大数据算法

第四讲 外存算法概述

哈尔滨工业大学
王宏志
wangzh@hit.edu.cn

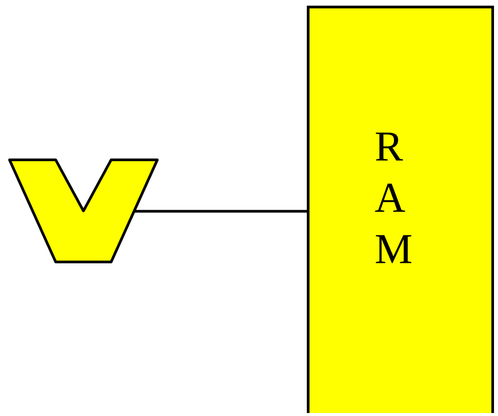
本讲内容

4.1 外存存储结构与外存算法

4.2 外存算法示例：外存排序算法

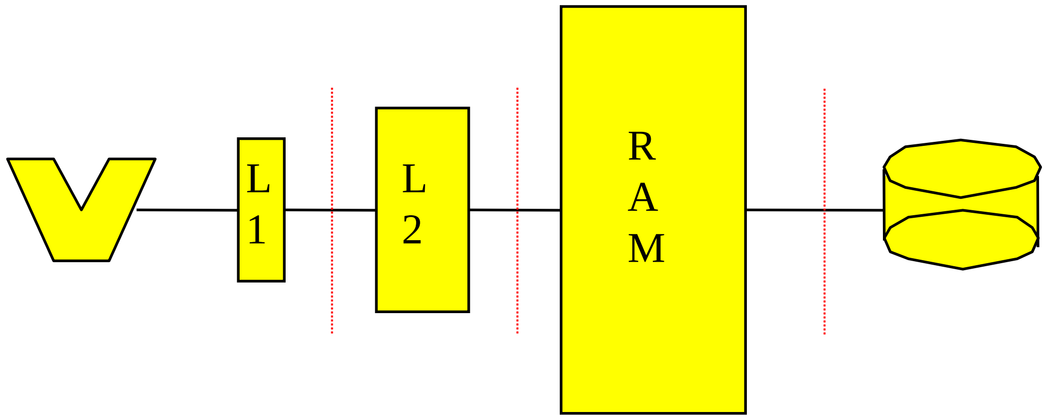
4.3 外存数据结构示例：外存查找树

随机存取机模型



- 标准计算理论模型：
  - 无限内存
  - 统一访问代价
  - 简单的模型为计算机行业成功的关键

分层存储



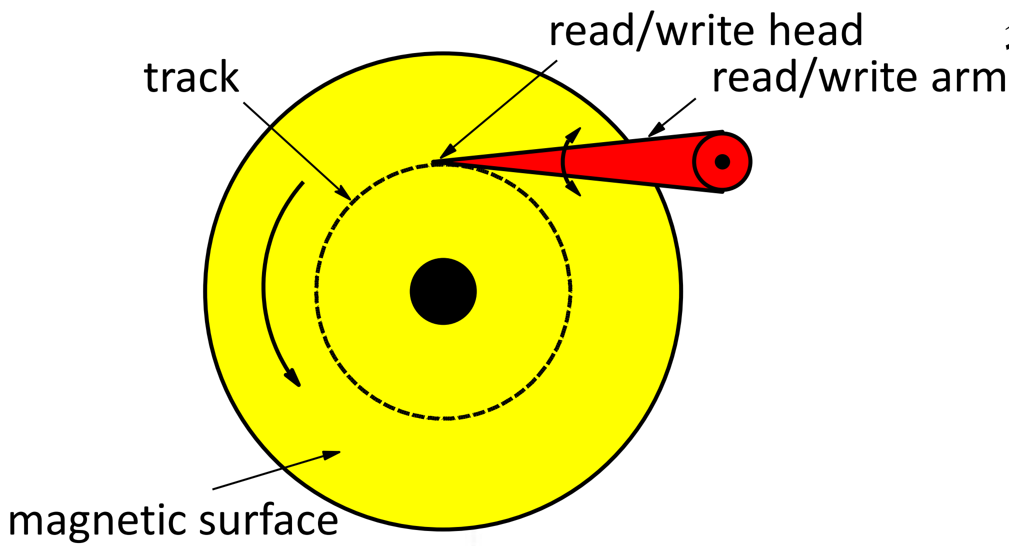
- 现代计算机有复杂的存储层次
  - 存储量得到较大提升，但较慢的层次进一步远离CPU
  - 以块为单位的数据移动

慢速 I/O

现代CPU和磁盘之间在速度上的差异是类似于在一办公桌上使用转笔刀或坐飞机到世界的另一边和在办公桌用卷笔刀削铅笔在速度上的差异。”

