**EXERCISE -** 11

**BACKTRACKING**

**Aim:** Write a java program to implement all pair shortest path problem.

**File name:** APSP.java

**Program:**

**public class APSP {**

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

**int[][] p = { { 0, 6,14},**

**{ 6, 0, 2},**

**{14, 2, 0}};**

**int n = p.length;**

**OptimalAPSP(p,n);**

**}**

**public static void OptimalAPSP(int a[][],int n){**

**for(int k = 0;k<n;k++){ //recurring : Dynamic programming**

**for(int i = 0; i<n;i++){**

**for(int j = 0; j<n;j++){**

**a[i][j] = Math.min(a[i][j],a[i][k]+a[k][j]);**

**}**

**}**

**}**

**System.out.println("All Pair Shortest Path is:");**

**for(int i = 0; i<n;i++){**

**for(int j = 0; j<n;j++){**

**System.out.print(a[i][j]+", ");**

**}**

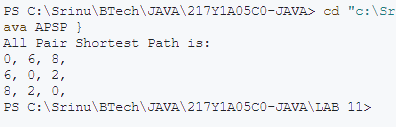
**System.out.println();**

**}**

**}**

**}**

**Output:**

****

**Aim:** Write a java program to implement graph coloring problem.

**File name:** GraphColoring.java

**Program:**

**import java.util.Arrays;**

**public class GraphColoring{**

**private int V;**

**private int[] colors;**

**private int[][] graph;**

**public GraphColoring(int[][] graph) {**

**this.graph = graph;**

**V = graph.length;**

**colors = new int[V];**

**Arrays.fill(colors, -1);**

**}**

**public void graphColoring() {**

**if (colorGraph(0))**

**System.out.println("Graph can be colored: " + Arrays.toString(colors));**

**else**

**System.out.println("Graph cannot be colored.");**

**}**

**private boolean isSafe(int v, int c) {**

**return Arrays.stream(graph[v]).noneMatch(neighbour -> c == colors[neighbour]);**

**}**

**private boolean colorGraph(int v) {**

**if (v == V) return true;**

**for (int c = 0; c < V; c++) {**

**if (isSafe(v, c)) {**

**colors[v] = c;**

**if (colorGraph(v + 1)) return true;**

**colors[v] = -1;**

**}**

**}**

**return false;**

**}**

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

**int[][] graph = {**

**{0, 1, 1, 1},**

**{1, 0, 1, 0},**

**{1, 1, 0, 1},**

**{1, 0, 1, 0}**

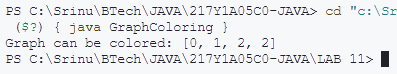
**};**

**new GraphColoring(graph).graphColoring();**

**}**

**}**

**Output:**

****

**Aim:** Write a java program to implement traveling scale person problem.

**File name:** TSPDynamic.java

**Program:**

**import java.util.Arrays;**

**public class TSPDynamic {**

**private int V;**

**private int[][] graph;**

**private int[][] dp;**

**public TSPDynamic(int[][] graph) {**

**this.graph = graph;**

**this.V = graph.length;**

**this.dp = new int[V][1 << V];**

**for (int[] row : dp) Arrays.fill(row, -1);**

**}**

**private int tsp(int current, int mask) {**

**if (mask == (1 << V) - 1) return graph[current][0];**

**if (dp[current][mask] != -1) return dp[current][mask];**

**int minCost = Integer.MAX\_VALUE;**

**for (int next = 0; next < V; next++) {**

**if (next != current && (mask & (1 << next)) == 0) {**

**int newMask = mask | (1 << next);**

**int cost = graph[current][next] + tsp(next, newMask);**

**minCost = Math.min(minCost, cost);**

**}**

**}**

**return dp[current][mask] = minCost;**

**}**

**public int minCost() {**

**return tsp(0, 1);**

**}**

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

**int[][] graph = {**

**{0, 10, 15, 20},**

**{10, 0, 35, 25},**

**{15, 35, 0, 30},**

**{20, 25, 30, 0}**

**};**

**TSPDynamic tspDP = new TSPDynamic(graph);**

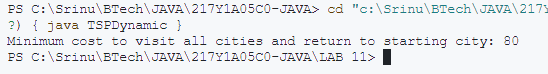
**int minCost = tspDP.minCost();**

**System.out.println("Minimum cost to visit all cities and return to starting city: " + minCost);**

**}**

**}**

**Output:**

****