Aditi Jorapur
CS 146 Section 4
Potika
9/28/22

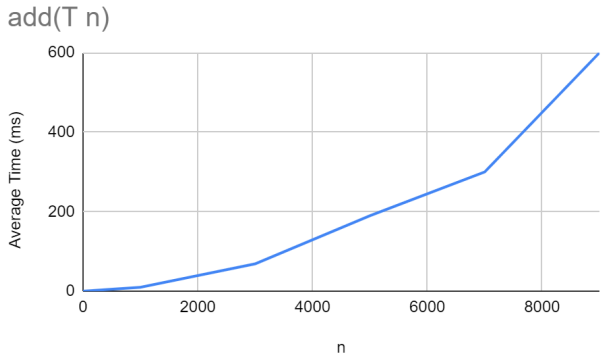
Project 1 Code Report

Introduction: The purpose of this project was to create our own SinglyLinkedList without the help of any collections. In this project, I made a SinglyLinkedList class in this project that takes in the generic type <T>. By using the generic type, the singly linked list can be used with any data type not only specific ones.

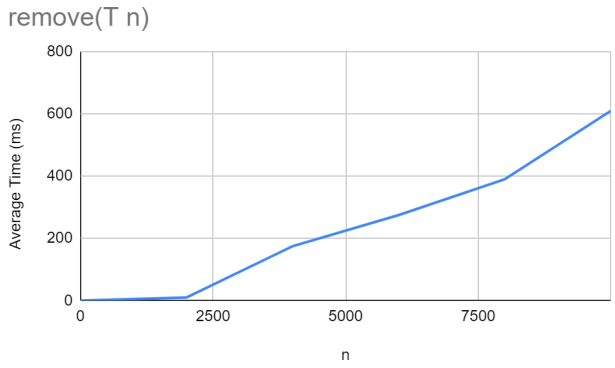
I first create the singly linked list and create a private instance variable called head and initialize it by setting it to null. Inside the SinglyLinkedList Class, I created an inner class called Node. In this class I created two instance variables called data which is the value of the Node and next which holds the next location of the node in the LinkedList. The Node Class has access to the methods in the SinglyLinkedList Class. These methods are add, remove, numItems, isMember, reference, union, intersection, and clone. A toString method has also been created to aid in the JUnit tests made to test the code for the class. The add method adds a node at the beginning of the LinkedList. The remove method removes a node from the LinkedList. The numItems method counts the number of items in the LinkedList and returns the size. The isMember method checks to see if the data is in the LinkedList. The reference method returns the Node object for a specific data. The union method returns a new SinglyLinkedList with all the distinct elements from both sets. The intersection method returns a new SinglyLinkedList with all the common distinct elements in two sets. The clone method duplicates a SinglyLinkedList.

Node Class:
This is an inner class in the SinglyLinkedList class. It gives the class access to the Node object. SinglyLinkedIIsts consists of Node objects which is why we use the inner Node class. The SinglyLinkedList class implements Generics <T> which means any data type can be used as a Node. For example, you can use integers, Strings, or any other data types in the class.

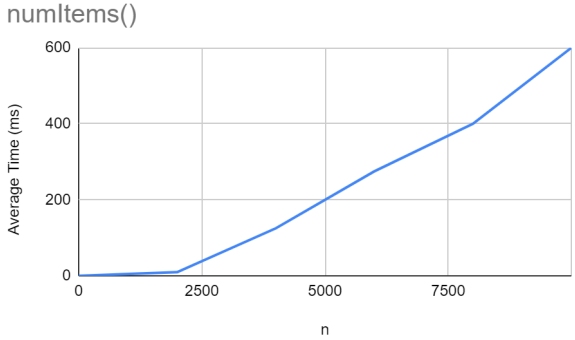
public void add(T n): *Big O Notation- O(n)*
This public method first creates a new node with the data called node. Then, it checks to see if the list is empty by checking to see if the head is equal to null. If it is empty, it adds the new node at the front of the list. Then, it checks if the node is already a member of the set. If it is not a member of the set, then the node is added at the head of the LinkedList.



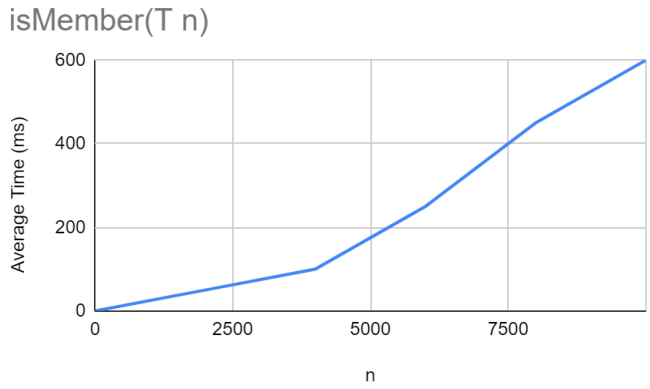
public void remove(T n): *Big O Notation- O(n)*
This public method removes a node from the SinglyLinkedList. First it checks whether the list is empty by checking if the head is null. If it is empty, then the method just returns and doesn't do anything. Then it checks whether the node we want to remove is in the head. If it is in the head, it will remove the node from the head and set the next node to the new head. Then, it creates a temporary node called node which is equal to the head. The method traverses through the LinkedList until it finds the next node that has the same data as 'n'. If the data is equal to 'n', then we point to the node after to remove the node.



