



MALAD KANDIVALI EDUCATION SOCIETY'S  
NAGINDAS KHANDWALA COLLEGE OF COMMERCE, ARTS &  
MANAGEMENT STUDIES & SHANTABEN NAGINDASKHANDWALA  
COLLEGE OF SCIENCE  
MALAD [W], MUMBAI – 64  
AUTONOMOUS INSTITUTION  
(Affiliated To University Of Mumbai)  
Reaccredited 'A' Grade by NAAC | ISO 9001:2015 Certified

### CERTIFICATE

Name: Mr. Yug Kalubhai Patel

Roll No: 331

Programme: BSc IT

Semester: III

This is certified to be a bonafide record of practical works done by the above student in the college laboratory for the course **Data Structures (Course Code: 2032UISPR)** for the partial fulfilment of Third Semester of BSc IT during the academic year 2020-21.

The journal work is the original study work that has been duly approved in the year 2020-21 by the undersigned.

---

External Examiner

---

Mr. Gangashankar Singh  
(Subject-In-Charge)

Date of Examination: (College Stamp)

**Subject: Data Structures****INDEX**

Sr No	Date	Topic	Sign
1	04/09/2020	Implement the following for Array: a) Write a program to store the elements in 1-D array and provide an option to perform the operations like searching, sorting, merging, reversing the elements. b) Write a program to perform the Matrix addition, Multiplication and Transpose Operation.	
2	11/09/2020	Implement Linked List. Include options for insertion, deletion and search of a number, reverse the list and concatenate two linked lists.	
3	18/09/2020	Implement the following for Stack: a) Perform Stack operations using Array implementation. b) c) d) (i) using recursion, (ii) using iteration	
4	25/09/2020	Perform Queues operations using Circular Array implementation.	
5	01/10/2020	Write a program to search an element from a list. Give user the option to perform Linear or Binary search.	
6	09/10/2020	WAP to sort a list of elements. Give user the option to perform sorting using Insertion sort, Bubble sort or Selection sort.	
7	16/10/2020	Implement the following for Hashing: a) Write a program to implement the collision technique. b) Write a program to implement the concept of linear probing.	
8	23/10/2020	Write a program for inorder, postorder and preorder traversal of tree.	

## PRACTICAL 1 : Implement the following for Array

### Practical 1A

**Aim :** Write a program to store the elements in 1-D array and provide an option to perform the operations like searching, sorting, merging, reversing the elements.

**Theory :** Storing Data in Arrays. Assigning values to an element in an array is similar to assigning values to scalar variables. Simply reference an individual element of an array using the array name and the index inside parentheses, then use the assignment operator (=) followed by a value.

Following are the basic operations supported by an array.

Traverse – print all the array elements one by one.

Insertion – Adds an element at the given index.

Deletion – Deletes an element at the given index.

Search – Searches an element using the given index or by the value.

## Code :

By 1 contributor

```
64 lines (22 sloc) | 818 Bytes
1 class ArrayModification:
2
3
4
5     def linear_search(self,lst,n):
6
7         for i in range(len(lst)):
8
9             if lst[i] == n:
10
11                 return f'Position :{i}'
12
13         return -1
14
15
16
17     def insertion_sort(self,lst):
18
19         for i in range(len(lst)):
20
21
22             index = lst[i]
23
24
25
26             k = i - 1
27
28
29
30             while k >= 0 and lst[k] > index:
31
32                 lst[k + 1] = lst[k]
33
34                 k -= 1
35
36
37
38             lst[k+1] = index
39
40
41
42         return lst
43
44
45
46
47     def merge(self,lst1,lst2):
48
49         return ArrayModification.insertion_sort(lst1 + lst2)
50
51
52
53     def reverse(self,lst):
54
55         return lst[::-1]
56
57     lst = [2,9,1,7,3,5,2]
58
59     Arrmod = ArrayModification()
60
61     print(Arrmod.linear_search(lst,3))
62
63
64
```

## Output :

Position :4

[Program finished]

## Practical 1B

**Aim :** Write a program to perform the Matrix addition, Multiplication and Transpose Operation.

**Theory :** add() – add elements of two matrices.

subtract() – subtract elements of two matrices.

divide() – divide elements of two matrices.

multiply() – multiply elements of two matrices.

dot() – It performs matrix multiplication, does not element wise multiplication.

sqrt() – square root of each element of matrix.

sum(x, axis) – add to all the elements in matrix. Second argument is optional, it is used when we want to compute the column sum if axis is 0 and row sum if axis is 1.

“T” – It performs transpose of the specified matrix.

## Code :

By 1 contributor

58 lines (27 sloc) | 707 Bytes

Raw Blame

```
1 Mat1 = [[3, 4, -6],
2 [12,71,24],
3 [21,3,21]]
4
5 Mat2 = [[2, 16, -16],
6 [1,7,-3],
7 [-1,3,3]]
8
9 Mat3 = [[0,0,0,],
10 [0,0,0,],
11 [0,0,0,]]
12
13 # Matrix Addition
14
15 for i in range(len(Mat1)):
16
17     for j in range(len(Mat2[0])):
18
19         for k in range(len(Mat2)):
20
21             Mat3[i][j] += Mat1[i][k] + Mat2[k][j]
22
23
24
25 print(Mat3)
26
27 # Matrix Multiplication
28
29 Mat3 = [[0, 0, 0, 0],
30 [0, 0, 0, 0],
31 [0, 0, 0, 0],
32 [0, 0, 0, 0]]
33
34 for i in range(len(Mat1)):
35
36     for j in range(len(Mat2[0])):
37
38         for k in range(len(Mat2)):
39
40             Mat3[i][j] += Mat1[i][k] * Mat2[k][j]
41
42
43 print(Mat3)
44
45 #matrix transpose
46
47 for i in map(list, zip(*Mat1)):
48
49     print(i)
```

## Output :

```
[3, 27, -15], [109, 133, 91], [47, 71, 29]
[16, 58, -78, 0], [71, 761, -333, 0], [24, 420, -282, 0]
[3, 12, 21]
[4, 71, 3]
[-6, 24, 21]
[Program finished]
```

## PRACTICAL 2

**Aim :** Implement Linked List. Include options for insertion, deletion and search of a number, reverse the list and concatenate two linked lists.

**Theory :** A linked list is a sequence of data elements, which are connected together via links. Each data element contains a connection to another data element in form of a pointer. Python does not have linked lists in its standard library. We implement the concept of linked lists using the concept of nodes as discussed in the previous chapter. We have already seen how we create a node class and how to traverse the elements of a node. In this chapter we are going to study the types of linked lists known as singly linked lists. In this type of data structure there is only one link between any two data elements. We create such a list and create additional methods to insert, update and remove elements from the list.

### Insertion in a Linked List

Inserting element in the linked list involves reassigning the pointers from the existing nodes to the newly inserted node. Depending on whether the new data element is getting inserted at the beginning or at the middle or at the end of the linked list.

### Deleting an Item form a Linked List

We can remove an existing node using the key for that node. In the below program we locate the previous node of the node which is to be deleted. Then point the next pointer of this node to the next node of the node to be deleted.

### Searching in linked list

Searching is performed in order to find the location of a particular element in the list. Searching any element in the list needs traversing through the list and make the comparison of every element of the list with the specified element. If the element is matched with any of the list element then the location of the element is returned from the function.

## Reversing a Linked List

To reverse a `LinkedList` recursively we need to divide the `LinkedList` into two parts: `head` and `remaining`. `Head` points to the first element initially. `Remaining` points to the next element from the `head`. We traverse the `LinkedList` recursively until the second last element.

## Concatenating Linked Lists

Concatenate the two lists by traversing the first list until we reach it's a tail node and then point the next of the tail node to the head node of the second list. Store this concatenated list in the first list  
Following are the basic operations supported by an array.

Traverse – print all the array elements one by one.

Insertion – Adds an element at the given index.

Deletion – Deletes an element at the given index.

Search – Searches an element using the given index or by the value.

