Dynamic Programming

1. Catalan's Number

```java
import java.util.Scanner;

public class CatalansNumber {
    //RECURSION - O(2^n)
    public static int catalanRec(int n) {
        if (n == 0 || n == 1) {
            return 1; // Base case
        }
        int ans = 0; // Cn
        for (int i = 0; i <= n - 1; i++) { // Correct loop range
            ans += catalanRec(i) * catalanRec(n - i - 1);
        }
        return ans;
    }
    //MEMOIZATION - O(n)
    public static int catalanMemo(int n, int dp[]){
        if (n==0 || n==1) {
            return 1;
        }
        if (dp[n] != -1) {
            return dp[n];
        }
        int ans = 0;
        for(int i=0; i<n; i++){
            ans += catalanMemo(i, dp) * catalanMemo(n-i-1, dp);
        }
        return dp[n] = ans;
    }
    //TABULATION - O(n*n)
    public static int catalanTab(int n){
        int dp[] = new int[n+1];
        dp[0] = 1;
        dp[1] = 1;
        for(int i=2; i<=n; i++){
            for(int j=0; j<i; j++){
                dp[i] += dp[j] * dp[i-j-1];
            }
        }
        return dp[n];
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter a Number: ");
        int n = sc.nextInt();

        System.out.println("Catalan Number using Recursion: " + catalanRec(n));

        int dp[] = new int[n+1];
        for(int i=0; i<=n; i++){
            dp[i] = -1;
        }
        System.out.println("Catalan Number using Memoization: " + catalanMemo(n, dp));
        System.out.println("Catalan Number using Tabulation: " + catalanTab(n));

        sc.close();
    }
}
```

2. Climbing Stairs

```
import java.util.Arrays;

public class ClimbingStairs {

    // RECURSION - O(2^n)
    public static int countWaysRec(int n){
        if (n == 0) {
            return 1;
        }
        if (n < 0) {
            return 0;
        }
        return countWaysRec(n-1) + countWaysRec(n-2);
    }
    // MEMOIZATION - O(n)
    public static int countWaysMemo(int n, int ways[]){
        if (n == 0) {
            return 1;
        }
        if (n < 0) {
            return 0;
        }
        if (ways[n] != -1) {
            return ways[n];
        }
        ways[n] = countWaysMemo(n-1, ways) + countWaysMemo(n-2, ways);
        return ways[n];
    }
    // TABULATION - O(n)
    public static int countWaysTab(int n){
        int dp[] = new int[n+1];
        dp[0] = 1;
        for(int i=1; i<=n; i++){
            if (i == 1) {
                dp[i] = dp[i-1] + 0;
            } else {
                dp[i] = dp[i-1] + dp[i-2];
            }
        }
        return dp[n];
    }
    public static void main(String[] args) {
        int n=4;
        int ways [] = new int[n+1];
        Arrays.fill(ways, -1);
        System.out.println(countWaysTab(n));
    }
}
```

3. Coin Change

```java
public class CoinChange {
    public static int coinChange(int coins[], int sum){
        int n = coins.length;
        int dp[][] = new int[n+1][sum+1];
        //intialise - sum is 0
        // i->coins, j->sum/change
        for(int i=0; i<n+1; i++){
            dp[i][0] = 1;
        }
        for(int j=0; j<sum+1; j++){
            dp[0][j] = 0;
        }
        //O(n*sum)
        for (int i = 1; i < n+1; i++) {
            for (int j = 1; j < sum + 1; j++) {
                if (coins[i-1] <= j) {
                    dp[i][j] = dp[i][j-coins[i-1]] + dp[i-1][j];
                } else {
                    dp[i][j] = dp[i-1][j];
                }
            }
        }
        return dp[n][sum];
    }
    public static void main(String[] args) {
        int coins[] = {2, 5, 3, 6};
        int sum = 10;   //ans  = 5 i.e, {2, 2, 2, 2, 2} {2, 2, 3, 3} {2, 2, 6} {2, 3, 5} {5, 5}
        System.out.println("No. of way to give Coins Change is: " + coinChange(coins, sum));
    }
}
```

4. Counting BST

```java
public class CountingBSTs {
    public static int countBST(int n){
        int dp[] = new int[n+1];
        dp[0] = 1;
        dp[1] = 1;
        for(int i=2; i<n+1; i++){
            // Ci -> BST (i nodes) -> dp[i]
            for(int j=0; j<i; j++){
                int left = dp[j];
                int right = dp[i-j-1];
                dp[i] += left * right;
            }
        }
        return dp[n];
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a Number of Nodes: ");
        int n = sc.nextInt();
        System.out.println("Number of BSTs possible for given Nodes: " + countBST(n));
    }
}
```

