Monotonic Data Structures --- Concepts & Qns

by . codestorywithMIK

codestorywithmik

CSwithMIK

codestorywithMIK

Motivation :-

Always remember that, you are not late.
You are not behind, you are just in
the middle of your comeback story...

Video - 3

Monotonic Queue

It's a data structure where :-

* Elements are stored in either monotonic decreasing or monotonic increasing order.

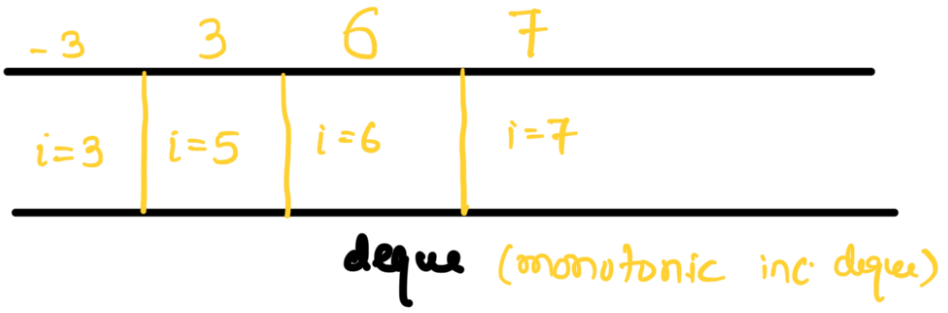
* Often implemented using a deque, allowing efficient insertion & deletion from both ends.

Monotonic Increasing Queue

Keeps element in increasing order.

keeps element in increasing order

arr = { 1, 3, -1, -3, 5, 3, 6, 7 }
0 1 2 3 4 5 6 7



deque (monotonic inc. deque)

```
deque<int> deq ; // stores index of the element

for (int i= 0;  i<n ;  i++) {

        while ( !deq.empty() &&  array[deq.back()] > arr[i]) {

                deq.pop_back(); //we want increasing

        }

        deq.push_back(i)  // maintains increasing order
}
```

: Template :

How does this even help us ???

Monotonic Queue is often used in combination with **Sliding Window...**

"Let's suppose you are asked to find the minimum element in subarrays of size K in an array.

arr = {1, 3, -1, -3, 5, 3, 6, 7} , K = 3

(indices: 0, 1, 2, 3, 4, 5, 6, 7)

Brute Force

-1, -3,

T.C = O(n * K)
S.C = O(1)

```
for(i=0 ; i<n; i++){
    minVal = ∞;
    for(j=i; j<=i+K-1 && j<n ; j++){
        minVal = min(minVal, nums[j]);
    }
    result.push_back(minVal);
}
```

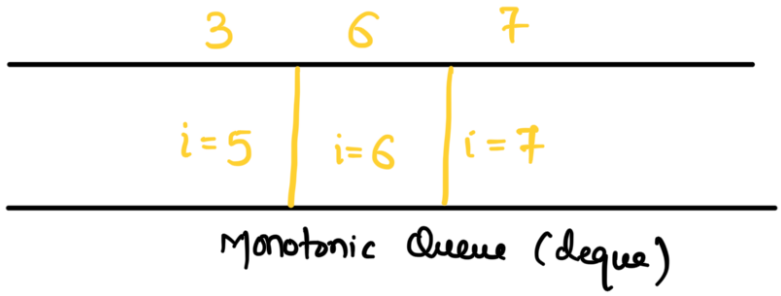
Optimally using - Monotonic Queue ...

arr = {1, 3, -1, -3, 5, 3, 6, 7}, K = 3

(indices: 0, 1, 2, 3, 4, 5, 6, 7)

i >= K-1

i - K = 6 - 3



que.front() <= i-K

que.pop_front()

{-1, -3, -3, -3, 3, 3}

Subarray → K size → Minimum

(Monotonic Increasing Deque)

T.C = O(2*n) ≅ O(n), S.C = O(n)

Correcting template :-

```
deque<int> deq; // stores index of the element

for(int i=0; i<n; i++){

    while(!deq.empty() && deq.front() <= i-k){
        deq.pop_front();
    }
    while(!deq.empty() && array[deq.back()] > arr[i]){
        deq.pop_back(); //we want increasing
    }
    deq.push_back(i) // maintains increasing order
    if(i >= k-1){ // now we will be getting k-sized window;
        result.push_back(deq.front()); //min element.
    }
}
```

: Template :

Similarly Monotonic Decreasing Deque can be

Deque can be derived ...

```
deque<int> deq ; // stores index of the element

for (int i= 0;   i<n ;  i++) {

      while (!deq.empty() && deq.front() <= i-k) {
                  deq.pop_front();
      }
     while (!deq.empty() &&  array[deq.back()] < arr[i]) {

                  deq.pop_back(); //we want decreasing

     }

    deq.push_back(i)  // maintains  decreasing order
    if (i >= K-1){ // now we will be getting K-sized window;
          result.push_back(deq.front()); // max-element
    }
```

: Template :

```
}
```

K - size Subarray - max-element

Leetcod-239

"Sliding window Max"

Remember...

Queue (Deque) ⟶ window size (K) + Max/Min find

Monotonic Increasing ⟶ Minimum in window (deq.front());

Monotonic Decreasing ⟶ Maximum in window (deq.front());

→ Sliding window maximum/minimum ✓✓

→ Find the Power of K-size Subarrays I ✓✓

→ Shortest subarray with sum atleast K ✓✓

→ Constrained Subsequence Sum ✓✓

→ Jump Game VI (combination of Monotonic queue + DP) ✓✓