上海师范大学 Shanghai Normal University

《算法设计与分析》

3-分治法 (Divide and Conquer)

杨启哲

上海师范大学信机学院计算机系

2023 年 9 月 21 日

主要内容

- 归并排序和快速排序
- 分治思想与主定理
- 利用分治思想设计算法

归并排序和快速排序

归并排序 (MergeSort)

假设现在存在两个已经排好序的数组 A 和 B，分别有 M 和 N 个元素。我们希望将它们合并成一个排好序的数组 C。

例 1.

令 $A = [2, 13, 43, 45, 89]$，$B = [6, 24, 51, 90, 93]$，则将其排列后得到 C:

$$C = [2, 6, 13, 24, 43, 45, 51, 89, 90, 93]$$

- 最好情况下需要比较多少次? $\min\{M, N\}$
- 最坏情况下需要比较多少次? $M + N - 1$

归并算法

上海师范大学 Shanghai Normal University

算法：MERGE$(A[1,\ldots,n], p, q, r)$

输入： n 元数组 $A[1,\ldots,n]$ 和整数 p, q, r，满足 $1 \leqslant p \leqslant q < r \leqslant n$，且 $A[p,\ldots,q]$ 和 $A[q+1,\ldots,r]$ 都已排好序

输出： 排好序的 $A[p,\ldots,r]$

```
1:  s ← p, t ← q + 1, k ← p
2:  while s ⩽ q and t ⩽ r do
3:      if A[s] ⩽ A[t] then
4:          B[k] ← A[s]
5:          s ← s + 1
6:      else
7:          B[k] ← A[t]
8:          t ← t + 1
9:      k ← k + 1
10: if s = q + 1 then B[k,...r] ← A[t,...,r]
11: else B[k,...r] ← A[s,...,q]
12: A[p,...,r] ← B[p,...,r]
```

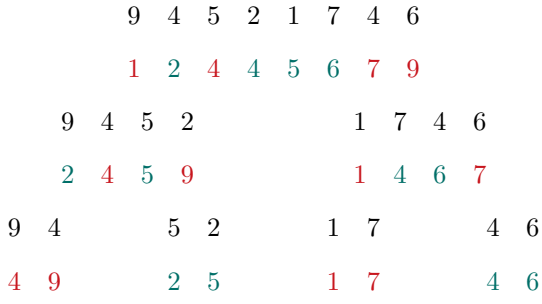
这里 $B[p,\ldots r]$ 是一个辅助数组.

归并排序的基本思想

- 将数组 A 分成两个一样大的子数组 A_1 和 A_2.
- 递归去排序 A_1 和 A_2.
- 将排好序的 A_1 和 A_2 合并成一个排好序的数组.

归并排序的运算过程

考虑下列数组:



归并排序算法

上海师范大学 Shanghai Normal University

算法： $\mathrm{MergeSort}(A[1,\ldots,n])$

输入： n 元数组 $A[1,\ldots,n]$

输出： 排好序的 $A[1,\ldots,n]$

1: $\mathrm{mergesort}(A,1,n)$

过程： $\mathrm{mergesort}(A,low,high)$

2: **if** $low < high$ **then**

3: $\quad mid \leftarrow \lfloor (low+high)/2 \rfloor$

4: $\quad \mathrm{mergesort}(A,low,mid)$

5: $\quad \mathrm{mergesort}(A,mid+1,high)$

6: $\quad \mathrm{MERGE}(A,low,mid,high)$

归并排序复杂性

令 $C(n)$ 为算法 MergeSort 对一个 n 元数组排序所需的比较次数，简单起见，假设 n 是 2 的幂，则我们有:

$$C(n) = \begin{cases} 0 & n = 1 \\ 2C(\frac{n}{2}) + m & n > 1,\ \frac{n}{2} \leqslant m \leqslant n-1 \end{cases}$$

令 $C_1(n)$ 和 $C_2(n)$ 分别表示 $C(n)$ 递推时 m 分别取最小值和最大值，即:

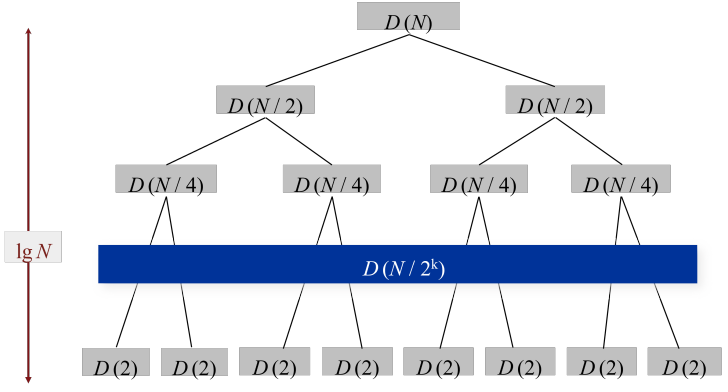
$$C_1(n) = \begin{cases} 0 & n = 1 \\ 2C_1(\frac{n}{2}) + \frac{n}{2} & n > 1 \end{cases},\ C_2(n) = \begin{cases} 0 & n = 1 \\ 2C_2(\frac{n}{2}) + n - 1 & n > 1 \end{cases}$$

显然我们有:$C_1(n) \leqslant C(n) \leqslant C_2(n)$.

