CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY
An Autonomous Institute I Affiliated to Osmania University
Kokapet Village, Gandipet Mandal, Hyderabad, Telangana-500075, www.cbit.ac.in

COMMITTED TO RESEARCH, INNOVATION AND EDUCATION 46 years

A

PROJECT REPORT

ON

TEXT EDITOR WITH ADVANCED FEATURES

RollNo of Students:

160123737112
160123737121
160123737130

Institution and Department:
IT Department, CBIT

Guide/Supervisor's Name:
Ms. P. Kiranmaie Mam (Assistant Professor)

Submission Date:
16 NOV 2024

Table of Contents

Abstract :

This project involves the development of a text editor that provides a range of essential features, including text insertion, deletion, undo/redo functionality, text import from local files and URLs, find and replace functionality, and a word count feature. The primary objective of this text editor is to showcase how fundamental data structures such as lists and stacks can be used to efficiently handle user input and edit operations. The project integrates file-handling features, allowing users to import content from local files and external web URLs using Python libraries like requests and BeautifulSoup. The text editor includes features such as undo/redo functionality using stacks, find and replace using simple string operations, and word count for document analysis. The use of stacks to manage undo/redo operations and lists to store the text enables seamless manipulation of text data. This project demonstrates a simple, user-friendly text editor with advanced features and efficient performance based on these core data structures.

Introduction :

Text editors are ubiquitous tools in computing, widely used for tasks ranging from simple note-taking to software development. They provide the basic functionality of adding, removing, and modifying text. However, many text editors offer minimal functionality beyond basic text manipulation. In real-world applications, users often require advanced features like undo/redo, find and replace, and importing content from external sources. This project aims to implement a simple yet advanced text editor that can handle these tasks efficiently. By integrating key data structures such as lists and stacks, the editor provides a flexible and reliable user experience.

Problem Statement

Most text editors provide basic functionalities such as inserting or deleting text. However, they often lack features that enhance user experience, such as the ability to undo and redo changes or to search and replace text. These limitations hinder productivity, especially in cases where users are working with long or complex documents. This project addresses this gap by developing a text editor that not only provides basic editing functionalities but also supports undo/redo, text search, and import from external sources, making it more versatile and user-friendly.

Objective

The goal of this project is to design and implement a text editor that provides the following features:

- Text insertion and deletion
- Undo/redo functionality using stacks
- Find and replace functionality for searching and modifying text
- Word count functionality for document analysis
- Importing text from local files and external URLs
- Saving and opening files to persist text data

Scope

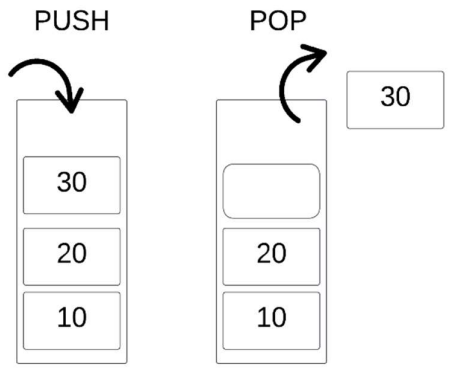
This project focuses on the core functionalities of a text editor, including the use of undo/redo, find and replace, and import/export features. While it doesn't support advanced formatting or multi-user collaboration, it provides a solid foundation for any text manipulation needs. The project demonstrates the application of basic data structures in real-world software development.

Methodology :

Data Structures Used

- **List**: The text is stored as a list of characters or strings. This choice allows efficient insertion and deletion at arbitrary positions in the text.

- **Stack**: A stack is used to implement the undo and redo functionality. Each action (like text insertion or deletion) is pushed onto the undo stack. When a user chooses to undo an operation, the action is popped from the undo stack and applied in reverse, while it's pushed onto the redo stack. Similarly, redo operations work by popping from the redo stack and applying the change again.



Features :

The core functionality of the text editor relies on basic operations such as inserting text, deleting text, and modifying text. The algorithm for each feature is as follows:

- **Insert Text**: When text is inserted at a specific position, the previous state of the text (prior to insertion) is pushed onto the undo stack.
- **Delete Text**: When text is deleted, the deleted text is saved in the undo stack to allow for re-insertion during an undo operation.
- **Undo**: The undo operation pops the most recent action from the undo stack and applies the reverse of that action (e.g., deletes inserted text or re-inserts deleted text).

