1. **Implement stack using array**

Difficulty: **Basic**Accuracy: **54.76%**Submissions: **268K+**Points: **1**Average Time: **25m**

Write a program to implement a Stack using Array. Your task is to use the class as shown in the comments in the code editor and complete the functions push() and pop() to implement a stack. The push() method takes one argument, an integer **'x'** to be pushed into the stack and **pop()** which returns an integer present at the top and popped out from the stack. If the stack is empty then return **-1** from the pop() method.

Note: The input is given in form of queries. Since there are two operations push() and pop(), there is two types of queries as described below:
(i) 1 x (a query of this type means pushing 'x' into the stack)
(ii) 2 (a query of this type means to pop an element from the stack and print the popped element)
Input contains separated by space and as described above.

Examples :

Input: 1 2 1 3 2 1 4 2

Output: 3, 4

Explanation:

push(2) the stack will be {2}

push(3) the stack will be {2 3}

pop() poped element will be 3,

the stack will be {2}

push(4) the stack will be {2 4}

pop() poped element will be 4

Input: 2 1 4 1 5 2

Output: -1, 5

Expected Time Complexity: O(1)
Expected Space Complexity: O(1)

Constraints:
1 <= numbers of calls made to push, pop <= 100
1 <= x <= 100

2. Queue Using Array

3. Difficulty: **Basic**Accuracy: **47.24%**Submissions: **232K+**Points: **1**Average Time: **15m**
4. Implement a Queue using an Array. Queries in the Queue are of the following type:

   (i) 1 x (a query of this type means pushing **'x'** into the queue)

   (ii) 2 (a query of this type means to pop an element from the queue and print the popped element. If the queue is empty then return -1)
5. We just have to implement the functions **push** and **pop** and the driver code will handle the output.
6. **Examples:**
7. **Input:** Queries = 1 2 1 3 2 1 4 2
8. **Output:** 2 3
9. **Explanation:** For query 1 2 the queue will be {2} 1 3 the queue will be {2 3} 2 poped element will be 2 the queue will be {3} 1 4 the queue will be {3 4} 2 popped element will be 3
10. **Input:** Queries = 1 3 2 2 1 4
11. **Output:** 3 -1
12. **Explanation:** For query 1 3 the queue will be {3} 2 popped element will be 3 the queue will be empty 2 there is no element in the queue and hence -1 1 4 the queue will be {4}.
13. **Input:** Queries = 1 3 2 2 1 3
14. **Output:** 3 -1
15. **Explanation:** For query 1 3 the queue will be {3} 2 popped element will be 3 the queue will be empty 2 there is no element in the queue and hence -1 1 3 the queue will be {3} and hence -1 1 3 the queue will be {3}.
16. **Constraints:**

    $1 \leq$ number of query$\leq 10^5$

    $0 \leq x \leq 10^5$

3. **225. Implement Stack using Queues**

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (push, top, pop, and empty).
Implement the MyStack class:

- void push(int x) Pushes element x to the top of the stack.
- int pop() Removes the element on the top of the stack and returns it.
- int top() Returns the element on the top of the stack.
- boolean empty() Returns true if the stack is empty, false otherwise.

Notes:

- You must use **only** standard operations of a queue, which means that only push to back, peek/pop from front, size and is empty operations are valid.
- Depending on your language, the queue may not be supported natively. You may simulate a queue using a list or deque (double-ended queue) as long as you use only a queue's standard operations.

Example 1:
Input
["MyStack", "push", "push", "top", "pop", "empty"]
[[], [1], [2], [], [], []]
Output
[null, null, null, 2, 2, false]

Explanation
MyStack myStack = new MyStack();
myStack.push(1);
myStack.push(2);
myStack.top(); // return 2
myStack.pop(); // return 2
myStack.empty(); // return False

Constraints:

- 1 <= x <= 9
- At most 100 calls will be made to push, pop, top, and empty.
- All the calls to pop and top are valid

4. **232. Implement Queue using Stacks**

Easy

Topics

Companies

Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty).

