Data Structure

Practical – 1:

Aim: Write a program to implement ADT

Code A: 1D Array

```
import array as arr

# Define arrays
a = arr.array('i', [1, 2, 3, 1])
b = arr.array('i', [4, 5, 6, 7])
c = arr.array('i', [8, 9, 1, 2])

# Insert element into array 'a'
a.insert(1, 4)

# Printing original array
print("The new created array is:", end="")
for i in range(len(a)):
   print(a[i], end=" ")
print()

# Pop an element from array 'a'
print("The popped element is:", end="")
print(a.pop(2))

# Printing array after popping
```

```python
print("The array after popping is:", end="")
for i in range(len(a)):
    print(a[i], end=" ")
print()

# Using remove() to remove the 1st occurrence of 1
a.remove(1)

# Printing array after removing
print("The array after removing is:", end="")
for i in range(len(a)):
    print(a[i], end=" ")
print()

# Merge arrays 'b' and 'c'
a = b + c

# Printing array after merging
print("The array after Merging is:", end="")
for i in range(len(a)):
    print(a[i], end=" ")
print()

# Sort the array
a = arr.array('i', sorted(a))
```

```
# Printing array after sorting
print("The array after Sorting is:", end="")
for i in range(len(a)):
    print(a[i], end=" ")
print()
```

Output:

```
The new created array is:1 4 2 3 1
The popped element is:2
The array after popping is:1 4 3 1
The array after removing is:4 3 1
The array after Merging is:4 5 6 7 8 9 1 2
The array after Sorting is:1 2 4 5 6 7 8 9
```

Code (Extra):

```
class OneDArrayADT:
    def __init__(self):
        self.array = []

    def add_element(self, element):
        self.array.append(element)

    def delete_element(self, element):
        if element in self.array:
            self.array.remove(element)
            print(f"Element {element} deleted.")
```

```
        else:
            print("Element not found in the array.")

    def search_element(self, element):
        return element in self.array

    def sort_array(self):
        self.array.sort()
        print("Array sorted.")

    def merge_arrays(self, other_array):
        self.array.extend(other_array)
        print("Arrays merged.")

    def display_array(self):
        print("Array:", self.array)


# Example usage
if __name__ == "__main__":
    array_adt = OneDArrayADT()

    array_adt.add_element(5)
    array_adt.add_element(10)
    array_adt.add_element(2)
    array_adt.add_element(8)
```

```
    print("Initial array:")
    array_adt.display_array()

    print("\nSearching for 10:", array_adt.search_element(10))
    print("Searching for 15:", array_adt.search_element(15))

    array_adt.sort_array()
    print("\nArray after sorting:")
    array_adt.display_array()

    other_array = [7, 3, 12]
    array_adt.merge_arrays(other_array)
    print("\nArray after merging:")
    array_adt.display_array()

    array_adt.delete_element(8)
    print("\nArray after deleting 8:")
    array_adt.display_array()
```

Output:

```
Initial array:
Array: [5, 10, 2, 8]

Searching for 10: True
Searching for 15: False
```

Array sorted.

Array after sorting:

Array: [2, 5, 8, 10]

Arrays merged.

Array after merging:

Array: [2, 5, 8, 10, 7, 3, 12]

Element 8 deleted.

Array after deleting 8:

Array: [2, 5, 10, 7, 3, 12]

