Array

Problems Discuss

Back(/tag/array/discuss) **Summary of Sliding Window Patterns for Subarray / Substring**

(/yeet_the_leet) yeet_the_leet (/yeet_the_leet) ★ 174 Last Edit: October 16, 2022 10:51 AM 6.9K VIEWS Share

155

Hi everyone!

I'm a university student who recently studied a lot of sliding window for summer intern interviews. I really appreciate people who write posts like this - they helped me so much. So I would also like to share some ideas, about sliding window, and what is not sliding window.

Sliding Window is very common in interviews and many questions that look like "(*Longest/Shortest/Number of*) (*Substrings/Subarrays*) with (*At most/Exactly*) K elements that fit (*some condition*)" have a common pattern. They are usually O(n).

Sliding Window Questions

- ✓ 3. Longest Substring Without Repeating Characters (https://leetcode.com/problems/longest-substring-without-repeating-characters/)
- ✓ 340. Longest Substring with At Most K Distinct Characters (https://leetcode.com/problems/longest-substring-with-at-most-k-distinct-characters/)
- 76. Minimum Window Substring (https://leetcode.com/problems/minimum-window-substring/solution/)
- 1004. Max Consecutive Ones III (https://leetcode.com/problems/max-consecutive-ones-iii/)
- ✓ 904. Fruit Into Baskets (https://leetcode.com/problems/fruit-into-baskets/)
- 424. Longest Repeating Character Replacement (https://leetcode.com/problems/longest-repeating-character-replacement/)
- 930. Binary Subarrays with Sum (https://leetcode.com/problems/binary-subarrays-with-sum/)

- 992. Subarrays with K Different Integers (https://leetcode.com/problems/subarrays-with-k-different-integers/)
- 1248. Count Number of Nice Subarrays (https://leetcode.com/problems/count-number-of-nice-subarrays/)
- 1358. Number of Substrings Containing All Three Characters (https://leetcode.com/problems/number-of-substrings-containing-all-three-characters)

Not sliding window/not this pattern

- Subarray Sum Equals K (https://leetcode.com/problems/subarray-sum-equals-k/solution/)
- Longest Subarray with Absolute Difference <= Limit (https://leetcode.com/problems/longest-continuous-subarray-with-absolute-diff-less-than-or-equal-to-limit/)
- Longest Palindromic Substring (https://leetcode.com/problems/longest-palindromic-substring/)

Idea/Intuition

We have a "window" of 2 pointers, `left` and `right`, and we keep increasing the right pointer.

- If the element at the right pointer makes the window not valid, we keep moving the left pointer to shrink the window until it becomes valid again.
- Then, we update the global min/max with the result from the valid window.
- To check if it is valid, we need to store the "state" of the window (ex. frequency of letters, number of distinct integers).

This for Variable sliding Window

```
for(right = 0; right < n; right++):
    update window with element at right pointer
    while (condition not valid):
        remove element at left pointer from window, move left pointer to the right
    update global max
```

Length of Substring/Subarray questions:

The first type of common question we will look at is, Max/Min *length* of subarray that fits some condition.

3. Longest substring without repeating characters (https://leetcode.com/problems/longest-substring-without-repeating-characters/)

```
        unordered_map<char, int> counts; // Frequencies of chars in the window
        int res = 0;
        int i = 0; // Left pointer
        for(int j = 0; j < s.length(); j++){
            counts[s[j]] ++; // Add right pointer to window
            while(counts[s[j]] > 1){ // While the element at right pointer created a repeat
                counts[s[i++]] --; // While condition not valid, remove element at left poi
nter from window by decreasing its count, and then increment left pointer. In this case, it
is while s[j] is a duplicate (we will stop after removing the duplicate copy of s[j]).
            } // Now the condition is valid
            res = max(res, j-i+1); // Update global max with the length of current valid su
bstring
        }
        return res;
```

