# LeetCode Training Day 19 Monotone Stack

We all know queue is a first in, first out data structure and stack is last in first out.

Monotone stack is that we try to keep a series of elements in as ascending order or descending order to get a minimum or maximum permutation.

In some case we want to add a slide window for monotone stack, then we can use a data structure called deque which allow you to pop items from front and push and pop items at the back.

## 496. Next Greater Element I

Easy

The **next greater element** of some element x in an array is the **first greater** element that is **to the right** of x in the same array.

You are given two **distinct 0-indexed** integer arrays nums1 and nums2, where nums1 is a subset of nums2.

For each 0 <= i < nums1.length, find the index j such that nums1[i] == nums2[j] and determine the **next greater element** of nums2[j] in nums2. If there is no next greater element, then the answer for this query is -1.

Return *an array*ans*of length*nums1.length*such that*ans[i]*is the****next greater element****as described above.*

**Example 1:**

**Input:** nums1 = [4,1,2], nums2 = [1,3,4,2]

**Output:** [-1,3,-1]

**Explanation:** The next greater element for each value of nums1 is as follows:

- 4 is underlined in nums2 = [1,3,4,2]. There is no next greater element, so the answer is -1.

- 1 is underlined in nums2 = [1,3,4,2]. The next greater element is 3.

- 2 is underlined in nums2 = [1,3,4,2]. There is no next greater element, so the answer is -1.

**Example 2:**

**Input:** nums1 = [2,4], nums2 = [1,2,3,4]

**Output:** [3,-1]

**Explanation:** The next greater element for each value of nums1 is as follows:

- 2 is underlined in nums2 = [1,2,3,4]. The next greater element is 3.

- 4 is underlined in nums2 = [1,2,3,4]. There is no next greater element, so the answer is -1.

**Constraints:**

* 1 <= nums1.length <= nums2.length <= 1000
* 0 <= nums1[i], nums2[i] <= 104
* All integers in nums1 and nums2 are **unique**.
* All the integers of nums1 also appear in nums2.

**Follow up:** Could you find an O(nums1.length + nums2.length) solution?

### Analysis:

This problem actually ask you to look for the next greater number in nums2, then you use hash table to map the answer to nums1, since nums1 is a subset of nums2.

/// <summary>

/// Leet code #496. Next Greater Element I

///

/// You are given two arrays (without duplicates) nums1 and nums2 where nums1’s elements

/// are subset of nums2. Find all the next greater numbers for nums1's elements in the

/// corresponding places of nums2.

///

/// The Next Greater Number of a number x in nums1 is the first greater number to its right

/// in nums2. If it does not exist, output -1 for this number.

///

/// Example 1:

/// Input: nums1 = [4,1,2], nums2 = [1,3,4,2].

/// Output: [-1,3,-1]

/// Explanation:

/// For number 4 in the first array, you cannot find the next greater number for it in

/// the second array, so output -1.

/// For number 1 in the first array, the next greater number for it in the second array is 3.

/// For number 2 in the first array, there is no next greater number for it in the second array,

/// so output -1.

///

/// Example 2:

///

/// Input: nums1 = [2,4], nums2 = [1,2,3,4].

/// Output: [3,-1]

/// Explanation:

/// For number 2 in the first array, the next greater number for it in the second array is 3.

/// For number 4 in the first array, there is no next greater number for it in the second array,

/// so output -1.

///

/// Note:

///

/// 1.All elements in nums1 and nums2 are unique.

/// 2.The length of both nums1 and nums2 would not exceed 1000.

/// </summary>

vector<int> LeetCodeStack::nextGreaterElement(vector<int>& nums1, vector<int>& nums2)

{

vector<int> result;

map<int, int> num\_map;

stack<int> stack;

for (size\_t i = 0; i < nums2.size(); i++)

{

while (!stack.empty() && stack.top() < nums2[i])

{

num\_map[stack.top()] = nums2[i];

stack.pop();

}

stack.push(nums2[i]);

}

for (size\_t i = 0; i < nums1.size(); i++)

{

if (num\_map.count(nums1[i]) > 0)

{

result.push\_back(num\_map[nums1[i]]);

}

else

{

result.push\_back(-1);

}

}

return result;

}

## 503. Next Greater Element II

Medium

Given a circular integer array nums (i.e., the next element of nums[nums.length - 1] is nums[0]), return *the****next greater number****for every element in* nums.

The **next greater number** of a number x is the first greater number to its traversing-order next in the array, which means you could search circularly to find its next greater number. If it doesn't exist, return -1 for this number.

**Example 1:**

**Input:** nums = [1,2,1]

**Output:** [2,-1,2]

Explanation: The first 1's next greater number is 2;

The number 2 can't find next greater number.

The second 1's next greater number needs to search circularly, which is also 2.

**Example 2:**

**Input:** nums = [1,2,3,4,3]

**Output:** [2,3,4,-1,4]

**Constraints:**

* 1 <= nums.length <= 104
* -109 <= nums[i] <= 109

### Analysis:

You just need to process the num2 two iterations. For all index more than nums2.size, you can ignore.

