Chapter 1

算法与算法分析

本章作为整个讲义的第一章，将为同学们介绍以下问题:

- 什么是算法?
- 我们要怎么样证明算法的正确性?
- 如何衡量算法会运行多长时间？占用多大的空间?
- 算法设计与分析课程主要学习的内容是什么?
- 算法设计与分析课程在整个计算机科学或软件工程学科中的位置是什么?

1.1 什么是算法?

关于算法（Algorithm）究竟是什么这个问题，网上有很多的解答，在这里我们不妨列举几条:

- 算法在数学和计算机科学之中，指一个被定义好的、计算机可施行其指示的有限步骤或次序。(来自维基百科)
- 算法是有限条指令的序列，这个指令序列确定了解决某个问题的运算或操作的步骤。(来自教材《算法设计与分析》)
- 算法就是求解问题的步骤，它必须是确定的（无歧义)、可行的（算法中的运算都是基本的，理论上能够用纸和笔完成)、有限的（算法必须能在有限的步骤内实现)。此外，算法必须有输出（计算结果)，通常还至少有一个输入量。(来自参考书《算法概论》)
- 所谓算法就是定义良好的计算过程，它取一个或一组值作为输入，并产生出一个或一组值作为输出。亦即，算法就是一系列的计算步骤，用来将输入数据转换成输出结果。(来自参考书《算法导论》)

从上面的介绍中我们不难发现，对于算法是什么的解释虽然有很多种，但是都离不开这么几点：首先，算法是被定义好的，也就是说一旦一个算法被设计出来，它的行为就是固定的。其次，算法必须是有限的，也就是说，无论算法运行了多长时间，总有一个时间点算法的运行会结束。再次，算法具有输入和输出数据，也就是说，算法的运行就是处理给定的输入得到输出结果的过程。

上面的关于算法的定义描述在我看来略显表面，我更喜欢美国麻省理工学院算法导论课程中对算法给出的定义，可能在我的复述之下显得不是那么严谨，但是却更加深刻一些:

- 算法是一组有限的指令序列，它用有限的步骤，处理任意规模大小的输入数据，最终给出输出结果。

这里面揭示了更加重要的两点，算法的步骤是固定的、有限的，而输入数据可以是任意规模大小的。

之所以我认为这个定义更加深刻，是因为它为算法的设计、证明、和分析提供了重要的指导:

- 首先，在算法的设计层面上来说，要想使用有限的步骤处理任意规模的数据，说明了算法的设计当中会不可避免的使用循环和递归。

- 其次，进一步地，既然使用循环和递归来设计了算法，那么对于算法的正确性的证明，就不可避免的要用到数学归纳法。

- 最后，我们还是考虑，既然使用了循环和递归来设计了算法，那么循环的次数，递归的深度就会随着输入数据的规模而变化，那么我们就要想一些方式来分析算法的运行效率。

作为一个例子，我们接下来要设计几个算法，来探讨判断在给定的一个含有 n 个整数的，且是由小到大排好序的数组中，是否存在数 x 这个问题。

首先各位同学已经学过了《程序设计基础》课，那么很容易地，同学们可以写出一个简单的线性查找算法，如 LINEAR-SEARCH 所示。该算法的输入是一个数组 $A[1..n]$ 和一个数 *target*，其中 $A[1] \le A[2] \le ... \le A[n]$，算法的输出是：若存在 $1 \le j \le n$ 使得 $A[j] = n$，则返回 *True*，否则返回 *False*。

LINEAR-SEARCH(A, target)

```
1: for i = 1 to n do
2:     if a[i] == target then
3:         return True
4: return False
```

算法我们已经给出了，接下来我们要回答的问题就是:（1）如何证明这个算法是正确的?（2）这个算法要运行多长时间？要用多大的空间?

1.2 怎样证明算法是正确的?

为了回答这个问题，我们要先复习一些离散数学课程中的一些基本概念。第一个问题是: 什么是证明?