计算 $C_1(n)$

$$C_1(n) = 2C_1(\tfrac{n}{2}) + \tfrac{n}{2}$$

图: 从图展开的角度



N	$= N/2$
$2\ (N/4)$	$= N/2$
$4\ (N/8)$	$= N/2$
...	
$2^k\ (N/2^{k+1})$	$= N/2$
...	
$N/2\ (1)$	$= N/2$
$N/2\ \lg N$	

从展开递推式的角度计算 $C_2(n)$

$$
\begin{aligned}
C_2(n) &= 2C_2(\frac{n}{2}) + n - 1 \\
&= 2(2C_2(\frac{n}{4}) + \frac{n}{2} - 1) + n - 1 \\
&= 4C_2(\frac{n}{4}) + n - 2 + n - 1 \\
&= 4(2C_2(\frac{n}{8}) + \frac{n}{4} - 1) + n - 2 + n - 1 \\
&\vdots \\
&= 2^k C_2(\frac{n}{2^k}) + kn - 2^{k-1} - 2^{k-2} - \ldots - 2 - 1 \\
&= 2^k C_2(1) + kn - 2^k + 1 \\
&= kn - 2^k + 1 \\
&= n\log n - n + 1
\end{aligned}
$$

因此我们有：

引理 2.

算法 MergeSort 对大小为 n 的数组排序，执行元素比较的总次数介于 $\frac{n \log n}{2}$ 到 $n \log n - n + 1$ 之间。

更为严谨的叙述

事实上，对于任意情况的 n 实际递推式应该是：

$$C(n) = \begin{cases} 0 & n = 1 \\ C(\lfloor \frac{n}{2} \rfloor) + C(\lceil \frac{n}{2} \rceil) + m & n > 1,\ \frac{n}{2} \leqslant m \leqslant n - 1 \end{cases}$$

但我们可以严格的证明，此时 $C(n)$ 的界依旧是 $\Theta(n \log n)$. 因此一般情况下，我们会适当忽略取整的影响。

$n \log n$ Versus n^2

我们再用一个例子来比较一下 $O(n \log n)$ 和 $O(n^2)$ 的差距。

- 家用笔记本电脑的 CPU 每秒可以执行 10^8 次操作。
- 超级计算机的 CPU 每秒可以执行 10^{12} 次操作。

计算机类型	插入排序 (n^2)			归并排序 ($n \log n$)		
	千级别	百万级别	十亿级别	千级别	百万级别	十亿级别
家用笔记本电脑	立即	2.8 小时	317 年	立即	1 秒	18 分钟
超级计算机	立即	1 秒	1 个礼拜	立即	立即	立即

Good algorithms are better than supercomputers!

归并排序的性质

- 归并排序是稳定的么？
  - *Yes!*
- 归并排序是 in-place 的么？
  - No! 合并需要 $O(n)$ 的额外空间，同时递归调用也有 $O(n)$ 的空间开销。

自底向上的归并排序

上海师范大学 Shanghai Normal University

我们考虑另一种归并排序的方法。依旧考虑下列数组:

9 4 5 2 1 7 4 6

9 4　　5 2　　1 7　　4 6

4 9　　2 5　　1 7　　4 6

2 4 5 9　　1 4 6 7

1 2 4 4 5 6 7 9

自底向上的归并排序算法

上海师范大学 Shanghai Normal University

算法： $\mathrm{BottomUpMergeSort}(A[1,\ldots,n])$

输入： n 元数组 $A[1,\ldots,n]$

输出： 排好序的 $A[1,\ldots,n]$

```
1: t ← 1
2: while t < n do
3:     s ← t, t ← 2s, i ← 0
4:     while i + t ⩽ n do
5:         MERGE(A, i + 1, i + s, i + t)
6:         i ← i + t
7:     if i + s < n then
8:         MERGE(A, i + 1, i + s, n)
```

自底向上 Versus 自顶向下?

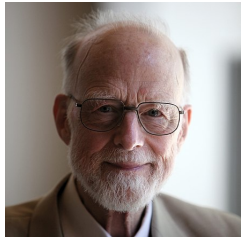
- MergeSort 和 BottomUpMergeSort 哪个更快?
  - 通过分析可知，两个算法需要的比较次数和所需的空间几乎是一样的!

自底向上 Versus 自顶向下?

1. 你觉得哪个算法更好理解?
   - MergeSort 还是 BottomUpMergeSort?
2. 你更喜欢哪个算法?