340. Longest Substring with At Most K Distinct Characters (https://leetcode.com/problems/longest-substring-with-at-most-k-distinct-characters/)
This is a more generalized version of At Most 2 or 1 distinct characters. We keep a hashmap of the counts of characters in the current window, and also a numDistinct variable for how many distinct characters are in the current window.

```
int lengthOfLongestSubstringKDistinct(string &s, int k) {
    unordered_map<char, int> counts; // Frequencies of chars in the window
    int res = 0;
    int numDistinct = 0;
    int i = 0; // Left pointer
    for(int j = 0; j < s.length(); j++){
        if(counts[s[j]] == 0) numDistinct++;
        counts[s[j]] ++; // Add right pointer to window
        while(numDistinct > k){
            counts[s[i]] --;
            if(counts[s[i]] == 0) numDistinct --;
            i++;
        } // Now the condition is valid
        res = max(res, j-i+1);
    }
    return res;
}
```

76. Minimum Window Substring (https://leetcode.com/problems/minimum-window-substring/solution/)
Common in interviews, and although it says Hard, it is very similar to Medium sliding window questions. Instead of counting `numDistinct` for number of distinct characters in the window, we count how many distinct characters of t are fully contained in the current window (how many are `remain` ing) so that if `remain = 0`, then all the characters of t are contained in the current window of s.

```
string minWindow(string s, string t) {
    unordered_map<char, int> count;
    int min_j = INT_MAX, min_i = 1, i = 0;
    // Initialize with count of chars of t
    for(int i = 0; i < t.length(); i++){
        count[t[i]] ++;
    }
    int remain = count.size(); // Number of distinct characters in t

    for(int j = 0; j < s.length(); j++){
        count[s[j]] --;
        if(count[s[j]] == 0) remain--; // The count of s[j] in the current window of s has
 equalled the count of s[j] in all of t, so we are done with this character
        while(remain == 0){
            if(j-i < min_j - min_i){ // Update global variables since we need to return the
string
                min_j = j;
                min_i = i;
            }
            // Remove i from the window and increment i
            count[s[i]] ++;
            if(count[s[i]] > 0) remain++;
            i++;
        }
    }
    return min_j == INT_MAX ? "" : s.substr(min_i, min_j - min_i+1);
}
```

Differently Worded Max/Min Length Sliding Window Questions

These questions may look very different from the previous ones, but are just worded differently; the main idea is the same.

1004. Max Consecutive Ones III (https://leetcode.com/problems/max-consecutive-ones-iii/)
This means, longest subarray with at most K `0` s.

```
int longestOnes(vector<int>& A, int K) {
    vector<int> freq(2);
    int i = 0, r=0;
    for(int j = 0; j < A.size(); j++){
        freq[A[j]] ++;
        while(freq[0] > K && i <= j){
            freq[A[i]] --;
            i++;
        }
        r = max(r, j-i+1);
    }
    return r;
}
```

904. Fruit Into Baskets (https://leetcode.com/problems/fruit-into-baskets/)
This is essentially the same as Longest substring with at most K distinct characters, but it is subarrays with at most 2 distinct numbers.

```
    int totalFruit(vector<int>& a) {
        vector<int> freq(a.size(), 0);
        int res = 0, i=0, distinct=0;
        for(int j = 0; j < a.size(); j++){
            freq[a[j]] ++;
            if(freq[a[j]] == 1) distinct++;

            if(distinct <= 2)
                res= max(res,  j-i+1);

            while(distinct > 2 && i < j){
                freq[a[i]] --;
                if(freq[a[i]] == 0) distinct--;
                i++;
            }
        }
        return res;
    }
```

424. Longest Repeating Character Replacement (https://leetcode.com/problems/longest-repeating-character-replacement/)
This can be thought of as, Longest substring where (count of most frequent character) + k < length.

