VIJAY M

22IT125   DAY -6

CODING PROBLEM AND PRACTICES

1.Minimum Path Sum

Time Complexity : O(n*n)

Solution:

```java
import java.util.ArrayList;
import java.util.List;

class GfG {
    static int
    minSumPathRec(ArrayList<ArrayList<Integer> > triangle,
            int i, int j) {



        if (i == triangle.size())
            return 0;



        return triangle.get(i).get(j)
            + Math.min(
            minSumPathRec(triangle, i + 1, j),
            minSumPathRec(triangle, i + 1, j + 1));
    }

    static int
    minSumPath(ArrayList<ArrayList<Integer> > triangle) {
        return minSumPathRec(triangle, 0, 0);
    }

    public static void main(String[] args) {
```

```java
        ArrayList<ArrayList<Integer> > triangle
                = new ArrayList<>();
        triangle.add(new ArrayList<>(List.of(2)));
        triangle.add(new ArrayList<>(List.of(3, 9)));
        triangle.add(new ArrayList<>(List.of(1, 6, 7)));

        System.out.println(minSumPath(triangle));
    }
}
```

Output:

2.Validate Binary Search Tree

Time Complexity : O(n2)

Solution :

```java
class Node {
    int data;
    Node left, right;
```

```java
    Node(int value) {
        data = value;
        left = right = null;
    }
}

class GfG {

    static int maxValue(Node node) {
        if (node == null) return Integer.MIN_VALUE;
        return
              Math.max(node.data,
                    Math.max(maxValue(node.left), maxValue(node.right)));
    }

    static int minValue(Node node) {
        if (node == null) return Integer.MAX_VALUE;
        return
              Math.min(node.data,
                    Math.min(minValue(node.left), minValue(node.right)));
    }

    static boolean isBST(Node node) {
        if (node == null) return true;

        if (node.left != null && maxValue(node.left) >= node.data)
            return false;

        if (node.right != null && minValue(node.right) <= node.data)
            return false;

        return isBST(node.left) && isBST(node.right);
    }
```
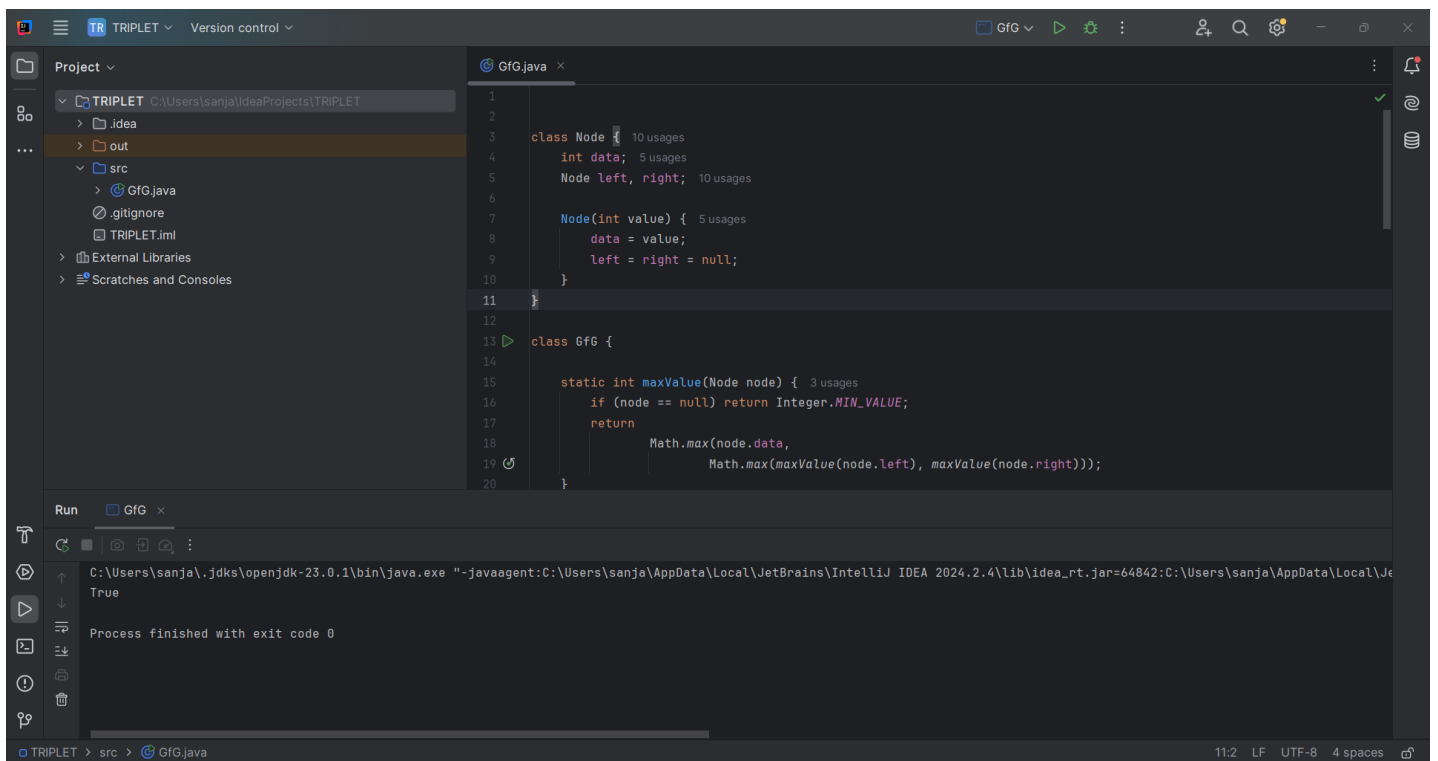
```java
public static void main(String[] args) {

    Node root = new Node(4);
    root.left = new Node(2);
    root.right = new Node(5);
    root.left.left = new Node(1);
    root.left.right = new Node(3);

    if (isBST(root)) {
        System.out.println("True");
    }
    else {
        System.out.println("False");
    }
}
}
```