/// <summary>

/// Leet code #503. Next Greater Element II

///

/// Given a circular array (the next element of the last element is the

/// first element of the array), print the Next Greater Number for

/// every element. The Next Greater Number of a number x is the first

/// greater number to its traversing-order next in the array, which means

/// you could search circularly to find its next greater number. If it

/// doesn't exist, output -1 for this number.

///

/// Example 1:

///

/// Input: [1,2,1]

/// Output: [2,-1,2]

/// Explanation: The first 1's next greater number is 2;

/// The number 2 can't find next greater number;

/// The second 1's next greater number needs to search circularly, which is also 2.

/// </summary>

vector<int> LeetCodeStack::nextGreaterElements(vector<int>& nums)

{

vector<int> result(nums.size(), -1);

stack<pair<int, int>> stack;

for (size\_t i = 0; i < 2 \* nums.size(); i++)

{

while (!stack.empty() && stack.top().first < nums[i % nums.size()])

{

result[stack.top().second] = nums[i % nums.size()];

stack.pop();

}

if (i < nums.size())

{

stack.push(make\_pair(nums[i], i));

}

}

return result;

}

## 1475. Final Prices With a Special Discount in a Shop

Easy

Given the array prices where prices[i] is the price of the ith item in a shop. There is a special discount for items in the shop, if you buy the ith item, then you will receive a discount equivalent to prices[j] where j is the **minimum** index such that j > i and prices[j] <= prices[i], otherwise, you will not receive any discount at all.

*Return an array where the ith element is the final price you will pay for the ith item of the shop considering the special discount.*

**Example 1:**

**Input:** prices = [8,4,6,2,3]

**Output:** [4,2,4,2,3]

**Explanation:**

For item 0 with price[0]=8 you will receive a discount equivalent to prices[1]=4, therefore, the final price you will pay is 8 - 4 = 4.

For item 1 with price[1]=4 you will receive a discount equivalent to prices[3]=2, therefore, the final price you will pay is 4 - 2 = 2.

For item 2 with price[2]=6 you will receive a discount equivalent to prices[3]=2, therefore, the final price you will pay is 6 - 2 = 4.

For items 3 and 4 you will not receive any discount at all.

**Example 2:**

**Input:** prices = [1,2,3,4,5]

**Output:** [1,2,3,4,5]

**Explanation:** In this case, for all items, you will not receive any discount at all.

**Example 3:**

**Input:** prices = [10,1,1,6]

**Output:** [9,0,1,6]

**Constraints:**

1 <= prices.length <= 500

1 <= prices[i] <= 10^3

### Analysis:

For every price, find a higher or equal price before and discount it. Keep all unprocessed price in stack

/// <summary>

/// Leet code #1475. Final Prices With a Special Discount in a Shop

///

/// Easy

///

/// Given the array prices where prices[i] is the price of the ith item

/// in a shop. There is a special discount for items in the shop, if you

/// buy the ith item, then you will receive a discount equivalent to

/// prices[j] where j is the minimum index such that j > i and

/// prices[j] <= prices[i], otherwise, you will not receive any discount

/// at all.

///

/// Return an array where the ith element is the final price you will pay

/// for the ith item of the shop considering the special discount.

///

/// Example 1:

/// Input: prices = [8,4,6,2,3]

/// Output: [4,2,4,2,3]

/// Explanation:

/// For item 0 with price[0]=8 you will receive a discount equivalent to

/// prices[1]=4, therefore, the final price you will pay is 8 - 4 = 4.

/// For item 1 with price[1]=4 you will receive a discount equivalent to

/// prices[3]=2, therefore, the final price you will pay is 4 - 2 = 2.

/// For item 2 with price[2]=6 you will receive a discount equivalent to

/// prices[3]=2, therefore, the final price you will pay is 6 - 2 = 4.

/// For items 3 and 4 you will not receive any discount at all.

///

/// Example 2:

/// Input: prices = [1,2,3,4,5]

/// Output: [1,2,3,4,5]

/// Explanation: In this case, for all items, you will not receive any

/// discount at all.

/// Example 3:

/// Input: prices = [10,1,1,6]

/// Output: [9,0,1,6]

/// Constraints:

/// 1. 1 <= prices.length <= 500

/// 2. 1 <= prices[i] <= 10^3

/// </summary>

vector<int> LeetCodeStack::finalPrices(vector<int>& prices)

{

vector<int> result;

stack<pair<int, int>> stack;

for (size\_t i = 0; i < prices.size(); i++)

{

while ((!stack.empty()) && (stack.top().first >= prices[i]))

{

result[stack.top().second] -= prices[i];

stack.pop();

}

stack.push(make\_pair(prices[i], i));

result.push\_back(prices[i]);

}

return result;

}

## 316. Remove Duplicate Letters

Medium

Given a string s, remove duplicate letters so that every letter appears once and only once. You must make sure your result is **the smallest in lexicographical order** among all possible results.

**Example 1:**

**Input:** s = "bcabc"

**Output:** "abc"

**Example 2:**

**Input:** s = "cbacdcbc"

**Output:** "acdb"

**Constraints:**

* 1 <= s.length <= 104
* s consists of lowercase English letters.

