134. Gas Station

题目描述：https://leetcode.com/problems/gas-station/

> 环形路，路上有很多加油站。第i个加油站可加的油为gas[i]，但是从第i个加油站到第i+1个站的消耗为cost[i],求问一辆油箱为空的汽车可否从某加油站开始走直到再走回来？如果可以返回开始的加油站位置，否则返回-1；

解题思路：

首先明确：如果Σcost<=Σgas则肯定能回来。

解法一：

O(n*n)：对于每个站都计算一遍直到找到可以回来的站。

解法二：

直观来考虑，我们首先攒满足够的油，然后再去走费油的路比较好。
于是我们可以将这个题目抽象为一个数组，diff[i] = gas[i] - cost[i]
则攒满足够的油就是求diff的子序列和最大。则出发点就是该子序列的头。如果该数求出来都不能>=0,则无法完成。因为如果可以完成最不济也是结果 == 0
但是我们还得考虑因为是个环形，所以最大的可能跨在头和尾之间，因此我们要求1.最大连续子序列 2.Total-最小连续子序列。取最大的那个。如果是1.则返回头，如果是2则返回尾巴下一个。

解法三：

这道题最直观的思路，是逐个尝试每一个站点，从站 i 点出发，看看是否能走完全程。如果不行，就接着试着从站点 i+1出发。

假设从站点 i 出发，到达站点 k 之前，依然能保证油箱里油没见底儿，从k 出发后，见底儿了。那么就说明 diff[i] + diff[i+1] + ... + diff[k] < 0，而除掉diff[k]以外，从diff[i]开始的累加都是 >= 0的。也就是说diff[i] 也是 >= 0的，这个时候我们还有必要从站点 i + 1 尝试吗？仔细一想就知道：车要是从站点 i+1出发，到达站点k后，甚至还没到站点k，油箱就见底儿了，因为少加了站点 i 的油。。。

因此，当我们发现到达k 站点邮箱见底儿后，i 到 k 这些站点都不用作为出发点来试验了，肯定不满足条件，只需要从k+1站点尝试即可！因此解法时间复杂度从O(n2)降到了 O(2n)。之所以是O(2n)，是因为将k+1站作为始发站，车得绕圈开回k，来验证k+1是否满足。

等等，真的需要这样吗？

我们模拟一下过程：

a. 最开始，站点0是始发站，假设车开出站点p后，油箱空了，假设sum1 = diff[0] +diff[1] + ... + diff[p]，可知sum1 < 0；

b. 根据上面的论述，我们将p+1作为始发站，开出q站后，油箱又空了，设sum2 = diff[p+1] +diff[p+2] + ... + diff[q]，可知sum2 < 0。

c. 将q+1作为始发站，假设一直开到了未循环的最末站，油箱没见底儿，设sum3 = diff[q+1] +diff[q+2] + ... + diff[size-1]，可知sum3 >= 0。

要想知道车能否开回 q 站，其实就是在sum3 的基础上，依次加上 diff[0] 到 diff[q]，看看sum3在这个过程中是否会小于0。但是我们之前已经知道 diff[0] 到 diff[p-1] 这段路，油箱能一直保持非负，因此我们只要算算sum3 + sum1是否 <0，就知道能不能开到 p+1站了。如果能从p+1站开出，只要算算sum3 + sum1 + sum2 是否 < 0，就知都能不能开回q站了。

因为 sum1, sum2 都 < 0，因此如果 sum3 + sum1 + sum2 >=0 那么 sum3 + sum1 必然 >= 0，也就是说，只要sum3 + sum1 + sum2 >=0，车必然能开回q站。而sum3 + sum1 + sum2 其实就是 diff数组的总和 Total，遍历完所有元素已经算出来了。因此 Total 能否 >= 0，就是是否存在这样的站点的 充分必要条件。

0，就是是否存在起作的站点的充分必要条件。

这样时间复杂度进一步从O(2n)降到了 O(n)。

代码：

解法二：

```
class Solution {
public:
    int canCompleteCircuit(vector<int>& gas, vector<int>& cost) {
        int len = gas.size();
        if(len == 0 || len != cost.size()) {
            return -1;
        }
        vector<int> a(len,0);
        for(int i = 0; i < len; i++) {
            a[i] = gas[i] - cost[i];
        }
        int total = a[0];
        int maxLoc = 0, max = a[0], Max = a[0], MaxLoc = 0;
        int minLoc = 0, min = a[0], Min = a[0];
        for(int i = 1; i < len; i++) {
            total += a[i];
            if(max < 0) {
                maxLoc = i;
                max = a[i];
            }
            else {
                max += a[i];
            }
            if(max > Max) {
                Max = max;
                MaxLoc = maxLoc;
            }
            if(min > 0) {
                min = a[i];
            }else {
                min += a[i];
            }
            if(min < Min) {
                Min = min;
                minLoc = i;
            }
        }
        if(total < 0) {
            return -1;
        }
        if(total - Min > Max) {
            return (minLoc+1)%len;
        }
        return MaxLoc;
    }
};
```

解法三：

```
class Solution {
```

```
public:
    int canCompleteCircuit(vector<int>& gas, vector<int>& cost) {
        if(cost.size() != gas.size() || gas.size() == 0) {
            return -1;
        }
        int start = 0;
        int sum = 0;
        int total = 0;
        for(int i = 0; i < gas.size(); i++) {
            total += gas[i] - cost[i];
            if(sum < 0) {
                sum = gas[i] - cost[i];
                start = i;
            }
            else {
                sum += gas[i] - cost[i];
            }
        }
        return total>=0?start:-1;
    }
};
```