（D. Comer)

- 磁盘访问比主存访问的速度慢106倍



– 磁盘系统通过传输大规模连续的数据块来平摊巨大的访问代价（8-16K字节）

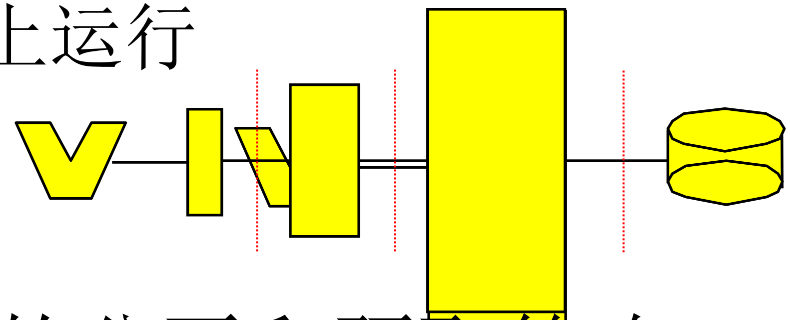
– 重要的是要利用块高效存储/访问数据

可扩展性问题

大多数程序在RAM模型上运行

-由于操作系统按需访问块

可以在大型数据集上运行

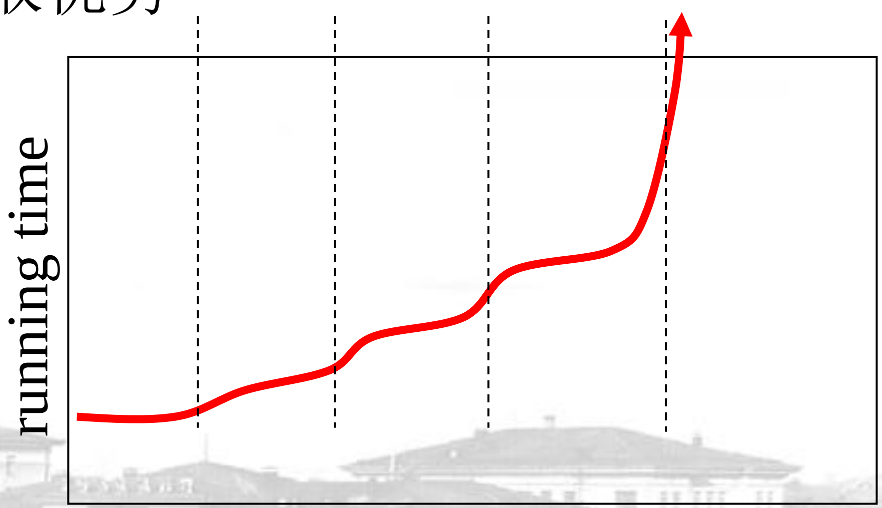


现代操作系统采用先进的分页和预取策略

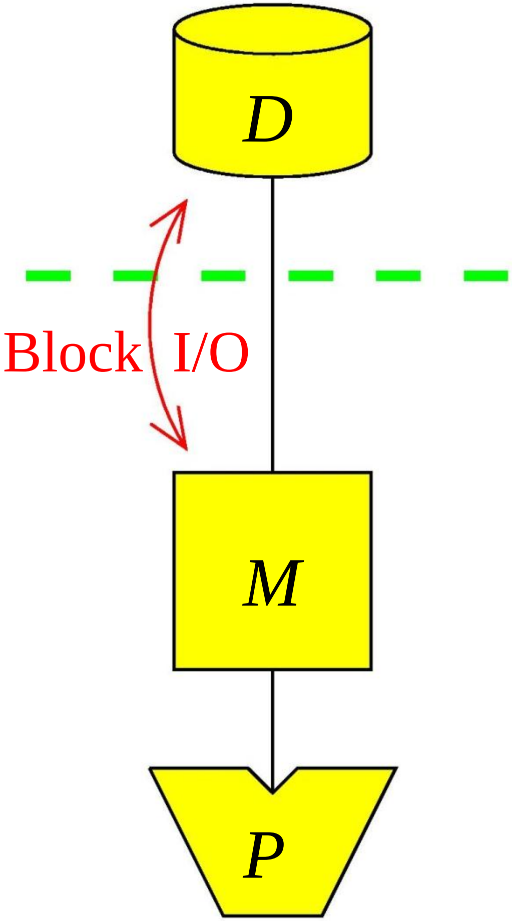
但是，如果程序分散的访问磁盘上的数据，即使是好操作系统也无法利用数据块存取优势

⇓

可扩展性的问题！



外部存储器模型



N = # 问题实例数据项个数

B = # 每个磁盘块中数据项个数

M = # 内存能容纳的数据项个数

T = # 输出数据项个数

I/O:内存和磁盘之间移动的块数

为了方便，我们假设：

$M > B^2$

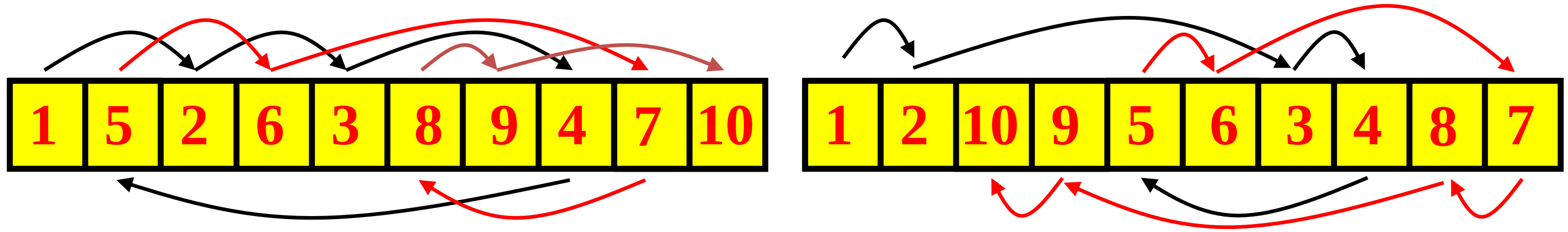
基本界限

	内存算法	外存算法
• 浏览:	N	$\frac{N}{B}$
• 排序:	$N \log N$	$\frac{N}{B}\log_{M/B}\frac{N}{B}$
• 置换	N	$\min\{N, \frac{N}{B}\log_{M/B}\frac{N}{B}\}$
• 查找:	$\log_2 N$	$\log_B N$

- 注意:
  - 线性 I/O: $O(N/B)$
  - 置换不是线性的
  - 置换和排序范围在所有的实际情况是平等的
  - B是很重要的因素: $\frac{N}{B} < \frac{N}{B}\log_{M/B}\frac{N}{B} \ll N$
  - 无法用搜索树优化排序

可扩展性问题：块访问的影响

- 例如：遍历链表（链表排序）
  数组大小N = 10（个元素）
  磁盘区域大小B = 2 （个元素）
  主存大小M = 4 （个元素） (2个磁盘块)



算法 1: N=10 I/Os　　　　算法 2: N/B=5 I/Os

- N 和 N/B之间因为磁盘块大而差异较大
  - 例如: $N = 256 \times 10^6$, $B = 8000$, 1ms磁盘访问时间
    $\Rightarrow$ N 次I/O 需要 256×10^3 sec = 4266 min = 71 hr
    $\Rightarrow$ N/B 次I/Os 需要 256/8 sec = 32 sec

