CS3310 Report

Genetic Palindromes, Linked Lists

Western Michigan University

Instructor: Dr. Gupta

By Anfaal Faisal

Specifications:

The main goal of the assignment:

Develop a data structure that implements the same principles of operation as in linked lists.

Sub Goals:

1. Learn the principles of doubly linked lists
2. Based on the knowledge gained, complete own implementation of the doubly linked list
3. implement a method that counts the number of list items
4. Implement a delete list method
5. implement the method of adding data from the beginning and end of the list
6. implement a method that adds the contents of one list to another
7. implement a method of inserting a list item at a specified position
8. Using the doubly linked list class, save, sort, and search for information.
9. Implement an algorithm for selecting palindromes from a genomic sequence
10. Test the running time of the algorithm on different sets of character sequences
11. Create javadoc file from comments with description of methods and fields of application classes

Design

Functions/global variables List for class DoublyLinkList.java

private int length = 0;

Element begin = null;  
Element end = null;

public Element PushToBegin(T data)

public Element PushEnd(T data)

public DoublyLinkList<T> Append( DoublyLinkList<T> b )

public void sortedInsert(Element element)

public void InsertNth(int index, T value)

public T PopBegin()

public T PopEnd()

public T GetNth(int index)

int Count(T need)

public int Length()

Functions/global variables List for class Element.java

private T data;  
private Element AddressNext;  
private Element AddressPrevious;  
public Element(){ }

public Element(T data)

public Element(T Data,Element next, Element prev)

public void setData(T data)

public void setAddressNext(Element addressNext)

public void setAddressPrevious(Element addressPrevious)

public T getData()

public Element getAddressNext()

public Element getAddressPrevious()

Functions/global variables List for class Main.java

static public DoublyLinkList<Integer> BuildOneTwoThree()

public static void Counter()

public static void Demonstration()

public static void Mirror(String str,int number)

static DoublyLinkList<String> *AllSequenxes* = new DoublyLinkList<String>();

Space ComplexityAnalysis:

For class Main.java

|  |  |  |
| --- | --- | --- |
| **Function** | **Inputs** | **Complexity** |
| Main | 0 | O(n) |
| Mirror | 2 | O(n) |
| Counter | 1 | O(1) |
| Demonstration | 0 | O(1) |
| BuildOneTwoThree | 0 | O(1) |

For class DoublyLinkList.java

|  |  |  |
| --- | --- | --- |
| **Function** | **Inputs** | **Complexity** |
| Count | 1 | O(1) |
| GetNth | 1 | O(1) |
| PopBegin | 0 | O(1) |
| PopEnd | 0 | O(1) |
| PushEnd | 1 | O(n) |
| PushToBegin | 1 | O(n) |
| Append | 1 | O(n) |
| Length | 0 | O(1) |
| sortedInsert | 1 | O(n) |
| GetNth | 1 | O(n) |

For class Element.java

|  |  |  |
| --- | --- | --- |
| **Function** | **Inputs** | **Complexity** |
| setData | 1 | O(1) |
| getAddressPrevious | 0 | O(1) |
| setAddressPrevious | 1 | O(1) |
| getData | 0 | O(1) |
| getAddressNext | 0 | O(1) |
| setAddressNext | 1 | O(1) |

Time complexity analysis:

For class Main.java

|  |  |  |
| --- | --- | --- |
| **Function** | **Inputs** | **Complexity** |
| Main | 0 | O(1) |
| Mirror | 2 | O(n) |
| Counter | 1 | O(1) |
| Demonstration | 0 | O(1) |
| BuildOneTwoThree | 0 | O(1) |

For class DoublyLinkList.java

|  |  |  |
| --- | --- | --- |
| **Function** | **Inputs** | **Complexity** |
| Count | 1 | O(n) |
| GetNth | 1 | O(n) |
| PopBegin | 0 | O(1) |
| PopEnd | 0 | O(1) |
| PushEnd | 1 | O(1) |
| PushToBegin | 1 | O(1) |
| Append | 1 | O(n) |
| Length | 0 | O(1) |
| sortedInsert | 1 | O(n) |
| GetNth | 1 | O(n) |

