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)

Class: S.Y. B.Sc. IT Sem- III Roll No: 331

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. b) Implement Tower of Hanoi. c) WAP to scan a polynomial using linked list and add two polynomials. d) WAP to calculate factorial and to compute the factors of a given no. (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 elementin an array is similar to assigning values to scalar variables. Simply reference an individual element of anarray 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 :

1 contributor

64 lines (22 sloc) | 818 Bytes Raw Blame

```
class ArrayModification:



    def linear_search(self,lst,n):

        for i in range(len(lst)):

            if lst[i] == n:

                return f'Position :{i}'

        return -1



    def insertion_sort(self,lst):

        for i in range(len(lst)):



            index = lst[i]



            k = i - 1



            while k >= 0 and lst[k] > index:

                lst[k + 1] = lst[k]

                k -= 1



            lst[k+1]  = index



        return lst



    def merge(self,lst1,lst2):

        return ArrayModification.insertion_sort(lst1 + lst2)



    def reverse(self,lst):

        return lst[::-1]

lst = [2,9,1,7,3,5,2]

Arrmod = ArrayModification()

print(Arrmod.linear_search(lst,3))



```

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 :

1 contributor

58 lines (27 sloc) | 707 Bytes | Raw | Blame

```
Mat1 = [[3, 4, -6],

       [12,71,24],

       [21,3,21]]

Mat2 = [[2, 16, -16],

           [1,7,-3],

           [-1,3,3]]

Mat3 = [[0,0,0,],

     [0,0,0,],

     [0,0,0,]]

# Matrix Addition

for i in range(len(Mat1)):

    for j in range(len(Mat2[0])):

        for k in range(len(Mat2)):

            Mat3[i][j] += Mat1[i][k] + Mat2[k][j]



print(Mat3)

# Matrix Multiplication

Mat3 = [[0, 0, 0, 0],

        [0, 0, 0, 0],

        [0, 0, 0, 0]]

for i in range(len(Mat1)):

    for j in range(len(Mat2[0])):

        for k in range(len(Mat2)):

            Mat3[i][j] += Mat1[i][k] * Mat2[k][j]



print(Mat3)

#matrix transpose

for i in map(list, zip(*Mat1)):

    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 theLinkedList 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

```python
#Implement Linked List. Include options for insertion, deletion and search of a number, reverse the list and concatenate
#3005 (New Rollno:372) Shravan SYIT
class Node:
    def __init__(self, element, next=None, prev=None):
        self.element = element
        self.next = next
        self.prev = prev

    def display(self):
        print(self.element)

class LinkedList:
    def __init__(self):
        self.start = None
        self.end = None
        self.length = 0

    def is_empty(self) -> bool:
        return self.length == 0

    def get_size(self) -> int:
        return self.length

    def display(self):
        if self.is_empty():
            print("Doubly Linked List is empty")
            return
        first = self.start
        print("The List: ", end='')
        print("[" + first.element, end='')
        first = first.next
        while first:
            print(", " + first.element, end='')
            first = first.next
        print("]")

    def add_head(self, e):
        old_head = self.start
        self.start = Node(e)
        self.start.next = old_head
        if old_head is not None:
            old_head.prev = self.start
        if self.end is None:
            self.end = self.start
        self.length += 1

    def get_head(self) -> Node:
        return self.start

    def remove_head(self):
        if self.is_empty():
            print("Doubly Linked List is empty")
            return
        elif self.length == 1:
            self.start = None
            self.end = None
        else:
            self.start = self.start.next
        self.length -= 1

    def add_tail(self, e):
        new_value = Node(e)
        if self.get_tail() is not None:
            old_tail = self.end
            self.end = new_value
            self.end.prev = old_tail
            old_tail.next = self.end
            self.length += 1
        else:
            self.add_head(e)

    def get_tail(self) -> Node:
        return self.end

    def remove_tail(self):
        if self.is_empty():
            print("Doubly Linked List is empty")
            return
        elif self.length == 1:
            self.start = None
            self.end = None
        else:
            self.end = self.end.prev
        self.length -= 1

    def add_between_list(self, index, element):
        if index > self.length or index < 0:
            print("Index out of bounds")
        elif index == self.length:
            self.add_tail(element)
        elif index == 0:
            self.add_head(element)
```

```
        elif index == 0:
            self.add_head(element)
        else:
            prev_node = self.get_node_at(index - 1)
            current_node = self.get_node_at(index)
            new_value = Node(element)
            prev_node.next = new_value
            new_value.next = current_node
            new_value.prev = prev_node
            current_node.prev = new_value
            self.length += 1

    def get_node_at(self, index, direction="auto") -> Node:
        if index < 0 or index >= self.length:
            print("Index out of bounds")
        else:
            element_node = self.start
            counter = 0
            while counter < index:
                element_node = element_node.next
                counter += 1
            return element_node

    def remove_between_list(self, index):
        if index < 0 or index >= self.length:
            print("Index out of bounds")
        elif index == self.length - 1:
            self.remove_tail()
        elif index == 0:
            self.remove_head()
        else:
            prev_node = self.get_node_at(index - 1)
            next_node = self.get_node_at(index + 1)
            prev_node.next = next_node
            next_node.prev = prev_node
            self.length -= 1