快速排序 (QuickSort)

- 快速排序入选 20 世纪最伟大的 10 个算法。
  - 一个知乎上的介绍：https://zhuanlan.zhihu.com/p/340354313
- 算法设计者：Sir Charles Antony Richard Hoare
  - 1980 年图灵奖获得者。
  - 其他贡献如：霍尔逻辑、通信顺序进程 (CSP) 等。



寻找到 x 的正确位置-划分算法 Partition

考虑如下一个数组:

$$2 \quad 13 \quad 43 \quad 45 \quad 89 \quad 23 \quad 67 \quad 88 \quad 90 \quad 34 \quad 56 \quad 78$$

随便取其中一个值，比如令 $x = 34$，现在其在数列中第 10 个位置，不是它所在的正确位置。正确位置如下所示:

$$2 \quad 13 \quad 23 \quad 34 \quad 43 \quad 45 \quad 56 \quad 67 \quad 78 \quad 88 \quad 89 \quad 90$$

定义 3.

我们称 x 在数组 A 中处在正确的位置，如果其左面的元素都小于等于 x，右面的元素都大于等于 x。

划分算法 Partition

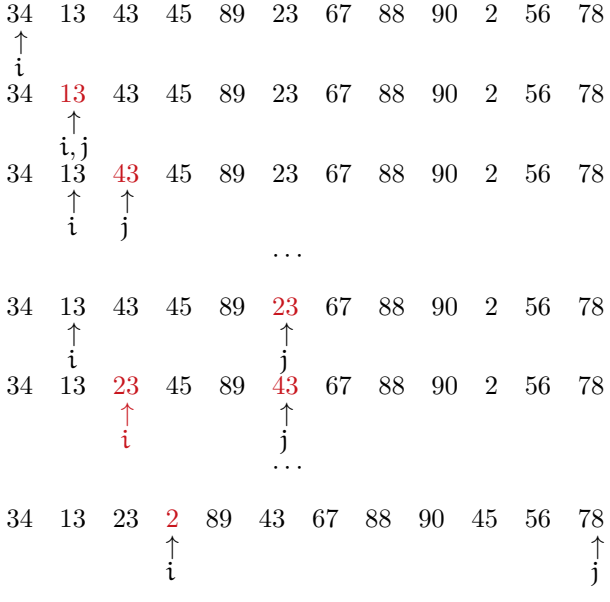
算法： $\mathrm{Partition}(A[low, \ldots, high])$

输入： 数组 $A[low, \ldots, high]$

输出： 将 $A[low]$ 放到正确位置后的数组 A 和 x 的正确位置 w

```
1: i ← low
2: x ← A[low]
3: for j ← low + 1 to high do
4:     if A[j] ⩽ x then
5:         i ← i + 1
6:         if i ≠ j then exchange A[i] and A[j]
7: exchange A[low] and A[i]
8: w ← i return A, w
```

Partition 例子



Partition 的性质

正确性

在运行算法 Partition 之后 x 会处在正确的位置上。

复杂性

算法 Partition 需要比较的次数恰好是 $n-1$，所需要额外的空间是 $O(1)$.

主元 pivot

我们将 Partition 的 x 称为主元 (pivot), 可以看到 Partition 实际上就是基于主元对数组进行了一个划分。

快速排序算法

上海师范大学 Shanghai Normal University

算法： $\mathrm{QuickSort}(A[1,\ldots,n])$

输入： 数组 $A[1,\ldots,n]$

输出： 排好序的 $A[1,\ldots,n]$

1: $\mathrm{quicksort}(A,1,n)$

过程： $\mathrm{quicksort}(A,low,high)$

2: **if** $low < high$ **then**

3: $\quad (A,w) \leftarrow \mathrm{Partition}(A,low,high)$

4: $\quad \mathrm{quicksort}(A,low,w-1)$

5: $\quad \mathrm{quicksort}(A,w+1,high)$

QuickSort 时间复杂性分析

- 最坏情况：每次划分都只划分成 1 个元素和剩余的 $n-1$ 个元素。
  - $T(n) = (n-1) + (n-2) + \cdots + 1 = \Theta(n^2)$
- 最好情况：每次划分都选中了中位数。
  - $T(n) = 2T(\frac{n}{2}) + (n-1) = \Theta(n \log n)$.
- 平均情况：等概率的计算然后加权。
  - $T(n) = n - 1 + (\frac{T(0)+T(n-1)}{n}) + (\frac{T(1)+T(n-2)}{n}) + \cdots + (\frac{T(n-1)+T(0)}{n})$.
  - $T(n) \sim 2(n+1)\ln N \simeq\approx n \log n$

QuickSort 的性质

稳定性

- QuickSort 是不稳定的。
  - **反例：** $A = [3,\ 5,\ 5,\ 2]$.

in-place?

- QuickSort 是 in-place 的。

QuickSort 和 MergeSort 的比较

通过我们的计算，可以发现快排的复杂性算出来要比归并排序更慢一些，但是实际中很多情况下快排的效率要比归并排序高很多，这是为什么？

- 两者都有一个常数的处理时间。而在实际处理当中，由于归并需要辅助数组来进行合并，因此当对数组进行排序时，归并排序需要额外的空间，而 QuickSort 则是 in-place 的，导致实际使用起来快速排序更有效率。
- 在处理链表的时候归并排序会有较好的表现。

QuickSort 的效率

我们继续用上述的例子来展现下 QuickSort 的效率。

- 家用笔记本电脑的 CPU 每秒可以执行 10^8 次操作。
- 超级计算机的 CPU 每秒可以执行 10^{12} 次操作。

计算机类型	归并排序 ($n \log n$)			快速排序 ($n \log n$)		
	千级别	百万级别	十亿级别	千级别	百万级别	十亿级别
家用笔记本电脑	立即	1 秒	18 分钟	立即	0.6 秒	12 分钟
超级计算机	立即	立即	立即	立即	立即	立即

Good algorithms are better than supercomputers!