Implement the MyQueue class:

- void push(int x) Pushes element x to the back of the queue.
- int pop() Removes the element from the front of the queue and returns it.
- int peek() Returns the element at the front of the queue.
- boolean empty() Returns true if the queue is empty, false otherwise.

Notes:

- You must use **only** standard operations of a stack, which means only push to top, peek/pop from top, size, and is empty operations are valid.
- Depending on your language, the stack may not be supported natively. You may simulate a stack using a list or deque (double-ended queue) as long as you use only a stack's standard operations.

Example 1:

Input

["MyQueue", "push", "push", "peek", "pop", "empty"]

[[], [1], [2], [], [], []]

Output

[null, null, null, 1, 1, false]

Explanation

MyQueue myQueue = new MyQueue();

myQueue.push(1); // queue is: [1]

myQueue.push(2); // queue is: [1, 2] (leftmost is front of the queue)

myQueue.peek(); // return 1

myQueue.pop(); // return 1, queue is [2]

myQueue.empty(); // return false

Constraints:

- 1 <= x <= 9
- At most 100 calls will be made to push, pop, peek, and empty.
- All the calls to pop and peek are valid.

5. **Stack using Linked List**

Difficulty: **Easy**Accuracy: **53.98%**Submissions: **177K+**Points: **2**Average Time: **40m**

Let's give it a try! You have a linked list and must implement the functionalities push and pop of stack using this given linked list. Your task is to use the class as shown in the comments in the code editor and complete the functions push() and pop() to implement a stack.
The push() method takes one argument, an integer **'x'** to be pushed into the stack and **pop()** which returns an integer present at the top and popped out from the stack. If the stack is empty then return **-1** from the pop() method.
Note: The input is given in the form of queries. Since there are two operations push() and pop(), there is two types of queries as described below:
(i) 1 (a query of this type takes **x** as another parameter and pushes it into the stack)
(ii) 2 (a query of this type means to pop an element from the stack and return the popped element)
Input is separated by space and as described above.

Examples :

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

Output: [3, 4]

Explanation:

push(2) : the stack will be {2}

push(3) : the stack will be {2 3}

pop()   : poped element will be 3,the stack will be {2}

push(4) : the stack will be {2 4}

pop()   : poped element will be 4

Input: [[2], [1,4], [1,5], [2]]

Output: [-1, 4]
Explanation:

pop()   : the stack is empty so its -1.

push(4) : the stack will be {4}

push(5) : the stack will be {4 5}

pop()   : poped element will be 5, the stack will be {4}

Expected Time Complexity: O(1)
Expected Auxillary Space: O(1)

Constraints:
1 <= numbers of calls made to push, pop <= 100
1 <= x <= 100

6. **Queue using Linked List**

Difficulty: **Basic**Accuracy: **45.6%**Submissions: **150K+**Points: **1**Average Time: **20m**

Implement a Queue using Linked List.
A Query **Q** is of 2 Types
(i) 1 x (a query of this type means pushing **'x'** into the queue)
(ii) 2 (a query of this type means to pop an element from the queue and print the poped element)

Examples:

Input: Q = 5, Queries = 1 2 1 3 2 1 4 2
Output: 2 3
Explanation:
[1,2] queue will be 2
[1,3] queue will be 2,3
[2] poped element will be 2 the queue will be 3
[1, 4] queue will be 3, 4
[2] poped element will be 3

Input: Q = 4, Queries = 1 2 2 2 1 3

Output: 2 -1

Explanation:

[1, 2] queue will be 2

[2] poped element will be 2 then

the queue will be empty.

[2] the queue is empty and hence -1

[1, 3] the queue will be 3

Constraints:
1 <= Q <= 100
1 <= x <= 100

Try more examples

7. **20. Valid Parentheses**

Easy

Topics

Companies

Hint

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

Example 1:

Input: s = "()"

Output: true

Example 2:

Input: s = "()[]{}"

Output: true

Example 3:

Input: s = "(]"

Output: false

Example 4:

Input: s = "([])"

Output: true

Constraints:

- 1 <= s.length <= 10^4
- s consists of parentheses only '()[]{}'.

