

此时 $\text{sum}-k=6-6=0$ ，所以，在 `map` 中查询累加和 0 最早出现的位置，发现累加和 0 最早出现在 -1 位置，所以 `arr[j+1..i]` 即 `arr[0..2]`（也即 `[1,2,3]`）被找到。

具体过程请参看如下代码中的 `maxLength` 方法。

```
public int maxLength(int[] arr, int k) {
    if (arr == null || arr.length == 0) {
        return 0;
    }
    HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
    map.put(0, -1); // 重要
    int len = 0;
    int sum = 0;
    for (int i = 0; i < arr.length; i++) {
        sum += arr[i];
        if (map.containsKey(sum - k)) {
            len = Math.max(i - map.get(sum - k), len);
        }
        if (!map.containsKey(sum)) {
            map.put(sum, i);
        }
    }
    return len;
}
```

理解了原问题的解法后，补充问题是可以迅速解决的。第一个补充问题，先把数组 `arr` 中的正数全部变成 1，负数全部变成 -1，0 不变，然后求累加和为 0 的最长子数组长度即可。第二个补充问题，先把数组 `arr` 中的 0 全部变成 -1，1 不变，然后求累加和为 0 的最长子数组长度即可。两个补充问题的代码略。

未排序数组中累加和小于或等于给定值的最长子数组长度

【题目】

给定一个无序数组 `arr`，其中元素可正、可负、可 0，给定一个整数 k 。求 `arr` 所有的子数组中累加和小于或等于 k 的最长子数组长度。

例如：`arr=[3,-2,-4,0,6]`， $k=-2$ ，相加和小于或等于 -2 的最长子数组为 `{3,-2,-4,0}`，所以结果返回 4。

【难度】

校 ★★★★★

【解答】

本书提供的方法可以做到时间复杂度为 $O(M\log N)$ ，额外空间复杂度为 $O(N)$ 。

依次求以数组的每个位置结尾的、累加和小于或等于 k 的最长子数组长度，其中最长的那个子数组的长度就是我们要的结果。为了便于读者理解，我们举一个比较具体的例子。

假设我们处理到位置 30，从位置 0 到位置 30 的累加和为 100 ($\text{sum}[0..30]=100$)，现在想求以位置 30 结尾的、累加和小于或等于 10 的最长子数组长度。再假设从位置 0 开始累加到位置 10 的时候，累加和第一次大于或等于 90 ($\text{sum}[0..10]\geq 90$)，那么可以知道以位置 30 结尾的相加和小于或等于 10 的最长子数组就是 $\text{arr}[11..30]$ 。也就是说，如果从 0 位置到 j 位置的累加和为 $\text{sum}[0..j]$ ，此时想求以 j 位置结尾的相加和小于或等于 k 的最长子数组长度。那么只要知道大于或等于 $\text{sum}[0..j]-k$ 这个值的累加和最早出现在 j 之前的什么位置就可以，假设那个位置是 i 位置，那么 $\text{arr}[i+1..j]$ 就是在 j 位置结尾的相加和小于或等于 k 的最长子数组。

为了很方便地找到大于或等于某一个值的累加和最早出现的位置，可以按照如下方法生成辅助数组 helpArr 。

1. 首先生成 arr 每个位置从左到右的累加和数组 sumArr 。以 $[1,2,-1,5,-2]$ 为例，生成的 $\text{sumArr}=[0,1,3,2,7,5]$ 。注意， sumArr 中的第一个数为 0，表示当没有任何一个数时的累加和为 0。

2. 生成 sumArr 的左侧最大值数组 helpArr ， $\text{sumArr}=\{0,1,3,2,7,5\} \rightarrow \text{helpArr}=\{0,1,3,3,7,7\}$ 。为什么原来的 sumArr 数组中的 2 和 5 变为 3 和 7 呢？因为我们只关心大于或等于某一个值的累加和最早出现的位置，而累加和 3 出现在 2 之前，并且大于或等于 3 必然大于 2。所以，当然要保留一个更大的、出现更早的累加和。

3. helpArr 是 sumArr 每个位置上的左侧最大值数组，那么它当然是有序的。在这样一个有序的数组中，就可以二分查找大于或等于某一个值的累加和最早出现的位置。例如，在 $[0,1,3,3,7,7]$ 中查找大于或等于 4 这个值的位置，就是第一个 7 的位置。

以原题中给的例子来说明整个计算过程。

$\text{arr}=[3,-2,-4,0,6]$ ， $k=-2$ 。

1. $\text{arr}=[3,-2,-4,0,6]$ ，求得 arr 的累加数组 $\text{sumArr}=[0,3,1,-3,-3,3]$ ，进一步求得 sumArr 的左侧最大值数组 $[0,3,3,3,3,3]$ 。

