恰当地使用 two pointer 可以使时间复杂度保持在 O(n)

Two pointer所适合的场景，基础的变式是过两个移动不同步长的pointer来完成一些事情， 也在暗指，本身遍历的这个array必须存在一些特性使得pointer可以有不同的移动。如排好了序。

借此稍微总结一下， 运用到two pointer的场景和技巧。

Question [EASY]合并两个sorted array变成一个sorted array。 Given two sorted arrays A and B, each having length N and M respectively. Form a new sorted merged array having values of both the arrays in sorted format.

利用设置在两个sorted array开头的指针， 来达到m+n时间复杂度的效果。 比较简单。

Question [EASY]一次循环找到链表的中间元素。How to find middle element of linked list in one pass?

Question [EASY]判断一个链表是否存在环。 How to find if linked list has a loop ?

Question [EASY]找到链表中倒数第三个元素。How to find 3rd element from end in a linked list in one pass?

以上都是关于单向链类似的问题， 利用two pointer都可以得到快速的解答。

Question [MEDIUM]找到不大于M的连续最大和子数列。 Given an array having N positive integers, find the contiguous subarray having sum as great as possible,, but not greater than M.

其实第二题还涉及到了另一个技巧， 就是在对于部分求和问题里， 使用cumulative sum array是一个可能的切入口。 在将原数列生成对应的cumulative sum array之后， 这个题目也就相应转换为找到两个index， 使得对于这个递增的和数列， 满足：

cum[endIndex] - cum[startIndex-1] <= M and cum[endIndex+1] - cum[startIndex-1] > M的条件， 而endIndex和startIndex在原数列里对应的子数列， 就是满足要求的最大和子数列。

转换了题意之后， 对于这个递增数列， 可以接着用two pointer的思想来处理， 设置start和end两个pointer从头开始， 右移end指针只到不能满足需求为止，然后右移start指针来减少sum使得end指针可以继续右移。 记录下每次start指针右移是的sum， 最大的那个sum所对应的指针位置， 对应回原数列， 就是我们想要找到的连续最大和子数列。可以说， 这道题的突破口是利用cumulative sum来创造一个递增的数列， 从而使two pointer的实现方式更为简洁。

Example2 - continuous minimum distinct subarray

Question [MEDIUM]找到至少含有K个不同数字的连续最小和子数列。 Given an array containing N integers, you need to find the length of the smallest contiguous subarray that contains atleast K distinct elements in it. Output “−1−1” if no such subarray exists.

从题意上和上一问求连续最大和子数列很像， 其实处理方式也有共同之处， 利用cumulative sum来负责和的部分， 利用set的实现来负责distince element的部分， 和之前相比end指针移动的条件， 更换为使得set中元素至少有K个， 记录此时sum和对应得end - start得长度， 然后移动start指针， 更新set元素， 由此往复。n的时间复杂度。