如何以计算机的方式去思考

从上大学第一天开始接触编程，老师便给我们讲过各式各样的算法。从各种查找、排序，到递归、贪心等算法，大一的时候一直在和这些算法搏斗。直到工作后，为了应付面试，仍不得不回过头去啃算法书或者去刷一些算法习题，才能够拾回一些上学时的记忆。为什么算法就这么难以记住呢？或者说，为何计算机的算法不能更直观一些呢？

因为计算机的算法就是反人性的，从本质上来说，这是计算机的思维方式和人脑思维方式的区别而造成的。

人脑思维的机制至今没有一个确定的理论，暂时认为是化学物质和电信号的作用。虽然没有科学的解释，但是我们每个人都有一颗大脑，我们每个人都可以感受到自己的思维方式。

而计算机则是人类创造的，从设计之初它便不是以模拟人脑为目的，因此它有其独特的工作方式，只有理解了计算机的工作方式，才可以学会以它的方式去思考， 才可以写出最适合计算机运行的程序代码。

在排序数组中寻找特定数字 —— 人脑 vs 计算机 round 1

我们通过一个具体的例子，来说明人脑和计算机的思维方式不同，假设我们想要从一个已经排好序的数组中找出一个特定的数字。

已知排序好的数组是 1 2 3 5 7 13 34 67 90 127 308，我们希望找到是否 13 这个数在数组内。

人脑是如何去完成任务的呢？

人脑处理这样的问题几乎是“作弊”的，我们可以一目十行，我们在眼镜一扫视的情况下就发现了 13，所以如果我问自己我是如何找到 13 的，我只能说我“看见”了。

而计算机是如何来完成这个任务呢？

最简单也是最笨的算法就是从数组开始一个一个的读入数组，我相信每个学习过编程基础的同学都可以写出类似下面的代码。

```
boolean isNumInArray(int num, int[] array) {
    for (int i = 0; i < array.length; i++) {
        if (array[i] == num) {
            return true;
        }
    }
    return false;
}
```

计算机需要从数组的第一个元素开始，一个一个的去查当前的数组的元素，和 13 相比，看看是不是相等。为了找出 13 这个数，计算机要做 6 次循环操作，而人几乎是瞬间就看到了答案。

为何计算机解决问题的方式这么“笨”呢？我们先得从计算机的工作原理说起。

CPU 的工作方式

CPU 作为计算机的最核心的部件，也是算法的主要运载体。

CPU 并不会像人一样思考，它只懂得一些基本的指令。每一个 CPU 都有其指令集，指令集是存储在 CPU 内部，对 CPU 运算进行指导和优化的硬程序。通俗一点说，指令集就是 CPU 的所有思维方式。比如常见的指令集中都会有 ADD 指令，这个指令可以将两个寄存器中的值相加，并将存储到另一个寄存器中；与此相对应的也会有 SUB 指令，用于将两个寄存器值相减。如果你去查阅各种 CPU 指令集的手册，会发现基本上都会包含基本的加减乘除指令，以及向内存中存、取数据的指令。而常见的 CPU 指令集，最多也就是几百条指令。也就是说 CPU 只会这几百个命令。

而人脑相对于 CPU，有强大的记忆和联想能力，比如你看到 1+1，就想到 2，看到红灯，就会想到停下来，看到门，就知道去开门把手，这些都是你不假思索可以立刻反映出来的东西。

所以，CPU 会的东西（指令）比人少多了，那 CPU 岂不是很笨？没错，CPU 就是很笨，但是 CPU 的优点也是人脑所无法比拟的：

1. 虽然 CPU 只会干简单的事情（几百种指令），但是它可以在固定的时间（指令执行时间）内保证正确的运算出正确的结果。而人脑不可能保证在固定的时间内一定产生“同样”的思维结果。
2. 现代化的 CPU 工艺可以在一秒钟内执行百万次以上的指令，而人脑的思维速度则比不上，我们一个“念头”最短也需要零点零几秒的反应时间。

综上所述，CPU 是一个既笨又快的家伙。

计算机存储

计算机的常见存储有寄存器、高速缓存、内存、硬盘等。

寄存器就相当于人脑中立刻可以想起来的东西，CPU 所做的一切运算都是针对于在寄存器中的数据进行的。寄存器存储了计算机当前要做什么计算（指令寄存器），要计算的数据（数据寄存器），计算到哪一步了（段寄存器）等信息。无论是最早的有寄存器的 CPU 还是最新最强的的 CPU，它们的寄存器数量最多也只有几十个（特殊情况有几百个），也就是说 CPU 同一时刻能够立刻使用过的信息也就是这几十个数字。

内存则是计算机的主力存储设施，它可以存储运行中的程序的信息，内存相当于图书馆的书架，CPU 需要用某一段内存中的数据是，需要通过 LOAD 指令，同时附上一个书架编号（内存地址），然后内存控制器可以将对应的地址的数据通过总线传输给 CPU，CPU 则将载入的结果放入寄存器中使用。内存存取的速度远小于寄存器，但是访问分布在内存各个区间的数据的速度基本是相等的。

由于大部分时候 CPU 需要读取连续的一段内存来进行运算，因此通常 CPU 会有高速缓存将最近使用过的内存整块缓存起来，而使得 CPU 不必每执行一步就需要去读一次内存。高速缓存的速度介于寄存器和内存之间，但远高于内存。高速缓存的大小一般在几兆到十几兆之间。

硬盘属于外部存储，老式的机械硬盘中会有一个可转的磁头，在读取磁盘文件的时候需要将磁头转到对应的位置，磁盘的速度远低于内存，并且如果磁盘的磁头如果停留在某个位置时，随机磁盘上不同位置的信息，会受到磁头运动的物理速度限制而出现速度不均等的情况。新式的固态硬盘采用了和内存相似的存储介质，在随机访问的性能上提升很大。

所以，计算机有一颗只能记得一点点事情的小脑袋（寄存器），但是能够拥有相对较大的快速记忆（缓存），拥有远超过人类的知识储备（内存），并且还随身携带了巨大的移动图书馆（硬盘），所以从存储上来看，计算机像是一个有先天缺陷的雨人（Rain Man）。

所以，我们来分析一下 round 1 中为何计算机到底做了怎样的操作？

首先我们看我们函数的定义

```
boolean isNumInArray(int num, int[] array)
```

在调用函数的底层实现中，参数是被分配到两个寄存器中。 `isNumInArray` 这个函数，在被调用时，第一个参数 `num` 的值 `13` 会被载入到寄存器（r1）， 的第二个参数 `array` ，传入 CPU 的时候就只是 `array` 在内存中的地址信息，被存储在另一个寄存器（r2）。

而在第四行 `array[i] == num` 时，CPU 需要做三件事才可以完成这工作：

1. 通过 ADD 指令，根据 array 的地址（r2）和 i（r4）的数字，计算需要读取的内存地址
2. 通过 LOAD 指令将内存地址对应的数载入到寄存器（r3）
3. 通过 CMP 指令比较 num（r1）和 r3 的值，结果存储在结果存储器中

而根据操作 3 的结果，如果结果不相等，则 CPU 需要将循环计数器 `i` 加上 1 存入寄存器 r4，再次进行上面的计算。所不同的是，第二到第 N 次的步骤二会比第一次要快很多，因为整个数组的内容已经被高速缓存所捕获。

所以，我们可以看出为何计算机在解决这个问题上显得如此愚笨：

1. 计算机的输入收到限制。计算机一次只能读入单个值（有高速缓存的帮助这并不太糟糕），且在寄存器中放有限的几个值，而人类可以通过视觉等一次性读入多个值存储在脑海中。
2. 计算机的指令有限制，只能支持基本的运算指令。而人脑可以有丰富的指令，比如直接通过一堆刚刚看到的数字中视觉模式匹配出 13 这个数字。

在排序数组中寻找特定数字 —— 人脑 vs 计算机 round 2

计算机在上一轮和人脑的 PK 中败下阵来，然而这并不是很公平，因为数组的数量只有短短的几个，而计算机可以存储的上限远不止于如此。于是我们开始第二次的比拼。 这次我们将输入扩大

1 2 3 5 7 13 34 67 90 127 308 502 ... 2341245 ... （100 万个

查找的数变成了 2341245。

这次人脑和计算机的表现又如何呢？

对于一个普通人，我们假设这 100 万个数字是打印在一本字典里的，那么他如何找出 100 万个有序数组中的某个数字呢？

这时人类引以自豪的“一目十行”的能力已经微乎其微，当数字的位数增大时，且不说一眼比较一个数字是否和目标数字相同已经困难，即使真的有一目十行的本事，在 100 万这样的数字面前也是微乎其微。

