ALGORITHM & DATA STRUCTURE

算法与数据结构

什么是算法？什么是数据结构？

- 算法：解题方案的准确而完整的描述，是一系列解决问题的清晰指令
  - 如何处理数据
- 数据结构：相互之间存在一种或多种特定关系的数据元素的集合
  - 如何组织数据

为什么要学习算法和数据结构?

- 对于特定的问题，我们可以选取特定的算法来显著提高计算机的效率
  - 对于长度为5,000, 000的数据，随机情况下，使用对分查找的平均速度比逐个查找**快约224, 4683倍**，如果前者需要**3分钟**，那么后者就需要**1.2年！！**
  - 对于一个列表，随机情况下，连续插入5,000, 000个数据， 使用二叉树进行插入元素的平均速度比使用数组插入元素**快约112,341倍**，如果前者需要**3分钟**，后者需要**234天！！**
- 这里的数据其实都不算大，而且数据类型也是简单的。如果遇到更复杂的情况，算法和数据结构的好坏将会更加明显。

ALGORITHM AND DATA STRUCTURE

算法

背景介绍

- 我们衡量一个算法的好坏有（只要）两个维度，时间与空间：时间指的是一个算法处理一组数据需要时间；空间指的是一个算法处理一组数据所需要的存储容量。
- 对于时间复杂度，我们用一个数学函数 T(n) 描述，如 $T(n) \leq n^2 + 2n + 1$， 表示了这个算法的**最坏**运行时间随着数据量 n 的变化；当 n 很大的时候，这个函数 T(n) 受 n^2 度影响更大，所以对于这个函数，我们简单地写为 $T(N) = O(N^2)$; 同样地，对于算法的最好情况，平均情况也有相关定义，限于时间关系，就不在这里介绍了。
- 常见的算法速度「等级」： O(1) O(logN) O(N) O(NlogN) $O(N^2)$ ...
- 注意：上面的 log 的底数是 2

判断算法运行时间的标准

- 赋值、基本运算（加减乘除余）、数组的读取（一次）需要的时间都是O(1)
- 对于条件语句，如 if语句，取最大的运行时间的语块作为所需时间
- 对于循环语句，最大运行时间为循环的最大次数乘以循环语块的最大运行时间，比如说我们最多需要遍历N个数组元素，那么所需时间就是O(N)：读取一个数组元素需要的时间为O(1)，我们需要遍历N次，所以所需时间为O(N)
- 注意：我们在分析算法的时候，如果引入了外来的函数，也需要把这个外来的函数的时间复杂度计算进去

```
array = [ 2, 3, 1, 9, 5, 0, 8, 7, 4, 6]
function search(num, array):
    for element in array:
        if element == num:
            return element
    return none
```

```
array = [ 2, 3, 1, 9, 5, 0, 8, 7, 4, 6]
function search(num, array)
    for element in array
        if element == num
            return element
    return none

```

要搜索一组无序的数据（数组），就必须遍历这组数据里面的所有元素
也就是说算法的时间复杂度T(N) = O(N)
上面的算法在最差的情况下，也就是说我们需要搜索的数是6，那么这个算法就需要循环10次（这里的数组长度为10）；同理，对于长度为N的数组我们至多循环N次，所有我们这个算法的时间复杂度T(N) = O(N)

```
array = [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
func binary_search(_ num: Int, array: Int) -> Bool
    lower = 0, upper = array.length
    middle = (lower + upper) / 2
    while lower < upper:
        if array[middle] > num {
            upper = middle
        } else if array[middle] < num {
            lower = middle
        } else {
            return true }
        middle = (lower + upper) / 2
    return false }
```

```
array = [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
function binary_search(num, array):
    for element in array:
        if element > num:
            binary_search(num, array.second_half)
        else if element < num:
            binary_search(num, array.first_half)
        else:
            return num
    return none
```