Great algorithms are better than good algorithms!

Partition 主元的选择

前面提到过，快速排序的效率和主元的选择有关，如何选择主元避免极端情况的出现是提升快速排序效率的关键。

- 随机选择。RandomizedQuickSort.
- 平衡快排。BalancedQuickSort.
- 三路快排。3-wayQuickSort.

Dijstra 3-wayQuickSort

3 路快排对于有大量重复元素的数组排序效率会显得非常高，其核心思想是将数组划分成三部分，分别是小于主元、等于主元和大于主元的部分。

Dijstra 三路划分

- 令 $x = A[low]$ 为主元, 维护 3 个变量 lo, i, hi, 其中 i 是遍历下标，lo, hi 初始在 $low, high$ 的位置上。
- 从左到右遍历 $A[i]$ 直至 $i > hi$:
  1. 如果 $A[i] < x$，交换 $A[lt]$ 和 $A[i]$，lt 和 i 都加 1.
  2. 如果 $A[i] > x$，交换 $A[gt]$ 和 $A[i]$，gt 减 1.
  3. 如果 $A[i] = x$，i 加 1.

排序算法总结

排序算法	平均时间	最坏时间	最好时间	稳定性	in-place?
SelectSort	$\frac{n^2}{2}$	$\frac{n^2}{2}$	$\frac{n^2}{2}$	$\times$	✓
InsertSort	$\frac{n^2}{4}$	$\frac{n^2}{2}$	n	✓	✓
ShellSort	?	?	n	$\times$	✓
RadixSort	$O(nk)$	$O(nk)$	$O(nk)$	✓	$\times$
MergeSort	$n \log n$	$n \log n$	$n \log n$	checkmark	$\times$
QuickSort	$2n \ln n$	$\frac{n^2}{2}$	$n \log n$	$\times$	✓
3-wayQuickSort	$2n \ln n$	$\frac{n^2}{2}$	n	$\times$	✓

分治思想与主定理

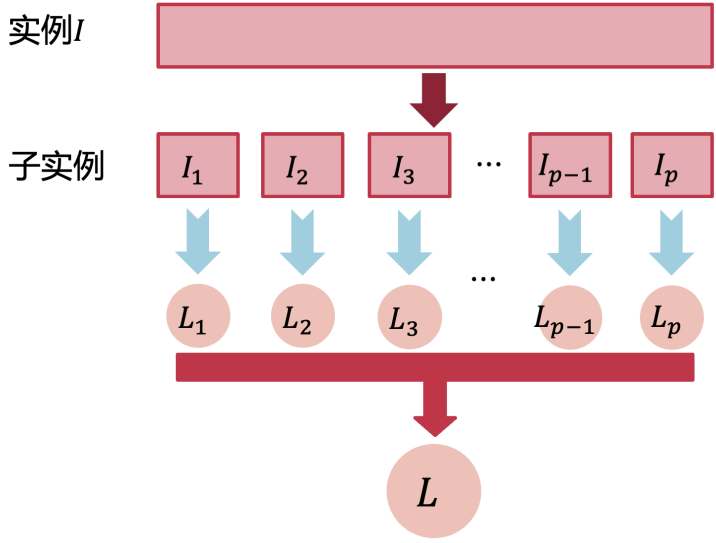
分治思想框架

无论是排序还是归并，都可以用下述的分治思想框架来描述：

分治算法框架

1. **划分 (Divide Step)**：将原问题分解为若干个规模较小，相互独立，与原问题形式相同的子问题。
2. **治理 (Conquer Step)**：递归的求解各个子问题。若子问题规模足够小，则直接求解。
3. **合并 (Combine Step)**：将各个子问题的解合并为原问题的解。

分治思想框架-图示



主定理

假设在一个分治算法中，我们将其分解成了 a 个规模为 $\frac{n}{b}$ 的子问题，递归求解后再花 $O(n^d)$ 的时间将解合并起来，则该算法的时间复杂性可通过下列递推式计算而来：

$$T(n) = \begin{cases} O(1) & n = 1 \\ aT(\frac{n}{b}) + O(n^d) & n > 1 \end{cases}$$

怎么快速计算上述递推式的结果？

定理 4 **[主定理 (Master Theorem)].**

上述递推式满足：

$$T(n) = \begin{cases} O(n^d) & a < b^d \\ O(n^d \log n) & a = b^d \\ O(n^{\log_b a}) & a > b^d \end{cases}$$

主定理的例子

例 5.