8. **155. Min Stack**

Medium

Topics

Companies

Hint

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the MinStack class:

- MinStack() initializes the stack object.
- void push(int val) pushes the element val onto the stack.
- void pop() removes the element on the top of the stack.
- int top() gets the top element of the stack.
- int getMin() retrieves the minimum element in the stack.

You must implement a solution with O(1) time complexity for each function.

Example 1:

Input

["MinStack","push","push","push","getMin","pop","top","getMin"]

[[],[-2],[0],[-3],[],[],[],[]]

Output

[null,null,null,null,-3,null,0,-2]

Explanation

MinStack minStack = new MinStack();

minStack.push(-2);

minStack.push(0);

minStack.push(-3);

minStack.getMin(); // return -3

minStack.pop();

minStack.top(); // return 0

minStack.getMin(); // return -2

Constraints:

- $-2^{31} <= val <= 2^{31} - 1$
- Methods pop, top and getMin operations will always be called on **non-empty** stacks.
- At most $3 * 10^4$ calls will be made to push, pop, top, and getMin

1. **Infix to Postfix**

Difficulty: **Medium**Accuracy: **52.94%**Submissions: **121K+**Points: **4**

Given an infix expression in the form of string **s**. Convert this infix expression to a postfix expression.

- **Infix expression:** The expression of the form a **op** b. When an operator is in between every pair of operands.
- **Postfix expression:** The expression of the form a b **op**. When an operator is followed for every pair of operands.

Note: The order of precedence is: ^ greater than * equals to / greater than + equals to -. Ignore the right associativity of **^**.

Examples :

Input: s = "a+b*(c^d-e)^(f+g*h)-i"

Output: abcd^e-fgh*+^*+i-

Explanation: After converting the infix expression into postfix expression, the resultant expression will be abcd^e-fgh*+^*+i-

Input: s = "A*(B+C)/D"

Output: ABC+*D/

Explanation: After converting the infix expression into postfix expression, the resultant expression will be ABC+*D/

Input: s = "(a+b)*(c+d)"

Output: ab+cd+*

Constraints:
1 ≤ s.length ≤ 30

2. **Prefix to Infix Conversion**

Difficulty: **Medium**Accuracy: **69.51%**Submissions: **26K+**Points: **4**Average Time: **30m**

You are given a string **S** of size **N** that represents the prefix form of a valid mathematical expression. The string **S** contains only lowercase and uppercase alphabets as operands and the operators are +, -, *, /, %, and ^.Convert it to its infix form.

Example 1:

Input:

*-A/BC-/AKL

Output:

((A-(B/C))*((A/K)-L))

Explanation:

The above output is its valid infix form.

Your Task:

Your task is to complete the function string preToInfix(string pre_exp), which takes a prefix string as input and return its infix form.

Expected Time Complexity: O(N).

Expected Auxiliary Space: O(N).

Constraints:

$3<=|S|<=10^4$

3. **Prefix to Postfix Conversion**

Difficulty: **Medium**Accuracy: **75.66%**Submissions: **22K+**Points: **4**Average Time: **30m**

You are given a string that represents the prefix form of a valid mathematical expression. Convert it to its postfix form.

Example:

Input:

*-A/BC-/AKL

Output:

ABC/-AK/L-*

Explanation:

The above output is its valid postfix form.

Your Task:

Complete the function **preToPost(string pre_exp)**, which takes a prefix string as input and returns its postfix form.

Expected Time Complexity: O(N).

Expected Auxiliary Space: O(N).

Constraints:

3<=pre_exp.length()<=100

4. **Postfix to Infix Conversion**

Difficulty: **Medium**Accuracy: **49.41%**Submissions: **42K+**Points: **4**Average Time: **30m**

You are given a string that represents the postfix form of a valid mathematical expression. Convert it to its infix form.

Example:

Input:

ab*c+

Output:

((a*b)+c)

Explanation:

The above output is its valid infix form.

