Monotonic Queues and Monotonic Optimization in Dynamic Programming

Sarah Zhang

February 2020

1 Introduction

A monotonic queue, also known as a double-ended queue, refers to a data structure in which each element monotonically increases or decreases with its subscript. The elements can be deleted at both the beginning and end of the queue, but they can only be inserted at the end. Since the elements are monotonic, the head element is either the maximum or minimum value depending on which monotonic queue you are constructing (decreasing or increasing order). Due to the nature of the double-ended queue, each element is enqueued once and dequeued at most once. Thus, the computational complexity of maintaining a monotonic queue is $O(n)$.

2 Example 1: Sliding Window Maximum/Minimum

2.1 Problem

Given an array A of size n, 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 maximum and minimum sliding window, respectively.

Input: A = [1, 3, -1, -3, 5, 3, 6, 7], and $k = 3$
Output: [3, 3, 5, 5, 6, 7] and [-1, -3, -3, -3, 3, 3]

Window Position	Maximum Sliding Window	Minimum Sliding Window
[1 3 -1] -3 5 3 6 7	3	-1
1 [3 -1 -3] 5 3 6 7	3	-3
1 3 [-1 -3 5] 3 6 7	5	-3
1 3 -1 [-3 5 3] 6 7	5	-3
1 3 -1 -3 [5 3 6] 7	6	3
1 3 -1 -3 5 [3 6 7]	7	3

2.2 Basic Solution

The basic method is to repeatedly scan each window to find the maximum or minimum value. We begin by looping through the array A of size n and then scanning the first k elements, the second k elements, the third k elements, and so forth. The time complexity for this method can be written as $O(N * k)$.

2.3 Segment Trees Solution

Other methods such as segment trees can find the maximum/minimum more efficiently than the basic method, but it has a time complexity of $O(NlogN)$ which is not linear.

2.4 Monotonic Queues Solution

To solve the sliding window maximum/minimum problem in linear time, we can utilize monotonic queues. The steps for maintaining a maximum monotonic queue are shown below:

1. Determine if the head of the queue is within the sliding window range. If it no longer in the window range, increase the head of the queue.

2. Every time when an element is added, it is compared with the end of the queue. If the new element is greater than the tail of the queue, then reduce the tail because the tail element will be removed. Continue to do this until the nature and constraints of the monotonic queue are satisfied.

3. Now the monotonic queue is in decreasing order and the head will contain the maximum value we are looking for.

Algorithm 1 Sliding Window Maximum with Monotonic Queues

```
$A[n+1] \leftarrow$ *input*
$Q[n+1] \leftarrow$ *monotonic queue maintained by values*
$N[n+1] \leftarrow$ *monotonic queue maintained by subscript*
$F[n+1] \leftarrow$ *store the maximum value of each sliding window*
function MONOTONIC_QUEUE_MAX
    $head \leftarrow 1$
    $tail \leftarrow 1$
    for $i = 1$ to $n$ do
        while <head is not within sliding window range> do
            $head \leftarrow head + 1$
        while <current value $\geq$ value of Q[tail]> do
            $tail \leftarrow tail - 1$
        $tail \leftarrow tail + 1$
        $N[tail] \leftarrow i$
        $Q[tail] \leftarrow A[i]$
        $F[i] \leftarrow Q[head]$
```

2.5 Summary & Recap

	Method	Time Complexity	Performance
1	Basic Method	$O(N * k)$	Poor
2	Segment Trees Method	$O(NlogN)$	Moderate
3	Monotonic Queues Method	$O(N)$	Best

3 Example 2: Maximum Continuous Sum

3.1 Problem

Given an integer array A with size n, find a subsequence with the maximum value that can't exceed m in length.

Input: A = [1, -3, 5, 1, -2, 3], $n = 6$, and $m = 4$
Output: 7

3.2 Solution

First, construct an array S which satisfies the following:

$$S[i] = \sum_{i=1}^{n} A[i]$$

We know that $S[0] = 0$ and $S[i] = S[i-1] + A[i]$. Let $F[i]$ be the maximum continuous sum from 1 to n.

$$F[i] = max \begin{cases} S[i] - S[i-1] \\ S[i] - S[i-2] \\ ... \\ S[i] - S[i-m] \end{cases}$$

Since $S[i]$ is a static value, it does not change and we can transform this function into:

$$F[i] = S[i] - min \begin{cases} S[i-1] \\ S[i-2] \\ ... \\ S[i-m] \end{cases}$$

The second part of this equation is simply a sliding window minimum problem and we can easily use monotonic queues to obtain the final answer.