Output:

3.Word Ladder

Time Complexity : O(N * M)

Solution:

```java
import java.util.*;

class GFG
{

    static int shortestChainLen(String start,
                         String target,
                         Set<String> D)
    {

        if(start == target)
            return 0;

        if (!D.contains(target))
            return 0;

        int level = 0, wordlength = start.length();

        Queue<String> Q = new LinkedList<>();
        Q.add(start);

        while (!Q.isEmpty())
        {

            ++level;

            int sizeofQ = Q.size();

            for (int i = 0; i < sizeofQ; ++i)
```

```java
        {
            char []word = Q.peek().toCharArray();
            Q.remove();

            for (int pos = 0; pos < wordlength; ++pos)
            {

                char orig_char = word[pos];

                for (char c = 'a'; c <= 'z'; ++c)
                {
                    word[pos] = c;

                    if (String.valueOf(word).equals(target))
                        return level + 1;

                    if (!D.contains(String.valueOf(word)))
                        continue;
                    D.remove(String.valueOf(word));

                    Q.add(String.valueOf(word));
                }

                word[pos] = orig_char;
            }
        }
    }

    return 0;
}
public static void main(String[] args)
{

    Set<String> D = new HashSet<String>();
```

```java
        D.add("poon");
        D.add("plee");
        D.add("same");
        D.add("poie");
        D.add("plie");
        D.add("poin");
        D.add("plea");
        String start = "toon";
        String target = "plea";
        System.out.print("Length of shortest chain is: "
                + shortestChainLen(start, target, D));
    }
}
```

Output:

4. Word Ladder 2

Time Complexity : O(N^2)