Your Task:

Complete the function string postToInfix(string post_exp), which takes a postfix string as input and returns its infix form.

Expected Time Complexity: O(N).

Expected Auxiliary Space: O(N).

Constraints:

$3<=post\_exp.length()<=10^4$

5. **Infix to Postfix**

Difficulty: **Medium**Accuracy: **52.94%**Submissions: **121K+**Points: **4**

Given an infix expression in the form of string **s**. Convert this infix expression to a postfix expression.

- **Infix expression:** The expression of the form a **op** b. When an operator is in between every pair of operands.
- **Postfix expression:** The expression of the form a b **op**. When an operator is followed for every pair of operands.

Note: The order of precedence is: ^ greater than * equals to / greater than + equals to -. Ignore the right associativity of **^**.

Examples :

Input: s = "a+b*(c^d-e)^(f+g*h)-i"

Output: abcd^e-fgh*+^*+i-

Explanation: After converting the infix expression into postfix expression, the resultant expression will be abcd^e-fgh*+^*+i-

Input: s = "A*(B+C)/D"

Output: ABC+*D/

Explanation: After converting the infix expression into postfix expression, the resultant expression will be ABC+*D/

Input: s = "(a+b)*(c+d)"

Output: ab+cd+*

Constraints:
1 ≤ s.length ≤ 30

Try more examples

1. **496. Next Greater Element I**

Easy

Topics

Companies

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] <= 10^4
- All integers in nums1 and nums2 are **unique**.
- All the integers of nums1 also appear in nums2.

2.

503. Next Greater Element II

Medium

Topics

Companies

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 <= 10^4
- -10^9 <= nums[i] <= 10^9

3. **Nearest Smaller Element**

ProgrammingStacks And Queues

easy

52.0% Success

665

22

Bookmark

Asked In:

Given an array, find the **nearest** smaller element G[i] for every element A[i] in the array such that the element has an **index smaller than i**.

More formally,

G[i] for an element A[i] = an element A[j] such that

j is maximum possible AND

j < i AND

A[j] < A[i]

Elements for which no smaller element exist, consider next smaller element as -1.

Input Format

The only argument given is integer array A.

Output Format

Return the integar array G such that G[i] contains nearest smaller number than A[i].If no such element occurs G[i] should be -1.

For Example

Input 1:

A = [4, 5, 2, 10, 8]

Output 1:

G = [-1, 4, -1, 2, 2]

Explaination 1:

index 1: No element less than 4 in left of 4, G[1] = -1

index 2: A[1] is only element less than A[2], G[2] = A[1]

index 3: No element less than 2 in left of 2, G[3] = -1

index 4: A[3] is nearest element which is less than A[4], G[4] = A[3]

index 4: A[3] is nearest element which is less than A[5], G[5] = A[3]

Input 2:

A = [3, 2, 1]

Output 2:

[-1, -1, -1]

Explaination 2:

index 1: No element less than 3 in left of 3, G[1] = -1

index 2: No element less than 2 in left of 2, G[2] = -1

index 3: No element less than 1 in left of 1, G[3] = -1

4. **Number of greater elements to the right**

Difficulty: **Medium**Accuracy: **56.74%**Submissions: **35K+**Points: **4**Average Time: **10m**

Given an array of **N** integers and **Q** queries of indices. For each query indices[i], determine the count of elements in arr that are **strictly greater** than arr[indices[i]] to its right (after the position indices[i]).

Examples :

Input: arr[] = [3, 4, 2, 7, 5, 8, 10, 6], queries = 2, indices[] = [0, 5]

Output: [6, 1]

Explanation: The next greater elements to the right of 3(index 0) are 4,7,5,8,10,6. The next greater elements to the right of 8(index 5) is only 10.

Input: arr[] = [1, 2, 3, 4, 1], queries = 2, indices[] = [0, 3]

Output: [3, 0]

Explanation: The count of numbers to the right of index 0 which are greater than arr[0] is 3 i.e. (2, 3, 4). Similarly, the count of numbers to the right of index 3 which are greater than arr[3] is 0, since there are no greater elements than 4 to the right of the array.

