## 一句话算法

本系列搜集的并不是完整的算法题，而是从算法题中萃取出来的算法关键点的思路。

很多人平时并不会刷题，而如果只在面试前临时刷题，只能是找找感觉，像我这样记性不好的人没过多久就又会忘记。

但是如果只是记住关键点，就简单很多，在遇到算法题后，适当的做分解，就能有大致地思路，茅塞顿开，大概率就能把题解出来。

[一句话算法](https://github.com/pwstrick/daily/blob/master/article/one/one.md)选取的题目来源于[力扣](https://leetcode-cn.com/)、[牛客网](https://www.nowcoder.com/)、《[前端程序员算法宝典](https://book.douban.com/subject/34451262/)》、《[剑指offer](https://book.douban.com/subject/27008702/)》、[小浩算法](https://www.geekxh.com)、技术博文等渠道。

每个问题后会给出至少一种解题思路，部分给出了具体的代码实现，其标题可点击。

### 经典算法

1. 杨辉三角形
   * 公式 C(n+1,i) = C(n,i) + C(n,i-1)
2. 百钱买百鸡
   * 设有 i 只公鸡，j 只母鸡，k 只小鸡，并且 i+j+k 的总数为 100，i×5+j×3+k/3 总价为 100
3. 不使用临时变量，交互两个变量的值
   * 解构，[a,b] = [b,a]
   * 加法，a=a+b; b=a-b; a=a-b
   * 异或，a^=b; b^=a; a^=b
4. 打印从1到最大的n位数
   * n可能是大数，得用字符串或数组表示大数

### 排序

1. [冒泡排序](https://github.com/pwstrick/daily/issues/982)
   * 从第一个记录开始依次对相邻的两个记录进行比较，当前面的记录大于后面的记录时，交换其位置
2. [插入排序](https://github.com/pwstrick/daily/issues/983)
   * 假设第一个记录自成一个有序序列，从第二个记录开始，按照记录的大小依次将当前记录插到有序序列中
3. [归并排序](https://github.com/pwstrick/daily/issues/984)
   * 利用递归与分治技术将序列划分成越来越小的半子表，再对半子表排序，最后用递归将排好序的半子表合并成为越来越大的有序序列
4. [快速排序](https://github.com/pwstrick/daily/issues/985)
   * 将序列分为两部分，前一部分的所有记录均比后一部分的所有记录小，然后再依次对前后两部分的记录进行快速排序，递归该过程
5. [选择排序](https://github.com/pwstrick/daily/issues/986)
   * 比较得到最小的记录，然后将它与第一个记录进行位置交换，接着不包含第一个记录比较次小的记录，重复该过程
6. [希尔排序](https://github.com/pwstrick/daily/issues/987)
   * 将序列分成多个子序列，然后对各个子序列分别进行直接插入排序，再对所有元素进行一次直接插入排序
7. [堆排序](https://github.com/pwstrick/daily/issues/988)
   * 将序列看成一棵顺序存储的二叉树，然后调整成一个大顶堆，再将堆的最后一个元素与堆顶元素（即根结点）进行交换
8. [桶排序](https://github.com/pwstrick/daily/issues/989)
   * 将数据分到有限数量的桶里，每个桶再分别排序，有可能使用别的排序算法或是以递归方式继续使用桶排序
9. [计数排序](https://github.com/pwstrick/daily/issues/990)
   * 只能对整数排序，使用额外的countArr数组，其中第 i 个元素是序列中值等于 i 的元素个数，再根据countArr对元素排序

### 数组

数组是某种类型的数据按照一定的顺序组成的数据集合。

1. [将 1～1000 放在含有 1001 个元素的数组中，找出唯一的重复元素](https://github.com/pwstrick/daily/issues/991)
   * Hash法，将数组中的元素映射到Hash数组中，并将值赋为 1，当元素映射的Hash数组处的值为1时，它就是重复元素
   * 累加求和法，将数组中的所有元素相加，然后减去 N 个元素的和，得到的差即为重复元素
   * 异或法，相同元素异或为 0。将数组的所有元素进行一次异或，其结果再与 1~N 进行异或，最终结果为重复元素
2. 第 K 小的数
   * 排序法，在排序后的数组中，下标为 K-1 的值就是第K小的数
3. [连续最大和](https://github.com/pwstrick/daily/issues/992)
   * 双重循环并重复利用已经计算的子数组和，Sum[i] = max(arr[i], Sum[i-1] + arr[i])
4. 循环移位
   * 空间换时间，把A数组的第 (n-k+1)~n 位的元素存储到辅助数组T中，再把A中余下的元素也存储到T中，最后将T中的元素复制回A
5. 只有一个元素出现一次，其余元素均出现三次，找出那个只出现一次的元素
   * 把原数组去重、再乘以 3 得到的值，刚好就是要找的元素的 2 倍