对于这个算法，我们每次进行循环的时候都会抛掉一半的数据量，也就是说在最坏的情况下，这个算法的复杂度是O(logN)，注意 log 的底数是 2 不是 10

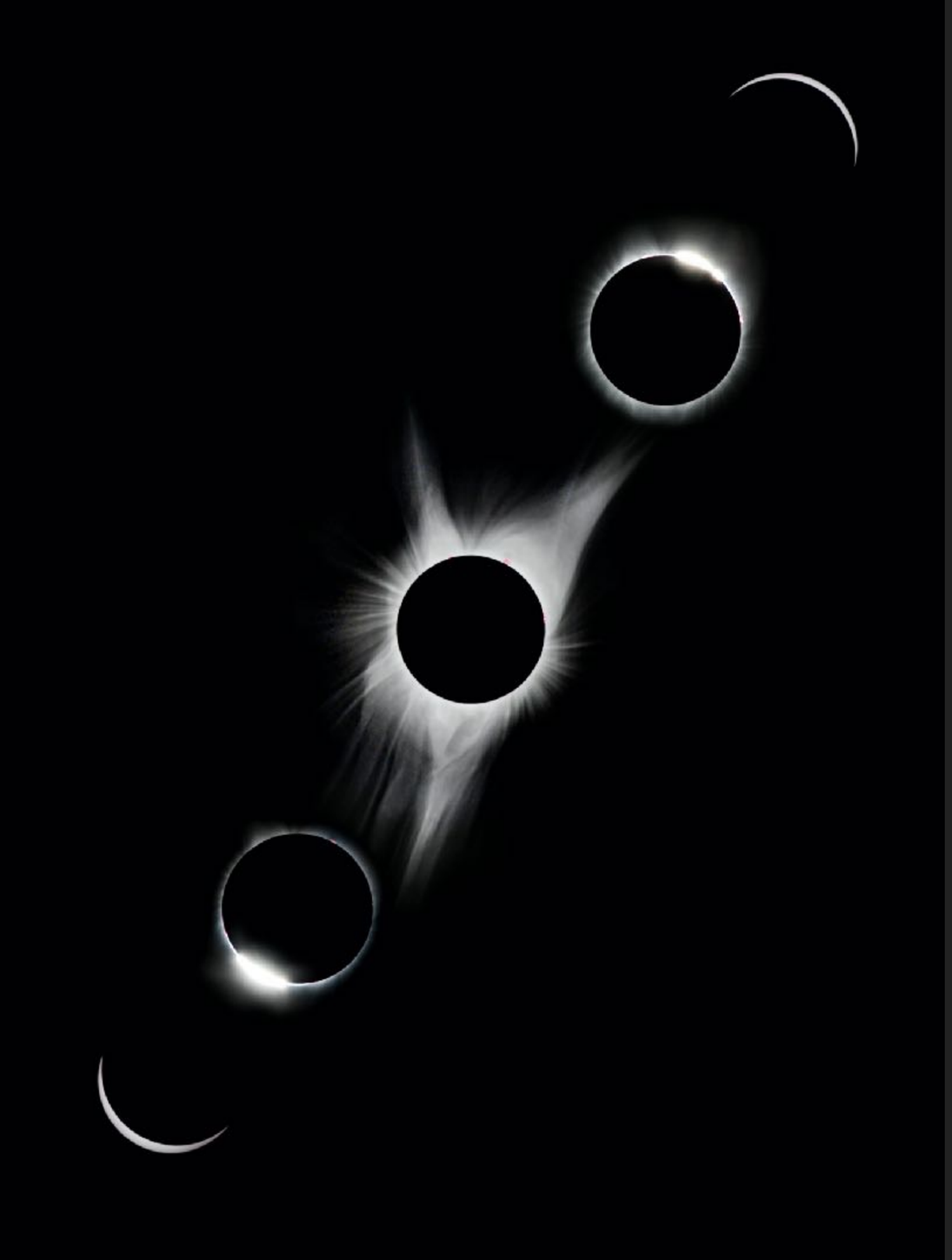
ALGORITHM AND DATA STRUCTURE

数据结构

如何创建一个可以存储「无限」长度数据的数组？

解决方案

- 我们每次在数组的容量达到限度的时候重新创建一个新的数组，并且这个数组的长度是之前数组长度加上1
  - 分析：这个数据结构过于低效，当我们的数组的容量达到限度时，如果需要添加N次数据，那么我们就需要将这个数组重新声明N次，这样太低效了
- 我们每次在数组的容量达到限度的时候重新创建一个新的数组，并且这个数组的长度是之前数组长度的两倍
  - 分析：这个数据结构可以接受，当我们的数组的容量达到限度时，如果需要添加N次数据，那么我们就需要将这个数组重新声明logN次，这样就比较不错了

