GeeksforGeeks
A computer science portal for geeks

Practice | GATE CS | Placements | Videos | Contribute

Window Sliding Technique

This technique shows how a nested for loop in few problems can be converted to single for loop and hence reducing the time complexity.

2.4

Let's start with a problem for illustration where we can apply this technique –

```
Given an array of integers of size 'n'.
Our aim is to calculate the maximum sum of 'k'
consecutive elements in the array.

Input  : arr[] = {100, 200, 300, 400}
         k = 2
Output : 700

Input  : arr[] = {1, 4, 2, 10, 23, 3, 1, 0, 20}
         k = 4
Output : 39
We get maximum sum by adding subarray {4, 2, 10, 23}
of size 4.

Input  : arr[] = {2, 3}
         k = 3
Output : Invalid
There is no subarray of size 3 as size of whole
array is 2.
```

So, let's analyze the problem with **Brute Force Approach**. We start with first index and sum till **k-th** element. We do it for all possible consecutive blocks or groups of k elements. This method requires nested for loop, the outer for loop starts with the starting element of the block of k elements and the inner or the nested loop will add up till the k-th element.

Consider the below C++ implementation :

```
// O(n*k) solution for finding maximum sum of
// a subarray of size k
#include <iostream>
using namespace std;

// Returns maximum sum in a subarray of size k.
int maxSum(int arr[], int n, int k)
{
    // Initialize result
    int max_sum = INT_MIN ;
```

```
    // Consider all blocks starting with i.
    for (int i=0; i<n-k+1; i++)
    {
        int current_sum = 0;
        for (int j=0; j<k; j++)
            current_sum = current_sum + arr[i+j];

        // Update result if required.
        max_sum = max(current_sum , max_sum );
    }

    return max_sum;
}

// Driver code
int main()
{
    int arr[] = {1, 4, 2, 10, 2, 3, 1, 0, 20};
    int k = 4;
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << maxSum(arr, n, k);
    return 0;
}
```

Output :

```
24
```

It can be observed from the above code that the time complexity is **O(k*n)** as it contains two nested loops.

Window Sliding Technique

The technique can be best understood with the window pane in bus, consider a window of length **n** and the pane which is fixed in it of length **k**. Consider, initially the pane is at extreme left i.e., at 0 units from the left. Now, co-relate the window with array arr[] of size n and plane with current_sum of size k elements. Now, if we apply force on the window such that it moves a unit distance ahead. The pane will cover next **k** consecutive elements.

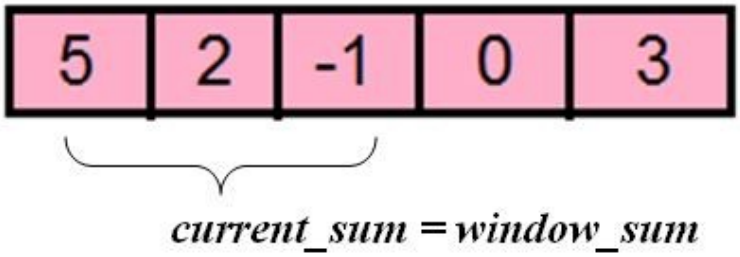
Consider an array **arr[]** = {5 , 2 , -1 , 0 , 3} and value of **k** = 3 and **n** = 5

Applying sliding window technique :

1. We compute the sum of first k elements out of n terms using a linear loop and store the sum in variable window_sum.
2. Then we will graze linearly over the array till it reaches the end and simultaneously keep track of maximum sum.
3. To get the current sum of block of k elements just subtract the first element from the previous block and add the last element of the current block .

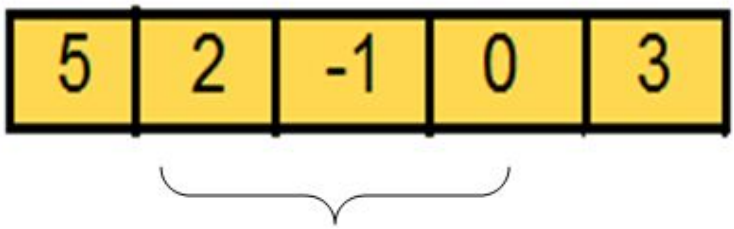
The below representation will make it clear how the window slides over the array.