### Analysis:

First you count all the distinct characters, then track the result string in stack, when you see a reverse order, if the previous character will appear again in later sequence, you know you can pop it up.

/// <summary>

/// Leet code #316. Remove Duplicate Letters

///

/// Given a string which contains only lowercase letters, remove duplicate

/// letters so that every letter appear once and only once. You must make sure

/// your result is the smallest in lexicographical order among all possible

/// results.

///

/// Example:

///

/// Given "bcabc"

/// Return "abc"

///

/// Given "cbacdcbc"

/// Return "acdb"

/// </summary>

string LeetCodeStack::removeDuplicateLetters(string s)

{

// count remaining characters,

vector<int> char\_count(26);

// count in string characters

vector<int> char\_used(26);

string result;

for (size\_t i = 0; i < s.size(); i++)

{

char\_count[s[i] - 'a']++;

}

for (size\_t i = 0; i < s.size(); i++)

{

if (char\_used[s[i] - 'a'] == 0)

{

// pop out duplicate and bigger characters.

while (!result.empty() && result.back() >= s[i] &&

char\_count[result.back() - 'a'] > 0)

{

char\_used[result.back() - 'a']--;

result.pop\_back();

}

result.push\_back(s[i]);

char\_used[s[i] - 'a']++;

}

char\_count[s[i] - 'a']--;

}

return result;

}

## 402. Remove K Digits

Medium

Given string num representing a non-negative integer num, and an integer k, return *the smallest possible integer after removing* k *digits from* num.

**Example 1:**

**Input:** num = "1432219", k = 3

**Output:** "1219"

**Explanation:** Remove the three digits 4, 3, and 2 to form the new number 1219 which is the smallest.

**Example 2:**

**Input:** num = "10200", k = 1

**Output:** "200"

**Explanation:** Remove the leading 1 and the number is 200. Note that the output must not contain leading zeroes.

**Example 3:**

**Input:** num = "10", k = 2

**Output:** "0"

**Explanation:** Remove all the digits from the number and it is left with nothing which is 0.

**Constraints:**

1 <= k <= num.length <= 105

num consists of only digits.

num does not have any leading zeros except for the zero itself.

### Analysis:

Start to pop up all bigger numbers ahead, in the end skip leading 0.

/// <summary>

/// Leet code #402. Remove K Digits

///

/// Given a non-negative integer num represented as a string, remove k digits

/// from the number so that the new number is the smallest possible.

/// Note:

/// The length of num is less than 10002 and will be ≥ k.

/// The given num does not contain any leading zero.

/// Example 1:

/// Input: num = "1432219", k = 3

/// Output: "1219"

/// Explanation: Remove the three digits 4, 3, and 2 to form the new number

/// 1219 which is the smallest.

///

/// Example 2:

/// Input: num = "10200", k = 1

/// Output: "200"

/// Explanation: Remove the leading 1 and the number is 200. Note that the

/// output must not contain leading zeroes.

///

/// Example 3:

/// Input: num = "10", k = 2

/// Output: "0"

/// Explanation: Remove all the digits from the number and it is left with

/// nothing which is 0.

/// </summary>

string LeetCodeStack::removeKdigits(string num, int k)

{

string result;

// pop up big leading digits from front

for (size\_t i = 0; i < num.size(); i++)

{

while (!result.empty() && (num[i] < result.back()) && k > 0)

{

result.pop\_back();

k--;

}

if (!result.empty() || num[i] != '0') result.push\_back(num[i]);

}

// pop up extra digits from end

while (!result.empty() && k > 0)

{

result.pop\_back();

k--;

}

if (result.empty()) result = "0";

return result;

}

## 739. Daily Temperatures

Medium

Given an array of integers temperatures represents the daily temperatures, return *an array* answer *such that* answer[i] *is the number of days you have to wait after the* ith *day to get a warmer temperature*. If there is no future day for which this is possible, keep answer[i] == 0 instead.

**Example 1:**

**Input:** temperatures = [73,74,75,71,69,72,76,73]

**Output:** [1,1,4,2,1,1,0,0]

**Example 2:**

**Input:** temperatures = [30,40,50,60]

**Output:** [1,1,1,0]

**Example 3:**

**Input:** temperatures = [30,60,90]

**Output:** [1,1,0]

**Constraints:**

1 <= temperatures.length <= 105

30 <= temperatures[i] <= 100

### Analysis:

On day x, you check all the previous days and see which days with temperature lower than today, and pop them up from stack, mark the result. In the end all the days without processed, are the ones you can not get a day warm than that day.

/// <summary>

/// Leet code #739. Daily Temperatures

///

/// Given a list of daily temperatures, produce a list that, for each day

/// in the input, tells you how many days you would have to wait until a

/// warmer temperature. If there is no future day for which this is

/// possible, put 0 instead.

/// For example, given the list temperatures = [73, 74, 75, 71, 69, 72,

/// 76, 73], your output should be [1, 1, 4, 2, 1, 1, 0, 0].

/// Note: The length of temperatures will be in the range [1, 30000]. Each

/// temperature will be an integer in the range [30, 100].

