**TSP:**

import java.util.\*;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.LinkedHashSet;

public class TSP {

public static int TSP(int[][] DP, int mark, int position, int number, int[][] adj, List<Integer> path) {

int completed\_visit = (1 << number) - 1;

if (mark == completed\_visit) {

path.add(0);

return adj[position][0];

}

if (DP[mark][position] != -1) {

return DP[mark][position];

}

int answer = Integer.MAX\_VALUE;

int nextCity = -1;

for (int city = 0; city < number; city++) {

if ((mark & (1 << city)) == 0) {

int newAnswer = adj[position][city] + TSP(DP, mark | (1 << city), city, number, adj, path);

if (newAnswer < answer) {

answer = newAnswer;

nextCity = city;

}

}

}

DP[mark][position] = answer;

if (nextCity != -1) {

path.add(nextCity);

}

return answer;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the number of locations:");

int number = sc.nextInt();

System.out.println("Enter the start location (0-indexed):");

int s = sc.nextInt();

int[][] adj = new int[number][number];

System.out.println("Enter the elements of the Adjacency matrix:");

for (int i = 0; i < number; i++) {

for (int j = 0; j < number; j++) {

adj[i][j] = sc.nextInt();

}

}

int[][] DP = new int[1 << number][number];

for (int i = 0; i < (1 << number); i++) {

for (int j = 0; j < number; j++) {

DP[i][j] = -1;

}

}

DP[(1 << number) - 1][0] = 0;

List<Integer> path = new ArrayList<>();

int minCost = TSP(DP, 1 << s, s, number, adj, path);

System.out.println("Minimum cost: " + minCost);

Set<Integer> hashSet = new LinkedHashSet<Integer>(path);

System.out.print("Optimal Path: ");

for (int city : hashSet) {

System.out.print((char)(city+65) + " ");

}

System.out.println((char)(s+65));

sc.close();

}

}

**Spell Checker:**

import java.util.Scanner;

public class SpellChecker {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the word to check for spelling errors:");

String str = sc.nextLine();

System.out.println("Enter the correct string to check with :");

String str2 = sc.nextLine();

System.out.println("Minimum number of edits required are :" +

spellcheck(str, str2));

sc.close();

}

public static int spellcheck(String s, String t) {

int lenS = s.length();

int lenT = t.length();

if (lenS == 0) {

return lenT;

}

if (lenT == 0) {

return lenS;

}

int[][] dp = new int[lenS + 1][lenT + 1];

for (int i = 0; i <= lenS; i++) {

dp[i][0] = i;

}

for (int j = 0; j <= lenT; j++) {

dp[0][j] = j;

}

for (int i = 1; i <= lenS; i++) {

for (int j = 1; j <= lenT; j++) {

if (s.charAt(i - 1) == t.charAt(j - 1)) {

dp[i][j] = dp[i - 1][j - 1];

} else {

dp[i][j] = 1 + Math.min(Math.min(dp[i - 1][j], dp[i][j -

1]), dp[i - 1][j - 1]);

}

}

}

return dp[lenS][lenT];

}

}

**Hamiltonian Cycle:**

import java.util.\*;

public class HamiltonianCycle{

    private static int N;

    private static int[][] graph;

    private static int[] path;

    private static boolean[] visited;

    private static Map<List<Integer>, Boolean> visitedPaths;

    private static int hamiltonianCycleCount = 0;

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of nodes:");

        N = sc.nextInt();

        graph = new int[N][N];

        path = new int[N];

        visited = new boolean[N];

        visitedPaths = new HashMap<>();

        System.out.println("Enter the adjacency matrix for the graph (0/1 indicating edges):");

        for (int i = 0; i < N; i++) {

            for (int j = 0; j < N; j++) {

                graph[i][j] = sc.nextInt();

            }

        }

        System.out.println("Enter the source node (1 to " + N + "):");

        int sourceNode = sc.nextInt();

        findHamiltonianCycle(0, sourceNode);

        if (hamiltonianCycleCount > 0) {

            System.out.println("Total Hamiltonian Cycles with source node " + sourceNode + ": " + hamiltonianCycleCount);

        } else {

            System.out.println("No Hamiltonian Cycle found with source node " + sourceNode);

        }

        sc.close();

    }

    private static void findHamiltonianCycle(int pos, int sourceNode) {

        if (pos == N) {

            if (graph[path[N - 1] - 1][path[0] - 1] == 1 && path[0] == sourceNode) {

                // Check if last node connects back to the source node to form a cycle

                List<Integer> currentPath = new ArrayList<>();

                for (int i = 0; i < N; i++) {

                    currentPath.add(path[i]);

                }

                if (!visitedPaths.containsKey(currentPath)) {

                    hamiltonianCycleCount++;

                    visitedPaths.put(currentPath, true);

                    System.out.print("Hamiltonian Cycle " + hamiltonianCycleCount + ": ");

                    for (int node : path) {

                        System.out.print(node + " ");

                    }

                    System.out.println(path[0]);

                }

            }

            return;

        }

        for (int v = 1; v <= N; v++) {

            if (isSafe(pos, v)) {

                path[pos] = v;

                visited[v - 1] = true;

                findHamiltonianCycle(pos + 1, sourceNode);

                visited[v - 1] = false;

                path[pos] = 0;

            }

        }

    }

    private static boolean isSafe(int pos, int v) {

        if (pos == 0) {

            return true;

        }

        if (!visited[v - 1] && graph[path[pos - 1] - 1][v - 1] == 1) {

            return true;

        }

        return false;

    }

}

**Graph Coloring:**

import java.util.\*;

public class GraphColoring {

    private int[][] a;

    private int[] Colors;

    private int numE;

    private int numC;

    public GraphColoring(int[][] a, int numC) {

        this.a = a;

        this.numE = a.length;

        this.numC = numC;

        this.Colors = new int[numE];

    }

    public boolean VColor(int exam, int color) {

        for (int i = 0; i < numE; i++) {

            if (a[exam][i] == 1 && Colors[i] == color) {

                return false;

            }

        }

        return true;

    }

    public boolean schedule(int exam) {

        if (exam == numE) {

            return true;

        }

        for (int color = 1; color <= numC; color++) {

            if (VColor(exam, color)) {

                Colors[exam] = color;

                if (schedule(exam + 1)) {

                    return true;

                }

                Colors[exam] = 0;

            }

        }

        return false;

    }

    public boolean scheduleExams() {

        return schedule(0);

    }

    public void print() {

        for (int i = 0; i < numE; i++) {

            System.out.println("Exam " + (i + 1) + " is scheduled at time slot " + Colors[i]);

        }

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of exams: ");

        int numE = scanner.nextInt();

        int[][] a = new int[numE][numE];

        System.out.println("Enter the adjacency matrix (1 for conflict, 0 for no conflict):");

        for (int i = 0; i < numE; i++) {

            for (int j = 0; j < numE; j++) {

                a[i][j] = scanner.nextInt();

            }

        }

        System.out.print("Enter the number of time slots (colors): ");

        int numC = scanner.nextInt();

        GraphColoring scheduler = new GraphColoring(a, numC);

        if (scheduler.scheduleExams()) {

            System.out.println("Exams scheduled successfully:");

            scheduler.print();

        } else {

            System.out.println("Failed to schedule exams.");

        }

        scanner.close();

    }

}