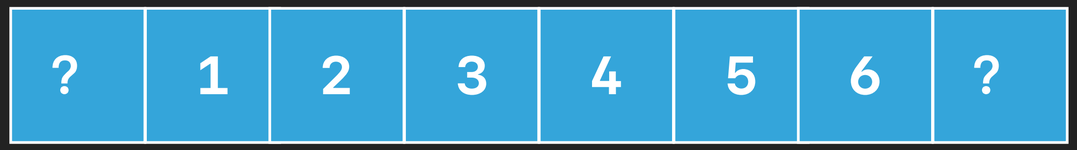


DATA STRUCTURE

链表

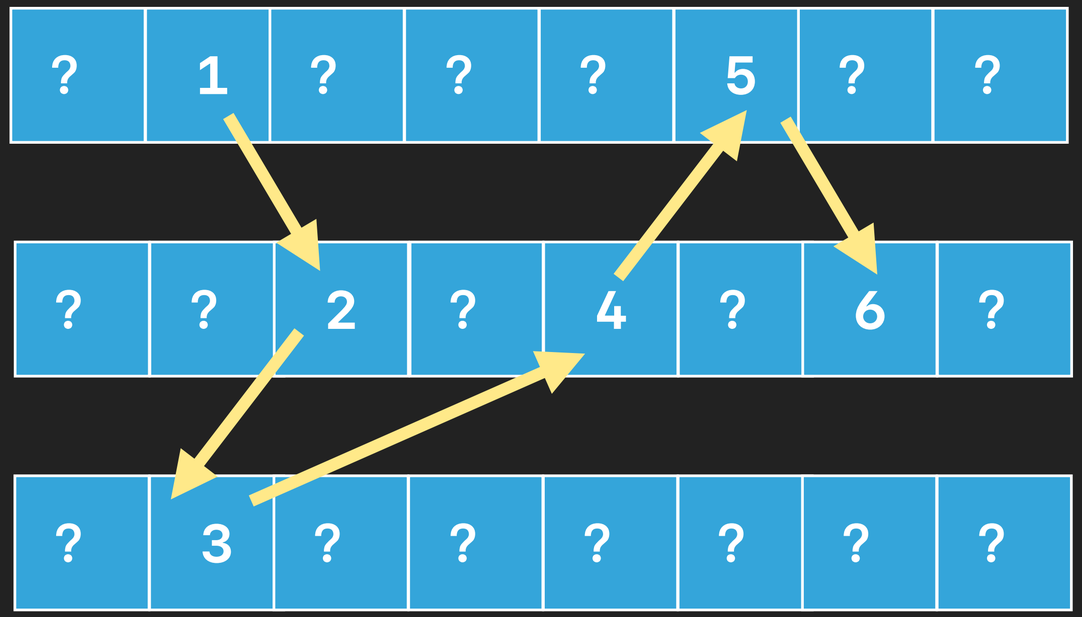
两种不同的数据结构在计算机内存的存储方式

数组



连续存储

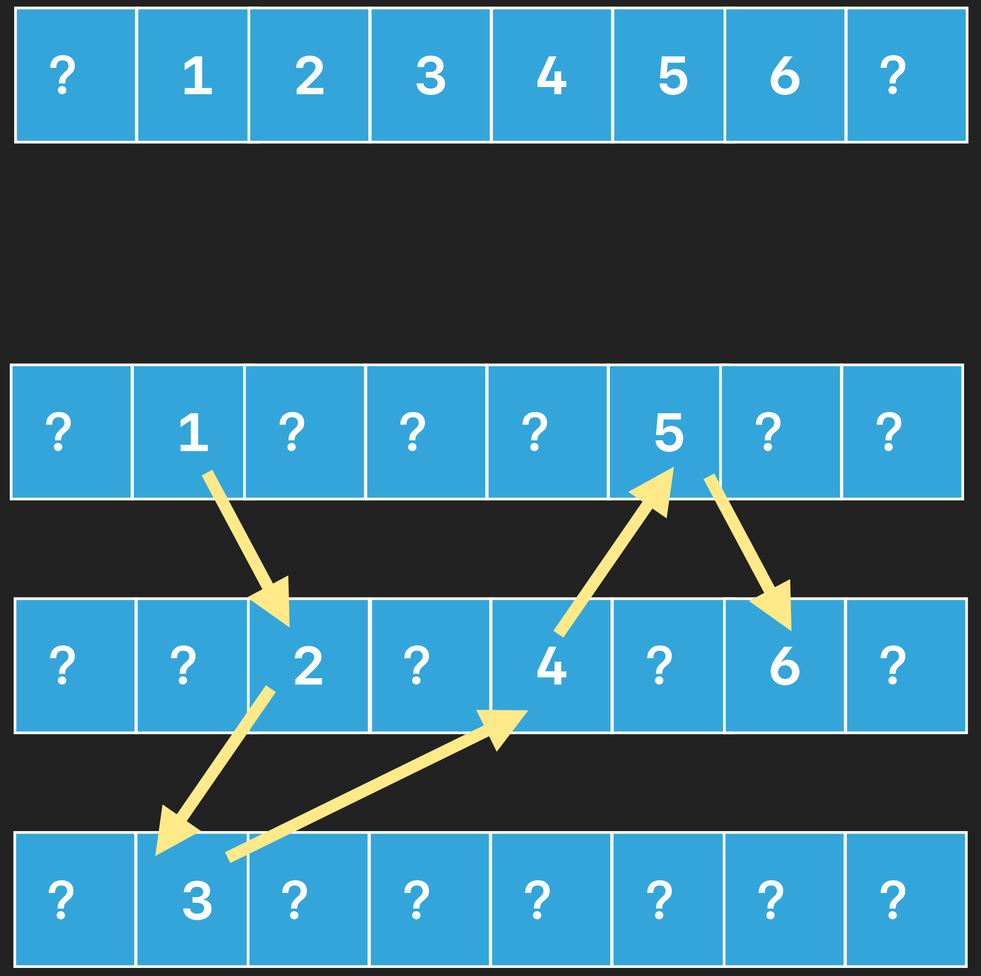
链表



离散存储

使用链表的好处

- 我们添加一个新元素的时候只需要把最后一个节点指向下一个元素所在的节点就可以了，也就是说我们可以「无限」存储一组数据，只要内存足够大。
- 我们可以很方便地删除链表中的任意一个元素。比如说我们需要删除图中的 2，我们只需要把 1 节点的箭头指向 3 就好了
- 同时，对于插入一个元素我们也可以通过改变箭头的指向来很方便的做到



DATA STRUCTURE

哈希表

简单介绍

- 我们之前提到，我们读取数组中的一个元素所需要的时间是O(1)，那么我们是否可以对这个进行一些应用呢?
  - 对于这个数组 `array = [2, 3, 1, 9, 5, 0, 8, 7, 4, 6]`
  - 如果我们需要确定这个数组是否存在**我们需要找的数** `n`，只需要这个数组转换为 `[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`，然后我们只要判断 `array[n]` 是否存在就可以了，并且这个算法的复杂度只有 `O(1)` `!!!`
  - 实际上只需要`[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]`，数字表示索引数的数目
  - binary_search
- 当然我们的这个哈希表也有缺点：遇到稀疏的数会浪费大量空间。

当前的很多高级语言，如Java，Swift，Python，C#等，或许早就把这里面的算法和数据结构（及更好的）都在语言的标准库实现了，但这不代表这些算法和数据结构就没用了。在遇到传统的编程问题的时候，我们或许可以用语言的标准库完成，但是当你遇到一些新问题的时候或者定义你自己的数据结构的时候（在编程中随处可见），这个语言的标准库实现就很难帮到你，特别是对于后者。

算法与数据结构的意义