/// </summary>

vector<int> LeetCodeStack::dailyTemperatures(vector<int>& temperatures)

{

vector<int> result(temperatures.size());

stack<pair<int, int>> temp\_stack;

for (size\_t i = 0; i < temperatures.size(); i++)

{

while (!temp\_stack.empty())

{

auto temperature = temp\_stack.top();

// if new temperature is lower then simply push it.

if (temperature.first >= temperatures[i]) break;

// set the day and pop up

result[temperature.second] = i - temperature.second;

temp\_stack.pop();

}

temp\_stack.push(make\_pair(temperatures[i], i));

}

return result;

}

## 1081. Smallest Subsequence of Distinct Characters

Medium

Return the lexicographically smallest subsequence of text that contains all the distinct characters of text exactly once.

**Example 1:**

**Input:** "cdadabcc"

**Output:** "adbc"

**Example 2:**

**Input:** "abcd"

**Output:** "abcd"

**Example 3:**

**Input:** "ecbacba"

**Output:** "eacb"

**Example 4:**

**Input:** "leetcode"

**Output:** "letcod"

**Note:**

1. 1 <= text.length <= 1000
2. text consists of lowercase English letters.

### Analysis:

First you count all the distinct characters, then track the result string in stack, when you see a reverse order, if the previous character will appear again in later sequence, you know you can pop it up.

/// <summary>

/// Leet code #1081. Smallest Subsequence of Distinct Characters

///

/// Return the lexicographically smallest subsequence of text that contains

/// all the distinct characters of text exactly once.

///

/// Example 1:

/// Input: "cdadabcc"

/// Output: "adbc"

///

/// Example 2:

/// Input: "abcd"

/// Output: "abcd"

///

/// Example 3:

/// Input: "ecbacba"

/// Output: "eacb"

///

/// Example 4:

/// Input: "leetcode"

/// Output: "letcod"

///

/// Note:

///

/// 1. 1 <= text.length <= 1000

/// 2. text consists of lowercase English letters.

/// </summary>

string LeetCodeStack::smallestSubsequence(string text)

{

vector<int> count(26), used(26);

for (size\_t i = 0; i < text.size(); i++)

{

count[text[i] - 'a']++;

}

string result;

for (size\_t i = 0; i < text.size(); i++)

{

if (result.empty())

{

result.push\_back(text[i]);

used[x] = 1;

}

else

{

int x = text[i] - 'a';

if (used[x] == 1)

{

count[x]--;

continue;

}

while (!result.empty())

{

char ch = result.back();

if (text[i] > ch) break;

int k = ch - 'a';

if (count[k] == 1) break;

count[k]--;

used[k] = 0;

result.pop\_back();

}

result.push\_back(text[i]);

used[x] = 1;

}

}

return result;

}

## 907. Sum of Subarray Minimums

Medium

Given an array of integers arr, find the sum of min(b), where b ranges over every (contiguous) subarray of arr. Since the answer may be large, return the answer **modulo** 109 + 7.

**Example 1:**

**Input:** arr = [3,1,2,4]

**Output:** 17

**Explanation:**

Subarrays are [3], [1], [2], [4], [3,1], [1,2], [2,4], [3,1,2], [1,2,4], [3,1,2,4].

Minimums are 3, 1, 2, 4, 1, 1, 2, 1, 1, 1.

Sum is 17.

**Example 2:**

**Input:** arr = [11,81,94,43,3]

**Output:** 444

**Constraints:**

* 1 <= arr.length <= 3 \* 104
* 1 <= arr[i] <= 3 \* 104

### Analysis:

We keep an increasing monotone stack with item value and index, when we scan to a new position in the original array, we focus on calculating all the subarray ending with that position, so first we pop out all the elements in stack greater or equal to the item, because we know that current item is minimum for all these subarray, then we calculate the distance between the current item and previous item and this is how many subarray which current minimum item can impact, we calculate the sum. For the items which are smaller current item the previous calculated sum can be simply inherited.

/// <summary>

/// Leet code #907. Sum of Subarray Minimums

///

/// Given an array of integers A, find the sum of min(B), where B ranges over

/// every (contiguous) subarray of A.

///

/// Since the answer may be large, return the answer modulo 10^9 + 7.

///

/// Example 1:

///

/// Input: [3,1,2,4]

/// Output: 17

/// Explanation: Subarrays are [3], [1], [2], [4], [3,1], [1,2], [2,4],

/// [3,1,2], [1,2,4], [3,1,2,4].

/// Minimums are 3, 1, 2, 4, 1, 1, 2, 1, 1, 1. Sum is 17.

///

/// Note:

///

/// 1. 1 <= A.length <= 30000

/// 2. 1 <= A[i] <= 30000

/// </summary>

int LeetCodeStack::sumSubarrayMins(vector<int>& A)

{

// A array of pair, dp[i][0] is the position in A, dp[i][1] is the value of

// and dp[i][2] is the partial sum

size\_t M = 1000000007;

vector<vector<int>> dp;

dp.push\_back({ -1, 0, 0 });

int result = 0;

for (int i = 0; i < (int)A.size(); i++)

{

// kick out all the values greater than current

while (dp.back()[1] > A[i]) dp.pop\_back();

// starting from one position after previous smaller element to current

// all subarry choose current value

int sum = A[i] \* (i - dp.back()[0]);

sum %= M;

// for any subarry ending current but including previous smaller number

// and the more smaller number before previous...

sum += dp.back()[2];

sum %= M;

result += sum;

result %= M;

dp.push\_back({ i, A[i], sum });

}

return result;

}

# Advanced Problems

## 239. Sliding Window Maximum

Hard

You are given an array of integers nums, 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*.

