数据结构与算法

Data Structure and Algorithm

极夜酱

目录

Part I

数据结构与算法

Chapter 1 数据结构与算法

1.1 算法

1.1.1 算法（Algorithm）

算法是一个很古老的概念，最早来自数学领域。

有一个关于算法的小故事：在很久很久以前，曾经有一个顽皮又聪明的熊孩子，天天在课堂上调皮捣蛋。终于有一天，老师忍无可忍，对熊孩子说："臭小子，你又调皮啊！今天罚你做加法，算出 $1+2+3+\cdots+9999+10000$ 累加的结果，算不完不许回家！"

老师以为，熊孩子会按部就班地一步一步计算：

$$1+2=3$$

$$3+3=6$$

$$6+4=10$$

$$10+5=15$$

$$\cdots$$

这还不得算到明天天亮？够这小子受的！老师心里幸灾乐祸地想着。谁知仅仅几分钟后……

"老师，我算完了！结果是 50005000，对不对?"

"这，这，这……你小子怎么算得这么快？我读书多，你骗不了我的！"

看着老师惊讶的表情，熊孩子微微一笑，讲出了他的计算方法。

首先把从 1 到 10000 这 10000 个数字两两分组相加：

$$1 + 10000 = 10001$$

$$2 + 9999 = 10001$$

$$3 + 9998 = 10001$$

$$4 + 9997 = 10001$$

$$\cdots$$

一共有 $10000 \div 2 = 5000$ 组，所以 1 到 10000 相加的总和可以这样来计算：

$$(1 + 10000) \times 10000 \div 2 = 50005000$$

这个熊孩子就是后来著名的犹太数学家约翰 · 卡尔 · 弗里德里希 · 高斯，而他所采用的这种等差数列求和的方法，被称为高斯算法。

算法是解决问题的一种方法或一个过程，是一个由若干运算或指令组成的有穷序列。求解问题的算法可以看作是输入实例与输出之间的函数。

算法有 5 个特点：

1. 有穷性（finiteness）：算法必须能在执行有限个步骤之后终止。
2. 确定性（definiteness）：算法的每一步骤必须有确切的定义。
3. 输入项（input）：一个算法有 0 个或多个输入。
4. 输出项（output）：一个算法有一个或多个输出，没有输出的算法是毫无意义的。
5. 可行性（effectiveness）：算法中执行的任何计算步骤都是可以被分解为基本的可执行的操作步。

1.1.2 算法描述

算法是可完成特定任务的一系列步骤，算法的计算过程定义明确，通过一些值作为输入并产生一些值作为输出。

流程图（flow chart）是算法的一种图形化表示方式，使用一组预定义的符号来说明如何执行特定任务。

- 圆角矩形：开始和结束
- 矩形：数据处理
- 平行四边形：输入/输出
- 菱形：分支判断条件
- 流程线：步骤

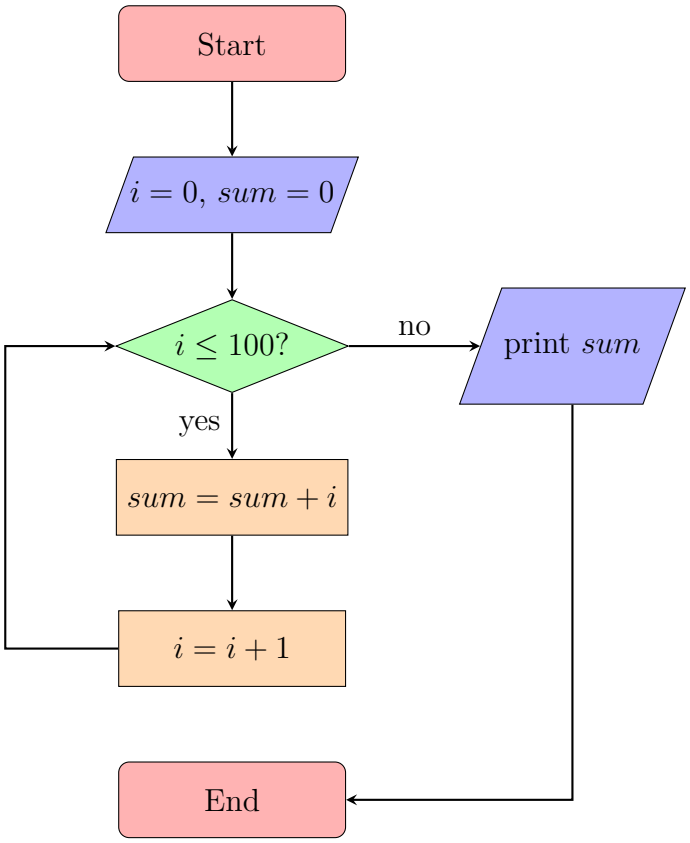

图 1.1: 计算 $\sum_{i=1}^{100} i$ 的流程图

伪代码（pseudocode）是一种非正式的，类似于英语结构的，用于描述模块结构图的语言。使用伪代码的目的是使被描述的算法可以容易地以任何一种编程语言实现。

Algorithm 1 插入排序

```
1: procedure INSERTIONSORT(A[0..n-1])
2:     for j = 2 to n - 1 do
3:         key = A[j]
4:         i = j - 1
5:         while i > 0 and A[i] > key do
6:             A[i+1] = A[i]
7:             i = i - 1
8:         end while
9:         A[i+1] = key
10:    end for
11:    return A
12: end procedure
```