Algorithm 2 Maximum Continuous Sum

```
$A[n+1] \leftarrow$ input
$S[n+1] \leftarrow$ store the continuous sum of the elements in A
$Q[n+1] \leftarrow$ monotonic queue maintained by values
$N[n+1] \leftarrow$ monotonic queue maintained by subscript
answer $\leftarrow$ -infinity
function MAXIMUM_CONTINUOUS_SUM
    $head \leftarrow 1$
    $tail \leftarrow 1$
    for $i = 1$ to $n$ do
        while <head is not within sliding window range> do
            $head \leftarrow head + 1$
        $answer \leftarrow max(answer, S[i] - Q[head])$
        while <current value < value of Q[tail]> do
            $tail \leftarrow tail - 1$
        $tail \leftarrow tail + 1$
        $N[tail] \leftarrow i$
        $Q[tail] \leftarrow S[i]$
```

4 Example 3: Mowing the Lawn (USACO Gold)

4.1 Problem

After winning the annual town competition for best lawn a year ago, Farmer John has grown lazy; he has not mowed the lawn since then and thus his lawn has become unruly. However, the competition is once again coming soon, and FJ would like to get his lawn into tiptop shape so that he can claim the title. Unfortunately, FJ has realized that his lawn is so unkempt that he will need to get some of his N ($1 \leq N \leq 100,000$) cows, who are lined up in a row and conveniently numbered $1...N$, to help him. Some cows are more efficient than others at mowing the lawn; cow i has efficiency E_i ($1 \leq \mathrm{E}_i \leq 100,000,000$). FJ has noticed that cows near each other in line often know each other well; he has also discovered that if he chooses more than K ($1 \leq K \leq N$) consecutive (adjacent) cows to help him, they will ignore the lawn and start a party instead. Thus, FJ needs you to assist him: determine the largest total cow efficiency FJ can obtain without choosing more than K consecutive cows.

Input:
$N = 5$, $K = 2$
1 2 3 4 5
Output: 12

4.2 Solution

Since each cow may or may not be chosen to mow the lawn, we use 0 to represent the cows that are not chosen and 1 to represent the cows that are chosen. Let $DP[i][0]$ represent a cow i that is not chosen to mow the lawn and $DP[i][1]$ represent a cow i that is chosen to mow the lawn. We know that $S[0] = 0$ and $S[i] = S[i-1] + A[i]$. Let $DP[i][0]$ or $DP[i][1]$ be the maximum total cow efficiency for cow i.

$$DP[i][0] = max \begin{cases} DP[i-1][0] \\ DP[i-1][1] \end{cases}$$

$$DP[i][1] = max \begin{cases} S[i] - S[i-1] + DP[i-1][0] \\ S[i] - S[i-2] + DP[i-2][0] \\ ... \\ S[i] - S[i-K] + DP[i-K][0] \end{cases}$$

Since $S[i]$ is a static value, it does not change and we can transform this function into:

$$F[i] = S[i] + max \begin{cases} DP[i-1][0] - S[i-1] \\ DP[i-2][0] - S[i-2] \\ ... \\ DP[i-K][0] - S[i-K] \end{cases}$$

The second part of this equation is simply a sliding window maximum problem and we can easily use monotonic queues to obtain the final answer.

Algorithm 3 Mowing the Lawn

```
$A[N+1] \leftarrow$ *input*
$S[N+1] \leftarrow$ *store the continuous sum of the elements in A*
$Q[N+1] \leftarrow$ *monotonic queue maintained by values*
$DP[N+1][N+1] \leftarrow$ *store the maximum total cow efficiency*
**function** MOWING_THE_LAWN
    $head \leftarrow 1$
    $tail \leftarrow 1$
    **for** $i = 1$ to $N$ **do**
        $DP[i][0] \leftarrow max(DP[i-1][0], DP[i-1][1])$
        **while** <head is not within sliding window range> **do**
            $head \leftarrow head + 1$
        $DP[i][1] \leftarrow S[i] - S[Q[head]] + DP[Q[head]][0]$
        **while** $DP[i][0] - S[i] > DP[Q[tail]][0] - S[Q[tail]]$ **do**
            $tail \leftarrow tail - 1$
        $tail \leftarrow tail + 1$
        $Q[tail] \leftarrow i$
    $answer = max(DP[N][0], DP[N][1])$
```

5 Conclusion

The underlying pattern for a monotonic queue problem is shown below:

$$F[i] = A[i] \pm max/min \begin{cases} B[i-1] \pm C[i-1] \\ B[i-2] \pm C[i-2] \\ ... \\ B[i-k] \pm C[i-k] \end{cases}$$

Since $A[i]$ is a static value, it is irrelevant to the max/min block. Remember that when you see this type of pattern, the problem can be easily solved and optimized with monotonic queues in $O(n)$ time.