Constraints:

$1 <= N <= 10^4$

$1 <= arr[i] <= 10^5$

$1 <= queries <= 100$

$0 <= indices[i] <= N - 1$

5. **42. Trapping Rain Water**

Hard

Topics

Companies

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Example 1:

Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]

Output: 6

Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

Example 2:

Input: height = [4,2,0,3,2,5]

Output: 9

Constraints:

- n == height.length
- $1 <= n <= 2 * 10^4$
- $0 <= height[i] <= 10^5$

6. **907. Sum of Subarray Minimums**

Medium

Topics

Companies

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** $10^9 + 7$.

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 <= $3 * 10^4$
- 1 <= arr[i] <= $3 * 10^4$

7. **735. Asteroid Collision**

Medium

Topics

Companies

Hint

We are given an array asteroids of integers representing asteroids in a row. The indices of the asteriod in the array represent their relative position in space.

For each asteroid, the absolute value represents its size, and the sign represents its direction (positive meaning right, negative meaning left). Each asteroid moves at the same speed.

Find out the state of the asteroids after all collisions. If two asteroids meet, the smaller one will explode. If both are the same size, both will explode. Two asteroids moving in the same direction will never meet.

Example 1:

Input: asteroids = [5,10,-5]

Output: [5,10]

Explanation: The 10 and -5 collide resulting in 10. The 5 and 10 never collide.

Example 2:

Input: asteroids = [8,-8]

Output: []

Explanation: The 8 and -8 collide exploding each other.

Example 3:

Input: asteroids = [10,2,-5]

Output: [10]

Explanation: The 2 and -5 collide resulting in -5. The 10 and -5 collide resulting in 10.

Constraints:

- 2 <= asteroids.length <= 10^4
- -1000 <= asteroids[i] <= 1000
- asteroids[i] != 0

8. **2104. Sum of Subarray Ranges**

Medium

Topics

Companies

Hint

You are given an integer array nums. The **range** of a subarray of nums is the difference between the largest and smallest element in the subarray.

Return *the **sum of all** subarray ranges of* nums*.*

A subarray is a contiguous **non-empty** sequence of elements within an array.

Example 1:

Input: nums = [1,2,3]

Output: 4

Explanation: The 6 subarrays of nums are the following:

[1], range = largest - smallest = 1 - 1 = 0

[2], range = 2 - 2 = 0

[3], range = 3 - 3 = 0

[1,2], range = 2 - 1 = 1

[2,3], range = 3 - 2 = 1

[1,2,3], range = 3 - 1 = 2

So the sum of all ranges is 0 + 0 + 0 + 1 + 1 + 2 = 4.

Example 2:

Input: nums = [1,3,3]

Output: 4

Explanation: The 6 subarrays of nums are the following:

[1], range = largest - smallest = 1 - 1 = 0

[3], range = 3 - 3 = 0

[3], range = 3 - 3 = 0

[1,3], range = 3 - 1 = 2

[3,3], range = 3 - 3 = 0

[1,3,3], range = 3 - 1 = 2

So the sum of all ranges is 0 + 0 + 0 + 2 + 0 + 2 = 4.

Example 3:

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

Output: 59

Explanation: The sum of all subarray ranges of nums is 59.

Constraints:

- 1 <= nums.length <= 1000
- -10^9 <= nums[i] <= 10^9

9. **402. Remove K Digits**

Medium

Topics

Companies

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:

- $1 <= k <= num.length <= 10^5$
- num consists of only digits.
- num does not have any leading zeros except for the zero itself.

10.

