

sum 此时开始表示 arr[left+1..right] 的累加和。

- 如果 sum 小于 k ，说明 arr[left..right] 还需要加上 right 后面的值，其和才可能达到 k ，所以，令 right 加 1，sum+=arr[right]。需要注意的是，right 加 1 后是否越界。
- 如果 sum 大于 k ，说明所有从 left 位置开始，在 right 之后的位置结束的子数组，即 arr[left..i(i>right)]，累加和一定大于 k 。所以，令 left 加 1，这表示我们开始考查以 left 之后的位置开始的子数组，同时令 sum-=arr[left]，sum 此时表示 arr[left+1..right] 的累加和。

5. 如果 right<arr.length，重复步骤 4。否则直接返回 len，全部过程结束。

具体请参看如下代码中的 getMaxLength 方法。

```
public int getMaxLength(int[] arr, int k) {
    if (arr == null || arr.length == 0 || k <= 0) {
        return 0;
    }
    int left = 0;
    int right = 0;
    int sum = arr[0];
    int len = 0;
    while (right < arr.length) {
        if (sum == k) {
            len = Math.max(len, right - left + 1);
            sum -= arr[left++];
        } else if (sum < k) {
            right++;
            if (right == arr.length) {
                break;
            }
            sum += arr[right];
        } else {
            sum -= arr[left++];
        }
    }
    return len;
}
```

未排序数组中累加和为给定值的最长子数组系列问题

【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为 k 的最长子数组长度。

【补充题目】

给定一个无序数组 `arr`，其中元素可正、可负、可 0。求 `arr` 所有的子数组中正数与负数个数相等的最长子数组长度。

【补充题目】

给定一个无序数组 `arr`，其中元素只是 1 或 0。求 `arr` 所有的子数组中 0 和 1 个数相等的最长子数组长度。

【难度】

尉 ★★☆☆

【解答】

本书提供的方法可以做到时间复杂度为 $O(N)$ 、额外空间复杂度为 $O(N)$ ，首先来看原问题。

为了说明解法，先定义 s 的概念， $s(i)$ 代表子数组 `arr[0..i]` 所有元素的累加和。那么子数组 `arr[j..i]` ($0 \leq j \leq i < \text{arr.length}$) 的累加和为 $s(i) - s(j-1)$ ，因为根据定义， $s(i) = \text{arr}[0..i]$ 的累加和 = `arr[0..j-1]` 的累加和 + `arr[j..i]` 的累加和，又有 `arr[0..j-1]` 的累加和为 $s(j-1)$ 。所以，`arr[j..i]` 的累加和为 $s(i) - s(j-1)$ ，这个结论是求解这道题的核心。

原问题解法只遍历一次 `arr`，具体过程为：

1. 设置变量 `sum=0`，表示从 0 位置开始一直加到 i 位置所有元素的和。设置变量 `len=0`，表示累加和为 k 的最长子数组长度。设置哈希表 `map`，其中，`key` 表示从 `arr` 最左边开始累加的过程中出现过的 `sum` 值，对应的 `value` 值则表示 `sum` 值最早出现的位置。

2. 从左到右开始遍历，遍历的当前元素为 `arr[i]`。

1) 令 `sum=sum+arr[i]`，即之前所有元素的累加和 $s(i)$ ，在 `map` 中查看是否存在 `sum-k`。

- 如果 `sum-k` 存在，从 `map` 中取出 `sum-k` 对应的 `value` 值，记为 j ， j 代表从左到右不断累加的过程中第一次加出 `sum-k` 这个累加和的位置。根据之前得出的结论，`arr[j+1..i]` 的累加和为 $s(i) - s(j)$ ，此时 $s(i) = \text{sum}$ ，又有 $s(j) = \text{sum} - k$ ，所以 `arr[j+1..i]` 的累加和为 k 。同时因为 `map` 中只记录每一个累加和最早出现的位置，所以此时的 `arr[j+1..i]` 是在必须以 `arr[i]` 结尾的所有子数组中，最长的累加和为 k 的子数组，如果该子数组的长度大于 `len`，就更新 `len`。

- 如果 $\text{sum}-k$ 不存在, 说明在必须以 $\text{arr}[i]$ 结尾的情况下没有累加和为 k 的子数组。

2) 检查当前的 sum (即 $s(i)$) 是否在 map 中。如果不存在, 说明此时的 sum 值是第一次出现的, 就把记录 (sum, i) 加入到 map 中。如果 sum 存在, 说明之前已经出现过 sum , map 只记录一个累加和最早出现的位置, 所以此时什么记录也不加。

3. 继续遍历下一个元素, 直到所有的元素遍历完。

大体过程如上, 但还有一个很重要的问题需要处理。根据 $\text{arr}[j+1..i]$ 的累加和为 $s(i)-(j)$, 所以, 如果从 0 位置开始累加, 会导致 $j+1 > i$ 。也就是说, 所有从 0 位置开始的子数组都没有考虑过。所以, 应该从 -1 位置开始累加, 也就是在遍历之前先把 $(0, -1)$ 这个记录放进 map , 这个记录的意义是如果任何一个数也不加时, 累加和为 0。这样, 从 0 位置开始的子数组就被我们考虑到了。

比如, 数组 $[1, 2, 3, 3]$, $k=6$ 。如果从 0 位置开始累加, 也就是遍历之前不加入 $(0, -1)$ 记录, 当遍历到第一个 3 时, $\text{sum}=6$, 在 map 中的记录是:

key	value
1	0 -> 累加和 1 最早出现在 0 位置
3	1 -> 累加和 3 最早出现在 1 位置

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, 发现没有出现过。那么子数组 $[1, 2, 3]$ 就被我们忽略。接下来遍历到第二个 3 时, $\text{sum}=9$, 在 map 中的记录是:

key	value
1	0 -> 累加和 1 最早出现在 0 位置
3	1 -> 累加和 3 最早出现在 1 位置
6	2 -> 累加和 2 最早出现在 2 位置

此时 $\text{sum}-k=9-6=3$, 所以在 map 中查询累加和 3 最早出现的位置, 发现累加和 3 最早出现在 1 位置, 所以 $\text{arr}[j+1..i]$ 即 $\text{arr}[2..3]$ (也即 $[3, 3]$) 被找到。但很明显, $[1, 2, 3]$ 这个子数组才是正确的, 所以不加入 $(0, -1)$ 会导致这样的问题。

如果遍历之前先加入 $(0, -1)$ 这个记录, 当遍历到第一个 3 时, $\text{sum}=6$, 在 map 中的记录是:

key	value
0	-1 -> 累加和 0 最早出现在 -1 位置, 即一个元素也没有时, 累加和为 0
1	0 -> 累加和 1 最早出现在 0 位置
3	1 -> 累加和 3 最早出现在 1 位置

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 ★★★☆