- 证明，就是判断一个命题（或谓词）公式是不是重言式（也叫永真式)。

在算法设计与分析课程中，为了证明算法的正确性，我们通常要把算法正确与否的论断用一个命题描述出来。然而，光用命题描述还不够，我们还需要使用谓词和蕴含逻辑。

- 谓词，是指真值依赖于变量的命题。
- 蕴含逻辑 $p \Rightarrow q$ 的真值为假，当且仅当命题 p 为真而命题 q 为假。

在这里要注意的是，对于蕴含逻辑而言，如果命题 p 的真值为假，那么整个蕴含式一定为真。也就是我们常说的: 当前提为假时，任何结论都可以得到。

举个例子: 我对你说“如果太阳从西边升起来了，那么我就是亿万富翁”，这句话并没有欺骗你。因为太阳并不从西边升起来，所以其实你也并不需要关心我到底是不是亿万富翁。[1]

除此之外，正如我们之前所提到的，对于算法的正确性的证明，通常还需要用到数学归纳法。在离散数学中，数学归纳法被描述为归纳公理。

- 归纳公理: 对于谓词 $P(n)$，若 $P(0)$ 为真，且 $(\forall n \in \mathbb{N})(P(n) \Rightarrow P(n+1))$ 为真，那么 $(\forall n \in \mathbb{N})P(n)$ 都为真。

展开来写就是:

- 归纳公理: 对证明基本情况于谓词 $P(n)$，若 $P(0)$, $P(0) \Rightarrow P(1), P(1) \Rightarrow P(2), ...$ 都为真，那么 $P(0), P(1), P(2), ...$ 都为真。

也就是说，如果我们要应用数学归纳法，通常要经过三个步骤: 第一个步骤是确定谓词 $P(n)$ 的含义是什么，第二个步骤是要证明基本情况即 $P(0)$ 为真，第三个步骤是证明一般情况，即 $(\forall n \in \mathbb{N})(P(n) \Rightarrow P(n+1))$ 为真。而证明 $P(n) \Rightarrow P(n+1)$ 为真，可以分情况讨论，第一种情况是当 $P(n)$ 为假的时候，如我们之前讨论的那样，这种情况 $P(n) \Rightarrow P(n+1)$ 一定为真，因此我们只需要考虑第二种情况，也就是当 $P(n)$ 为真的时候，$P(n+1)$ 是不是也为真。

现在我们回到一开始，讨论怎样证明线性查找算法是正确的。首先，我们需要将我们要证明的算法的正确性写成定理的形式，也就是一个命题。

[1]这也说明了“如果我当初 XXXX，那么我现在就 XXXXX 了”这个句式所表达的是一句废话。

- 定理：对于给定的一个含有 n 个整数的由小到大排好序的数组，线性查找算法（LINEAR-SEARCH）能够成功判断出其中是否存在整数 *target*。

要证明这个命题是重言式，我们需要应用归纳公理来证明下面这个引理:

- 算法 LINEAR-SEARCH 能够进入到第 k 轮迭代当且仅当数组 $A[1..k-1]$ 中不存在 *target*。

第一个步骤是确定谓词 $P(k)$，在这里我们简单的将命题本身作为谓词，即:

- $P(k)$: 算法 LINEAR-SEARCH 能够进入到第 k 轮迭代当且仅当数组 $A[1..k-1]$ 中不存在 *target*。

第二个步骤是证明基本情况，$P(1)$ 为真，也就是说，即算法能够进入第 1 轮迭代，当且仅当子数组 $A[1..0]$ 中不存在 *target*。要证明当且仅当成立，需要证明两个方向。首先我们证明 “⇒” 方向，这时算法能够进入第 1 轮迭代，此时子数组 $A[1..0]$ 为空，不存在 *target*。紧接着我们证明 “⇐” 方向，我们注意到无论如何算法总能进入第一轮迭代，因此基本情况成立。

第三个步骤是证明一般情况，$\forall k \in \mathbb{N}(P(k) \Rightarrow P(k+1))$ 为真。若 $P(k)$ 为真，要证明 $P(k+1)$ 为真，由归纳假设，算法能够进入第 k 轮迭代当且仅当子数组 $A[1..k-1]$ 中不存在 *target*。

接下来我们首先证明 “⇒”: 即算法能够进入第 $k+1$ 轮迭代，这蕴含了算法能够进入第 k 轮迭代，目标是证明子数组 $A[1..k]$ 中不存在 *target*。如果第 k 个数字是 *target*，那么算法直接返回 *True*，不会进入第 $k+1$ 轮迭代，此时 $P(k+1)$ 为真。如果第 k 个数字不是 *target*，那么由归纳假设，前 k 个数字中都不存在 *target*，此时 $P(k+1)$ 也为真。

