Subset sum possible

public class SubsetSumProblem {

    public static void main(String[] args) {

        int[] arr={2,3,7,8,10};

        int sum=16;

        boolean s= knap(arr,sum);

        System.out.println(s);

    }

    public static boolean knap(int[] arr,int sum){

        boolean[][] dp= new boolean[arr.length+1][sum+1];

        if(sum==0){

            return true;

        }

        if(arr.length==0){

            return false;

        }

        for(int i=0;i<arr.length+1;i++){

            for(int j=0;j<sum+1;j++){

                if(i==0 && j==0){

                    dp[i][j]=true;

                }

                if(j==0){

                    dp[i][j]=true;

                }

                if(i==0){

                    dp[i][j]=false;

                }

            }

        }

               for(int i=1;i<arr.length+1;i++){

            for(int j=1;j<sum+1;j++){

                if(arr[i-1]<=j){

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

            }else{

                dp[i][j]=dp[i-1][j];

            }

        }}

        return dp[arr.length][sum];

    }

}

Equal Sum Partition

S1-s2=0 ,s1+s2=sum => s1=sum/2

public class EqualSumPartition {

    public static void main(String[] args) {

        int[] arr={3,9,11,5};

        boolean s= fir(arr);

        System.out.println(s);

    }

    public static boolean fir(int[] arr){

        int sum=0;

        for(int i=0;i<arr.length;i++){

           sum +=arr[i];

        }

        if(sum==0){

            return true;

        }

        if(arr.length==0){

            return false;

        }

        if(sum%2==1){

            return false;

        }else{

            Boolean s= knap(arr,sum/2);

            return s;

        }

    }

    public static boolean knap(int[] arr,int sum){

        boolean[][] dp= new boolean[arr.length+1][sum+1];

        if(sum==0){

            return true;

        }

        if(arr.length==0){

            return false;

        }

        for(int i=0;i<arr.length+1;i++){

            for(int j=0;j<sum+1;j++){

                if(i==0 && j==0){

                    dp[i][j]=true;

                }

                if(j==0){

                    dp[i][j]=true;

                }

                if(i==0){

                    dp[i][j]=false;

                }

            }

        }

        for(int i=1;i<arr.length+1;i++){

            for(int j=1;j<sum+1;j++){

                if(arr[i-1]<=j){

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

            }else{

                dp[i][j]=dp[i-1][j];

            }

        }}

        return dp[arr.length][sum];

    }

}

Min subset sum diff

S1-s2=diff, s1+s2=sum => s1= sum+diiff/2

Count subset Sum

public class CountSubsetSum {

    public static void main(String[] args) {

        int[] arr={3,3,3,3};

        int sum=6;

        int s= knap(arr,sum);

        System.out.println(s);

    }

    public static int knap(int[] arr,int sum){

        int[][] dp= new int[arr.length+1][sum+1];

        dp[0][0] = 1;

        for(int i = 1; i <= sum; i++)

            dp[0][i] = 0;

        for(int i=1;i<arr.length+1;i++){

            for(int j=0;j<sum+1;j++){

                if(arr[i-1]<=j){

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

                }else{

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

            }

        }}

        return dp[arr.length][sum];

    }

}

Count subset sum diff

S1-s2=diff, s1+s2=sum => s1= sum+diiff/2

public class CountSubGivenDiff {

    public static void main(String[] args) {

        int[] arr={1,2,3,1,2};

        int diff=1;

        int s= fir(arr,diff);

        System.out.println(s);

    }

    public static int fir(int[] arr,int diff){

        int sum=0;

        for(int i=0;i<arr.length;i++){

           sum +=arr[i];

        }

        if(sum==0){

            return 0;

        }

        if(arr.length==0){

            return 0;

        }

        int s= knap(arr,(sum+diff)/2);

        return s;

    }

    public static int knap(int[] arr,int sum){

        int[][] dp= new int[arr.length+1][sum+1];

        dp[0][0] = 1;

        for(int i = 1; i <= sum; i++)

            dp[0][i] = 0;

        for(int i=1;i<arr.length+1;i++){

            for(int j=0;j<sum+1;j++){

                if(arr[i-1]<=j){

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

                }else{

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

            }

        }}

        return dp[arr.length][sum];

    }

}

Min sum diff of subset

S1-s2=min diff=> s1+s2=sum => min diff= sum-2\*j

Last row contain all possible

public class MinSumDiff {

    public static void main(String[] args) {

        int[] arr={1,6,5,11};

        int s= fir(arr);

        System.out.println(s);

    }

    public static int fir(int[] arr){

        int sum=0;

        for(int i=0;i<arr.length;i++){

           sum +=arr[i];

        }

        if(sum==0){

            return 0;

        }

        if(arr.length==0){

            return 0;

        }

        int s= knap(arr,sum);

        return s;

    }

    public static int knap(int[] arr,int sum){

        boolean[][] dp= new boolean[arr.length+1][sum+1];

        if(sum==0){

            return 0;

        }

        if(arr.length==0){

            return 0;

        }

        for(int i=0;i<arr.length+1;i++){

            for(int j=0;j<sum+1;j++){

                if(i==0 && j==0){

                    dp[i][j]=true;

                }

                if(j==0){

                    dp[i][j]=true;

                }

                if(i==0){

                    dp[i][j]=false;

                }

            }

        }

        for(int i=1;i<arr.length+1;i++){

            for(int j=1;j<sum+1;j++){

                if(arr[i-1]<=j){

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

            }else{

                dp[i][j]=dp[i-1][j];

            }

        }}

        int min=Integer.MAX\_VALUE;

        for(int j=1;j<(sum+1)/2;j++){

            if(dp[arr.length][j]==true){

                min=Math.min(min,sum-2\*j);

            }

        }

        return min;

    }

}

Rod cutting

Same as basic knapsack

public class RodCutting {

    public static void main(String[] args) {

        int[] length={1,2,3,4};

        int[] price={2,4,6,9};

        int n=4;

        int[][] dp=new int[n+1][length.length+1];

        int a=knap(length,price,n,dp);

        System.out.println(a);

    }

    static int knap(int[] length, int[] price,int n, int[][] dp){

        if(length.length==0){

            return 0;

        }

        for(int i=1;i<length.length+1;i++){

            for(int j=1;j<n+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][length.length];

    }

}

Max ways coins

Same as count subset sum

public class MaxWaysCoins {

    public static void main(String[] args) {    //coin change problen 1

        int[] arr={1,2,3};    //coins

        int sum=5;

        int s=knap(arr,sum);

        System.out.println(s);

    }

    public static int knap(int[] arr,int sum){

        int[][] dp= new int[arr.length+1][sum+1];

        dp[0][0] = 1;

        for(int i = 1; i <= sum; i++)

            dp[0][i] = 0;

        for(int i=1;i<arr.length+1;i++){

            for(int j=0;j<sum+1;j++){

                if(arr[i-1]<=j){

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

                }else{

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

            }

        }}

        return dp[arr.length][sum];

    }

}

Coin change problem

Easy h smjna h

public class CoisChangeProblem {

    public static void main(String[] args) {    //coin change problen 2  #min coins to make sum

        int[] arr={1,2,3,5};    //coins

        int sum=5;

        int s=knap(arr,sum);

        System.out.println(s);

    }

    public static int knap(int[] arr,int sum){

        int[][] dp= new int[arr.length+1][sum+1];

        dp[0][0] = Integer.MAX\_VALUE-1;

        for(int i = 1; i <= sum; i++)

            dp[0][i] = Integer.MAX\_VALUE-1;

        for(int i=1;i<=arr.length;i++)

           dp[i][0]=0;

        for(int j=1;j<=sum;j++){

            if(j% arr[0]==0){

                dp[1][j]=j/arr[0];

            }else{

                dp[1][j]=Integer.MAX\_VALUE-1;

            }

        }

        for(int i=2;i<arr.length+1;i++){

            for(int j=1;j<sum+1;j++){

                if(arr[i-1]<=j){

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

                }else{

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

            }

        }}

        return dp[arr.length][sum];

    }

}

Lcs

Longest common subsequence

public class LongestComSubsequence {

    public static void main(String[] args) {

        String x="abac";

        String y="cab";

        int m=x.length(),n=y.length();

        int s=lcs(x,y,m,n);

        System.out.println(s);

    }

    public static int lcs(String x,String y,int m,int n){

        int[][] dp= new int [m+1][n+1];

        if(m ==0 || n==0){

           return 0;

        }

        for(int i=0;i<m+1;i++){

           for(int j=0;j<n+1;j++){

               if(j==0 || i==0){

                   dp[i][j]=0;

               }

           }

       }

       for(int i=1;i<m+1;i++){

           for(int j=1;j<n+1;j++){

               if(x.charAt(i-1)==y.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]);

               }

           }

       }

       return dp[m][n];

    }

}

Longest Common substring

 class LongComSubstring {

    public static void main(String[] args) {

        String x="abcdefm";

        String y="acefm";

        int m=x.length(),n=y.length();

        int s=lcs(x,y,m,n);

        System.out.println(s);

    }

    public static int lcs(String x,String y,int m,int n){

        int[][] dp= new int [m+1][n+1];

        if(m ==0 || n==0){

           return 0;

        }

        for(int i=0;i<m+1;i++){

           for(int j=0;j<n+1;j++){

               if(j==0 || i==0){

                   dp[i][j]=0;

               }

           }

       }

       int s=0;

       for(int i=1;i<m+1;i++){

           for(int j=1;j<n+1;j++){

               if(x.charAt(i-1)==y.charAt(j-1)){

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

                 s=Math.max(s,dp[i][j]);

               }else{

                 dp[i][j]=0;

               }

           }

       }

       return s;

    }

}

Print longest subsequence

public class Printsubsequence {

    public static void main(String[] args) {

        String x="acbcf";

        String y="abcdaf";

        int m=x.length(),n=y.length();

        String s=lcs(x,y,m,n);

        System.out.println(s);

    }

    public static String lcs(String x,String y,int m,int n){

        int[][] dp= new int [m+1][n+1];

        if(m ==0 || n==0){

           return "";

        }

        for(int i=0;i<m+1;i++){

           for(int j=0;j<n+1;j++){

               if(j==0 || i==0){

                   dp[i][j]=0;

               }

           }

       }

       String sm="";

       for(int i=1;i<m+1;i++){

           for(int j=1;j<n+1;j++){

               if(x.charAt(i-1)==y.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]);

               }

           }

       }

       int i=m,j=n;

       while(i>0 && j>0){

        if(x.charAt(i-1)==y.charAt(j-1)){

            sm +=x.charAt(i-1);

            i--;

            j--;

        }else{

            if(dp[i-1][j]<dp[i][j-1]){

                j--;

            }else{

                i--;

            }

        }

       }

       return sm; /// reverse string found you can change it.

    }

}

Shortest Common Subsequence

public class ShortestSuperseuence {

    public static void main(String[] args) {

        String x="abac";

        String y="cab";

        int m=x.length(),n=y.length();

        int s=lcs(x,y,m,n);

        System.out.println(s);

    }

    public static int lcs(String x,String y,int m,int n){

        int[][] dp= new int [m+1][n+1];

        if(m ==0 || n==0){

           return 0;

        }

        for(int i=0;i<m+1;i++){

           for(int j=0;j<n+1;j++){

               if(j==0 || i==0){

                   dp[i][j]=0;

               }

           }

       }

       for(int i=1;i<m+1;i++){

           for(int j=1;j<n+1;j++){

               if(x.charAt(i-1)==y.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]);

               }

           }

       }

       return m+n-dp[m][n];

    }

}

Min insertion deletion

public class MinInsertDel {

    public static void main(String[] args) {

        String x="abac";

        String y="cab";

        int m=x.length(),n=y.length();

        int s=lcs(x,y,m,n);

        System.out.println(s);

    }

    public static int lcs(String x,String y,int m,int n){

        int[][] dp= new int [m+1][n+1];

        if(m ==0 || n==0){

           return 0;

        }

        for(int i=0;i<m+1;i++){

           for(int j=0;j<n+1;j++){

               if(j==0 || i==0){

                   dp[i][j]=0;

               }

           }

       }

       for(int i=1;i<m+1;i++){

           for(int j=1;j<n+1;j++){

               if(x.charAt(i-1)==y.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]);

               }

           }

       }

       return m+n-(2\*dp[m][n]);         // (1 insetion 2 deletion)   Deletion=> m-dp[m][n]  Insertion=>n=dp[m][n]

    }

}

Longest Palindromic Subsequence

public class PalindromicSubseq {

    public static void main(String[] args) {

        String x="abac";

        String y="";

        for(int i=x.length()-1;i>=0;i--){

            y+=x.charAt(i);

        }

        int m=x.length(),n=y.length();

        int s=lcs(x,y,m,n);

        System.out.println(s);

    }

    public static int lcs(String x,String y,int m,int n){

        int[][] dp= new int [m+1][n+1];

        if(m ==0 || n==0){

           return 0;