### 数字运算

1. [判断一个自然数是否是某个数的二次方](https://github.com/pwstrick/daily/issues/993)
   * 二分查找法，判断 mid=(1+n)/2 的二次方 power 与 m 的大小；如果 power＞m，那么就说明要在 [1, mid-1] 区间继续查找；否则就在 [mid+1, n] 区间继续查找
2. [判断一个数是否为 2 的 n 次方](https://github.com/pwstrick/daily/issues/994)
   * 对 1 执行移位操作，每次左移一位，即通过移位得到的值必定是 2 的 n 次方
   * 2 的 n 次方对应的二进制中只有一位是 1，并且 num&(num-1) 运算结果是 0
3. [不使用除号实现整数相除](https://github.com/pwstrick/daily/issues/995)
   * 使被除数不断减去除数，直到相减的结果小于除数为止
4. [用递增运算符（++）实现加减乘除运算](https://github.com/pwstrick/daily/issues/996)
   * 加法，a+b 即对 a 执行 b 次递增操作
   * 减法，a-b（a≥b）即不断地对 b 执行递增操作，直到等于 a 为止，记录执行递增操作的次数
   * 乘法，a×b 即利用已经实现的加法操作把 a 相加 b 次，就得到了 a×b 的积
   * 除法，a/b 即利用乘法，使 b 不断乘以 1，2，…，n，直到 b×n＞b 时，就可以得到商 n-1
5. [比较两个数的大小（不能使用比较运算符和if语句）](https://github.com/pwstrick/daily/issues/997)
   * 绝对值法，如果 |a-b|==a-b，那么 max(a,b)=a；否则 max(a,b)=b
   * 二进制法，如果 a＞b，那么 a-b 的二进制最高位为 0，与任何数执行与操作的结果还是 0
6. [求二进制数中 1 的个数](https://github.com/pwstrick/daily/issues/998)
   * 移位法，当这个数的最后一位是 1 时，则计数器加 1。然后右移丢弃最后一位，循环该操作
   * 每进行一次 n&(n-1) 计算，其结果中都会少了一位 1，不断循环直至为 0，记录操作次数
7. [不使用循环输出 1～100](https://github.com/pwstrick/daily/issues/999)
   * 递归自身，每次加 1
8. [求最小公倍数](https://github.com/pwstrick/daily/issues/1000)
   * 最小公倍数 = 两整数的乘积 ÷ 最大公约数。求最大公约数的辗转相除法，a%b 得余数 c，若 c=0，那么 b 是最大公约数，否则 a=b，b=c，再重复执行

### 链表

存储特点：可以用任意一组存储单元来存储单链表中的数据元素（存储单元可以是不连续的）。

1. [链表逆序](https://github.com/pwstrick/daily/issues/1001)
   * 从链表的第二个结点开始，把遍历到的结点插入到头结点的后面，直至结束，head→1→2→3→4，变为 head→2→1→3→4
2. [移除重复项](https://github.com/pwstrick/daily/issues/1002)
   * 利用双重循环，外层循环用指针从第一个结点开始遍历，内层循环用另外一个指针遍历其余结点，删除外层循环中数据相同的结点
3. [找出链表倒数第 K 个结点](https://github.com/pwstrick/daily/issues/1003)
   * 使用两个指针，快指针比慢指针先前移 K 步，然后两个指针同时移动。当快指针到底后，慢指针的位置就是所要找的结点
4. [单链表是否有环](https://github.com/pwstrick/daily/issues/1004)
   * 使用两个指针，快指针每次前移两步，慢指针前移一步，当两个指针指向相同结点时，就证明有环
5. [相邻元素翻转](https://github.com/pwstrick/daily/issues/1005)
   * 调换相邻两个结点指针的指向，如果链表有奇数个结点，那么除最后一个外的其他结点进行奇偶对调
6. [合并两个有序链表](https://github.com/pwstrick/daily/issues/1006)
   * 用两个指针遍历两个链表，如果head1指向的数据小于head2的，则将head1指向的结点归入合并后的链表中，否则用head2的

