|  |  |  |  |
| --- | --- | --- | --- |
| **SR.NO** | **NAME OF PROGRAM** | **PAGE NO.** | **SIGN** |
| 1 | Write a python program to create a sparse matrix using csc\_matrix() . | 2-3 |  |
| 2 | Write a python program to implement different array operations. | 4-6 |  |
| 3 | Write a python function to add a node in singly linked list. | 7-8 |  |
| 4 | Write a python function to delete a node in singly linked list. | 9-12 |  |
| 5 | Write a python function to add a node in doubly linked list. | 13-14 |  |
| 6 | Write a python function to delete a node in doubly linked list. | 15-18 |  |
| 7 | Write a function to push, pop element to a stack using link list. | 19-21 |  |
| 8 | Write a function to insert, delete, IsFull and IsEmpty to a queue using link list. | 22-25 |  |
| 9 | Write python function to insert node into Binary search tree. | 26-27 |  |
| 10 | Accept vertices and edges for a graph and represent it as adjacency list. | 28-29 |  |
| 11 | Write python program to sort an array using bubble sort. | 30 |  |
| 12 | Write python program to sort an array using merge sort. | 31-32 |  |

**INDEX**

|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Roll Number: 31 Sign: Date: / /** |
| **Student Name: Vaishnavi Suresh Gawli** |
| **Subject Name: Data Structure and Algorithm** |
| **Program Title: 1.Write a python program to create a sparse matrix using csc\_matrix() .** |

**Solution:**

import numpy as np

from scipy.sparse import csc\_matrix

# Create a dense matrix (2D array)

dense\_matrix = np.array([

[1, 0, 0, 4],

[0, 0, 0, 0],

[0, 2, 0, 0],

[5, 0, 3, 0]

])

# Convert the dense matrix to a sparse matrix in Compressed Sparse Column (CSC) format

sparse\_matrix = csc\_matrix(dense\_matrix)

# Display the sparse matrix

print("Sparse Matrix in CSC format:")

print(sparse\_matrix)

# Display the dense representation of the sparse matrix

print("\nDense Matrix:")

print(sparse\_matrix.toarray())

# Display the number of stored elements

print("\nNumber of non-zero elements:", sparse\_matrix.nnz)

**OUTPUT:**

Sparse Matrix in CSC format:

(0, 0) 1

(0, 3) 4

(2, 1) 2

(3, 0) 5

(3, 2) 3

Dense Matrix:

[[1 0 0 4]

[0 0 0 0]

[0 2 0 0]

[5 0 3 0]]

Number of non-zero elements: 5

|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Roll Number: 31 Sign: Date: / /** |
| **Student Name: Vaishnavi Suresh Gawli** |
| **Subject Name: Data Structure and Algorithm** |
| **Program Title: 2. Write a python program to implement different array operations.** |

**Solution:**

# Importing the array module

import array

# Function to display the array

def display\_array(arr):

print("Current Array:", arr.tolist())

# Array operations

def array\_operations():

# Creating an array

arr = array.array('i', [10, 20, 30, 40, 50])

print("Array created:")

display\_array(arr)

# Append an element to the array

arr.append(60)

print("\nAfter appending 60:")

display\_array(arr)

# Insert an element at a specific index

arr.insert(2, 25)

print("\nAfter inserting 25 at index 2:")

display\_array(arr)

# Remove an element from the array

arr.remove(40)

print("\nAfter removing 40:")

display\_array(arr)

# Access an element by index

print("\nElement at index 3:", arr[3])

# Update an element at a specific index

arr[1] = 15

print("\nAfter updating element at index 1 to 15:")

display\_array(arr)

# Find the index of a specific element

print("\nIndex of element 50:", arr.index(50))

# Pop an element from the array

popped = arr.pop(4)

print(f"\nAfter popping element at index 4 (popped element: {popped}):")

display\_array(arr)

# Execute the operations

print("Array Operations:")

array\_operations()

**OUTPUT:**

Array Operations:

Array created:

Current Array: [10, 20, 30, 40, 50]

After appending 60:

Current Array: [10, 20, 30, 40, 50, 60]

After inserting 25 at index 2:

Current Array: [10, 20, 25, 30, 40, 50, 60]

After removing 40:

Current Array: [10, 20, 25, 30, 50, 60]

Element at index 3: 30

After updating element at index 1 to 15:

Current Array: [10, 15, 25, 30, 50, 60]

Index of element 50: 4

After popping element at index 4 (popped element: 50):