队列和堆栈

- 队列:
  - 维护在主存中的push和pop块

⇓

$O(1/B)$次Push/Pop操作

- 堆栈:
  - 维护在主存储器PUSH / POP块

⇓

$O(1/B)$次Push/Pop操作

本讲内容

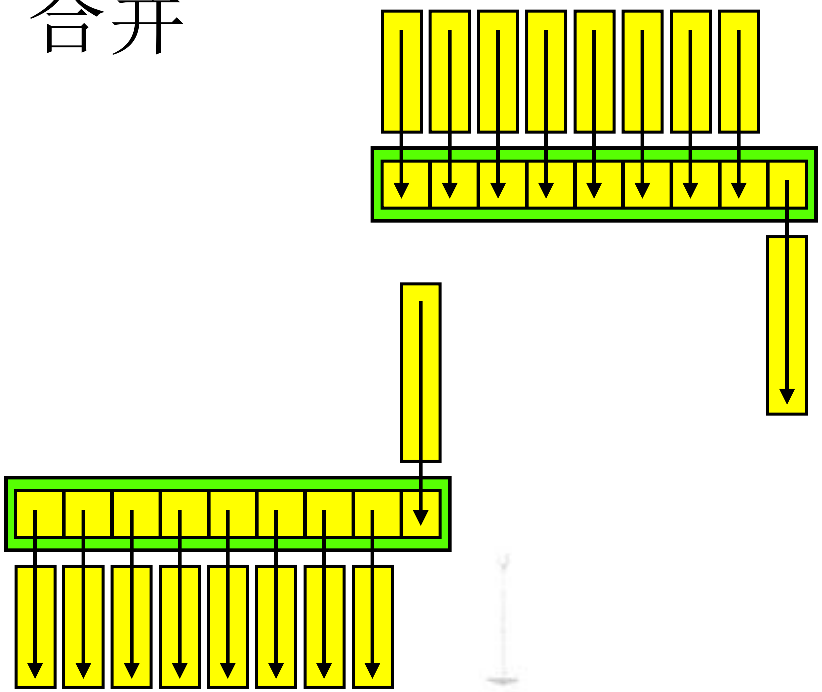
4.1 外存存储结构与外存算法

4.2 外存算法示例：外存排序算法

4.3 外存数据结构示例：外存查找树

排序

- $<M/B$ 个排序列表(队列) 可以在$O(N/B)$ I/Os内合并

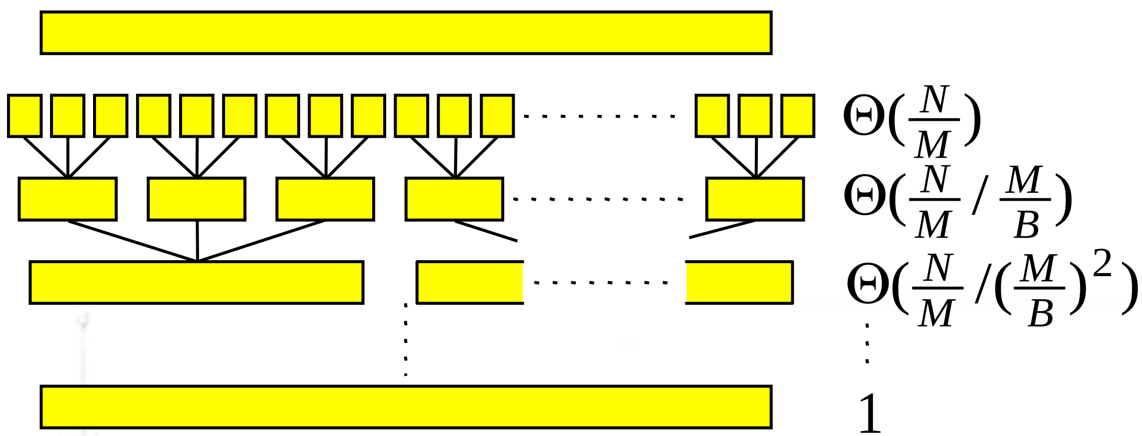


内存中M/B个磁盘块

- 未排序的列表（队列）可以使用$<M/B$个分割元素利用$O(N/B)$次 I/O实现划分

排序

- 归并排序:
  - 创建*N/M*个内存大小的有序列表
  - 重复归并列表，每次*Θ*(*M/B*)路

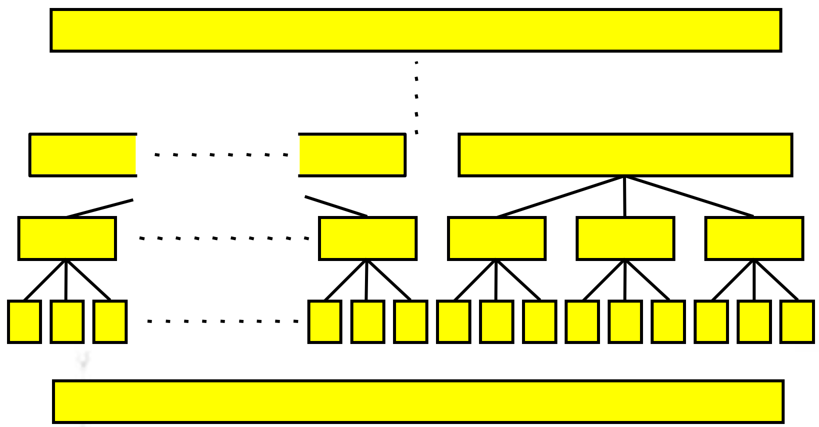


$\Rightarrow O(\log_{M/B} \frac{N}{M})$个阶段，每个阶段$O(N/B)$次I/O$\Rightarrow$

$\Rightarrow O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ 次I/O

排序

- 分配排序(多路快速排序)
  - 计算 $\Theta(M/B)$个分割元素
  - 将无序列表分散为$\Theta(M/B)$个等长无序列表
  - 递归分割列表，直到每个列表的大小能放到内存中



$\Rightarrow O(\log_{M/B} \frac{N}{M})$ 个阶段

$\Rightarrow O(\frac{N}{B}\log_{M/B} \frac{N}{B})$ 次 I/O，如果分割元素可以用$O(N/B)$次I/O计算出来

计算分割元素

- 在（确定性）内存快速排序中，分割元素（中位数）可以用线性时间选择

- 选择算法：寻找第i个元素的排序顺序

  1)各组5个元素中选择中位数

  2)递归选择中位数：~ *N/5 选定的5 个元素*

  3)用中位数将元素分配到两个列表

  4)对两个列表之一递归选择

  分析:

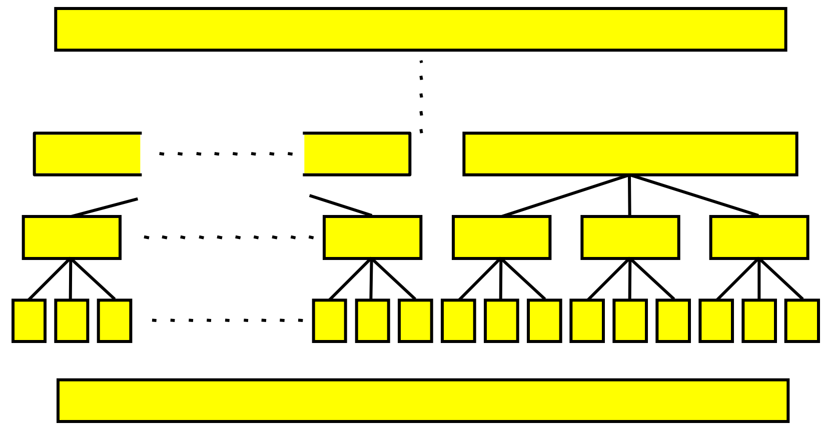
  – 步骤 1 和 3 在*O*(*N*/*B*)次 I/O内完成

  – 步骤4最多在~$\frac{7}{10}N$个元素上递归执行

  $\Rightarrow T(N) = O(N/B) + T(N/5) + T(7N/10) = O(N/B)$ 次I/O

排序

- 分配排序（多路快速排序）：



- 计算分割元素:
  - $\Theta(M/B)$ 次线性I/O选择$\Rightarrow$ $O(NM/B^2)$ 次I/O的算法
  - 但是可以用选择算法在$O(N/B)$ 次I/O内计算$\sqrt{M/B}$个分离元素,分割成大小<$\frac{3}{2}\frac{N}{\sqrt{M/B}}$的列表

$\Rightarrow O(\log_{\sqrt{M/B}} \frac{N}{M}) = O(\log_{M/B} \frac{N}{M})$ 个阶段$\Rightarrow$ $O(\frac{N}{B}\log_{M/B}\frac{N}{B})$ 算法

计算分割元素

1) 抽取 $\frac{4N}{\sqrt{M/B}}$ 个元素:
   - 创建*N/M* 个内存 大小的排序列表
   - 从每个排序列表中逢第 $\frac{1}{4}\sqrt{M/B}$ 个元素则选择

2) 从抽样中选择 $\sqrt{M/B}$ 个分割元素:
   - 使用选择算法 $\sqrt{M/B}$ 次，逢第 $\frac{4N}{\sqrt{M/B}} \Big/ \sqrt{M/B} = \frac{4N}{M/B}$ 个元素则选择

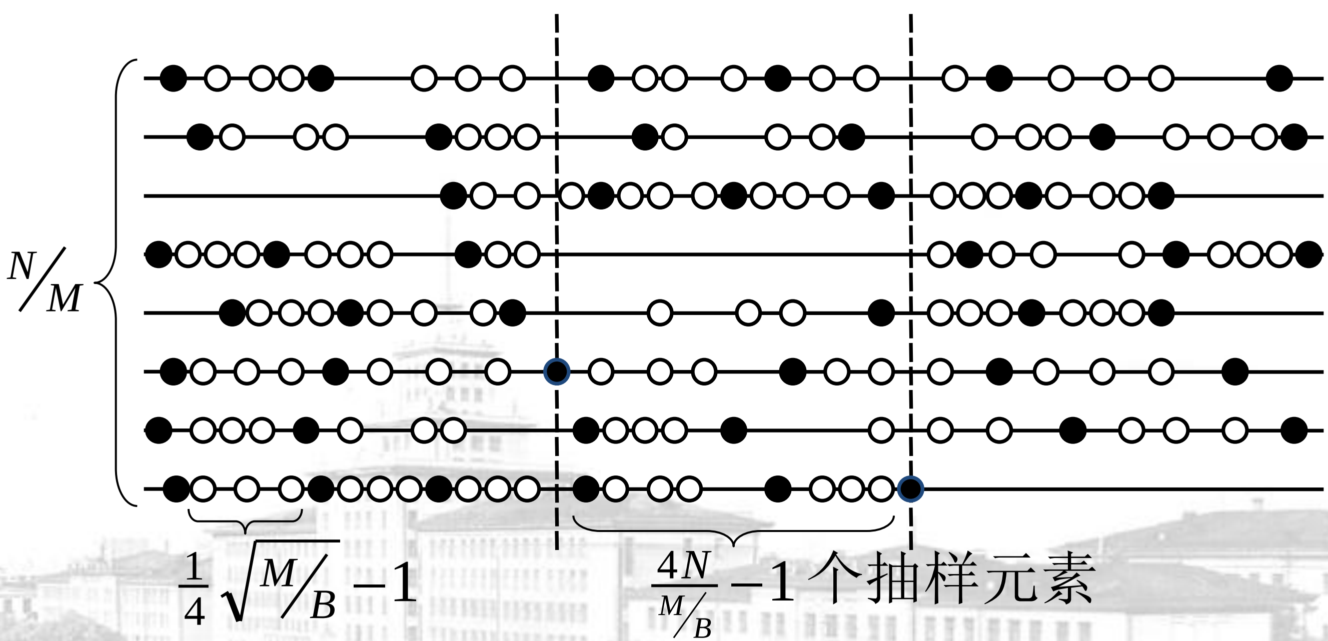
- 分析:
  - 步骤 1 在*O*(*N*/*B*)次I/O内完成
  - 步骤2 在 $\sqrt{M/B} \cdot O(\frac{N}{\sqrt{M/B}B}) = O(N/B)$ 次I/O内完成

