Maximum Size Subarray Sum Equals k

Given an array *nums* and a target value *k*, find the maximum length of a subarray that sums to *k*. If there isn't one, return 0 instead.

Example 1:

Given *nums* = `[1, -1, 5, -2, 3]`, *k* = `3`,
return `4`. (because the subarray `[1, -1, 5, -2]` sums to 3 and is the longest)

Example 2:

Given *nums* = `[-2, -1, 2, 1]`, *k* = `1`,
return `2`. (because the subarray `[-1, 2]` sums to 1 and is the longest)

Follow Up:
Can you do it in O(*n*) time?

Solution 1

```
public int maxSubArrayLen(int[] nums, int k) {
    int sum = 0, max = 0;
    HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
    for (int i = 0; i < nums.length; i++) {
        sum = sum + nums[i];
        if (sum == k) max = i + 1;
        else if (map.containsKey(sum - k)) max = Math.max(max, i - map.get(sum -
k));
        if (!map.containsKey(sum)) map.put(sum, i);
    }
    return max;
}
```

The HashMap stores the sum of all elements before index i as key, and i as value. For each i, check not only the current sum but also (currentSum - previousSum) to see if there is any that equals k, and update max length.

PS: An "else" is added. Thanks to beckychiu1988 for comment.

written by he17 original link here

Solution 2

```
class Solution {
public:
    int maxSubArrayLen(vector<int>& nums, int k) {
        unordered_map<int, int> sums;
        int cur_sum = 0;
        int max_len = 0;
        for (int i = 0; i < nums.size(); i++) {
            cur_sum += nums[i];
            if (cur_sum == k) {
                max_len = i + 1;
            } else if (sums.find(cur_sum - k) != sums.end()) {
                max_len = max(max_len, i - sums[cur_sum - k]);
            }
            if (sums.find(cur_sum) == sums.end()) {
                sums[cur_sum] = i;
            }
        }
        return max_len;
    }
};
```

written by RayZ_O original link here

Solution 3

The subarray sum reminds me the range sum problem. Preprocess the input array such that you get the range sum in constant time. sum[i] means the sum from 0 to i inclusively the sum from i to j is sum[j] - sum[i - 1] except that from 0 to j is sum[j].

j-i is equal to the length of subarray of original array. we want to find the max(j - i) for any sum[j] we need to find if there is a previous sum[i] such that sum[j] - sum[i] = k Instead of scanning from 0 to j -1 to find such i, we use hashmap to do the job in constant time. However, there might be duplicate value of of sum[i] we should avoid overriding its index as we want the max j - i, so we want to keep i as left as possible.

```
public class Solution {
    public int maxSubArrayLen(int[] nums, int k) {
        if (nums == null || nums.length == 0)
            return 0;
        int n = nums.length;
        for (int i = 1; i < n; i++)
            nums[i] += nums[i - 1];
        Map<Integer, Integer> map = new HashMap<>();
        map.put(0, -1); // add this fake entry to make sum from 0 to j consistent
        int max = 0;
        for (int i = 0; i < n; i++) {
            if (map.containsKey(nums[i] - k))
                max = Math.max(max, i - map.get(nums[i] - k));
            if (!map.containsKey(nums[i])) // keep only 1st duplicate as we want
first index as left as possible
                map.put(nums[i], i);
        }
        return max;
    }
}
```

written by xuyirui original link here