CS2045M: 高级数据结构与算法分析 **2025-2026 学年秋冬学期**

Lecture 15: 外部排序

编写人：吴一航 yhwu_is@zju.edu.cn

很高兴，我们走到了这门课，也是浙江大学数据结构与算法课程系列的最后一个章节。作为两个学期数据结构与算法的收尾，我们将第一次讨论考虑外存的算法（当然之前的 B+ 树和倒排索引讨论了外存中的数据结构）。

15.1 外部排序的引入

大部分内部排序算法都用到内存可直接寻址的事实。希尔排序用一个时间单位比较元素 $A[i]$ 和 $A[i-h_k]$。堆排序用一个时间单位比较元素 $A[i]$ 和 $A[i*2+1]$。使用三数中值分割法的快速排序以常数个时间单位比较 $A[Left]$、$A[Center]$ 和 $A[Right]$。如果输入数据在磁带上，那么所有这些操作就失去了它们的效率，因为磁带上的元素只能被顺序访问（磁头只能顺序移动，如果不顺序访问那就很耗时间）。即使数据在一张磁盘上，由于转动磁盘和移动磁头所需的延迟，仍然存在实际上的效率损失。

基于这一原因，在外部排序时适合使用的是归并排序，因为归并排序的时间复杂度来源于归并，而归并只需要顺序扫描两个有序数组，然后写入的时候也是顺序写入，因此适用于外部排序。

15.2 简单算法

15.2.1 算法描述

PPT 第 3 页给出了一个简单的归并想法，更一般化的描述如下：设有四盘磁带，$T_{a_1}, T_{a_2}, T_{b_1}, T_{b_2}$，它们是两盘输入磁带和两盘输出磁带，并且功能是交替的（后面会看到）。

设数据最初在 T_{a_1} 上，并设初始数组有 N 个元素，并且内存可以一次容纳（和排序）M 个记录。一种自然的做法是第一步从输入磁带一次读入 M 个记录，在内部将这些记录排序，然后再把这些排过序的记录交替地写到 T_{b_1} 或 T_{b_2} 上。我们将每组排序好的记录叫做一个顺串（run）。

现在 T_{b_1} 和 T_{b_2} 各包含一组顺串，下一步，b 就变成了输入磁带，a 因为上面的内容没用了所以可以变成输出磁带。将 T_{b_1} 和 T_{b_2} 上每个磁带的第一个顺串取出并将二者合并，把结果写到 T_{a_1} 上，该结果是一个二倍长的顺串。然后，再从每盘磁带取出下一个顺串，合并，并将结果写到 T_{a_2} 上。继续这个过程，交替使用 T_{a_1} 和 T_{a_2}，直到 T_{b_1} 或 T_{b_2} 为空。此时，或者它们均为空，或者剩下一个顺串。对于后者，把剩下的顺串拷贝到适当的磁盘末尾即可。

进一步地，重复相同的步骤，这一次用两盘 a 磁带作为输入，两盘 b 磁带作为输出，结果得到一些 $4M$ 的顺串。继续这个过程直到得到长为 N 的一个顺串。显然，因为一趟工作顺串长度加倍，该算法将需要 $\lceil \log_2(N/M) \rceil$ 趟工作（这里的趟就是 PPT 上的 pass，表示一轮从输入磁盘到输出磁盘的工作），外加一趟构造初始的顺串。例如 PPT 上的例子，第一趟构造出长度为 3 的顺串，接下来是三趟工作：顺串长度依次由 3 变 6，6 变 12，12 变 13，因此共 4 趟操作。

【讨论】若有 1000 万个记录，每个记录 128 个字节，并有 4 兆字节的内存，需要多少趟操作?

事实上第一趟将建立 320 个顺串，然后套用公式再需要 9 趟以完成排序，因此共需要 10 趟操作。

15.2.2 优化目标

显然这一基础算法还有很多值得优化的地方，PPT 第 4 页给出了一些优化方向，分别对应后面要讨论的优化策略，那么闲话少说，直接进入优化策略的讨论。

15.3 多路合并

第一个优化方向就是减少工作的趟数，显然如果使用 k 路的归并，也就是每次归并 k 条纸带上对应位置的顺串，那么每次合并后顺串长度增加 k 倍，因此加上初始的 1 趟，只需要 $1 + \lceil \log_k(N/M) \rceil$ 趟即可完成排序，减少了趟数，因此减少了磁带移动的次数，从而减少总时间。这样的算法称为 k 路合并。

k 路合并有一个实现上需要注意的点，因为是 k 个顺串要合并，因此需要不断的在 k 个元素中选取最小值放到输出的磁带上，这个操作可以使用优先队列来实现，PPT 第 5 页给出了这样的例子。