**Example 1:**

**Input:** nums = [1,3,-1,-3, -9, 5,3,6,7], k = 3

**Output:** [3,3,-1,5,5,6,7]

**Explanation:**

Window position Max

--------------- -----

[1 3 -1] -3 5 3 6 7 **3**

1 [3 -1 -3] 5 3 6 7 **3**

1 3 [-1 -3 5] 3 6 7  **5**

1 3 -1 [-3 5 3] 6 7 **5**

1 3 -1 -3 [5 3 6] 7 **6**

1 3 -1 -3 5 [3 6 7] **7**

**Example 2:**

**Input:** nums = [1], k = 1

**Output:** [1]

**Example 3:**

**Input:** nums = [1,-1], k = 1

**Output:** [1,-1]

**Example 4:**

**Input:** nums = [9,11], k = 2

**Output:** [11]

**Example 5:**

**Input:** nums = [4,-2], k = 2

**Output:** [4]

**Constraints:**

1 <= nums.length <= 105

-104 <= nums[i] <= 104

1 <= k <= nums.length

### Analysis:

Keep monotone descrease stack in a slide window queue, for this purpose you need to use deque, which you can pop up from both ends. If there is no deque in the programming language, you may need to implement your double linked list.

/// <summary>

/// Leet code #239. Sliding Window Maximum

///

/// Given an array nums, 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.

/// For example,

/// Given nums = [1,3,-1,-3,5,3,6,7], and k = 3.

/// Window position Max

/// --------------- -----

/// [1 3 -1] -3 5 3 6 7 3

/// 1 [3 -1 -3] 5 3 6 7 3

/// 1 3 [-1 -3 5] 3 6 7 5

/// 1 3 -1 [-3 5 3] 6 7 5

/// 1 3 -1 -3 [5 3 6] 7 6

/// 1 3 -1 -3 5 [3 6 7] 7

/// Therefore, return the max sliding window as [3,3,5,5,6,7].

/// Note:

/// You may assume k is always valid, ie: 1 ¡Ü k ¡Ü input array's size for

/// non-empty array.

/// Follow up:

/// Could you solve it in linear time?

/// Hint:

/// 1.How about using a data structure such as deque (double-ended queue)?

/// 2.The queue size need not be the same as the window¡¯s size.

/// 3.Remove redundant elements and the queue should store only elements

/// that need to be considered.

/// </summary>

vector<int> LeetCodeStack::maxSlidingWindow(vector<int>& nums, int k)

{

vector<int> result;

deque<pair<int, int>> max\_window;

for (size\_t i = 0; i < nums.size(); i++)

{

if (max\_window.empty())

{

max\_window.push\_back(make\_pair(i, nums[i]));

}

else

{

if ((int)i - k == max\_window.front().first) max\_window.pop\_front();

while (!max\_window.empty() && max\_window.back().second < nums[i])

{

max\_window.pop\_back();

}

max\_window.push\_back(make\_pair(i, nums[i]));

}

if (i >= k - 1)

{

result.push\_back(max\_window.front().second);

}

}

return result;

}

## 1438. Longest Continuous Subarray With Absolute Diff Less Than or Equal to Limit

Medium

Given an array of integers nums and an integer limit, return the size of the longest **non-empty** subarray such that the absolute difference between any two elements of this subarray is less than or equal to limit*.*

**Example 1:**

**Input:** nums = [8,2,4,7], limit = 4

**Output:** 2

**Explanation:** All subarrays are:

[8] with maximum absolute diff |8-8| = 0 <= 4.

[8,2] with maximum absolute diff |8-2| = 6 > 4.

[8,2,4] with maximum absolute diff |8-2| = 6 > 4.

[8,2,4,7] with maximum absolute diff |8-2| = 6 > 4.

[2] with maximum absolute diff |2-2| = 0 <= 4.

[2,4] with maximum absolute diff |2-4| = 2 <= 4.

[2,4,7] with maximum absolute diff |2-7| = 5 > 4.

[4] with maximum absolute diff |4-4| = 0 <= 4.

[4,7] with maximum absolute diff |4-7| = 3 <= 4.

[7] with maximum absolute diff |7-7| = 0 <= 4.

Therefore, the size of the longest subarray is 2.

**Example 2:**

**Input:** nums = [10,1,2,4,7,2], limit = 5

**Output:** 4

**Explanation:** The subarray [2,4,7,2] is the longest since the maximum absolute diff is |2-7| = 5 <= 5.

**Example 3:**

**Input:** nums = [4,2,2,2,4,4,2,2], limit = 0

**Output:** 3

**Constraints:**

* 1 <= nums.length <= 105
* 1 <= nums[i] <= 109
* 0 <= limit <= 109

### Analysis:

Keep maximum number in decreasing order, and minimum number in increasing order to indicating from current index to end of scanning point what is the maximum number and what is the minimum number in the sub array.