Current Array: [10, 15, 25, 30, 60]

|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Roll Number: 31 Sign: Date: / /** |
| **Student Name: Vaishnavi Suresh Gawli** |
| **Subject Name: Data Structure and Algorithm** |
| **Program Title: 3.Write a python function to add a node in singly linked list.** |

**Solution:**

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class SinglyLinkedList:

def \_\_init\_\_(self):

self.head = None

def add\_node(self, data):

new\_node = Node(data) # Create a new node with the given data

if not self.head: # If the list is empty, make the new node the head

self.head = new\_node

else:

# Traverse to the end of the list

current = self.head

while current.next:

current = current.next

# Add the new node at the end

current.next = new\_node

def display(self):

current = self.head

while current:

print(current.data, end=" -> ")

current = current.next

print("None")

# Example usage

sll = SinglyLinkedList()

sll.add\_node(10)

sll.add\_node(20)

sll.add\_node(30)

print("Singly Linked List after adding nodes:")

sll.display()

**OUTPUT:**

Linked List:

1. -> 20 -> 30 -> None

|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Roll Number: 31 Sign: Date: / /** |
| **Student Name: Vaishnavi Suresh Gawli** |
| **Subject Name: Data Structure and Algorithm** |
| **Program Title: 4. Write a python function to delete a node in singly linked list.** |

**Solution:**

class Node:

def \_\_init\_\_(self, data):

self.data = data # Store the data

self.next = None # The next node is initially None

class LinkedList:

def \_\_init\_\_(self):

self.head = None # The list is initially empty

# Method to add a node to the linked list

def add\_node(self, data):

new\_node = Node(data) # Create a new node with the given data

# If the list is empty, make the new node the head of the list

if not self.head:

self.head = new\_node

return

# Traverse to the end of the list

last = self.head

while last.next:

last = last.next

# Link the last node to the new node

last.next = new\_node

# Method to delete a node in the linked list

def delete\_node(self, key):

current = self.head

# Case 1: If the list is empty, do nothing

if not current:

print("The list is empty!")

return

# Case 2: If the node to be deleted is the head node

if current.data == key:

self.head = current.next # Move the head to the next node

current = None # Free the memory (delete the node)

return

# Case 3: If the node to be deleted is not the head node

prev = None

while current and current.data != key:

prev = current

current = current.next

# If the key was not found

if not current:

print(f"Node with data {key} not found!")

return

# Unlink the node from the list

prev.next = current.next

current = None # Free the memory (delete the node)

# Method to print the linked list

def print\_list(self):

current = self.head

while current:

print(current.data, end=" -> ")

current = current.next

print("None")

# Example usage:

linked\_list = LinkedList()

# Adding nodes to the linked list

linked\_list.add\_node(10)

linked\_list.add\_node(20)

linked\_list.add\_node(30)

linked\_list.add\_node(40)

print("Original Linked List:")

linked\_list.print\_list()

# Deleting a node (e.g., node with data 20)

linked\_list.delete\_node(20)

print("\nLinked List after deleting node with data 20:")

linked\_list.print\_list()

# Deleting the head node (e.g., node with data 10)

linked\_list.delete\_node(10)

print("\nLinked List after deleting head node (data 10):")

linked\_list.print\_list()

# Trying to delete a non-existing node (e.g., node with data 100)

linked\_list.delete\_node(100)

**OUTPUT:**

Original Linked List:

10 -> 20 -> 30 -> 40 -> None

Linked List after deleting node with data 20:

10 -> 30 -> 40 -> None

Linked List after deleting head node (data 10):

30 -> 40 -> None

Node with data 100 not found!

|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Roll Number: 31 Sign: Date: / /** |
| **Student Name: Vaishnavi Suresh Gawli** |
| **Subject Name: Data Structure and Algorithm** |
| **Program Title: 5. Write a python function to add a node in doubly linked list.** |

**Solution:**

class Node:

def \_\_init\_\_(self, data):

self.data = data # Store data

self.prev = None # Pointer to the previous node

self.next = None # Pointer to the next node

class DoublyLinkedList:

def \_\_init\_\_(self):

self.head = None # Initialize the list with no nodes

# Function to add a node at the end of the list

def append(self, data):

new\_node = Node(data) # Create a new node

# If the list is empty, make the new node the head

if not self.head:

self.head = new\_node

return

# Otherwise, traverse to the last node

last\_node = self.head

while last\_node.next:

last\_node = last\_node.next

# Update pointers

last\_node.next = new\_node # Link the last node to the new node

new\_node.prev = last\_node # Set the new node's prev to the last node

# Function to print the doubly linked list

def print\_list(self):

current = self.head

while current:

print(current.data, end=" <-> ")

current = current.next

print("None")

