**S1 2022-2023**

**DSECLZG519**

**Data Structures and Algorithm Design**

**Assignment 1 – PS14 – Name Chain**

# Team Details

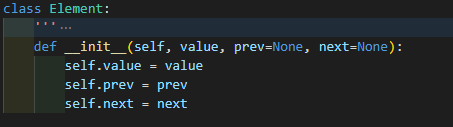
|  |  |  |  |
| --- | --- | --- | --- |
| **Sl No** | **Name** | **BITS ID No** | **Contribution** |
| 1 | D M Pravin Kumar | 2022DA04333 | 100% |
| 2 | Deepak Kumar |  |  |
| 3 | Manindra Kumar | 2022DA04310 | 100% |
| 4 | Manish Kumar | 2022DA04332 | 100% |

# Problem Statement

Store the names of people standing in the line using a Doubly Linked List where every node represents a single person.

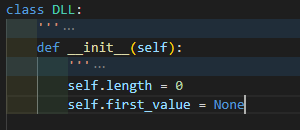
# Program

Element Class



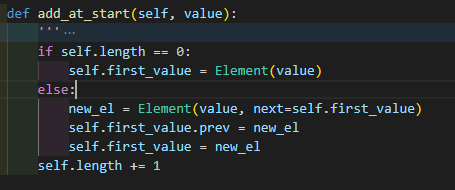
* \_\_init\_\_ runs when creating an object of type Element. The time complexity is not dependent on the size of the input. Hence creation of an Element is O(1).

DLL Class



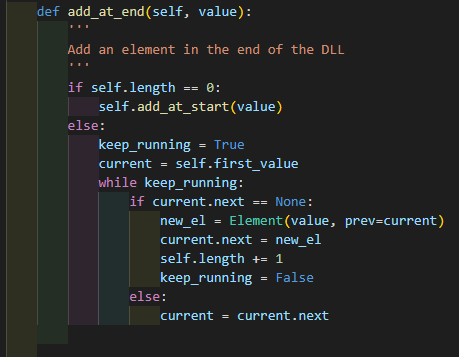
* \_\_init\_\_ runs when creating an object of type DLL. The time complexity is not dependent on the size of the input. Hence creation of an Element is O(1).

**Add\_at\_start(value)**



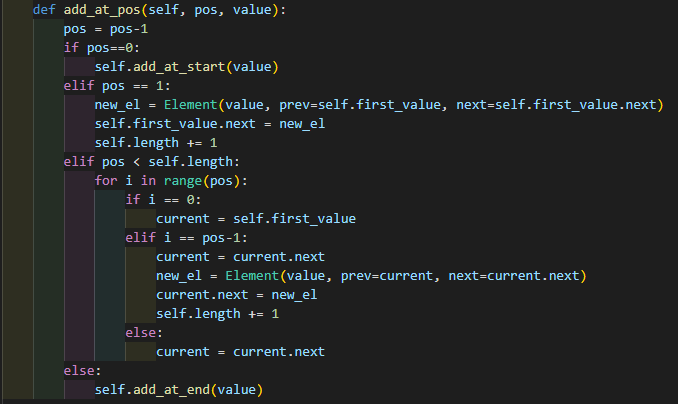
* This method has an if -else loop and runs to update the first value of the Doubly Linked list after checking if the DLL (Doubly Linked List) is empty or not. In the worst case (i.e., DLL not empty) it creates new Element and assigns the self.first\_value to the new element and the previous self.first\_value to the next of the current element.
* Time Complexity is a constant as this does not depend on the size of the DLL – O(1)

**Add\_at\_end(value)**



* This checks if the list is empty, if it is empty adds an element at start, Time complexity is O(1) as discussed above.
* If the list is not empty. DLL is parsed to reach the last element of the list and then the element is created and added to the end of the list.
* Time complexity for adding is constant but the time complexity for parsing is dependent on the size of the Doubly Linked List. Hence the worst case time complexity of this method is O(n) if n is the length of the Doubly Linked List.

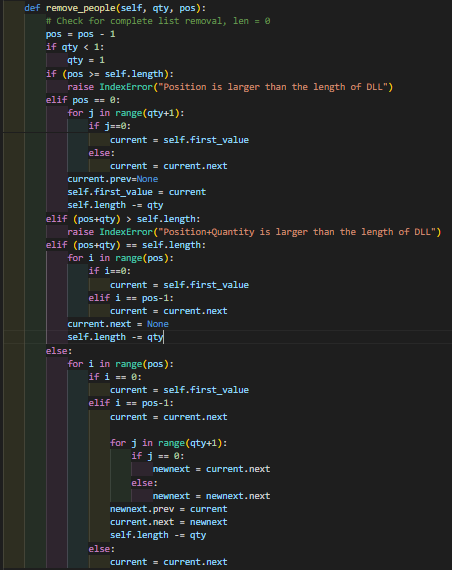
**Add\_at\_pos(pos, value)**



This method utilizes the previous methods (add\_at\_start, add\_at\_end) if the position specified is matching either of the conditions else, the loop is parsed and at the position where specified as an input argument , the element is added and the previous element’s next and the next element’s prev are modified to point at the new element.