/// <summary>

/// Leet code #1438. Longest Continuous Subarray With Absolute Diff

/// Less Than or Equal to Limit

///

/// Medium

///

/// Given an array of integers nums and an integer limit, return the

/// size of the longest continuous subarray such that the absolute

/// difference between any two elements is less than or equal to limit.

///

/// In case there is no subarray satisfying the given condition return 0.

///

/// Example 1:

/// Input: nums = [8,2,4,7], limit = 4

/// Output: 2

/// Explanation: All subarrays are:

/// [8] with maximum absolute diff |8-8| = 0 <= 4.

/// [8,2] with maximum absolute diff |8-2| = 6 > 4.

/// [8,2,4] with maximum absolute diff |8-2| = 6 > 4.

/// [8,2,4,7] with maximum absolute diff |8-2| = 6 > 4.

/// [2] with maximum absolute diff |2-2| = 0 <= 4.

/// [2,4] with maximum absolute diff |2-4| = 2 <= 4.

/// [2,4,7] with maximum absolute diff |2-7| = 5 > 4.

/// [4] with maximum absolute diff |4-4| = 0 <= 4.

/// [4,7] with maximum absolute diff |4-7| = 3 <= 4.

/// [7] with maximum absolute diff |7-7| = 0 <= 4.

/// Therefore, the size of the longest subarray is 2.

///

/// Example 2:

/// Input: nums = [10,1,2,4,7,2], limit = 5

/// Output: 4

/// Explanation: The subarray [2,4,7,2] is the longest since the maximum

/// absolute diff is |2-7| = 5 <= 5.

///

/// Example 3:

/// Input: nums = [4,2,2,2,4,4,2,2], limit = 0

/// Output: 3

///

/// Constraints:

/// 1. 1 <= nums.length <= 10^5

/// 2. 1 <= nums[i] <= 10^9

/// 3. 0 <= limit <= 10^9

/// </summary>

int LeetCodeArray::longestSubarray(vector<int>& nums, int limit)

{

deque<int> min\_list;

deque<int> max\_list;

int first = 0;

int last = 0;

int result = 0;

min\_list.push\_back(nums[0]);

max\_list.push\_back(nums[0]);

while (last < (int)nums.size())

{

if (max\_list.front() - min\_list.front() <= limit)

{

result = max(result, last - first + 1);

last++;

if (last < (int)nums.size())

{

while (!min\_list.empty() && min\_list.back() > nums[last]) min\_list.pop\_back();

while (!max\_list.empty() && max\_list.back() < nums[last]) max\_list.pop\_back();

min\_list.push\_back(nums[last]);

max\_list.push\_back(nums[last]);

}

}

else

{

if (!min\_list.empty() && min\_list.front() == nums[first]) min\_list.pop\_front();

if (!max\_list.empty() && max\_list.front() == nums[first]) max\_list.pop\_front();

first++;

}

}

return result;

}

## 2030. Smallest K-Length Subsequence With Occurrences of a Letter

Hard

You are given a string s, an integer k, a letter letter, and an integer repetition.

Return *the****lexicographically smallest****subsequence of* s*of length* k *that has the letter* letter *appear****at least*** repetition *times*. The test cases are generated so that the letter appears in s **at least** repetition times.

A **subsequence** is a string that can be derived from another string by deleting some or no characters without changing the order of the remaining characters.

A string a is **lexicographically smaller** than a string b if in the first position where a and b differ, string a has a letter that appears earlier in the alphabet than the corresponding letter in b.

**Example 1:**

**Input:** s = "leet", k = 3, letter = "e", repetition = 1

**Output:** "eet"

**Explanation:** There are four subsequences of length 3 that have the letter 'e' appear at least 1 time:

- "lee" (from "**lee**t")

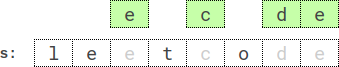
- "let" (from "**le**e**t**")

- "let" (from "**l**e**et**")

- "eet" (from "l**eet**")

The lexicographically smallest subsequence among them is "eet".

**Example 2:**



**Input:** s = "leetcode", k = 4, letter = "e", repetition = 2

**Output:** "ecde"

**Explanation:** "ecde" is the lexicographically smallest subsequence of length 4 that has the letter "e" appear at least 2 times.

**Example 3:**

**Input:** s = "bb", k = 2, letter = "b", repetition = 2

**Output:** "bb"

**Explanation:** "bb" is the only subsequence of length 2 that has the letter "b" appear at least 2 times.

**Constraints:**

* 1 <= repetition <= k <= s.length <= 5 \* 104
* s consists of lowercase English letters.
* letter is a lowercase English letter, and appears in s at least repetition times.

### Analysis:

Pop up decreasing order characters if we have more than k occurrence for that letter.

/// <summary>

/// Leet Code 2030. Smallest K-Length Subsequence With Occurrences

/// of a Letter

///

/// Hard

///

/// You are given a string s, an integer k, a letter letter, and an

/// integer repetition.

///

/// Return the lexicographically smallest subsequence of s of length

/// k that has the letter letter appear at least repetition times.