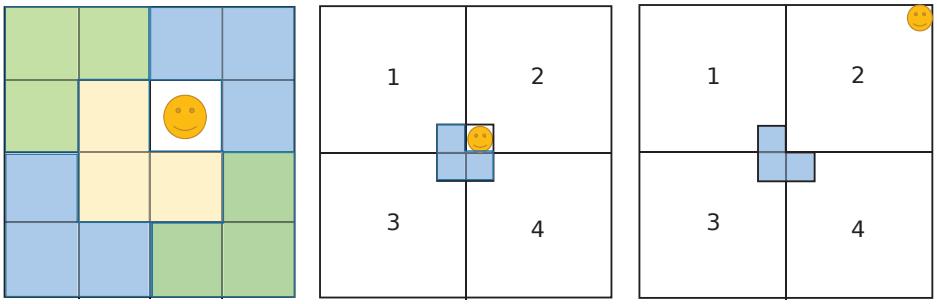
再证明 “⇐”，即若子数组 $A[1..k]$ 中不存在 *target*，那么算法能够进入第 $k+1$ 轮迭代。由归纳假设，若子数组 $A[1..k-1]$ 中不存在 *target*，那么算法能够进入第 k 轮迭代，此时，考察算法的第 2 到 3 行，由于第 k 个元素不是 *target*，所以算法不被中断，继续进行下一轮迭代，即能够进入第 $k+1$ 轮迭代。

接下来我们证明原定理：对于给定的一个含有 n 个整数的由小到大排好序的数组，线性查找算法（LINEAR-SEARCH）能够成功判断出其中是否存在整数 *target*。分两种情况：（1）数组中存在 *target*: 如果数组 $A[1..n]$ 中存在 *target*，那么算法进入不到第 $n+1$ 轮迭代，中间必定返回 *True*。（2）数组中不存在 *target*，那么数组将进入第 $n+1$ 轮迭代，此时返回 *False*。

这样，我们就使用数学归纳法证明了整个算法为是正确的。

1.3 几个其它应用数学归纳法的例子

为了让同学们更好的掌握数学归纳法，我在这里再列举几个应用数学归纳法完成证明的例子。



(a) 4×4 铺设瓷砖示例 (b) 尝试使用数学归纳法 (c) 放在角落里

Figure 1.1: 铺设瓷砖示例

1.3.1 铺瓷砖问题

假设有一个正方形的院落，院落的边长是 2 的幂，现在要在院子中央的 4 块单位中找一块放置一个雕像，求证剩下的部分可以被用长度为 2 的 L 型瓷砖铺满。

如图1.1(a)所示，对于一个 4 的院落而言，在中央选择一个单位放置雕塑后，剩下的空间可以使用边长为 2 的 L 型瓷砖铺满。

接下来我们证明，对于任意一个边长为 2^n 的院落，选择中央一个单位放置雕塑后，剩余的部分一定可以使用边长为 2 的 L 型瓷砖铺满。

我们将使用数学归纳法来证明如上命题。第一步，确定谓词 $P(n)$，在这里我们简单的将命题本身作为谓词，即:

- $P(n)$: 对于任意一个边长为 2^n 的院落，选择中央一个单位放置雕塑后，剩余的部分一定可以使用边长为 2 的 L 型瓷砖铺满。

第二步，证明基本情况 $P(0)$，这时院落的边长为 1，直接将雕塑放在院落中，没有剩余的空间，命题得证。

第三步，证明 $(\forall n \in \mathbb{N})(P(n) \Rightarrow P(n+1))$ 为真。假设对于一个边长为 2^n 的院落存在一种合理的铺设方案，那么对于任意一个边长为 2^{n+1} 的院落，我们考虑从中间将该院落划分为四个等面积为 2^n 的部分。如图1.1(b)所示，编号 1-4 是长度为 2^n 的子块，若每一个子块都可以将雕塑放在一个角落里，然后能够使用边长为 2 的 L 型瓷砖铺满剩余的部分，那么就可以将雕塑放在其中一个子块的角落上，然后剩余的空白正好可以使用一个 L 型瓷砖铺设上。接下来问题就转换为了要证明：对于任意一个边长为 2^n 的院落，可以将雕塑摆放在角落里，然后剩余的部分可以使用边长为 2 的 L 型瓷砖铺满。