    def merge(self, linked_list_value):
        if not self.is_empty():
            last_node = self.get_tail()
            last_node.next = linked_list_value.start
            if not linked_list_value.is_empty():
                linked_list_value.start.prev = last_node
                self.end = linked_list_value.end
            self.length = self.length + linked_list_value.length
        else:
            self.start = linked_list_value.start
            self.end = linked_list_value.end
            self.length = linked_list_value.length

    def reverse(self):
        if self.is_empty():
            print("Doubly Linked List is empty")
            return
        elif self.length == 1:
            return
        elif self.length == 2:
            temp = self.start
            self.start = self.end
            self.end = temp
            self.start.prev = None
            self.start.next = self.end
            self.end.prev = self.start
            self.end.next = None
        elif self.length == 3:
            mid = self.__get_node_from_start(1)
            temp = self.start
            self.start = self.end
            self.end = temp
            self.start.prev = None
            self.start.next = mid
            mid.prev = self.start
            mid.next = self.end
            self.end.prev = mid
            self.end.next = None
        elif self.length > 3:
            temp_lst = []
            first = self.start
            temp_lst.append(first)
            first = first.next
            while first:
                temp_lst.append(first)
                first = first.next
            temp_lst.reverse()
            for i in range(0, len(temp_lst)):
                if i == 0:
                    self.start = temp_lst[i]
                    self.start.prev = None
                    self.start.next = temp_lst[i + 1]
                elif i == 1:
                    temp_lst[i].prev = self.start
                    temp_lst[i].next = temp_lst[i + 1]
                elif i == len(temp_lst) - 1:
                    self.end = temp_lst[i]
                    self.end.prev = temp_lst[i - 1]
                    self.end.next = None
                else:
```

```
            first = self.start
            temp_lst.append(first)
            first = first.next
            while first:
                temp_lst.append(first)
                first = first.next
            temp_lst.reverse()
            for i in range(0, len(temp_lst)):
                if i == 0:
                    self.start = temp_lst[i]
                    self.start.prev = None
                    self.start.next = temp_lst[i + 1]
                elif i == 1:
                    temp_lst[i].prev = self.start
                    temp_lst[i].next = temp_lst[i + 1]
                elif i == len(temp_lst) - 1:
                    self.end = temp_lst[i]
                    self.end.prev = temp_lst[i - 1]
                    self.end.next = None
                else:
                    temp_lst[i].prev = temp_lst[i - 1]
                    temp_lst[i].next = temp_lst[i + 1]
            temp_lst[len(temp_lst) - 2].prev = temp_lst[len(temp_lst) - 2 - 1]
            temp_lst[len(temp_lst) - 2].next = self.end


one = LinkedList()
print(one.is_empty())
one.add_between_list(0, "zero")
one.add_between_list(1, "one")
one.add_between_list(2, "two")
one.display()
print(one.is_empty())
one.remove_between_list(1)
one.display()
one.add_between_list(1, "one")
one.display()

two = LinkedList()
two.add_head("first_name")
two.add_tail("last_name")
print("Head: ", end='')
two.get_head().display()
print("Tail: ", end='')
two.get_tail().display()
two.remove_tail()
two.get_tail().display()
two.remove_head()
try:
    two.get_head().display()
except AttributeError as e:
    print(e, "\nNo head found")
two.display()

two = LinkedList()
two.add_between_list(0, "zero")
two.add_between_list(1, "one")
two.add_between_list(2, "two")
two.add_between_list(3, "three")
two.display()
two.get_node_at(3).display()
two.get_node_at(3, "start").display()
two.get_node_at(3, "end").display()

three = LinkedList()
three.add_between_list(0, "0")
three.add_between_list(1, "1")
three.add_between_list(2, "2")
three.add_between_list(3, "4")
two.display()
print("Head: ", end='')
two.get_head().display()
print("Tail: ", end='')
two.get_tail().display()
three.display()
print("Head: ", end='')
three.get_head().display()
print("Tail: ", end='')
three.get_tail().display()
two.merge(three)
two.display()
print("Head: ", end='')
two.get_head().display()
print("Tail: ", end='')
two.get_tail().display()
print(f"Size: {two.get_size()}")
two.display()
two.reverse()
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 objectsthat 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 :

1 contributor

55 lines (45 sloc) | 1.21 KB Raw Blame

```python
#3005 (New Rollno:372) Shravan SYIT
#Implementation of the Tower Of Hanoi.
class Stack:

    def __init__(self):
        self.stack_arr = []

    def push(self,value):
        self.stack_arr.append(value)

    def pop(self):
        if len(self.stack_arr) == 0:
            print('Stack is empty!')
            return None
        else:
            self.stack_arr.pop()

    def get_head(self):
        if len(self.stack_arr) == 0:
            print('Stack is empty!')
            return None
        else:
            return self.stack_arr[-1]

    def display(self):
        if len(self.stack_arr) == 0:
            print('Stack is empty!')
            return None
        else:
            print(self.stack_arr)

A = Stack()
B = Stack()
C = Stack()
def Hanoi(n, fromrod,to,temp):
    if n == 1:
        fromrod.pop()
        to.push('disk 1')
        if to.display() != None:
            print(to.display())

    else:

        Hanoi(n-1, fromrod, temp, to)
        fromrod.pop()
        to.push(f'disk {n}')
        if to.display() !=  None:
            print(to.display())
        Hanoi(n-1, temp, to, fromrod)

n = int(input('Enter the number of the disk in rod A : '))
for i in  range(n):
    A.push(f'disk {i+1} ')

Hanoi(n, A, C, B)
```

