Cha3-认识 Unity3d 的好基友 C# 08

在上一章的内容中，我们一起学习了 C#中的类、对象和方法。

而在这一课的内容中，我们将学习 C#和 Unity 之外的一个重要理论基础。掌握了这个理论基础之后，不管学习任何开发语言，都会如鱼得水，轻松自在~

让我们开始吧~

个人微信号：iseedo

微信公众号：vrlife

08 认识 Unity3d 的好基友 C#-和数据结构和算法

什么是数据结构

在之前的内容中，我们已经接触了 C#的几种基础数据类型，那么什么是数据结构呢？

维基百科之中对数据结构有一个很简单明了的定义：

a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data

具体来说，数据结构的作用是：

1.定义了一组数值的集合

2. 定义了数值之间的关系

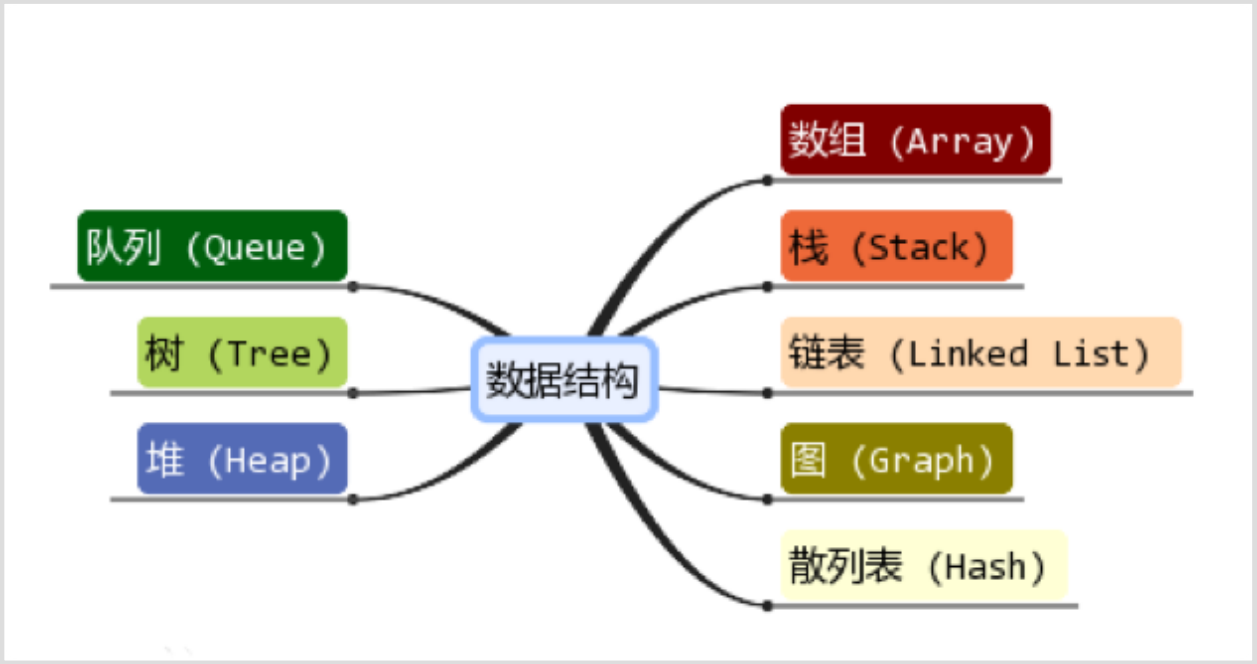
3. 通过算法，定义了对数据的操作（查找、增加、删除、修改）方式

这样解释起来相信大家都直接要懵逼了。

简单来说，假如在开发中我们需要把一组数值用某种特定的方式存储，那么我们就有了数据结

构。首先要有一组数值而不是一个数值，其次，这些数值之间存在某种关系。当然最后，我们还可以通过算法来找到或处理我们需要的数据。

在之前的学习中，其实我们已经接触了一种最简单的数据结构，也就是数组。



实际上常用的数据结构还有以下这些：

以上这八大数据结构可谓是最常见的，也是最重要的。

因为数据结构和算法属于通用的内容，这里我直接引用 CSDN 上一篇文章中关于数据结构的内容介绍（原文 https://blog.csdn.net/yeyazhishang/article/details/82353846）：

每一种数据结构都有着独特的数据存储方式，下面为大家介绍它们的结构和优缺点。

1、数组

数组可以在内存中连续存储多个元素，在内存中的分配也是连续的。

数组中的元素通过数组下标进行访问，数组下标从 0 开始。例如下面这段代码就是将数组的第一个元素赋值为 1。

```
int[] data = new int[100]；
data[0]  = 1;
```