### 栈和队列

栈就像一个很窄的桶，先存进去的数据只能最后被取出来，是 LIFO 结构（Last In First Out，后进先出）。

队列像日常排队买东西的人的队列，先排队的人先买，后排队的人后买，是 FIFO 结构（First In First Out，先进先出）。

1. [翻转栈中的元素](https://github.com/pwstrick/daily/issues/1007)
   * 申请一个额外的队列，先把栈中的元素依次出栈放到队列里，然后把队列里的元素按照出队的顺序入栈
2. [O(1) 求栈的最小元素](https://github.com/pwstrick/daily/issues/1008)
   * 使用两个栈结构，一个存储数据，一个存储最小元素，在入栈存储时，判断是否是最小元素
3. [用两个栈实现队列](https://github.com/pwstrick/daily/issues/1009)
   * 栈 A 提供入队的功能，栈 B 出队的功能。若 B 不为空，则弹出 B 的数据；若 B 为空，则弹出 A 的数据放入 B 中，再弹出 B 的数据

### 二叉树

二叉树是 n（n≥0）个有限元素的集合，该集合或者为空，或者由一个称为根（root）的元素及两个不相交的、被分别称为左子树和右子树的二叉树组成。

1. [实现二叉树](https://github.com/pwstrick/daily/issues/1010)
   * 用数组构造，数组的键名代表二叉树的结点，数组的键值代表二叉树的结点值
   * 用链表构造，在链表中创建根结点、左子树和右子树
2. 把有序整数数组放到二叉树中
   * 取数组的中间元素作为根结点，将数组分成两部分，对数组的两部分用递归分别构建左右子树
3. 查找最大值和最小值
   * 在初始化时，左子树比父节点小，右子树要大，最小值和最大值分别从左子树和右子树中递归查找
4. [遍历](https://github.com/pwstrick/daily/issues/1011)
   * 前序，先访问当前结点，然后访问左子树，再访问右子树
   * 中序，先访问左子树，然后访问当前结点，再访问右子树
   * 后序，先访问左子树，然后访问右子树，再访问当前结点
   * 层序，自上而下，自左至右逐层访问树的结点
5. 最大深度
   * 递归，每个结点的深度等于其左右子树最大深度加 1
6. DFS
   * 深度优先搜索，尽可能深的搜索树的分支，一直进行到源结点可达的所有结点为止
7. [BFS](https://github.com/pwstrick/daily/issues/1038)
   * 广度优先搜索，从上到下先把每一层遍历完之后再遍历下一层
8. [平衡二叉树](https://github.com/pwstrick/daily/issues/1039)
   * 二叉树每个结点的左右子树的高度差的绝对值不超过 1
9. 满二叉树
   * 所有结点都存在左右子树，并且所有叶子结点都在同一层上
10. 完全二叉树
    * 除了叶子结点之外，每个结点的度（结点所拥有的子树个数）都为 2，满二叉树肯定是完全二叉树

### 动态规划（DP）

思想的本质是求一个最优解，也就是将一个规模比较大的问题，分解成若干规模较小的问题，再从下往上先计算小问题的最优解并存储下来，然后以此为基础求取大问题的最优解。需要能推倒出状态转移方程。

1. [最大和的连续子数组](https://www.geekxh.com/1.2.%E5%8A%A8%E6%80%81%E8%A7%84%E5%88%92%E7%B3%BB%E5%88%97/202.html#_01%E3%80%81%E9%A2%98%E7%9B%AE%E5%88%86%E6%9E%90)
   * 状态转移方程：dp[i] = max(arr[i], dp[i−1]+arr[i])，其中dp[i]表示以arr[i]结尾的连续子数组的最大和
2. [在二维网格中找出一条从左上角到右下角的最小路径和](https://www.geekxh.com/1.2.%E5%8A%A8%E6%80%81%E8%A7%84%E5%88%92%E7%B3%BB%E5%88%97/205.html)
   * 状态转移方程：dp[i][j] = min(dp[i-1][j],dp[i][j-1]) + grid[i][j]，其中dp[i][j]表示包含第 i 行 j 列元素的最小路径和
3. 斐波那契数列
   * 状态转移方程：dp[i] = dp[i-1] + dp[i-2]（i>=2），其中dp[i]表示第 i 个斐波那契数