Output :


```
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 :

1 contributor

55 lines (45 sloc) | 1.21 KB Raw Blame

```python
#3005 (New Rollno:372) Shravan SYIT
#Implementation of the Tower Of Hanoi.
class Stack:

    def __init__(self):
        self.stack_arr = []

    def push(self,value):
        self.stack_arr.append(value)

    def pop(self):
        if len(self.stack_arr) == 0:
            print('Stack is empty!')
            return None
        else:
            self.stack_arr.pop()

    def get_head(self):
        if len(self.stack_arr) == 0:
            print('Stack is empty!')
            return None
        else:
            return self.stack_arr[-1]

    def display(self):
        if len(self.stack_arr) == 0:
            print('Stack is empty!')
            return None
        else:
            print(self.stack_arr)

A = Stack()
B = Stack()
C = Stack()
def Hanoi(n, fromrod,to,temp):
    if n == 1:
        fromrod.pop()
        to.push('disk 1')
        if to.display() != None:
            print(to.display())

    else:

        Hanoi(n-1, fromrod, temp, to)
        fromrod.pop()
        to.push(f'disk {n}')
        if to.display() !=  None:
            print(to.display())
        Hanoi(n-1, temp, to, fromrod)

n = int(input('Enter the number of the disk in rod A : '))
for i in  range(n):
    A.push(f'disk {i+1} ')

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 : 4x7 + 12x2 + 45

Code :