5. Edit Distance

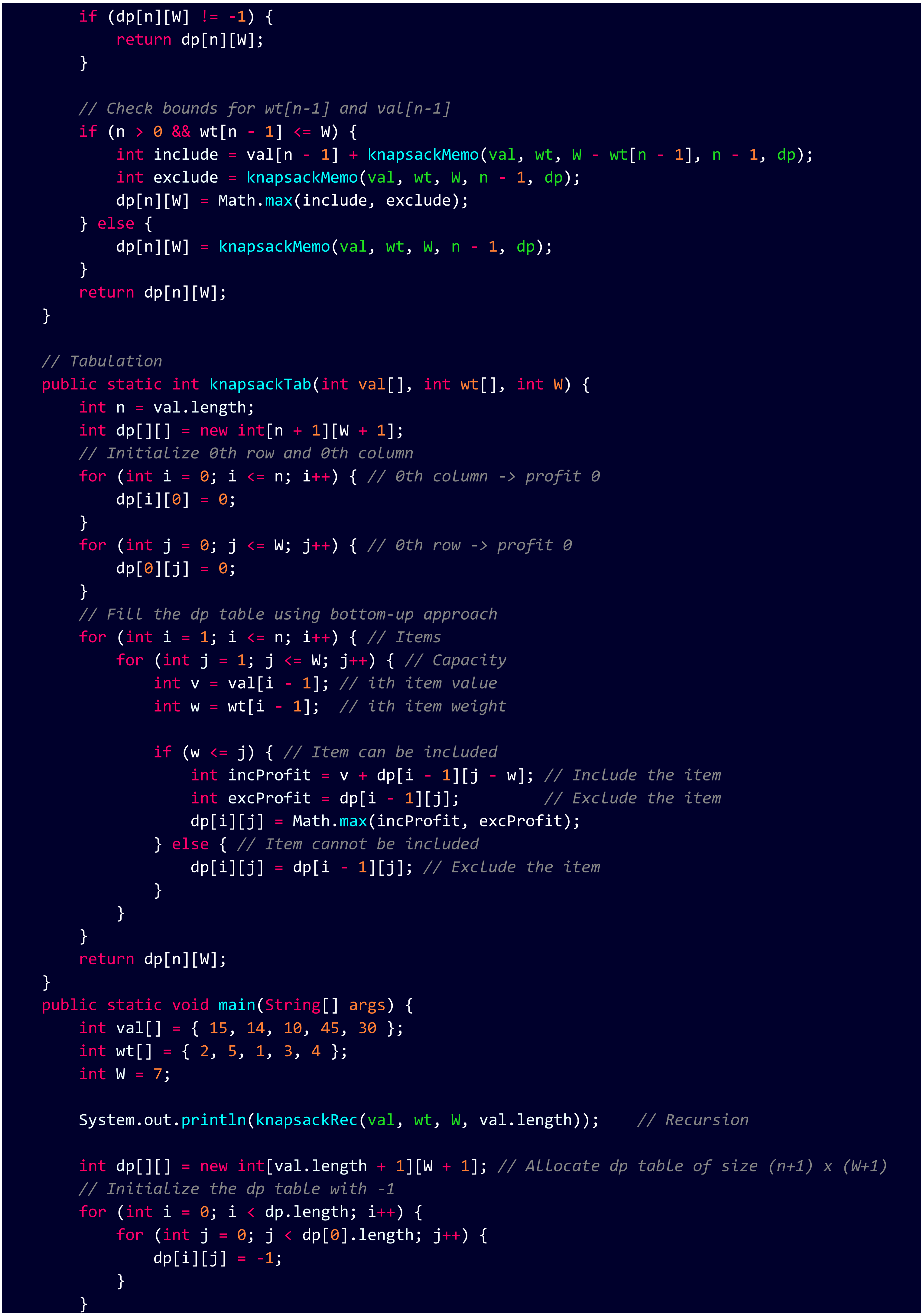
```java
public class EditDistance {
    public static int editDistance(String str1, String str2){
        int n = str1.length();
        int m = str2.length();
        int dp[][] = new int[n+1][m+1];
        for(int i=0; i<n+1; i++){
            for(int j=0; j<m+1; j++){
                if (i==0) {
                    dp[i][j] = j;
                }
                if (j == 0) {
                    dp[i][j] = i;
                }
            }
        }
        for(int i=1; i<n+1; i++){
            for(int j=1; j<m+1; j++){
                if (str1.charAt(i-1) == str2.charAt(j-1)) {
                    dp[i][j] = dp[i-1][j-1];
                } else {
                    int add = dp[i][j-1] + 1;
                    int del = dp[i-1][j] + 1;
                    int rep = dp[i-1][j-1] + 1;
                    dp[i][j] = Math.min(add, Math.min(del, rep));
                }
            }
        }
        return dp[n][m];
    }
    public static void main(String[] args) {
        String str1 = "intention";
        String str2 = "execution";
        System.out.println(editDistance(str1, str2));
    /*
     * intention -> inention (remove 't')
     * inention -> enention (replace 'i' with 'e')
     * enention -> exention (replace 'n' with x)
     * exention -> exection (replace 'n' with 'c')
     * exection -> execution (insert 'u')
     */
    }
}
```