  $\Rightarrow$ *O*(*N*/*B*)次I/O

计算分割元素

1) 抽取 $\frac{4N}{\sqrt{M/B}}$ 个元素:
- 创建*N/M* 个内存 大小的排序列表
- 从每个排序列表中逢第$\frac{1}{4}\sqrt{M/B}$个元素则选择

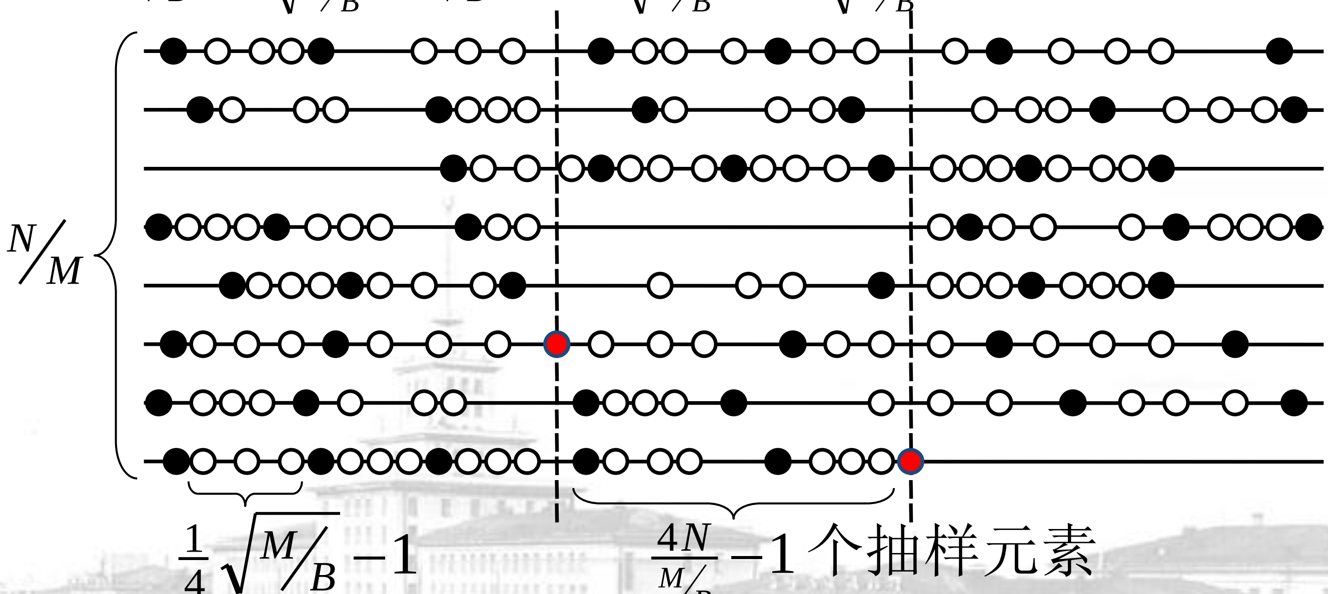
2) 从抽样中选择$\sqrt{M/B}$个分割元素:
- 使用选择算法$\sqrt{M/B}$ 次，逢第 $\frac{4N}{M/B}$ 个元素则选择