## Code :

```

256 lines (235 sloc) 7.44 kB
Raw Blame ⌂ ⌂

1 #Implement Linked List. Include options for insertion, deletion and search of a number, reverse the list and concatenate
2 #3005 (New Rollno:372) Shravan SYIT
3 class Node:
4     def __init__(self, element, next=None, prev=None):
5         self.element = element
6         self.next = next
7         self.prev = prev
8
9     def display(self):
10        print(self.element)
11
12 class LinkedList:
13     def __init__(self):
14         self.start = None
15         self.end = None
16         self.length = 0
17
18     def is_empty(self) -> bool:
19         return self.length == 0
20
21     def get_size(self) -> int:
22         return self.length
23
24     def display(self):
25         if self.is_empty():
26             print("Doubly Linked List is empty")
27             return
28         first = self.start
29         print("The List: ", end='')
30         print("[ " + first.element, end=' ')
31         first = first.next
32         while first:
33             print(", " + first.element, end=' ')
34             first = first.next
35         print("]")
36
37     def add_head(self, e):
38         old_head = self.start
39         self.start = Node(e)
40         self.start.next = old_head
41         if old_head is not None:
42             old_head.prev = self.start
43         if self.end is None:
44             self.end = self.start
45         self.length += 1
46
47     def get_head(self) -> Node:
48         return self.start
49
50     def remove_head(self):
51         if self.is_empty():
52             print("Doubly Linked List is empty")
53             return
54         elif self.length == 1:
55             self.start = None
56             self.end = None
57         else:
58             self.start = self.start.next
59             self.length -= 1
60
61     def add_tail(self, e):
62         new_value = Node(e)
63         if self.get_tail() is not None:
64             old_tail = self.end
65             self.end = new_value
66             self.end.prev = old_tail
67             old_tail.next = self.end
68             self.length += 1
69         else:
70             self.add_head(e)
71
72     def get_tail(self) -> Node:
73         return self.end
74
75     def remove_tail(self):
76         if self.is_empty():
77             print("Doubly Linked List is empty")
78             return
79         elif self.length == 1:
80             self.start = None
81             self.end = None
82         else:
83             self.end = self.end.prev
84             self.length -= 1
85
86     def add_between_list(self, index, element):
87         if index > self.length or index < 0:
88             print("Index out of bounds")
89         elif index == self.length:
90             self.add_tail(element)
91         elif index == 0:
92             self.add_head(element)

```

```

    97     if index == 0:
    98         self.add_head(element)
    99     else:
   100         prev_node = self.get_node_at(index - 1)
   101         current_node = self.get_node_at(index)
   102         new_value = Node(element)
   103         prev_node.next = new_value
   104         new_value.next = current_node
   105         new_value.prev = prev_node
   106         current_node.prev = new_value
   107         self.length += 1
   108
   109     def get_node_at(self, index, direction="auto") -> Node:
   110         if index < 0 or index >= self.length:
   111             print("Index out of bounds")
   112         else:
   113             element_node = self.start
   114             counter = 0
   115             while counter < index:
   116                 element_node = element_node.next
   117                 counter += 1
   118             return element_node
   119
   120     def remove_between_list(self, index):
   121         if index < 0 or index >= self.length:
   122             print("Index out of bounds")
   123         elif index == self.length - 1:
   124             self.remove_tail()
   125         elif index == 0:
   126             self.remove_head()
   127         else:
   128             prev_node = self.get_node_at(index - 1)
   129             next_node = self.get_node_at(index + 1)
   130             prev_node.next = next_node
   131             next_node.prev = prev_node
   132             self.length -= 1
   133
   134     def merge(self, linked_list_value):
   135         if not self.is_empty():
   136             last_node = self.get_tail()
   137             last_node.next = linked_list_value.start
   138             if not linked_list_value.is_empty():
   139                 linked_list_value.start.prev = last_node
   140                 self.end = linked_list_value.end
   141                 self.length = self.length + linked_list_value.length
   142             else:
   143                 self.start = linked_list_value.start
   144                 self.end = linked_list_value.end
   145                 self.length = linked_list_value.length
   146
   147     def reverse(self):
   148         if self.is_empty():
   149             print("Doubly Linked List is empty")
   150             return
   151         elif self.length == 1:
   152             return
   153         elif self.length == 2:
   154             temp = self.start
   155             self.start = self.end
   156             self.end = temp
   157             self.start.prev = None
   158             self.start.next = self.end
   159             self.end.prev = self.start
   160             self.end.next = None
   161         elif self.length == 3:
   162             mid = self.__get_node_from_start(1)
   163             temp = self.start
   164             self.start = self.end
   165             self.end = temp
   166             self.start.prev = None
   167             self.start.next = mid
   168             mid.prev = self.start
   169             mid.next = self.end
   170             self.end.prev = mid
   171             self.end.next = None
   172         elif self.length > 3:
   173             temp_lst = []
   174             first = self.start
   175             temp_lst.append(first)
   176             first = first.next
   177             while first:
   178                 temp_lst.append(first)
   179                 first = first.next
   180             temp_lst.reverse()
   181             for i in range(0, len(temp_lst)):
   182                 if i == 0:
   183                     self.start = temp_lst[i]
   184                     self.start.prev = None
   185                     self.start.next = temp_lst[i + 1]
   186                 elif i == 1:
   187                     temp_lst[i].prev = self.start
   188                     temp_lst[i].next = temp_lst[i + 1]
   189                 elif i == len(temp_lst) - 1:
   190                     self.end = temp_lst[i]
   191                     self.end.prev = temp_lst[i - 1]
   192                     self.end.next = None
   193

```

```

158     first = self.start
159     temp_lst.append(first)
160     first = first.next
161     while first:
162         temp_lst.append(first)
163         first = first.next
164     temp_lst.reverse()
165     for i in range(0, len(temp_lst)):
166         if i == 0:
167             self.start = temp_lst[i]
168             self.start.prev = None
169             self.start.next = temp_lst[i + 1]
170         elif i == 1:
171             temp_lst[i].prev = self.start
172             temp_lst[i].next = temp_lst[i + 1]
173         elif i == len(temp_lst) - 1:
174             self.end = temp_lst[i]
175             self.end.prev = temp_lst[i - 1]
176             self.end.next = None
177         else:
178             temp_lst[i].prev = temp_lst[i - 1]
179             temp_lst[i].next = temp_lst[i + 1]
180     temp_lst[len(temp_lst) - 2].prev = temp_lst[len(temp_lst) - 2 - 1]
181     temp_lst[len(temp_lst) - 2].next = self.end
182
183
184 one = LinkedList()
185 print(one.is_empty())
186 one.add_between_list(0, "zero")
187 one.add_between_list(1, "one")
188 one.add_between_list(2, "two")
189 one.display()
190 print(one.is_empty())
191 one.remove_between_list(1)
192 one.display()
193 one.add_between_list(1, "one")
194 one.display()
195
196 two = LinkedList()
197 two.add_head("first_name")
198 two.add_tail("last_name")
199 print("Head: ", end="")
200 two.get_head().display()
201 print("Tail: ", end="")
202 two.get_tail().display()
203 two.remove_tail()
204 two.get_tail().display()
205 two.remove_head()
206 try:
207     two.get_head().display()
208 except AttributeError as e:
209     print(e, "\nNo head found")
210 two.display()
211
212 two = LinkedList()
213 two.add_between_list(0, "zero")
214 two.add_between_list(1, "one")
215 two.add_between_list(2, "two")
216 two.add_between_list(3, "three")
217 two.display()
218 two.get_node_at(3).display()
219 two.get_node_at(3, "start").display()
220 two.get_node_at(3, "end").display()
221
222 three = LinkedList()
223 three.add_between_list(0, "0")
224 three.add_between_list(1, "1")
225 three.add_between_list(2, "2")
226 three.add_between_list(3, "4")
227 three.display()
228 print("Head: ", end="")
229 three.get_head().display()
230 print("Tail: ", end="")
231 three.get_tail().display()
232 print("Head: ", end="")
233 three.get_head().display()
234 print("Tail: ", end="")
235 three.get_tail().display()
236 three.merge(three)
237 three.display()
238 print("Head: ", end="")
239 three.get_head().display()
240 print("Tail: ", end="")
241 three.get_tail().display()
242
243 two.display()
244 print("Head: ", end="")
245 two.get_head().display()
246 print("Tail: ", end="")
247 two.get_tail().display()
248 print("Size: (%s)" % two.get_size())
249 two.display()
250 two.reverse()
251 two.display()