# Example usage:

dll = DoublyLinkedList()

# Add nodes to the list

dll.append(10)

dll.append(20)

dll.append(30)

# Print the doubly linked list

dll.print\_list()

**OUTPUT**:

10 <-> 20 <-> 30 <-> None

|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Roll Number: 31 Sign: Date: / /** |
| **Student Name: Vaishnavi Suresh Gawli** |
| **Subject Name: Data Structure and Algorithm** |
| **Program Title: 6. Write a python function to delete a node in doubly linked list.** |

**Solution:**

class Node:

def \_\_init\_\_(self, data):

self.data = data # Store data

self.prev = None # Pointer to the previous node

self.next = None # Pointer to the next node

class DoublyLinkedList:

def \_\_init\_\_(self):

self.head = None # Initialize the list with no nodes

# Function to append a node at the end

def append(self, data):

new\_node = Node(data)

if not self.head:

self.head = new\_node

return

last\_node = self.head

while last\_node.next:

last\_node = last\_node.next

last\_node.next = new\_node

new\_node.prev = last\_node

# Function to delete a node from the doubly linked list

def delete\_node(self, key):

# If the list is empty, return

if not self.head:

print("The list is empty.")

return

# Case 1: The node to be deleted is the head node

if self.head.data == key:

temp = self.head

self.head = self.head.next # Move the head to the next node

if self.head:

self.head.prev = None # Set the prev pointer of the new head to None

temp = None # Delete the old head

return

# Case 2: The node to be deleted is in the middle or at the end

current = self.head

while current:

if current.data == key:

break

current = current.next

# If the node was not found in the list

if current is None:

print(f"Node with data {key} not found.")

return

# Case 2a: Node is not the last node

if current.next:

current.next.prev = current.prev # Link the next node to the previous one

# Case 2b: Node is not the first node

if current.prev:

current.prev.next = current.next # Link the previous node to the next one

current = None # Delete the node

# Function to print the doubly linked list

def print\_list(self):

current = self.head

while current:

print(current.data, end=" <-> ")

current = current.next

print("None")

# Example usage:

dll = DoublyLinkedList()

# Add nodes to the list

dll.append(10)

dll.append(20)

dll.append(30)

dll.append(40)

# Print the list before deletion

print("Before deletion:")

dll.print\_list()

# Delete a node with data 20

dll.delete\_node(20)

# Print the list after deletion

print("After deletion:")

dll.print\_list()

# Try to delete a node that doesn't exist

dll.delete\_node(100)

**OUTPUT:**

Before deletion:

10 <-> 20 <-> 30 <-> 40 <-> None

After deletion:

10 <-> 30 <-> 40 <-> None

Node with data 100 not found.

|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Roll Number: 31 Sign: Date: / /** |
| **Student Name: Vaishnavi Suresh Gawli** |
| **Subject Name: Data Structure and Algorithm** |
| **Program Title: 7. Write a function to push, pop element to a stack using link list.** |

**Solution:**

class Node:

def \_\_init\_\_(self, data):

self.data = data # Store data

self.next = None # Pointer to the next node

class Stack:

def \_\_init\_\_(self):

self.top = None # Initialize the stack with no elements (empty stack)

# Push function: Add an element to the top of the stack

def push(self, data):

new\_node = Node(data) # Create a new node with the given data

new\_node.next = self.top # Set the new node's next to the current top

self.top = new\_node # Make the new node the new top of the stack

# Pop function: Remove the top element from the stack

def pop(self):

# Check if the stack is empty

if self.is\_empty():

print("Stack is empty, cannot pop.")

return None

# Remove the top node

popped\_node = self.top

self.top = self.top.next # Move the top pointer to the next node

popped\_data = popped\_node.data

popped\_node = None # Free the memory of the popped node

return popped\_data

# Peek function: Get the top element without removing it

def peek(self):

if self.is\_empty():

print("Stack is empty.")

return None

return self.top.data

# Function to check if the stack is empty

def is\_empty(self):

return self.top is None

# Function to print the stack elements (from top to bottom)

def print\_stack(self):

current = self.top

if self.is\_empty():

print("Stack is empty.")

return

while current:

print(current.data, end=" -> ")

current = current.next

print("None")

# Example usage:

stack = Stack()

# Push elements to the stack

stack.push(10)

stack.push(20)

stack.push(30)

# Print the stack

print("Stack after pushes:")

stack.print\_stack()

# Pop an element from the stack

print("\nPopped element:", stack.pop())

# Print the stack after popping

print("Stack after pop:")

stack.print\_stack() # Output: 20 -> 10 -> None

# Peek at the top element

print("\nTop element:", stack.peek())

# Pop all elements

stack.pop()

stack.pop()

# Check if the stack is empty

print("\nIs stack empty?", stack.is\_empty())

**OUTPUT:**

Stack after pushes:

30 -> 20 -> 10 -> None

Popped element: 30

Stack after pop:

20 -> 10 -> None

Top element: 20

Is stack empty? True

|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Roll Number: 31 Sign: Date: / /** |
| **Student Name: Vaishnavi Suresh Gawli** |
| **Subject Name: Data Structure and Algorithm** |
| **Program Title: 8. Write a function to insert, delete, IsFull and IsEmpty to a queue using link list.** |

**Solution:**

class Node:

def \_\_init\_\_(self, data):

self.data = data # Store data

self.next = None # Pointer to the next node

class Queue:

def \_\_init\_\_(self):

self.front = None # Pointer to the front of the queue

self.rear = None # Pointer to the rear of the queue

self.size = 0 # Tracks the current size of the queue

# Enqueue function: Add an element to the rear of the queue

def enqueue(self, data):

new\_node = Node(data) # Create a new node with the given data

if self.isEmpty():

self.front = self.rear = new\_node # If the queue is empty, set both front and rear to the new node

else:

self.rear.next = new\_node # Link the current rear node to the new node

self.rear = new\_node # Move the rear pointer to the new node

self.size += 1 # Increment size

# Dequeue function: Remove an element from the front of the queue

def dequeue(self):

if self.isEmpty():

print("Queue is empty, cannot dequeue.")

return None

dequeued\_node = self.front

self.front = self.front.next # Move the front pointer to the next node

if self.front is None: # If the queue becomes empty, reset rear to None

self.rear = None

dequeued\_data = dequeued\_node.data

dequeued\_node = None # Free the memory of the dequeued node

self.size -= 1 # Decrement size

return dequeued\_data

# Function to check if the queue is empty

def isEmpty(self):

return self.front is None

# Function to check if the queue is full (in a linked list, it's never full unless out of memory)

def isFull(self):

# Technically, the queue using a linked list will never be full unless the system runs out of memory.

return False

# Function to print the elements of the queue

def print\_queue(self):

if self.isEmpty():

print("Queue is empty.")

return

current = self.front

while current:

print(current.data, end=" -> ")

current = current.next

print("None")

# Function to get the size of the queue

def get\_size(self):

return self.size

# Example usage:

queue = Queue()

# Enqueue elements to the queue

queue.enqueue(10)

queue.enqueue(20)

queue.enqueue(30)

# Print the queue

print("Queue after enqueues:")

queue.print\_queue()

# Dequeue an element from the queue

print("\nDequeued element:", queue.dequeue()) # Output: Dequeued element: 10

# Print the queue after dequeue

print("Queue after dequeue:")

queue.print\_queue()

# Check if the queue is empty

print("\nIs queue empty?", queue.isEmpty())

# Get the size of the queue

print("\nSize of queue:", queue.get\_size())

# Dequeue all elements

queue.dequeue()

queue.dequeue()

# Check if the queue is empty after all dequeues

print("\nIs queue empty?", queue.isEmpty())

**OUTPUT:**

Queue after enqueues:

10 -> 20 -> 30 -> None

Dequeued element: 10

Queue after dequeue:

20 -> 30 -> None

Is queue empty? False

Size of queue: 2

Is queue empty? True

|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Roll Number: 31 Sign: Date: / /** |
| **Student Name: Vaishnavi Suresh Gawli** |
| **Subject Name: Data Structure and Algorithm** |
| **Program Title: 9. Write python function to insert node into Binary search tree.** |

**Solution:**

class Node:

def \_\_init\_\_(self, key):

self.key = key

self.left = None

self.right = None

class BinarySearchTree:

def \_\_init\_\_(self):

self.root = None

def insert(self, root, key):

# If the tree is empty, return a new node

if root is None:

return Node(key)