计算分割元素

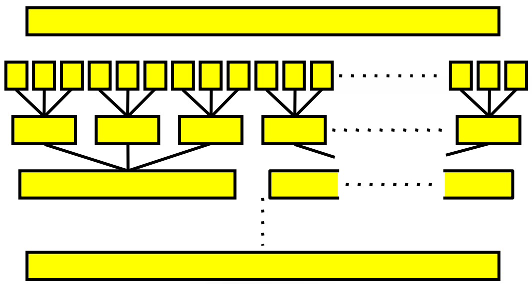
- 范围R内元素定义为连续的分割元素
- R中的抽样个数: $\frac{4N}{M/B} - 1$
- R中抽样元素间的元素个数 $(\frac{4N}{M/B} - 1)\cdot(\frac{1}{4}\sqrt{M/B} - 1)$
- R 中抽样元素和 R外抽样元素间的元素个数: $2\frac{N}{M}\cdot(\frac{1}{4}\sqrt{M/B} - 1)$

$$\Rightarrow < \frac{4N}{M/B} + (\frac{N}{\sqrt{M/B}} - \frac{4N}{M/B}) + \frac{N}{2B\sqrt{M/B}} < \frac{3}{2}\frac{N}{\sqrt{M/B}}$$



排序小结

- 外部合并或分布排序需要$O(\frac{N}{B}\log_{M/B}\frac{N}{B})$次I/O
  - 基于归并排序的M/ B路归并排序
  - 基于$\sqrt{M/B}$路分布和分割元素查找的分布排序
- 最优？