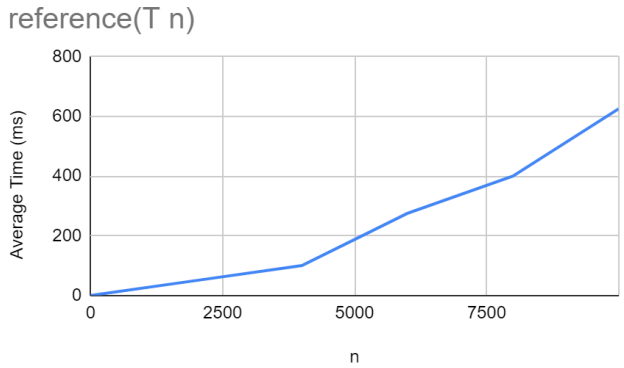
public void numItems(): *Big O Notation- O(n)*
This method returns the number of items in the set. It sets the temporary variable current to the head and then traverses through the LinkedList until it reaches the end. It then returns the size.



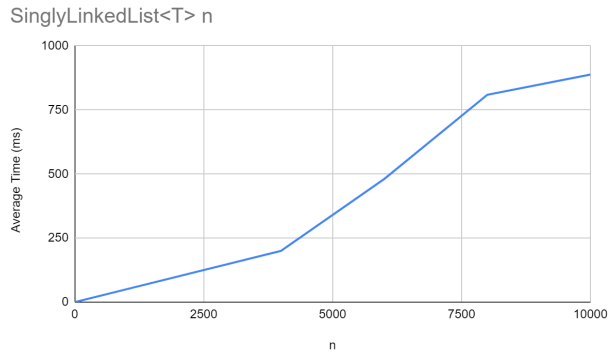
public void isMember(T n): *Big O Notation- O(n)*
This method checks to see if the data given is a member of the set. First it sets the temporary node current to head. If the list is empty, it will return false because the data is not in the LinkedList. Otherwise it will traverse through the list and if the data is found, it will return true. If the data is not found, it will return false.



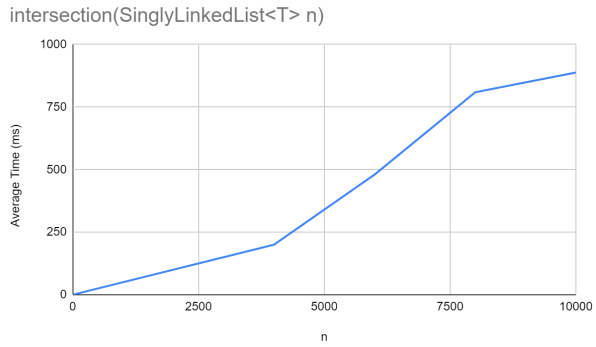
public Node reference(T n): *Big O Notation- O(n)*
This method returns a reference node to a specific node in the set. First, it creates a temporary node called current and sets it to head. Then, it traverses through the SinglyLinkedList and if the data is found in the LinkedList, it will return the node. If the list is empty, it will return null.



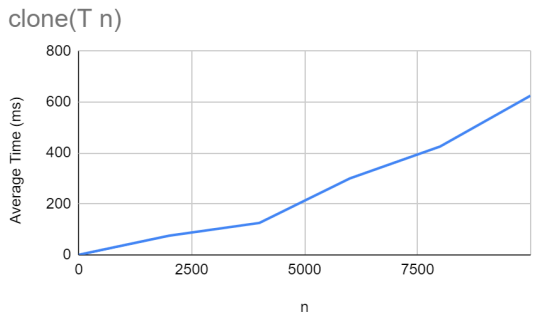
public SinglyLinkedList<T> union(SinglyLinkedList<T> n): *Big O Notation- O(n * n) → O(n^2)*
This method finds the and returns the set of all distinct elements. It first creates setUnion which is a singlyLinkedList to return the set of all distinct elements. It creates two new temporary elements that are equal to the head. If either of the lists are empty, it will return null because there are no common nodes between the two sets. If either of the lists are empty, it will return the other list. Otherwise, it traverses through the lists and adds the data in setUnion. Then, it will return the list with the distinct elements.



public SinglyLinkedList<T> intersection(SinglyLinkedList<T> n): *Big O Notation- O(n*n) → O(n^2)*
This method finds the set of all common distinct elements between two sets. It creates a new SinglyLinkedList called setIntersection and two temporary nodes that store the heads of both of the sets. If either one of the lists are empty, it will return the empty setIntersection list. Otherwise, it will traverse through the linked list and check if the node is found in the other list. If it is found, it will be added to the setIntersection list. Then, it will return the new SinglyLinkedList of all the common distinct elements.



public SinglyLinkedList<T> clone(SinglyLinkedList<T> input): *Big O Notation- O(n)*
This method clones a given singly linked list. It creates a new node called current which is set to the head of the given list. It checks to see if the list isn't empty and if it isn't, then it sets the head of the new list to the one of input. Then, it traverses through the list and duplicates the nodes into the new list, and returns the new list.



public String toString():
I created a toString method to print out the sets I created for the JUnit tests. First, it started off by creating a temporary node called current as the head. Then it creates a String answer which starts off with a bracket. It then traverses through the LinkedList and adds the data to the answer String.

Conclusion:
Before working on this project, I had never completely worked with LinkedLists before. By working on this project I learned how to implement a SinglyLinkedList and I also learned about all of the components I need to create on my own to build a SinglyLinkedList. Previously, I was always given starter code when working with linkedlists so I never thought about the things that go into implementing one on my own. I learned about the inner Node class and how that is used in the SinglyLinkedList Class. I also implemented Generics which helps make the LinkedList class generic allowing for the convenience of any data type to be used. Creating the JUnit test cases also helped me fix many of my methods. In some cases, I was returning the wrong things which I didn't realize until I ran the JUnit cases. I was having some trouble having the JUnit test cases run but I was able to fix that issue by restarting my ide and reconfiguring the JUnit path.