B: 2D Array

Code:

```
# Function to create a 2D array
def create_2d_array(rows, columns):
    array = []
    for i in range(rows):
        row = []
        for j in range(columns):
            value = int(input(f"Enter element for row {i+1}, column {j+1}: "))
            row.append(value)
        array.append(row)
    return array

# Function to display a 2D array
```

```python
def display_2d_array(array):
    for row in array:
        for element in row:
            print(element, end=" ")
        print()

# Function to perform addition of two 2D arrays
def add_arrays(array1, array2):
    result = []
    for i in range(len(array1)):
        row = []
        for j in range(len(array1[0])):
            element = array1[i][j] + array2[i][j]
            row.append(element)
        result.append(row)
    return result

# Function to perform subtraction of two 2D arrays
def subtract_arrays(array1, array2):
    result = []
    for i in range(len(array1)):
        row = []
        for j in range(len(array1[0])):
            element = array1[i][j] - array2[i][j]
            row.append(element)
        result.append(row)
```

```
    return result

# Function to perform multiplication of two 2D arrays
def multiply_arrays(array1, array2):
    result = []
    for i in range(len(array1)):
        row = []
        for j in range(len(array2[0])):
            element = 0
            for k in range(len(array2)):
                element += array1[i][k] * array2[k][j]
            row.append(element)
        result.append(row)
    return result

# Function to perform transpose of a 2D array
def transpose_array(array):
    result = []
    for j in range(len(array[0])):
        row = []
        for i in range(len(array)):
            element = array[i][j]
            row.append(element)
        result.append(row)
    return result
```

```python
# Main program
rows = int(input("Enter the number of rows: "))
columns = int(input("\nEnter the number of columns: "))

print("\nEnter elements for the first array:")
array1 = create_2d_array(rows, columns)

print("\nEnter elements for the second array:")
array2 = create_2d_array(rows, columns)

print("\nFirst array:")
display_2d_array(array1)

print("\nSecond array:")
display_2d_array(array2)

# Addition
addition_result = add_arrays(array1, array2)
print("\nAddition result:")
display_2d_array(addition_result)

# Subtraction
subtraction_result = subtract_arrays(array1, array2)
print("\nSubtraction result:")
display_2d_array(subtraction_result)
```

```
# Multiplication
multiplication_result = multiply_arrays(array1, array2)
print("\nMultiplication result:")
display_2d_array(multiplication_result)

# Transpose
transpose_result = transpose_array(array1)
print("\nTranspose result:")
display_2d_array(transpose_result)
```

Output:

```
Enter the number of rows: 2

Enter the number of columns: 2

Enter elements for the first array:
Enter element for row 1, column 1: 1
Enter element for row 1, column 2: 2
Enter element for row 2, column 1: 3
Enter element for row 2, column 2: 4

Enter elements for the second array:
Enter element for row 1, column 1: 5
Enter element for row 1, column 2: 6
Enter element for row 2, column 1: 7
Enter element for row 2, column 2: 8
```

First array:

1 2

3 4

Second array:

5 6

7 8

Addition result:

6 8

10 12

Subtraction result:

-4 -4

-4 -4

Multiplication result:

19 22

43 50

Transpose result:

1 3

2 4

Code (Extra):

```
class TwoDArrayADT:
    def __init__(self):
        self.array = []

    def add_row(self, row):
        self.array.append(row)

    def delete_row(self, index):
        if 0 <= index < len(self.array):
            del self.array[index]
            print(f"Row {index} deleted.")
        else:
            print("Invalid row index.")

    def search_element(self, element):
        for row_idx, row in enumerate(self.array):
            if element in row:
                return True, row_idx
        return False, -1

    def sort_array(self):
        for row in self.array:
            row.sort()
        print("Array sorted.")

    def merge_arrays(self, other_array):
```

```
        self.array.extend(other_array)
        print("Arrays merged.")

    def display_array(self):
        for row in self.array:
            print(row)


# Example usage
if __name__ == "__main__":
    array_adt = TwoDArrayADT()

    array_adt.add_row([5, 10, 2])
    array_adt.add_row([8, 3, 12])
    array_adt.add_row([7, 1, 9])

    print("Initial array:")
    array_adt.display_array()

    print("\nSearching for element 10:")
    found, row_idx = array_adt.search_element(10)
    if found:
        print(f"Element found in row {row_idx}")
    else:
        print("Element not found.")
```

```
    print("\nSorting the array:")
    array_adt.sort_array()
    array_adt.display_array()

    other_array = [[11, 6, 4], [15, 8, 7]]
    array_adt.merge_arrays(other_array)
    print("\nArray after merging:")
    array_adt.display_array()

    array_adt.delete_row(1)
    print("\nArray after deleting row 1:")
    array_adt.display_array()
```

Output:

```
Initial array:
[5, 10, 2]
[8, 3, 12]
[7, 1, 9]

Searching for element 10:
Element found in row 0

Sorting the array:
Array sorted.
[2, 5, 10]
[3, 8, 12]
```

[1, 7, 9]

Arrays merged.

Array after merging:

[2, 5, 10]

[3, 8, 12]

[1, 7, 9]

[11, 6, 4]

[15, 8, 7]

Row 1 deleted.