本讲内容

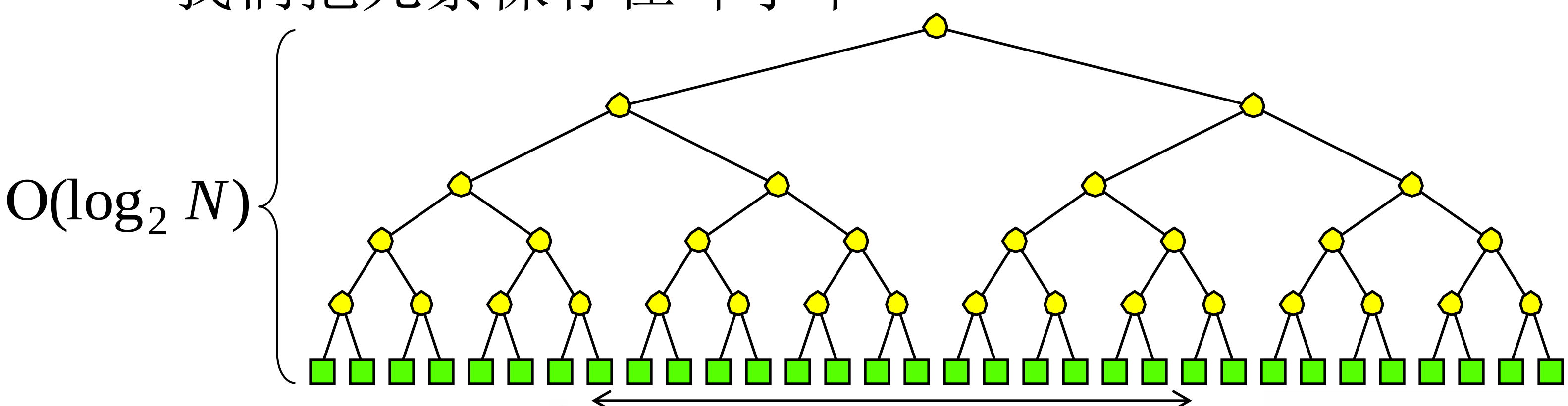
4.1 外存存储结构与外存算法

4.2 外存算法示例：外存排序算法

4.3 外存数据结构示例：外存查找树

外部搜索树

- 二叉搜索树:
  - 在N个元素之间搜索的标准方法
  - 我们把元素保存在叶子中

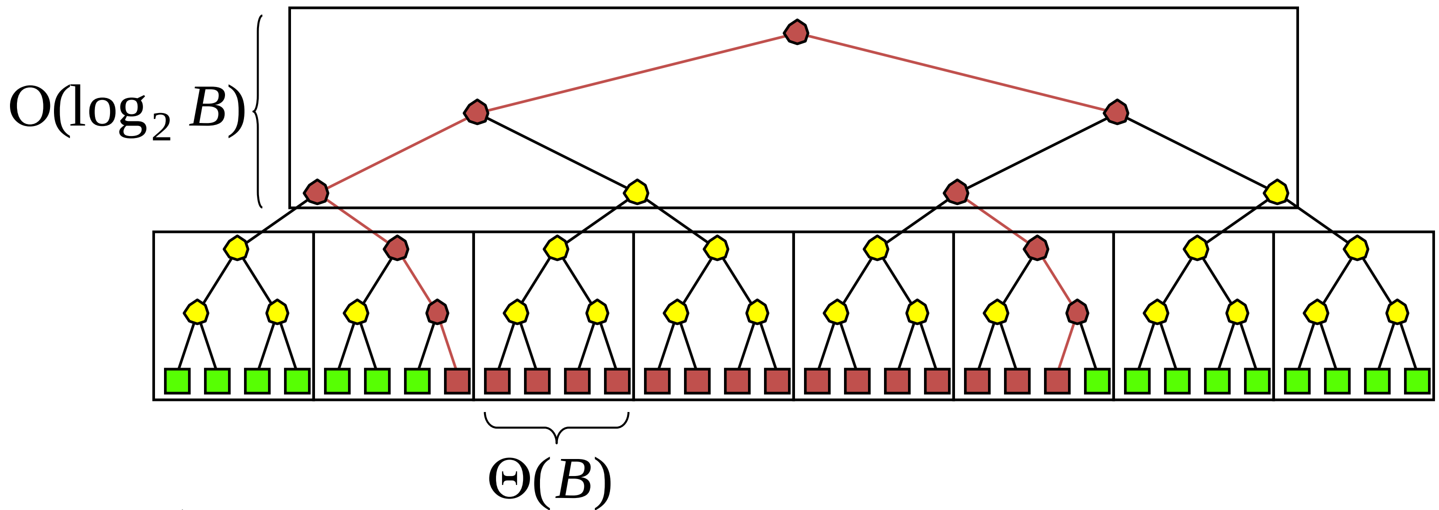


  - 搜索路径至少需要访问一条根到叶的路
  - 如果结点存储在磁盘的任意位置

    $\Rightarrow$ 搜索: $O(\log_2 N)$次I/O

    $\Rightarrow$范围搜索$O(\log_2 N+T)$次 I/O

外部搜索树