84. Largest Rectangle in Histogram

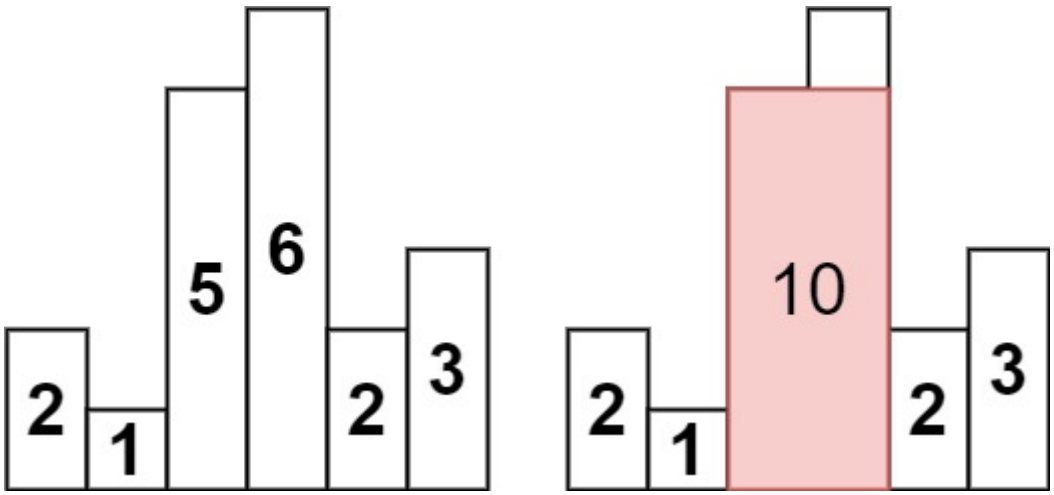
Hard

Topics

Companies

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:



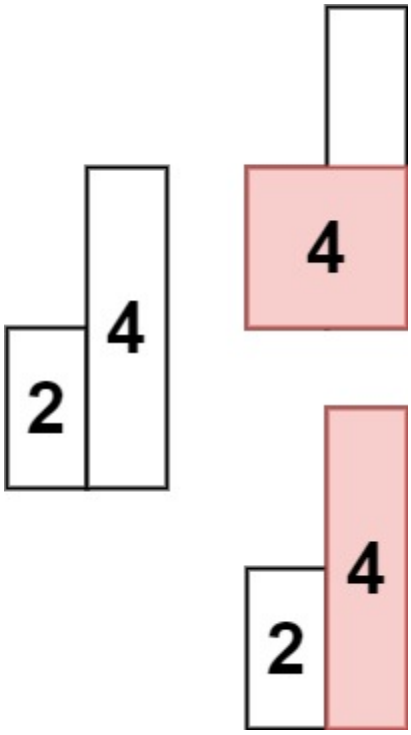
Input: heights = [2,1,5,6,2,3]

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 <= heights.length <= 10^5
- 0 <= heights[i] <= 10^4

11. [85. Maximal Rectangle]

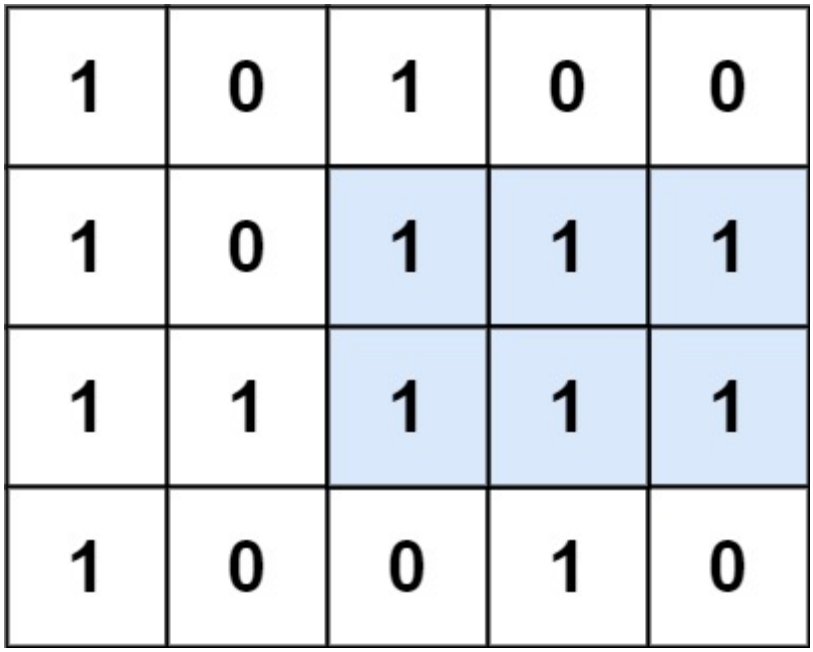
Hard

Topics

Companies

Given a rows x cols binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return *its area*.