1. 在归并排序中，我们有：

$$C(n) = 2C(\frac{n}{2}) + O(n)$$

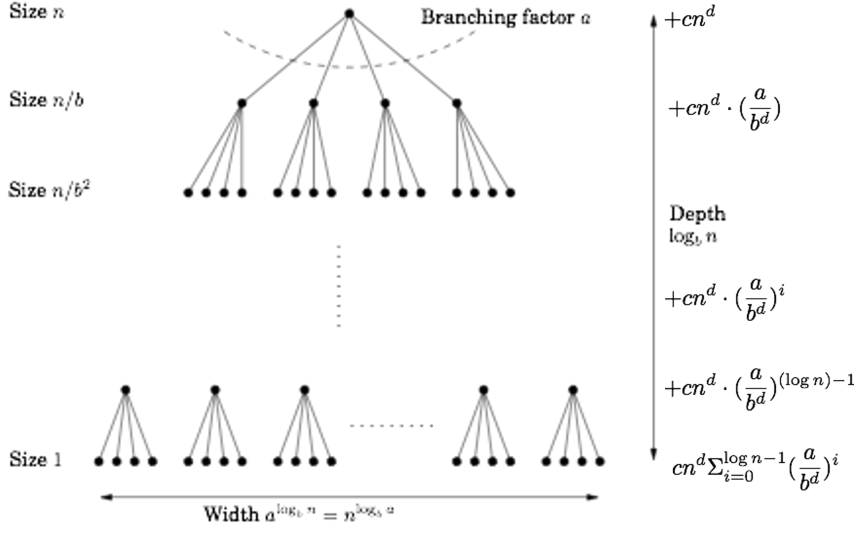
其中 $a = 2, b = 2, d = 1, a = b^d$，因此 $C(n) = O(n \log n)$.

2. 在二分搜索中，我们有：

$$C(n) = C(\frac{n}{2}) + O(n)$$

其中 $a = 1, b = 2, d = 1, a < b^d$，因此 $C(n) = O(n)$.

主定理的证明 (I)


Size n
Branching factor a
Size n/b
Size n/b^2
Size 1
Width $a^{\log_b n} = n^{\log_b a}$
$+cn^d$
$+cn^d \cdot (\frac{a}{b^d})$
Depth
$\log_b n$
$+cn^d \cdot (\frac{a}{b^d})^i$
$+cn^d \cdot (\frac{a}{b^d})^{(\log n)-1}$
$cn^d \Sigma_{i=0}^{\log n-1} (\frac{a}{b^d})^i$

主定理的证明 (II)

考察 $\frac{a}{b^d}$ 的值:

- $\frac{a}{b^d} < 1$, 则级数是递减的，因此其和可以仅由第一项来表示，即 $T(n) = O(n^d)$.
- $\frac{a}{b^d} = 1$, 则所有的 $O(\log n)$ 项均等于 $O(n^d)$，即 $T(n) = O(n^d \log n)$.
- $\frac{a}{b^d} > 1$, 则级数是递增的，因此其和可以用最后一项来表示，即

$$T(n) = cn^d(\frac{a}{b^d})^{\log_b n} = cn^{\log_b a}.$$

主定理的运用

计算下列递推式的结果

1. $f(n) = f(\frac{n}{2}) + 1$.
2. $f(n) = 2f(\frac{n}{2}) + n^2$.
3. $f(n) = 9f(\frac{n}{3}) + n$.
4. $f(n) = 5f(\frac{n}{2}) + n^3$.

1. $f(n) = O(\log n)$.
2. $f(n) = O(n^2)$.
3. $f(n) = O(n^2)$.
4. $f(n) = O(n^3)$.

利用分治思想设计算法

寻找最大最小元素 MinMax

上海师范大学 Shanghai Normal University

问题 6 **[寻找最大值和最小值元素].**

给定一个数组，如何寻找其中的最大值和最小值?

暴力枚举算法: $\mathrm{MinMax}_{\mathrm{direct}}(A[1,\ldots,n])$

输入: 数组 $A[1,\ldots,n]$

输出: 数组 A 中的最大值和最小值

1: $x \leftarrow A[1],\ y \leftarrow A[1]$
2: **for** $i \leftarrow 2$ to n **do**
3: **if** $A[i] < x$ **then** $x \leftarrow A[i]$
4: **if** $A[i] > y$ **then** $y \leftarrow A[i]$
5: **return** (x, y)

一共需要$2n-2$次比较次数!

利用分治的 MinMax 算法

上海师范大学 Shanghai Normal University

算法：$MinMax(A[1,\ldots,n])$

输入: 数组 $A[1,\ldots,n]$

输出: 数组 A 中的最大值和最小值

1: $minmax(1,n)$

过程: $minmax(low,high)$

2: **if** $high-low=1$ **then**

3: **if** $A[low]<A[high]$ **then return** $(A[low],A[high])$.

4: **else return** $(A[high],A[low])$.

