**Easy**

**232. Implement Queue using Stacks**

**225. Implement Stack using Queues**

**496. Next Greater Element I**

**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.

**150. Evaluate Reverse Polish Notation**

**Input:** tokens = ["2","1","+","3","\*"]

**Output:** 9

**Explanation:** ((2 + 1) \* 3) = 9

**20. Valid Parentheses**

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

**Output:** true

**Medium**

**735. Asteroid Collision**

**Input:** asteroids = [1, 5, 3, -4, 6, -7, 8]

**Output:** [-7, 8]

**Explanation:** first 3 and 4 will collide and 3 will explode. Then 5 and 4 will collide then 4 will explode.

Then 6 and 7 will collide and 6 will will explode. Then 5 and 7 will collide and 5 will explode. Then 1 and 7 will collide and will explode.

So in stack we will have only -7, 8

**GFG. Celebrity Problem**

**Input:**

MATRIX = { {0, 0, 1, 0},

{0, 0, 1, 0},

{0, 0, 0, 0},

{0, 0, 1, 0} }

**Output:**id = 2

**Explanation:** The person with ID 2 does not know anyone but everyone knows him

**146. LRU Cache**

**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

**155. Min Stack**

**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

**503. Next Greater Element II**

**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.

**901. Online Stock Span**

**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

**402. Remove K Digits**

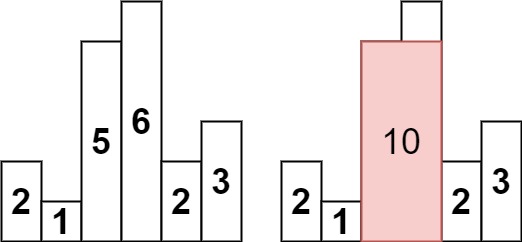
**Input:** num = "1432219", k = 3

**Output:** "1219"

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

**Hard**

**84. Largest Rectangle in Histogram**



**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.

**460. LFU Cache**

**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)

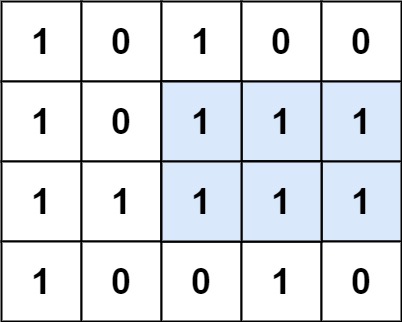
lfu.get(3); // return 3

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

lfu.get(4); // return 4

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

**85. Maximal Rectangle**



**Input:** matrix = [["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]]

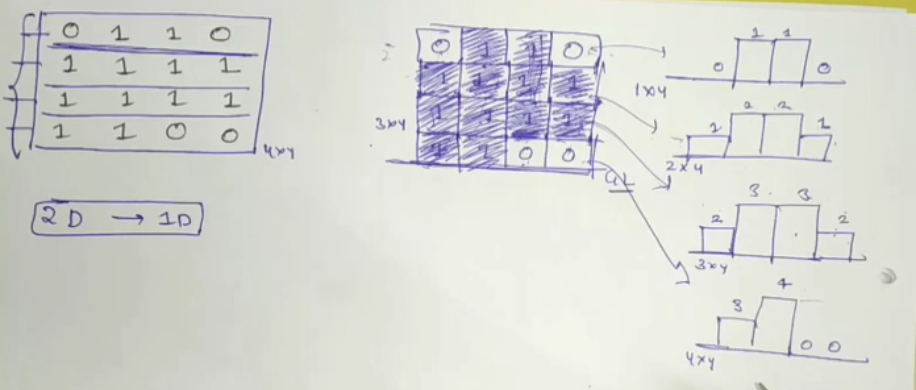
**Output:** 6

**char**[][] matrix = { { '0', '1', '1', '0' },

{ '1', '1', '1', '1' },

{ '1', '1', '1', '1' },

{ '1', '1', '0', '0' } };



So now this binary matrix is converted into 4 histograms. Max area of binary matrix will be the max area from all these 4 histograms.

For this first we find out the histogram for the 1st row.

**int** result = maxHistogram(matrix[0]);

Then we iterate from the 2nd row and add the element if element == 1 and after adding the elements we need to find out MAH on this 2nd row elements…and so on

**for** (**int** i = 1; i < row; i++) {

**for** (**int** j = 0; j < column; j++) {

**if** (matrix[i][j] == 1)

matrix[i][j] += matrix[i - 1][j];

}

result = Math.*max*(result, maxHistogram(matrix[i]));

}

**42. Trapping Rain Water**

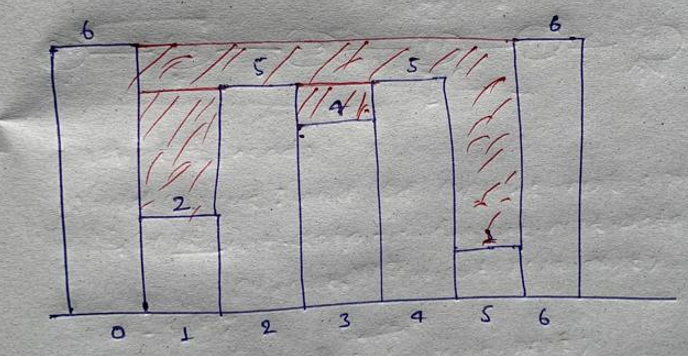


**Input:** height = [0,1,0,2,1,0,1,3,2,1,2,1]

**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.

**int**[] arr = { 6, 2, 5, 4, 5, 1, 6 };



Here we need to find out the total amount of water stored in between buildings.

Here we need to find out max value in left and right for each building.

Once we find max from left and right we need to find min from this left and right because water amount will depend on min building height only. and amount of water for each building will be total amount of water on that building – building height

Eg:

For building 2, max value in left = 6 and max value in right = 6

So min of(6, 6) is 6

So height of water is 6 – 2(height of building)

And in last sum up all the amount of water and return the result.