优点：

1、按照索引查询元素速度快

2、按照索引遍历数组方便

缺点：

1、数组的大小固定后就无法扩容了

2、数组只能存储一种类型的数据

3、添加，删除的操作慢，因为要移动其他的元素。

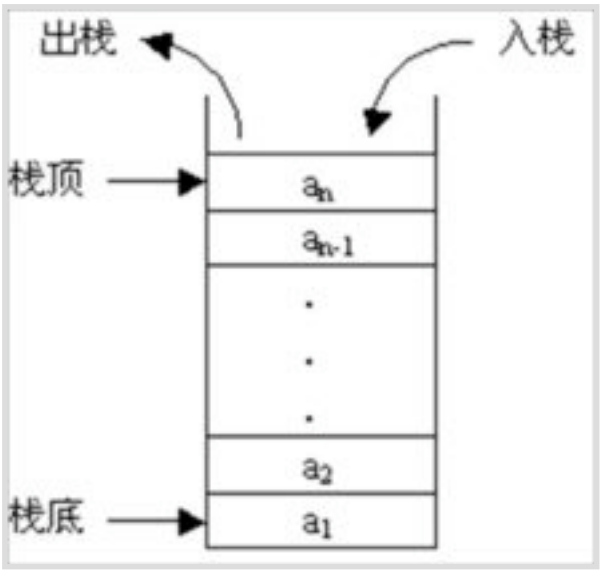
适用场景：

频繁查询，对存储空间要求不大，很少增加和删除的情况。

2、栈

栈是一种特殊的线性表，仅能在线性表的一端操作，栈顶允许操作，栈底不允许操作。 栈的特点是：先进后出，或者说是后进先出，从栈顶放入元素的操作叫入栈，取出元素叫出栈。

栈的结构就像一个箱子，越先放进去的东西越晚才能拿出来，



所以，栈常应用于实现递归功能方面的场景，例如斐波那契数列。

3、队列

队列与栈一样，也是一种线性表，不同的是，队列可以在一端添加元素，在另一端取出元素，也就是：先进先出。从一端放入元素的操作称为入队，取出元素为出队：

使用场景：因为队列先进先出的特点，在多线程阻塞队列管理中非常适用。

4、链表

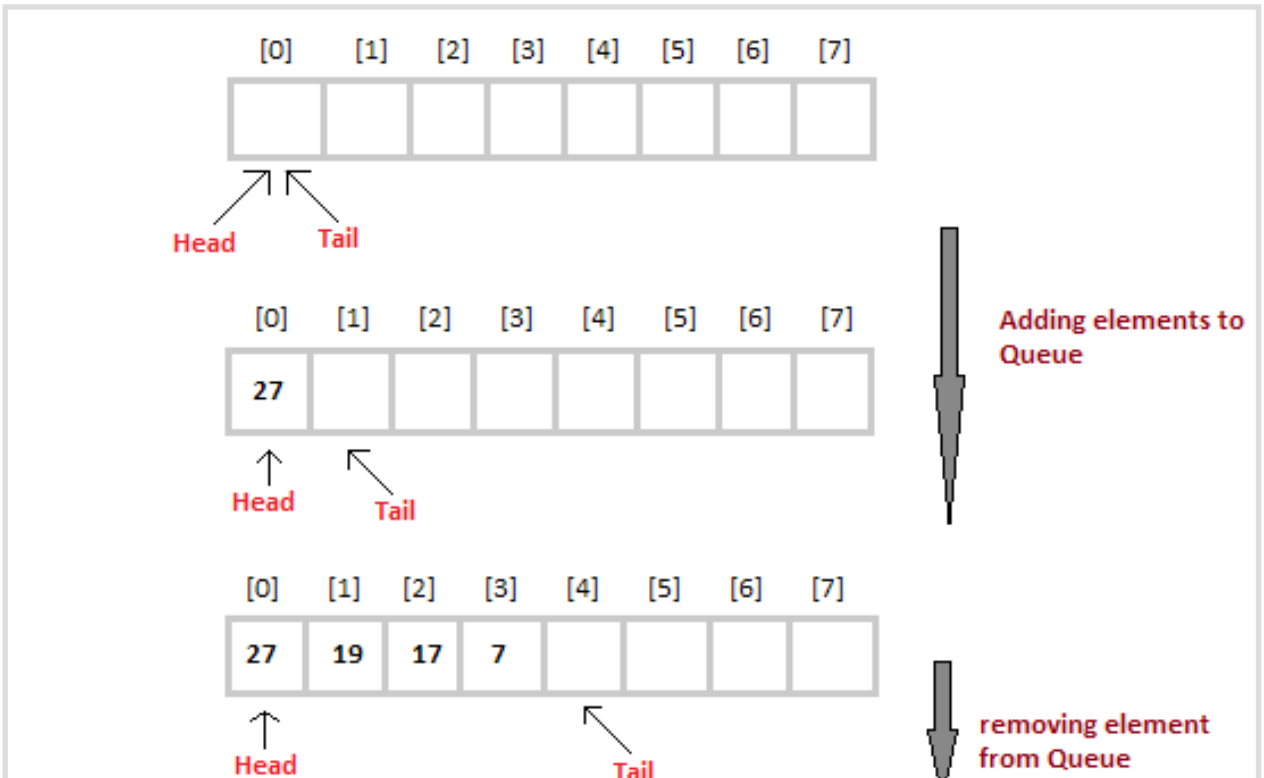
链表是物理存储单元上非连续的、非顺序的存储结构，数据元素的逻辑顺序是通过链表的指针地址实现，每个元素包含两个结点，一个是存储元素的数据域 (内存空间)，另一个是指向下一个结点地址的指针域。根据指针的指向，链表能形成不同的结构，例如单链表，双向链表，循环链表等。