- **Redo**: The redo operation applies the most recent undone action from the redo stack.
- **Find and Replace**: The editor searches for occurrences of a target string and replaces them with a new string.
- **Import from File/URL**: The editor uses file I/O functions and the requests library to fetch text from external sources.

Algorithm :

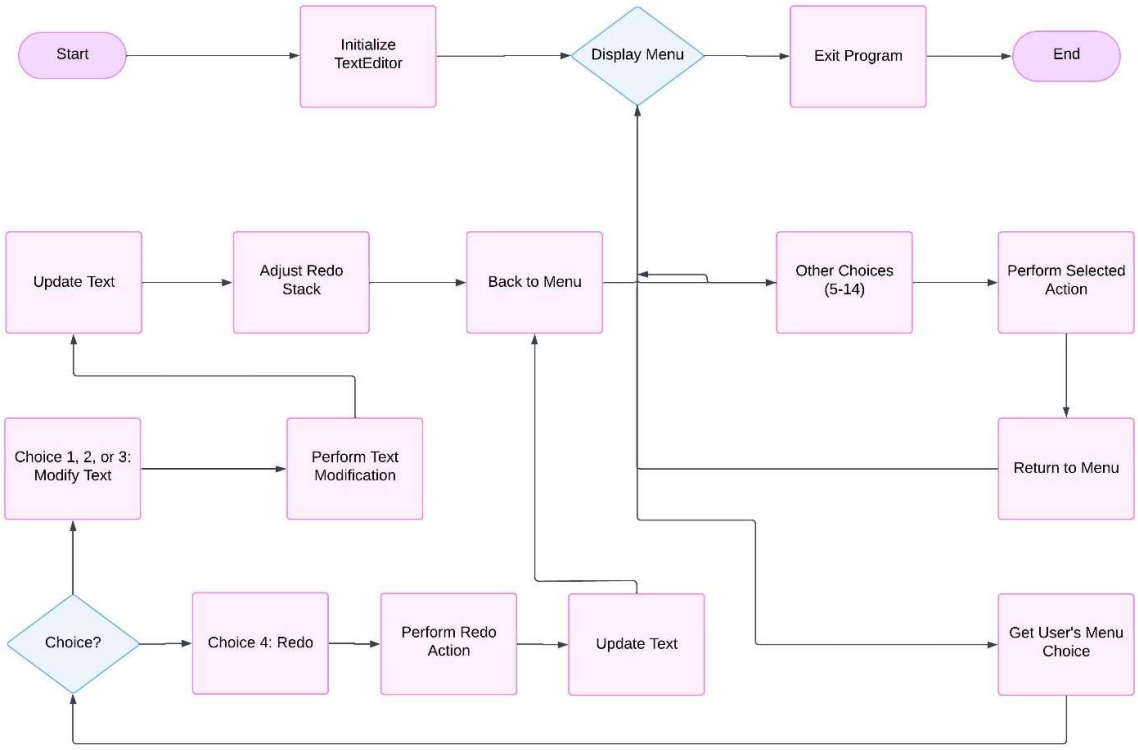
1. Initialize TextEditor:
    - Create a TextEditor object with empty text, undo_stack, and redo_stack.
2. Display Menu:
    - Show options (Insert, Delete, Undo, Redo, View Text, etc.).
3. Get User Input:
    - Prompt the user to enter a choice.
4. Execute Choice:
    - Based on the user's choice, perform the corresponding action:
        - Insert Text:
            - Get text and position from the user.
            - Insert text at the specified position.
            - Add an entry to the undo_stack.
            - Clear the redo_stack.
        - Delete Text:
            - Get start and end positions.
            - Remove text between those positions.
            - Add an entry to the undo_stack.
            - Clear the redo_stack.
        - Undo:
            - Retrieve the last action from undo_stack.
            - Revert the action and move it to redo_stack.
        - Redo:
            - Retrieve the last action from redo_stack.
            - Reapply the action and move it to undo_stack.
        - View Text:

        - Display the current text.
    - Clear Text:
        - Save current text in undo_stack.
        - Clear text and redo_stack.
    - Find and Replace / Replace All:
        - Replace specified text, add operation to undo_stack, and clear redo_stack.
    - Save / Open File:
        - Write to or read from a file.
        - Update undo_stack and clear redo_stack as needed.
    - Import from URL / File:
        - Fetch and append content from a URL or file.
        - Add current state to undo_stack and clear redo_stack.
    - Word Count:
        - Count words in text and display the count.
    - Exit:
        - End the program.
5. Loop Back to Menu:
    - After completing an action, display the menu again until the user chooses to exit.