1.2 算法效率

1.2.1 算法效率

算法有高效的，也有拙劣的。在高斯的故事中，高斯所用的算法显然是更加高效的算法，它利用等差数列的规律，四两拨千斤，省时省力地求出了最终结果。而老师心中所想的算法，按部就班地一个数一个数进行累加，则是一种低效、笨拙的算法。虽然这种算法也能得到最终结果，但是其计算过程要低效得多。

在计算机领域，我们同样会遇到各种高效和拙劣的算法。衡量算法好坏的重要标准有两个：时间复杂度、空间复杂度。

让我们来想象一个场景：某一天，小灰和大黄同时加入了同一家公司。老板让他们完成一个需求。一天后，小灰和大黄交付了各自的代码，两人的代码实现的功能差不多。但是，大黄的代码运行一次要花 100 ms，占用内存 5 MB；小灰的代码运行一次要花 100 s，占用内存 500 MB。

“小灰，收拾东西走人，明天不用来上班了!”

小灰虽然也按照老板的要求实现了功能，但他的代码存在两个很严重的问题：运行时间长、占用空间大。

算法效率分析指的是算法求解一个问题所需要的时间资源和空间资源。效率可以通过对算法执行基本运算（步数）的数目进行估算，度量一个算法运算时间的三种方式：

- 最好情形时间复杂度
- 最坏情形时间复杂度
- 平均情形时间复杂度

最坏情形是任何规模为 n 的问题实例运行时间的上界，即任何规模为 n 的实例，其运行时间都不会超过最坏情况的运行时间。

对某些算法，最坏情况经常发生。例如在某个数据库中查询不存在的某条诗句就是查询算法的最坏情形。平均情形有时跟最坏情形差不多。

1.2.2 时间复杂度（Time Complexity）

算法的效率主要取决于算法本身，与计算模型（例如计算机）无关，这样可以通过分析算法的运行时间从而比较出算法之间的快慢。分析一个算法的运行时间应该主要关注与问题规模有关的主要项，其它低阶项，甚至主要项的常数系数都可以忽略。

渐进时间复杂度用大写 O 来表示，所以也被称为大 O 表示法。
时间复杂度有如下原则：

1. 如果运行时间是常数量级，则用 $O(1)$ 表示。
2. 只保留时间函数中的最高阶项。
3. 如果最高阶项存在，则省去最高阶项前面的系数。

在编程的世界中有各种各样的算法，有许多不同形式的时间复杂度，例如：$O(1)$、$O(logn)$、$O(n)$、$O(nlogn)$、$O(n^2)$、$O(2^n)$、$O(n!)$ 等。

1.2.3 空间复杂度（Space Complexity）

内存空间是有限的，在时间复杂度相同的情况下，算法占用的内存空间自然是越小越好。如何描述一个算法占用的内存空间的大小呢？这就用到了算法的另一个重要指标——空间复杂度。

和时间复杂度类似，空间复杂度是对一个算法在运行过程中临时占用存储空间大小的量度，它同样使用了大 O 表示法。

正所谓鱼和熊掌不可兼得，很多时候，我们不得不在时间复杂度和空间复杂度之间进行取舍。在绝大多数时候，时间复杂度更为重要一些，我们宁可多分配一些内存空间，也要提升程序的执行速度。

1.3 基础算法

1.3.1 暴力枚举

暴力破解法也称穷举法，思想就是列举出所有可能情况，然后根据条件判断此答案是否合适，合适就保留，不合适就丢弃。暴力法主要利用计算机运算速度快、精确度高的特点。因此暴力法是通过牺牲时间来换取答案的全面性。

鸡兔同笼

上有三十五头，下有九十四足，问鸡兔各几何？

```
void count(int head, int foot) {
    for(int chicken = 0; chicken <= head; chicken++) {
        int rabbit = head - chicken;
        if(chicken*2 + rabbit*4 == foot) {
            printf("鸡：%2d\t兔：%2d\n", chicken, rabbit);
        }
    }
}
```

百钱买百鸡

公鸡 5 文钱 1 只，母鸡 3 文钱 1 只，小鸡 1 文钱 3 只，如果用 100 文钱买 100 只鸡，那么公鸡、母鸡和小鸡各应该买多少只？

```
void buy(int n, int money) {
    for(int x = 0; x <= n/5; x++) {
        for(int y = 0; y <= n/3; y++) {
            int z = n - x - y;
            if(z > 0 && z % 3 == 0 && 5*x + 3*y + z/3 == money) {
                printf("公鸡：%3d\t母鸡：%3d\t小鸡：%3d\n", x, y, z);
            }
        }
```

```
    }
}
```

