Work-Integrated Learning Programmes Division

MTech in Data Science & Engineering

S2_2022-2023,DSECLZG519 - Data Structures and Algorithms Design

Assignment 1 – PS03 - TextEditor
GROUP - 113

Student Registration Number	Name	Percentage of contribution out of 100%
2022dc04425	Rahul Rohilla	100%
2022dc04404	Rohini	100%
2022dc04402	Srijita Roy	100%
2022dc04282	Kovvuri Satyanarayana Reddy	100%

Introduction

This document outlines the design of a command-based text editor which uses a doubly linked list data structure. The text editor enables the user to do operations like add text, move the cursor left and right, delete text, and print the content.

Requirements

- To construct a command based text-editor using doubly-linked-list.
- Each node in the linked-list should store one character.
- The TextEditor should support the ability to add text, move the cursor left and right, delete text, and print the output.
- Analyse the time complexity of the code.
- Implement the algorithm using Python 3.7+.

Input and Output Files

- Read the input from inputPS03.txt
- After performing all the command operations given in input file, write the result into outputPS03.txt

Classes and their functions used

1. **node class:**
    a. **__init__(self, val):** This is the constructor function of the class node. It takes (val) as a parameter and initialises a node object with the value (val), and defines the next and prev pointers of the node to None.
2. **DoubleLinkedList class:**
    a. **__init__(self):** This is the constructor method of the DoubleLinkedList class. It initialises a double linked list by creating a head node with the value "<start>". It also sets the cursor initially to the head node.
    b. **insert(self, val):** This method is used to insert a new node with a given value (val) next to the cursor node. It creates a new node with the given value, sets its prev pointer to the current cursor node, and its next pointer to the node that follows the cursor. It updates the prev and next pointers of adjacent nodes accordingly. Finally, it moves the cursor to the newly inserted node.
    c. **delete(self):** This method is used to delete the current node (cursor node) from the double linked list. It first checks if the cursor is at the head node ("<start>") and returns if it is. Otherwise, it updates the prev and next

pointers of adjacent nodes to skip the current node. It then deletes the current node and moves the cursor to the previous node.

d. **display(self):** This method is used to display the values of nodes in the double linked list from the first node to the last node. It starts from the head node and iterates through the list, printing the value of each node.

3. **TextEditor class:**
    a. **__init__(self):** This is the constructor method of the TextEditor class. It initialises a text editor object by creating an instance of the **DoubleLinkedList class.**
    b. **AddText(self, letter):** This method is used to add text (a single character) to the double linked list. It calls the **insert method** of the DoubleLinkedList class to insert the letter next to the cursor node.
    c. **DeleteText(self, no_of_spaces):** This method is used to delete a specified number of characters (spaces) from the double linked list. It calls the **delete method** of the DoubleLinkedList class repeatedly for the given number of spaces.
    d. **MoveRight(self, no_of_spaces):** This method is used to move the cursor to the right by a specified number of positions (spaces). It updates the cursor by setting it to the next node, repeatedly for the given number of spaces, as long as there is a next node available.
    e. **MoveLeft(self, no_of_spaces):** This method is used to move the cursor to the left by a specified number of positions (spaces). It updates the cursor by setting it to the previous node, repeatedly for the given number of spaces, as long as there is a previous node available.
    f. **PrintText(self):** This method is used to print the text stored in the double linked list. It opens the "outputPS03.txt" file in append mode and iterates through the double linked list, printing each node's value to the file.

Algorithm/Steps followed

1. The code defines classes for node, DoubleLinkedList, and TextEditor, which are used to represent the double linked list and the text editor functionalities.
2. It reads the commands from the **input file (inputPS03.txt).**
3. It initialises a text editor instance.
4. It clears the contents of the **output file (outputPS03.txt)** before starting the program.
5. The code iterates through each command in the input file.
6. For each command:

a. If the command is **"AddText"**, it adds the specified text to the text editor using the **AddText method**. For example, the command "AddText HW" adds the characters "HW" to the text editor. Each character is stored in one node.
b. If the command is **"MoveLeft"**, it moves the cursor to the left by the specified number of spaces using the **MoveLeft method**. For example, the command "MoveLeft 2" moves the cursor two spaces to the left.
c. If the command is **"MoveRight"**, it moves the cursor to the right by the specified number of spaces using the **MoveRight method**. For example, the command "MoveRight 1" moves the cursor one space to the right.
d. If the command is **"DeleteText"**, it deletes the specified number of characters from the text editor using the **DeleteText method**. For example, the command "DeleteText 4" deletes four characters from the text editor.
e. If the command is **"PrintText",** it prints the current text stored in the text editor by calling the **PrintText method**. The text is written to the **output file (outputPS03.txt).**

7. After executing all the commands, the resulting text in the text editor is written to the output file.

Design

- The code demonstrates a design that implements a basic text editor using a double linked list data structure. It includes:
    - Operations:
        - Adding Text.
        - Deleting Text.
        - Moving Cursor.
        - Printing Text.
    - Command Execution:
        - The code reads each command from the input file and determines the main command and any additional parameters.
        - Based on the main command, it calls the corresponding method of the TextEditor class to perform the operation.
        - The code executes the commands one by one, modifying the text editor's state and updating the cursor position accordingly.
- The design separates the responsibilities into different classes: the **node class** handles the node functionality, the **DoubleLinkedList class** manages the double linked list operations, and the **TextEditor class** provides the text editing functionality by utilising the double linked list.

- This design allows for **modular code structure** and **encapsulation of data and operations**, making it easier to manage and maintain the text editor's functionality.
- Upper limit on adding the number of characters for the AddText command has not been added intentionally, hence the code will not throw an exception for full capacity.

Time Complexity Analysis

1. **Adding Text:** Adding a single character to the double linked list has a constant time complexity because it involves creating a new node, updating the pointers of adjacent nodes, and moving the cursor to the newly inserted node. Therefore, the time complexity is **O(n)**, where k is the total number of characters added.
2. **Deleting Text:** Deleting a character from the double linked list also has a constant time complexity because it involves updating the pointers of adjacent nodes and deleting the current node. Therefore, the time complexity is **O(n)**, where d is the total number of deletion operations.
3. **Moving Cursor**: Moving the cursor to the left or right by a specified number of spaces also has a constant time complexity because it involves updating the cursor by previous node (for left) and by next node (for right). Therefore, the time complexity of moving the cursor for all commands is **O(n)**, where m is the total number of movement operations.
4. **Printing Text:** The time complexity of printing the text is proportional to the number of characters in the double linked list, given by **O(n)**, as printing the text involves iterating through the double linked list and writing each character to the output file.

The chosen operations in the code are efficient for the given design because they have a **linear time complexity** proportional to the number of characters in the double linked list **(O(n))**. This design choice allows the text editor to handle operations quickly and efficiently, regardless of the size of the text.

Alternate way of modelling the code with the cost implications

An alternative design is to use a dynamic array instead of a double linked list to represent the text content in the text editor. This design choice has cost implications. Adding and deleting text may have occasional higher time complexity due to resizing and shifting operations. However, moving the cursor and printing the text remain efficient with a constant or linear time complexity, respectively. The choice between a dynamic array and a double linked list depends on the text editor's requirements, usage patterns, and the trade-off between time complexity and memory usage.