Array after deleting row 1:

[2, 5, 10]

[1, 7, 9]

[11, 6, 4]

[15, 8, 7]

Practical – 2: Write a program to implement Singly Linked list with insertion, deletion, traversal operations

Code:

```
class Node:
    def _init_(self, data):
        self.data = data
```

```
        self.next = None

class SinglyLinkedList:
    def _init_(self):
        self.head = None

    def is_empty(self):
        return self.head is None

    def add_first(self, data):
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node

    def add_last(self, data):
        new_node = Node(data)
        if self.is_empty():
            self.head = new_node
        else:
            curr_node = self.head
            while curr_node.next is not None:
                curr_node = curr_node.next
            curr_node.next = new_node

    def remove_first(self):
        if self.is_empty():
```

```
        return
    else:
        self.head = self.head.next

def remove_last(self):
    if self.is_empty():
        return
    else:
        curr_node = self.head
        while curr_node.next.next is not None:
            curr_node = curr_node.next
        curr_node.next = None

def insert_after(self, data, after_data):
    if self.is_empty():
        return
    else:
        curr_node = self.head
        while curr_node.data != after_data:
            curr_node = curr_node.next
            if curr_node is None:
                return
        new_node = Node(data)
        new_node.next = curr_node.next
        curr_node.next = new_node
```

```
    def display(self):
        curr_node = self.head
        while curr_node is not None:
            print(curr_node.data)
            curr_node = curr_node.next

if _name_ == "_main_":
    sll = SinglyLinkedList()
    while True:
        print("\nSingly Linked List Menu:")
        print("1. Add First")
        print("2. Add Last")
        print("3. Remove First")
        print("4. Remove Last")
        print("5. Insert After")
        print("6. Display")
        print("7. Exit")

        choice = int(input("Enter your choice: "))

        if choice == 1:
            data = int(input("Enter the data to add at the beginning: "))
            sll.add_first(data)
        elif choice == 2:
            data = int(input("Enter the data to add at the end: "))
            sll.add_last(data)
```

```
        elif choice == 3:
            sll.remove_first()
        elif choice == 4:
            sll.remove_last()
        elif choice == 5:
            data = int(input("Enter the data to insert: "))
            after_data = int(input("Enter the data after which to insert: "))
            sll.insert_after(data, after_data)
        elif choice == 6:
            print("Singly Linked List:")
            sll.display()
        elif choice == 7:
            print("Exiting Program.")
            break
        else:
            print("Invalid choice. Please try again.")
```

Output:

Singly Linked List Menu:

1. Add First

2. Add Last

3. Remove First

4. Remove Last

5. Insert After

6. Display

7. Exit

Enter your choice: 1

Enter the data to add at the beginning: 12

Singly Linked List Menu:

1. Add First

2. Add Last

3. Remove First

4. Remove Last

5. Insert After

6. Display

7. Exit

Enter your choice: 1

Enter the data to add at the beginning: 13

Singly Linked List Menu:

1. Add First

2. Add Last

3. Remove First

4. Remove Last

5. Insert After

6. Display

7. Exit

Enter your choice: 2

Enter the data to add at the end: 15

Singly Linked List Menu:

1. Add First

2. Add Last

3. Remove First

4. Remove Last

5. Insert After

6. Display

7. Exit

Enter your choice: 2

Enter the data to add at the end: 16

Singly Linked List Menu:

1. Add First

2. Add Last

3. Remove First

4. Remove Last

5. Insert After

6. Display

7. Exit

Enter your choice: 3

Singly Linked List Menu:

1. Add First

2. Add Last

3. Remove First

4. Remove Last

5. Insert After

6. Display

7. Exit

Enter your choice: 4

Singly Linked List Menu:

1. Add First

2. Add Last

3. Remove First

4. Remove Last

5. Insert After

6. Display

7. Exit

Enter your choice: 5

Enter the data to insert: 17

Enter the data after which to insert: 15

Singly Linked List Menu:

1. Add First

2. Add Last

3. Remove First

4. Remove Last

5. Insert After

6. Display

7. Exit

Enter your choice: 6

Singly Linked List:

12

15

17

Singly Linked List Menu:

1. Add First

2. Add Last

3. Remove First

4. Remove Last

5. Insert After

6. Display

7. Exit

Enter your choice: 7

Exiting Program.

Practical – 3: Write a program to implement Doubly Linked list with insertion, deletion, traversal operations.

Code:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.prev = None
        self.next = None
```

```
class DoublyLinkedList:
    def __init__(self):
        self.head = None

    def insert_at_end(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node
            new_node.prev = current

    def delete_node(self, data):
        if not self.head:
            print("List is empty.")
            return

        if self.head.data == data:
            self.head = self.head.next
            if self.head:
                self.head.prev = None
            return
```

```
        current = self.head
        while current:
            if current.data == data:
                if current.next:
                    current.next.prev = current.prev
                if current.prev:
                    current.prev.next = current.next
                return
            current = current.next

        print(f"Element {data} not found in the list.")

    def display(self):
        current = self.head
        while current:
            print(current.data, end=" <-> ")
            current = current.next
        print("None")


def main():
    linked_list = DoublyLinkedList()

    while True:
        print("\nDoubly Linked List Menu:")
```

```
        print("1. Insert")
        print("2. Delete")
        print("3. Display")
        print("4. Exit")
        choice = int(input("Enter your choice: "))

        if choice == 1:
            data = int(input("Enter element to insert: "))
            linked_list.insert_at_end(data)
            print(f"Element {data} inserted.")
        elif choice == 2:
            data = int(input("Enter element to delete: "))
            linked_list.delete_node(data)
        elif choice == 3:
            print("Doubly Linked List:")
            linked_list.display()
        elif choice == 4:
            print("Exiting program.")
            break
        else:
            print("Invalid choice. Please choose again.")


if __name__ == "__main__":
    main()
```

Output:

Doubly Linked List Menu:

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 1

Enter element to insert: 19

Element 19 inserted.

Doubly Linked List Menu:

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 1

Enter element to insert: 30

Element 30 inserted.

Doubly Linked List Menu:

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 1

Enter element to insert: 25

Element 25 inserted.

Doubly Linked List Menu:

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 2

Enter element to delete: 25

Doubly Linked List Menu:

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 3

Doubly Linked List:

19 <-> 30 <-> None

Doubly Linked List Menu:

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 4

Exiting program.

Practical – 4: Write a program to implement Stack with insertion, deletion, traversal operations .

Code:

```
class Stack:
    def __init__(self):
        self.items = []

    def push(self, item):
        self.items.append(item)
        print(f"Pushed {item} onto the stack.")

    def pop(self):
        if not self.is_empty():
            popped_item = self.items.pop()
            print(f"Popped {popped_item} from the stack.")
            return popped_item
        else:
            print("Stack is empty.")
            return None

    def peek(self):
        if not self.is_empty():
            return self.items[-1]
        else:
            print("Stack is empty.")
```

```python
            return None

    def is_empty(self):
        return len(self.items) == 0

    def length(self):
        return len(self.items)

    def search(self, value):
        if value in self.items:
            print(f"{value} found at index {self.items.index(value)}.")
        else:
            print(f"{value} not found in the stack.")

    def display(self):
        if not self.is_empty():
            print("Stack contents:")
            for item in reversed(self.items):
                print(item)
        else:
            print("Stack is empty.")


def main():
    stack = Stack()
```

```
    while True:
        print("\nStack Menu:")
        print("1. Push\n2. Pop\n3. Peek\n4. Display\n5. Length\n6. Search 
value\n7. Exit")
        choice = int(input("Enter your choice: "))

        if choice == 1:
            item = input("Enter item to push: ")
            stack.push(item)
        elif choice == 2:
            popped_item = stack.pop()
            if popped_item is not None:
                print(f"Popped item: {popped_item}")
        elif choice == 3:
            peeked_item = stack.peek()
            if peeked_item is not None:
                print(f"Peeked item: {peeked_item}")
        elif choice == 4:
            stack.display()
        elif choice == 5:
            print(f"Stack length: {stack.length()}")
        elif choice == 6:
            value = input("Enter value to search: ")
            stack.search(value)
        elif choice == 7:
            print("Exiting program.")
            break
```

```
        else:
            print("Invalid choice. Please choose again.")