链表的优点：

链表是很常用的一种数据结构，不需要初始化容量，可以任意加减元素；

添加或者删除元素时只需要改变前后两个元素结点的指针域指向地址即可，所以添加，删除很快；

缺点：



因为含有大量的指针域，占用空间较大；

查找元素需要遍历链表来查找，非常耗时。

适用场景：

数据量较小，需要频繁增加，删除操作的场景

5、树

树是一种数据结构，它是由 n（n>=1）个有限节点组成一个具有层次关系的集合。把它叫做“树”是因为它看起来像一棵倒挂的树，也就是说它是根朝上，而叶朝下的。它具有以下的特点：

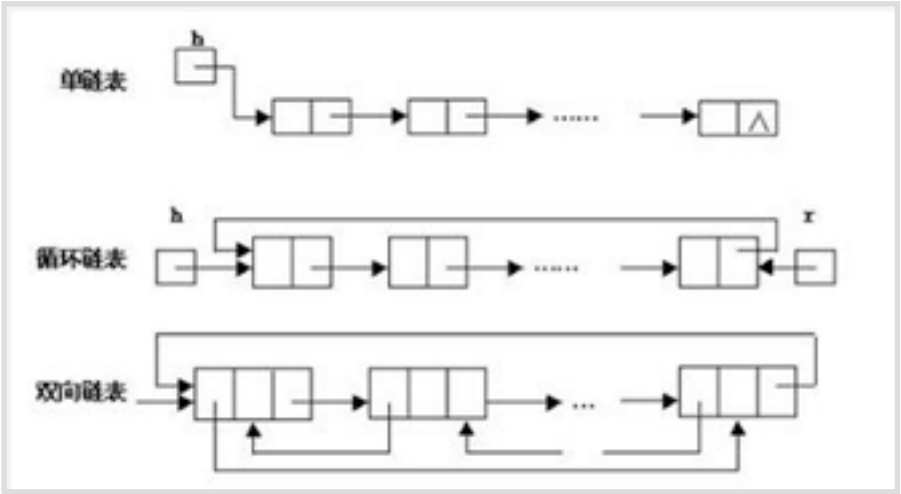
每个节点有零个或多个子节点；

没有父节点的节点称为根节点；

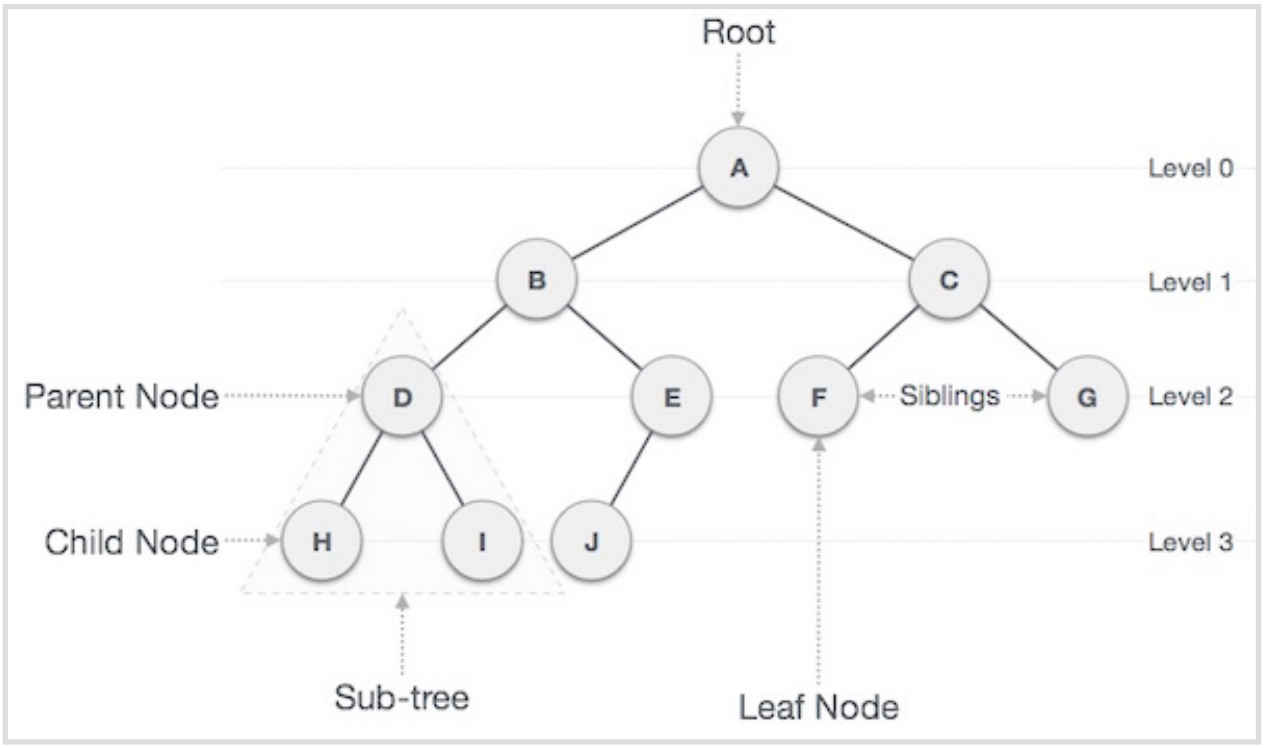
每一个非根节点有且只有一个父节点；

除了根节点外，每个子节点可以分为多个不相交的子树；

在日常的应用中，我们讨论和用的更多的是树的其中一种结构，就是二叉树。



二叉树是树的特殊一种，具有如下特点：