Solution:

```java
import java.util.*;
public class GFG
{
    public static class node
    {
        String word;
        int len;
        public node(String word, int len)
        {
            this.word = word;
            this.len = len;
        }
    }

    public static boolean isAdj(String a, String b)
    {
        int count = 0;
        for (int i = 0; i < a.length(); i++)
        {
            if (a.charAt(i) != b.charAt(i))
                count++;
        }
        if (count == 1)
            return true;
        return false;
    }

    public static int ladderLength(String beginWord, String endWord,
                    ArrayList<String> wordList)
    {
```

```java
Queue<node> q1 = new LinkedList<>();
Queue<node> q2 = new LinkedList<>();
HashMap<String, Integer> vis1 = new HashMap<>();
HashMap<String, Integer> vis2 = new HashMap<>();

node start = new node(beginWord, 1);
node end = new node(endWord, 1);

vis1.put(beginWord, 1);
q1.add(start);
vis2.put(endWord, 1);
q2.add(end);

while (q1.size() > 0 && q2.size() > 0)
{

    node curr1 = q1.remove();
    node curr2 = q2.remove();
    for (int i = 0; i < wordList.size(); i++)
    {

        if (isAdj(curr1.word,wordList.get(i)) &&
             vis1.containsKey(wordList.get(i)) == false)
        {

            node temp = new node(wordList.get(i),
                 curr1.len + 1);
            q1.add(temp);
            vis1.put(wordList.get(i), curr1.len + 1);


            if (temp.word.equals(endWord))
            {
```

```java
                return temp.len;
            }

            if (vis2.containsKey(temp.word))
            {
                return temp.len + vis2.get(temp.word) - 1;
            }
        }
    }

    for (int i = 0; i < wordList.size(); i++)
    {

        if (isAdj(curr2.word,wordList.get(i)) &&
                vis2.containsKey(wordList.get(i)) == false)
        {

            node temp = new node(wordList.get(i),
                    curr2.len + 1 );
            q2.add(temp);
            vis2.put(wordList.get(i), curr2.len + 1);

            if (temp.word.equals(beginWord))
            {
                return temp.len;
            }

            if (vis1.containsKey(temp.word))
            {
                return temp.len + vis1.get(temp.word) - 1;
            }
        }
    }
}
return 0;
```

```java
    }
    public static void main(String args[])
    {
        ArrayList<String> wordList = new ArrayList<>();
        wordList.add("poon");
        wordList.add("plee");
        wordList.add("same");
        wordList.add("poie");
        wordList.add("plie");
        wordList.add("poin");
        wordList.add("plea");
        String start = "toon";
        String target = "plea";
        System.out.println(ladderLength(start, target, wordList));
    }
}
```

Output:

5.Course Schedule

Time Complexity : O(n)

Solution:

```java
import java.util.ArrayList;
import java.util.Scanner;

class Course {
    String courseName;
    String courseCode;
    String instructor;
    String timeSlot;

    public Course(String courseName, String courseCode, String instructor, String timeSlot) {
        this.courseName = courseName;
        this.courseCode = courseCode;
        this.instructor = instructor;
        this.timeSlot = timeSlot;
    }

    public void displayCourseInfo() {
        System.out.println("Course Name: " + courseName);
        System.out.println("Course Code: " + courseCode);
        System.out.println("Instructor: " + instructor);
        System.out.println("Time Slot: " + timeSlot);
        System.out.println("-------------------------");
    }
}

public class CourseSchedule {
    private ArrayList<Course> courses;

    public CourseSchedule() {
```

```java
        courses = new ArrayList<>();
    }

    public void addCourse(String courseName, String courseCode, String instructor,
String timeSlot) {
        Course course = new Course(courseName, courseCode, instructor, timeSlot);
        courses.add(course);
    }

    public void displayAllCourses() {
        if (courses.isEmpty()) {
            System.out.println("No courses scheduled.");
        } else {
            for (Course course : courses) {
                course.displayCourseInfo();
            }
        }
    }

    public void searchCourseByCode(String courseCode) {
        boolean found = false;
        for (Course course : courses) {
            if (course.courseCode.equalsIgnoreCase(courseCode)) {
                course.displayCourseInfo();
                found = true;
                break;
            }
        }
        if (!found) {
            System.out.println("Course with code " + courseCode + " not found.");
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```java
CourseSchedule schedule = new CourseSchedule();

while (true) {
    System.out.println("Course Schedule Program");
    System.out.println("1. Add a Course");
    System.out.println("2. Display All Courses");
    System.out.println("3. Search Course by Code");
    System.out.println("4. Exit");
    System.out.print("Choose an option: ");
    int choice = scanner.nextInt();
    scanner.nextLine();
    switch (choice) {
        case 1:
            // Add a course
            System.out.print("Enter course name: ");
            String name = scanner.nextLine();
            System.out.print("Enter course code: ");
            String code = scanner.nextLine();
            System.out.print("Enter instructor name: ");
            String instructor = scanner.nextLine();
            System.out.print("Enter time slot: ");
            String timeSlot = scanner.nextLine();
            schedule.addCourse(name, code, instructor, timeSlot);
            System.out.println("Course added successfully.");
            break;

        case 2:
            schedule.displayAllCourses();
            break;

        case 3:
            System.out.print("Enter course code to search: ");
            String searchCode = scanner.nextLine();
            schedule.searchCourseByCode(searchCode);
            break;
```

```java
        case 4:
            System.out.println("Exiting...");
            scanner.close();
            return;

        default:
            System.out.println("Invalid choice. Please try again.");
            }
        }
    }
}
```