于是乎，我们老老实实的去从头到尾比较数字，一页一页的翻开，去看当前的页中有没有数字，没有的话就去翻下一页。

这个思路是不是很熟悉？没错，这就是计算机的思维，和我们上一节中所描述的计算机编码几乎是一样的，除了人可以一眼多看几个数据外。

然而，人类在比较大数是否相等的速度，以及翻字典的速度可远远比不上计算机去读完这 100 万个数的速度，同样是“笨鸟”，计算机每秒百万次的运算能力几乎可以在瞬间就完成这样的任务。

也就是说，在大规模输入的情况下，人脑的思维方式“退化”成和计算机近似，但是被计算机压倒性的性能优势给击败。

在排序数组中寻找特定数字 —— 人脑 vs 计算机 round 3

在第二轮中，人脑败给了计算机，但这样的比拼无疑于两只笨鸟比谁更快。有没有聪明一些的方法呢？

没错，我们学过二分查找（Binary Search）的算法可以派上用场了。

步骤一：有这么有一本打印了 100 万个数字的字典摆在我们的面前，我们不知道要找的数字会在哪里，那么我们先折半打开字典（不用那么精确也没关系），看当前页的第一个数字和最后一个数字，我们要找的数字是否在这个范围内，如果在那么我们可以继续在当前页找这个数字。

步骤二：如果当前页的第一个数字还是比我们要找的数字大，那么我们可以将字典的后半部分撕了（因为我们要找的数字不可能在后半部分了），继续上面的步骤。

步骤三：如果当前页的最后一个数字比我们要找的数字小，那么我们可以将字典的前半部分撕了（理由同上），继续步骤一。

这样我们会讲这本字典越撕越薄，最坏的情况下我们会撕到最后一页，这一页要么有这个数字，要么没有这个数字，但是我们保证按照上面的步骤进行我们不会错过任何可能含有这个数字一页。

这个逻辑和计算机算法中的二分查找原理是一样的，我们来看看实际的算法代码是如何实现的

```
boolean isNumInArray(int num, int[] array, int start, int end) {
    if(num < arr[start] || key > arr[end] || start > end){
        return false;
    }

    int middle = (start + end) / 2; //找到对折点
    if(array[middle] > num) {
        return isNumInArray(arr, key, start, middle - 1); //撕掉后一半
    } else if(array[middle] < num){
        return isNumInArray(arr, key, middle + 1, end); //撕掉前一半
    }else {
        return middle;
    }
}
```

我们可以看出，和人类的思维方式比，计算机不会翻“一页”，它只会翻看一个数字，但是其他的思维方式是一模一样的。利用这样的算法，人类虽然从结果上还是比计算机要慢，但是双方都找到了最适合的方法，达到自我效率的最大提升。

在排序数组中寻找特定数字 —— 更多的思考

那么我们回过头来看，为什么我要假设这 100 万数字打印在字典上呢？因为字典和计算机内存的模型很像。

计算机可以通过内存地址来直接访问内存，这一点和通过字典的页码来翻到某一页，这一点是近似的。

在计算机编码中我们可以知道数组的长度，而通过折半的方法找到中间的数，字典有厚度，我们可以通过厚度减半来找到中间的页码，这一点也是相似的。

试想一样，如果 100 万的数字不是打印在字典，而是印在一条公路上，我们是否还可以用上一节的算法来人肉二分查找？答案是不可以，因为跑到公路的一半会消耗你很多的体力，如果采用二分法查找比起 round 1 中的最笨办法只会让你耗费更多的体力。因为公路这一存储的概念，对应的便不是内存的模型，而是磁带（Tape）的模型，那么对于这样的模型，我相信不论是人或者是计算机， 都需要调整算法，来达到最高的效率。

总结

通过以上的例子，我们可以看到，计算机的算法反人性，是因为计算机不是一个“正常人”，它有自己的缺陷，也有自己的长处。很多时候我们觉的算法不直观，不是因为我们的思维能力比计算机差，而恰恰是因为作为人类我们同时接触的信息太多，所会的东西也太多而阻塞了我们的思维。那么这种时候，不妨将自己“堕落”成一台“鼠目寸光”和“所知甚少”的计算机，这时可能会有更清晰的思路。