1、每个结点最多有两颗子树，结点的度最大为 2。

2、左子树和右子树是有顺序的，次序不能颠倒。

3、即使某结点只有一个子树，也要区分左右子树。

二叉树是一种比较有用的折中方案，它添加，删除元素都很快，并且在查找方面也有很多的算法优化，所以，二叉树既有链表的好处，也有数组的好处，是两者的优化方案，在处理大批量的动态数据方面非常有用。

扩展：

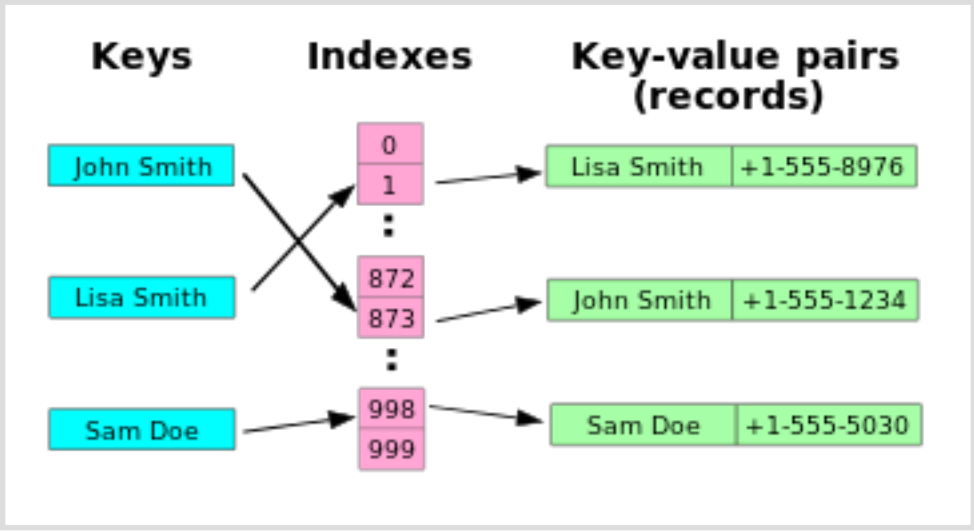
二叉树有很多扩展的数据结构，包括平衡二叉树、红黑树、B+树等，这些数据结构二叉树的基础上衍生了很多的功能，在实际应用中广泛用到，例如 mysql 的数据库索引结构用的就是 B+树，还有 HashMap 的底层源码中用到了红黑树。这些二叉树的功能强大，但算法上比较复杂，想学习的话还是需要花时间去深入的。

6、散列表

散列表，也叫哈希表，是根据关键码和值 (key 和 value) 直接进行访问的数据结构，通过 key 和 value 来映射到集合中的一个位置，这样就可以很快找到集合中的对应元素。

记录的存储位置=f(key)