Output:

6.Design Tic Tae Toe

Time Complexity :

Solution:

```java
import java.util.Scanner;

public class TicTacToe {
    private char[][] board;
    private int
            currentPlayer;
    public TicTacToe()
    {
        board = new char[3][3];
        for (int i = 0; i < 3; ++i) {
            for (int j = 0; j < 3; ++j) {
                board[i][j] = ' ';
            }
        }
        currentPlayer = 1;
    }
    public void printBoard()
    {
        for (int i = 0; i < 3; ++i) {
            for (int j = 0; j < 3; ++j) {
                System.out.print(board[i][j]);
                if (j < 2) {
                    System.out.print(" | ");
                }
            }
            System.out.println();
            if (i < 2) {
                System.out.println("---------");
            }
        }
    }
```

```java
}
public boolean isBoardFull()
{
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            if (board[i][j] == ' ') {
                return false;
            }
        }
    }
    return true;
}
public boolean makeMove(int row, int column)
{
    if (row < 0 || row >= 3 || column < 0 || column >= 3
            || board[row][column] != ' ') {
        return false; // Invalid move
    }
    board[row][column]
            = (currentPlayer == 1) ? 'X' : 'O';
    currentPlayer
            = 3 - currentPlayer; // Switch player (1 to 2 or

    return true;
}
public boolean checkWinner()
{

    for (int i = 0; i < 3; ++i) {
        if (board[i][0] != ' '
                && board[i][0] == board[i][1]
                && board[i][1] == board[i][2]) {
            return true;
        }
        if (board[0][i] != ' '
```

```java
                && board[0][i] == board[1][i]
                && board[1][i] == board[2][i]) {
            return true;
        }
    }
    if (board[0][0] != ' ' && board[0][0] == board[1][1]
            && board[1][1] == board[2][2]) {
        return true;
    }
    if (board[0][2] != ' ' && board[0][2] == board[1][1]
            && board[1][1] == board[2][0]) {
        return true;
    }
    return false;
}
public static void main(String[] args)
{

    TicTacToe game = new TicTacToe();
    Scanner scanner = new Scanner(System.in);
    int row, column;
    while (!game.isBoardFull() && !game.checkWinner()) {
        game.printBoard();
        System.out.print(
                "Player " + game.currentPlayer
                        + ", enter your move (row and column): ");
        row = scanner.nextInt();
        column = scanner.nextInt();

        if (game.makeMove(row, column)) {
            System.out.println("Move successful!");
        }
        else {
            System.out.println(
                    "Invalid move. Try again.");
        }
```

```java
        }
        game.printBoard();
        if (game.checkWinner()) {
            System.out.println("Player "
                    + (3 - game.currentPlayer)
                    + " wins!");
        }
        else {
            System.out.println("It's a draw!");
        }
        scanner.close();
    }
}
```

Output: