算法导论学习笔记

Yaohui Li

2022 年 9 月 15 日

目录

第一部分　基础知识 **2**

第一章　算法在计算中的作用 **3**

1.1　算法 3

1.2　作为一种技术的算法 3

1.3　思考题 4

第二章　算法基础 **5**

2.1　插入排序 5

2.2　分析算法 10

2.3　设计算法 11

2.3.1　分治法 11

2.3.2　分析分治算法 11

第一部分

基础知识

第一章　算法在计算中的作用

1.1　算法

算法 (algorithm) 就是任何良定义的计算过程, 该过程取某个值或值的集合作为**输入**并产生某个值或值的集合作为**输出**. 这样算法就是把输入转换成输出的计算步骤的一个序列.

练习

1.1-1 **给出现实生活中需要排序的一个例子或者现实生活中需要计算凸壳的例子.**
需要排序的例子为: 展示排行榜, 微博热搜榜单等. 需要计算凸壳 (凸包) 的例子: 割平面法求解线性整数规划问题.

1.1-2 **除速度外, 在真实环境中还可能使用哪些其它有关效率的量度?**
计算步骤的复杂性, 计算过程所需的内存空间大小等.

1.1-3 **选择一种你以前已知的数据结构, 并讨论其优势和局限.**
以数组为例. 优势: 结构简单, 容易实现. 局限: 删除元素效率低下.

1.1-4 **前面给出的最短路径与旅行商问题有哪些相似之处? 又有哪些不同?**
相似之处: 都属于图论中的问题, 且优化目标均为最小化总距离. 不同点: 受到的约束不同, 最短路径问题不需要遍历每一个节点.

1.1-5 **提供一个现实生活的问题, 其中只有最佳解才行. 然后提供一个问题, 其中近似最佳的解也足够好.**
必须找到最佳解的问题: 找到班级中身高最高的同学. 近似最佳的解也足够好的问题: 装机时, 找到一个比总用电功率大的任意一个电源.

1.2　作为一种技术的算法

算法的效率主要体现在时间和空间这两个方面, 因为计算机不可能是无限快的, 存储空间也不是免费的.

练习

1.2-1 **给出在应用层需要算法内容的应用的一个例子，并讨论涉及的算法的功能.**
加载包含大量图片的网页时, 图片的加载顺序问题. 算法应当首先加载用户最先看到的图片.

1.2-2 **假设我们正比较插入排序与归并排序在相同机器上的实现. 对规模为 n 的输入，插入排序运行 $8n^2$ 步，而归并排序运行 $64n\lg n$ 步. 问对哪些 n 值，插入排序优于归并排序?**
若需插入排序优于归并排序, 则需 n 满足不等式: $8n^2 \leq 64n\lg n$. 解得: $2 \leq n \leq 43$.

1.2-3 **n 的最小值为何值时，运行时间为 $100n^2$ 的一个算法在相同机器上快于运行时间为 2^n 的另一个算法?**
原问题等价于求解满足不等式: $100n^2 \leq 2^n$ 的最小的 n. 解得: $n \geq 15$.

1.3　思考题

1-1 **(运行时间的比较) 假设求解问题的算法需要 $f(n)$ 毫秒，对下表中的每个函数 $f(n)$ 和时间 t，确定可以在时间 t 内求解的问题的最大规模 n.**
基本的计算思路是求解满足不等式: $f(n) \leq t$ 的最大的 n. 求解结果如下表所示:

	1 秒钟 (10^3 毫秒)	1 分钟 (6×10^4 毫秒)	1 小时 (3.6×10^6 毫秒)	1 天 (8.64×10^7 毫秒)	1 月 (2.60×10^9 毫秒)	1 年 (3.15×10^{10} 毫秒)	1 世纪 (3.15×10^{12} 毫秒)
$\lg n$	2^{10^3}	$2^{6\times10^4}$	$2^{3.6\times10^6}$	$2^{8.64\times10^7}$	$2^{2.60\times10^9}$	$2^{3.15\times10^{10}}$	$2^{3.15\times10^{12}}$
$\sqrt{n}$	10^6	3.6×10^9	1.30×10^{13}	7.65×10^{15}	6.72×10^{18}	9.95×10^{20}	9.95×10^{24}
n	10^3	6×10^4	3.6×10^6	8.64×10^7	2.60×10^9	3.15×10^{10}	3.15×10^{12}
$n\lg n$	140	4895	2.04×10^5	3.94×10^6	9.79×10^7	1.05×10^9	8.67×10^7
n^2	31	244	1897	9295	50990	177482	1774823
n^3	10	39	153	442	1375	3158	14658
2^n	9	15	21	26	31	34	41
$n!$	6	8	9	11	12	13	15