这里的对应关系 f 成为散列函数，又称为哈希 (hash 函数)，而散列表就是把 Key 通过一个固定的算法函数既所谓的哈希函数转换成一个整型数字，然后就将该数字对数组长度进行取余，取余结果就当作数组的下标，将 value 存储在以该数字为下标的数组空间里，这种存储空间可以充分利用数组的



查找优势来查找元素，所以查找的速度很快。

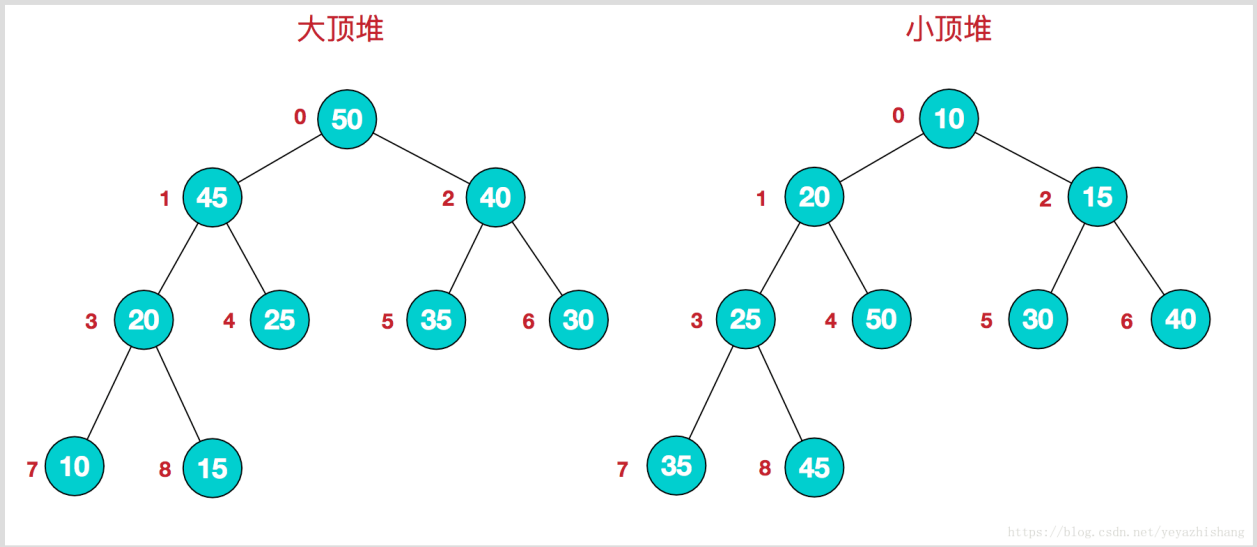
哈希表在应用中也是比较常见的，就如 Java 中有些集合类就是借鉴了哈希原理构造的，例如 HashMap，HashTable 等，利用 hash 表的优势，对于集合的查找元素时非常方便的。

哈希表的应用场景很多，当然也有很多问题要考虑，比如哈希冲突的问题，如果处理的不好会浪费大量的时间，导致应用崩溃。

7、堆

堆是一种比较特殊的数据结构，可以被看做一棵树的数组对象，具有以下的性质：

堆中某个节点的值总是不大于或不小于其父节点的值；



堆总是一棵完全二叉树。

将根节点最大的堆叫做最大堆或大根堆，根节点最小的堆叫做最小堆或小根堆。常见的堆有二叉堆、斐波那契堆等。

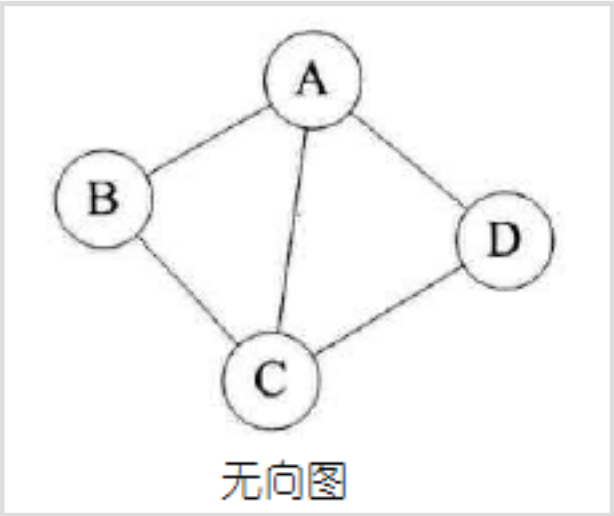
堆的定义如下：n 个元素的序列{k1,k2,ki,…,kn}当且仅当满足下关系时，称之为堆。

(ki <= k2i,ki <= k2i+1)或者(ki >= k2i,ki >= k2i+1), (i = 1,2,3,4…n/2)，满足前者的表达式的成为小顶

堆，满足后者表达式的为大顶堆，这两者的结构图可以用完全二叉树排列出来，示例图如下：

因为堆有序的特点，一般用来做数组中的排序，称为堆排序。

8、图



图是由结点的有穷集合 V 和边的集合 E 组成。其中，为了与树形结构加以区别，在图结构中常常将结点称为顶点，边是顶点的有序偶对，若两个顶点之间存在一条边，就表示这两个顶点具有相邻关系。

按照顶点指向的方向可分为无向图和有向图：

图是一种比较复杂的数据结构，在存储数据上有着比较复杂和高效的算法，分别有邻接矩阵 、邻接表、十字链表、邻接多重表、边集数组等存储结构。

什么是算法

说完了数据结构，再来看看什么是算法。

算法，顾名思义，计算的方法，其实就是当我们用计算机来解决某个问题的时候，用于具体实现的思路。

举个日常生活中最常用的算法，当我们驾车（当然是黄金马车了~）或者步行从弥林赶往临冬城的时候，为了尽快抵达目的地，我们不仅仅要考虑各个城市之间道路的连通性和距离，还需要考虑哪些城市暂时是无法通过的（敌军掌握中）等等因素。那么如规划一条最优路径，就需要用到算法了。

这里再举个更简单的例子：

想象在一个密封的盒子里面有一堆糖果，其中只有 1 个糖果是红色的外包装，其它都是金色的，那么如何快速从盒子里面找到这颗红色糖果？

其实大家立马就会想到两种算法：

算法 1：顺序查找法，一个个拿出来看

算法 2：折半查找法，每次抓一半的糖果出来，看里面有没有，如果没有，再抓另一半。

显然，折半查找法比顺序查找法的效率要高很多，除非你的运气逆天了~

很多时候，同一个问题可以用不同的 算法来解决，而算法的质量优劣将直接影响到整个程序的效率。对算法的评价主要从时间复杂度和空间复杂度来考虑。

所谓的时间复杂度指的是执行算法所需要的计算工作量，而空间复杂度指的是算法需要消耗的内存空间。

但是不管时间复杂度和空间复杂度如何，其实对一个算法最重要的是，它需要是对的~

几种经典的算法思路如下：

递推法

递推是序列计算机中的一种常用算法。它是按照一定的规律来计算序列中的每个项，通常是通过计算机前面的一些项来得出序列中的指定项的值。其思想是把一个复杂的庞大的计算过程转化为简单过程的多次重复，该算法利用了计算机速度快和不知疲倦的机器特点。

递归法

程序调用自身的编程技巧称为递归（recursion）。一个过程或函数在其定义或说明中有直接或间接调用自身的一种方法，它通常把一个大型复杂的问题层层转化为一个与原问题相似的规模较小的问题来求解，递归策略只需少量的程序就可描述出解题过程所需要的多次重复计算，大大地减少了程序的代码量。递归的能力在于用有限的语句来定义对象的无限集合。一般来说，递归需要有边界条件、递归前进段和递归返回段。当边界条件不满足时，递归前进；当边界条件满足时，递归返回。

注意：

(1) 递归就是在过程或函数里调用自身;

(2) 在使用递归策略时，必须有一个明确的递归结束条件，称为递归出口。

穷举法

穷举法，或称为暴力破解法，其基本思路是：对于要解决的问题，列举出它的所有可能的情况，逐个判断有哪些是符合问题所要求的条件，从而得到问题的解。它也常用于对于密码的破译，即将密码进行逐个推算直到找出真正的密码为止。例如一个已知是四位并且全部由数字组成的密码，其可能共有 10000 种组合，因此最多尝试 10000 次就能找到正确的密码。理论上利用这种方法可以破解任何一种密码，问题只在于如何缩短试误时间。因此有些人运用计算机来增加效率，有些人辅以字典来缩小密码组合的范围。

贪心算法

贪心算法是一种对某些求最优解问题的更简单、更迅速的设计技术。

用贪心法设计算法的特点是一步一步地进行，常以当前情况为基础根据某个优化测度作最优选择，而不考虑各种可能的整体情况，它省去了为找最优解要穷尽所有可能而必须耗费的大量时间，它采用自顶向下,以迭代的方法做出相继的贪心选择,每做一次贪心选择就将所求问题简化为一个规模更小的子问题, 通过每一步贪心选择,可得到问题的一个最优解，虽然每一步上都要保证能获得局部最优解，但由此产生的全局解有时不一定是最优的，所以贪婪法不要回溯。

贪婪算法是一种改进了的分级处理方法，其核心是根据题意选取一种量度标准，然后将这多个输入排成这种量度标准所要求的顺序，按这种顺序一次输入一个量，如果这个输入和当前已构成在这种量度意义下的部分最佳解加在一起不能产生一个可行解，则不把此输入加到这部分解中。这种能够得到某种量度意义下最优解的分级处理方法称为贪婪算法。

对于一个给定的问题，往往可能有好几种量度标准。初看起来，这些量度标准似乎都是可取的，但实际上，用其中的大多数量度标准作贪婪处理所得到该量度意义下的最优解并不是问题的最优解，而是次优解。因此，选择能产生问题最优解的最优量度标准是使用贪婪算法的核心。

一般情况下，要选出最优量度标准并不是一件容易的事，但对某问题能选择出最优量度标准后，用贪婪算法求解则特别有效。

分治法

分治法是把一个复杂的问题分成两个或更多的相同或相似的子问题，再把子问题分成更小的子问题……直到最后子问题可以简单的直接求解，原问题的解即子问题的解的合并。

分治法所能解决的问题一般具有以下几个特征：

(1) 该问题的规模缩小到一定的程度就可以容易地解决；

(2) 该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质；

(3) 利用该问题分解出的子问题的解可以合并为该问题的解；

(4) 该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子子问题。

动态规划法

动态规划是一种在数学和计算机科学中使用的，用于求解包含重叠子问题的最优化问题的方法。其基本思想是，将原问题分解为相似的子问题，在求解的过程中通过子问题的解求出原问题的解。动态规划的思想是多种算法的基础，被广泛应用于计算机科学和工程领域。

动态规划程序设计是对解最优化问题的一种途径、一种方法，而不是一种特殊算法。不象前面所述的那些搜索或数值计算那样，具有一个标准的数学表达式和明确清晰的解题方法。动态规划程序设计往往是针对一种最优化问题，由于各种问题的性质不同，确定最优解的条件也互不相同，因而动态规划的设计方法对不同的问题，有各具特色的解题方法，而不存在一种万能的动态规划算法，可以解决各类最优化问题。因此读者在学习时，除了要对基本概念和方法正确理解外，必须具体问题具体分析处理，以丰富的想象力去建立模型，用创造性的技巧去求解。

迭代法

迭代法也称辗转法，是一种不断用变量的旧值递推新值的过程，跟迭代法相对应的是直接法（或者称为一次解法），即一次性解决问题。迭代法又分为精确迭代和近似迭代。“二分法”和“牛顿迭代

法”属于近似迭代法。迭代算法是用计算机解决问题的一种基本方法。它利用计算机运算速度快、适合做重复性操作的特点，让计算机对一组指令（或一定步骤）进行重复执行，在每次执行这组指令（或这些步骤）时，都从变量的原值推出它的一个新值。

分支界限法

分枝界限法是一个用途十分广泛的算法，运用这种算法的技巧性很强，不同类型的问题解法也各不相同。

分支定界法的基本思想是对有约束条件的最优化问题的所有可行解（数目有限）空间进行搜索。该算法在具体执行时，把全部可行的解空间不断分割为越来越小的子集（称为分支），并为每个子集内的解的值计算一个下界或上界（称为定界）。在每次分支后，对凡是界限超出已知可行解值那些子集不再做进一步分支，这样，解的许多子集（即搜索树上的许多结点）就可以不予考虑了，从而缩小了搜索范围。这一过程一直进行到找出可行解为止，该可行解的值不大于任何子集的界限。因此这种算法一般可以求得最优解。

与贪心算法一样，这种方法也是用来为组合优化问题设计求解算法的，所不同的是它在问题的整个可能解空间搜索，所设计出来的算法虽其时间复杂度比贪婪算法高，但它的优点是与穷举法类似，都能保证求出问题的最佳解，而且这种方法不是盲目的穷举搜索，而是在搜索过程中通过限界，可以中途停止对某些不可能得到最优解的子空间进一步搜索（类似于人工智能中的剪枝），故它比穷举法效率更高。

回溯法

回溯法（探索与回溯法）是一种选优搜索法，按选优条件向前搜索，以达到目标。但当探索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择，这种走不通就退回再走的技术为回溯法，而满足回溯条件的某个状态的点称为“回溯点”。

其基本思想是，在包含问题的所有解的解空间树中，按照深度优先搜索的策略，从根结点出发深度探索解空间树。当探索到某一结点时，要先判断该结点是否包含问题的解，如果包含，就从该结点出发继续探索下去，如果该结点不包含问题的解，则逐层向其祖先结点回溯。（其实回溯法就是对隐式图的深度优先搜索算法）。 若用回溯法求问题的所有解时，要回溯到根，且根结点的所有可行的子树都要已被搜索遍才结束。 而若使用回溯法求任一个解时，只要搜索到问题的一个解就可以结束。