```

## Output :

```
True
The List: [zero, one, two]
False
The List: [zero, two]
The List: [zero, one, two]
Head: first_name
Tail: last_name
first_name
'NoneType' object has no attribute 'display'
No head found
Doubly Linked List is empty
The List: [zero, one, two, three]
three
three
three
The List: [zero, one, two, three]
Head: zero
Tail: three
The List: [0, 1, 2, 4]
Head: 0
Tail: 4
The List: [zero, one, two, three, 0, 1, 2, 4]
Head: zero
Tail: 4
Size: 8
The List: [zero, one, two, three, 0, 1, 2, 4]
The List: [4, 2, 1, 0, three, two, one, zero]
```

[Program finished]

## PRACTICAL 3 : Implement the following for Array

### Practical 3a

**Aim :** Perform Stack operations using Array implementation.

**Theory :** Stacks is one of the earliest data structures defined in computer science. In simple words, Stack is a linear collection of items. It is a collection of objects that supports fast last-in, first-out (LIFO) semantics for insertion and deletion. It is an array or list structure of function calls and parameters used in modern computer programming and CPU architecture. Similar to a stack of plates at a restaurant, elements in a stack are added or removed from the top of the stack, in a “last in, first out” order. Unlike lists or arrays, random access is not allowed for the objects contained in the stack.

There are two types of operations in Stack-

Push– To add data into the stack.

Pop– To remove data from the stack

## Code :

By 1 contributor

55 lines (45 sloc) | 1.21 KB

```
1 #3005 (New Rollno:372) Shravan SYIT
2 #Implementation of the Tower Of Hanoi.
3 class Stack:
4
5     def __init__(self):
6         self.stack_arr = []
7
8     def push(self,value):
9         self.stack_arr.append(value)
10
11    def pop(self):
12        if len(self.stack_arr) == 0:
13            print('Stack is empty!')
14            return None
15        else:
16            self.stack_arr.pop()
17
18    def get_head(self):
19        if len(self.stack_arr) == 0:
20            print('Stack is empty!')
21            return None
22        else:
23            return self.stack_arr[-1]
24
25    def display(self):
26        if len(self.stack_arr) == 0:
27            print('Stack is empty!')
28            return None
29        else:
30            print(self.stack_arr)
31
32 A = Stack()
33 B = Stack()
34 C = Stack()
35 def Hanoi(n, fromrod,to,temp):
36     if n == 1:
37         fromrod.pop()
38         to.push('disk 1')
39         if to.display() != None:
40             print(to.display())
41
42     else:
43
44         Hanoi(n-1, fromrod, temp, to)
45         fromrod.pop()
46         to.push(f'disk {n}')
47         if to.display() != None:
48             print(to.display())
49         Hanoi(n-1, temp, to, fromrod)
50
51 n = int(input('Enter the number of the disk in rod A : '))
52 for i in range(n):
53     A.push(f'disk {i+1} ')
54
55 Hanoi(n, A, C, B)
```

## Output :

← TAB \_ :

4  
['1', '2', '5', '3']  
[Program finished]

## Practical 3b

**Aim :** Implement Tower of Hanoi.

**Theory :** We are given n disks and a series of rods, we need to transfer all the disks to the final rod under the given constraints–  
We can move only one disk at a time.  
Only the uppermost disk

**Code :**



The screenshot shows a code editor interface with the following details:

- Contributors:** 1 contributor
- Code Statistics:** 55 lines (45 sloc) | 1.21 KB
- Actions:** Raw, Blame, Edit, Delete

```

1 #3005 (New Rollno:372) Shravan SYIT
2 #Implementation of the Tower Of Hanoi.
3 class Stack:
4
5     def __init__(self):
6         self.stack_arr = []
7
8     def push(self,value):
9         self.stack_arr.append(value)
10
11    def pop(self):
12        if len(self.stack_arr) == 0:
13            print('Stack is empty!')
14            return None
15        else:
16            self.stack_arr.pop()
17
18    def get_head(self):
19        if len(self.stack_arr) == 0:
20            print('Stack is empty!')
21            return None
22        else:
23            return self.stack_arr[-1]
24
25    def display(self):
26        if len(self.stack_arr) == 0:
27            print('Stack is empty!')
28            return None
29        else:
30            print(self.stack_arr)
31
32 A = Stack()
33 B = Stack()
34 C = Stack()
35 def Hanoi(n, fromrod,to,temp):
36     if n == 1:
37         fromrod.pop()
38         to.push('disk 1')
39         if to.display() != None:
40             print(to.display())
41
42     else:
43
44         Hanoi(n-1, fromrod, temp, to)
45         fromrod.pop()
46         to.push(f'disk {n}')
47         if to.display() != None:
48             print(to.display())
49         Hanoi(n-1, temp, to, fromrod)
50
51 n = int(input('Enter the number of the disk in rod A : '))
52 for i in range(n):
53     A.push(f'disk {i+1} ')
54
55 Hanoi(n, A, C, B)

```

## Output :

```
Enter the number of the disk in rod A : 3
['disk 1']
['disk 2']
['disk 2', 'disk 1']
['disk 3']
['disk 1']
['disk 3', 'disk 2']
['disk 3', 'disk 2', 'disk 1']

[Program finished]
```

## Practical 3c

**Aim :** WAP to scan a polynomial using linked list and add two polynomials.

**Theory :** Polynomial is a mathematical expression that consists of variables and coefficients. for example  $x^2 - 4x + 7$

In the Polynomial linked list, the coefficients and exponents of the polynomial are defined as the data node of the list.

For adding two polynomials that are stored as a linked list. We need to add the coefficients of variables with the same power. In a linked list node contains 3 members, coefficient value link to the next node. a linked list that is used to store Polynomial looks like –

Polynomial :  $4x^7 + 12x^2 + 45$

## Code :

```

362 lines (330 sloc)  12.1 KB
#3003 (New Rollno:372) Shravani SYIT
class DoublyNode:
    def __init__(self, element, next=None, prev=None):
        self.element = element
        self.next = next
        self.prev = prev
    def display(self):
        print(self.element)

class DoublyLinkedList:
    def __init__(self):
        self.__head = None
        self.__tail = None
        self.__size = 0
    def is_empty(self) -> bool:
        return self.__size == 0
    def get_size(self) -> int:
        return self.__size
    def __display_backward(self):
        if self.is_empty():
            print("Doubly Linked List is empty")
            return
        last = self.__tail
        print("The List: ", end="")
        print("[" + last.element, end="")
        last = last.prev
        while last:
            print(", " + last.element, end="")
            last = last.prev
        print("]")
    def __display_forward(self):
        if self.is_empty():
            print("Doubly Linked List is empty")
            return
        first = self.__head
        print("The List: ", end="")
        print("[" + first.element, end="")
        first = first.next
        while first:
            print(", " + first.element, end="")
            first = first.next
        print("]")
    def display(self, direction="start"):
        if direction == "end":
            self.__display_backward()
        elif direction == "start":
            self.__display_forward()
        else:
            self.__display_forward()
    def add_head(self, e):
        old_head = self.__head
        self.__head = DoublyNode(e)
        self.__head.next = old_head
        if old_head is not None:
            old_head.prev = self.__head
        if self.__tail is None:
            self.__tail = self.__head
        self.__size += 1
    def get_head(self) -> DoublyNode:
        return self.__head
    def remove_head(self):
        if self.is_empty():
            print("Doubly Linked List is empty")
            return
        elif self.__size == 1:
            self.__head = None
            self.__tail = None
        else:
            self.__head = self.__head.next
            self.__size -= 1
    def add_tail(self, e):
        new_value = DoublyNode(e)
        if self.get_tail() is not None:
            old_tail = self.__tail
            self.__tail = new_value
            self.__tail.prev = old_tail
            old_tail.next = self.__tail
            self.__size += 1
        else:
            self.add_head(e)