/// The test cases are generated so that the letter appears in s at

/// least repetition times.

///

/// A subsequence is a string that can be derived from another string

/// by deleting some or no characters without changing the order of

/// the remaining characters.

///

/// A string a is lexicographically smaller than a string b if in

/// the first position where a and b differ, string a has a letter

/// that appears earlier in the alphabet than the corresponding letter

/// in b.

///

/// Example 1:

/// Input: s = "leet", k = 3, letter = "e", repetition = 1

/// Output: "eet"

/// Explanation: There are four subsequences of length 3 that have

/// the letter 'e' appear at least 1 time:

/// - "lee" (from "leet")

/// - "let" (from "leet")

/// - "let" (from "leet")

/// - "eet" (from "leet")

/// The lexicographically smallest subsequence among them is "eet".

///

/// Example 2:

/// Input: s = "leetcode", k = 4, letter = "e", repetition = 2

/// Output: "ecde"

/// Explanation: "ecde" is the lexicographically smallest

/// subsequence of length 4 that has the letter "e" appear at

/// least 2 times.

///

/// Example 3:

/// Input: s = "bb", k = 2, letter = "b", repetition = 2

/// Output: "bb"

/// Explanation: "bb" is the only subsequence of length 2

/// that has the letter "b" appear at least 2 times.

///

/// Constraints:

/// 1. 1 <= repetition <= k <= s.length <= 5 \* 10^4

/// 2. s consists of lowercase English letters.

/// 3. letter is a lowercase English letter, and appears in s at

/// least repetition times.

/// </summary>

string LeetCodeStack::smallestSubsequence(string s, int k, char letter, int repetition)

{

vector<int> dp(s.size());

int count = 0;

for (int i = s.size() - 1; i >= 0; i--)

{

if (s[i] == letter)

{

count++;

}

dp[i] = count;

}

string result;

int letter\_count = 0;

for (size\_t i = 0; i < s.size(); i++)

{

if (!result.empty() && result.back() > s[i])

{

if ((result.size() + (s.size() - i) > k) &&

((letter\_count + dp[i] > repetition) ||

(letter\_count + dp[i] == repetition && result.back() != letter)))

{

if (result.back() == letter) letter\_count--;

result.pop\_back();

i--;

continue;

}

}

result.push\_back(s[i]);

if (s[i] == letter) letter\_count++;

}

string tail;

while (result.size() + tail.size() > k)

{

if (result.back() == letter)

{

if (letter\_count == repetition)

{

tail.push\_back(result.back());

}

else

{

letter\_count--;

}

}

result.pop\_back();

}

result.append(tail);

return result;

}

## 84. Largest Rectangle in Histogram

Hard

Given an array of integers heights representing the histogram's bar height where the width of each bar is 1, return *the area of the largest rectangle in the histogram*.

**Example 1:**

A picture containing text, clock

Description automatically generated

**Input:** heights = [2,1,5,6,2,3]

**Output:** 10

**Explanation:** The above is a histogram where width of each bar is 1.

The largest rectangle is shown in the red area, which has an area = 10 units.

**Example 2:**

A picture containing text, clock, sign, clipart

Description automatically generated

**Input:** heights = [2,4]

**Output:** 4

**Constraints:**

* 1 <= heights.length <= 105
* 0 <= heights[i] <= 104

### Analysis:

If a bar is higher than previous, push to stack, if the bar is lower than previous one, keep pop up all bars higher than this one and calculate area, then push the new bar. In the end pop up all bars and calculate result.

/// <summary>

/// Leet Code 84. Largest Rectangle in Histogram

///

/// Hard

///

/// Given an array of integers heights representing the histogram's bar

/// height where the width of each bar is 1, return the area of the

/// largest rectangle in the histogram.

///

/// Example 1:

/// Input: heights = [2,1,5,6,2,3]

/// Output: 10

/// Explanation: The above is a histogram where width of each bar is 1.

/// The largest rectangle is shown in the red area, which has an

/// area = 10 units.

///

/// Example 2:

/// Input: heights = [2,4]

/// Output: 4

///

/// Constraints:

/// 1. 1 <= heights.length <= 10^5

/// 2. 0 <= heights[i] <= 10^4

/// </summary>

int LeetCodeStack::largestRectangleArea(vector<int>& heights)

{

int max\_area = 0;

stack<pair<int, int>> height\_stack;

for (size\_t i = 0; i <= heights.size(); i++)

{

int height = (i == heights.size()) ? 0 : heights[i];

if (height\_stack.empty() || (height > height\_stack.top().second))

{

height\_stack.push(make\_pair(i, height));

}

else

{

int end = i;

pair<int, int> pair;

while ((!height\_stack.empty()) && (height <= height\_stack.top().second))

{

pair = height\_stack.top();

height\_stack.pop();

max\_area = max(max\_area, (end - pair.first) \* pair.second);

}

height\_stack.push(make\_pair(pair.first, height));

}

}

return max\_area;

}

## 1425. Constrained Subsequence Sum

Hard

Given an integer array nums and an integer k, return the maximum sum of a **non-empty** subsequence of that array such that for every two **consecutive** integers in the subsequence, nums[i] and nums[j], where i < j, the condition j - i <= k is satisfied.

A *subsequence* of an array is obtained by deleting some number of elements (can be zero) from the array, leaving the remaining elements in their original order.

**Example 1:**

**Input:** nums = [10,2,-10,5,20], k = 2

**Output:** 37

**Explanation:** The subsequence is [10, 2, 5, 20].

**Example 2:**

**Input:** nums = [-1,-2,-3], k = 1

**Output:** -1

**Explanation:** The subsequence must be non-empty, so we choose the largest number.

**Example 3:**

**Input:** nums = [10,-2,-10,-5,20], k = 2

**Output:** 23

**Explanation:** The subsequence is [10, -2, -5, 20].

**Constraints:**

* 1 <= k <= nums.length <= 105
* -104 <= nums[i] <= 104

### Analysis:

The target is to get maximum sub-sequence sum but keep the gap is less than k, so at each scanning position, we just care all previous answer which is within range K, if we can not get a positive one, we can start a sequence again. Keep the accumulated sum in decreasing order with end index, because if later you get a greater result later, you do not need previous one. You can also discard all negative accumulated sum if you want.

/// <summary>

/// Leet code #1425. Constrained Subset Sum

///

/// Hard

///

/// Given an integer array nums and an integer k, return the maximum sum

/// of a non-empty subset of that array such that for every two

/// consecutive integers in the subset, nums[i] and nums[j], where i < j,

/// the condition j - i <= k is satisfied.

///

/// A subset of an array is obtained by deleting some number of elements

/// (can be zero) from the array, leaving the remaining elements in their

/// original order.

/// Example 1:

///

/// Input: nums = [10,2,-10,5,20], k = 2

/// Output: 37

/// Explanation: The subset is [10, 2, 5, 20].

/// Example 2:

///

/// Input: nums = [-1,-2,-3], k = 1

/// Output: -1

/// Explanation: The subset must be non-empty, so we choose the largest

/// number.

///

/// Example 3:

///

/// Input: nums = [10,-2,-10,-5,20], k = 2

/// Output: 23

/// Explanation: The subset is [10, -2, -5, 20].

///

/// Constraints:

/// 1. 1 <= k <= nums.length <= 10^5

/// 2. -10^4 <= nums[i] <= 10^4

/// </summary>

int LeetCodeStack::constrainedSubsetSum(vector<int>& nums, int k)

{

deque<pair<int, int>> sum\_chain;

int result = INT\_MIN;

for (size\_t i = 0; i < nums.size(); i++)

{

int sum;

if (sum\_chain.empty())

{

sum = nums[i];

}

else

{

if ((int)i - sum\_chain.front().first > k) sum\_chain.pop\_front();

sum = max(nums[i], nums[i] + sum\_chain.front().second);

while (!sum\_chain.empty() && sum >= sum\_chain.back().second)

{

sum\_chain.pop\_back();

}

}

sum\_chain.push\_back(make\_pair(i, sum));

result = max(sum, result);

}

return result;

}

## 862. Shortest Subarray with Sum at Least K

Hard

Given an integer array nums and an integer k, return *the length of the shortest non-empty****subarray****of*nums*with a sum of at least*k. If there is no such **subarray**, return -1.

A **subarray** is a **contiguous** part of an array.

**Example 1:**

**Input:** nums = [1], k = 1

**Output:** 1

**Example 2:**

**Input:** nums = [1,2], k = 4

**Output:** -1

**Example 3:**

**Input:** nums = [2,-1,2], k = 3

**Output:** 3

**Constraints:**

* 1 <= nums.length <= 105
* -105 <= nums[i] <= 105
* 1 <= k <= 109

### Analysis:

The target is keep the accumulated sum increasing and process all subarray with difference of sum greater than or equal to k.

/// <summary>

/// Leet code #862. Shortest Subarray with Sum at Least K

///

/// Return the length of the shortest, non-empty, contiguous subarray of A

/// with sum at least K.

///

/// If there is no non-empty subarray with sum at least K, return -1.

///

/// Example 1:

/// Input: A = [1], K = 1

/// Output: 1

///

/// Example 2:

/// Input: A = [1,2], K = 4

/// Output: -1

///

/// Example 3:

/// Input: A = [2,-1,2], K = 3

/// Output: 3

///

/// Note:

/// 1. 1 <= A.length <= 50000

/// 2. -10 ^ 5 <= A[i] <= 10 ^ 5

/// 3. 1 <= K <= 10 ^ 9

/// </summary>

int LeetCodeStack::shortestSubarray(vector<int>& nums, int k)

{

int result = INT\_MAX;

deque<pair<long long, int>> window;

long long sum = 0;

window.push\_back(make\_pair(0, -1));

for (int i = 0; i < (int)nums.size(); i++)

{

if (nums[i] >= k) return 1;

sum += nums[i];

while (!window.empty() && sum <= window.back().first)

{

window.pop\_back();

}

while (!window.empty() && sum - window.front().first >= k)

{

result = min(result, i - window.front().second);

window.pop\_front();

}

window.push\_back(make\_pair(sum, i));

}

if (result == INT\_MAX) return -1;

return result;

}