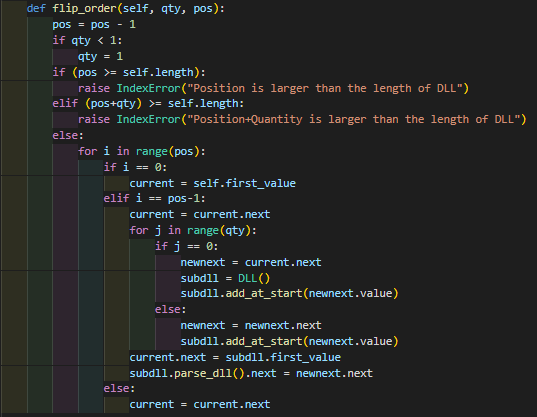
The worst case is when the position the element has to be added is at the end of the list, hence the time complexity for this method is also O(n) where n is the length of the DLL.

**remove\_people(qty, pos)**



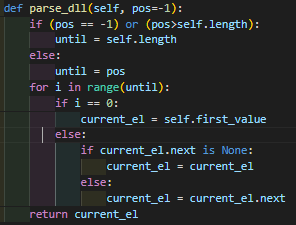
* In this method the worst case is when the input parameters pos and qty (representing position and quantity respectively) are maximum. i.e., pos + qty = length of the DLL (n). Hence the time complexity of the worst case for this method is to parse the entire list one element at a time and then remove elements from the list one at a time. Hence the time complexity is O(n).

**Flip\_order(qty, pos)**



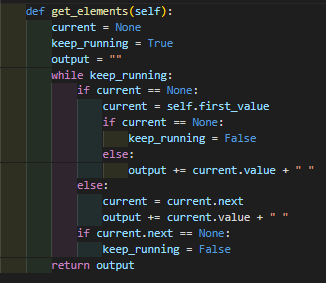
* In this method, the worst case of implementation is when the qty+pos is equal to the length of the DLL. In this case the DLL has to be parsed until the correct pos is reached, then a subDLL has to be created to included the flip elements by adding through add\_at\_start method. This flips the order of the qty that has to be flipped. The next pointer for pos-1 element has to be modified to pointed to the first\_value of the subDLL and the last element of the subDLL shall point to the next element after (pos+qty) is parsed.
* The worst case is when the pos+qty is equal to thelength of the list and the list has to be parsed once completely hence the time complexity of this method is O(n)

**Parse\_dll(pos)**



* This method is used to parse the DLL until a specified position of the DLL is reached and returns the element at the position provided by the user. The worst case is when the list has to be parsed until end of the DLL, hence the time complexity of this method is O(n)

**Get\_elements(dll):**



* This method parses the DLL and adds the current value to a string separated by spaces. The worst case complexity is proportional to the length of the DLL and hence the time complexity is O(N)

**Output\_text(inputfile, DLL)**



* This function implements reading of an input file and then parsing each line of the input file, splitting the line on the provided split(::) and then performing action on the DLL as per the method specified in each line. Hence for a file with n lines the lines are read (O(n) and then a for loop is used to parse and perform action on each line (n \* n for parsing and using a method.) Since the worst case method is O(n), parsing and performing an action will be O(n2). Ignoring the smaller order terms, the time complexity of the function is O(n2).

# Conclusion

The worst case time complexity for reading a fille creating and adding elements and other methods on a Doubly Linked List and output the file to disk is O(n2).

The following actions performed on a Doubly Linked List has the following time complexities.

|  |  |
| --- | --- |
| **Action** | **Worst Case Time Complexity** |
| Creating Element | O(1) |
| Creating Doubly Linked List | O(1) |
| Add element at start of DLL | O(1) |
| Add Element at end of DLL | O(n) |
| Add Element at a position of DLL | O(n) |
| Remove people from DLL (position , quantity specified) | O(n) |
| Flip order of the DLL starting from position and Quantity specified | O(n) |
| Parsing the DLL | O(n) |
| Get Elements of DLL | O(n) |

Reading an file and performing actions per each line of the file O(n).