对于这个子命题，我们可以再用一个数学归纳法来证明。同样地，第一步，确定谓词 $P(n)$，我们依然使用命题本身作为谓词，即:

- $P(n)$: 对于任意一个边长为 2^n 的院落，可以将雕塑摆放在角落里，然后剩余的部分可以使用边长为 2 的 L 型瓷砖铺满。

第二步，证明基本情况，即 $P(0)$。此时院落的边长为 1，将雕塑摆放在这里就可以了，命题得证。

接下来是第三步，证明 $(\forall n \in \mathbb{N})(P(n) \Rightarrow P(n+1))$。如图1.1(c)所示，假设对于任意边长为长为 2^n 的院落，可以将雕塑摆放在角落里，然后剩余的部分可以使用边长为 2 的 L 型瓷砖铺满，那么对于任意一个边长为 2^{n+1} 的院落，我们依然将它分成 4 个边长为 2^n 的子院落。由假设我们知道，1、2、3、4 号院落可以将雕塑放在角落里然后剩余的部分用 L 型瓷砖铺满，那么我们把 1、3、4 号院落的空缺部分放在一起，在 2 号院落的空缺部分摆放雕塑，这样一来空缺的部分正好可以使用 L 型瓷砖铺设上，原命题得证。

综上所述，我们也成功的证明了一开始的命题，即：对于任意一个边长为 2^n 的院落，选择中央一个单位放置雕塑后，剩余的部分一定可以使用边长为 2 的 L 型瓷砖铺满。

1.3.2 一种错误的证明：证明世界上所有马的颜色都相同

使用数学归纳法的时候一定要注意每个细节，接下来我们给出一种典型的错误证明，来证明这个命题：世界上所有马的颜色都相同。第一步是确定谓词：$P(n)$ 世界上任意 n 匹马的颜色都相同。第二步是证明基本情况，$P(0)$，即世界上的 0 匹马的颜色都相同，该命题成立，因为空集中不可能出现两种不同颜色的马。第三步是证明一般情况，即 $\forall n \in \mathbb{N}(P(n) \Rightarrow P(n+1))$，对于任意一个含有 $n+1$ 匹马的集合，我们将其表示为 $H = \{h_1, h_2, .., h_{n+1}\}$，接下来我们将原集合分为两个大小为 n 子集 $H_1 = \{h_1, h_2, ..., h_n\}$ 和 $H_2 = \{h_2, h_3, ..., h_{n+1}\}$，此时假设 $P(n)$ 为真，即任意 n 匹马的颜色都相同，由假设，我们可以知道集合 H_1 和 H_2 中的所有马的颜色都相同，那么我们可以得到原集合 H 中的所有马的颜色都相同，即原命题得证。

上面的证明显然是错误的，因为我们知道现实中并不是所有马的颜色都相同的，那么上面的证明的问题出在了哪里呢?

问题出在了 $P(1) \Rightarrow P(2)$ 这一步不成立，因为此时我们不能从两个子集 $H_1 = \{h_1\}$ 和 $H_2 = \{h_2\}$ 的马的颜色都相同推导到 $H = \{h_1, h_2\}$ 的颜色都相同的。这说明了我们在证明一般情况的时候一定要仔细判别是否是对于任意 n，蕴含式都成立。

