|  |
| --- |
| // ACTIVITY SELECTION:  import java.util.List;  import java.util.Arrays;  import java.util.Comparator;  class Activity {  int start, end;  Activity(int start, int end) {  this.start = start;  this.end = end;  }  }  public class ActivitySelection {  public static void selectActivities(List<Activity> activities) {  // Sort by end time  activities.sort(Comparator.comparingInt(a -> a.end));  int lastEnd = 0;  for (Activity act : activities) {  if (act.start >= lastEnd) {  System.out.println("Selected: (" + act.start + ", " + act.end + ")");  lastEnd = act.end;  }  }  }  public static void main(String[] args) {  List<Activity> activities = Arrays.asList(  new Activity(1, 4),  new Activity(3, 5),  new Activity(0, 6),  new Activity(5, 7),  new Activity(8, 9),  new Activity(5, 9)  );  selectActivities(activities);  }  } |
| // CATALAN NUMBER:  public class CatalanNumber {  public static long catalan(int n) {  long[] dp = new long[n + 1];  dp[0] = dp[1] = 1;  for (int i = 2; i <= n; i++) {  dp[i] = 0;  for (int j = 0; j < i; j++) {  dp[i] += dp[j] \* dp[i - j - 1];  }  }  return dp[n];  }  public static void main(String[] args) {  int n = 5;  System.out.println("Catalan Number C(" + n + ") = " + catalan(n));  }  }  Applications of Catalan numbers  Catalan numbers have various applications in mathematics and computer science, including:  - Counting binary trees  - Counting triangulations of polygons  - Counting lattice paths  - Modeling recursive structures |
| //MIN COINS:  import java.util.Arrays;  public class MinCoins {  public static int minCoins(int[] coins, int amount) {  int[] dp = new int[amount + 1];  Arrays.fill(dp, Integer.MAX\_VALUE);  dp[0] = 0;  for (int coin : coins) {  for (int i = coin; i <= amount; i++) {  if (dp[i - coin] != Integer.MAX\_VALUE)  dp[i] = Math.min(dp[i], 1 + dp[i - coin]);  }  }  return dp[amount] == Integer.MAX\_VALUE ? -1 : dp[amount];  }  public static void main(String[] args) {  int[] coins = {1, 2, 5};  int amount = 11;  System.out.println("Minimum coins: " + minCoins(coins, amount));  }  } |
| // NQUEEN PROBLEM:  public class NQueenProblem {  final int N = 4;  void printSolution(int board[][])  {  for (int i = 0; i < N; i++) {  for (int j = 0; j < N; j++)  System.out.print(" " + board[i][j]  + " ");  System.out.println();  }  }  /\* A utility function to check if a queen can  be placed on board[row][col]. Note that this  function is called when "col" queens are already  placed in columns from 0 to col -1. So we need  to check only left side for attacking queens \*/  boolean isSafe(int board[][], int row, int col)  {  int i, j;  /\* Check this row on left side \*/  for (i = 0; i < col; i++)  if (board[row][i] == 1)  return false;  /\* Check upper diagonal on left side \*/  for (i = row, j = col; i >= 0 && j >= 0; i--, j--)  if (board[i][j] == 1)  return false;  /\* Check lower diagonal on left side \*/  for (i = row, j = col; j >= 0 && i < N; i++, j--)  if (board[i][j] == 1)  return false;  return true;  }  /\* A recursive utility function to solve N  Queen problem \*/  boolean solveNQUtil(int board[][], int col)  {  /\* base case: If all queens are placed  then return true \*/  if (col >= N)  return true;  /\* Consider this column and try placing  this queen in all rows one by one \*/  for (int i = 0; i < N; i++) {  /\* Check if the queen can be placed on  board[i][col] \*/  if (isSafe(board, i, col)) {  /\* Place this queen in board[i][col] \*/  board[i][col] = 1;  /\* recur to place rest of the queens \*/  if (solveNQUtil(board, col + 1) == true)  return true;  /\* If placing queen in board[i][col]  doesn't lead to a solution then  remove queen from board[i][col] \*/  board[i][col] = 0; // BACKTRACK  }  }  /\* If the queen can not be placed in any row in  this column col, then return false \*/  return false;  }  /\* This function solves the N Queen problem using  Backtracking. It mainly uses solveNQUtil () to  solve the problem. It returns false if queens  cannot be placed, otherwise, return true and  prints placement of queens in the form of 1s.  Please note that there may be more than one  solutions, this function prints one of the  feasible solutions.\*/  boolean solveNQ()  {  int board[][] = { { 0, 0, 0, 0 },  { 0, 0, 0, 0 },  { 0, 0, 0, 0 },  { 0, 0, 0, 0 } };  if (solveNQUtil(board, 0) == false) {  System.out.print("Solution does not exist");  return false;  }  printSolution(board);  return true;  }  public static void main(String args[])  {  NQueenProblem Queen = new NQueenProblem();  Queen.solveNQ();  }  } |
| // SUBSET SUM:  import java.util.ArrayList;  import java.util.List;  public class SubsetSum {  // Flag to check if there exists a subset with the given  // sum  static boolean flag = false;  // Print all subsets if there is at least one subset of  // set[] with the sum equal to the given sum  static void printSubsetSum(int i, int n, int[] set,  int targetSum,  List<Integer> subset)  {  // If targetSum is zero, then there exists a subset.  if (targetSum == 0) {  // Prints a valid subset  flag = true;  System.out.print("[ ");  for (int j = 0; j < subset.size(); j++) {  System.out.print(subset.get(j) + " ");  }  System.out.print("]");  return;  }  if (i == n) {  // Return if we have reached the end of the  // array  return;  }  // Not considering the current element  printSubsetSum(i + 1, n, set, targetSum, subset);  // Consider the current element if it is less than  // or equal to the targetSum  if (set[i] <= targetSum) {  // Push the current element in the subset  subset.add(set[i]);  // Recursive call for considering the current  // element  printSubsetSum(i + 1, n, set,  targetSum - set[i], subset);  // Pop-back element after the recursive call to  // restore the subset's original configuration  subset.remove(subset.size() - 1);  }  }  public static void main(String[] args)  {  // Test case 1  int[] set1 = { 1, 2, 1 };  int sum1 = 3;  int n1 = set1.length;  List<Integer> subset1 = new ArrayList<>();  System.out.println("Output 1:");  printSubsetSum(0, n1, set1, sum1, subset1);  System.out.println();  flag = false;  // Test case 2  int[] set2 = { 3, 34, 4, 12, 5, 2 };  int sum2 = 30;  int n2 = set2.length;  List<Integer> subset2 = new ArrayList<>();  System.out.println("Output 2:");  printSubsetSum(0, n2, set2, sum2, subset2);  if (!flag) {  System.out.println("There is no such subset");  }  }  } |
|  |