Termination:

- The program exits when the user selects "Exit".

Flowchart :



Implementation :

Code:

```python
import requests
from bs4 import BeautifulSoup

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

1. Insert Text

- Purpose: To insert text at a specified position in the current document.
- Details: The text to be inserted is added at the given position, and the previous state of the document is saved in the undo stack so that it can be undone later.

```
def insert_text(self, text, position):
    print(f"Inserting text at position {position}: '{text}'")
    self.undo_stack.append(("delete", text, position))
    self.text = self.text[:position] + text + self.text[position:]
    self.redo_stack.clear()
    print(f"Text after insertion: {self.text}")
```

Output:

```
Text Editor Menu:
1. Insert Text
2. Delete Text
3. Undo
4. Redo
5. View Text
6. Clear Text
7. Find and Replace
8. Replace All
9. Save File
10. Open File
11. Word Count
12. Import from Another File
13. Import from URL
14. Exit
Enter your choice: 1
Enter text to insert: hello world
Enter position to insert: 0
Inserting text at position 0: 'hello world'
Text after insertion: hello world
```

2. Delete Text

- Purpose: To delete text from the document between a start and an end position.
- Details: The deleted text is saved to the undo stack so it can be reinserted during an undo operation.

```
def delete_text(self, start, end):
    print(f"Deleting text from position {start} to {end}")
    deleted_text = self.text[start:end]
    self.undo_stack.append(("insert", deleted_text, start))
    self.text = self.text[:start] + self.text[end:]
    self.redo_stack.clear()
    print(f"Text after deletion: {self.text}")
```

Output:

```
Text Editor Menu:
1. Insert Text
2. Delete Text
3. Undo
4. Redo
5. View Text
6. Clear Text
7. Find and Replace
8. Replace All
9. Save File
10. Open File
11. Word Count
12. Import from Another File
13. Import from URL
14. Exit
Enter your choice: 2
Enter start position to delete: 0
Enter end position to delete: 6
Deleting text from position 0 to 6
Text after deletion: world
```

3. Undo

- Purpose: To undo the most recent text insertion or deletion.
- Details: Pops the last action from the undo stack and reverses it. This action is then moved to the redo stack.

```
def undo(self):
    print("Undo operation")
    if self.undo_stack:
        action = self.undo_stack.pop()
        if action[0] == "insert":
            self.text = self.text[:action[2]] + action[1] + self.text[action[2]:]
        elif action[0] == "delete":
            self.text = self.text[:action[2]] + self.text[action[2] + len(action[1]):]
        self.redo_stack.append(action)
        print(f"Text after undo: {self.text}")
    else:
        print("Nothing to undo")
```

Output:

```
Text Editor Menu:
1. Insert Text
2. Delete Text
3. Undo
4. Redo
5. View Text
6. Clear Text
7. Find and Replace
8. Replace All
9. Save File
10. Open File
11. Word Count
12. Import from Another File
13. Import from URL
14. Exit
Enter your choice: 3
Undo operation
Text after undo: hello world
```

4. Redo

- Purpose: To redo the most recent undone operation.
- Details: Pops the last undone action from the redo stack and applies it again (either re-insert or delete the text).

```python
def redo(self):
    if self.redo_stack:
        action = self.redo_stack.pop()
        if action[0] == "insert":
            self.text = self.text[:action[2]] + self.text[action[2] + len(action[1]):]
        elif action[0] == "delete":
            self.text = self.text[:action[2]] + action[1] + self.text[action[2]:]
        self.undo_stack.append(action)
        self.view_text()
        print(self.text)
```

Output:

```
Text Editor Menu:
1. Insert Text
2. Delete Text
3. Undo
4. Redo
5. View Text
6. Clear Text
7. Find and Replace
8. Replace All
9. Save File
10. Open File
11. Word Count
12. Import from Another File
13. Import from URL
14. Exit
Enter your choice: 4
Viewing current text
world
```

5. View Text

- Purpose: To display the current content of the text editor.
- Details: Simply returns the current text content for viewing.

```
def view_text(self):
    print("Viewing current text")
    return self.text
```

6. Clear Text

- Purpose: To clear all text from the document.
- Details: The current state of the text is saved in the undo stack so that it can be restored in case of an undo operation.