```

```

    self.add_head(e)

12 def get_tail(self) -> DoublyNode:
13     return self.__tail

14 def remove_tail(self):
15     if self.is_empty():
16         print("Doubly Linked List is empty")
17         return
18     elif self.__size == 1:
19         self.__head = None
20         self.__tail = None
21     else:
22         self.__tail = self.__tail.prev
23         self.__size -= 1

24 def add_between_list(self, index, element):
25     if index > self.__size or index < 0:
26         print("Index out of bounds")
27     elif index == self.__size:
28         self.add_tail(element)
29     elif index == 0:
30         self.add_head(element)
31     else:
32         prev_node = self.get_node_at(index - 1)
33         current_node = self.get_node_at(index)
34         new_value = DoublyNode(element)
35         prev_node.next = new_value
36         new_value.next = current_node
37         new_value.prev = prev_node
38         current_node.prev = new_value
39         self.__size += 1

40 def __get_node_from_start(self, index):
41     element_node = self.__head
42     counter = 0
43     while counter < index:
44         element_node = element_node.next
45         counter += 1
46     return element_node

47 def __get_node_from_end(self, index):
48     element_node = self.__tail
49     counter = self.__size
50     while counter > index + 1:
51         element_node = element_node.prev
52         counter -= 1
53     return element_node

54 def get_node_at(self, index, direction="auto") -> DoublyNode:
55     if index < 0 or index >= self.__size:
56         print("Index out of bounds")
57     elif direction == "start":
58         return self.__get_node_from_start(index)
59     elif direction == "end":
60         return self.__get_node_from_end(index)
61     elif self.__size == 1 or index == self.__size // 2:
62         return self.__get_node_from_start(index)
63     elif index > self.__size // 2:
64         return self.__get_node_from_end(index)

65 def remove_between_list(self, index):
66     if index < 0 or index >= self.__size:
67         print("Index out of bounds")
68     elif index == self.__size - 1:
69         self.remove_tail()
70     elif index == 0:
71         self.remove_head()
72     else:
73         prev_node = self.get_node_at(index - 1)
74         next_node = self.get_node_at(index + 1)
75         prev_node.next = next_node
76         next_node.prev = prev_node
77         self.__size -= 1

78 def search(self, search_value) -> bool:
79     index = 0
80     while index < self.__size:
81         value = self.get_node_at(index)
82         if value.element == search_value:
83             print("Found value " + str(search_value) + " at location: " + str(index))
84             return True
85         index += 1
86     print("Couldn't find value " + str(search_value) + "!")
87     return False

88 def merge(self, linked_list_value):
89     if not self.is_empty():
90         last_node = self.get_tail()
91         last_node.next = linked_list_value.__head
92         if not linked_list_value.is_empty():
93             linked_list_value.__head.prev = last_node
94             self.__tail = linked_list_value.__tail
95             self.__size = self.__size + linked_list_value.__size
96     else:
97         self.__head = linked_list_value.__head
98         self.__tail = linked_list_value.__tail

```

```

143     elif direction == "start":
144         return self.__get_node_from_start(index)
145     elif direction == "end":
146         return self.__get_node_from_end(index)
147     elif self.__size == 1 or index <= self.__size // 2:
148         return self.__get_node_from_start(index)
149     elif index > self.__size // 2:
150         return self.__get_node_from_end(index)
151
152     def remove_between_list(self, index):
153         if index < 0 or index >= self.__size:
154             print("Index out of bounds")
155         elif index == self.__size - 1:
156             self.remove_tail()
157         elif index == 0:
158             self.remove_head()
159         else:
160             prev_node = self.get_node_at(index - 1)
161             next_node = self.get_node_at(index + 1)
162             prev_node.next = next_node
163             next_node.prev = prev_node
164             self.__size -= 1
165
166     def search(self, search_value) -> bool:
167         index = 0
168         while index < self.__size:
169             value = self.get_node_at(index)
170             if value.element == search_value:
171                 print("Found value '" + str(search_value) + "' at location: " + str(index))
172                 return True
173             index += 1
174         print("Couldn't find value '" + str(search_value) + "'")
175         return False
176
177     def merge(self, linked_list_value):
178         if not self.is_empty():
179             last_node = self.get_tail()
180             last_node.next = linked_list_value.__head
181             if not linked_list_value.is_empty():
182                 linked_list_value.__head.prev = last_node
183                 self.__tail = linked_list_value.__tail
184                 self.__size = self.__size + linked_list_value.__size
185             else:
186                 self.__head = linked_list_value.__head
187                 self.__tail = linked_list_value.__tail
188                 self.__size = linked_list_value.__size
189
190     def reverse(self):
191         if self.is_empty():
192             print("Doubly Linked List is empty")
193             return
194         elif self.__size == 1:
195             return
196         elif self.__size == 2:
197             temp = self.__head
198             self.__head = self.__tail
199             self.__tail = temp
200             self.__head.prev = None
201             self.__head.next = self.__tail
202             self.__tail.prev = self.__head
203             self.__tail.next = None
204         elif self.__size == 3:
205             mid = self.__get_node_from_start(1)
206             temp = self.__head
207             self.__head = self.__tail
208             self.__tail = temp
209             self.__head.prev = None
210             self.__head.next = mid
211             mid.prev = self.__head
212             mid.next = self.__tail
213             self.__tail.prev = mid
214             self.__tail.next = None
215         elif self.__size > 3:
216             temp_lst = []
217             first = self.__head
218             temp_lst.append(first)
219             first = first.next
220             while first:
221                 temp_lst.append(first)
222                 first = first.next
223             temp_lst.reverse()
224             for i in range(0, len(temp_lst)):
225                 if i == 0:
226                     self.__head = temp_lst[i]
227                     self.__head.prev = None
228                     self.__head.next = temp_lst[i + 1]
229                 elif i == 1:
230                     temp_lst[i].prev = self.__head
231                     temp_lst[i].next = temp_lst[i + 1]
232                 elif i == len(temp_lst) - 1:

```

```

232         elif i == len(temp_lst) - 1:
233             self.__tail = temp_lst[i]
234             self.__tail.prev = temp_lst[i - 1]
235             self.__tail.next = None
236         else:
237             temp_lst[i].prev = temp_lst[i - 1]
238             temp_lst[i].next = temp_lst[i + 1]
239             temp_lst[len(temp_lst) - 2].prev = temp_lst[len(temp_lst) - 2 - 1]
240             temp_lst[len(temp_lst) - 2].next = self.__tail
241
242     @staticmethod
243     def test_main():
244         dll = DoublyLinkedList()
245         print(dll.is_empty())
246         dll.add_between_list(0, "zero")
247         dll.add_between_list(1, "one")
248         dll.add_between_list(2, "two")
249         dll.display()
250         print(dll.is_empty())
251         print("After removing")
252         dll.remove_between_list(1)
253         dll.display()
254         print("After adding again")
255         dll.add_between_list(1, "one")
256         dll.display()
257
258         dll2 = DoublyLinkedList()
259         dll2.add_head("first_name")
260         dll2.add_tail("last_name")
261         print("Head: ", end="")
262         dll2.get_head().display()
263         print("Tail: ", end="")
264         dll2.get_tail().display()
265         dll2.remove_tail()
266         dll2.get_tail().display()
267         dll2.remove_head()
268         try:
269             dll2.get_head().display()
270         except AttributeError as e:
271             print(e, "\nNo head found")
272         dll2.display()
273
274         dll3 = DoublyLinkedList()
275         dll3.add_between_list(0, "zero")
276         dll3.add_between_list(1, "one")
277         dll3.add_between_list(2, "two")
278         dll3.add_between_list(3, "three")
279         dll3.display()
280         dll3.search("two")
281         dll3.search("four")
282         dll3.get_node_at(3).display()
283         dll3.get_node_at(3, "start").display()
284         dll3.get_node_at(3, "end").display()
285
286         dll4 = DoublyLinkedList()
287         dll4.add_between_list(0, "0")
288         dll4.add_between_list(1, "1")
289         dll4.add_between_list(2, "2")
290         dll4.add_between_list(3, "4")
291         print("The Two Lists: ")
292         dll3.display()
293         print("Head: ", end="")
294         dll3.get_head().display()
295         print("Tail: ", end="")
296         dll3.get_tail().display()
297         dll4.display()
298         print("Head: ", end="")
299         dll4.get_head().display()
300         print("Tail: ", end="")
301         dll4.get_tail().display()
302         dll3.merge(dll4)
303         print("After Merging")
304         dll3.display()
305         print("Head: ", end="")
306         dll3.get_head().display()
307         print("Tail: ", end="")
308         dll3.get_tail().display()
309         print(f"Size: {dll3.get_size()}")
310         dll3.display()
311         dll3.reverse()
312         print("After Reversing")
313         dll3.display()
314         print("Reverse Display")
315         dll3.display("end")
316
317
318 if __name__ == "__main__":
319     polynomial_addition_1 = DoublyLinkedList()
320     order = int(input('Enter the order of the polynomials : '))
321     print("Polynomial 1:")
322     polynomial_addition_1.add_head(int(input("Enter coefficient of power {order} : ")))
323     for i in reversed(range(order)):
324         polynomial_addition_1.add_tail(int(input("Enter coefficient of power {i} : ")))
325     polynomial_addition_2 = DoublyLinkedList()
326     print("Polynomial 2:")
327     polynomial_addition_2.add_head(int(input("Enter coefficient of power {order} : ")))
328     for i in reversed(range(order)):
329         polynomial_addition_2.add_tail(int(input("Enter coefficient of power {i} : ")))

