《算法设计与分析》

2-归纳法 (Induction)

杨启哲

上海师范大学信机学院计算机系

2024 年 9 月 15 日

主要内容

- 归纳法
- 排序方法回顾

基于递归的算法设计思想

- 归纳法。
  - 迭代 (Iteration)，或者尾递归 (Tail recursion)
  - 容易给出简单的归纳证明
- 分治法 (Divide and Conquer)。
  - 子问题互相不会重叠。
- 动态规划 (Dynamic Programming)。
  - 子问题存在重叠。

归纳法

选择排序 (Selection Sort)

选择排序每次从未排序的部分中选择最小的元素，将其放到已排好数据中的最后面。

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

输入： n 元数组 $A[1,\ldots,n]$

输出： 非降序排列好的数组 $A[1,\ldots,n]$

```
Sort(1)
过程: Sort(i)
if i < n then
    k ← i
    for j ← i + 1 to n do
        if A[j] < A[k] then k ← j
    if k ≠ i then exchange A[i] and A[k]
    Sort(i + 1)
```


选择排序

假设选择排序运用在如下的数组上：

$$9, 8, 9, 6, 2, 56$$

经过第一次 Sort 过程后，数组变为：

$$2, 8, 9, 6, 9, 56$$

此时问题的规模从一开始的 6 变成了 5，即每执行一次 Sort 过程，问题的规模都会减少 1。

从而我们可以通过数学归纳法给出正确性证明。

引理 1.
第 i 次调用 Sort 后可以将第 i 小的元素放到 $A[i]$ 上。

SelectSort 算法时间分析

令 $C(n)$ 表示输入 n 个元素时，算法内元素比较的次数。

$C(n)$ 满足：

$$C(n) = \begin{cases} 0 & 若\ n = 1 \\ C(n-1) + (n-1) & 若\ n \geqslant 2 \end{cases}$$

不难算出，上式的解为：

$$C(n) = \sum_{j=1}^{n-1} j = \frac{n(n-1)}{2}$$

从而 SelectSort 的时间复杂性为 $\Theta(n^2)$.

寻找多数元素

问题 2 **[寻找多数元素].**
令 $A[1,\ldots,n]$ 是一个整数序列，如果存在一个元素 a 在 A 中出现了超过 $\lfloor \frac{n}{2} \rfloor$ 次，则称 a 是 A 的多数元素。如何设计一个算法来找出 A 的多数元素？

例 3.
考察如下两个序列：

$$1\ 2\ 3\ 3\ 4\ 2\ 2\ 2\ 3\ 2$$
$$1\ 5\ 1\ 1\ 4\ 2\ 1\ 3\ 1\ 1$$

第一个数列没有多数元素，第二个数列则有多数元素，为 1。

寻找多数元素-想法

- 逐一比较每个元素，统计出现次数。
  - 这样的算法复杂性至少是 $O(n^2)$.
- 先排序，再计算统计次数。
  - 这样的算法复杂性至少是 $O(n\log n)$.

有没有可能给出 $O(n)$ 时间的算法?

引理 4.

如果一个数组去除了两个不同的元素，原来数组里的多数元素依旧是新数组的多数元素。

寻找多数元素

算法： $\mathrm{Majority}(A[1,\dots N])$

输入: n 元数组 $A[1,\dots,n]$

输出: 若存在多数元素，则输出多数元素，否则输出 $none$

```
c ← candidate(1), count ← 0
for j ← 1 to n do
    if A[j] = c then count ← count + 1
if count > ⌊n/2⌋ then return c
else return none
```

过程: $candidate(m)$

```
j ← m, c ← A[m], count ← 1
while j < n and count > 0 do
    j ← j + 1
    if A[j] = c then count ← count + 1
    else count ← count − 1
if j = n then return c
else return candidate(j + 1)
```


正确性分析

引理 5.

如果数组 $A[i,\dots n]$ 存在多数元素，则 $candidate(i)$ 返回该多数元素。

证明. 对 i 作归纳法（对数组 A 的元素个数）

BASE : $i = N$ 时显然成立。

INDUCTION : 假设命题对 $j > i$ 成立，则令其多数元素为 a，有如下两种情况:

- $A[i] = a$，则要么 $candidate(i)$ 返回 a，要么 $candidate(i)$ 会调用某个 $candidate(j)$，其中 $j > i$。注意到由引理 4可知，$A[j,\dots n]$ 的多数元素依旧是 a，因此 $candidate(j)$ 返回 a。
- $A[i] \neq a$, 则 $candidate(i)$ 会调用某个 $candidate(j)$，其中 $j > i$。注意到由引理 4可知，$A[j,\dots n]$ 的多数元素依旧是 a，因此由归纳假设 $candidate(j)$ 返回 a。

从而命题对 i 也成立，得证。 □

多项式求值

问题 6 **[多项式求值].**

给定一个多项式 $P(x) = a_0 + a_1x + \dots + a_nx^n$，以及 x 的一个值，计算 $P(x)$ 的值。

暴力求解

如果直接对每一项分别求值，则一共需要:

$$1 + 2 + \dots + n = \frac{n(n+1)}{2}$$

次乘法和 n 次加法，这是一个非常**低效**的算法。

Horner 规则

通过归纳法，我们可以得出一种更高效的算法。

Horner 规则

多项式 $P(x) = a_0 + a_1x + \ldots + a_nx^n$ 可以改写成如下形式：

$$P(x) = a_0 + a_1x + \ldots + a_nx^n = a_0 + x(a_1 + x(a_2 \ldots + a_{n-1} + a_nx) \ldots))$$

例 7.

$$p(n) = n^3 + 3n^2 + 2n + 1 = n^2(n+3) + 2n + 1 = n(n(n+3)+2) + 1$$

不难发现，通过 Horner 规则，计算一个 n 次的多项式，我们只需要进行 n 次乘法和 n 次加法操作即可。

多项式求值算法

算法： $\text{Horner}(a_0, a_1, \ldots, a_n, x)$

输入： $n+2$ 个实数 $a_0, a_1, \ldots, a_n, x$

输出： $P(x) = a_nx^n + a_{n-1}x^{n-1} + \ldots + a_1x + a_0$

```
p ← a_n
for j ← 1 to n do
    p ← xp + a_{n-j}
return p
```

利用归纳的方法，我们得到了一个比直接运算要快一个数量级的算法。

生成排列

现在考察下面这样一个问题：

问题 8.

给定一个正整数 n，设计一个算法，输出 $1, \ldots, n$ 的所有排列。

例 9.

对于数组 $P = [1, 2, 3]$，其所有排列为：

$$1, 2, 3; 1, 3, 2; 2, 1, 3;$$
$$2, 3, 1; 3, 1, 2; 3, 2, 1;$$

生成排列-第一种算法

假定我们已经有了生成 $n-1$ 个数的排列方法，则对于 $2, \ldots n$ 的任何一个排列 $l_1, \ldots, l_{n-1}$，下述排列是 $1, \ldots, n$ 的一个排列：

$$1,\ l_1,\ l_2, \ldots,\ l_{n-1}$$

并且所有以 1 为开头的排列都对应着某个 $2, \ldots, n$ 的不同排列。

这给了我们一个**通过归纳**获取所有排列的算法。

生成排列-第一种算法

算法：Permutations1(n)

输入: 正整数 n
输出: $1, \ldots n$ 的所有排列

```
for j ← 1 to n do
    P[j] ← j
perm1(1)
```

过程: perm1(m)

```
if m = n then 输出：P[1,...n]
else
    for j ← m to n do
        交换 P[j] 和 P[m]
        perm1(m + 1)
        交换 P[j] 和 P[m]
```


Permutations1 算法分析

算法时间分析

- 算法第一步执行了 $n!$ 次，一次输出操作需要 n 时间，因此一共需要 $n \cdot n!$ 时间来进行输出。
- 循环执行次数满足：

$$f(n) = \begin{cases} 0 & \text{若 } n = 1 \\ nf(n-1) + n & \text{若 } n \geqslant 2 \end{cases}$$

- 令 $h(n) = \frac{f(n)}{n!}$，则我们有：

$$h(n) = h(n-1) + \frac{1}{(n-1)!} = \ldots = \sum_{j=1}^{n-1} \frac{1}{j!} < e - 1$$

从而 $f(n) = n!h(n) = \Theta(n!)$. 整个算法的时间复杂性为 $\Omega(n \cdot n!)$.

生成排列-第二种算法

我们还可以通过**另一种归纳**的方式给出所有的生成排列。

- 考虑一个排列，其初始有 n 个位置需要填充。
- 当我们填充掉其中一个位置以后，其只剩 $n-1$ 个位置，我们便可以用归纳的方式继续填充从而获取所有的排列。

生成排列-第二种算法

算法：Permutations2(n)

输入: 正整数 n
输出: $1, \ldots n$ 的所有排列

```
for j ← 1 to n do
    P[j] ← 0
perm2(n)
```

过程: perm2(m)

```
if m = 0 then 输出：P[1,...n]
else
    for j ← 1 to n do
        if P[j] = 0 then
            P[j] ← m
            perm2(m − 1)
            P[j] ← 0
```

Permutations2 算法分析

算法时间分析

- 算法第一步执行了 $n!$ 次，一次输出操作需要 n 时间，因此一共需要 $n \cdot n!$ 时间来进行输出。
- 循环执行次数满足：

$$f(m) = \begin{cases} 0 & 若\ m = 0 \\ mf(m-1) + n & 若\ m \geqslant 1 \end{cases}$$

这里需要注意，n 是常数，与 m 无关

- 令 $h(m) = \frac{f(m)}{m!}$，则我们有：

$$h(m) = h(m-1) + \frac{n}{m!} = \ldots = n \sum_{j=1}^{m-1} \frac{1}{j!} < (e-1)n$$

从而 $f(n) = \Theta(n \cdot n!)$. 整个算法的时间复杂性为 $\Omega(n \cdot n!)$.

基数排序 (Radix Sort)

上海师范大学 Shanghai Normal University

我们下面来介绍一个和一般排序算法并不一样的方式-基数排序。其核心在于比较相同位数上的数字。

例 10.

我们现在对 7467，1247，3275，6792，9187，9134，4675，1239 进行排列。

初始状态：	第一轮：	第二轮：	第三轮：	第四轮：
7467	6792	9134	9134	1239
1247	9134	1239	9187	1247
3275	3275	1247	1239	3275
6792	4675	7467	1247	4675
9187	7467	3275	3274	6792
9134	9187	4675	7467	7467
4675	1247	9187	4675	9134
1239	1239	6792	6792	9187

基数排序算法

上海师范大学 Shanghai Normal University

算法：$\mathrm{RadixSort}(L, k)$

输入： 一个有 n 个数的表 $L = \{a_1, \ldots a_n\}$ 和 k 位数字
输出： 按非降序排列的 L

```
for j ← 1 to k do
    Prerpare 10 empty lists L_0, L_1, ... L_9
    while L is not empty do
        Remove the first element a from L
        Append a to L_i where i is the jth digit of a
    L ← L_0
    for i ← 1 to 9 do
        L ← L, L_i
return L
```


RadixSort 算法分析

上海师范大学 Shanghai Normal University

算法的正确性可以通过归纳证明：

引理 11.

在算法 RadixSort 中，如果第 i 位数字已经排好序，则第 $i, i-1, \ldots, 1$ 位数字都已经排好序了。

不难得出，该算法的时间复杂性为 $\Theta(nk)$.

和**快速排序**以及**归并排序**相比，谁的算法更快？

- 比较型排序的下界为 $\Omega(n \log n)$.
- 基数排序的复杂度取决于最大的值和采取的进位制，即假设最大的值为 N 并且使用的进位制为 B，那么基数排序的复杂性为 $\log_B N \cdot n$.

排序方法回顾

插入排序

算法：InsertSort($A[1,\ldots,n]$)

输入: n 元数组 $A[1,\ldots,n]$
输出: 非降序排列好的数组 $A[1,\ldots,n]$

```
Sort(n)
过程: Sort(i)
if i > 1 then
    x ← A[i]
    Sort(i − 1)
    j ← i − 1
    while j > 0 and A[j] > x do
        A[j + 1] ← A[j]
        j ← j − 1
    A[j + 1] ← x
```


InsertSort 算法时间分析

最快运行时间

InsertSort 碰到输入为升序排列的数组时需要的比较次数最少。每次只需要对 $A[i]$ 和 $A[i-1]$ 进行比较，一共需要 $n-1$ 次比较。

最慢运行时间

InsertSort 碰到输入为降序排列的数组时需要的比较次数最多。每次都需要对 $A[i]$ 和 $A[i-1],\ldots A[1]$ 进行比较，因此此时一共需要 $1+2+\ldots+n-1=\frac{n(n-1)}{2}$ 次比较。

InsertSort 算法平均时间分析

引理 12.
算法 InsertSort 执行的平均比较次数是 $\Theta(n^2)$。

证明. 为了简化证明，不妨令数组 $A[1,\ldots,n]$ 恰为 $1,\ldots,n$ 的一个排列。

考察 $A[i]$ 插入 $A[1,\ldots,i]$ 的情形，，如果其确切位置是 j，则 $j\geqslant 2$ 时需要的比较次数为 $i-j+1$，$j=1$ 时需要的比较次数为 $i-j$，因此需要的平均比较次数为:

$$\frac{i-1}{i}+\sum_{j=2}^{i}\frac{i-j+1}{i}=\frac{i-1}{i}+\sum_{j=1}^{n-1}\frac{j}{i}=\frac{i}{2}-\frac{1}{i}+\frac{1}{2}$$