- On top of the hashmap for character counts, we also maintain `maxCount` the count of the most frequent character so far.
- When moving the left pointer we don't need to decrease maxCount, because if maxCount is too big, the "right - left + 1 - k > maxCount" is less likely to be true and we don't move the left pointer (as expected).
- But we do need to maintain the frequencies map so we can increase maxCount if needed.

```
    int characterReplacement(string s, int k) {
        if(s.length() == 0) return 0;
        if(s.length() <= k) return s.length();
        unordered_map<char, int> m;
        int res = 0; int l = 0;
        int maxCount = 0;

        for(int i = 0; i < s.length(); i++){
            m[s[i]] ++;
            maxCount = max(maxCount, m[s[i]]);
            while(i - l + 1 - k > maxCount){
                m[s[l]] --;
                l++;
            }
            res = max(res, i-l+1);

        }
        return res;
    }
```

“Number of Subarrays” Questions

Before, we were finding the **min/max length** of subarray. Now we want the **number of** subarrays.

930. Binary Subarrays with Sum (https://leetcode.com/problems/binary-subarrays-with-sum/)
Another difference this question has to the previous ones, is that it asks for subarrays with **exactly** `K` 1s.
Previous ones were **at most** `K` distinct characters.
So, one way is we can do the `at most` , and then `atMost(k) - atMost(k-1)` .

```
    int numSubarraysWithSum(vector<int>& A, int S) {
         return atMost(A, S) - atMost(A, S - 1);
     }

     int atMost(vector<int>& A, int S) {
         if (S < 0) return 0;
         int res = 0, i = 0;
         for (int j = 0; j <A.size(); j++) {
             S -= A[j];
             while (S < 0){
                 S += A[i]; i++;}

             res += j - i + 1;
         }
         return res;
     }
```

You might have noticed, why are we doing `res += j-i+1` instead of `res = max/min (res, j-i+1)`.

If [i ... j] is a valid subarray, ex. [1,2,3,4]

- There are 4 subarrays with length 1, [1], [2], [3], [4]
- 3 with length 2: [1,2], [2,3], [3,4]
- 2 with length 3
- 1 with length 4

So if at every step we added the length:

- [1] -> add 1
- [1,2] -> add 2
- [1,2,3] -> add 3
- [1,2,3,4] -> add 4

the sum 1+2+3+4 would be the same as the number of subarrays. So we can add the lengths of the valid subarrays.

"Prefixed" Sliding Window

If the subarray `[i,j]` contains `K` 1s, and the first `p` numbers of the subarray are 0, then that means all the subarrays from `[i,j]` to `[i+p, j]` inclusive are valid, which is `p+1` subarrays.

- In the first loop, we move the left pointer to the right while the sum is too high, just like in a standard sliding window question.
    - We set p to 0 because i is changing.
- In the 2nd loop, after finding some [i,j], we find how many leading 0s are in this subarray.
- Finally, we need to check if the current sum equals the target again - because the first loop didn't check if the sum is too low.

```
int numSubarraysWithSum(vector<int>& A, int S) {
    int p=0, i=0, r=0, c=0;
    for(int j = 0; j < A.size(); j++){
        c+= A[j];
        while(c > S && i < j){
            p=0;
            c -= A[i];
            i++;
        }
        while(A[i] == 0 && i < j){
            p++;
            c -= A[i];
            i++;
        }
        if(c == S) r += p+1;
    }
    return r;
```

We can also do this question with the same method as “Subarray Sum Equals K” (mentioned at the end) and count prefix sums.

992. Subarrays with K different integers (https://leetcode.com/problems/subarrays-with-k-different-integers/)

Prefixed Sliding Window
If the subarray `[i,j]` contains X distinct numbers, and the first `p` numbers are duplicates (are also in `[i + p, j]`, that means all the subarrays from `[i,j], [i+1, j], … [i+p, j]` have X distinct numbers as well. Then if `[i,j]` is valid, there will be `p+1` valid subarrays.

```
    int subarraysWithKDistinct(vector<int>& A, int K) {
        unordered_map<int, int> m;
        int i = 0, p=0, d=0, r = 0;

        for(int j = 0; j < A.size(); j++){
            m[A[j]] ++;

            if(m[A[j]] == 1) d++; // num distinct

            if(d > K) {
                m[A[i]] --;
                d--;
                i++;
                p = 0;
            }

            // while A[i] has a duplicate in the current [i..j] subarray
            while(m[A[i]] > 1){
                m[A[i]] --;
                i++; p++;
            }

            if(d == K) r += p+1;

        }
        return r;

    }
```