```

```

224     dll3 = DoublyLinkedList()
225     dll3.add_between_list(0, "zero")
226     dll3.add_between_list(1, "one")
227     dll3.add_between_list(2, "two")
228     dll3.add_between_list(3, "three")
229     dll3.display()
230     dll3.search("two")
231     dll3.search("four")
232     dll3.get_node_at(3).display()
233     dll3.get_node_at(3, "start").display()
234     dll3.get_node_at(3, "end").display()
235
236     dll4 = DoublyLinkedList()
237     dll4.add_between_list(0, "0")
238     dll4.add_between_list(1, "1")
239     dll4.add_between_list(2, "2")
240     dll4.add_between_list(3, "4")
241     print("The Two Lists: ")
242     dll3.display()
243     print("Head: ", end='')
244     dll3.get_head().display()
245     print("Tail: ", end='')
246     dll3.get_tail().display()
247     dll4.display()
248     print("Head: ", end='')
249     dll4.get_head().display()
250     print("Tail: ", end='')
251     dll4.get_tail().display()
252     dll3.merge(dll4)
253     print("After Merging")
254     dll3.display()
255     print("Head: ", end='')
256     dll3.get_head().display()
257     print("Tail: ", end='')
258     dll3.get_tail().display()
259     print(f"Size: {dll3.get_size()}")
260     dll3.display()
261     dll3.reverse()
262     print("After Reversing")
263     dll3.display()
264     print("Reverse Display")
265     dll3.display("end")
266
267
268 if __name__ == "__main__":
269     polynomial_addition_1 = DoublyLinkedList()
270     order = int(input('Enter the order of the polynomials : '))
271     print("Polynomial 1:")
272     polynomial_addition_1.add_head(int(input(f"Enter coefficient of power {order} : ")))
273     for i in reversed(range(order)):
274         polynomial_addition_1.add_tail(int(input(f"Enter coefficient of power {i} : ")))
275     polynomial_addition_2 = DoublyLinkedList()
276     print("Polynomial 2:")
277     polynomial_addition_2.add_head(int(input(f"Enter coefficient of power {order} : ")))
278     for i in reversed(range(order)):
279         polynomial_addition_2.add_tail(int(input(f"Enter coefficient of power {i} : ")))
280     superscript_map = {"0": "⁰", "1": "¹", "2": "²", "3": "³", "4": "⁴", "5": "⁵", "6": "⁶",
281                     "7": "⁷", "8": "⁸", "9": "⁹"}
282     trans = str.maketrans(''.join(superscript_map.keys()), ''.join(superscript_map.values()))
283     addend_1 = []
284     addend_2 = []
285     addend_str_1 = ""
286     addend_str_2 = ""
287     result = []
288     result_str = ""
289     for i in reversed(range(order + 1)):
290         addend_1.append(polynomial_addition_1.get_node_at(i).element)
291         addend_2.append(polynomial_addition_2.get_node_at(i).element)
292         result.append(polynomial_addition_1.get_node_at(i).element + polynomial_addition_2.get_node_at(i).element)
293     for ele in reversed(range(order + 1)):
294         addend_str_1 = addend_str_1 + "%d" % (addend_1[ele]) + "x" + (str(ele).translate(trans))
295         addend_str_2 = addend_str_2 + "%d" % (addend_2[ele]) + "x" + (str(ele).translate(trans))
296         result_str = result_str + "%d" % (result[ele]) + "x" + (str(ele).translate(trans))
297
298     addend_str_1 = addend_str_1.replace("*", "")
299     addend_str_1 = addend_str_1.lstrip("+")
300     addend_str_1 = addend_str_1.rstrip("x⁰")
301
302     sign = addend_str_2[0]
303     addend_str_2 = addend_str_2.replace("*", "")
304     addend_str_2 = addend_str_2.lstrip("+")
305     addend_str_2 = addend_str_2.rstrip("-")
306     addend_str_2 = addend_str_2.rstrip("x⁰")
307
308     result_str = result_str.replace("=", "")
309     result_str = result_str.lstrip("+")
310     result_str = result_str.rstrip("x⁰")
311
312     print(addend_str_1 + "+sign+" + addend_str_2 + "=" + result_str)

```

## Output :

---

```
Enter the order of the polynomials : 1
Polynomial 1:
Enter coefficient of power 1 : 1
Enter coefficient of power 0 : 0
Polynomial 2:
Enter coefficient of power 1 : 2
Enter coefficient of power 0 : 0
1x+0 + 2x+0 = 3x+0
```

[Program finished]

## Practical 3d

**Aim :** WAP to calculate factorial and to compute the factors of a given no.

- (i) using recursion, (ii) using iteration

### Theory :

The factorial of a number is the product of all the integers from 1 to that number.

For example, the factorial of 6 is  $1*2*3*4*5*6 = 720$ . Factorial is not defined for negative numbers and the factorial of zero is one,  $0! = 1$ .

### Recursion

In Python, we know that a function can call other functions. It is even possible for the function to call itself. These types of construct are termed as recursive functions.

### Iteration

Repeating identical or similar tasks without making errors is something that computers do well and people do poorly. Repeated execution of a set of statements is called iteration.

Because iteration is so common, Python provides several language features to make it easier.

## Code :

By 1 contributor

56 lines (50 sloc) | 1.72 KB

```
1 #3005 (New Rollno:372) Shravan SYIT
2 class Fact:
3     def __init__(self, number):
4         self.number = number
5
6     def factorial_iteration(self):
7         factorial = 1
8         if self.number < 0:
9             raise Exception("Factorial of Negative number doesn't exist")
10        elif self.number == 0:
11            return factorial
12        else:
13            for i in range(1, self.number + 1):
14                factorial = factorial * i
15        return factorial
16
17    def factorial_recursion(self, number=None):
18        if number is None:
19            number = self.number
20        if number < 0:
21            raise Exception("Factorial of Negative number doesn't exist")
22        elif number == 0 or number == 1:
23            return 1
24        else:
25            return number * self.factorial_recursion(number - 1)
26
27    def factors_iteration(self):
28        factors = []
29        for i in range(1, self.number + 1):
30            if self.number % i == 0:
31                factors.append(i)
32        return factors
33
34    def factors_recursion(self, number=None, lst=None):
35        factorial = self.number
36        if number is None and lst is None:
37            number = self.number
38            lst = []
39        if number == 0 or number < 0:
40            return lst
41        elif number == 1:
42            lst.append(1)
43            return sorted(lst)
44        elif factorial % number == 0:
45            lst.append(number)
46            return self.factors_recursion(number - 1, lst)
47        else:
48            return self.factors_recursion(number - 1, lst)
49
50
51 if __name__ == '__main__':
52     fact = Fact(int(input("Enter a number: ")))
53     print(fact.factorial_iteration())
54     print(fact.factorial_recursion())
55     print(fact.factors_iteration())
56     print(fact.factors_recursion())
```

## Output :

---

```
Enter a number: 1
```

```
1
1
[1]
[1]
```

```
[Program finished]
```

## PRACTICAL 4

**Aim :** Perform Queues operations using Circular Array implementation.

**Theory :** Circular queue avoids the wastage of space in a regular queue implementation using arrays.

Circular Queue works by the process of circular increment i.e. when we try to increment the pointer and we reach the end of the queue, we start from the beginning of the queue.

