编程问题

（2-sum problem）给定一个无序数组 A，给定一个和值 sum，找出数组中所有相加等于 A 的数对。

两种解决思路，一种是优化时间的，一种是优化空间的：

1.优化时间：时复O(N) 空复O(M)，M是数组中值的范围

使用一个 HashMap，将整个数组的所有数值都 hash。之后遍历数组一遍，对每一个 A[i] 都检查 sum-A[i] 是否存在于 HashMap 中。

NB. 为了使的遍历时的查找操作是真正的 O(1) 时间，我们需要使的冲突非常小（理想状况下没有冲突），这对 hash 函数选取有较高的要求。如果希望 hash 函数有简单的实现，直接的方法是让 HashMap 的大小为数组中值的范围 M=max(A)-min(A)，此时空间占用是 O(M)。只有此时才能绝对保证整个算法的时复是 O(N)。

2.优化空间：时复O(NlogN)，空复O(1)

对数组调用 quick sort，耗时 O(NlogN)，之后采取两个下标 i=0, j=len(A)-1 两个相加小于 sum，i推进，两个相加大于 sum，j推进，直到 i，j 交错。

(3-sum problem) 给定一个无序数组 A，给定一个和值 sum，找出数组中所有相加等于 A 的三元组（triple）。

O(N^2) 解决思路1：

1. 先对数组进行排序，耗时 O(NlogN)，之后遍历数组中的每一个元素（O(N)），每次遍历都用两个指针 i，j 采用 2-sum 中的首尾逼近法去寻找是否存在 sum-A[i]-A[j]，这个过程耗时是 O(N)。总共耗时 O(N^2)

O(N^2) 解决思路2：

1. 先使用 hashtable 对整个数组进行散列，构建的过程时间是 O(N)，空间占用是 O(N).
2. 之后用两个下标 i，j 两层循环遍历数组，在内部利用 hashtable 去查找是否存在 sum-A[i]-A[j]，查找开销是 O(1)
3. 好处是时间快了一点，但是空间开销是 O(N)，如果要 hashtable 达到严格 O(1) 查找操作，空间开销可能要 O(M)

拓展到 4sum 问题

先全部排序，然后用 i 遍历元素，每次对 i+1 及之后的元素使用 3sum

给定链表的头指针，和一个节点指针，在 O(1) 时间删除该节点

```
node.data = node.next.data
node.next = node.next.next
```

用下一个节点的数据直接替换该节点的数据。注意给定的不能是尾节点

给定一个单链表 head，输出逆转之后的反序单链表

使用三个指针 pre，current，next 遍历一遍即可

```
def reverse(head):
     if head == None or head.next == None:
          return head

     pre = None
     current = head
     next = None

     while current != None:
          next = current.next
          current.next = pre
          pre = current
          current = next

     return pre
```

给定一个单向链表，要求找到倒数第 k 个节点?
设置两个引用 p1，p2，一开始都指向 head，然后 p2 向前走 k 个节点，此时 p1 和 p2 之间相隔了 k 个节点。
此后 p1，p2 同时向前每次前进一个节点，直到 p2 走到链表末端。

求链表的中间节点
先计数

先用一个节点遍历一次列表，记录长度为 count，之后问题变为求第 count/2 位置的节点。

只遍历一遍的方法

如果只能遍历一遍，则采用两个指针，一个 fast 每次走两步，一个 slow 每次走一步。直到快的指针移动到尾节点的时候，慢的指针指的位置就是我们要找的链表中间位置。

给定一个链表，查找是否有环
通过两个指针，分别从链表的头节点出发，一个每次向后移动一步，另一个移动两步，两个指针移动速度不一样，如果存在环，那么两个指针一定会在环里相遇。

```
def hasCircile(head):
     fast = head
     slow = head

     while fast != None and fast.next != None:
          fast = fast.next.next
          slow = slow.next
          if fast == slow:
               return True
     return False
```

如果找出环的入口位置?
当找到环的时候，把 fast 重新指向 head，之后 fast 和 slow 都同时按每次一步走，最后两者再相遇的时候，就是环的入口。
为什么?
假设第一次相遇的时候，从 head 到环的入口共有 a 步，从进入环的入口到相遇点共有 b 步，设环路长度为 L，则：
对 slow: a + b = n
对 fast: a + b + kL = 2n
可知 kL = n = a + b
由 kL = n 可知 slow 走的步数正好是环长的整数倍，此时 slow 已经在环里，如果它再走 n 步的话，还会回到相遇点
由 kL = a + b 可知，从 head 走出 n 步的话，会达到相遇点，并且同时在走的 slow 也会走到这里。

可知 fast 和 slow 只有前 a 步不同，故相遇的时候是入口。

两个链表寻找公共节点
两层遍历：
使用两层循环来遍历两个链表，看是否能走到一样的节点。耗时 O(mn)

使用标记数组：
修改节点的数据结构，添加一个标记位：visited。遍历第一个列表，全部标注为 true，再访遍历第二个列表。耗时 O(m+n)，占用空间 O(m+n)，而且要求修改数据结构。
如果不能修改数据结构，则考虑使用 hash set 来存储遍历第一个列表是见到的所有 node。遍历第二个列表时，只要查询是否节点存在于 hash set 即可，耗时 O(m+n)，占用空间 O(m+n)。

计算链表长度差：
先遍历第一个链表，记录长度为 count1，耗时 O(m)
再遍历第二个链表，记录长度为 count2，耗时 O(n)
计算长度差 diff = count1-count2 假设第一个链表更长
让第一个链表先走到 diff 的位置，此时两个链表剩下的长度相同
之后让两个链表一同前进，看是否会遇到相同的节点。
耗时 O(m+n)，占用空间 O(1)。

构建有环链表：
遍历第一个列表，记录最后一个节点，并让其指向列表的第一个节点。
此时问题变成了第二个列表是否有环的问题。
耗时 O(m+n)，占用空间 O(1)。

(lowest-common ancestor) 对于给定的一颗二叉树，输入两个节点，求它们的最低公共节点。
考虑节点存在父引用：
变成两个链表找公共节点问题，沿着 parent 引用一直前进即可。

给定一个整数 N，请按照字典序打印出 1-N 之间的所有数，例如 N=12，输出：

1
10
11
12
2
3
4
5
6
7
8
9

贪吃蛇的实现？

用什么样的数据结构来表示贪吃蛇比较好？贪吃蛇的数据结构实现

---

n x m的格子，通过每个格子需要消耗不同的体力值，可以上下左右移动，求从左上角跑到右下角消耗体力最小的路径。

可以利用 Dijkstra 算法，找出单源最短路径。
每一个格子和它周围邻接的四个格子都有一条边，这条边的 cost 就是目标格子消耗的体力值。

整数数组最大连续和子数组？

整数数组最大连续乘积子数组？