362 lines (330 sloc) | 12.1 KB

Raw | Blame

```python
#3005 (New Rollno:372) Shravan 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)

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

    def remove_tail(self):
        if self.is_empty():
            print("Doubly Linked List is empty")
            return
        elif self.__size == 1:
            self.__head = None
            self.__tail = None
        else:
            self.__tail = self.__tail.prev
        self.__size -= 1

    def add_between_list(self, index, element):
        if index > self.__size or index < 0:
            print("Index out of bounds")
        elif index == self.__size:
            self.add_tail(element)
        elif index == 0:
            self.add_head(element)
        else:
            prev_node = self.get_node_at(index - 1)
            current_node = self.get_node_at(index)
            new_value = DoublyNode(element)
            prev_node.next = new_value
            new_value.next = current_node
            new_value.prev = prev_node
            current_node.prev = new_value
            self.__size += 1

    def __get_node_from_start(self, index):
        element_node = self.__head
        counter = 0
        while counter < index:
            element_node = element_node.next
            counter += 1
        return element_node

    def __get_node_from_end(self, index):
        element_node = self.__tail
        counter = self.__size
        while counter > index + 1:
            element_node = element_node.prev
            counter -= 1
        return element_node

    def get_node_at(self, index, direction="auto") -> DoublyNode:
        if index < 0 or index >= self.__size:
            print("Index out of bounds")
        elif direction == "start":
            return self.__get_node_from_start(index)
        elif direction == "end":
            return self.__get_node_from_end(index)
        elif self.__size == 1 or index <= self.__size // 2:
            return self.__get_node_from_start(index)
        elif index > self.__size // 2:
            return self.__get_node_from_end(index)

    def remove_between_list(self, index):
        if index < 0 or index >= self.__size:
            print("Index out of bounds")
        elif index == self.__size - 1:
            self.remove_tail()
        elif index == 0:
            self.remove_head()
        else:
            prev_node = self.get_node_at(index - 1)
            next_node = self.get_node_at(index + 1)
            prev_node.next = next_node
            next_node.prev = prev_node
            self.__size -= 1

    def search(self, search_value) -> bool:
        index = 0
        while index < self.__size:
            value = self.get_node_at(index)
            if value.element == search_value:
                print("Found value '" + str(search_value) + "' at location: " + str(index))
                return True
            index += 1
        print("Couldn't find value '" + str(search_value) + "'")
        return False

    def merge(self, linked_list_value):
        if not self.is_empty():
            last_node = self.get_tail()
            last_node.next = linked_list_value.__head
            if not linked_list_value.is_empty():
                linked_list_value.__head.prev = last_node
                self.__tail = linked_list_value.__tail
            self.__size = self.__size + linked_list_value.__size
        else:
            self.__head = linked_list_value.__head
            self.__tail = linked_list_value.__tail
```

```
        elif direction == "start":
            return self.__get_node_from_start(index)
        elif direction == "end":
            return self.__get_node_from_end(index)
        elif self.__size == 1 or index <= self.__size // 2:
            return self.__get_node_from_start(index)
        elif index > self.__size // 2:
            return self.__get_node_from_end(index)

    def remove_between_list(self, index):
        if index < 0 or index >= self.__size:
            print("Index out of bounds")
        elif index == self.__size - 1:
            self.remove_tail()
        elif index == 0:
            self.remove_head()
        else:
            prev_node = self.get_node_at(index - 1)
            next_node = self.get_node_at(index + 1)
            prev_node.next = next_node
            next_node.prev = prev_node
            self.__size -= 1

    def search(self, search_value) -> bool:
        index = 0
        while index < self.__size:
            value = self.get_node_at(index)
            if value.element == search_value:
                print("Found value '" + str(search_value) + "' at location: " + str(index))
                return True
            index += 1
        print("Couldn't find value '" + str(search_value) + "'")
        return False

    def merge(self, linked_list_value):
        if not self.is_empty():
            last_node = self.get_tail()
            last_node.next = linked_list_value.__head
            if not linked_list_value.is_empty():
                linked_list_value.__head.prev = last_node
                self.__tail = linked_list_value.__tail
            self.__size = self.__size + linked_list_value.__size
        else:
            self.__head = linked_list_value.__head
            self.__tail = linked_list_value.__tail
            self.__size = linked_list_value.__size

    def reverse(self):
        if self.is_empty():
            print("Doubly Linked List is empty")
            return
        elif self.__size == 1:
            return
        elif self.__size == 2:
            temp = self.__head
            self.__head = self.__tail
            self.__tail = temp
            self.__head.prev = None
            self.__head.next = self.__tail
            self.__tail.prev = self.__head
            self.__tail.next = None
        elif self.__size == 3:
            mid = self.__get_node_from_start(1)
            temp = self.__head
            self.__head = self.__tail
            self.__tail = temp
            self.__head.prev = None
            self.__head.next = mid
            mid.prev = self.__head
            mid.next = self.__tail
            self.__tail.prev = mid
            self.__tail.next = None
        elif self.__size > 3:
            temp_lst = []
            first = self.__head
            temp_lst.append(first)
            first = first.next
            while first:
                temp_lst.append(first)
                first = first.next
            temp_lst.reverse()
            for i in range(0, len(temp_lst)):
                if i == 0:
                    self.__head = temp_lst[i]
                    self.__head.prev = None
                    self.__head.next = temp_lst[i + 1]
                elif i == 1:
                    temp_lst[i].prev = self.__head
                    temp_lst[i].next = temp_lst[i + 1]
                elif i == len(temp_lst) - 1:
```

```
