# STACK UNDO/REDO CASE STUDY

Submitted by:

AARUSH C S

ATHUL K KOSHY

MOHAMMED RIZWAN PP

MOIDEEN NIHAL

Course: BCA (Hons) AI

Subject: Data Structures and Algorithms

# Case Study Report: Stack-Based Undo/Redo Feature in Text Editors

## 1. Introduction

In modern text editors, the ability to undo and redo actions plays a crucial role in improving user experience. The undo/redo feature allows users to revert or restore previous changes efficiently, ensuring better control over text modifications.

This case study explores the implementation of an Undo/Redo feature using the stack data structure, enabling users to manage text edits effectively.

## 2. Problem Statement

Text editors require an efficient mechanism to track user actions and allow them to undo or redo changes. Traditional methods may lead to inefficiencies in handling multiple operations, increasing memory usage. The challenge is to implement an optimized undo/redo functionality using stacks for quick access to previous states.

## 3. Objectives

- Implement an undo/redo system using two stacks to manage text changes.

- Provide an intuitive interface for users to perform undo and redo actions efficiently.

- Optimize memory usage by removing outdated undo/redo actions when necessary.

## 4. Literature Review

The stack data structure follows Last In, First Out (LIFO) principle, making it ideal for handling undo and redo operations. Previous studies suggest that using two stacks (undoStack & redoStack) improves efficiency. Modern text editors such as Microsoft Word and Google Docs implement stack-based approaches for undo/redo functionalities.

## 5. Methodology

Data Structure Used: Two stacks (undoStack and redoStack)

### Algorithm:

1. When the user types a new text, push the change onto undoStack.

2. When the user clicks "Undo":

   - Pop the last action from undoStack and push it onto redoStack.

3. When the user clicks "Redo":

   - Pop the last action from redoStack and push it back onto undoStack.

4. If a new change is made after using "Undo", clear redoStack to maintain consistency

## 6. Implementation

Example Scenario:

- User types: 'Hello' → Undo Stack: [Hello], Redo Stack: []
- User types: 'World' → Undo Stack: [Hello, World], Redo Stack: []
- User clicks 'Undo' → Moves 'World' to the Redo Stack
- User clicks 'Redo' → Moves 'World' back to the Undo Stack

## 7. Results and Analysis

The implementation successfully enables smooth text editing with undo/redo functionality.

Stack-based operations ensure O(1) time complexity for undo and redo actions.

## 8. Challenges and Solutions

- Challenge: Clearing the redo stack when a new edit is made after an undo.

Solution: Implement logic to clear redoStack after a new action.

## 9. Conclusion

The stack-based undo/redo system is an efficient way to manage text modifications in editors.

## 10. References

"Data Structures and Algorithms in Python" – Michael T. Goodrich

Google Docs Developer Documentation on Undo/Redo Management

CODE:-

```
class TextEditor:
    def __init__(self):
        self.text = ""
        self.undo_stack = []
        self.redo_stack = []

    def type(self, new_text):
        # Save current state for undo
        self.undo_stack.append(self.text)
        # Clear redo stack on new action
        self.redo_stack.clear()
        # Add new text
        self.text += new_text

    def undo(self):
        if not self.undo_stack:
            print("Nothing to undo.")
            return
        # Save current state to redo stack
        self.redo_stack.append(self.text)
        # Revert to last state
        self.text = self.undo_stack.pop()

    def redo(self):
        if not self.redo_stack:
```

```
            print("Nothing to redo.")
            return
        # Save current state for undo
        self.undo_stack.append(self.text)
        # Redo the last undone state
        self.text = self.redo_stack.pop()

    def show(self):
        print(f"Current Text: '{self.text}'")

# Example usage:
editor = TextEditor()
editor.type("Hello ")
editor.type("World")
editor.show()  # Output: 'Hello World'

editor.undo()
editor.show()  # Output: 'Hello '

editor.redo()
editor.show()  # Output: 'Hello World'

editor.undo()
editor.type("Everyone")
editor.show()  # Output: 'Hello Everyone'
```