Here, the circular increment is performed by modulo division with the queue size. That is,

if REAR + 1 == 5 (overflow!), REAR = (REAR + 1)%5 = 0 (start of queue)

The circular queue work as follows:

two pointers FRONT and REAR

FRONT track the first element of the queue

REAR track the last elements of the queue

initially, set value of FRONT and REAR to -1

### 1. Enqueue Operation

check if the queue is full

for the first element, set value of FRONT to 0

circularly increase the REAR index by 1 (i.e. if the rear reaches the end, next it would be at the start of the queue)

add the new element in the position pointed to by REAR

### 2. Dequeue Operation

check if the queue is empty

return the value pointed by FRONT

circularly increase the FRONT index by 1

for the last element, reset the values of FRONT and REAR to -1

## Code :

```

131 lines (107 sloc) | 3.54 KB
1 #3005 (New Rollno:372) Shravan SYIT
2 class ArrayQueue:
3     default_capacity = 10
4
5     def __init__(self):
6
7         self._data = [None] * ArrayQueue.default_capacity
8         self._size = 0
9         self._front = 0
10
11    def len(self):
12
13        return self._size
14
15    def is_empty(self):
16
17        return self._size == 0
18
19    def first(self):
20
21        if self.is_empty():
22            raise Exception("Queue is empty")
23        return self._data[self._front]
24
25    def dequeue(self):
26
27        if self.is_empty():
28            raise Exception("Queue is empty")
29        answer = self._data[self._front]
30        self._data[self._front] = None
31        self._front = (self._front + 1) % len(self._data)
32        self._size -= 1
33        return answer
34
35    def enqueue(self, e):
36
37        if self._size == len(self._data):
38            self._resize(2 * len(self._data))
39        avail = (self._front + self._size) % len(self._data)
40        self._data[avail] = e
41        self._size += 1
42
43    def _resize(self, cap):
44
45        old = self._data
46        self._data = [None] * cap
47        walk = self._front
48        for k in range(self._size):
49            self._data[k] = old[walk]
50            walk = (1 + walk) % len(old)
51        self._front = 0
52
53    class DEQueue(ArrayQueue):
54        def __init__(self):
55            ArrayQueue.__init__(self)
56            self._back = 0
57
58        def last(self):
59            if self.is_empty():
60                raise Exception("Queue is empty")
61            return self._data[self._back]
62
63        def enqueue_front(self, e):
64            self.enqueue(e)
65            self._back = (self._front + self._size - 1) % len(self._data)
66
67        def enqueue_back(self, e):
68            if self._size == len(self._data):
69                self._resize(2 * len(self._data))
70            self._front = (self._front - 1) % len(self._data)
71            self._data[self._front] = e
72            self._size += 1
73            self._back = (self._front + self._size - 1) % len(self._data)
74
75        def dequeue_front(self):
76            if self.is_empty():
77                raise Exception("Queue is empty")
78            back = (self._front + self._size - 1) % len(self._data)
79            answer = self._data[back]
80            self._data[back] = None
81            self._size -= 1
82            self._back = (self._front + self._size - 1) % len(self._data)
83            return answer
84
85        def dequeue_back(self):
86            answer = self.dequeue()
87            self._back = (self._front + self._size - 1) % len(self._data)
88            return answer
89
90        def _resize(self, cap):
91            ArrayQueue._resize(self, cap)
92            self._back = (self._front + self._size - 1) % len(self._data)

```

```

44         old = self._data
45         self._data = [None] * cap
46         walk = self._front
47         for k in range(self._size):
48             self._data[k] = old[walk]
49             walk = (1 + walk) % len(old)
50         self._front = 0
51
52
53     class DEQueue(ArrayQueue):
54         def __init__(self):
55             ArrayQueue.__init__(self)
56             self._back = 0
57
58         def last(self):
59             if self.is_empty():
60                 raise Exception('Queue is empty')
61             return self._data[self._back]
62
63         def enqueue_front(self, e):
64             self.enqueue(e)
65             self._back = (self._front + self._size - 1) % len(self._data)
66
67         def enqueue_back(self, e):
68             if self._size == len(self._data):
69                 self._resize(2 * len(self._data))
70             self._front = (self._front - 1) % len(self._data)
71             self._data[self._front] = e
72             self._size += 1
73             self._back = (self._front + self._size - 1) % len(self._data)
74
75         def dequeue_front(self):
76             if self.is_empty():
77                 raise Exception('Queue is empty')
78             back = (self._front + self._size - 1) % len(self._data)
79             answer = self._data[back]
80             self._data[back] = None
81             self._size -= 1
82             self._back = (self._front + self._size - 1) % len(self._data)
83             return answer
84
85         def dequeue_back(self):
86             answer = self.dequeue()
87             self._back = (self._front + self._size - 1) % len(self._data)
88             return answer
89
90         def _resize(self, cap):
91             ArrayQueue._resize(self, cap)
92             self._back = (self._front + self._size - 1) % len(self._data)
93
94         def show_all(self):
95             print(f"Size: {self._size}", end=', ')
96             print(f"Front: {self._front}", end=', ')
97             print(f"Back: {self._back}", end=', ')
98             try:
99                 print("First: ", self.first(), end=', ')
100                print("Last: ", self.last(), end='\n')
101            except:
102                print("Queue is empty")
103                print(f"Queue: {self._data}")
104
105
106    if __name__ == "__main__":
107        deq = DEQueue()
108        for i in range(1, 4):
109            deq.enqueue_front("f_" + str(i))
110            deq.show_all()
111        for i in range(1, 4):
112            deq.dequeue_front()
113            deq.show_all()
114        for i in range(1, 4):
115            deq.enqueue_back("b_" + str(i))
116            deq.show_all()
117        for i in range(1, 4):
118            deq.dequeue_back()
119            deq.show_all()
120        for i in range(1, 13):
121            deq.enqueue_front("f_" + str(i))
122            deq.show_all()
123        for i in range(1, 11):
124            deq.enqueue_back("b_" + str(i))
125            deq.show_all()
126        for i in range(1, 13):
127            deq.dequeue_front()
128            deq.show_all()
129        for i in range(1, 11):
130            deq.dequeue_back()
131            deq.show_all()

```

**Output :**


---

```

Size: 1, Front: 0, Back: 0, First: f_1, Last: f_1,
Queue: ['f_1', None, None, None, None, None, None, None, None]
]
Size: 2, Front: 0, Back: 1, First: f_1, Last: f_2,
Queue: ['f_1', 'f_2', None, None, None, None, None, None, Non
e]
Size: 3, Front: 0, Back: 2, First: f_1, Last: f_3,
Queue: ['f_1', 'f_2', 'f_3', None, None, None, None, None, No
ne]
Size: 2, Front: 0, Back: 1, First: f_1, Last: f_2,
Queue: ['f_1', 'f_2', None, None, None, None, None, None, Non
e]
Size: 1, Front: 0, Back: 0, First: f_1, Last: f_1,
Queue: ['f_1', None, None, None, None, None, None, None, None
]
Size: 0, Front: 0, Back: 9, Queue is empty
Queue: [None, None, None, None, None, None, None, None, None]
Size: 1, Front: 9, Back: 9, First: b_1, Last: b_1,
Queue: [None, None, None, None, None, None, None, None, 'b_1'
]
Size: 2, Front: 8, Back: 9, First: b_2, Last: b_1,
Queue: [None, None, None, None, None, None, None, None, 'b_2
', 'b_1
']
Size: 3, Front: 7, Back: 9, First: b_3, Last: b_1,
Queue: [None, None, None, None, None, None, None, 'b_3', 'b_2
', 'b_1
']
Size: 2, Front: 8, Back: 9, First: b_2, Last: b_1,
Queue: [None, None, None, None, None, None, None, None, 'b_2
', 'b_1
']
Size: 1, Front: 9, Back: 9, First: b_1, Last: b_1,
Queue: [None, None, None, None, None, None, None, None, 'b_1
']
Size: 0, Front: 0, Back: 9, Queue is empty
Queue: [None, None, None, None, None, None, None, None, None]
Size: 1, Front: 0, Back: 0, First: f_1, Last: f_1,
Queue: ['f_1', None, None, None, None, None, None, None, None
]
Size: 2, Front: 0, Back: 1, First: f_1, Last: f_2,
Queue: ['f_1', 'f_2', None, None, None, None, None, None, Non
e]
Size: 3, Front: 0, Back: 2, First: f_1, Last: f_3,
Queue: ['f_1', 'f_2', 'f_3', None, None, None, None, None, No
ne]
Size: 4, Front: 0, Back: 3, First: f_1, Last: f_4,
Queue: ['f_1', 'f_2', 'f_3', 'f_4', None, None, None, None, N
one]
Size: 5, Front: 0, Back: 4, First: f_1, Last: f_5,
Queue: ['f_1', 'f_2', 'f_3', 'f_4', 'f_5', None, None, None, N
one]
Size: 6, Front: 0, Back: 5, First: f_1, Last: f_6,
Queue: ['f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', None, None, N
one]
Size: 7, Front: 0, Back: 6, First: f_1, Last: f_7,
Queue: ['f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', None, N
one]
Size: 8, Front: 0, Back: 7, First: f_1, Last: f_8,
Queue: ['f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', Non
e, None]
Size: 9, Front: 0, Back: 8, First: f_1, Last: f_9,
Queue: ['f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_
9', None]