5: $mid \leftarrow \lfloor \frac{low+high}{2} \rfloor$

6: $(x_1,y_1) \leftarrow minmax(low,mid)$

7: $(x_2,y_2) \leftarrow minmax(mid+1,high)$

8: **return** $(\min(x_1,x_2),\max(y_1,y_2))$

MinMax 的比较次数

令算法 MinMax 对于 n 个数组所需要的比较次数为 $T(n)$，则 $T(n)$ 满足:

$$T(n)=\begin{cases}1 & n\leqslant 2\\ 2T(\frac{n}{2})+2 & n>2\end{cases}$$

假设 $n=2^k$，则计算 $T(n)$ 可得:

$$T(n)=2T(\frac{n}{2})+2=\ldots=2^{k-1}T(2)+2^{k-1}+\ldots+2=\frac{n}{2}+2^k-2=\frac{3n}{2}-2.$$

现在我们讨论一个基本的运算操作，乘法。

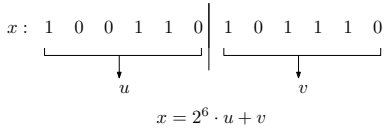
问题 7 **[大整数乘法].**

给定两个 n 位长的整数，求这两个整相乘之后的积。

$$
\begin{array}{cccccccc}
 & & & & 1 & 1 & 0 & 1 \\
 & & & \times & 1 & 0 & 1 & 1 \\
\hline
 & & & & 1 & 1 & 0 & 1 \\
 & & & 1 & 1 & 0 & 1 & \\
 & & 0 & 0 & 0 & 0 & & \\
 & 1 & 1 & 0 & 1 & & & \\
\hline
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1
\end{array}
$$

一共需要 $O(n^2)$ 次操作。

大整数乘法-分治 (I)



考虑 2 个 n 位长的整数，我们可以将其分解成两个 $n/2$ 位长的整数，即如上所示:

$$x = 2^{\frac{n}{2}} \cdot u_x + v_x, \quad y = 2^{\frac{n}{2}} \cdot u_y + v_y.$$

则我们有:

$$x \cdot y = (2^{\frac{n}{2}} \cdot u_x + v_x) \cdot (2^{\frac{n}{2}} \cdot u_y + v_y) = 2^n \cdot (u_x \cdot u_y) + 2^{\frac{n}{2}} \cdot ((u_x \cdot v_y) + (v_x \cdot u_y)) + (v_x \cdot v_y).$$

如果使用上面的方式，即把原来的乘法变成 4 个长度为 $n/2$ 的整数的乘法，那么整个算法的复杂性满足 $T(n) = 4T(\frac{n}{2}) + O(n)$，即依旧只有 $T(n) = O(n^2)$.

大整数乘法-分治 (II)

我们注意到：

$$u_x \cdot v_y + v_x \cdot u_y = (u_x + v_x) \cdot (u_y + v_y) - u_x \cdot u_y - v_x \cdot v_y.$$

因此我们只需要计算 3 个大小为 $\frac{n}{2}$ 的乘法，便可以通过下述等式：

$$x \cdot y = 2^n (u_x \cdot u_y) + 2^{\frac{n}{2}} ((u_x + v_x) \cdot (u_y + v_y) - u_x \cdot u_y - v_x \cdot v_y) + (v_x \cdot v_y). \quad (1)$$

计算出 $x \cdot y$ 的值。这样的算法时间复杂性满足：

$$T(n) = 3T(\frac{n}{2}) + O(n) = O(n^{\log 3}).$$

$O(n^{\log 3})$ 的算法来计算乘法

我们现在可以回答最初提到的问题：$\mathrm{Fib}_3(n)$ 确实比 $\mathrm{Fib}_2(n)$ 更快!

矩阵乘法

现在我们来思考两个 $n \times n$ 的矩阵相乘:

$$\begin{bmatrix} c_{11} & \dots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \dots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & \dots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \dots & b_{nn} \end{bmatrix}$$

其中

$$c_{ij} = a_{i1}b_{1j} + \dots a_{in}b_{nj}$$

直接计算矩阵乘法

如果直接计算两个矩阵的乘积，我们需要$O(n^3)$次操作。

矩阵乘法-分块算法

依旧用类似乘法的思想，我们可以将矩阵分解成四个 $\frac{n}{2} \times \frac{n}{2}$ 的矩阵进行计算：

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

其中 C_{ij} 满足：

- $C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$.
- $C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$.
- $C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$.
- $C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}$.

分块算法

矩阵分块算法需要对 8 个 $\frac{n}{2} \times \frac{n}{2}$ 的矩阵进行计算，因此其时间复杂性满足

$$T(n) = 8T(\frac{n}{2}) + O(n^2) = O(n^3).$$

矩阵乘法-Strassen 算法 (I)

我们来介绍 Strassen 算法，其只用 7 个小矩阵的乘积来计算 $A \times B$. 计算如下的矩阵乘积:

1. $D_1 = (A_{11} + A_{22})(B_{11} + B_{22})$.
2. $D_2 = (A_{21} + A_{22})B_{11}$.
3. $D_3 = A_{11}(B_{12} - B_{22})$.
4. $D_4 = A_{22}(B_{21} - B_{11})$.
5. $D_5 = (A_{11} + A_{12})B_{22}$.
6. $D_6 = (A_{21} - A_{11})(B_{11} + B_{12})$.
7. $D_7 = (A_{12} - A_{22})(B_{21} + B_{22})$.