We could also do the `atMost(k) - atMost(k-1)` strategy:

```
    int subarraysWithKDistinct(vector<int>& A, int K) {
         return atMost(A, K) - atMost(A, K - 1);
     }
     int atMost(vector<int>& A, int K) {
         int i = 0, res = 0;
         unordered_map<int, int> count;

         for (int j = 0; j < A.size(); j++) {
             count[A[j]]++;
             if (count[A[j]] == 1) K--;
             while (K < 0) {
                 count[A[i]]--;
                 if (count[A[i]] == 0) K++;
                 i++;
             }
             res += j - i + 1;
         }
         return res;
     }
```

Count Number of Nice Subarrays (https://leetcode.com/problems/count-number-of-nice-subarrays/)
Number of subarrays with K odd numbers; this is very similar to the previous 2 questions, subarrays with K distinct integers, or with K 1s.

Prefixed:
If `[i...j]` has X odd numbers, and the first `p` numbers in this subarray (`nums[i], nums[i+1], …, nums[i+p-1]`) are even, then `[i..j]` contains `p+1` subarrays with X odd numbers. So, if `[i...j]` has K odd numbers, then it will contain `p+1` valid subarrays.

```
    int numberOfSubarrays(vector<int>& nums, int k) {
        int r = 0, i=0, p=0;
        vector<int> freq(2, 0);
        for(int j = 0; j < nums.size(); j++){
            freq[nums[j] % 2] ++;
            while(freq[1] > k && i < j){
                p = 0; freq[nums[i] % 2] --; i++;
            }
            while(nums[i]% 2 == 0 && i < j){
                p++;
                freq[nums[i]%2] --; i++;
            }
            if(freq[1] == k) r += p+1;

        }
        return r;
    }
```

atMost:

```
    int numberOfSubarrays(vector<int>& nums, int k) {
        return atMost(nums, k) - atMost(nums, k - 1);
    }

    int atMost(vector<int>& A, int k) {
        int res = 0, i = 0, n = A.size();
         vector<int> freq(2, 0);
        for (int j = 0; j < n; j++) {
            freq[A[j] % 2] ++;
            while(freq[1] > k){
                freq[A[i] % 2] --;
                i++;
            }
            res += j - i + 1;
        }
        return res;
    }
```

Subarrays containing all three characters (https://leetcode.com/problems/number-of-substrings-containing-all-three-characters)

Similar to subarrays with K distinct integers, this one is Subarrays with 3 distinct characters.

```
    int numberOfSubstrings(string s) {
        int  j=0, distinct = 0, res=0, prefix=0;
        vector<int> freq(3);

        for(int i = 0; i < s.length(); i++){
            if(freq[s[i] - 'a']== 0) distinct++;
            freq[s[i] - 'a'] ++;

            while(j < i && freq[s[j]-'a'] > 1){
                freq[s[j]-'a']--;
                j++;
                prefix++;
            }
            if(distinct == 3){
                res += prefix+1;
            }
        }
        return res;
    }
```

When does this template not work?

A common problem I faced when studying Leetcode is not knowing when to apply a pattern or not.

Here are some questions that look like this pattern, but are not:

1: Sliding window doesn't work if knowing one element at the edges of the window, does not tell you how to update the state of the window, or whether it becomes valid.

For example, here is a very interesting Sliding Window I got asked on an interview:
Longest Subarray with Absolute Diff <= Limit (https://leetcode.com/problems/longest-continuous-subarray-with-absolute-diff-less-than-or-equal-to-limit/)

The previous types of sliding window don’t work because we need the **max and min of the window**. How would we do that while loop, `while (condition is not valid) left++`, if we don't know whether `left` is the max or min of the window?