6. Fibonacci

```java
public class Fibonacci {

    // RECURSION - O(2^n)
    public static int fibRec(int n){
        if (n == 0 || n == 1) {
            return n;
        }
        return fibRec(n-1) + fibRec(n-2);
    }
    // MEMOIZATION - O(n)
    public static int fibMemo(int n, int f[]){
        if (n==0 || n==1) {
            return n;
        }
        if (f[n] != 0) {
            return f[n];
        }
        f[n] = fibMemo(n-1, f) + fibMemo(n-2, f);
        return f[n];
    }

    // TABULATION - O(n)
    public static int fibTab(int n){
        int dp[] = new int[n+1];
        dp[0] = 0;
        dp[1] = 1;
        for(int i=2; i<=n; i++){
            dp[i] = dp[i-1] + dp[i-2];
        }
        return dp[n];
    }

    public static void main(String[] args) {
        int n=6;
        int f[] = new int[n+1]; //it's for memoization i.e, har i pe 0 store ho jayega
        System.out.println(fibMemo(n, f));
    }
}
```

7. Knapsack

```
public class Knapsack {
    // Recursion - O(2^n)
    public static int knapsackRec(int val[], int wt[], int W, int n) {
        if (W == 0 || n == 0) {
            return 0;
        }
        if (wt[n - 1] <= W) { // valid
            // include
            int ans1 = val[n - 1] + knapsackRec(val, wt, W - wt[n - 1], n-1);
            // exclude
            int ans2 = knapsackRec(val, wt, W, n - 1);
            return Math.max(ans1, ans2);
        } else {
            return knapsackRec(val, wt, W, n - 1);
        }
    }

    // Memoization - O(n*W)
    public static int knapsackMemo(int val[], int wt[], int W, int n, int dp[][]) {
        if (W == 0 || n == 0) {
            return 0;
        }

        // Check if result is already calculated
```

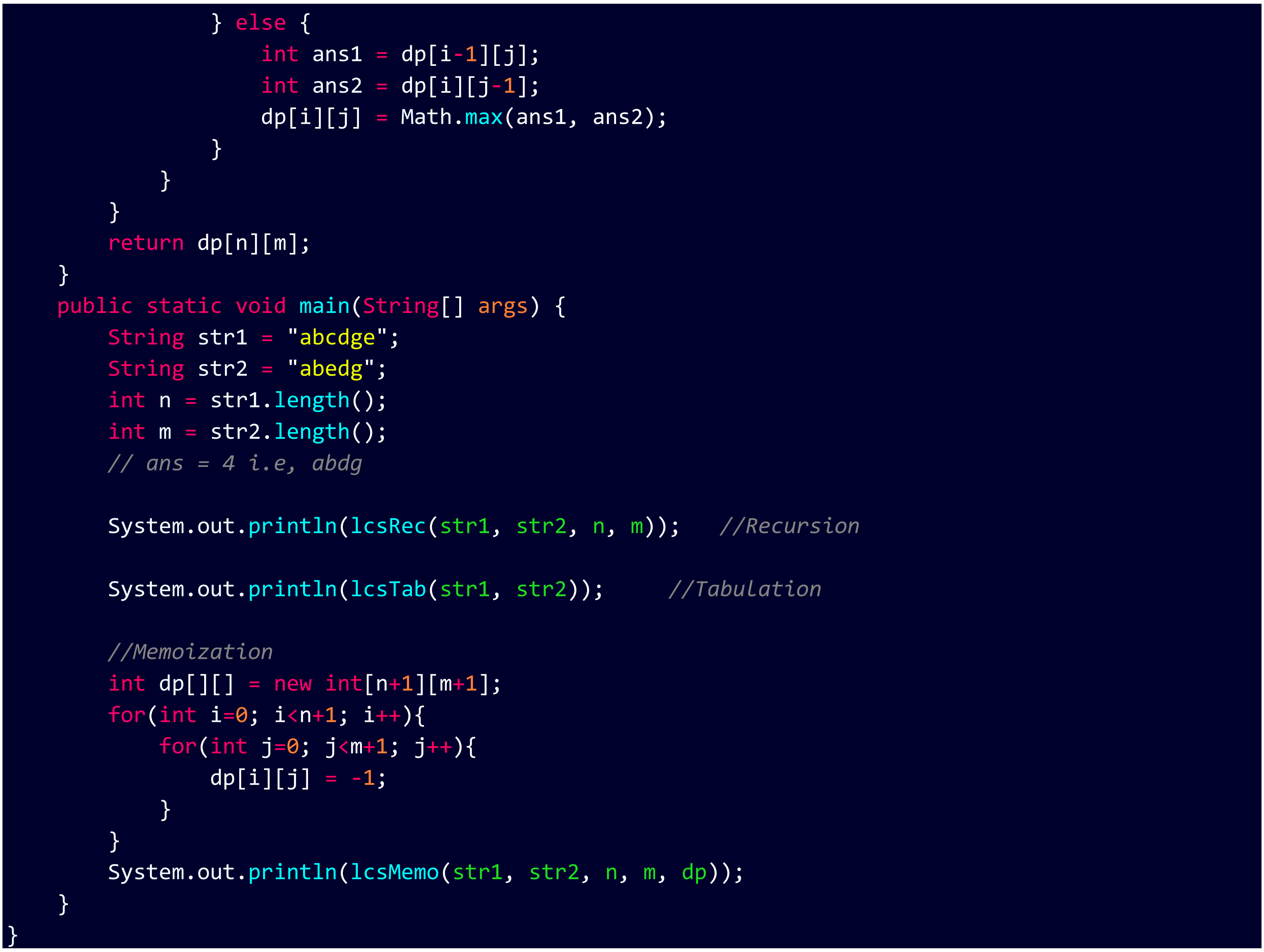


```
        if (dp[n][W] != -1) {
            return dp[n][W];
        }

        // Check bounds for wt[n-1] and val[n-1]
        if (n > 0 && wt[n - 1] <= W) {
            int include = val[n - 1] + knapsackMemo(val, wt, W - wt[n - 1], n - 1, dp);
            int exclude = knapsackMemo(val, wt, W, n - 1, dp);
            dp[n][W] = Math.max(include, exclude);
        } else {
            dp[n][W] = knapsackMemo(val, wt, W, n - 1, dp);
        }
        return dp[n][W];
    }

    // Tabulation
    public static int knapsackTab(int val[], int wt[], int W) {
        int n = val.length;
        int dp[][] = new int[n + 1][W + 1];
        // Initialize 0th row and 0th column
        for (int i = 0; i <= n; i++) { // 0th column -> profit 0
            dp[i][0] = 0;
        }
        for (int j = 0; j <= W; j++) { // 0th row -> profit 0
            dp[0][j] = 0;
        }
        // Fill the dp table using bottom-up approach
        for (int i = 1; i <= n; i++) { // Items
            for (int j = 1; j <= W; j++) { // Capacity
                int v = val[i - 1]; // ith item value
                int w = wt[i - 1];  // ith item weight

                if (w <= j) { // Item can be included
                    int incProfit = v + dp[i - 1][j - w]; // Include the item
                    int excProfit = dp[i - 1][j];         // Exclude the item
                    dp[i][j] = Math.max(incProfit, excProfit);
                } else { // Item cannot be included
                    dp[i][j] = dp[i - 1][j]; // Exclude the item
                }
            }
        }
        return dp[n][W];
    }
    public static void main(String[] args) {
        int val[] = { 15, 14, 10, 45, 30 };
        int wt[] = { 2, 5, 1, 3, 4 };
        int W = 7;

        System.out.println(knapsackRec(val, wt, W, val.length));    // Recursion

        int dp[][] = new int[val.length + 1][W + 1]; // Allocate dp table of size (n+1) x (W+1)
        // Initialize the dp table with -1
        for (int i = 0; i < dp.length; i++) {
            for (int j = 0; j < dp[0].length; j++) {
                dp[i][j] = -1;
            }
        }
```

```java
        System.out.println(knapsackMemo(val, wt, W, val.length, dp));

        // Tabulation - O(n*W)
        int maxProfit = knapsackTab(val, wt, W);
        System.out.println(maxProfit);
    }
}
```

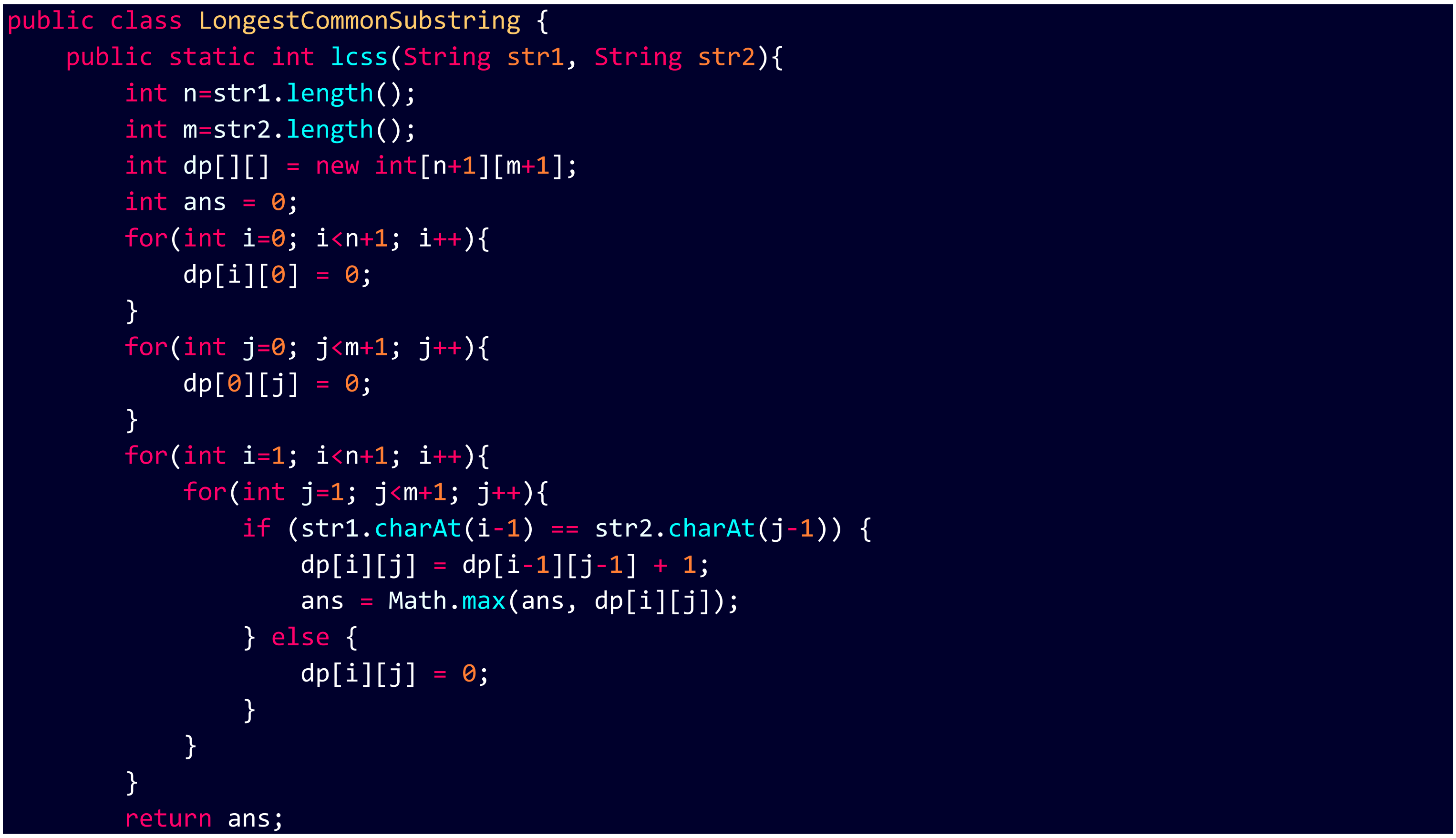
8. Longest Common Subsequence

```java
public class LongestCommonSubsequence {
    //RECURSION
    public static int lcsRec(String str1, String str2, int n, int m){
        if (n==0 || m==0) {
            return 0;
        }
        if (str1.charAt(n-1) == str2.charAt(m-1)) {
            return lcsRec(str1, str2, n-1, m-1) + 1;
        } else {
            int ans1 = lcsRec(str1, str2, n-1, m);
            int ans2 = lcsRec(str1, str2, n, m-1);
            return Math.max(ans1, ans2);
        }
    }
    //MEMOIZATION
    public static int lcsMemo(String str1, String str2, int n, int m, int dp[][]){
        if (n==0 || m==0) {
            return 0;
        }
        if (dp[n][m] != -1) {
            return dp[n][m];
        }
        if (str1.charAt(n-1) == str2.charAt(m-1)) {
            return dp[n][m] = lcsMemo(str1, str2, n-1, m-1, dp) + 1;
        } else {
            int ans1 = lcsMemo(str1, str2, n-1, m, dp);
            int ans2 = lcsMemo(str1, str2, n, m-1, dp);
            return dp[n][m] = Math.max(ans1, ans2);
        }
    }
    //TABULATION
    public static int lcsTab(String str1, String str2){
        int n = str1.length();
        int m = str2.length();
        int dp[][] = new int[n+1][m+1];
        for(int i=0; i<n+1; i++){
            for(int j=0; j<m+1; j++){
                if (i==0 || j==0) {
                    dp[i][j] = 0;
                }
            }
        }
        for(int i=1; i<n+1; i++){
            for(int j=1; j<m+1; j++){
                if (str1.charAt(i-1) == str2.charAt(j-1)) {
                    dp[i][j] = dp[i-1][j-1] + 1;
```



```java
                } else {
                    int ans1 = dp[i-1][j];
                    int ans2 = dp[i][j-1];
                    dp[i][j] = Math.max(ans1, ans2);
                }
            }
        }
        return dp[n][m];
    }
    public static void main(String[] args) {
        String str1 = "abcdge";
        String str2 = "abedg";
        int n = str1.length();
        int m = str2.length();
        // ans = 4 i.e, abdg

        System.out.println(lcsRec(str1, str2, n, m));   //Recursion

        System.out.println(lcsTab(str1, str2));     //Tabulation

        //Memoization
        int dp[][] = new int[n+1][m+1];
        for(int i=0; i<n+1; i++){
            for(int j=0; j<m+1; j++){
                dp[i][j] = -1;
            }
        }
        System.out.println(lcsMemo(str1, str2, n, m, dp));
    }
}
```

9. Longest Common Substring



```java
public class LongestCommonSubstring {
    public static int lcss(String str1, String str2){
        int n=str1.length();
        int m=str2.length();
        int dp[][] = new int[n+1][m+1];
        int ans = 0;
        for(int i=0; i<n+1; i++){
            dp[i][0] = 0;
        }
        for(int j=0; j<m+1; j++){
            dp[0][j] = 0;
        }
        for(int i=1; i<n+1; i++){
            for(int j=1; j<m+1; j++){
                if (str1.charAt(i-1) == str2.charAt(j-1)) {
                    dp[i][j] = dp[i-1][j-1] + 1;
                    ans = Math.max(ans, dp[i][j]);
                } else {
                    dp[i][j] = 0;
                }
            }
        }
        return ans;
    }
    public static void main(String[] args) {
        String str1 = "ABCDGH";
        String str2 = "ACDGHR";
        System.out.println("Longest Common Substring: " + lcss(str1, str2));    //CDGH
    }
}
```

10. Longest Increasing Subsequence

```java
import java.util.Arrays;
import java.util.HashSet;

public class LongestIncreasingSubsequence {
    public static int lis(int arr1[]) {
        HashSet<Integer> set = new HashSet<>();
        for (int i = 0; i < arr1.length; i++) {
            set.add(arr1[i]);
        }
        int arr2[] = new int[set.size()];
        int i = 0;
        for (int num : set) {
            arr2[i] = num;
            i++;
        }
        Arrays.sort(arr2);
        return lcs(arr1, arr2);
    }
    public static int lcs(int arr1[], int arr2[]) {
        int n = arr1.length;
        int m = arr2.length;
        int dp[][] = new int[n + 1][m + 1];
        for (int i = 0; i < n + 1; i++) {
            dp[i][0] = 0;
        }
        for (int j = 0; j < m + 1; j++) {
            dp[0][j] = 0;
        }
        for (int i = 1; i < n + 1; i++) {
            for (int j = 1; j < m + 1; j++) {
                if (arr1[i - 1] == arr2[j - 1]) {
                    dp[i][j] = dp[i - 1][j - 1] + 1;
                } else {
                    int ans1 = dp[i - 1][j];
                    int ans2 = dp[i][j - 1];
                    dp[i][j] = Math.max(ans1, ans2);
                }
            }
        }
        return dp[n][m];
    }
    public static void main(String[] args) {
        int arr[] = {50, 3, 10, 7, 40, 80};
        System.out.println(lis(arr));   // 4 - 3, 7, 40, 80
    }
}
```

11. Minimum Array Jumps

```java
import java.util.Arrays;
import java.util.Scanner;

public class MinArrayJumps {
    public static int minJumps(int nums[]){
        int n = nums.length;
        int dp[] = new int[n];
        Arrays.fill(dp, -1);
        dp[n-1] = 0;
        for(int i=n-2; i>=0; i--){
            int steps = nums[i];
            int ans = Integer.MAX_VALUE;
            for(int j=i+1; j<=i+steps && j<n; j++){
                if (dp[j] != -1) {
                    ans = Math.min(ans, dp[j] + 1);
                }
            }
            if (ans != Integer.MAX_VALUE) {
                dp[i] = ans;
            }
        }
        return dp[0];
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter size of an array: ");
        int n = sc.nextInt();
        int nums[] = new int[n];
        System.out.println("Enter elements of an array: ");
        for(int i=0; i<n; i++){
            nums[i] = sc.nextInt();
        }
        System.out.println("Minimum number of jumps to reach the end of array is: " +
minJumps(nums));

        sc.close();
    }
}
```