从而:

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} D_1 + D_4 - D_5 + D_7 & D_3 + D_5 \\ D_2 + D_4 & D_1 + D_3 - D_2 + D_6 \end{bmatrix}$$

矩阵乘法-Strassen 算法 (II)

上海师范大学 Shanghai Normal University

Strassen 算法时间复杂性

用 Strassen 算法计算矩阵乘积的时间复杂性满足:

$$T(n) = 7T(\frac{n}{2}) + O(n^2) = O(n^{\log_2 7}) \approx O(n^{2.81}).$$

最快的矩阵算法

人们目前还不知道矩阵的乘法能否只用 $n^{2+o(1)}$ 时间, 目前最好的下界是 $O(n^2 \log n)$, 而最快的算法目前则需要 $O(n^{2.371552})$.

参考资料:

- 下界: On the complexity of matrix product
- 上界: New Bounds for Matrix Multiplication: from Alpha to Omega

矩阵乘法-Strassen 算法 (II)

上界算法的历史:

Timeline of matrix multiplication exponent

Year	Bound on omega	Authors
1969	2.8074	Strassen[1]
1978	2.796	Pan[11]
1979	2.780	Bini, Capovani [it], Romani[12]
1981	2.522	Schönhage[13]
1981	2.517	Romani[14]
1981	2.496	Coppersmith, Winograd[15]
1986	2.479	Strassen[16]
1990	2.3755	Coppersmith, Winograd[17]
2010	2.3737	Stothers[18]
2013	2.3729	Williams[19][20]
2014	2.3728639	Le Gall[21]
2020	2.3728596	Alman, Williams[6][22]
2022	2.371866	Duan, Wu, Zhou[3]
2023	2.371552	Williams, Xu, Xu, and Zhou[2]

选择第 k 小的元素、中位数

我们再回到数组上的元素寻找问题。

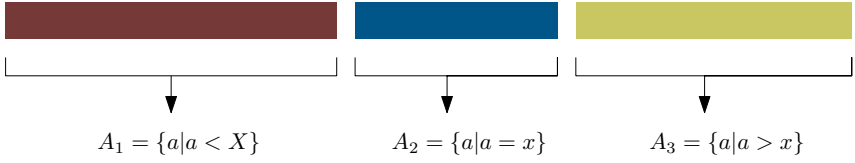
问题 8 **[寻找第 k 小的元素].**

给定一个数组 $A[1,\ldots,n]$ 和整数 k, 请给出一个算法找出数列中第 k 小的元素

- 如果令 k 为 $\lfloor \frac{n}{2} \rfloor$，则就是寻找一个数组的中位数问题。
- 如果直接利用排序再寻找第 k 小的元素，其时间复杂性至少为 $O(n \log n)$.
- 问题是排序其实知道了所有的顺序，因此有没有可能，用 $O(n)$ 的时间就找到第 k 小的元素?

划分算法 Partition 回顾

Partition 算法可以将一个元素放至准确的位置，同时也将其划分成了有规律的三部分：



- 利用 Partition，我们可以确定所要找的第 k 小的元素是在 A_1，A_2，A_3 中的一个集合。
- 如何保证每次 A_1，A_3 都有比较大的规模？
  - 随机化的选择。-可以证明期望是 $O(n)$ 的。
  - 确定性的主元选择方法，保证 A_1，A_3 都有 $\Omega(n)$ 的大小。

合适主元的寻找

考虑如下的数组，我们将其写成一张表：

3	4	328	183	92	95	9	25
2	2	74	38	29	27	727	45
4	84	83	17	451	7	41	79
99	84	45	37	5	7	1	26
92	21	75	38	3	91	42	

合适主元的寻找

考虑如下的数组，我们将其写成一张表：

2	4	328	183	92	95	9	25
3	2	74	38	29	27	727	45
4	84	83	17	451	7	41	79
92	84	45	37	5	7	1	26
99	21	75	38	3	91	42	

合适主元的寻找

考虑如下的数组，我们将其写成一张表：

2	2	328	183	92	95	9	25
3	4	74	38	29	27	727	45
4	21	83	17	451	7	41	79
92	84	45	37	5	7	1	26
99	84	75	38	3	91	42	

合适主元的寻找

考虑如下的数组，我们将其写成一张表：

2	2	45	183	92	95	9	25
3	4	74	38	29	27	727	45
4	21	75	17	451	7	41	79
92	84	83	37	5	7	1	26
99	84	328	38	3	91	42	

合适主元的寻找

考虑如下的数组，我们将其写成一张表：

2	2	45	17	92	95	9	25
3	4	74	37	29	27	727	45
4	21	75	38	451	7	41	79
92	84	83	38	5	7	1	26
99	84	328	183	3	91	42	

合适主元的寻找

考虑如下的数组，我们将其写成一张表：

2	2	45	17	3	95	9	25
3	4	74	37	5	27	727	45
4	21	75	38	29	7	41	79
92	84	83	38	92	7	1	26
99	84	328	183	451	91	42	