- We need to keep **monotonic queues** of the index of max and min elements in the current window. Store indices not values, so we can know the length of sliding window.
- The front of q1 is the max, and the front of q2 is the min.
- Elements in q1 are decreasing and elements in q2 are increasing. Every time we add new right pointer element, we have to pop old elements from the deques to maintain the order.
- Finally, if the limit is exceeded, we pop the old max/min from the left of the queues, and use the index of the popped element to check whether we need to pop the other queue or update res.

```
    int longestSubarray(vector<int>& nums, int limit) {
        if(nums.size() == 0) return 0;
        int i = 0, res = 0;
        deque<int> q1, q2; // for max and min
        for(int j = 0; j < nums.size(); j++){
            while(!q1.empty() && nums[q1.back()] < nums[j] ) q1.pop_back();
            while(!q2.empty() && nums[q2.back()] > nums[j] ) q2.pop_back();
            q1.push_back(j);
            q2.push_back(j);
            while(!q1.empty() && !q2.empty() && (nums[q1.front()] - nums[j] > limit || nums
[j] - nums[q2.front()] > limit )){
                if(nums[q1.front()] - nums[j] > limit) {  i = max(i, q1.front() +1); q1.pop
_front();}
                if(nums[j] - nums[q2.front()] > limit) {i = max(i, q2.front() +1); q2.pop_f
ront();}
            }
            res = max(res, j-i+1);

        }
        return res;
    }
```

I recommend looking at **Monotonic Stack/Queue** problems as it is a very interesting category!

2: Another type of question that doesn't work, is if adding one new element could either increase or decrease the window's state.

- For `Longest substring with at most K distinct characters`, we know that adding a new character to the window can only increase the numDistinct. Removing from the window can only decrease it.
- But for `Subarray Sums Equals K`, with *negative numbers*, this is not the case.

So the usual sliding window template does not work, and we need to keep a `HashMap` of prefix sums, and at each new element check how many previous places had a prefix sum of [current prefix sum - k].
This is out of the scope of this post, but their solution is very detailed! (https://leetcode.com/problems/subarray-sum-equals-k/solution/)

```
    int subarraySum(vector<int>& nums, int k) {
        unordered_map<int, int> m;
        m[0] = 1;
        int total = 0;
        int count = 0;
        for(int i = 0; i < size(nums); i++){
            count += nums[i];

            if(m.find(count - k) != m.end()) total += m[count-k];
            m[count] += 1;

        }
        return total;

    }
```

Subarray Sum/Product is also a whole other category of problem! Very interesting. For example:
https://leetcode.com/problems/subarray-product-less-than-k/ (https://leetcode.com/problems/subarray-product-less-than-k/)
https://leetcode.com/problems/subarray-sums-divisible-by-k/ (https://leetcode.com/problems/subarray-sums-divisible-by-k/)

3: Similar to the 1st reason: If given an invalid subarray it is hard to check whether adding or removing from only one end at a time would ever make it valid.

5. Longest Palindromic Substring (https://leetcode.com/problems/longest-palindromic-substring/)

Most "palindrome" questions are not sliding window, because the nature of palindrome means you either expand the 2 pointers from the center or move the pointers inwards, one starts at 0 and one starts at n-1.

If we tried to apply the pattern on longest palindrome question,

- it is hard to check if adding the element at right pointer makes the window valid
- and if the new element makes it invalid, hard to check how many times we need to move the left pointer to make it valid

The solution (https://leetcode.com/problems/longest-palindromic-substring/solution/) is to consider every possible "center" of the palindrome and expand the 2 pointers outward.