if __name__ == "__main__":
    main()
```

Output:

Stack Menu:

1. Push

2. Pop

3. Peek

4. Display

5. Length

6. Search Value

7. Exit

Enter your choice: 1

Enter item to push: 11

Pushed 11 onto the stack.

Stack Menu:

1. Push

2. Pop

3. Peek

4. Display

5. Length

6. Search Value

7. Exit

Enter your choice: 1

Enter item to push: 12

Pushed 12 onto the stack.

Stack Menu:

1. Push

2. Pop

3. Peek

4. Display

5. Length

6. Search Value

7. Exit

Enter your choice: 1

Enter item to push: 13

Pushed 13 onto the stack.

Stack Menu:

1. Push

2. Pop

3. Peek

4. Display

5. Length

6. Search Value

7. Exit

Enter your choice: 2

Popped 13 from the stack.

Popped item: 13

Stack Menu:

1. Push
2. Pop
3. Peek
4. Display
5. Length
6. Search Value
7. Exit

Enter your choice: 3

Peeked item: 12

Stack Menu:

1. Push
2. Pop
3. Peek
4. Display
5. Length
6. Search Value
7. Exit

Enter your choice: 4

Stack contents:

12

11

Stack Menu:

1. Push
2. Pop
3. Peek
4. Display
5. Length
6. Search Value
7. Exit

Enter your choice: 5

Stack length: 2

Stack Menu:

1. Push
2. Pop
3. Peek
4. Display
5. Length
6. Search Value
7. Exit

Enter your choice: 6

Enter value to search: 11

11 found at index 0.

Stack Menu:

1. Push

2. Pop

3. Peek

4. Display

5. Length

6. Search Value

7. Exit

Enter your choice: 7

Exiting program.

Practical – 5: Write a program to implement Queue with insertion, deletion, traversal operations.

Code:

```
class Queue:
    def __init__ (self):
        self.queue = []
        self.front = 0
        self.rear =- 1

    def enqueue(self,item):
        self.queue.append(item)
        self.rear += 1

    def dequeue(self):
        print("len self.queue=", len(self.queue))
        if len(self.queue)<0:
```

```
            return None
        if self.front <= self.rear:
            answer = self.queue.pop(self.front)
            self.rear -= 1
            return self.queue
        else:
            return None

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

    def size(self):
        return len(self.queue)

q = Queue()
q.enqueue(1)
q.enqueue(2)
q.enqueue(3)
q.enqueue(4)
q.enqueue(5)
q.enqueue(6)
q.display()
q.dequeue()
q.dequeue()
q.dequeue()
q.dequeue()
```

```
q.dequeue()
q.dequeue()
q.dequeue()
print("After removing an element")
q.display()
```

Output:

```
[1, 2, 3, 4, 5, 6]
len self.queue= 6
len self.queue= 5
len self.queue= 4
len self.queue= 3
len self.queue= 2
len self.queue= 1
len self.queue= 0
After removing an element
[]
```

Practical – 6: Write a program to implement Priority Queue with insertion, deletion, traversal

Operations.

Code:

```
import heapq


class PriorityQueue:
```

```
    def __init__ (self):
        self.queue = []
        self.index = 0

    def insert (self,item,priority):
        heapq.heappush(self.queue,(priority, self.index, item))
        self.index += 1

    def delete(self):
        if self.queue:
            priority, _, item = heapq.heappop(self.queue)
            return item
        else:
            return None

    def __len__(self):
        return len(self.queue)

    def traverse(self):
        for priority, _, item in self.queue:
            print(f"Priority: {priority}, Item: {item}")

#Example usage
pq = PriorityQueue()

pq.insert("Task 1", 5)
```

```python
pq.insert("Task 2", 3)
pq.insert("Task 3", 7)
pq.insert("Task 4", 1)

print("Priority Queue after insertion:")
pq.traverse()

deleted_item = pq.delete()
if deleted_item:
    print(f"Deleted item: {deleted_item}")

print("\nPriority Queue after deletion:")
pq.traverse()
```

Output:

```
Priority Queue after insertion:
Priority: 1, Item: Task 4
Priority: 3, Item: Task 2
Priority: 7, Item: Task 3
Priority: 5, Item: Task 1
Deleted item: Task 4