合适主元的寻找

考虑如下的数组，我们将其写成一张表：

2	2	45	17	3	7	9	25
3	4	74	37	5	7	727	45
4	21	75	38	29	27	41	79
92	84	83	38	92	91	1	26
99	84	328	183	451	95	42	

合适主元的寻找

考虑如下的数组，我们将其写成一张表：

2	2	45	17	3	7	1	25
3	4	74	37	5	7	9	45
4	21	75	38	29	27	41	79
92	84	83	38	92	91	42	26
99	84	328	183	451	95	727	

合适主元的寻找

考虑如下的数组，我们将其写成一张表：

2	2	45	17	3	7	1	25
3	4	74	37	5	7	9	26
4	21	75	38	29	27	41	45
92	84	83	38	92	91	42	79
99	84	328	183	451	95	727	

合适主元的寻找

考虑如下的数组，我们将其写成一张表：

2	2	45	17	3	7	1	25
3	4	74	37	5	7	9	26
4	21	75	38	29	27	41	78
92	84	83	38	92	91	42	79
99	84	328	183	451	95	727	

合适主元的寻找

考虑如下的数组，我们将其写成一张表：

2	2	7	3	17	7	45	25
3	4	7	5	37	7	74	26
4	21	27	29	38	41	75	78
92	84	91	92	38	42	83	79
99	84	95	451	183	727	328	

合适主元的寻找

考虑如下的数组，我们将其写成一张表：

2	2	7	3	17	7	45	25
3	4	7	5	37	7	74	26
4	21	27	29	38	41	75	78
92	84	91	92	38	42	83	79
99	84	95	451	183	727	328	

- 在元素 29 的左上方，所有的元素都不会大于 29.
- 在元素 29 的右下方，所有的元素都不会小于 29.

寻找第 k 小的元素-Select 算法

上海师范大学 Shanghai Normal University

算法: $Select(A[1,\ldots,n],k)$

输入: 数组 $A[1,\ldots,n]$ 和整数 k.

输出: A 中第 k 小的元素

1: $select(A,1,n,k)$

过程: $select(A,low,high,k)$

2: $p \leftarrow high - low + 1$

3: **if** $p \leqslant 44$ **then sort** A and **return** $A[k]$.

4: $q \leftarrow \lceil p/5 \rceil$ Divide A into $\lceil p/5 \rceil$ groups of 5 elements each. If there are any groups of size < 5, leave them out.

5: Sort each of q groups individually and extract its median. Let the set of medians be M.

6: $mm \leftarrow select(M,1,q,\lceil \frac{q}{2} \rceil)$

7: Partition $A[low,\ldots,high]$ into three arrays:

$$A_1 = \{a | a < mm\},\ A_2 = \{a | a = mm\},\ A_3 = \{a | a > mm\}$$

8: **CASE:**

$\lvert A_1\rvert \geqslant k$	**return** $select(A_1,1,\lvert A_1\rvert,k)$
$\lvert A_1\rvert + \lvert A_2\rvert \geqslant k$	**return** mm
$\lvert A_1\rvert + \lvert A_2\rvert < k$	**return** $Select(A_3,1,k-\lvert A_1\rvert-\lvert A_2\rvert)$

Select 算法时间分析

- 令 B_1 表示 A 中大于等于元素 mm 的集合，显然 $|A_1| + |B_1| = n$，并且：

$$|B_1| \geqslant 3 \cdot \lceil \frac{1}{2} \cdot \lfloor \frac{n}{5} \rfloor \rceil \geqslant \frac{3}{2} \cdot (\frac{n-4}{5}) = \frac{3n-12}{10}$$

  从而 $n > 44$ 时 $|A_1| \leqslant \frac{7n+12}{10} \leqslant \lfloor \frac{3n}{4} \rfloor$.

- 同理可得 $n > 44$ 时 $|A_3| \leqslant \frac{7n+12}{10} \leqslant \lfloor \frac{3n}{4} \rfloor$.
- 整个算法的运行时间 $T(n)$ 满足：

$$T(n) = \begin{cases} O(1) & n < 44 \\ T(\lfloor \frac{n}{5} \rfloor) + T(\lceil \frac{3n}{4} \rceil) + cn & n \geqslant 44 \end{cases}$$

- 注意到 $\frac{1}{5} + \frac{3}{4} < 1$，我们有$T(n) = O(n)$!

关于 Select 算法

- 尽管 Select 是一个确定的最坏情况也是 $O(n)$ 的算法，但实际中我们还是用随机版本的主元选择，为什么？
  - 过大的常数导致 Select 的运行时间偏慢。
- Select 选择 5 个一组，是否还有其他的选择？
  - 任何一个 $\geqslant 5$ 的奇数都可以。
  - 选择的分组越大，算法越慢。

总结

本节内容

- 归并排序和快速排序
- 分治算法的设计框架、主定理
- 利用分治思想来设计算法
  - 寻求最大最小值 MinMax
  - 大整数乘法 Multiplication
  - 矩阵乘法 Strassen
  - 寻找第 k 小的元素 Select

第三周作业

本周作业和编程作业都已经发布到课程主页上。

截至时间：2023 年 10 月 8 日周日晚上 11:59。