**public class NQueens {**

**private static boolean isSafe(boolean[][] board, int row, int col) {**

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

**if ( board[i][col] ) {**

**return false;}}**

**int maxLeft = Math.min(row, col);**

**for (int i = 1; i <= maxLeft; i++) {**

**if(board[row-i][col-i]) {**

**return false;}}**

**int maxRight = Math.min(row, board.length - col - 1);**

**for (int i = 1; i <= maxRight; i++) {**

**if(board[row-i][col+i]) {**

**return false;**

**}}return true;}**

**static int queens(boolean[][] board, int row) {**

**if (row == board.length) {**

**display(board);**

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

**return 1;}**

**int count = 0;**

**for (int col = 0; col < board.length; col++) {**

**if(isSafe(board, row, col)) {**

**board[row][col] = true;**

**count += queens(board, row + 1);**

**board[row][col] = false;**

**}}return count;}**

**private static void display(boolean[][] board) {**

**for(boolean[] row : board) {**

**for(boolean element : row) {**

**if (element) {System.out.print("Q ");**

**} else {System.out.print(". ");}}**

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

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

**int n = 8;**

**boolean[][] board = new boolean[n][n];**

**int countSol=queens(board,0);**

**if(countSol==0){**

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

**System.out.println("Sorry!! solution doesn't exist!!...");**

**}else{System.out.println(countSol);}}}**

**#FractionalKnapsack**

**import java.util.Arrays;**

**class FractionalKnapsack {**

**static class Item {**

**int weight, value;**

**Item(int v, int w) { value = v; weight = w; }}**

**public static double fractionalKnapsack(int W, Item[] items) {**

**Arrays.sort(items, (a, b) -> Double.compare((double) b.value / b.weight, (double) a.value / a.weight));**

**double totalValue = 0;**

**for (Item item : items) {**

**if (W == 0) break;**

**int weightToTake = Math.min(W, item.weight);**

**totalValue += weightToTake \* ((double) item.value / item.weight);**

**W -= weightToTake;}**

**return totalValue;}**

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

**Item[] items = { new Item(60, 10), new Item(100, 20), new Item(120, 30) };System.out.println(fractionalKnapsack(50, items)); }}**

**#FractionalKnapsack**

**import java.util.Arrays;**

**class FractionalKnapsack {**

**static class Item {**

**int weight, value;**

**Item(int v, int w) { value = v; weight = w; }**

**}**

**public static double fractionalKnapsack(int W, Item[] items) {**

**Arrays.sort(items, (a, b) -> Double.compare((double) b.value / b.weight, (double) a.value / a.weight));**

**double totalValue = 0;**

**for (Item item : items) {**

**if (W == 0) break;**

**int weightToTake = Math.min(W, item.weight);**

**totalValue += weightToTake \* ((double) item.value / item.weight);**

**W -= weightToTake;}**

**return totalValue;}**

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

**Item[] items = { new Item(60, 10), new Item(100, 20), new Item(120, 30) };System.out.println(fractionalKnapsack(50, items)); }}**

**public class NQueens {**

**private static boolean isSafe(boolean[][] board, int row, int col) {**

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

**if ( board[i][col] ) {**

**return false;}}**

**int maxLeft = Math.min(row, col);**

**for (int i = 1; i <= maxLeft; i++) {**

**if(board[row-i][col-i]) {**

**return false;}}**

**int maxRight = Math.min(row, board.length - col - 1);**

**for (int i = 1; i <= maxRight; i++) {**

**if(board[row-i][col+i]) {**

**return false;**

**}}return true;}**

**static int queens(boolean[][] board, int row) {**

**if (row == board.length) {**

**display(board);**

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

**return 1;}**

**int count = 0;**

**for (int col = 0; col < board.length; col++) {**

**if(isSafe(board, row, col)) {**

**board[row][col] = true;**

**count += queens(board, row + 1);**

**board[row][col] = false;**

**}}return count;}**

**private static void display(boolean[][] board) {**

**for(boolean[] row : board) {**

**for(boolean element : row) {**

**if (element) {System.out.print("Q ");**

**} else {System.out.print(". ");}}**

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

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

**int n = 8;**

**boolean[][] board = new boolean[n][n];**

**int countSol=queens(board,0);**

**if(countSol==0){**

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

**System.out.println("Sorry!! solution doesn't exist!!...");**

**}else{System.out.println(countSol);}}}**

**#MATRIX CHAIN MULTIPLICATION**

**import java.util.\*;**

**class MatrixChainOrder {**

**static String[][] dpSeq;**

**static int minMultRec(int arr[], int i, int j) {**

**if (i + 1 == j)**

**return 0;**

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

**for (int k = i + 1; k < j; k++) {**

**int curr = minMultRec(arr, i, k)**

**+ minMultRec(arr, k, j)**

**+ arr[i] \* arr[k] \* arr[j];**

**if (curr < res) {**

**res = curr;**

**dpSeq[i][j] = "(" + dpSeq[i][k] + dpSeq[k][j] + ")";**

**}}**

**return res;}**

**static int matrixMultiplication(int arr[]) {**

**int n = arr.length;**

**dpSeq = new String[n][n];**

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

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

**if (i + 1 == j) {**

**dpSeq[i][j] = "A" + i;**

**} else {**

**dpSeq[i][j] = "";**

**}}}**

**return minMultRec(arr, 0, n - 1);**

**}**

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

**int arr[] = { 1, 2, 3, 4, 5, 6 };**

**int res = matrixMultiplication(arr);**

**System.out.println("Minimum number of multiplications: " + res);**

**System.out.println("Optimal Parenthesization: " + dpSeq[0][arr.length - 1]);**

**}**

**}**

**#MATRIX CHAIN MULTIPLICATION**

**import java.util.\*;**

**class MatrixChainOrder {**

**static String[][] dpSeq;**

**static int minMultRec(int arr[], int i, int j) {**

**if (i + 1 == j)**

**return 0;**

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

**for (int k = i + 1; k < j; k++) {**

**int curr = minMultRec(arr, i, k)**

**+ minMultRec(arr, k, j)**

**+ arr[i] \* arr[k] \* arr[j];**

**if (curr < res) {**

**res = curr;**

**dpSeq[i][j] = "(" + dpSeq[i][k] + dpSeq[k][j] + ")";**

**}}**

**return res;}**

**static int matrixMultiplication(int arr[]) {**

**int n = arr.length;**

**dpSeq = new String[n][n];**

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

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

**if (i + 1 == j) {**

**dpSeq[i][j] = "A" + i;**

**} else {**

**dpSeq[i][j] = "";**

**}}}**

**return minMultRec(arr, 0, n - 1);**

**}**

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

**int arr[] = { 1, 2, 3, 4, 5, 6 };**

**int res = matrixMultiplication(arr);**

**System.out.println("Minimum number of multiplications: " + res);**

**System.out.println("Optimal Parenthesization: " + dpSeq[0][arr.length - 1]);**

**}**

**}**