从而算法的平均比较次数为:

$$\sum_{i=2}^{n}\left(\frac{i}{2}-\frac{1}{i}+\frac{1}{2}\right)=\frac{n(n+1)}{4}-\frac{1}{2}-\sum_{i=2}^{n}\frac{1}{i}+\frac{n-1}{2}=\Theta(n^2)$$

□

希尔排序

希尔排序 (Shell sort)，又称缩小增量排序，1959 年由Donald Shell提出，是 InsertSort 的改进版本，也是第一个突破 $O(n^2)$ 的算法。

希尔排序示例

希尔排序的核心是步长 gap，我们以 $gap = \frac{n}{2^i}$ 的步长作介绍。

假设我们需要对下面 10 个元素进行排序：

$$7\ 6\ 9\ 3\ 1\ 5\ 2\ 4\ 0\ 8$$

第一次循环时 $gap = 5$, 则我们将上述数组分成五组：

$$7\ 9\ 1\ 2\ 0$$

$$6\ 3\ 5\ 4\ 8$$

即每一列为一组，对其每一组内进行插入排序，数组变为：

$$6\ 3\ 1\ 2\ 0\ 7\ 9\ 5\ 4\ 8$$

重复上述过程，在**最后一轮**的时候 $gap = 1$，即标准的**插入排序**。

希尔排序算法

- 希尔排序算法的核心在于 InsertSort 对于相对有序的数组会有很好的表现。
- 采用不同的步长会使得算法有不同的时间复杂性，一些经典的步长与对应时间可参看下表：

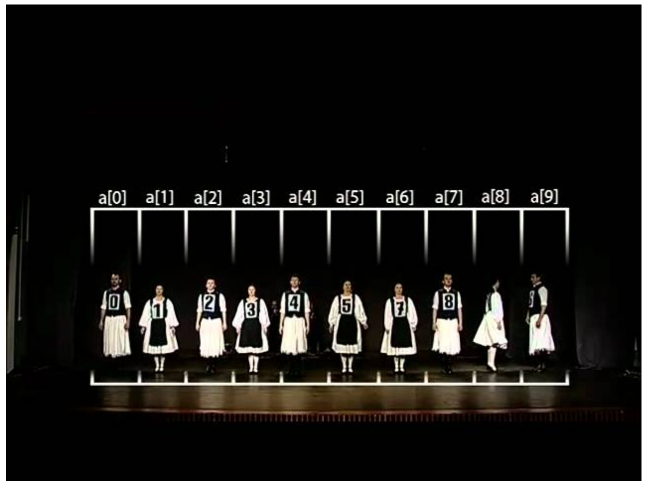
表: 希尔排序中使用的不同步长序列

步长	最坏时间时间复杂性
$\frac{n}{2^i}$	$O(n^2)$
$2^i - 1$	$O(n^{\frac{3}{2}})$
$2^p 3^q$	$O(n \log^2 n)$

- 尽管希尔排序达不到比较型排序算法的下界 $O(n \log n)$，其在中小规模中的优异表现使得其还是有着大量的应用。

排序甄别

考虑一下下述舞蹈，代表的是哪种排序？



排序甄别

考虑一下下述舞蹈，代表的是哪种排序？

排序甄别

考虑一下下述舞蹈，代表的是哪种排序？

排序算法的性质

定义 13 **[稳定性 (stability)].**
一个排序算法是稳定的，如果对于任意的两个相等的元素 a 和 b，如果 a 出现在 b 之前，那么排序后 a 仍然在 b 之前。

定义 14 **[in-place].**
一个排序算法是 in-place 的，如果其需要的额外的存储空间为 $O(\log n)$。

排序算法的比较

排序算法	平均时间复杂度	最坏时间复杂度	最好时间复杂度	稳定性	in-place?
SelectSort	$\frac{n^2}{2}$	$\frac{n^2}{2}$	$\frac{n^2}{2}$	×	✓
InsertSort	$\frac{n^2}{2}$	$\frac{n^2}{4}$	n	✓	✓
ShellSort	?	?	n	×	✓
RadixSort	$O(nk)$	$O(nk)$	$O(nk)$	✓	×

总结

本节内容

- 用归纳思想来设计分析算法
  - 选择排序 SelectSort、寻找多数元素 Majority
  - 多项式计算 Horner、生成排列 Permutations
  - 基数排序 RadixSort
- 排序算法回顾
  - 插入排序 InsertSort、希尔排序 ShellSort
  - 排序算法的性质