2. $j=0$ 时， $\text{sum}[0..0]=3$ ，所以在 helpArr 中二分查找大于或等于 $3-k=3-(-2)=5$ 这个值第一次出现的位置，结果是没有。所以，可知以位置 0 结尾的所有子数组累加后没有小于或

等于 k （即-2）的。

3. $j=1$ 时, $\text{sum}[0..1]=1$, 所以在 helpArr 中二分查找大于或等于 $1-k=1-(-2)=3$ 这个值第一次出现的位置, 在 helpArr 中的位置是 1, 对应的 arr 中的位置是 0, 所以, $\text{arr}[1..1]$ 是满足条件的最长数组。

4. $j=2$ 时, $\text{sum}[0..2]=-3$, 所以在 helpArr 中二分查找大于或等于 $-3-k=-3-(-2)=-1$ 这个值第一次出现的位置, 在 helpArr 中的位置是 0, 对应的 arr 中的位置是-1, 表示一个数都不累加的情况, 所以 $\text{arr}[0..2]$ 是满足条件的最长数组。

5. $j=3$ 时, $\text{sum}[0..3]=-3$, 所以在 helpArr 中二分查找大于或等于 $-3-k=-3-(-2)=-1$ 这个值第一次出现的位置, 在 helpArr 中的位置是 0, 对应的 arr 中的位置是-1, 表示一个数都不累加的情况, 所以 $\text{arr}[0..3]$ 是满足条件的最长数组。

6. $j=4$ 时, $\text{sum}[0..4]=3$, 所以在 helpArr 中二分查找大于或等于 $3-k=3-(-2)=5$ 这个值第一次出现的位置, 结果是没有。所以, 可知以位置 4 结尾的所有子数组累加后没有小于或等于 k （即-2）的。

全部过程请参看如下代码中的 `maxLength` 方法。

```
public int maxLength(int[] arr, int k) {
    int[] h = new int[arr.length + 1];
    int sum = 0;
    h[0] = sum;
    for (int i = 0; i != arr.length; i++) {
        sum += arr[i];
        h[i + 1] = Math.max(sum, h[i]);
    }
    sum = 0;
    int res = 0;
    int pre = 0;
    int len = 0;
    for (int i = 0; i != arr.length; i++) {
        sum += arr[i];
        pre = getLessIndex(h, sum - k);
        len = pre == -1 ? 0 : i - pre + 1;
        res = Math.max(res, len);
    }
    return res;
}

public int getLessIndex(int[] arr, int num) {
    int low = 0;
    int high = arr.length - 1;
    int mid = 0;
    int res = -1;
    while (low <= high) {
        mid = (low + high) / 2;
```

```
        if (arr[mid] >= num) {
            res = mid;
            high = mid - 1;
        } else {
            low = mid + 1;
        }
    }
    return res;
}
```

计算数组的小和

【题目】

数组小和的定义如下：

例如，数组 $s=[1,3,5,2,4,6]$ ，在 $s[0]$ 的左边小于或等于 $s[0]$ 的数的和为 0，在 $s[1]$ 的左边小于或等于 $s[1]$ 的数的和为 1，在 $s[2]$ 的左边小于或等于 $s[2]$ 的数的和为 $1+3=4$ ，在 $s[3]$ 的左边小于或等于 $s[3]$ 的数的和为 1，在 $s[4]$ 的左边小于或等于 $s[4]$ 的数的和为 $1+3+2=6$ ，在 $s[5]$ 的左边小于或等于 $s[5]$ 的数的和为 $1+3+5+2+4=15$ ，所以 s 的小和为 $0+1+4+1+6+15=27$ 。

给定一个数组 s ，实现函数返回 s 的小和。

【难度】

校 ★★☆☆

【解答】

用时间复杂度为 $O(N^2)$ 的方法比较简单，按照题目例子描述的求小和的方法求解即可，本书不再详述。下面介绍一种时间复杂度为 $O(N\log N)$ 、额外空间复杂度为 $O(N)$ 的方法，这是一种在归并排序的过程中，利用组间在进行合并时产生小和的过程。

1. 假设左组为 $l[]$ ，右组为 $r[]$ ，左右两个组的组内都已经有序，现在要利用外排序合并成一个大组，并假设当前外排序是 $l[i]$ 与 $r[j]$ 在进行比较。

2. 如果 $l[i] \leq r[j]$ ，那么产生小和。假设从 $r[j]$ 往右一直到 $r[]$ 结束，元素的个数为 m ，那么产生的小和为 $l[i] * m$ 。

3. 如果 $l[i] > r[j]$ ，不产生任何小和。

4. 整个归并排序的过程该怎么进行就怎么进行，排序过程没有任何变化，只是利用步