**Q1 Report**

**Class Diagram:**

**A close up of a map

Description automatically generated**

**Problem Solution Approach:**

In this problem instead of using a normal node to implement a LinkedList we used a node that contains an array as a data holder, and it has the same functionality as normal linked list but without having to create a new node every time we add an element.

It has every function a normal linked list var such as add(), remove(),set(),get()...etc.

**Test cases:**

|  |  |  |
| --- | --- | --- |
| Test scenario | Test case | Result |
| Adding an element | Providing an element with an index | inserting an element at the provided index |
| Providing an element without an index | Inserting the element at the end of the list |
| Removing an element | Providing an existing element | Removing the element provided |

**Running command and results:**

**A screenshot of a cell phone

Description automatically generated A screen shot of a smart phone

Description automatically generated**

**Q2 Report**

**Class Diagram:**

**A screenshot of a social media post

Description automatically generated**

**Problem Solution Approach:**

To Create a simple text editor first I used an List to collect the strings that are going to be edited (added, replaced or searched for) then edit them using various functions like add, replace and find then write the strings inside the List to a file using the write function and to measure the time it took for every function I wrote a test class which is the main class and logged all the required data inside a log file.

**Test Cases:**

|  |  |  |
| --- | --- | --- |
| Test Scenario | Test Case | Result |
| Adding to the head of the List | A String and 0 as Index is Provided | String is added to the head of the List |
| Adding to the Middle of the List | A String and an Index is Provided | String is added to the Middle of the List |
| Replacing a string | An old and a new String is provided | All occurrences of the old string are replaced with the new string |
| Finding a String | An existing String is provided | The index of the first character of the first occurrence of the string is returned |
| A non-existing string is provided | -1 is returned |

**Running command and results:**

**A close up of a screen

Description automatically generated**

**Theoretical analysis of the functions:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function | LinkedList | ArrayList | LL with iterator | AL with iterator |
| Read | Θ(n) because it depends on how many characters are in the file and O(1) for the add function inside of it because it is always adding to the end of the list | Θ(n) because it depends on how many characters are in the file and O(1) for the add function inside of it because it is always adding to the end of the list | Θ(n) because it depends on how many characters are in the file and O(1) for the add function inside of it because it is always adding to the end of the list | Θ(n) because it depends on how many characters are in the file and O(1) for the add function inside of it because it is always adding to the end of the list |
| Add | Best case is θ(n) adding to the head or the end of the list because it depends on how many Chars the String has  Worst case is θ(n2) if adding to the middle of the list because it has to travers to the index provided and then add the chars so overall O(n2) or Ω(n) | Best case is θ(n) adding to the end of the list because it depends on how many Chars the String has  Worst case is θ(n2) if adding to the head or middle of the list because it has to shift everything after the index provided and then add the chars so overall O(n2) or Ω(n) | Θ(n) because the iterator always adds at a rate of θ(1) for every Char and it depends on the number of Chars | Best case is θ(n) adding to the end of the list because it depends on how many Chars the String has  Worst case is θ(n2) if adding to the head or middle of the list because it has to shift everything after the index provided and then add the chars so overall O(n2) or Ω(n) |
| Replace | O(n3) because it has a nested loop which is O(n2) and inside them the remove and add operation have a complexity of O(n) because when adding or removing it has to traverse to the given index | O(n3) because it has a nested loop which is O(n2) and inside them the remove and add operation have a complexity of O(n) because when adding or removing it has to shift the elements after the given index | O(n2) because it has a nested loop which is O(n2) and inside them the remove and add operation have a complexity of O(1) because when adding or removing it uses the iterator which prevents the need to traverse the elements | O(n3) because it has a nested loop which is O(n2) and inside them the remove and add operation have a complexity of O(n) because when adding or removing it has to shift the elements after the given index |
| Find | θ(n) because it has to traverse the List to find the given index | θ(n) because it has to traverse the List to find the given index | θ(n) because it has to traverse the List to find the given index | θ(n) because it has to traverse the List to find the given index |

**Experimental Results:**

**Logging Methods of Linked List without iterator**

**Time elapsed for Read function = 60 Micro Second**

**Time elapsed for add function from the head of the list = 968 Micro Second**

**Time elapsed for add function from the middle of the list = 1023 Micro Second**

**Time elapsed for replace function = 978 Micro Second**

**Time elapsed for Find function = 72 Micro Second**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Logging Methods of Linked List with iterator:**

**Time elapsed for Read function = 232 Micro Second**

**Time elapsed for add function from the head of the list = 982 Micro Second**

**Time elapsed for add function from the middle of the list = 1000 Micro Second**

**Time elapsed for replace function = 921 Micro Second**

**Time elapsed for Find function = 30 Micro Second**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Logging Methods of Array List without iterator**

**Time elapsed for Read function = 38 Micro Second**

**Time elapsed for add function from the head of the list = 774 Micro Second**

**Time elapsed for add function from the middle of the list = 937 Micro Second**

**Time elapsed for replace function = 880 Micro Second**

**Time elapsed for Find function = 42 Micro Second**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Logging Methods of Array List with iterator**

**Time elapsed for Read function = 154 Micro Second**

**Time elapsed for add function from the head of the list = 796 Micro Second**

**Time elapsed for add function from the middle of the list = 925 Micro Second**

**Time elapsed for replace function = 914 Micro Second**

**Time elapsed for Find function = 27 Micro Second**