Implementation of MINSTACK (Minimum Stack)

Mr. Akash Yadav
Assistant Professor
Artificial Intelligence & Data Science

Lesson Plan

Subject/Course	**Competitive Coding**
Lesson Title	**Implementation of MINSTACK (Minimum Stack)**

Lesson Objectives
To implement a stack supporting push, pop, top, display, and getMin operations.
To handle overflow and underflow conditions.
To construct a stack of N capacity.
To efficiently retrieve the minimum element in O(1) time using an auxiliary stack.

Problem Statement:

Write a program for implementing a MINSTACK which should support operations like push, pop, overflow, underflow, display.

1. Construct a stack of N-capacity
2. Push elements
3. Pop elements
4. Top element
5. Retrieve the min element from the stack

Concept

A Stack follows the LIFO (Last In, First Out) principle.

Common operations:

- Push: Insert element at top
- Pop: Remove element from top
- Peek / Top: View top element

Why MINSTACK?

- A MinStack is an enhanced stack that also allows quick retrieval of the minimum element at any time.
- Naively, finding the minimum requires O(n) scanning.
- Using an auxiliary stack, we can track the minimum in O(1) time.

Algorithm/Logic

1. Initialize:

Create two stacks – mainStack and minStack.

2. Push(x):

Push x onto mainStack.
If minStack is empty or x <= minStack.top(), push x onto minStack.

3. Pop():

If mainStack is empty → Underflow.
Pop element from mainStack.
If popped element equals minStack.top(), also pop from minStack.

Algorithm/Logic

4. Top():

Return the top of mainStack if not empty.

5. GetMin():

Return top of minStack (it stores the minimum so far).

6. Display():

Print all elements of mainStack from top to bottom.

Visualization

Let's push → 5, 3, 7, 2

Step	Operation	mainStack	minStack	Min
1	push(5)	[5]	[5]	5
2	push(2)	[5,2]	[5,2]	2
3	push(8)	[5,2,8]	[5,2]	2
4	pop()	[5,2]	[5,2]	2
5	getMin()	[5,2]	[5,2]	✅ 2

Code Implementation

```
1  import java.util.Stack;
2
3  public class Practical1_MinStack {
4
5      static class MinStack {
6          Stack<Integer> mainStack = new Stack<>();
7          Stack<Integer> minStack = new Stack<>();
8
9          public void push(int x) {
10             mainStack.push(x);
11             if (minStack.isEmpty() || x <= minStack.peek())
12                 minStack.push(x);
13         }
14
15         public void pop() {
16             if (!mainStack.isEmpty()) {
17                 int val = mainStack.pop();
18                 if (val == minStack.peek())
19                     minStack.pop();
20             }
21         }
22
```

```
23          public int top() {
24              return mainStack.isEmpty() ? -1 : mainStack.peek();
25          }
26
27          public int getMin() {
28              return minStack.isEmpty() ? -1 : minStack.peek();
29          }
30      }
31
32      public static void main(String[] args) {
33          MinStack s = new MinStack();
34          s.push(5);
35          s.push(3);
36          s.push(7);
37          System.out.println("Current Min: " + s.getMin());
38          s.pop();
39          System.out.println("Top: " + s.top());
40          System.out.println("Current Min: " + s.getMin());
41      }
42  }
```

Output :

```
Current Min: 3
Top: 3
Current Min: 3
```

Time & Space Complexity

Operation	Time Complexity	Space Complexity	Why?
Push/Pop/Top/getMin	O(1)	O(N)	Constant access; Fixed arrays.
Display	O(N)	-	Linear print.

Summary

- MinStack is an improved version of a normal stack that supports push(), pop(), top(), and getMin() operations — all in O(1) time.
- It uses two stacks:
    a. mainStack stores all elements.
    b. minStack tracks the minimum value at each level.
- On push(), if the new element ≤ current minimum, it's also pushed into minStack.
- On pop(), if the popped element equals the top of minStack, both are popped. getMin() simply returns the top of minStack, giving the minimum instantly.

Summary

- This approach keeps all operations fast and efficient (O(1) time, O(n) space).

- **Advantage**: No need to traverse the stack to find the minimum.
- **Use Case**: Ideal for real-time applications where frequent min lookups are required.

Practice Questions:

1. Min Stack — LeetCode #155

🔗 https://leetcode.com/problems/min-stack/

Concept: Implement a stack that supports push(), pop(), top(), and getMin() in O(1).

Why Practice: Same as our class example — perfect for reinforcement.

Practice Questions:

2. Implement Queue using Stacks — LeetCode #232

🔗 https://leetcode.com/problems/implement-queue-using-stacks/

Concept: Implement a queue using one or two stacks.

Why Practice: Strengthens your stack understanding by reversing the logic.

Thanks