Priority Queue after deletion:
Priority: 3, Item: Task 2
Priority: 5, Item: Task 1
Priority: 7, Item: Task 3
```

Practical – 7: Write a program to implement Binary Tree with insertion, deletion, traversal operations

Code:

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


class BinaryTree:
    def __init__(self):
        self.root = None


    def insert(self, key):
        self.root = self._insert_recursive(self.root, key)


    def _insert_recursive(self, node, key):
        if node is None:
            return Node(key)
        if key < node.key:
            node.left = self._insert_recursive(node.left, key)
        elif key > node.key:
            node.right = self._insert_recursive(node.right, key)
        return node
```

```
def delete(self, key):
    self.root = self._delete_recursive(self.root, key)

def _delete_recursive(self, node, key):
    if node is None:
        return node
    if key < node.key:
        node.left = self._delete_recursive(node.left, key)
    elif key > node.key:
        node.right = self._delete_recursive(node.right, key)
    else:
        if node.left is None:
            return node.right
        elif node.right is None:
            return node.left
        temp = self._find_min(node.right)
        node.key = temp.key
        node.right = self._delete_recursive(node.right, temp.key)
    return node

def _find_min(self, node):
    current = node
    while current.left is not None:
        current = current.left
    return current
```

```
    def inorder_traversal(self):
        result = []
        self._inorder_recursive(self.root, result)
        return result

    def _inorder_recursive(self, node, result):
        if node:
            self._inorder_recursive(node.left, result)
            result.append(node.key)
            self._inorder_recursive(node.right, result)

if __name__ == "__main__":
    tree = BinaryTree()

    # Insert elements into the tree
    elements = [5, 3, 7, 2, 4, 6, 8]
    for element in elements:
        tree.insert(element)
    print("Inorder Traversal:")
    print(tree.inorder_traversal())

    # Delete an element from the tree
    delete_key = 4
    tree.delete(delete_key)
    print(f"Binary tree after deleting {delete_key}:")
    print(tree.inorder_traversal())
```

Output:

Inorder Traversal:

[2, 3, 4, 5, 6, 7, 8]

Binary tree after deleting 4:

[2, 3, 5, 6, 7, 8]

Practical – 8: Write a program to implement Huffman Coding

Code:

```
# Huffman Coding in python


string = 'BCAADDDCCACACAC'



# Creating tree nodes
class NodeTree(object):

    def __init__(self, left=None, right=None):
        self.left = left
        self.right = right

    def children(self):
        return (self.left, self.right)
```

```
    def nodes(self):
        return (self.left, self.right)

    def __str__(self):
        return '%s_%s' % (self.left, self.right)


# Main function implementing huffman coding
def huffman_code_tree(node, left=True, binString=''):
    if type(node) is str:
        return {node: binString}
    (l, r) = node.children()
    d = dict()
    d.update(huffman_code_tree(l, True, binString + '0'))
    d.update(huffman_code_tree(r, False, binString + '1'))
    return d


# Calculating frequency
freq = {}
for c in string:
    if c in freq:
        freq[c] += 1
    else:
        freq[c] = 1
```

```
freq = sorted(freq.items(), key=lambda x: x[1], reverse=True)

nodes = freq

while len(nodes) > 1:
    (key1, c1) = nodes[-1]
    (key2, c2) = nodes[-2]
    nodes = nodes[:-2]
    node = NodeTree(key1, key2)
    nodes.append((node, c1 + c2))

    nodes = sorted(nodes, key=lambda x: x[1], reverse=True)

huffmanCode = huffman_code_tree(nodes[0][0])

print(' Char | Huffman code ')
print('----------------------')
for (char, frequency) in freq:
    print(' %-4r |%12s' % (char, huffmanCode[char]))
```

Output:

```
Char | Huffman code
----------------------
 'C'  |          0
 'A'  |         11
 'D'  |        101
```

'B' | 100

Practical – 9: Implement Graph Insertion , Deletion and traversal .

Code:

```
class Node:
    def __init__ (self,data):
        self.data = data
        self.edge = []

    def add_edge(self,edge):
        self.edge.append(edge)

    def traverse(self):
        traversed = []
        queue = []
        for ed in self.edge:
            queue.append(ed)
        while len(queue)!=0:
            vertex = queue.pop()
            if vertex not in traversed:
                print(vertex.data)
                traversed.append(vertex)
                for ed in vertex.edge:
                    queue.append(ed)
```

```
graphroot = Node(3)

vert = Node(4)

vert.add_edge(graphroot)
graphroot.add_edge(vert)

vert2 = Node(6)
vert2.add_edge(graphroot)

graphroot.add_edge(vert)

vert3 = Node(5)
vert2.add_edge(vert)
vert.add_edge(vert2)

vert3.add_edge(vert2)
vert2.add_edge(vert3)

graphroot.traverse()
```

Output:

4

6

5

3

Practical – 10: Write a program to implement Travelling Salesman Problem

Code:

```
from sys import maxsize

from itertools import permutations

V = 4


def travellingSalesmanProblem(graph,s):

    vertex = []

    for i in range(V):

        if i != s:

            vertex.append(i)


    min_path = maxsize

    next_permutation = permutations(vertex)

    for i in next_permutation:

        current_pathweight = 0

        k = s

        for j in i:

            current_pathweight += graph[k][j]

            k = j

        current_pathweight += graph[k][s]

        min_path = min(min_path,current_pathweight)

    return min_path
```

```
if __name__ == "__main__":
    graph = [[0,10,15,20],[10,0,35,25],
             [15,35,0,30],[20,25,30,0]]
    s = 0
    print(travellingSalesmanProblem(graph,s))
```

Output:

80

Practical – 11: Write a program to create basic Hash Table for insertion, deletion, traversal operations(assume that there are no collisions)

Code:

```
class HashTable:
    def __init__(self,size):
        self.size = size
        self.table = [None] * size

    def hash_function(self,key):
        return key % self.size

    def insert(self,key,value):
        index = self.hash_function(key)
        self.table[index] = (key,value)
```

```
    def delete(self,key):
        index = self.hash_function(key)
        if self.table[index] and self.table[index][0]:
            self.table[index] = None

    def get(self,key):
        index = self.hash_function(key)
        if self.table[index] and self.table[index][0]:
            return self.table[index][1]
        return None

    def traverse(self):
        for entry in self.table:
            if entry:
                print(f"Key: {entry[0]}, Value: {entry[1]}")

#example usage
hash_table=HashTable(10)
k=int(input("enter key to insert:"))
v=input("enter value to insert:")
hash_table.insert(k,v)
k=int(input("enter key to insert:"))
v=input("enter value to insert:")
hash_table.insert(k,v)
k=int(input("enter key to insert:"))
v=input("enter value to insert:")
```

```
hash_table.insert(k,v)
#traversal
print("hash Table:")
hash_table.traverse()
k=int(input("enter key to Delete:"))
hash_table.delete(k)
#traversal after deletion
print("\nHash Table after deletion:")
hash_table.traverse()
```

Output:

```
enter key to insert:1
enter value to insert:2
enter key to insert:3
enter value to insert:4
enter key to insert:5
enter value to insert:6
hash Table:
Key: 1, Value: 2
Key: 3, Value: 4
Key: 5, Value: 6
enter key to Delete:1

Hash Table after deletion:
Key: 3, Value: 4
Key: 5, Value: 6
```

Practical – 12: Write a program to create hash table to handle collisions using overflow chaining.

Code:

```
class Node:
    def __init__(self, key, value):
        self.key = key
        self.value = value
        self.next = None


class HashTable:
    def __init__(self, size):
        self.size = size
        self.table = [None] * size


    def hash_function(self, key):
        return key % self.size


    def insert(self, key, value):
        index = self.hash_function(key)
        if self.table[index] is None:
            self.table[index] = Node(key, value)
        else:
            current = self.table[index]
            while current.next is not None:
                current = current.next
```

```
      current.next = Node(key, value)

  def delete(self, key):
    index = self.hash_function(key)
    if self.table[index] is not None:
      if self.table[index].key == key:
        self.table[index] = self.table[index].next
      else:
        current = self.table[index]
        while current.next is not None:
          if current.next.key == key:
            current.next = current.next.next
            return
          current = current.next

  def get(self, key):
    index = self.hash_function(key)
    current = self.table[index]
    while current is not None:
      if current.key == key:
        return current.value
      current = current.next
    return None

  def traverse(self):
    for index in range(self.size):
```

```
        current = self.table[index]
        while current is not None:
            print(f"Key: {current.key}, Value: {current.value}")
            current = current.next

# Example usage
hash_table = HashTable(10)

# Insertion
hash_table.insert(5, 'Apple')
hash_table.insert(2, 'Banana')
hash_table.insert(15, 'Cherry')
hash_table.insert(25, 'Grape')
hash_table.insert(12, 'Orange')

# Traversal
print("Hash Table:")
hash_table.traverse()

# Deletion
hash_table.delete(2)

# Traversal after deletion
print("\nHash Table after deletion:")
hash_table.traverse()
```

```
# Get value for a key
print("\nValue for key 12:", hash_table.get(12))
```

Output:

```
Hash Table:
Key: 2, Value: Banana
Key: 12, Value: Orange
Key: 5, Value: Apple
Key: 15, Value: Cherry
Key: 25, Value: Grape

Hash Table after deletion:
Key: 12, Value: Orange
Key: 5, Value: Apple
Key: 15, Value: Cherry
Key: 25, Value: Grape

Value for key 12: Orange
```