1.3.3 华容道的可解性

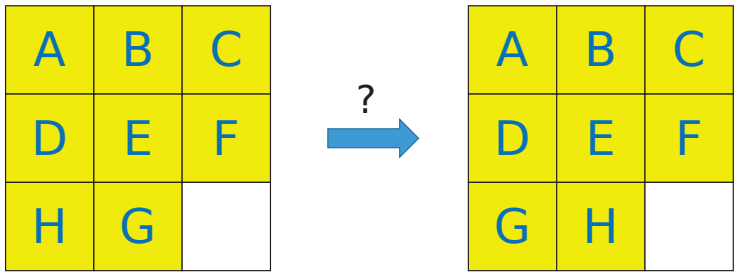


Figure 1.2: 华容道问题

接下来我们要证明一个 3×3 的华容道问题。如图 1.2的左半部分所示，一个 3×3 的矩阵上有 8 个英文字母，除了最后的 G 和 H 以外，其它的字母均是按顺序排列好的，初始状态下，空白的格子在右下角。游戏规则是：玩家的每一步移动，可以选择空白的格子旁边四联通方向之一的字母，把它移动到空白的位置，若最终可以得到图1.2的右半部分的局面，即矩阵上的 8 个字母均按顺序排列好，且空白格子在右下角，则游戏玩家获胜。我们的问题是，是否存在一种可行步骤方案，使得玩家能够获胜?

我们的答案是，从图1.2的左半部分到右半部分的可行步骤方案是不存在的。如何证明这个论断呢? 我们仍然要使用数学归纳法。在使用数学归纳法之前，我想向大家说明一点的就是，如果我们要证明一个 “不可能” 的论断，通常我们需要找出步骤当中的 “不变量”，然后试图寻找这个不变量与某个状态冲突的地方。

为了证明上述论断，我们需要先刻画我们的移动方式。在华容道游戏中，我们存在两种移动方式。第一种移动方式我们称为 “水平移动”，即将处在 (x,y) 位置上的方块移动至 $(x,y+1)$ 或 $(x,y-1)$。第二种移动方式我们称为 “竖直移动”，即将处在 (x,y) 位置上的方块移动至 $(x+1,y)$ 或 $(x-1,y)$。

定义好了上面的两种移动方式后，我们接下来证明下面的两个引理。

- 引理 1：水平移动不改变字母的相对顺序。
- 引理 2：竖直移动要么将一个字母提前到在它之前的两个字母之前，要么将一个字母移动到在它之后的两个字母之后。

我们先证明引理 1。由于水平移动是将处在 (x,y) 位置上的方块移动至 $(x,y+1)$ 或 $(x,y-1)$，这两种方式都不改变所在行的字母的相对顺序，同时，水平移动不影响所在列上的相对顺序，所以水平移动不会改变字母的相对顺序。

接着证明引理 2。对于竖直向下的移动，即将位于 (x,y) 上的字母移动至 $(x+1,y)$，由于 $(x+1,y)$ 是位于 (x,y) 之后的第三个格子，而竖直移动是将该待移动字母与该空格交换，所以竖直向下移动是将该字母移动到它之后的两个字母之后。类似的，对于竖直向上移动，即将位于 (x,y) 上的字母移动至 $(x-1,y)$，由于 $(x-1,y)$ 是位于 (x,y) 之前的第三个格子，而竖直移动是将该待移动字母与该空格交换，所以竖直向上移动是将该字母移动到它之前的两个字母之前。

在上述两个引理的基础上，我们可以证明下面这个引理:

- 引理 3：游戏的一次合法移动，不改变字母的逆序对数的奇偶性。

要证明这个引理，我们就要考虑华容道的一次移动，只可能把一个字母往后移两位，或者把一个字母往前移两位。由引理 1，水平移动不改变字母的相对顺序，也就对逆序对数没有影响。我们要考虑竖直移动的情况，由引理 2，我们知道竖直移动一次只影响之前或之后的两个字母。那么我们先考察往后移两位的情况，如果说该字母往后移动两位，该字母与后两个字母的逆序对数是 0 的情况下，移动后逆序对数增加了 2，紧接着，如果移动前该字母与后两个字母形成的的逆序对数是 1 的情况下，移动后逆序对数不

变，最后，如果移动前该字母与后两个字母形成的逆序对数是 2 的情况下，移动后逆序对的个数减少了 2。无论是那种情况，总的逆序对数的奇偶性都不变。类似的，对于将字母往前移动两位的情况，总的逆序对数的奇偶性也不变，此处省略其证明。

接下来我们使用数学归纳法，证明下面这个引理:

- 引理 4: 华容道游戏以图1.2左半部分的局面开始，移动任意 n 步后，字母的逆序对数都是奇数。

我们可以用数学归纳法证明这个引理，第一步确定谓词 $P(n)$，此处我们简单的将引理本身作为谓词，即:

- $P(n)$: 华容道游戏以图1.2左半部分的局面开始，移动任意 n 步后，字母的逆序对数都是奇数。

第二步，证明基本情况，即 $P(0)$，也就是说当华容道以图1.2左半部分的局面开始的时候，字母的逆序对数是奇数。这种局面下只有一个逆序对 (H,G)，是奇数，命题得证。

第三步，证明一般情况，即 $(\forall n \in \mathbb{N})(P(n) \Rightarrow P(n+1))$。对于任意的 $n+1$ 次移动，我们将最后一次移动回退，即回到只移动 n 次的状态，此时由归纳假设，字母的逆序对数是奇数，那么由引理 3，再恢复最后一次合法的移动，逆序对数仍然是奇数，命题得证。

最后我们可以证明下面的定理了:

- 定理: 华容道游戏以图1.2左半部分的局面开始，不存在一种可行的步骤方案，使得玩家能够得到右半部分的局面。

这个证明就比较容易了，由于开始局面的逆序对数是奇数，同时移动任意 n 步，游戏局面的字母逆序对数始终是奇数，而目标局面的逆序对数是偶数，因此不存在一种可行的步骤方案使得玩家能够得到右半部分的局面，即获胜局面。

1.4 怎样衡量算法运行的时间和占用的空间?

我们用很大的篇幅介绍了数学归纳法，用于证明算法的正确性。接下来我们要考虑的问题是，怎样衡量算法的运行时间和占用的空间? 这里我们主要关注的是算法的运行时间，因为现在我们计算机的内存越来越大了，很少需要再考虑怎样节省内存了。

1.4.1 算法运行时间的衡量方式

一种衡量算法运行时间的方法就是掐表，把算法运行的时间用秒表测量出来。但是这种方式的缺点在于不同的处理器本身的运算速度就不相同，因此我们在 A 处理器下运行 A 算法和在 B 处理器下运行 B 算法，然后对比两个算法的运行时间是不公平的。我们既然要衡量算法本身的运算速度，就希望能够忽略算法本身之外的各种因素例如处

理器的运算速度差异、不同编程语言的运算速度的差异。因此我们可以使用一种更简单的运行时间的衡量方式：去数一数算法执行了多少个指令，或者多少个步骤。

我们以 LINEAR-SORT 给出的线性查找算法为例，算法的第 1 行试图去遍历整个数组，因此算法最多会执行 n 个循环。但其实，我们算法究竟要执行多少个循环，是由 *target* 决定的。如果 *target* 出现在了数组的第一个位置，那么其实我们只需要执行一次循环即可。因此，数算法执行了多少个步骤，通常会遇到最好、平均、和最坏这三种情况。我们应该选择什么情况去着重考虑呢？答案是根据不同的情形去选择，但是一般情况下我们要考虑的是最坏情况。因为算法的设计者需要给算法的使用者一个 “保证”。也就是说，我告诉你 “这个算法最多也就运行 3 秒”，那么你心里会放心，无非就是等 3 秒就可以查看结果了。但是如果我们考虑的是最好的情况，也就是说我告诉你 “这个算法至少要运行 1 秒”，虽然 1 秒比 3 秒短，但是你可能就比较抓狂了，因为算法可能会运行 1 天，1 个月，1 年甚至更长的时间，而你却无法估计你在什么时候才能得到运行结果。

因此，我们在这里用 n 来近似地代表线性查找算法的指令执行数量，之所以说是近似地代表，是因为除了循环之外，还有其他语句例如第 2 行的判断语句也应该被计数。但是我们在比较不同算法的执行效率的时候，当 n 很大时，这些语句的执行次数是可以被忽略的。很快我们就会对比两种算法，来让同学们更直观的感受到哪个算法更快。