                elif i == len(temp_lst) - 1:
                    self.__tail = temp_lst[i]
                    self.__tail.prev = temp_lst[i - 1]
                    self.__tail.next = None
                else:
                    temp_lst[i].prev = temp_lst[i - 1]
                    temp_lst[i].next = temp_lst[i + 1]
            temp_lst[len(temp_lst) - 2].prev = temp_lst[len(temp_lst) - 2 - 1]
            temp_lst[len(temp_lst) - 2].next = self.__tail

    @staticmethod
    def test_main():
        dll = DoublyLinkedList()
        print(dll.is_empty())
        dll.add_between_list(0, "zero")
        dll.add_between_list(1, "one")
        dll.add_between_list(2, "two")
        dll.display()
        print(dll.is_empty())
        print("After removing")
        dll.remove_between_list(1)
        dll.display()
        print("After adding again")
        dll.add_between_list(1, "one")
        dll.display()

        dll2 = DoublyLinkedList()
        dll2.add_head("first_name")
        dll2.add_tail("last_name")
        print("Head: ", end='')
        dll2.get_head().display()
        print("Tail: ", end='')
        dll2.get_tail().display()
        dll2.remove_tail()
        dll2.get_tail().display()
        dll2.remove_head()
        try:
            dll2.get_head().display()
        except AttributeError as e:
            print(e, "\nNo head found")
        dll2.display()

        dll3 = DoublyLinkedList()
        dll3.add_between_list(0, "zero")
        dll3.add_between_list(1, "one")
        dll3.add_between_list(2, "two")
        dll3.add_between_list(3, "three")
        dll3.display()
        dll3.search("two")
        dll3.search("four")
        dll3.get_node_at(3).display()
        dll3.get_node_at(3, "start").display()
        dll3.get_node_at(3, "end").display()

        dll4 = DoublyLinkedList()
        dll4.add_between_list(0, "0")
        dll4.add_between_list(1, "1")
        dll4.add_between_list(2, "2")
        dll4.add_between_list(3, "4")
        print("The Two Lists: ")
        dll3.display()
        print("Head: ", end='')
        dll3.get_head().display()
        print("Tail: ", end='')
        dll3.get_tail().display()
        dll4.display()
        print("Head: ", end='')
        dll4.get_head().display()
        print("Tail: ", end='')
        dll4.get_tail().display()
        dll3.merge(dll4)
        print("After Merging")
        dll3.display()
        print("Head: ", end='')
        dll3.get_head().display()
        print("Tail: ", end='')
        dll3.get_tail().display()
        print(f"Size: {dll3.get_size()}")
        dll3.display()
        dll3.reverse()
        print("After Reversing")
        dll3.display()
        print("Reverse Display")
        dll3.display("end")


if __name__ == "__main__":
    polynomial_addition_1 = DoublyLinkedList()
    order = int(input('Enter the order of the polynomials : '))
    print("Polynomial 1:")
    polynomial_addition_1.add_head(int(input(f"Enter coefficient of power {order} : ")))
    for i in reversed(range(order)):
        polynomial_addition_1.add_tail(int(input(f"Enter coefficient of power {i} : ")))
    polynomial_addition_2 = DoublyLinkedList()
    print("Polynomial 2:")
    polynomial_addition_2.add_head(int(input(f"Enter coefficient of power {order} : ")))
    for i in reversed(range(order)):
```

```
        dll3 = DoublyLinkedList()
        dll3.add_between_list(0, "zero")
        dll3.add_between_list(1, "one")
        dll3.add_between_list(2, "two")
        dll3.add_between_list(3, "three")
        dll3.display()
        dll3.search("two")
        dll3.search("four")
        dll3.get_node_at(3).display()
        dll3.get_node_at(3, "start").display()
        dll3.get_node_at(3, "end").display()

        dll4 = DoublyLinkedList()
        dll4.add_between_list(0, "0")
        dll4.add_between_list(1, "1")
        dll4.add_between_list(2, "2")
        dll4.add_between_list(3, "4")
        print("The Two Lists: ")
        dll3.display()
        print("Head: ", end='')
        dll3.get_head().display()
        print("Tail: ", end='')
        dll3.get_tail().display()
        dll4.display()
        print("Head: ", end='')
        dll4.get_head().display()
        print("Tail: ", end='')
        dll4.get_tail().display()
        dll3.merge(dll4)
        print("After Merging")
        dll3.display()
        print("Head: ", end='')
        dll3.get_head().display()
        print("Tail: ", end='')
        dll3.get_tail().display()
        print(f"Size: {dll3.get_size()}")
        dll3.display()
        dll3.reverse()
        print("After Reversing")
        dll3.display()
        print("Reverse Display")
        dll3.display("end")


if __name__ == "__main__":
    polynomial_addition_1 = DoublyLinkedList()
    order = int(input('Enter the order of the polynomials : '))
    print("Polynomial 1:")
    polynomial_addition_1.add_head(int(input(f"Enter coefficient of power {order} : ")))
    for i in reversed(range(order)):
        polynomial_addition_1.add_tail(int(input(f"Enter coefficient of power {i} : ")))
    polynomial_addition_2 = DoublyLinkedList()
    print("Polynomial 2:")
    polynomial_addition_2.add_head(int(input(f"Enter coefficient of power {order} : ")))
    for i in reversed(range(order)):
        polynomial_addition_2.add_tail(int(input(f"Enter coefficient of power {i} : ")))
    superscript_map = {"0": "□", "1": "!", "2": "²", "3": "³", "4": "□", "5": "□", "6": "□",
                       "7": "□", "8": "□", "9": "□"}
    trans = str.maketrans(''.join(superscript_map.keys()), ''.join(superscript_map.values()))
    addend_1 = []
    addend_2 = []
    addend_str_1 = ""
    addend_str_2 = ""
    result = []
    result_str = ""
    for i in reversed(range(order + 1)):
        addend_1.append(polynomial_addition_1.get_node_at(i).element)
        addend_2.append(polynomial_addition_2.get_node_at(i).element)
        result.append(polynomial_addition_1.get_node_at(i).element + polynomial_addition_2.get_node_at(i).element)
    for ele in reversed(range(order + 1)):
        addend_str_1 = addend_str_1 + "%+d" % (addend_1[ele]) + "x" + (str(ele).translate(trans))
        addend_str_2 = addend_str_2 + "%+d" % (addend_2[ele]) + "x" + (str(ele).translate(trans))
        result_str = result_str + "%+d" % (result[ele]) + "x" + (str(ele).translate(trans))

    addend_str_1 = addend_str_1.replace("!", "")
    addend_str_1 = addend_str_1.lstrip("+")
    addend_str_1 = addend_str_1.rstrip("x□")

    sign = addend_str_2[0]
    addend_str_2 = addend_str_2.replace("!", "")
    addend_str_2 = addend_str_2.lstrip("+")
    addend_str_2 = addend_str_2.lstrip("-")
    addend_str_2 = addend_str_2.rstrip("x□")

    result_str = result_str.replace("!", "")
    result_str = result_str.lstrip("+")
    result_str = result_str.rstrip("x□")

    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 :

1 contributor

56 lines (50 sloc) | 1.72 KB Raw Blame