12. Minimum Partitioning

```java
import java.util.Scanner;

public class MinimumPartitioning {
    public static int minPartiton(int arr[]){
        int n = arr.length;
        int sum = 0;
        for(int i=0; i<arr.length; i++){
            sum += arr[i];
        }
        int W = sum/2;
        int dp[][] = new int[n+1][W+1];
        // bottom up
        for (int i = 1; i < n+1; i++) {
            for (int j = 1; j < W+1; j++) {
                if (arr[i-1] <= j) {
                    dp[i][j] = Math.max(arr[i-1] + dp[i-1][j-arr[i-1]], dp[i-1][j]);
                } else {
                    dp[i][j] = dp[i-1][j];
                }
            }
        }
        int sum1 = dp[n][W];
        int sum2 = sum - sum1;
        return Math.abs(sum1 - sum2);
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of array: ");
        int n = sc.nextInt();
        int arr[] = new int[n];
        System.out.println("Enter the elements of array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.println("Minimum partitioning difference: " + minPartiton(arr));

        sc.close();
    }
}
```

13. Mountain Ranges

```
import java.util.Scanner;

public class MountainRanges {
    public static int mountainRange(int n){
        int dp[] = new int[n+1];
        dp[0] = 1;
        dp[1] = 1;
        for(int i=2; i<n+1; i++){
            for (int j = 0; j < i; j++) {
                int inside = dp[j];
                int outside = dp[i-j-1];
                dp[i] += inside * outside;
            }
        }
        return dp[n];
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter number of pairs of strokes: ");
        int n = sc.nextInt();
        System.out.println("Number of Mountains can be Made: " + mountainRange(n));
    }
}
```

14. Rod Cutting

```
public class RodCutting {
    public static int rodCutting(int length[], int price[], int totRod){
        int n=price.length;
        int dp[][] = new int[n+1][totRod+1];
        for(int i=0; i<n+1; i++){
            for(int j=0; j<totRod+1; j++){
                if (i == 0 || j == 0) {
                    dp[i][j] = 0;
                }
            }
        }
        for(int i=1; i < n+1; i++){
            for(int j=1; j<totRod+1; j++){
                if (length[i-1] <= j) {
                    dp[i][j] = Math.max(price[i-1] + dp[i][j-length[i-1]], dp[i-1][j]);
                } else {
                    dp[i][j] = dp[i-1][j];
                }
            }
        }
        return dp[n][totRod];
    }
    public static void main(String[] args) {
        int length[] = {1, 2, 3, 4, 5, 6, 7, 8};
        int price[] = {1, 5, 8, 9, 10, 17, 17, 20};
        int rodlength = 8;
        System.out.println("Maximum Profit by cutting rod: " + rodCutting(length, price,
rodlength));
    }
}
```