```

---

---

```

9', None]
Size: 10, Front: 0, Back: 9, First: f_1, Last: f_10,
Queue: ['f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_
9', 'f_10']
Size: 11, Front: 0, Back: 10, First: f_1, Last: f_11,
Queue: ['f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_
9', 'f_10', 'f_11', None, None, None, None, None, None, None, None,
None]
Size: 12, Front: 0, Back: 11, First: f_1, Last: f_12,
Queue: ['f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_
9', 'f_10', 'f_11', 'f_12', None, None, None, None, None, None, Non
e, None]
Size: 13, Front: 19, Back: 11, First: b_1, Last: f_12,
Queue: ['f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_
9', 'f_10', 'f_11', 'f_12', None, None, None, None, None, None, Non
e, 'b_1']
Size: 14, Front: 18, Back: 11, First: b_2, Last: f_12,
Queue: ['f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_
9', 'f_10', 'f_11', 'f_12', None, None, None, None, None, None, Non
e, 'b_2', 'b_1']
Size: 15, Front: 17, Back: 11, First: b_3, Last: f_12,
Queue: ['f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_
9', 'f_10', 'f_11', 'f_12', None, None, None, None, None, 'b_3', 'b
_2', 'b_1']
Size: 16, Front: 16, Back: 11, First: b_4, Last: f_12,
Queue: ['f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_
9', 'f_10', 'f_11', 'f_12', None, None, None, 'b_4', 'b_3', 'b
_2', 'b_1']
Size: 17, Front: 15, Back: 11, First: b_5, Last: f_12,
Queue: ['f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_
9', 'f_10', 'f_11', 'f_12', None, None, None, 'b_5', 'b_4', 'b_3',
'b_2', 'b_1']
Size: 18, Front: 14, Back: 11, First: b_6, Last: f_12,
Queue: ['f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_
9', 'f_10', 'f_11', 'f_12', None, None, 'b_6', 'b_5', 'b_4', 'b_3',
'b_2', 'b_1']
Size: 19, Front: 13, Back: 11, First: b_7, Last: f_12,
Queue: ['f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_
9', 'f_10', 'f_11', 'f_12', None, 'b_7', 'b_6', 'b_5', 'b_4', 'b_3',
'b_2', 'b_1']
Size: 20, Front: 12, Back: 11, First: b_8, Last: f_12,
Queue: ['f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_
9', 'f_10', 'f_11', 'f_12', 'b_8', 'b_7', 'b_6', 'b_5', 'b_4', 'b_3
', 'b_2', 'b_1']
Size: 21, Front: 39, Back: 19, First: b_9, Last: f_12,
Queue: ['b_8', 'b_7', 'b_6', 'b_5', 'b_4', 'b_3', 'b_2', 'b_1', 'f_
1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_9', 'f_10',
'f_11', 'f_12', None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, 'b_9']
Size: 22, Front: 38, Back: 19, First: b_10, Last: f_12,
Queue: ['b_8', 'b_7', 'b_6', 'b_5', 'b_4', 'b_3', 'b_2', 'b_1', 'f_
1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_9', 'f_10',
'f_11', 'f_12', None, None, None, None, None, None, None, None, Non
e, None, None, None, None, None, None, None, None, 'b_10', 'b_9']
Size: 21, Front: 38, Back: 18, First: b_10, Last: f_11,
Queue: ['b_8', 'b_7', 'b_6', 'b_5', 'b_4', 'b_3', 'b_2', 'b_1', 'f_
1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_9', 'f_10',
'f_11', None, None, None, None, None, None, None, None, None, Non
e, None, None, None, None, None, None, None, 'b_10', 'b_

```

[Program finished] █

## Practical 5

**Aim :** Write a program to search an element from a list. Give user the option to perform Linear or Binary search.

**Theory :** Linear Search:

This linear search is a basic search algorithm which searches all the elements in the list and finds the required value. ... This is also known as sequential search.

Binary Search:

In computer science, a binary search or half-interval search algorithm finds the position of a target value within a sorted array. The binary search algorithm can be classified as a dichotomies divide-and-conquer search algorithm and executes in logarithmic time.

## Code :

1 contributor

38 lines (34 sloc) | 1.13 KB

Raw Blame

```
1 #to search an element from a list and give user the option to perform Linear or Binary search.
2 #3005 (New Rollno:372) Shravan SYIT
3 class Search:
4     def __init__(self, lst, ele):
5         self.lst = lst
6         self.ele = ele
7
8     def binary_search(self):
9         sorted_lst = sorted(self.lst)
10        start = 0
11        end = len(sorted_lst) - 1
12        while start <= end:
13            mid = (end + start) // 2
14            if sorted_lst[mid] < self.ele:
15                start = mid + 1
16            elif sorted_lst[mid] > self.ele:
17                end = mid - 1
18            else:
19                return mid
20        return False
21
22    def linear_search(self):
23        for i in range(len(self.lst)):
24            if self.lst[i] == self.ele:
25                return i
26        return False
27
28
29 if __name__ == '__main__':
30     test_list = [5,31,44,94,108,109]
31     test_value_1 = 42
32     test_value_2 = 44
33     test_search = Search(test_list, test_value_1)
34     print(test_search.binary_search())
35     print(test_search.linear_search())
36     test_search_2 = Search(test_list, test_value_2)
37     print(test_search_2.binary_search())
38     print(test_search_2.linear_search())
```

## Output :

```
False
False
2
2
[Program finished]
```

## Practical 6

**Aim :** WAP to sort a list of elements. Give user the option to perform sorting using Insertion sort, Bubble sort or Selection sort.

