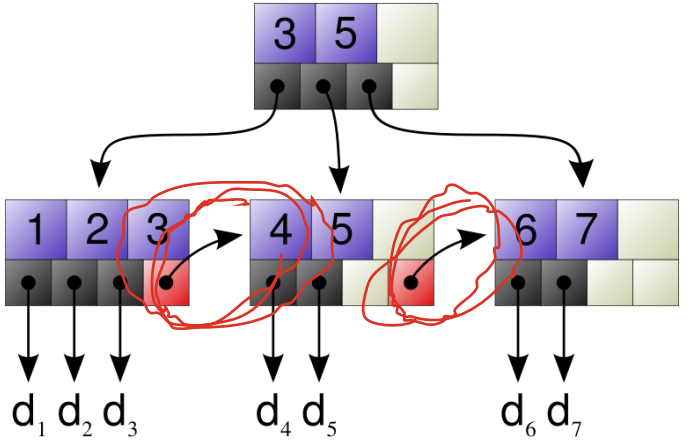
Google BigTable

http://en.wikipedia.org/wiki/BigTable

- BigTable 是一个快速、超大规模的数据库管理系统；
- 它是一个基于谷歌文件系统（GFS）构建的压缩、高性能和专有数据库系统；
- 该系统背弃了具有固定的列数关系数据库的惯例；
- GFS数据库是以“稀疏、分布式多维排序图”方式存储键-值对。
- 其思路类似于 B+ 树，可高效地插入、检索和删除节点。
- 什么是B+ 树？

B+ 树： http://en.wikipedia.org/wiki/B%2B_tree

- 一种用于表示排序数据的树
  可通过某种方式高效地插入、检索和删除记录，每个记录都由一个*键*进行标识：在键-值对中，数值即为“记录”。
- 一个动态的多级索引，每个索引节点中的键数存在最大和最小界限。
- 在 B+ 树中，所有记录都存储在该树的最低层级中。只有键存储在内部数据块中
- 在排序方向上，存在从姐妹节点到下一个节点的链接



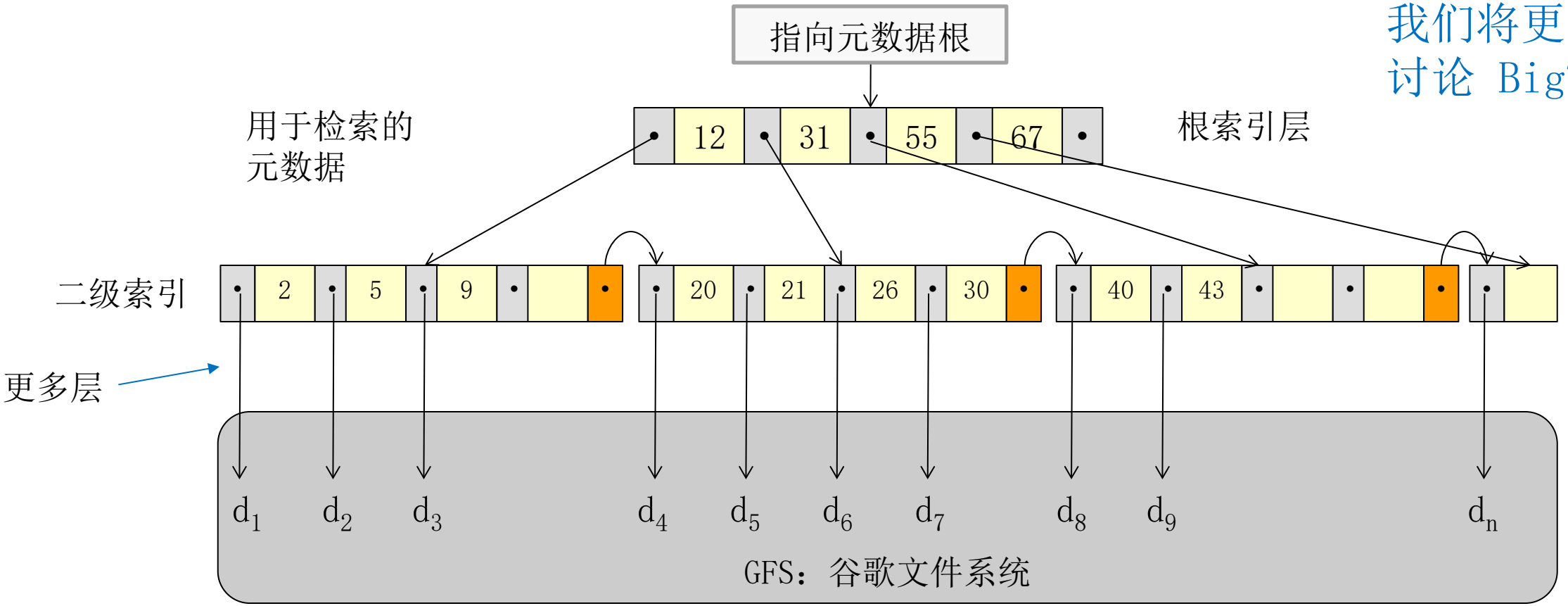
Data 都存在葉層，中間都是存Index

B+ 树仅用于存储索引

可用于从当前位置继续搜索

基于 B+ 树的 BigTable，具有多级索引

在学习谷歌云时，我们将更详细的讨论 BigTable



B-树与 B+ 树的总结

我们探讨了：

- 何谓 B-树（定义）
- B-树的属性及其与红黑树之间的区别
  - 完全平衡 — 高度较低
  - 子节点数量大
  - 每个节点的键数多
- 在磁盘访问相关操作的应用
  - 每个节点都以数据块的形式存储在磁盘上
  - 磁盘的数据块访问
- 搜索算法
- 在运行时间分析中增添新的维度
  - 一般的分步运行时间
  - 磁盘访问的次数 — 机器人转动和距离
- 字典操作：***搜索、插入和删除***
  - 执行步骤的运行时间 = $O(t \cdot \log_t n)$
  - 磁盘访问的次数= $O(\log_t n)$