```python
def clear_text(self):
    print("Clearing text")
    self.undo_stack.append(("clear", self.text))
    self.text = ""
    self.redo_stack.clear()
```

7. Find and Replace

- Purpose: To find a specific string and replace it with another.
- Details: This function uses Python's replace() method to perform the replacement and saves the previous state for undo functionality.

```python
def find_and_replace(self, find_text, replace_text):
    print(f"Find and replace: '{find_text}' with '{replace_text}'")
    self.undo_stack.append(("replace", find_text, replace_text, self.text))
    self.text = self.text.replace(find_text, replace_text)
    self.redo_stack.clear()
    print(f"Text after replace: {self.text}")
```

Output:

```
Enter your choice: 1
Enter text to insert: Hello World
Enter position to insert: 0
Inserting text at position 0: 'Hello World'
Text after insertion: Hello World

Text Editor Menu:
1. Insert Text
2. Delete Text
3. Undo
4. Redo
5. View Text
6. Clear Text
7. Find and Replace
8. Replace All
9. Save File
10. Open File
11. Word Count
12. Import from Another File
13. Import from URL
14. Exit
Enter your choice: 7
Enter text to find: World
Enter text to replace with: Kitty
Find and replace: 'World' with 'Kitty'
Text after replace: Hello Kitty
```

8. Replace All

- Purpose: To replace all occurrences of a string with another.
- Details: This function replaces all occurrences and provides the number of replacements made.

```
def replace_all(self, find_text, replace_text):
    print(f"Replace all occurrences of '{find_text}' with '{replace_text}'")
    self.undo_stack.append(("replace", find_text, replace_text, self.text))
    count = self.text.count(find_text)
    self.text = self.text.replace(find_text, replace_text)
    self.redo_stack.clear()
    print(f"Replaced {count} occurrences of '{find_text}' with '{replace_text}'.")
    print(f"Text after replacing all: {self.text}")
```

9. Save File

- Purpose: To save the current content to a file.
- Details: This function writes the current content to a specified file.
-

```
def save_file(self, filename):
    print(f"Saving text to file: {filename}")
    with open(filename, "w") as file:
        file.write(self.text)
    print(f"File saved as {filename}")
```

Output:


```
Text Editor Menu:
1. Insert Text
2. Delete Text
3. Undo
4. Redo
5. View Text
6. Clear Text
7. Find and Replace
8. Replace All
9. Save File
10. Open File
11. Word Count
12. Import from Another File
13. Import from URL
14. Exit
Saving text to file: demo
File saved as demo
```

10. Open File

- Purpose: To open and load the content of a file into the editor.
- Details: Reads the content of a file and replaces the current text in the editor with the file's content.

```
def open_file(self, filename):
    print(f"Opening file: {filename}")
    try:
        with open(filename, "r") as file:
            self.undo_stack.append(("open", self.text))
            self.text = file.read()
        print(f"File {filename} opened")
        self.redo_stack.clear()
    except FileNotFoundError:
        print(f"File not found: {filename}")
```

11. Word Count

- Purpose: To calculate the word count of the current document.
- Details: This function splits the current text by spaces and returns the number of words.

```
def word_count(self):
    return len(self.text.split())
```

12. Import from Another File

- Purpose: To import text from an external file and append it to the current document.
- Details: Reads text from a file and appends it to the existing text, saving the previous text for undo operations.

```
def import_from_file(self, filename):
    print(f"Importing content from file: {filename}")
    try:
        with open(filename, "r") as file:
            imported_text = file.read()
            self.undo_stack.append(("import", self.text))
            self.text += "\n" + imported_text
        print(f"Content from {filename} imported")
        self.redo_stack.clear()
    except FileNotFoundError:
        print(f"File not found: {filename}")
```

Output:

```
Text Editor Menu:
1. Insert Text
2. Delete Text
3. Undo
4. Redo
5. View Text
6. Clear Text
7. Find and Replace
8. Replace All
9. Save File
10. Open File
11. Word Count
12. Import from Another File
13. Import from URL
14. Exit
Enter your choice: 12
Enter filename to import: /content/external.txt
Importing content from file: /content/external.txt
Content from /content/external.txt imported

Text Editor Menu:
1. Insert Text
2. Delete Text
3. Undo
4. Redo
5. View Text
6. Clear Text
7. Find and Replace
8. Replace All
9. Save File
10. Open File
11. Word Count
12. Import from Another File
13. Import from URL
14. Exit
Enter your choice: 5
Viewing current text

This is the external file
```

13. Import from URL

- Purpose: To fetch and append text content from a URL.
- Details: Uses the requests library to fetch the page content and BeautifulSoup to extract text. The content is appended to the current text.