Example 1:



Input: matrix =

[["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]]

Output: 6

Explanation: The maximal rectangle is shown in the above picture.

Example 2:

Input: matrix = [["0"]]

Output: 0

Example 3:

Input: matrix = [["1"]]

Output: 1

Constraints:

- rows == matrix.length
- cols == matrix[i].length
- 1 <= row, cols <= 200
- matrix[i][j] is '0' or '1'.

1. **239. Sliding Window Maximum**

Hard

Topics

Companies

Hint

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,-3,5,3,6,7], k = 3

Output: [3,3,5,5,6,7]

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]

Constraints:

- $1 <= nums.length <= 10^5$
- $-10^4 <= nums[i] <= 10^4$
- $1 <= k <= nums.length$

2. **Online Stock Span**

Medium

Topics

Companies

Design an algorithm that collects daily price quotes for some stock and returns **the span** of that stock's price for the current day.

The **span** of the stock's price in one day is the maximum number of consecutive days (starting from that day and going backward) for which the stock price was less than or equal to the price of that day.

- For example, if the prices of the stock in the last four days is [7,2,1,2] and the price of the stock today is 2, then the span of today is 4 because starting from today, the price of the stock was less than or equal 2 for 4 consecutive days.
- Also, if the prices of the stock in the last four days is [7,34,1,2] and the price of the stock today is 8, then the span of today is 3 because starting from today, the price of the stock was less than or equal 8 for 3 consecutive days.

Implement the StockSpanner class:

- StockSpanner() Initializes the object of the class.
- int next(int price) Returns the **span** of the stock's price given that today's price is price.

Example 1:

Input

["StockSpanner", "next", "next", "next", "next", "next", "next", "next"]

[[], [100], [80], [60], [70], [60], [75], [85]]

Output

[null, 1, 1, 1, 2, 1, 4, 6]

Explanation

StockSpanner stockSpanner = new StockSpanner();

stockSpanner.next(100); // return 1

stockSpanner.next(80); // return 1

stockSpanner.next(60); // return 1

stockSpanner.next(70); // return 2

stockSpanner.next(60); // return 1

stockSpanner.next(75); // return 4, because the last 4 prices (including today's price of 75) were less than or equal to today's price.

stockSpanner.next(85); // return 6

Constraints:

- 1 <= price <= 10^5
- At most 10^4 calls will be made to next

3. **The Celebrity Problem**

Difficulty: **Medium**Accuracy: **38.33%**Submissions: **306K+**Points: **4**Average Time: **30m**

A celebrity is a person who is known to all but **does not know** anyone at a party. A party is being organized by some people. A square matrix **mat[][]** (n*n) is used to represent people at the party such that if an element of **row i and column j is set to 1** it means **ith person knows jth person**. You need to return the **index of the celebrity** in the party, if the celebrity does not exist, return **-1**.

Note: Follow **0-based** indexing.

Examples:

Input: mat[][] = [[1, 1, 0], [0, 1, 0], [0, 1, 1]]

Output: 1

Explanation: 0th and 2nd person both know 1st person. Therefore, 1 is the celebrity person.

Input: mat[][] = [[1, 1], [1, 1]]

Output: -1

Explanation: Since both the people at the party know each other. Hence none of them is a celebrity person.

Input: mat[][] = [[1]]

Output: 0

Constraints:
1 <= mat.size()<= 1000
0 <= mat[i][j]<= 1
mat[i][i] == 1

4. **146. LRU Cache**

Medium

Topics

Companies

Design a data structure that follows the constraints of a **Least Recently Used (LRU) cache**.