1.3.2 字符串逆序

将一个字符串中的字符顺序颠倒过来实现逆序。

字符串逆序

```
void reverse(char *str) {
    int i = 0;
    int j = strlen(str) - 1;
    while(i < j) {
        char temp = str[i];
        str[i] = str[j];
        str[j] = temp;
        i++;
        j--;
    }
}
```

1.3.3 随机算法

随机算法就是在算法中引入随机因素，通过随机数选择算法的下一步操作，它采用了一定程序的随机性作为其逻辑的一部分。

只有随机数生成器的情况下如何计算圆周率的近似值？蒙特卡洛算法就是一种随机算法，用于近似计算圆周率 π 的值。

蒙特卡洛算法是以概率和统计理论方法为基础的一种计算方法，将所求解的问题同一定的概率模型相联系，用电子计算机实现统计模拟或抽样，以获得问题的近似解。为象征性地表明这一方法的概率统计特征，故借用赌城蒙特卡罗命名。

蒙特卡洛算法的基本思想就是当样本数量足够大时，可以用频率去估计概率，这也是求圆周率 π 的常用方法。

1.4 数据结构

1.4.1 数据结构（Data Structure）

数据结构是算法基石，是计算机数据的组织、管理和存储的方式，数据结构指的是相互之间存在一种或多种特定关系的数据元素的集合。一个好的数据结构可以带来更高的运行或者存储效率。

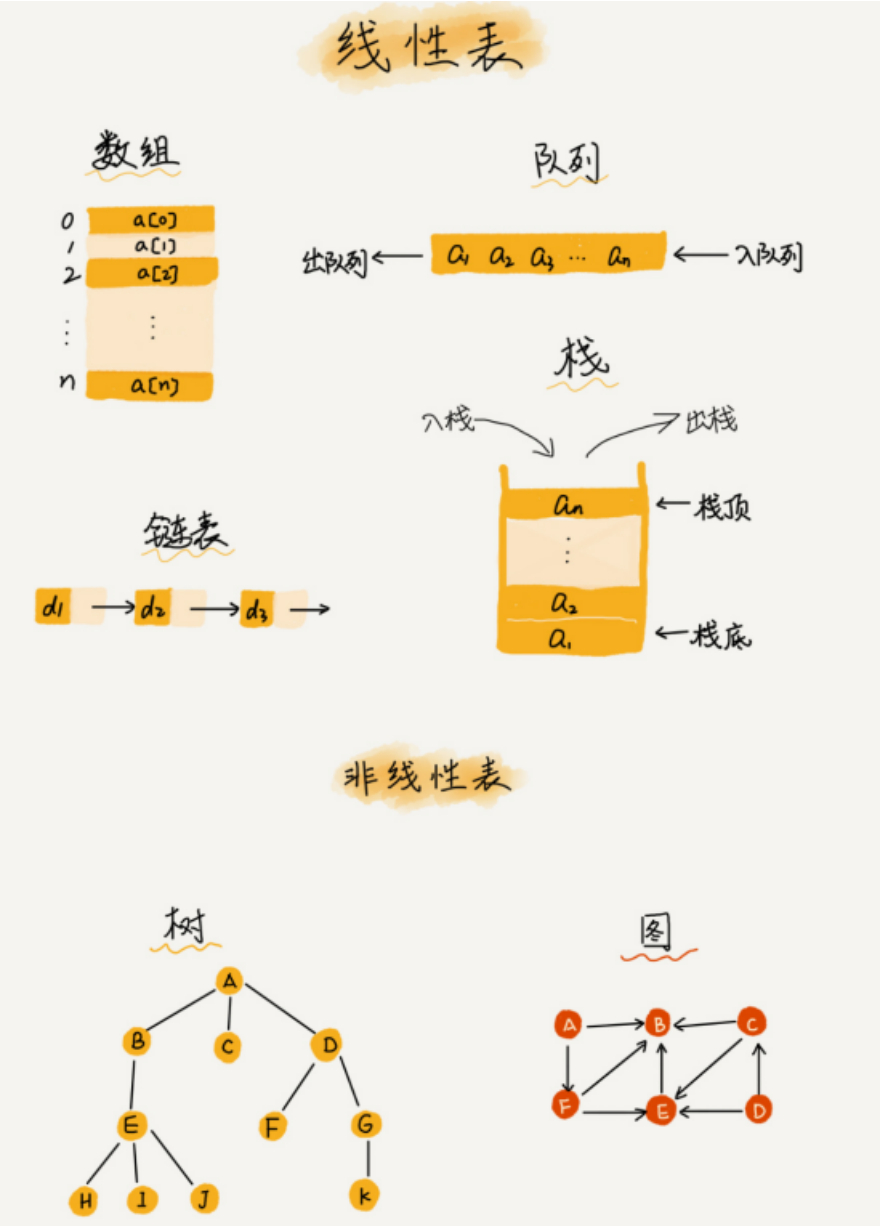

图 1.2: 数据结构