1.4.2 二分查找算法

对于1.1节中的查找问题，线性查找算法能够在 n 次循环中找出数组中是否存在 *target*，但是它没有利用数组中的元素已经从小到大排好顺序的这一特点。在这一节我们考察一个叫作 “二分查找” 的算法，如 BINARY-SEARCH 所示，该算法输入一个数组 $A[1..n]$ 和一个整数 *target*，其中 $A[1] \le A[2] \le ... \le A[n]$，算法的输出是：若存在 $1 \le j \le n$ 使得 $A[j] = n$，则返回 $True$，否则返回 $False$。

```
BINARY-SEARCH(A, target)
 1: l = 1, r = n
 2: while l ≤ r do
 3:     mid = (l + r)/2
 4:     if A[mid] == target then
 5:         return True
 6:     else if A[mid] < target then
 7:         l = mid + 1
 8:     else if A[mid] > target then
 9:         r = mid − 1
10: return False
```

接下来我们要:（1）证明算法的正确性,（2）衡量算法的运行时间。（此处举个例子, $[0,1,2,3,5,8,9], target = 5, target = 4$）

首先我们证明该算法的正确性, 对于二分查找算法我们使用排除的思想来证明它的正确性。我们将使用数学归纳法证明这个引理: 算法的进行步骤过程中, 如果 $l \le r$, 那么 *target* 一定不在区间 $[l, r]$ 标定的数组范围之外。第一步, 确定谓词 $P(n)$:

- $P(n)$: 算法在执行第 n 次循环时, 如果 $l \le r$, 那么 *target* 一定不在区间 $[l, r]$ 标定的数组范围之外。

第二步, 证明基本情况 $P(0)$, 初始情况下, 二分查找算法的 $[l, r]$ 所标定的范围是整个数组, 此时, *target* 一定不在区间 $[l, r]$ 标定的数组范围之外, 原命题得证。

第三步, 证明一般情况 $(\forall n \in \mathbb{N})(P(n) \Rightarrow P(n+1))$, 由归纳假设, 算法在执行到第 n 步时, *target* 一定不在区间 $[l, r]$ 标定的数组范围之外, 现在算法执行第 $n+1$ 步, 此时我们选择数组中间的下标 *mid*, 如果 *mid* 指向的数字是 *target* 则算法已经找到了数字, 否则如果 $target < A[mid]$, 此时设置 $r = mid - 1$, 那么此时对于新的 l 和 r 而言, *target* 也一定不在区间 $[l, r]$ 标定的数组范围之外。类似地, 如果 $target > A[mid]$, 此时设置 $l = mid + 1$, 那么对于新的 l 和 r 而言, *target* 也一定不在区间 $[l, r]$ 标定的数组范围之外。这样我们就证明了整个引理。

接下来我们可以证明算法的正确性了:

- 定理: 对于给定的一个含有 n 个整数的由小到大排好序的数组, 二分查找算法（BINARY-SEARCH）能够成功判断出其中是否存在整数 *target*。

首先由一开始的条件判断我们可以知道, 如果 *target* 大于了数组中的最大的数或者小于了数组中的最小的数, 那么 *target* 一定不在数组中, 此时算法一定返回 *False*。因此我们重点要考察循环执行的情况。

首先, 我们观察二分查找循环体中的第 6 行到第 9 行的几个条件判断。我们可以将这三个条件判断总结为两种情况:（1）$A[mid] = target$, 这种情况我们就找到了 *target*, 返回 *True*。（2）$A[mid] \neq target$, 这种情况下, $[l, r]$ 标定的范围会变小, 因为它至少排除掉了 *mid*（通过 $l = mid + 1$ 或者 $r = mid - 1$ 来实现）。

那么, 我们接下来考察整个算法, 循环过程中, 要么就找到了 *target*, 就返回 *True*, 要么没找到 *target*, 此时区间会收缩。此时如果 *target* 不在数组中, 那么 $[l, r]$ 会一直收缩, 直到这个区间标定的数组元素为空, 此时, 算法的第 2 行的循环条件为假, 循环终止, 跳出这个循环, 返回 *False*。

接下来我们换个角度来重新理解一下二分查找算法, 这里我们介绍一种设计套路, 叫作循环不变式。我们在使用数学归纳法证明二分查找算法的时候给出的谓词定义是算法在执行过程中, *target* 也一定不在区间 $[l, r]$ 标定的数组范围之外。这其实就是个循环不变式, 换句话说就是循环过程中, 变的是 l 和 r, 而不变的是 *target* 也一定不在区间 $[l, r]$ 标定的数组范围之外。随着循环的进行, 这个区间越来越小, 直到为空（退出