第二章　算法基础

2.1　插入排序

插入排序

插入排序的算法描述为:

INSERTION-SORT(A)

1 **for** $j = 2$ **to** $A.length$
2 　　$key = A[j]$
3 　　// Insert $A[j]$ into the sorted sequence $A[1..j-1]$.
4 　　$i = j - 1$
5 　　**while** $i > 0$ and $A[i] > key$
6 　　　　$A[i+1] = A[i]$
7 　　　　$i = i - 1$
8 　　$A[i+1] = key$

插入排序的 C++ 代码为:

```cpp
void InsertSort(std::vector<int>& v) {
    /* v is sorted */
    if (v.size() < 2) {
        return;
    }
    for (int j = 1; j != v.size(); ++j) {
        int key = v[j];
        /* Insert v[j] into the sorted sequence v[1..j-1] */
        int i = j - 1;
        while (i >= 0 && v[i] > key) {
            v[i + 1] = v[i];
            --i;
        }
        v[i + 1] = key;
    }
```

```
16 }
```

以数组 $A = [5, 2, 4, 6, 1, 3]$ 为例, 该数组的初始状态如图 2.1所示. 对该数组进行非降序的排列后, 其状态应该如图 2.2所示.

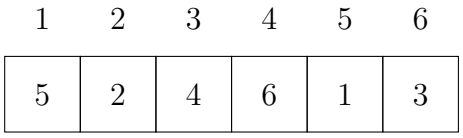

图 2.1: 排序前

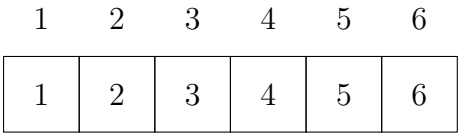

图 2.2: 排序后

采用插入排序的算法流程可以用图 2.3进行表示:

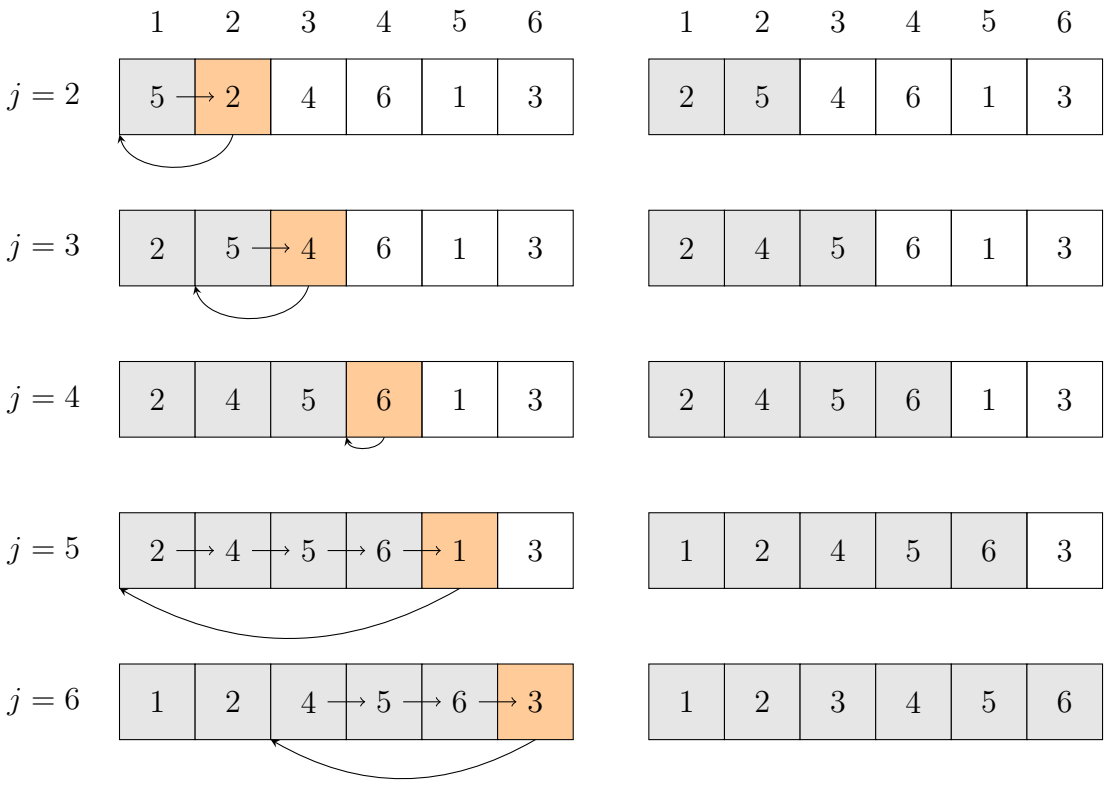

图 2.3: 数组 A 的插入排序流程

循环不变式

循环不变式主要用来帮助我们理解算法的正确性. 关于循环不变式, 我们必须证明三条性质:

初始化: 循环的第一次迭代之前, 它为真.

保持: 如果循环的某次迭代之前它为真, 那么下次迭代之前它仍为真.

终止: 在循环终止时, 不变式为我们提供一个有用的性质, 该性质有助于证明算法是正确的.

对于插入排序的例子而言, 循环不变式三个性质的证明如下:

初始化: 首先证明在第一次循环迭代前 ($j = 2$), 循环不变式成立. 此时的子数组 $A[1..j-1]$ 仅仅包含 1 个元素, 即 $A[1]$. 显然, 该子数组是有序的.

保持: 在某次迭代前, 若子数组 $A[1..j-1]$ 是有序的, 则将第 j 个元素 $A[j]$ 插入到子数组中正确的位置上 (假设为位置 i) 后, 子数组中原来的第 i 到 $j-1$ 的元素依次向后移动一个位置占据 $i+1$ 到 j 的位置. 在下次迭代前, 变量 j 加一后, 子数组仍然是有序的, 仍然为真.

终止: 算法终止是由于 j 大于数组 A 的长度 n, 且由于 j 是整数, 此时有 $j = n+1$. 那么根据前两条性质可知, 此时子数组 $A[1..j-1]$ 是有序的, 而子数组 $A[1..j-1]$ 恰好为排序后的原数组.

练习

2.1-1 以图 2-2 为模型, 说明 INSERTION-SORT 在数组 $A = [31, 41, 59, 26, 41, 58]$ 上的执行过程.

解答:

执行过程如图 2.4所示:

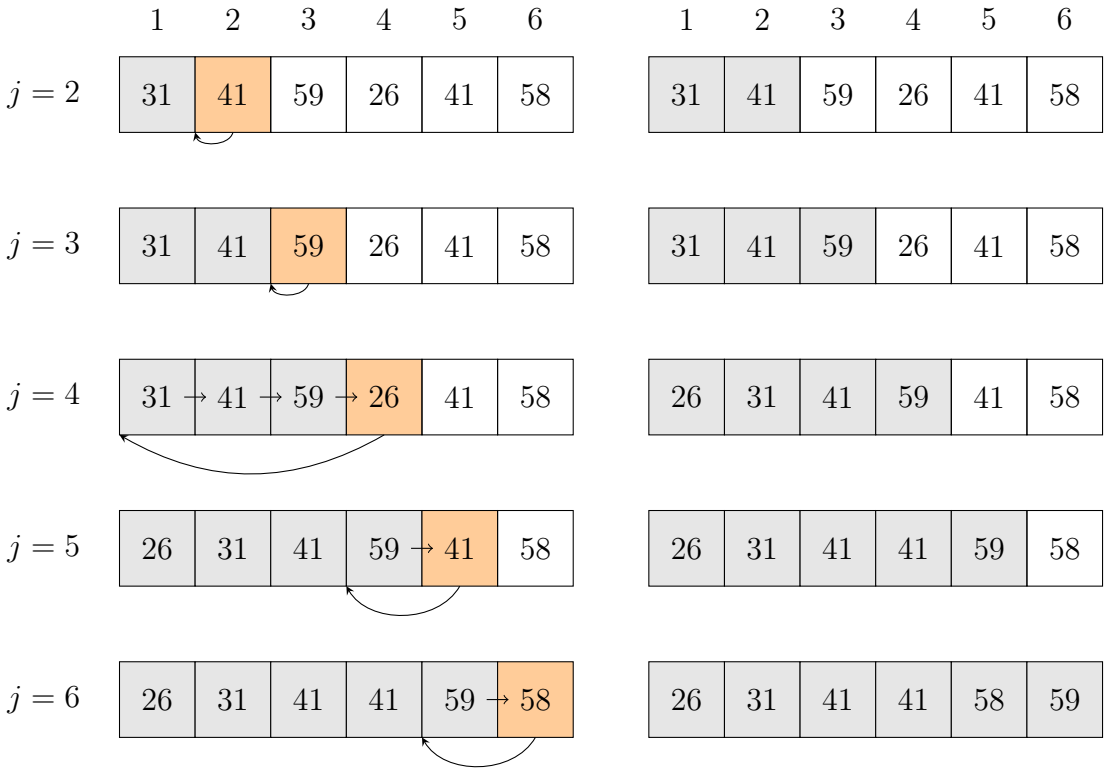

图 2.4: 数组 $A = [31, 41, 59, 26, 41, 58]$ 的插入排序执行过程

2.1-2 重写过程 INSERTION-SORT, 使之按非升序 (而不是非降序) 排列.

解答:

非升序版本的伪代码如下:

INSERTION-SORT(A)

```
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1..j-1].
4      i = j - 1
5      while i > 0 and A[i] < key          // The main difference is here
6          A[i+1] = A[i]
7          i = i - 1
8      A[i+1] = key
```

非升序版本和非降序版本的主要差别在于将 $A[j]$ 插入已排序的序列 $A[1..j-1]$ 这一步. 非升序版本的 Python 实现为:

```python
def insertion_sort_descending(A):
    for j in range(1, len(A)):
        key = A[j]
        i = j - 1
        while i >= 0 and key > A[i]:
            A[i + 1] = A[i]
            i = i - 1
        A[i + 1] = key
```

2.1-3 考虑以下**查找问题**:

输入: n 个数的一个序列 $A = \langle a_1, a_2, \ldots, a_n \rangle$ 和一个值 v.

输出: 下标 i 使得 $v = A[i]$ 或者当 v 不在 A 中出现时, v 为特殊值 NIL.

写出**线性查找**的伪代码, 它扫描整个序列来查找 v. 使用一个循环不变式来证明你的算法是正确的. 确保你的循环不变式满足三条必要的性质.

解答:

线性查找的算法描述为:

LINEAR-SEARCH(A, v)

```
1  for i = 1 to A.length
2      if A[i] == v
3          return i
4  return NIL
```

其 Python 实现为:

```python
def linear_search(A, v):
    for i in range(len(A)):
        if A[i] == v:
            return i
    return -1
```

循环不变式的证明如下:

初始化: 在循环迭代开始之前, 相当于没有遍历过任何元素, 自然不可能找到目标元素 v, 所以函数不会 return, 循环必定可以开始.

保持: 若循环能进行下去, 则意味着在子数组 $A[1..i-1]$ 内没有找到目标元素 v, 因此需要考察第 i 个元素. 在考察完第 i 个元素之后, "在子数组 $A[1..i-1]$ 内没有找到目标元素 v"这一命题仍然为真.

终止: 一种情况是, 当考察到第 i 个元素时, 若 $A[i]$ 等于 v, 则意味着已经找到目标元素, 直接返回 i. 另一种情况是, 循环迭代的终止是由于 i 大于数组 A 的长度 n, 此时 $i = n + 1$. 那么根据前两条性质可知, 子数组 $A[1..i-1]$ 内没有找到目标元素 v. 此时的子数组恰好是原问题数组, 意味着数组 A 内不存在目标元素 v, 故返回 NIL.

2.1-4 考虑把两个 n 位二进制整数加起来的问题, 这两个整数分别存储在两个 n 元数组 A 和 B 中. 这两个整数的和应按二进制形式存储在一个 $(n+1)$ 元数组 C 中. 请给出该问题的形式化描述, 并写出伪代码.