```python
#3005 (New Rollno:372) Shravan SYIT
class Fact:
    def __init__(self, number):
        self.number = number

    def factorial_iteration(self):
        factorial = 1
        if self.number < 0:
            raise Exception("Factorial of Negative number doesn't exist")
        elif self.number == 0:
            return factorial
        else:
            for i in range(1, self.number + 1):
                factorial = factorial * i
            return factorial

    def factorial_recursion(self, number=None):
        if number is None:
            number = self.number
        if number < 0:
            raise Exception("Factorial of Negative number doesn't exist")
        elif number == 0 or number == 1:
            return 1
        else:
            return number * self.factorial_recursion(number - 1)

    def factors_iteration(self):
        factors = []
        for i in range(1, self.number + 1):
            if self.number % i == 0:
                factors.append(i)
        return factors

    def factors_recursion(self, number=None, lst=None):
        factorial = self.number
        if number is None and lst is None:
            number = self.number
            lst = []
        if number == 0 or number < 0:
            return lst
        elif number == 1:
            lst.append(1)
            return sorted(lst)
        elif factorial % number == 0:
            lst.append(number)
            return self.factors_recursion(number - 1, lst)
        else:
            return self.factors_recursion(number - 1, lst)


if __name__ == '__main__':
    fact = Fact(int(input("Enter a number: ")))
    print(fact.factorial_iteration())
    print(fact.factorial_recursion())
    print(fact.factors_iteration())
    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 REARto -1