More Useful Links

- (Prefixed sliding window diagram) https://leetcode.com/problems/subarrays-with-k-different-integers/discuss/235235/C%2B%2BJava-with-picture-prefixed-sliding-window (https://leetcode.com/problems/subarrays-with-k-different-integers/discuss/235235/C%2B%2BJava-with-picture-prefixed-sliding-window)
- (List of similar questions) https://leetcode.com/problems/number-of-substrings-containing-all-three-characters/discuss/516977/JavaC++Python-Easy-and-Concise (https://leetcode.com/problems/number-of-substrings-containing-all-three-characters/discuss/516977/JavaC++Python-Easy-and-Concise)
- (Minimum Window Substring template) https://leetcode.com/problems/minimum-window-substring/discuss/26808/here-is-a-10-line-template-that-can-solve-most-substring-problems (https://leetcode.com/problems/minimum-window-substring/discuss/26808/here-is-a-10-line-template-that-can-solve-most-substring-problems)
- (What can or cannot be solved with Two Pointers) https://leetcode.com/problems/subarray-sum-equals-k/discuss/301242/General-summary-of-what-kind-of-problem-can-cannot-solved-by-Two-Pointers (https://leetcode.com/problems/subarray-sum-equals-k/discuss/301242/General-summary-of-what-kind-of-problem-can-cannot-solved-by-Two-Pointers)

Thanks for reading, and happy coding! Please let me know if I missed something or if you know of any interesting questions to learn!

string subarray two-pointers c++ array substring slidingwindow

Comments: 10 **Best** Most Votes Newest to Oldest Oldest to Newest

Type comment here... (Markdown is supported)

Post

(/buildwithmalik) buildwithmalik (/buildwithmalik) ★ 1160 January 5, 2022 1:47 PM

This post deserves an award. To be able to give so much value in one post, the author has to go through a lot of problems. Thank you so much.

9 Show 1 reply Reply

(/Gabriel_Belmont) Gabriel_Belmont (/Gabriel_Belmont) ★ 65 April 22, 2022 6:49 PM

give this guy a medal, a very good explanation, and a nice list. It will definitely help me and reduce my time of understanding the sliding window pattern problems.

3 Reply

(/pranavrocksharma) pranavrocksharma (/pranavrocksharma) ★ 27 Last Edit: July 24, 2022 12:33 AM

bro your subarray with k distinct is failing for the given test case , i copy pasted your code while checking
fix this line

```
if (count[A[j]] == 1) K--;
```

2 Reply

(/dcypher) dcypher (/dcypher) ★ 66 March 26, 2021 9:58 AM

Great post! keep it up!

1 Reply

(/AmeyDhimte) AmeyDhimte (/AmeyDhimte) ★ 0 December 27, 2022 1:50 PM

Thanks a lot bro. I'm glad I came across this article while I was starting to explore window sliding concept

Reply

0 Reply

(/binarybeauty45) binarybeauty45 (/binarybeauty45) ★ 0 August 1, 2022 12:32 AM

wow its really awesome .I m beginner in this coding world.I am too much confused how to solve substring subarrays type of problem .I was not understanding from where to start.
such a good explaination . Thank you so much for this post +
wonderful post just love it.

▲ 0 ▼ Reply

(/pranavrocksharma) pranavrocksharma (/pranavrocksharma) ★ 27 July 26, 2022 12:37 PM

Subarray product less than k can be solved using your template, only for -ve ranges exception is there

```
int numSubarrayProductLessThanK(vector<int>& nums, int k) {
    int count = 0, prod = 1;
    int left = 0, n = nums.size();
    for(int right=0; right<n; ++right){
        prod *= nums[right];
        while(prod>=k and left<=right){
            prod /= nums[left];
            ++left;
        }
```

Read More

▲ 0 ▼ Reply

(/pranavrocksharma) pranavrocksharma (/pranavrocksharma) ★ 27 July 23, 2022 2:42 AM

bro what a template and notes, hats off

▲ 0 ▼ Reply

(/Bodla) Bodla (/Bodla) ★ 15 June 17, 2022 2:12 AM

Best among Sliding window study guides. :)

▲ 0 ▼ Reply

(/nitink_33) nitink_33 (/nitink_33) ★ 491 February 9, 2022 9:44 PM

can someone make a public list of all these types of questions so that we can practice these good problems in a sequence

▲ 0 ▼ Reply