解答:

算法描述为:

```
ADD-BINARY(A, B)
 1  int[n + 1]C = {0, . . , 0}                    // n = A.length
 2  val = 0                                       // 用于保存下一位 (进位) 值的变量
 3  for i = n to 1
 4       sum = A[i] + B[i] + val
 5       if sum == 3                              // 当前位求和为 3
 6            val = 1                             // 进位 1
 7            C[i + 1] = 1                        // 当前位余 1
 8       elseif sum == 2                          // 当前位求和为 2
 9            val = 1                             // 进位 1
10            C[i + 1] = 0                        // 当前位余 0
11       else
12            val = 0                             // 不进位
13            C[i + 1] = sum                      // 当前位余 sum
14  C[1] = val
15  return C
```

2.2　分析算法

最坏情况与平均情况分析

对于插入排序, 最好情况下, $T(n) = an + b$, 最坏情况下, $T(n) = an^2 + bn + c$.

我们往往集中于只求**最坏情况运行时间**, 即对规模为 n 的任何输入, 算法的最长运行时间, 这样做的三点理由:

- 一个算法的最坏情况运行时间给出了任何输入的运行时间的一个上界. 知道了这个界, 就能确保该算法绝不需要更长的时间. 我们不必对运行时间做某种复杂的猜测并可以期望它不会变得更坏.

- 对某些算法, 最坏情况经常出现. 例如, 当在数据库中检索一条特定的信息时, 若该信息不在数据库中出现, 则检索算法的最坏情况会经常出现. 在某些应用中, 对缺失信息的检索可能是频繁的.

- "平均情况"往往与最坏的情况大致一样差. 假定随机选择 n 个数并应用插入排序. 需要多长时间来确定在子数组 $A[1..j-1]$ 的什么位置插入元素 $A[j]$? 平坤来说, $A[1..j-1]$ 中的一半元素小于 $A[j]$, 一半元素大于 $A[j]$. 所以, 平均来说, 我们检查子数组 $A[1..j-1]$ 的一半, 那么 t_j 大约为 $j/2$. 导致的平均情况运行时间结果像最坏情况运行时间一样, 也是输入规模的一个二次函数.

在某些特定的情况下, 我们会对一个算法的**平均情况**运行时间感兴趣; 贯穿于本书, 我们将看到**概率分析**技术被用于各种算法. 平均情况分析的范围有限, 因为对于特定的

问题, 什么构成一种“平均”输入并不明显. 我们常常假定给定规模的所有输入具有相同的可能性. 实际上, 该假设可能不成立, 但是, 有时可以使用**随机化算法**, 它做出一些随机的选择, 以允许进行概率分析并产生某个**期望**的运行时间. 在第 5 章以及后续的其它几章中, 我们将进一步探究随机化算法.

增长量级

我们真正感兴趣的是运行时间的**增长率**或**增长量级**. 一般会忽略低阶项, 保留高阶项. 以插入排序为例, 记最坏情况的运行时间 $\Theta(n^2)$(读作“theta n 平方”).

练习

2.2-1 用 Θ 记号表示函数 $n^3/1000 - 100n^2 - 100n + 3$.
解答: $\Theta(n^3)$

2.2-2 考虑排序存储在数组 A 中的 n 个数: 首先找出 A 中的最小元素并将其与 $A[1]$ 中的元素进行交换. 接着, 找出 A 中的次最小元素并将其与 $A[2]$ 中的元素进行交换. 对 A 中前 $n-1$ 个元素按该方式继续. 该算法称为**选择算法**, 写出其伪代码. 该算法维持的循环不变式是什么? 为什么它只需要对前 $n-1$ 个元素, 而不是对所有 n 个元素运行? 用 Θ 记号给出选择排序的最好情况与最坏情况的运行时间.
解答:
选择算法的伪代码为:

SELECTION-SORT(A)
1 **for** $i = 1$ **to** $A.length - 1$
2 　　$minIndex = i$
3 　　**for** $j = i + 1$ **to** $A.length$
4 　　　　**if** $A[j] \leq A[minIndex]$
5 　　　　　　$minIndex = j$
6 　　$temp = A[i]$
7 　　$A[i] = A[minIndex]$
8 　　$A[minIndex] = temp$

2.3　设计算法

2.3.1　分治法

2.3.2　分析分治算法