# Otherwise, recur down the tree

if key < root.key:

root.left = self.insert(root.left, key)

elif key > root.key:

root.right = self.insert(root.right, key)

# Return the unchanged root node

return root

def add(self, key):

self.root = self.insert(self.root, key)

# Helper function to print the tree (inorder traversal)

def inorder(self, root):

if root:

self.inorder(root.left)

print(root.key, end=" ")

self.inorder(root.right)

# Example usage

bst = BinarySearchTree()

bst.add(50)

bst.add(30)

bst.add(70)

bst.add(20)

bst.add(40)

bst.add(60)

bst.add(80)

print("Inorder traversal of the BST:")

bst.inorder(bst.root)

**OUTPUT:**

Inorder traversal of the BST:

1. 0 40 50 60 70 80

|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Roll Number: 31 Sign: Date: / /** |
| **Student Name: Vaishnavi Suresh Gawli** |
| **Subject Name: Data Structure and Algorithm** |
| **Program Title: 10. Accept vertices and edges for a graph and represent it as adjacency list.** |

**Solution:**

def create\_adjacency\_list(vertices, edges):

# Initialize an empty adjacency list

adjacency\_list = {vertex: [] for vertex in vertices}

# Add edges to the adjacency list

for edge in edges:

src, dest = edge

adjacency\_list[src].append(dest)

adjacency\_list[dest].append(src) # Uncomment for undirected graph

return adjacency\_list

# Input vertices and edges

vertices = input("Enter the vertices (comma-separated): ").split(",")

edges\_count = int(input("Enter the number of edges: "))

edges = []

print("Enter each edge in the format 'vertex1 vertex2':")

for \_ in range(edges\_count):

edge = input().split()

edges.append((edge[0], edge[1]))

# Create adjacency list

adj\_list = create\_adjacency\_list(vertices, edges)

# Display the adjacency list

print("\nAdjacency List:")

for vertex, neighbors in adj\_list.items():

print(f"{vertex}: {', '.join(neighbors)}")

**OUTPUT:**

Adjacency List:

A: B, C

B: A, D

C: A

D: B

|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Roll Number: 31 Sign: Date: / /** |
| **Student Name: Vaishnavi Suresh Gawli** |
| **Subject Name: Data Structure and Algorithm** |
| **Program Title: 11. Write python program to sort an array using bubble sort.** |

**Solution:**

# Bubble Sort Implementation

def bubble\_sort(arr):

n = len(arr)

# Traverse through all array elements

for i in range(n - 1):

# Last i elements are already sorted

for j in range(n - i - 1):

# Swap if the element found is greater than the next element

if arr[j] > arr[j + 1]:

arr[j], arr[j + 1] = arr[j + 1], arr[j]

# Example usage

array = [64, 34, 25, 12, 22, 11, 90]

print("Original array:", array)

bubble\_sort(array)

print("Sorted array:", array)

**OUTPUT:**

Original array: [64, 34, 25, 12, 22, 11, 90]

Sorted array: [11, 12, 22, 25, 34, 64, 90]

|  |
| --- |
| **PIRENS Institute of Business Management and Administration, Loni BK.** |
| **Roll Number: 31 Sign: Date: / /** |
| **Student Name: Vaishnavi Suresh Gawli** |
| **Subject Name: Data Structure and Algorithm** |
| **Program Title: 12. Write python program to sort an array using merge sort.** |

**Solution:**

def merge\_sort(arr):

if len(arr) > 1:

# Finding the mid of the array

mid = len(arr) // 2

# Dividing the array into two halves

left\_half = arr[:mid]

right\_half = arr[mid:]

# Recursive call to sort each half

merge\_sort(left\_half)

merge\_sort(right\_half)

# Merging the sorted halves

i = j = k = 0

# Copy data to temporary arrays L[] and R[]

while i < len(left\_half) and j < len(right\_half):

if left\_half[i] < right\_half[j]:

arr[k] = left\_half[i]

i += 1

else:

arr[k] = right\_half[j]

j += 1

k += 1

# Checking if any element was left

while i < len(left\_half):

arr[k] = left\_half[i]

i += 1

k += 1

while j < len(right\_half):

arr[k] = right\_half[j]

j += 1

k += 1

# Example usage

array = [38, 27, 43, 3, 9, 82, 10]

print("Original array:", array)

merge\_sort(array)

print("Sorted array:", array)

**OUTPUT:**

Original array: [38, 27, 43, 3, 9, 82, 10]

Sorted array: [3, 9, 10, 27, 38, 43, 82]