Maximum Value in a Sliding Window

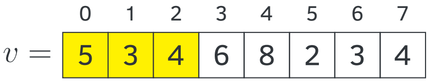
30 Nov 2018 By An7ar35

Description

Find max number in sliding window at each state. The window of size k moves from the beginning to the end of the array v by 1 element at each state.

For example:

Window size $k = 3$.
Array of numbers v of size 8.



	0	1	2	3	4	5	6	7
$v =$	5	3	4	6	8	2	3	4

Approach

The whole algorithm rests on the premise of keeping track of the max-value indices of the current overlapping windows (k). As the sliding window goes forward, we don't know if the next index i is going to be that of a value which will be the largest in following window so we add it to a Deque data-structure for consideration.

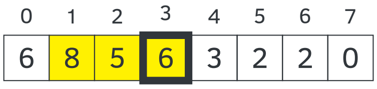
The `back()` of the deque is used to push max-value indices of current and future windows. The last one added is evaluated and replaced, when a larger candidate has been found, during each iteration of i .

The `front()` of the deque then ends up only containing the top value index for the current window. During each iteration the front element is added to the results and then removed on the next.

Why i is always added to the deque:

e.g.: Window size $k = 3$, iteration currently on $i = 2$.

On the { $8, 5, 6$ } window, 8 is the largest but we still need to keep track of 6 as it may be the largest in the next window: { $5, 6, 3$ }.
This is why we must always add the current i to the back of the deque.



0	1	2	3	4	5	6	7
6	8	5	6	3	2	2	0

Why we use i in the deque instead of the value directly:

By tracking indices instead of values directly, we can deduce if an element in the deque is within the current sliding window by:

1. checking if it we are on the last element of the first window or beyond (line 22 in the code) with $i_{current} > k - 1$ and
2. checking the element is gone out of scope of the current window (line 9 in the code) with $i_{deque.front()} == (i_{current} - k)$.

Step-by-step

Step	Current index i in v	Explanations	`deque`	`results`
1	0 1 2 3 4 5 6 7 5 3 4 6 8 2 3 4	Add 5 to the back.	{ 0 }	{ }
2	0 1 2 3 4 5 6 7 5 3 4 6 8 2 3 4	Is $5 < 3$? No, keep 5. Add 3 to the back.	{ 0, 1 }	

3	0 1 2 3 4 5 6 7
	5 3 4 6 8 2 3 4
		Is $3 < 4$? Yes, remove 3.
		Add 4 to the back.
		Is $i >= k - 1$? Yes, copy 5 to results.
			{ 0 }
			{ 0, 2 }
				{ 5 }

4	0 1 2 3 4 5 6 7
	5 3 4 6 8 2 3 4
		Is `front()` == $i - k$? Yes, remove 5.
		Is $4 < 6$? Yes, remove 4.
		Add 6 to the back.
		Is $i >= k - 1$? Yes, copy 6 to results.
			{ 0 }
			{ }
			{ 3 }
				{ 5, 6 }

5	0 1 2 3 4 5 6 7
	5 3 4 6 8 2 3 4
		Is `front()` == $i - k$? No. Leave it.
		Is $6 < 8$? Yes, remove 6.
		Add 8 to the back.
		Is $i >= k - 1$? Yes, copy 8 to results.
			{ }
			{ 4 }
				{ 5, 6, 8 }

6	0 1 2 3 4 5 6 7 5 3 4 6 8 2 3 4	Is `front()` == $i - k$? No. Leave it. Is $8 < 2$? No. Keep 8. Add 2 to the back. Is $i >= k - 1$? Yes, copy 8 to results.	{ 4, 5 }	{ 5, 6, 8, 8 }

7	0 1 2 3 4 5 6 7
	5 3 4 6 8 2 3 4
		Is `front()` == $i - k$? No. Leave it.
		Is $2 < 3$? Yes. remove 2.
		Add 3 to the back.
		Is $i >= k - 1$? Yes, copy 8 to results.
			{ 4 }
			{ 4, 6 }
				{ 5, 6, 8, 8, 8 }

8	0 1 2 3 4 5 6 7
	5 3 4 6 8 2 3 4
		Is `front()` == $i - k$? Yes. Remove 8.
		Is $3 < 4$? Yes. Remove 3.
		Add 4 to the back.
		Is $i >= k - 1$? Yes, copy 4 to results.
			{ 6 }
			{ }
			{ 7 }
				{ 5, 6, 8, 8, 8, 4 }

Code

```
std::vector<int> getWindowMax( std::vector<int> v, int k ) {
    auto deque   = std::deque<int>();
    auto results = std::vector<int>();

    /* iterate through the numbers 1 step at a time */
    for( int i = 0; i < v.size(); i++ ) {

        /* Clear remaining elements from previous window */
        if( !deque.empty() && deque.front() == i - k )
            deque.pop_front();

        /* Get rid of any elements from current window
         * that is smaller than the current value at index */
        while( !deque.empty() && v[deque.back()] < v[i] )
            deque.pop_back();

        /* Put current element's index at the back of the deque */
        deque.push_back( i );

        /* whatever is at the front of the queue
         * will be the max element in current window */
        if( i >= k - 1 )
            results.emplace_back( v[deque.front()] );
    }
    return results;
}
```