当然，上面提到的只是抽象的算法实现思路，具体的算法要根据具体的问题进行分析和判断，并参考现有的经典算法。

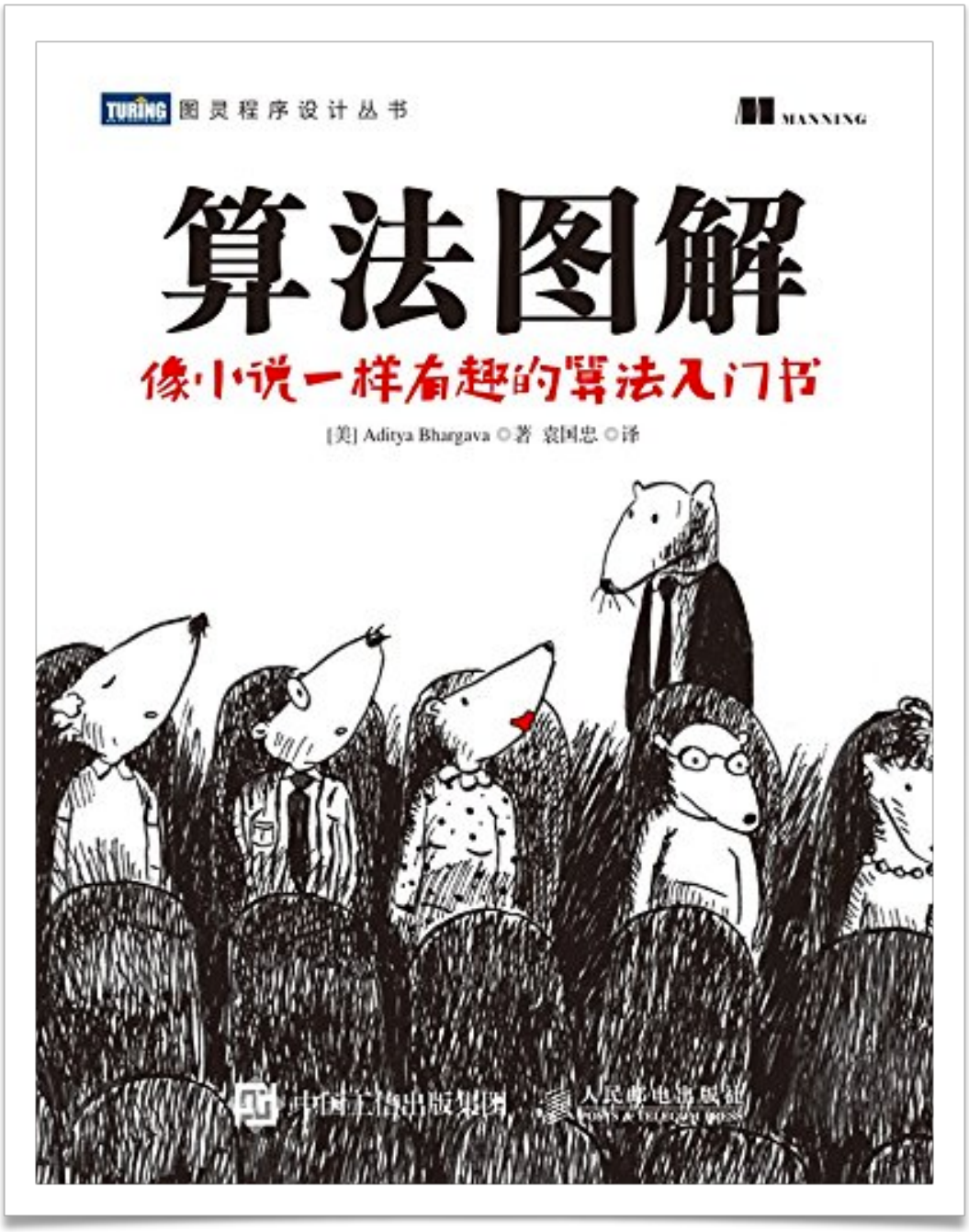
数据结构和算法的知识无比重要，但是也相对比较晦涩难懂。

限于篇幅和主题，这里不打算就这个话题展开了讲，否则光这个话题就可以单独开一门课了~

这里给大家安利一本书，非常适合初学入门的童鞋看~

相信看完这本书之后，你一定会对算法和数据结构有一个基础的和全面的认识。

购买链接：https://item.jd.com/12148832.html



强烈推荐大家在学习之余读完这本书，因为它不光对学习 Unity3d 游戏开发有用，对所有的编程开发都有极大的帮助~

好了，关于数据结构和算法的扫盲就到此结束了。

到目前为止，虽然我们已经掌握了关于 C#的很多知识，但是有点隔靴搔痒的味道。至少我们只能在 Console 视图中看到一些单调的调试信息。

而从下一课开始，我们将学习如何在 Unity 游戏项目中实际应用 C#，让游戏对象之间产生互动。

让我们下一课再见~