- BFS 块:
  - 块高度 $O(\log_2 N)/O(\log_2 B) = O(\log_B N)$
  - 按块输出元素

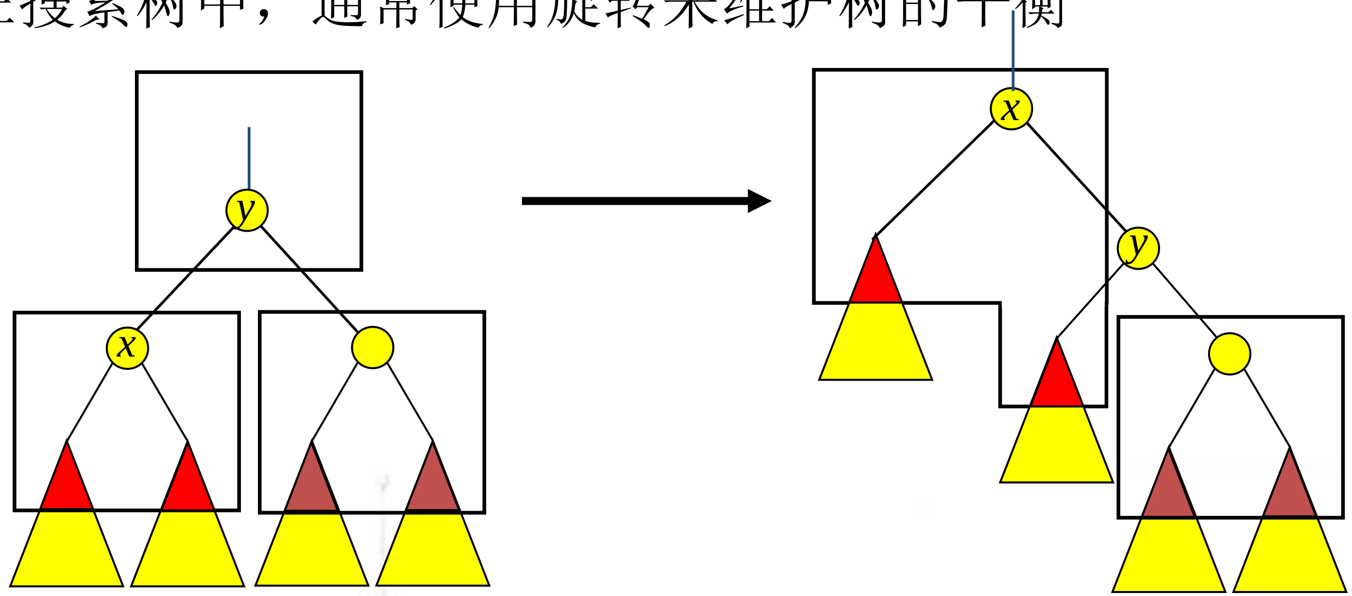
$\Downarrow$

范围搜索 $O(\log_B N + T/B)$ 次I/O

- 最优: $O(N/B)$ 空间和 $O(\log_B N + T/B)$ 次查询

外部搜索树

- 在更新过程中如何维护 BFS 块?
  - 在搜索树中，通常使用旋转来维护树的平衡



- 通过旋转来维护BFS块看起来是件非常困难的事情
  - 也需要确定输出（叶子）也是成块的!

致谢

- 本讲义部分内容来自于Lars Arge的讲义