Implement the LRUCache class:

- LRUCache(int capacity) Initialize the LRU cache with **positive** size capacity.
- int get(int key) Return the value of the key if the key exists, otherwise return -1.
- void put(int key, int value) Update the value of the key if the key exists. Otherwise, add the key-value pair to the cache. If the number of keys exceeds the capacity from this operation, **evict** the least recently used key.

The functions get and put must each run in O(1) average time complexity.

Example 1:

Input

["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get", "get"]

[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]

Output

[null, null, null, 1, null, -1, null, -1, 3, 4]

Explanation

LRUCache lRUCache = new LRUCache(2);

lRUCache.put(1, 1); // cache is {1=1}

lRUCache.put(2, 2); // cache is {1=1, 2=2}

lRUCache.get(1);   // return 1

lRUCache.put(3, 3); // LRU key was 2, evicts key 2, cache is {1=1, 3=3}

lRUCache.get(2);   // returns -1 (not found)

lRUCache.put(4, 4); // LRU key was 1, evicts key 1, cache is {4=4, 3=3}

lRUCache.get(1);   // return -1 (not found)

lRUCache.get(3);   // return 3

lRUCache.get(4);   // return 4

Constraints:

- 1 <= capacity <= 3000
- 0 <= key <= 10^4

- $0 <= value <= 10^5$
- At most $2 * 10^5$ calls will be made to get and put.

5. **460. LFU Cache**

Hard

Topics

Companies

Design and implement a data structure for a Least Frequently Used (LFU) cache.

Implement the LFUCache class:

- LFUCache(int capacity) Initializes the object with the capacity of the data structure.
- int get(int key) Gets the value of the key if the key exists in the cache. Otherwise, returns -1.
- void put(int key, int value) Update the value of the key if present, or inserts the key if not already present. When the cache reaches its capacity, it should invalidate and remove the **least frequently used** key before inserting a new item. For this problem, when there is a **tie** (i.e., two or more keys with the same frequency), the **least recently used** key would be invalidated.

To determine the least frequently used key, a **use counter** is maintained for each key in the cache. The key with the smallest **use counter** is the least frequently used key.

When a key is first inserted into the cache, its **use counter** is set to 1 (due to the put operation). The **use counter** for a key in the cache is incremented either a get or put operation is called on it.

The functions get and put must each run in O(1) average time complexity.

Example 1:

Input

["LFUCache", "put", "put", "get", "put", "get", "get", "put", "get", "get", "get"]

[[2], [1, 1], [2, 2], [1], [3, 3], [2], [3], [4, 4], [1], [3], [4]]

Output

[null, null, null, 1, null, -1, 3, null, -1, 3, 4]

Explanation

// cnt(x) = the use counter for key x

// cache=[] will show the last used order for tiebreakers (leftmost element is most recent)

LFUCache lfu = new LFUCache(2);

lfu.put(1, 1); // cache=[1,_], cnt(1)=1

lfu.put(2, 2); // cache=[2,1], cnt(2)=1, cnt(1)=1

lfu.get(1); // return 1

// cache=[1,2], cnt(2)=1, cnt(1)=2

lfu.put(3, 3); // 2 is the LFU key because cnt(2)=1 is the smallest, invalidate 2.

// cache=[3,1], cnt(3)=1, cnt(1)=2

lfu.get(2); // return -1 (not found)

lfu.get(3); // return 3

// cache=[3,1], cnt(3)=2, cnt(1)=2

lfu.put(4, 4); // Both 1 and 3 have the same cnt, but 1 is LRU, invalidate 1.

// cache=[4,3], cnt(4)=1, cnt(3)=2

lfu.get(1); // return -1 (not found)

lfu.get(3); // return 3

// cache=[3,4], cnt(4)=1, cnt(3)=3

lfu.get(4); // return 4

// cache=[4,3], cnt(4)=2, cnt(3)=3

Constraints:

- 1 <= capacity <= 10^4
- 0 <= key <= 10^5
- 0 <= value <= 10^9
- At most 2 * 10^5 calls will be made to get and put