1.Enqueue Operation

check if the queue is full
for the first element, set value of FRONTto 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 Raw Blame

```
#3005 (New Rollno:372) Shravan SYIT
class ArrayQueue:
    default_capacity = 10

    def __init__(self):

        self._data = [None] * ArrayQueue.default_capacity
        self._size = 0
        self._front = 0

    def len(self):

        return self._size

    def is_empty(self):

        return self._size == 0

    def first(self):

        if self.is_empty():
            raise Exception("Queue is empty")
        return self._data[self._front]

    def dequeue(self):

        if self.is_empty():
            raise Exception("Queue is empty")
        answer = self._data[self._front]
        self._data[self._front] = None
        self._front = (self._front + 1) % len(self._data)
        self._size -= 1
        return answer

    def enqueue(self, e):
        if self._size == len(self._data):
            self._resize(2 * len(self._data))
        avail = (self._front + self._size) % len(self._data)
        self._data[avail] = e
        self._size += 1

    def _resize(self, cap):

        old = self._data
        self._data = [None] * cap
        walk = self._front
        for k in range(self._size):
            self._data[k] = old[walk]
            walk = (1 + walk) % len(old)
        self._front = 0


class DEQueue(ArrayQueue):
    def __init__(self):
        ArrayQueue.__init__(self)
        self._back = 0

    def last(self):
        if self.is_empty():
            raise Exception('Queue is empty')
        return self._data[self._back]

    def enqueue_front(self, e):
        self.enqueue(e)
        self._back = (self._front + self._size - 1) % len(self._data)

    def enqueue_back(self, e):
        if self._size == len(self._data):
            self._resize(2 * len(self._data))
        self._front = (self._front - 1) % len(self._data)
        self._data[self._front] = e
        self._size += 1
        self._back = (self._front + self._size - 1) % len(self._data)

    def dequeue_front(self):
        if self.is_empty():
            raise Exception('Queue is empty')
        back = (self._front + self._size - 1) % len(self._data)
        answer = self._data[back]
        self._data[back] = None
        self._size -= 1
        self._back = (self._front + self._size - 1) % len(self._data)
        return answer

    def dequeue_back(self):
        answer = self.dequeue()
        self._back = (self._front + self._size - 1) % len(self._data)
        return answer

    def _resize(self, cap):
        ArrayQueue._resize(self, cap)
        self._back = (self._front + self._size - 1) % len(self._data)
```

```
        old = self._data
        self._data = [None] * cap
        walk = self._front
        for k in range(self._size):
            self._data[k] = old[walk]
            walk = (1 + walk) % len(old)
        self._front = 0


class DEQueue(ArrayQueue):
    def __init__(self):
        ArrayQueue.__init__(self)
        self._back = 0

    def last(self):
        if self.is_empty():
            raise Exception('Queue is empty')
        return self._data[self._back]

    def enqueue_front(self, e):
        self.enqueue(e)
        self._back = (self._front + self._size - 1) % len(self._data)

    def enqueue_back(self, e):
        if self._size == len(self._data):
            self._resize(2 * len(self._data))
        self._front = (self._front - 1) % len(self._data)
        self._data[self._front] = e
        self._size += 1
        self._back = (self._front + self._size - 1) % len(self._data)

    def dequeue_front(self):
        if self.is_empty():
            raise Exception('Queue is empty')
        back = (self._front + self._size - 1) % len(self._data)
        answer = self._data[back]
        self._data[back] = None
        self._size -= 1
        self._back = (self._front + self._size - 1) % len(self._data)
        return answer

    def dequeue_back(self):
        answer = self.dequeue()
        self._back = (self._front + self._size - 1) % len(self._data)
        return answer

    def _resize(self, cap):
        ArrayQueue._resize(self, cap)
        self._back = (self._front + self._size - 1) % len(self._data)

    def show_all(self):
        print(f"Size: {self._size}", end=', ')
        print(f"Front: {self._front}", end=', ')
        print(f"Back: {self._back}", end=', ')
        try:
            print("First: ", self.first(), end=', ')
            print("Last: ", self.last(), end=',\n')
        except:
            print("Queue is empty")
        print(f"Queue: {self._data}")


if __name__ == "__main__":
    deq = DEQueue()
    for i in range(1, 4):
        deq.enqueue_front("f_" + str(i))
        deq.show_all()
    for i in range(1, 4):
        deq.dequeue_front()
        deq.show_all()
    for i in range(1, 4):
        deq.enqueue_back("b_" + str(i))
        deq.show_all()
    for i in range(1, 4):
        deq.dequeue_back()
        deq.show_all()
    for i in range(1, 13):
        deq.enqueue_front("f_" + str(i))
        deq.show_all()
    for i in range(1, 11):
        deq.enqueue_back("b_" + str(i))
        deq.show_all()
    for i in range(1, 13):
        deq.dequeue_front()
        deq.show_all()
    for i in range(1, 11):
        deq.dequeue_back()
        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, None
]
Size: 2, Front: 0, Back: 1, First:  f_1, Last:  f_2,
Queue: ['f_1', 'f_2', None, 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, 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, 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, None
]
Size: 0, Front: 0, Back: 9, Queue is empty
Queue: [None, 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, 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, None, 'b_1'
]
Size: 0, Front: 0, Back: 9, Queue is empty
Queue: [None, 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, None
]
Size: 2, Front: 0, Back: 1, First:  f_1, Last:  f_2,
Queue: ['f_1', 'f_2', None, 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, 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, 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, None,
None]
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, None,
 None]
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, None
, None]
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, '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, 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, 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, No
ne, None, 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, None
, None, None, None, None, None, None, None, None, None, 'b_10', 'b_
```

```
Queue: ['b_8', 'b_7', 'b_6', 'b_5', 'b_4', 'b_3', 'b_2', 'b_1', 'f_
1', None, None, None, None, None, None, None, None, None, None, Non
e, None, None, None, None, None, None, None, None, None, None, None
, None, None, None, None, None, None, None, 'b_10', 'b_9']
Size: 10, Front: 38, Back: 7, First:  b_10, Last:  b_1,
Queue: ['b_8', 'b_7', 'b_6', 'b_5', 'b_4', 'b_3', 'b_2', 'b_1', Non
e, None, None, None, None, None, None, None, None, None, None, None
, None, None, None, None, None, None, None, None, None, None, None,
 None, None, None, None, None, None, None, 'b_10', 'b_9']
Size: 9, Front: 39, Back: 7, First:  b_9, Last:  b_1,
Queue: ['b_8', 'b_7', 'b_6', 'b_5', 'b_4', 'b_3', 'b_2', 'b_1', Non
e, None, None, None, None, None, None, None, None, None, None, None
, None, None, None, None, None, None, None, None, None, None, None,
 None, None, None, None, None, None, None, None, 'b_9']
Size: 8, Front: 0, Back: 7, First:  b_8, Last:  b_1,
Queue: ['b_8', 'b_7', 'b_6', 'b_5', 'b_4', 'b_3', 'b_2', 'b_1', Non
e, None, None, None, None, None, None, None, None, None, None, None
, None, None, None, None, None, None, None, None, None, None, None,
 None, None, None, None, None, None, None, None, None]
Size: 7, Front: 1, Back: 7, First:  b_7, Last:  b_1,
Queue: [None, 'b_7', 'b_6', 'b_5', 'b_4', 'b_3', 'b_2', 'b_1', None
, None, None, None, None, None, None, None, None, None, None, None,
 None, None, None, None, None, None, None, None, None, None, None,
None, None, None, None, None, None, None, None, None]
Size: 6, Front: 2, Back: 7, First:  b_6, Last:  b_1,
Queue: [None, None, 'b_6', 'b_5', 'b_4', 'b_3', 'b_2', 'b_1', None,
 None, None, None, None, None, None, None, None, None, None, None,
None, None, None, None, None, None, None, None, None, None, None, N
one, None, None, None, None, None, None, None, None]
Size: 5, Front: 3, Back: 7, First:  b_5, Last:  b_1,
Queue: [None, None, None, 'b_5', 'b_4', 'b_3', 'b_2', 'b_1', None,
None, None, None, None, None, None, None, None, None, None, None, N
one, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None]
Size: 4, Front: 4, Back: 7, First:  b_4, Last:  b_1,
Queue: [None, None, None, None, 'b_4', 'b_3', 'b_2', 'b_1', None, N
one, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, Non
e, None, None, None, None, None, None, None, None]
Size: 3, Front: 5, Back: 7, First:  b_3, Last:  b_1,
Queue: [None, None, None, None, None, 'b_3', 'b_2', 'b_1', None, No
ne, None, None, None, None, None, None, None, None, None, None, Non
e, None, None, None, None, None, None, None, None, None, None, None
, None, None, None, None, None, None, None, None]
Size: 2, Front: 6, Back: 7, First:  b_2, Last:  b_1,
Queue: [None, None, None, None, None, None, 'b_2', 'b_1', None, Non
e, None, None, None, None, None, None, None, None, None, None, None
, None, None, None, None, None, None, None, None, None, None, None,
 None, None, None, None, None, None, None, None]
Size: 1, Front: 7, Back: 7, First:  b_1, Last:  b_1,
Queue: [None, None, None, None, None, None, None, 'b_1', None, None
, None, None, None, None, None, None, None, None, None, None, None,
 None, None, None, None, None, None, None, None, None, None, None,
None, None, None, None, None, None, None, None, None]
Size: 0, Front: 8, Back: 7, Queue is empty
Queue: [None, None, None, None, None, None, None, None, None, None,
 None, None, None, None, None, None, None, None, None, None, None,
None, None, None, None, None, None, None, None, None, None, None, N
one, None, None, None, None, None, None, None, None]

[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 whichsearches 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 searchor half-
interval search algorithm finds the position of a target value within a sorted array. The binary searchalgorithm 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