This is the initial phase where we have calculated the initial window sum starting from index 0 . At this stage the window sum is 6. Now, we set the maximum_sum as current_window i.e 6.



$$current\_sum = window\_sum$$

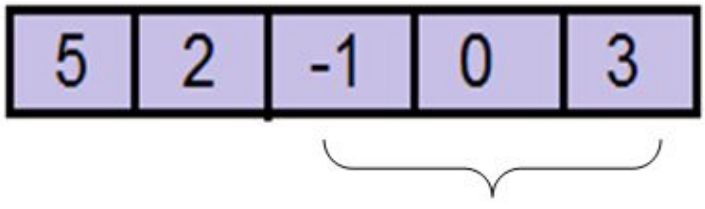
Now, we slide our window by a unit index. Therefore, now it discards 5 from the window and adds 0 to the window. Hence, we will get our new window sum by subtracting 5 and then adding 0 to it. So, our window sum now becomes 1. Now, we will compare this window sum with the maximum_sum. As it is smaller we wont the change the maximum_sum.



$$current\_sum = window\_sum + (-5) + (0)$$

Similarly, now once again we slide our window by a unit index and obtain the new window sum to be 2. Again we check if this current window sum is greater than the maximum_sum till now. Once, again it is smaller so we don't change the maximum_sum.

Therefore, for the above array our maximum_sum is 6.



$$current\_sum = window\_sum + (-2) + (3)$$

C++ code for the above description :

```
// O(n) solution for finding maximum sum of
// a subarray of size k
#include <iostream>
using namespace std;

// Returns maximum sum in a subarray of size k.
```

```
int maxSum(int arr[], int n, int k)
{
    // k must be greater
    if (n < k)
    {
       cout << "Invalid";
       return -1;
    }

    // Compute sum of first window of size k
    int max_sum = 0;
    for (int i=0; i<k; i++)
       max_sum += arr[i];

    // Compute sums of remaining windows by
    // removing first element of previous
    // window and adding last element of
    // current window.
    int window_sum = max_sum;
    for (int i=k; i<n; i++)
    {
       window_sum += arr[i] - arr[i-k];
       max_sum = max(max_sum, window_sum);
    }

    return max_sum;
}

// Driver code
int main()
{
    int arr[] = {1, 4, 2, 10, 2, 3, 1, 0, 20};
    int k = 4;
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << maxSum(arr, n, k);
    return 0;
}
```

Output:24

Now, it is quiet obvious that the Time Complexity is linear as we can see that only one loop runs in our code. Hence, our Time Complexity is **O(n)**.

We can use this technique to find max/min k-subarray, XOR, product, sum, etc. Refer sliding window problems for such problems.

This article is contributed by **Kanika Thakral**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

GATE CS Corner Company Wise Coding Practice

Arrays Technical Scripter sliding-window

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Recommended Posts:

Maximum sum subarray having sum less than or equal to given sum
Find a triplet that sum to a given value
Find maximum (or minimum) sum of a subarray of size k
How Google Search Works!!
Perfect Sum Problem (Print all subsets with given sum)
Maximum size subset with given sum
No of pairs (a[j] >= a[i]) with k numbers in range (a[i], a[j]) that are divisible by x
Sparse Table
Maximum array from two given arrays keeping order same
Probability of a random pair being the maximum weighted pair

(Login to Rate)

2.4 Average Difficulty : **2.4/5.0**
Based on **23** vote(s)

Basic Easy Medium Hard Expert

Add to TODO List

Mark as DONE

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

Share this post!

@geeksforgeeks, Some rights reserved Contact Us! About Us! Careers! Privacy Policy