15. String Conversion

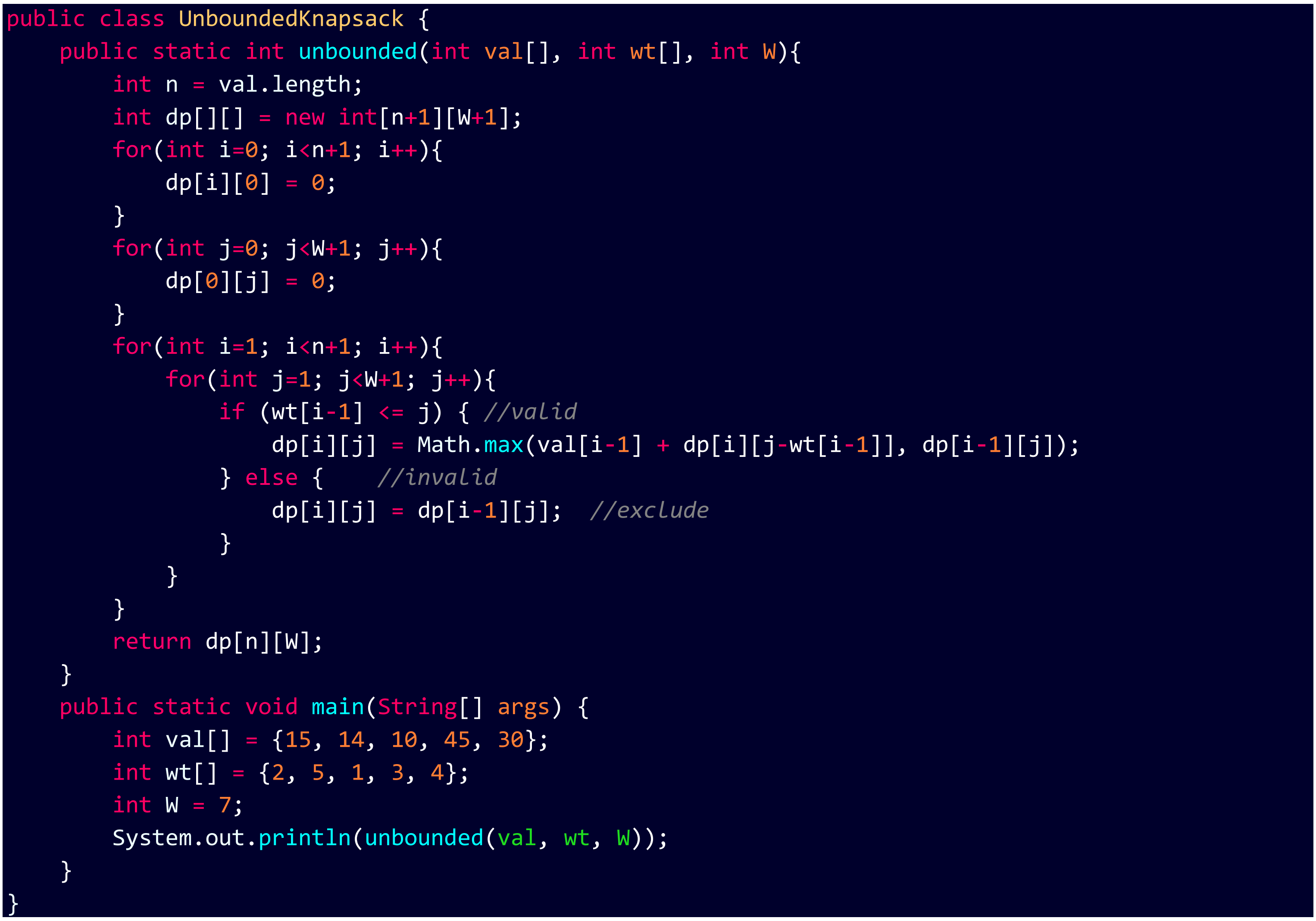
```
public class StringConversion {
    public static int[] convertString(String str1, String str2){
        int m = str1.length();
        int n = str2.length();
        //find the LCS length
        int lcsLen = lcs(str1, str2, m, n);
        //calculate deletions and insertions
        int delete = m - lcsLen;
        int insert = n - lcsLen;
        return new int[] {delete, insert};
    }
    public static int lcs(String str1, String str2, int m, int n){
        int dp[][] = new int[m+1][n+1];
        for(int i=1; i<=m; i++){
            for(int j=1; j<=n; j++){
                if (str1.charAt(i-1) == str2.charAt(j-1)) {
                    dp[i][j] = 1 + dp[i-1][j-1];
                } else {
                    dp[i][j] = Math.max(dp[i-1][j], dp[i][j-1]);
                }
            }
        }
```

```java
        return dp[m][n];
    }
    public static void main(String[] args) {
        String str1 = "heap";
        String str2 = "pea";
        int result[] = convertString(str1, str2);
        System.out.println("Deletions: " + result[0]);
        System.out.println("Insertions: " + result[1]);

    /*
     * Deletions will be of 2 characters 'h' and 'p'.
     * Insertions will be of 1 character i.e., 'p'.
     */
    }
}
```

16. Target Sum Subset

```java
public class TargetSumSubset {
    public static boolean target_sum_subset(int arr[], int sum){
        int n = arr.length;
        boolean dp[][] = new boolean[n+1][sum+1];
        //i = items & j = target sum
        for(int i=0; i<n+1; i++){
            dp[i][0] = true;
        }
        for(int i=1; i<n+1; i++){
            for(int j=1; j<sum+1; j++){
                int v = arr[i-1];
                if (v <= j && dp[i-1][j-v] == true) {   //include
                    dp[i][j] = true;
                } else if (dp[i-1][j] == true) {
                    dp[i][j] = true;    //exclude
                }
            }
        }
        print(dp);
        return dp[n][sum];
    }
    public static void print(boolean dp[][]){
        for(int i=0; i<dp.length; i++){
            for(int j=0; j<dp[0].length; j++){
                System.out.print(dp[i][j] + " ");
            }
            System.out.println();
        }
        System.out.println();
    }
    public static void main(String[] args) {
        int num[] = {4, 2, 7, 1, 3};
        int targetSum = 10;
        System.out.println("Final Answer: " + target_sum_subset(num, targetSum));
    }
}
```

17. Unbounded Knapsack



```
public class UnboundedKnapsack {
    public static int unbounded(int val[], int wt[], int W){
        int n = val.length;
        int dp[][] = new int[n+1][W+1];
        for(int i=0; i<n+1; i++){
            dp[i][0] = 0;
        }
        for(int j=0; j<W+1; j++){
            dp[0][j] = 0;
        }
        for(int i=1; i<n+1; i++){
            for(int j=1; j<W+1; j++){
                if (wt[i-1] <= j) { //valid
                    dp[i][j] = Math.max(val[i-1] + dp[i][j-wt[i-1]], dp[i-1][j]);
                } else {    //invalid
                    dp[i][j] = dp[i-1][j];  //exclude
                }
            }
        }
        return dp[n][W];
    }
    public static void main(String[] args) {
        int val[] = {15, 14, 10, 45, 30};
        int wt[] = {2, 5, 1, 3, 4};
        int W = 7;
        System.out.println(unbounded(val, wt, W));
    }
}
```