Enrol in top rated Coding Courses and get assured Scholarship | Apply Now FREE

takeUforward

~ Strive for Excellence

December 4, 2021 ▪ Arrays / Data Structure / Queue

Sliding Window Maximum

Problem Statement: Given an array of integers arr, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position. Return *the **max sliding window**.*

Examples:

```
Example 1:

Input: arr = [4,0,-1,3,5,3,6,8],
k = 3

Output: [4,3,5,5,6,8]

Explanation:
```

Subscribe

I want to receive latest posts and interview tips

Name*

John

Email*

abc@gmail.com

Join takeUforward

Search

Search

```
Window position
Max
-------------------------
-----
[4  0  -1] 3  5  3  6  8
4
 4 [0  -1  3] 5  3  6  8
3
 4  0 [-1  3  5] 3  6  8
5
 4  0  -1 [3  5  3] 6  8
5
 4  0  -1  3 [5  3  6] 8
6
 4  0  -1  3  5 [3  6  8]
8

For each window of size k=3, we
find the maximum element in the
window and add it to our output
array.
```

Example 2:

Input: arr= [20,25], k = 2

Output: [25]

Explanation: There's just one window is size 2 that is possible and the maximum of the two elements is our answer.

Solution:

***Disclaimer**: Don't jump directly to the solution, try it out yourself first.*

Solution 1(Brute Force):

Recent Posts

Pattern-1: Rectangular Star Pattern

Time and Space Complexity – Strivers A2Z DSA Course

Hashing | Maps | Time Complexity | Collisions | Division Rule of Hashing | Strivers A2Z DSA Course

Print 1 to N using Recursion

Print N to 1 using Recursion

Accolite Digital Amazon Arcesium Bank of America Barclays BFS Binary Search Binary Search Tree Commvault CPP DE Shaw DFS DSA Self Paced google HackerEarth Hashing

Intuition: We want to look for a window of size k at a time and then shift to the next window. So why not do exactly what we are asked to! We fix our window of size k at first and then calculate the maximum element in it. We then shift our window to the next position and do the same process until we exhaust all possibilities i.e we reach the end of the array.

Approach: We initially keep a left and right pointer to fix our window to a size of k. We compute the maximum element present in this window using the GetMax function. Further, update the left and right pointer by left++ and right++ every time to get to a new window of size k using a while loop. For every new window we encounter, we add the maximum element using the GetMax function to our data structure.

Code:

C++ Code

```
#include<bits/stdc++.h>

using namespace std;

void GetMax(vector < int > nums, int l,
  int i, maxi = INT_MIN;
  for (i = l; i <= r; i++)
    maxi = max(maxi, nums[i]);
  arr.push_back(maxi);
}
vector < int > maxSlidingWindow(vector <
  int left = 0, right = 0;
  int i, j;
```

infosys inorder Java Juspay Kreeti Technologies Morgan Stanley Newfold Digital Oracle post order queue recursion Samsung SDE Core Sheet SDE Sheet Searching set-bits sorting Strivers A2ZDSA Course sub-array subarray Swiggy takeuforward TCQ NINJA TCS TCS CODEVITA TCS Ninja TCS NQT VMware XOR

```
    vector < int > arr;
    while (right < k - 1) {
      right++;
    }
    while (right < nums.size()) {
      GetMax(nums, left, right, arr);
      left++;
      right++;
    }
    return arr;
  }
  int main() {
    int i, j, n, k = 3, x;
    vector < int > arr {
  4,0,
  -1,
  3,
  5,
  3,
  6,
  8};
    vector < int > ans;
    ans = maxSlidingWindow(arr, k);
    cout << "Maximum element in every " <<
    for (i = 0; i < ans.size(); i++)
      cout << ans[i] << "  ";
    return 0;
  }
```

Output:

Maximum element in every 3 window

4 3 5 5 6 8

Time Complexity: O(N^2)

Reason: One loop for traversing and another to findMax

Space Complexity: O(K)