15.4 多相（Polyphase）合并

尽管多路合并看起来很有吸引力，但是需要注意的是，k 路合并需要的磁带数目是 $2k$，因为每一趟工作需要 k 个输入磁带和 k 个输出磁带。因此，如果 k 很大，那么磁带数目就会很多，这是不太合理的（后面会解释为什么不合理）。因此讨论另一种优化方案：多相合并，能够大大减小归并需要的磁带数目：k 路合并只需要 $k + 1$ 条磁带。

作为例子，下面阐述只用三盘磁带如何完成 2 路合并。设有三盘磁带 T_1、T_2 和 T_3，在 T_1 上有一个输入文件，它将产生 34 个顺串。一种选择是在 T_2 和 T_3 的每一盘磁带中放入 17 个顺串。然后可以将结果合并到 T_1 上，得到一盘有 17 个顺串的磁带。由于所有的顺串都在一盘磁带上，因此现在必须把其中的一些顺串放到 T_2 上以进行另一次的合并。执行合并的逻辑方式是将前 8 个顺串从 T_1 拷贝到 T_2 并进行合并。这样的效果是，为了所做的每一趟合并，不得不附加额外的复制工作，而复制也需要磁头的移动，是很昂贵的操作，因此这种方法并不好。如果接着把所有的步骤都画完，会发现总共需要 1 趟初始 + 6 趟工作，外加 5 次复制操作。

另一种选择是把原始的 34 个顺串不均衡地分成两份。设把 21 个顺串放到 T_2 上而把 13 个顺串放到 T_3 上。然后，在 T_3 用完之前将 13 个顺串合并到 T_1 上。然后可以将具有 13 个顺串的 T_1 和 8 个顺串的 T_2 合并到 T_3 上。此时，合并 8 个顺串直到 T_2 用完为止，这样，在 T_1 上将留下 5 个顺串而在 T_3 上则有 8 个顺串。然后，再合并 T_1 和 T_3，等等，直到合并结束。PPT 第 6 页给出了动态图示展示上述全部过程。

顺串最初的分配有很大的关系。例如，若 22 个顺串放在 T_2 上，12 个在 T_3 上，则第一趟合并后得到 T_1 上的 12 个顺串以及 T_2 上的 10 个顺串。在另一次合并后，T_1 上有 10 个顺串而 T_3 上有 2 个顺串。此时，进展的速度慢了下来，因为在 T_3 用完之前只能合并两组顺串。这时 T_1 有 8 个顺串而 T_2 有 2 个顺串。同样，只能合并两组顺串，结果 T_1 有 6 个顺串且 T_3 有 2 个顺串。再经过三趟合并之后，T_2 还有 2 个顺串而其余磁带均已没有任何内容。此时必须将一个顺串拷贝到另外一盘磁带上，然后结束合并，所以非常复杂。

事实上，最初给出的 $21 + 13$ 的分配是最优的，否则都可能出现上面的需要复制或者有长短不对齐导致浪费的情况。如果顺串的个数是一个斐波那契数 F_n，那么分配这些顺串最好的方式是把它们分裂成两个斐波那契数 F_{n-1} 和 F_{n-2}。否则，为了将顺串的个数补足成一个斐波那契数就必须用一些哑顺串（dummy run）来填补磁带。

还可以把上面的做法扩充到 k 路合并，此时需要第 k 阶斐波那契数用于分配顺串，其中 k 阶斐波那契数定义为

$$F^{(k)}(n) = F^{(k)}(n-1) + F^{(k)}(n-2) + \cdots + F^{(k)}(n-k)$$

辅以适当的初始条件 $F^{(k)}(1) = F^{(k)}(2) = \cdots = F^{(k)}(k-2) = 0$，$F^{(k)}(k-1) = 1$。这样就可以用 $k+1$ 盘磁带完成 k 路合并。具体操作需要说明一下，因为并不是那么平凡的。事实上经过第一步拆分之后，每个磁带上的顺串个数应当分别为

$$\begin{aligned}
\text{磁带1}&: F^{(k)}(n-1) + F^{(k)}(n-2) + \cdots + F^{(k)}(n-k) \\
\text{磁带2}&: F^{(k)}(n-1) + F^{(k)}(n-2) + \cdots + F^{(k)}(n-k+1) \\
&\cdots \\
\text{磁带}k&: F^{(k)}(n-1) \\
\text{磁带}k+1&: 0
\end{aligned}$$

然后合并磁带 1 到 k 的前 $F^{(k)}(n-1)$ 个顺串，将合并后的结果放到磁带 $k+1$ 上，此时不同磁带上的顺串个数变为

$$\begin{aligned}
\text{磁带1}&: F^{(k)}(n-2) + F^{(k)}(n-3) + \cdots + F^{(k)}(n-k) \\
\text{磁带2}&: F^{(k)}(n-2) + F^{(k)}(n-3) + \cdots + F^{(k)}(n-k+1) \\
&\cdots \\
\text{磁带}k&: 0 \\
\text{磁带}k+1&: F^{(k)}(n-1) = F^{(k)}(n-2) + F^{(k)}(n-3) + \cdots + F^{(k)}(n-k-1)
\end{aligned}$$