**Theory : Sort:**

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Selection Sort:

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array

Insertion Sort:

Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

## Code :

1 contributor

42 lines (37 sloc) | 1.46 KB

Raw Blame

```
1 # to sort a list of elements. Give user the option to perform sorting using Insertion sort, Bubble sort or Selection sort.
2 #3005 (New Rollno:372) Shravan SYIT
3 class Sort:
4     def __init__(self, lst):
5         self.lst = lst
6
7     def bubble_sort(self):
8         sorted_lst = self.lst.copy()
9         for i in range(len(sorted_lst) - 1):
10             for j in range(0, len(sorted_lst) - i - 1):
11                 if sorted_lst[j] > sorted_lst[j + 1]:
12                     sorted_lst[j], sorted_lst[j + 1] = sorted_lst[j + 1], sorted_lst[j]
13         return sorted_lst
14
15     def selection_sort(self):
16         sorted_lst = self.lst.copy()
17         for i in range(len(sorted_lst)):
18             min_idx = i
19             for j in range(i + 1, len(sorted_lst)):
20                 if sorted_lst[min_idx] > sorted_lst[j]:
21                     min_idx = j
22             sorted_lst[i], sorted_lst[min_idx] = sorted_lst[min_idx], sorted_lst[i]
23         return sorted_lst
24
25     def insertion_sort(self):
26         sorted_lst = self.lst.copy()
27         for i in range(1, len(sorted_lst)):
28             key = sorted_lst[i]
29             j = i - 1
30             while j >= 0 and key < sorted_lst[j]:
31                 sorted_lst[j + 1] = sorted_lst[j]
32                 j -= 1
33             sorted_lst[j + 1] = key
34         return sorted_lst
35
36
37 if __name__ == '__main__':
38     test_list = [44,31,108,94,109]
39     test_sort = Sort(test_list)
40     print(test_sort.bubble_sort())
41     print(test_sort.selection_sort())
42     print(test_sort.insertion_sort())
```

## Output :

```
Before : [None, None, None, None]
Collision detected
Collision detected
After: [None, 1, None, 23]
```

```
[Program finished]
```

## Practical 7 : Implement the following for Hashing

**Hashing :**

Hashing is an important Data Structure which is designed to use a special function called the Hash function which is used to map a given value with a particular key for faster access of elements. The efficiency of mapping depends of the efficiency of the hash function used.

### Practical 7a

**Aim :** Write a program to implement the collision technique.

**Theory :**

**Collisions:**

A Hash Collision Attack is an attempt to find two input strings of a hash function that produce the same hash result. If two separate inputs produce the same hash output, it is called a collision.

**Collision Techniques:**

**Separate Chaining:**

The idea is to make each cell of hash table point to a linked list of records that have same hash function value.

**Open Addressing:**

Like separate chaining, open addressing is a method for handling collisions. In Open Addressing, all elements are stored in the hash table itself. So at any point, the size of the table must be greater than or equal to the total number of keys (Note that we can increase table size by copying old data if needed).

## Code :

1 contributor

28 lines (23 sloc) | 990 Bytes

Raw Blame ⌂

```
1 #3005 (New Rollno:372) Shravan SYIT
2 #Implement the following Hashing Techniques.
3 #Write a program to implement the collision technique.
4
5 class Hash:
6     def __init__(self, keys, lowerrange, higherrange):
7         self.value = self.hashfunction(keys,lowerrange, higherrange)
8
9     def get_key_value(self):
10        return self.value
11
12    def hashfunction(self,keys,lowerrange, higherrange):
13        if lowerrange == 0 and higherrange > 0:
14            return keys%(higherrange)
15
16 if __name__ == '__main__':
17     list_of_keys = [23,43,1,87]
18     list_of_list_index = [None,None,None,None]
19     print("Before : " + str(list_of_list_index))
20     for value in list_of_keys:
21         #print(Hash(value,0,len(list_of_keys)).get_key_value())
22         list_index = Hash(value,0,len(list_of_keys)).get_key_value()
23         if list_of_list_index[list_index]:
24             print("Collision detected")
25         else:
26             list_of_list_index[list_index] = value
27
28     print("After: " + str(list_of_list_index))
```

## Output :

```
Before : [None, None, None, None]
Collision detected
Collision detected
After: [None, 1, None, 23]
```

[Program finished]

## Practical 7b

**Aim :** Write a program to implement the concept of linear probing.

**Theory :** Linear probing is a scheme in computer programming for resolving collisions in hash tables, data structures for maintaining a collection of key–value pairs and looking up the value associated with a given key. ... Along with quadratic probing and double hashing, linear probing is a form of open addressing.

**Code :**

1 contributor

49 lines (44 sloc) | 1.83 KB

```

1 #3005 (New Rollno:372) Shravan SYIT
2 #Implement the following for Hashing:
3 #Write a program to implement the concept of linear probing.
4 class Hash:
5     def __init__(self, keys, lowerrange, higherrange):
6         self.value = self.hashfunction(keys,lowerrange, higherrange)
7
8     def get_key_value(self):
9         return self.value
10
11    def hashfunction(self,keys,lowerrange, higherrange):
12        if lowerrange == 0 and higherrange > 0:
13            return keys%(higherrange)
14
15    if __name__ == '__main__':
16        linear_probing = True
17        list_of_keys = [23,43,1,87]
18        list_of_list_index = [None,None,None]
19        print("Before : " + str(list_of_list_index))
20        for value in list_of_keys:
21            #print(Hash(value,0,len(list_of_keys)).get_key_value())
22            list_index = Hash(value,0,len(list_of_keys)).get_key_value()
23            print("hash value for " + str(value) + " is :" + str(list_index))
24            if list_of_list_index[list_index]:
25                print("Collision detected for " + str(value))
26            if linear_probing:
27                old_list_index = list_index
28                if list_index == len(list_of_list_index)-1:
29                    list_index = 0
30                else:
31                    list_index += 1
32                list_full = False
33                while list_of_list_index[list_index]:
34                    if list_index == old_list_index:
35                        list_full = True
36                        break
37                    if list_index+1 == len(list_of_list_index):
38                        list_index = 0
39                    else:
40                        list_index += 1
41                if list_full:
42                    print("List was full . Could not save")
43                else:
44                    list_of_list_index[list_index] = value
45
46            else:
47                list_of_list_index[list_index] = value
48
49        print("After: " + str(list_of_list_index))

```

## Output :

---

```
Before : [None, None, None, None]
hash value for 23 is :3
hash value for 43 is :3
hash value for 1 is :1
hash value for 87 is :3
After: [None, None, None, 87]
```

```
[Program finished]
```

## Practical 8

**Aim :** Write a program for inorder, postorder and preorder traversal of tree.

**Theory : Inorder:**

In case of binary search trees (BST), Inorder traversal gives nodes in non-decreasing order. To get nodes of BST in non-increasing order, a variation of Inorder traversal where Inorder traversal is reversed can be used.

**Preorder:**

Preorder traversal is used to create a copy of the tree. Preorder traversal is also used to get prefix expression on of an expression tree.

**Postorder:**

Postorder traversal is also useful to get the postfix expression of an expression tree.

## Code :

```

76 lines (68 sloc) | 1.97 KB
1 #3005 (New Rollno:372) Shravan SYIT
2 #Write a program for inorder, postorder and preorder traversal of tree.
3
4 class Node:
5     def __init__(self, key):
6         self.left = None
7         self.right = None
8         self.value = key
9
10    def PrintTree(self):
11        if self.left:
12            self.left.PrintTree()
13        print(self.value)
14        if self.right:
15            self.right.PrintTree()
16
17    def Printpreorder(self):
18        if self.value:
19            print(self.value)
20            if self.left:
21                self.left.Printpreorder()
22            if self.right:
23                self.right.Printpreorder()
24
25    def Printinorder(self):
26        if self.value:
27            if self.left:
28                self.left.Printinorder()
29            print(self.value)
30            if self.right:
31                self.right.Printinorder()
32
33    def Printpostorder(self):
34        if self.value:
35            if self.left:
36                self.left.Printpostorder()
37            if self.right:
38                self.right.Printpostorder()
39            print(self.value)
40
41    def insert(self, data):
42        if self.value:
43            if data < self.value:
44                if self.left is None:
45                    self.left = Node(data)
46                else:
47                    self.left.insert(data)
48            elif data > self.value:
49                if self.right is None:
50                    self.right = Node(data)
51                else:
52                    self.right.insert(data)
53            else:
54                self.value = data
55
56
57 if __name__ == '__main__':
58     root = Node(10)
59     root.left = Node(12)
60     root.right = Node(5)
61     print("Without any order")
62     root.PrintTree()
63     root_1 = Node(None)
64     root_1.insert(28)
65     root_1.insert(4)
66     root_1.insert(13)
67     root_1.insert(130)
68     root_1.insert(123)
69     print("Now ordering with insert")
70     root_1.PrintTree()
71     print("Pre order")
72     root_1.Printpreorder()
73     print("In Order")
74     root_1.Printinorder()
75     print("Post Order")
76     root_1.Printpostorder()

```

## Output :

```
Without any order
12
10
5
Now ordering with insert
4
13
28
123
130
Pre order
28
4
13
130
123
In Order
4
13
28
123
130
Post Order
13
4
123
130
28
```

[Program finished]