```python
def import_from_url(self, url):
    print(f"Importing content from URL: {url}")
    try:
        response = requests.get(url)
        response.raise_for_status()
        soup = BeautifulSoup(response.text, 'html.parser')
        imported_text = soup.get_text()

        self.undo_stack.append(("import", self.text))
        self.text += "\n" + imported_text
        print("Content imported from URL")
        self.redo_stack.clear()

    except requests.exceptions.RequestException as e:
        print(f"Failed to fetch content from URL: {e}")
```

Output:

```
Text Editor Menu:
1. Insert Text
2. Delete Text
3. Undo
4. Redo
5. View Text
6. Clear Text
7. Find and Replace
8. Replace All
9. Save File
10. Open File
11. Word Count
12. Import from Another File
13. Import from URL
14. Exit
Enter your choice: 13
Enter URL to import text from: https://www.lipsum.com/
Importing content from URL: https://www.lipsum.com/
Content imported from URL
```

Driver code:

```
def main():
    editor = TextEditor()

    while True:
        print("\nText Editor Menu:")
        print("1. Insert Text")
        print("2. Delete Text")
        print("3. Undo")
        print("4. Redo")
        print("5. View Text")
        print("6. Clear Text")
        print("7. Find and Replace")
        print("8. Replace All")
        print("9. Save File")
        print("10. Open File")
        print("11. Word Count")
        print("12. Import from Another File")
        print("13. Import from URL")
        print("14. Exit")

        choice = input("Enter your choice: ")

        if choice == "1":
            text = input("Enter text to insert: ")
            position = int(input("Enter position to insert: "))
            editor.insert_text(text, position)
        elif choice == "2":
            start = int(input("Enter start position to delete: "))
            end = int(input("Enter end position to delete: "))
            editor.delete_text(start, end)
        elif choice == "3":
            editor.undo()
        elif choice == "4":
            editor.redo()
        elif choice == "5":
            print(editor.view_text())
```

```
        elif choice == "4":
            editor.redo()
        elif choice == "5":
            print(editor.view_text())
        elif choice == "6":
            editor.clear_text()
        elif choice == "7":
            find_text = input("Enter text to find: ")
            replace_text = input("Enter text to replace with: ")
            editor.find_and_replace(find_text, replace_text)
        elif choice == "8":
            find_text = input("Enter text to find: ")
            replace_text = input("Enter text to replace with: ")
            editor.replace_all(find_text, replace_text)
        elif choice == "9":
            filename = input("Enter filename to save: ")
            editor.save_file(filename)
        elif choice == "10":
            filename = input("Enter filename to open: ")
            editor.open_file(filename)
        elif choice == "11":
            print("Word count:", editor.word_count())
        elif choice == "12":
            filename = input("Enter filename to import: ")
            editor.import_from_file(filename)
        elif choice == "13":
            url = input("Enter URL to import text from: ")
            editor.import_from_url(url)
        elif choice == "14":
            print("Exiting the editor.")
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

Results and Analysis

Performance Analysis

The time complexity of basic operations like insert and delete is O(n), where n is the length of the text, due to the need to rearrange the list. The undo/redo operations work in O(1) time since they involve simple stack manipulations. The search and replace operation has a time complexity of O(n*m), where n is the length of the text and m is the number of occurrences of the target string.

Comparison :

Compared to other text editors, the key advantage of this editor is its simplicity and its ability to handle multiple editing operations (insert, delete, undo, redo) seamlessly with the stack-based approach. The ability to import from URLs is another unique feature, as it enables easy integration of external content for editing.

Conclusion :

The text editor developed in this project meets the core objectives, implementing essential text manipulation features like insert, delete, undo/redo, and find and replace. By utilizing stacks for undo/redo operations and lists for text storage, the editor provides efficient and flexible management of text data. The import/export functionality enhances its utility for real-world applications.

Future Work

Future improvements could include adding more advanced text formatting features, like bold, italics, or fonts. Additionally, the integration of a Graphical User Interface could make the editor more user-friendly and accessible. Further optimization for large text files and more advanced search algorithms could be implemented to handle specific use cases.

Tools :

- LucidChart - https://lucid.app/lucidchart/
- Google Colab

References :

- Geeks for Geeks - https://www.geeksforgeeks.org/stack-data-structure/
- Tutorialspoint - https://www.tutorialspoint.com/beautiful_soup/index.htm