

1. Implement Queue using Stacks

Code:

```
import java.util.Stack;

class CustomQueue {
    private Stack<Integer> inputStack;
    private Stack<Integer> outputStack;

    public CustomQueue() {
        inputStack = new Stack<>();
        outputStack = new Stack<>();
    }

    private void shiftStacks() {
        while (!inputStack.isEmpty()) {
            outputStack.push(inputStack.pop());
        }
    }

    public void push(int x) {
        inputStack.push(x);
    }

    public int pop() {
        if (outputStack.isEmpty()) {
            shiftStacks();
        }
        return outputStack.pop();
    }

    public int peek() {
        if (outputStack.isEmpty()) {
            shiftStacks();
        }
        return outputStack.peek();
    }

    public boolean empty() {
        return inputStack.isEmpty() && outputStack.isEmpty();
    }
}
```

Submissions
Description
Editorial
Solutions

232. Implement Queue using Stacks

Solved

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:

Code Accepted

← All Submissions

Accepted 22 / 22 testcases passed

Abhinav091 submitted at Feb 16, 2025 15:26

Editorial Solution

Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

41.29 MB | Beats 83.23%

2. Implement Min Stack using Two Stacks

Code:

```
class MinStack {
    Stack<Integer> stack;
    Stack<Integer> MinStack;
    public MinStack() {
        stack=new Stack<>();
        MinStack=new Stack<>();
    }

    public void push(int val) {
        stack.push(val);
        if(MinStack.isEmpty() || val<=MinStack.peek()){
            MinStack.push(val);
        }
    }

    public void pop() {
        if(stack.isEmpty()){
            return;
        }
        int top= stack.pop();
        if(!MinStack.isEmpty()&&top==MinStack.peek()){
            MinStack.pop();
        }
    }

    public int top() {
```

```

        return stack.peek();
    }

    public int getMin() {
        return MinStack.peek();
    }
}

```

155. Min Stack

Solved

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.

← All Submissions

Accepted 31 / 31 testcases passed

Abhinav091 submitted at Feb 17, 2025 10:47

Editorial Solve

Runtime

5 ms | Beats 37.85%

Analyze Complexity

Memory

44.80 MB | Beats 67.74%

3. Implement Stack using Queue

Code:

```

class MyStack {
    private Queue<Integer> queue;

    public MyStack() {
        queue = new LinkedList<>();
    }

    public void push(int x) {
        queue.add(x);
        int size = queue.size();
        while (size-- > 1) {
            queue.add(queue.remove());
        }
    }

    public int pop() {
        return queue.remove();
    }
}

```

```

public int top() {
    return queue.peek();
}

public boolean empty() {
    return queue.isEmpty();
}
}

```

The screenshot shows the LeetCode interface for problem 225, "Implement Stack using Queues". The problem is marked as "Solved". The description states: "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, pop, peek/pop from front, size and is empty operations are valid." The submission details show it was "Accepted" with 18/18 testcases passed by user "Abhinav091" on Feb 16, 2025 at 15:30. Performance metrics are: Runtime 0 ms (Beats 100.00%) and Memory 41.24 MB (Beats 80.77%). A bar chart shows the runtime performance across different test cases, with the first case being the most time-consuming.

4. Implement stack using array

Code:

```

class Solution {
    public List<String> buildArray(int[] target, int n) {
        int[] array = new int[n];
        List<String> res = new ArrayList<>();
        int i = 0, c = 1;

        while (i < target.length) {
            array[c - 1] = c;
            res.add("Push");

            if (array[c - 1] == target[i]) {
                i++;
            } else {
                res.add("Pop");
            }
        }
    }
}

```

```

        C++;
    }
    return res;
}
}

```

1441. Build an Array With Stack Operations Solved

Medium Topics Companies Hint

You are given an integer array `target` and an integer `n`.

You have an empty stack with the two following operations:

- "Push": pushes an integer to the top of the stack.
- "Pop": removes the integer on the top of the stack.

You also have a stream of the integers in the range `[1, n]`.

Use the two stack operations to make the numbers in the stack (from the bottom to the top) equal to `target`. You should follow the following rules:

- If the stream of the integers is not empty, pick the next integer from the stream

Accepted 49 / 49 testcases passed
Abhinav091 submitted at Mar 19, 2025 15:25

Editorial Solution

Runtime: 0 ms | Beats 100.00%
Memory: 42.34 MB | Beats 98.82%

Analyze Complexity

5. Implement LRU Cache using Hash Table + Doubly Linked List

Code:

```

class Node {
    int key;
    int val;
    Node prev;
    Node next;

    public Node(int key, int val) {
        this.key = key;
        this.val = val;
        this.prev = null;
        this.next = null;
    }
}

class LRUCache {

    private int cap;
    private Map<Integer, Node> cache;
    private Node oldest;
    private Node latest;

    public LRUCache(int capacity) {

```

```

    this.cap = capacity;
    this.cache = new HashMap<>();
    this.oldest = new Node(0, 0);
    this.latest = new Node(0, 0);
    this.oldest.next = this.latest;
    this.latest.prev = this.oldest;
}

public int get(int key) {
    if (cache.containsKey(key)) {
        Node node = cache.get(key);
        remove(node);
        insert(node);
        return node.val;
    }
    return -1;
}

private void remove(Node node) {
    Node prev = node.prev;
    Node next = node.next;
    prev.next = next;
    next.prev = prev;
}

private void insert(Node node) {
    Node prev = latest.prev;
    Node next = latest;
    prev.next = next.prev = node;
    node.next = next;
    node.prev = prev;
}

public void put(int key, int value) {
    if (cache.containsKey(key)) {
        remove(cache.get(key));
    }
    Node newNode = new Node(key, value);
    cache.put(key, newNode);
    insert(newNode);
}

```

```

        if (cache.size() > cap) {
            Node lru = oldest.next;
            remove(lru);
            cache.remove(lru.key);
        }
    }
}

```

Submissions
Description
Editorial
Solutions

← All Solutions

- Time complexity: $O(1)$
- Space complexity: $O(capacity)$

Python
JavaScript
Java
C++

```

class Node {
    int key;
    int val;
    Node prev;
    Node next;

    public Node(int key, int val) {
        this.key = key;
        this.val = val;
        this.prev = null;
    }
}

```

Accepted
23 / 23 testcases passed

Abhinav091 submitted at Mar 19, 2025 15:45

Editorial
Solution

Runtime
48 ms | Beats 47.56%

Memory
115.81 MB | Beats 39.26%

Analyze Complexity