MinStack Class Documentation

Overview

The MinStack class is designed to function as a stack data structure that, in addition to standard stack operations, efficiently supports retrieving the minimum element in constant time. The class allows you to perform the following operations: push, pop, top, and getMin, all with O(1) time complexity.

Class Initialization

__init__()

- ***Description:*** Initializes the MinStack object by setting up two internal stacks:
  - The main stack (stack) that stores all elements pushed onto the stack.
  - A secondary stack (min_stack) that keeps track of the minimum element at each level of the stack.
- ***Parameters:*** None.
- ***Returns:*** None.

Methods

1. **push(val: int) -> None**

- ***Description:*** Pushes an element onto the stack and updates the minimum value stack if necessary.
- ***Parameters:***
  - *val (int):* The value to be pushed onto the stack.
- ***Behavior:***
  - The value val is added to the main stack.
  - If the min_stack is empty or if val is less than or equal to the current minimum value (which is the top of the min_stack), val is also pushed onto the min_stack.
- ***Returns:*** None.

2. **pop() -> None**

- ***Description:*** Removes the top element from the stack. If the removed element is the current minimum, it also removes it from the min_stack.
- ***Parameters:*** None.
- ***Behavior:***
  - The top element is removed from the main stack.
  - If this element is the same as the top element of the min_stack, it is also removed from the min_stack, thereby updating the current minimum.
- ***Returns:*** None.

3. **top() -> int**

- ***Description:*** Retrieves the top element of the stack without removing it.
- ***Parameters:*** None.
- ***Behavior:*** Returns the value at the top of the main stack.
- ***Returns:***
  - *int:* The top element of the stack.

4. **getMin() -> int**
    - ***Description:*** Retrieves the minimum element in the stack in constant time.
    - ***Parameters:*** None.
    - ***Behavior:*** Returns the value at the top of the min_stack, which represents the minimum value of the stack.
    - ***Returns:***
        - int: The minimum element in the stack.

Example Usage

- **Initialization:**
    - Create a MinStack object by calling MinStack().
- **Pushing Values:**
    - Use push(val) to add elements to the stack.
- **Retrieving the Top Element:**
    - Call top() to get the topmost element of the stack.
- **Popping Values:**
    - Use pop() to remove the top element from the stack.
- **Getting the Minimum Element:**
    - Call getMin() to retrieve the current minimum element in the stack.

Performance and Complexity

- **Time Complexity:**
    - All operations (push, pop, top, and getMin) are guaranteed to run in O(1) time.

- **Space Complexity:**
  - The space complexity is O(n) in the worst case, where n is the number of elements in the stack. This occurs when every element pushed is smaller than the previous one, leading to min_stack having the same size as stack.

Constraints

- **Value Range:**
  - The value val that is pushed onto the stack can range between -2^{31} and 2^{31} 1.
- **Method Calls:**
  - Methods pop, top, and getMin will only be called on non-empty stacks.
  - The stack can handle up to 30,000 operations efficiently, as each operation is performed in constant time.

Summary

The MinStack class provides a robust and efficient way to manage stack operations with the added capability of retrieving the minimum element in constant time. By leveraging a secondary stack (min_stack), it ensures that the minimum value is always accessible without the need for additional computation, maintaining optimal performance for large sequences of operations.