Reason: No.of windows

Java Code

```
import java.util.*;
class TUF {
    static void GetMax(int arr[], int l,
        int i, maxi = Integer.MIN_VALUE;
        for (i = l; i <= r; i++)
            maxi = Math.max(maxi, arr[i
        maxx.add(maxi);
    }
    static ArrayList < Integer > maxSlic
        int left = 0, right = 0;
        int i, j;
        ArrayList < Integer > maxx = new
        while (right < k - 1) {
            right++;
        }
        while (right < arr.length) {
            GetMax(arr, left, right, max
            left++;
            right++;
        }
        return maxx;
    }
    public static void main(String args[
        int i, j, n, k = 3, x;
        int arr[]={4,0,-1,3,5,3,6,8};
        ArrayList < Integer > ans;
        ans = maxSlidingWindow(arr, k);
        System.out.println("Maximum elem
        for (i = 0; i < ans.size(); i++)
            System.out.print(ans.get(i)

    }
}
```

Output:

Maximum element in every 3 window

4 3 5 5 6 8

Time Complexity: O(N^2)

Reason: One loop for traversing and another to findMax

Space Complexity: O(K)

Reason: No.of windows

Solution 2: Optimized Solution

Intuition : Can we do something better?

To understand this, we would first need to check whether we are doing any repetitions. To understand this, consider the following scenario:

Window : [1,2,3] and the next incoming value is 2

For this state, we get a maximum of 3. However, when our state changes to, [2,3,2] we again check what is the largest element even though we know that the outgoing element is not the largest one. Hence, the point of concern lies only when the outgoing element was the largest.

Approach

We address this problem with the help of a data structure that keeps checking whether the incoming element is larger than the already present elements. This could be implemented with the help of a de-queue. When shifting our window, we push the new element in from the rear of our de-queue. Following is a sample representation of our dequeue:

Every time before entering a new element, we first need to check whether the element present at the front is out of bounds of our present window size. If so, we need to pop that out. Also, we need to check from the rear that the element present is smaller than the incoming element. If yes, there’s no point storing them and hence we pop them out. Finally, the element present at the front would be our largest element.

Code:

C++ Code

```
#include<bits/stdc++.h>

using namespace std;

vector < int > maxSlidingWindow(vector <
  deque < int > dq;
  vector < int > ans;
  for (int i = 0; i < nums.size(); i++) {
    if (!dq.empty() && dq.front() == i -

    while (!dq.empty() && nums[dq.back()
      dq.pop_back();

    dq.push_back(i);
    if (i >= k - 1) ans.push_back(nums[d
  }
  return ans;
}
int main() {
  int i, j, n, k = 3, x;
  vector < int > arr {4,0,-1,3,5,3,6,8};
  vector < int > ans;
  ans = maxSlidingWindow(arr, k);
  cout << "Maximum element in every " <<
  for (i = 0; i < ans.size(); i++)
    cout << ans[i] << " ";
  return 0;
}
```

Output:

```
Maximum element in every 3 window

4 3 5 5 6 8
```

Time Complexity: O(N)

Space Complexity: O(K)

Java Code

```
  import java.util.*;
  class TUF {
      public static int[] maxSlidingWindo
          int n = a.length;
          int[] r = new int[n - k + 1];
          int ri = 0;
          // store index
          Deque < Integer > q = new ArrayD
```

```
        for (int i = 0; i < a.length; i+
            // remove numbers out of ran
            if (!q.isEmpty() && q.peek()
                q.poll();
            }
            // remove smaller numbers in
            while (!q.isEmpty() && a[q.p
                q.pollLast();
            }

            q.offer(i);
            if (i >= k - 1) {
                r[ri++] = a[q.peek()];
            }
        }
        return r;
    }
    public static void main(String args[
        int i, j, n, k = 3, x;
        int arr[]={4,0,-1,3,5,3,6,8};
        int ans[] = maxSlidingWindow(arr
        System.out.println("Maximum elem
        for (i = 0; i < ans.length; i++)
            System.out.print(ans[i] + "

    }
}
```

Output:

```
Maximum element in every 3 window
4 3 5 5 6 8
```

Time Complexity: O(N)

Space Complexity: O(K)

> Special thanks to **Naman Daga** for contributing to this article on takeUforward. If you also wish to share your knowledge with the takeUforward fam, please check out this article