Daily Coding Problem #18

Problem

This problem was asked by Google.

Given an array of integers and a number k, where 1 <= k <= length of the array, compute the maximum values of each subarray of length k.

For example, given array = [10, 5, 2, 7, 8, 7] and k = 3, we should get: [10, 7, 8, 8], since:

- 10 = max(10, 5, 2)
- 7 = max(5, 2, 7)
- 8 = max(2, 7, 8)
- 8 = max(7, 8, 7)

Do this in O(n) time and O(k) space. You can modify the input array in-place and you do not need to store the results. You can simply print them out as you compute them.

Solution

Even though the question states O(n), in an interview it's always useful to first write out a brute force solution, which may provide us with some insight on some deeper structure in the problem.

So let's first write out a naive solution: we can simply take each subarray of k length and compute their maxes.

```
def max_of_subarrays(lst, k):
    for i in range(len(lst) - k + 1):
        print(max(lst[i:i + k]))
```

This takes O(n * k) time, which doesn't get us quite to where we want. How can we make this faster?

One possible idea is this: we could use a max-heap of size k and add the first k elements to the heap initially, and then pop off the max and add the next element for the rest of the array. This is better, but adding and extracting from the heap will take O(log k), so this algorithm will take O(n * log k), which is still not enough. How can we do better?

Notice that, for example, the input [1, 2, 3, 4, 5, 6, 7, 8, 9] and k = 3, after evaluating the max of first range, since 3 is at the end, we only need to check whether 4 is greater than 3. If it is, then we can print 4 immediately, and if it isn't, we can stick with 3.

On the other hand, for the input [9, 8, 7, 6, 5, 4, 3, 2, 1] and k = 3, after evaluating the max of the first range, we can't do the same thing, since we can't use 9 again. We have to look at 8 instead, and then once we move on to the next range, we have to look at 7.

These two data points suggest an idea: we can keep a double-ended queue with max size k and only keep what we need to evaluate in it. That is, if we see [1, 3, 5], then we only need to keep [5], since we know that 1 and 3 cannot possibly be the maxes.

So what we can do is maintain an ordered list of indices, where we only keep the elements we care about, that is, we will maintain the loop invariant that our queue is always ordered so that we only keep the indices we care about (i.e, there are no elements that are greater after, since we would just pick the greater element as the max instead).

It will help to go over an example. Consider our test input: [10, 5, 2, 7, 8, 7] and k = 3. Our queue at each step would look like this (recall that these are indices):

Preprocessing

After processing 10: [0] After processing 5: [0, 1] # 5 is smaller than 10, and 10 is still valid until we hit the 3rd index After processing 2: [0, 1, 2] # 2 is smaller than 5, and 10 is still valid

Main Loop

Print value of first element in our queue: **10**

After processing 7: [4] # 10 is no longer valid (we can tell since the current index - 0 > k), so we dequeue from the front. 7 is bigger than 5 and 2, so we get rid of them from the back and replace it with the 7

Print value of first element in our queue: **7**

After processing 8: [5] # 8 is bigger than 7, so no point in keeping 7 around. We get rid of it from the back and replace it with the 8

Print value of first element in our queue: **8**

After processing 7: [5, 4] # 7 is smaller than 8, so we enqueue it from the back

Print value of first element in our queue: **8**

Code

```python
from collections import deque

def max_of_subarrays(lst, k):
    q = deque()
    for i in range(k):
        while q and lst[i] >= lst[q[-1]]:
            q.pop()
        q.append(i)

    # Loop invariant: q is a list of indices where their corresponding values are in descending order.
    for i in range(k, len(lst)):
        print(lst[q[0]])
        while q and q[0] <= i - k:
            q.popleft()
        while q and lst[i] >= lst[q[-1]]:
            q.pop()
        q.append(i)
    print(lst[q[0]])
```