```python
#to search an element from a list and give user the option to perform Linear or Binary search.
#3005 (New Rollno:372) Shravan SYIT
class Search:
    def __init__(self, lst, ele):
        self.lst = lst
        self.ele = ele

    def binary_search(self):
        sorted_lst = sorted(self.lst)
        start = 0
        end = len(sorted_lst) - 1
        while start <= end:
            mid = (end + start) // 2
            if sorted_lst[mid] < self.ele:
                start = mid + 1
            elif sorted_lst[mid] > self.ele:
                end = mid - 1
            else:
                return mid
        return False

    def linear_search(self):
        for i in range(len(self.lst)):
            if self.lst[i] == self.ele:
                return i
        return False


if __name__ == '__main__':
    test_list = [5,31,44,94,108,109]
    test_value_1 = 42
    test_value_2 = 44
    test_search = Search(test_list, test_value_1)
    print(test_search.binary_search())
    print(test_search.linear_search())
    test_search_2 = Search(test_list, test_value_2)
    print(test_search_2.binary_search())
    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 sortalgorithm 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

```python
#to sort a list of elements. Give user the option to perform sorting using Insertion sort, Bubble sort or Selection sort.
#3005 (New Rollno:372) Shravan SYIT
class Sort:
    def __init__(self, lst):
        self.lst = lst

    def bubble_sort(self):
        sorted_lst = self.lst.copy()
        for i in range(len(sorted_lst) - 1):
            for j in range(0, len(sorted_lst) - i - 1):
                if sorted_lst[j] > sorted_lst[j + 1]:
                    sorted_lst[j], sorted_lst[j + 1] = sorted_lst[j + 1], sorted_lst[j]
        return sorted_lst

    def selection_sort(self):
        sorted_lst = self.lst.copy()
        for i in range(len(sorted_lst)):
            min_idx = i
            for j in range(i + 1, len(sorted_lst)):
                if sorted_lst[min_idx] > sorted_lst[j]:
                    min_idx = j
            sorted_lst[i], sorted_lst[min_idx] = sorted_lst[min_idx], sorted_lst[i]
        return sorted_lst

    def insertion_sort(self):
        sorted_lst = self.lst.copy()
        for i in range(1, len(sorted_lst)):
            key = sorted_lst[i]
            j = i - 1
            while j >= 0 and key < sorted_lst[j]:
                sorted_lst[j + 1] = sorted_lst[j]
                j -= 1
            sorted_lst[j + 1] = key
        return sorted_lst


if __name__ == '__main__':
    test_list = [44,31,108,94,109]
    test_sort = Sort(test_list)
    print(test_sort.bubble_sort())
    print(test_sort.selection_sort())
    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 Structurewhich 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 thehash 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 hashresult. If two separate inputs produce the same hashoutput, 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

```python
#3005 (New Rollno:372) Shravan SYIT
#Implement the following Hashing Techniques.
#Write a program to implement the collision technique.

class Hash:
    def __init__(self, keys, lowerrange, higherrange):
        self.value = self.hashfunction(keys,lowerrange, higherrange)

    def get_key_value(self):
        return self.value

    def hashfunction(self,keys,lowerrange, higherrange):
        if lowerrange == 0 and higherrange > 0:
            return keys%(higherrange)

if __name__ == '__main__':
    list_of_keys = [23,43,1,87]
    list_of_list_index = [None,None,None,None]
    print("Before : " + str(list_of_list_index))
    for value in list_of_keys:
        #print(Hash(value,0,len(list_of_keys)).get_key_value())
        list_index = Hash(value,0,len(list_of_keys)).get_key_value()
        if list_of_list_index[list_index]:
            print("Collision detected")
        else:
            list_of_list_index[list_index] = value

    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 Raw Blame

```python
#3005 (New Rollno:372) Shravan SYIT
#Implement the following for Hashing:
#Write a program to implement the concept of linear probing.
class Hash:
    def __init__(self, keys, lowerrange, higherrange):
        self.value = self.hashfunction(keys,lowerrange, higherrange)

    def get_key_value(self):
        return self.value

    def hashfunction(self,keys,lowerrange, higherrange):
        if lowerrange == 0 and higherrange > 0:
            return keys%(higherrange)

if __name__ == '__main__':
    linear_probing = True
    list_of_keys = [23,43,1,87]
    list_of_list_index = [None,None,None,None]
    print("Before : " + str(list_of_list_index))
    for value in list_of_keys:
        #print(Hash(value,0,len(list_of_keys)).get_key_value())
        list_index = Hash(value,0,len(list_of_keys)).get_key_value()
        print("hash value for " + str(value) + " is :" + str(list_index))
        if list_of_list_index[list_index]:
            print("Collission detected for " + str(value))
            if linear_probing:
                old_list_index = list_index
                if list_index == len(list_of_list_index)-1:
                    list_index = 0
                else:
                    list_index += 1
                list_full = False
                while list_of_list_index[list_index]:
                    if list_index == old_list_index:
                        list_full = True
                        break
                    if list_index+1 == len(list_of_list_index):
                        list_index = 0
                    else:
                        list_index += 1
                if list_full:
                    print("List was full . Could not save")
                else:
                    list_of_list_index[list_index] = value

    else:
        list_of_list_index[list_index] = value

    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 s 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 Raw Blame

```python
#3005 (New Rollno:372) Shravan SYIT
#Write a program for inorder, postorder and preorder traversal of tree.

class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.value = key

    def PrintTree(self):
        if self.left:
            self.left.PrintTree()
        print(self.value)
        if self.right:
            self.right.PrintTree()

    def Printpreorder(self):
        if self.value:
            print(self.value)
            if self.left:
                self.left.Printpreorder()
            if self.right:
                self.right.Printpreorder()

    def Printinorder(self):
        if self.value:
            if self.left:
                self.left.Printinorder()
            print(self.value)
            if self.right:
                self.right.Printinorder()

    def Printpostorder(self):
        if self.value:
            if self.left:
                self.left.Printpostorder()
            if self.right:
                self.right.Printpostorder()
            print(self.value)

    def insert(self, data):
        if self.value:
            if data < self.value:
                if self.left is None:
                    self.left = Node(data)
                else:
                    self.left.insert(data)
            elif data > self.value:
                if self.right is None:
                    self.right = Node(data)
                else:
                    self.right.insert(data)
        else:
            self.value = data


if __name__ == '__main__':
    root = Node(10)
    root.left = Node(12)
    root.right = Node(5)
    print("Without any order")
    root.PrintTree()
    root_1 = Node(None)
    root_1.insert(28)
    root_1.insert(4)
    root_1.insert(13)
    root_1.insert(130)
    root_1.insert(123)
    print("Now ordering with insert")
    root_1.PrintTree()
    print("Pre order")
    root_1.Printpreorder()
    print("In Order")
    root_1.Printinorder()
    print("Post Order")
    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]
```