此时的顺串个数分布与之前的是类似的，只是下标减了 1，因此可以继续合并除磁带 k 外每个磁带的前 $F^{(k)}(n-2)$ 个顺串，将合并后的结果放到磁带 k 上，以此类推，直到顺串个数为 1，此时就完成了排序。

15.5 缓存并行处理

PPT 第 8-9 页考虑了实际实现外部排序的时候的一个问题。在实现外部排序的时候，肯定是一块一块地读入数据的，然后并不是每比较了一次就要往磁盘写一个元素，这样每次比较完就要等磁盘处理很长时间才能进行下一次比较，是非常耗时间的。实际上在 2 路合并中，应该是把内存划分为输入 2 个缓存区（buffer），输出 1 个缓存区，输入缓存区用来存放从输入磁盘读入的数据，然后两个输入缓存区的数据比较之后的排序结果先放到输出缓存区，当输出缓存区满了之后再一次性写入输出磁盘。

然而这样的实现仍然存在问题，那就是当输出缓存区要写回磁盘的时候，内存里也是什么事情都做不了。所以解决方案是，拆分成两个输出缓存区，当其中一个满了写回磁盘的时候，另一个输出缓存区继续接收数据，这样就可以实现并行处理了——一个在内存里操作，一个进行 I/O 交互。

事实上输入缓存也是一样的，如果仍然是 2 个输入缓存，当所有输入缓存中的数据都比较完了，这时也要被迫停止等待新的数据从输入磁盘中读入。因此要进一步划分为 4 个输入缓存，这样当其中两个输入缓存中的数据正在比较的时候，另外两个输入缓存可以同时并行地读入新的数据。

事实上这里就可以回答前面为什么 k 太高时，k 路合并尽管减少了趟数，但是并不一定更优的原因了。因为在 k 路合并中，根据前面的讨论，应当讲整个内存区域划分成 $2k$ 个输入缓存区和 2 个输出缓存区，这样当 k 很大的时候，输入缓存就会被划分得很细，一次能读入输入缓存的数据量就会减小（也就是 block 大小降低），那么 I/O 操作就会变多，因此 k 太大的时候尽管趟数降低导致 I/O 成本降低，但并不一定更优。因此这其中一定有一个最优的 k，当然这是与具体的机器硬件有关的。

15.6 替换选择

最后要考虑的优化方向是优化顺串的构造。迄今用到的策略是每个顺串的大小都是内存的大小，然而事实上这一长度还可以进一步扩展。原因在于某个内存位置中的元素一旦写入磁盘了，这个内存位置就可以空出来给数组后续的元素使用。只要后续的元素比现在写入的更大，就可以继续往顺串后面添加该元素。PPT 第 10 页给出了很清晰的动态例子，首先在内存中维护一个优先队列，每次从队列中取出比当前顺串中最后一个元素更大的最小的元素写入磁盘，然后再从磁盘中读入一个元素放入队列中，直到队列中的元素都比顺串的最后一个要小，就开启一个新的顺串。

上述方法通常称为替换选择（replace-ment selection）。显然，当原数组的顺序已经比较接近最终的顺序时，替换选择能够得到更长的顺串。当然，在平均情况下，替换选择生成的顺串长度为 $2M$，其中 M 是内存的大小。这意味着使用替换选择算法能使得顺串的数量降低，因此之后合并的趟数也会减少，从

而减少了 I/O 操作的次数。

当然，在生成了不同长度的顺串之后，还需要考虑如何合并这些顺串是最优的。PPT 第 11 页给出了一个例子，显然最优解就是哈夫曼树，因为目标是想让长的顺串尽可能少地参与合并，那么显然贪心地不断选择最小的两个顺串合并是最优的，当然感兴趣的读者可以用之前贪心算法的贪心选择性质和最优子结构两步证明套路来证明这一点，实际上贪心选择性质的证明，只需要交换两个顺串的合并次序发现长的放后面合并一定会让整体操作变小即可证明，最优子结构仍然是用反证法，读者只需要回顾贪心算法一节就能知道如何严谨证明，这里就不再赘述了，因为其实很显然。

祝贺大家顺利完成了这门课程的学习，希望这本讲义能够提供理解上的帮助，给予一些启发。尽管这门课的学习充满了痛苦，但我相信只要是平常认真学习了这些内容的就一定能有所收获。祝愿大家在期末考试中取得好成绩，再见！