For class Element.java

|  |  |  |
| --- | --- | --- |
| **Function** | **Inputs** | **Complexity** |
| setData | 1 | O(1) |
| getAddressPrevious | 0 | O(1) |
| setAddressPrevious | 1 | O(1) |
| getData | 0 | O(1) |
| getAddressNext | 0 | O(1) |
| setAddressNext | 1 | O(1) |

Result

Since the task was to analyze the sequence of characters. And the length of this sequence could be quite large. Therefore, a linked list was used for work. Since adding and removing elements from the beginning or the end in it is quite fast, regardless of the number of elements in the list. Because each character of the sequence was simply added to the current list, without changing the values ​​of the remaining elements of the list (except for the previous one). Therefore, it is worth noting that the time for adding is O (1). However, this data structure also has certain drawbacks, in order to get the results from a part of the list, we need to sort through all the elements until we get to the desired one, so here the processing time of the operations increases linearly O (n). The more elements, the more steps will have to be done to get information about the desired element.

Consider the operation of a method that receives a set of characters at the input:

Mirror(character set, set number)

initialize the empty list

in a loop put each character in a list

Next, select a set in a temporary variable of a given length

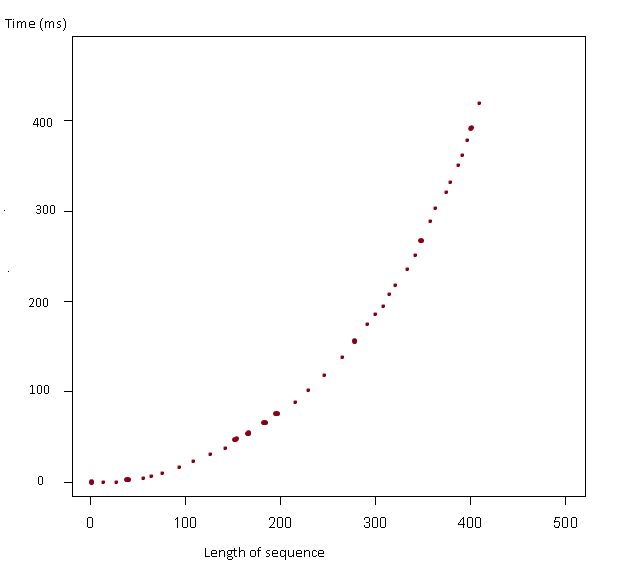
in a loop, we go through the values ​​on the contrary, through the symbols of this variable

and put them in the second temporary set

next, we compare each of the variables of these two sets according to the appropriate gene pairs

if the comparison is true and the word length is not interrupted, then the search word is placed in the set of search sequences

so we get a list of one input line



As you can see from the graph, the larger the size of the number of characters, the longer its processing time.

Next I will give the time spent searching for sequences depending on the number of characters, in addition, it is worth noting that the number of found and repeating palindromes also increases the algorithm's running time.

Below is a list of lines, as well as the time spent by the first palindrome, and the time spent by all the palindromes:

GTTTTGGAACCAAAACCTTGCAAAACCUUG

time to test whether a genomic subsequence in Sequence 2 is palindromic: 5 milliseconds

time to find all palindromic subsequences of length [4, 17] in Sequence 2 : 9 milliseconds

TTTTGGAACCAACCTTGCAAAACCUUG

time to test whether a genomic subsequence in Sequence 7 is palindromic: 4 milliseconds

time to find all palindromic subsequences of length [4, 17] in Sequence 7 : 7 milliseconds

GTTTTGGAACCAAAACCTTGCAAAACCUUG

time to test whether a genomic subsequence in Sequence 10 is palindromic: 24 milliseconds

time to find all palindromic subsequences of length [4, 17] in Sequence 10 : 37 milliseconds

CAAAACCUUGCAAAACCTTGGTTTTGGAAC

time to test whether a genomic subsequence in Sequence 45 is palindromic: 2 milliseconds

time to find all palindromic subsequences of length [4, 17] in Sequence 45 : 3 milliseconds