条件)。如果区间范围变成了空的，那么就说明 *target* 不在数组中，否则就会在区间缩短的过程中找到这个 *target*，这就是循环不变式。

闲扯两句：掌握循环不变式是一个程序员“成熟”的标志之一。写二分查找代码的程序员分两种，一种是“凭感觉”先写一个看似正确的二分查找，然后再运行运行，调试调试，最后跑几个测试样例试试看，貌似运行没有什么问题，但是产品一上线就全是漏洞。然而第二种，掌握了循环不变式的程序员会在写代码之前先设计好要维持的循环不变式，确定好退出条件，证明好算法对于任何输入都能够准确的得到结果，这样写出的代码才能不出问题，产品才足够健壮。

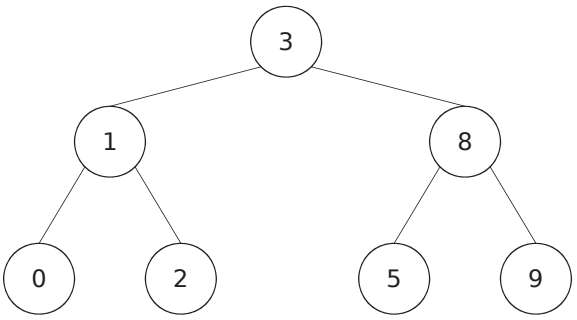


Figure 1.3: 平衡二叉查找树

接下来我们分析算法的运行时间，在这里我们简单地使用示例来说明，若原数组为 $[0,1,2,3,5,8,9]$，执行二分查找的过程相当于在如图1.3所示的平衡二叉查找树上进行查找。这种情况下，我们最多执行树的高度次的循环，在这个例子中是 3 次，对于长度为 n 的数组来说，是 $\log_2 n$。

接下来我们可以对比线性查找算法和二分查找算法，线性查找算法最坏情况下要执行 n 次循环，而二分查找算法最坏情况下只需要执行 $\log_2 n$ 次循环，因此我们可以说对于上面这个问题，二分查找算法比线性查找算法要快。

1.4.3 算法占用多大的空间?

要衡量算法占用了多大空间，我们可以直接的去数出来算法中用到了多少变量，然后计算出算法占用的空间。对于线性查找算法而言，我们在循环中使用了一个循环变量，如果是整型数的话，那么是占用 4 个字节。对于二分查找算法而言，我们要用到三个变量 l, r, mid，那么占用 12 个字节。

对于我们给出的两个示例而言都只用到了循环，然而某些算法需要用到递归，对了递归函数而言，算法要使用的变量每发生一次递归调用就会再开一遍变量的副本，因此还要统计递归的深度。我们将在后面介绍到递归算法的时候给出具体的分析示例。

1.5 算法设计与分析课程主要学习什么?

算法设计与分析是介绍“计算复杂性”基本理论的课程。此处的计算复杂性通常是指算法的运算时间，即“时间复杂度”。我们会涉及到一些“空间复杂度”的计算，用于衡量算法要使用的内存空间，但是正如之前提到的，鉴于现在计算机的内存越来越大，因而我们会在课程中较少的探讨空间复杂度。因此简而言之，算法设计与分析课程是通过介绍一些基本的问题求解策略，教给大家如何设计运行速度更快的正确的算法。

也就是说，算法设计与分析课程将通过展示若干算法的设计技巧，来演示如何证明算法的正确性，并分析算法的时间复杂度、空间复杂度。

1.6 算法设计与分析课程在计算机科学或软件工程学科中的位置是什么?

算法设计与分析课程是处在理论计算机科学（Theoretical Computer Science, TCS）研究方向之中。理论计算机科学这一研究方向主要是进行计算复杂性的研究。主要内容是判断一个问题是否存在一个“有效率”的算法来解决，并且优化算法的计算复杂性。北京交通大学软件学院在理论计算机科学方向上开设的课程还有离散数学和数据结构。

理论计算机科学是其它各个研究方向的基础，除了理论计算机科学这一研究方向之外，还存在着计算机系统、人工智能（计算机视觉、数据挖掘等）、计算机网络、计算机图形学、计算机安全、软件工程等研究方向，学院会在后续的学期中开设涉及到这些研究方向的课程。