        }

        for(int i=0;i<m+1;i++){

           for(int j=0;j<n+1;j++){

               if(j==0 || i==0){

                   dp[i][j]=0;

               }

           }

       }

       for(int i=1;i<m+1;i++){

           for(int j=1;j<n+1;j++){

               if(x.charAt(i-1)==y.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]);

               }

           }

       }

       return dp[m][n];

    }

}

Minimum deletion Palindrome or insertion palindrome same h

public class MindelPalindromic {

    public static void main(String[] args) {

        String x="agbcba";

        String y="";

        for(int i=x.length()-1;i>=0;i--){

            y+=x.charAt(i);

        }

        int m=x.length(),n=y.length();

        int s=lcs(x,y,m,n);

        System.out.println(s);

    }

    public static int lcs(String x,String y,int m,int n){

        int[][] dp= new int [m+1][n+1];

        if(m ==0 || n==0){

           return 0;

        }

        for(int i=0;i<m+1;i++){

           for(int j=0;j<n+1;j++){

               if(j==0 || i==0){

                   dp[i][j]=0;

               }

           }

       }

       for(int i=1;i<m+1;i++){

           for(int j=1;j<n+1;j++){

               if(x.charAt(i-1)==y.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]);

               }

           }

       }

       return x.length()-dp[m][n];

    }

}

Print Shortest common supersequence

public class Printshortsubseq {

    public static void main(String[] args) {

        String x="aaaaaaaa";

        String y="aaaaaaaa";

        int m=x.length(),n=y.length();

        String s=lcs(x,y,m,n);

        System.out.println(s);

    }

    public static String lcs(String x,String y,int m,int n){

        int[][] dp= new int [m+1][n+1];

        if(m ==0 || n==0){

           return "";

        }

        for(int i=0;i<m+1;i++){

           for(int j=0;j<n+1;j++){

               if(j==0 || i==0){

                   dp[i][j]=0;

               }

           }

       }

       String sm="";

       for(int i=1;i<m+1;i++){

           for(int j=1;j<n+1;j++){

               if(x.charAt(i-1)==y.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]);

               }

           }

       }

       int i=m,j=n;

       while(i>0 && j>0){

        if(x.charAt(i-1)==y.charAt(j-1)){

            sm +=x.charAt(i-1);

            i--;

            j--;

        }else{

            if(dp[i-1][j]<dp[i][j-1]){

                 sm+=y.charAt(j-1);

                j--;

            }else{

                sm+= x.charAt(i-1);

                i--;

            }

        }

       }

       while(i>0){

        sm += x.charAt(i-1);

        i--;

       }

       while(j>0){

        sm+= y.charAt(j-1);

        j--;

       }

       String l="";

       for(int h=sm.length()-1;h>=0;h--){

           l+=x.charAt(h);

       }

       return l;

    }

}

Longest Repeating SUBSEquence

public class LongRepeatSub {

    public static void main(String[] args) {

        String x="aabebcdd";

        String y="aabebcdd";

        int m=x.length(),n=y.length();

        int s=lcs(x,y,m,n);

        System.out.println(s);

    }

    public static int lcs(String x,String y,int m,int n){

        int[][] dp= new int [m+1][n+1];

        if(m ==0 || n==0){

           return 0;

        }

        for(int i=0;i<m+1;i++){

           for(int j=0;j<n+1;j++){

               if(j==0 || i==0){

                   dp[i][j]=0;

               }

           }

       }

       for(int i=1;i<m+1;i++){

           for(int j=1;j<n+1;j++){

               if(x.charAt(i-1)==y.charAt(j-1) && i!=j){

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

               }else{

                 dp[i][j]=Math.max(dp[i-1][j],dp[i][j-1]);

               }

           }

       }

       return dp[m][n];

    }

}

Sequence Pattern Matching

public class SequncePattern {

    public static void main(String[] args) {   // is x is a subsequence of  x & y

        String x="aefm";

        String y="acefm";

        int m=x.length(),n=y.length();

        boolean s=lcs(x,y,m,n);

        System.out.println(s);

    }

    public static boolean lcs(String x,String y,int m,int n){

        int[][] dp= new int [m+1][n+1];

        if(m ==0 || n==0){

           return false;

        }

        for(int i=0;i<m+1;i++){

           for(int j=0;j<n+1;j++){

               if(j==0 || i==0){

                   dp[i][j]=0;

               }

           }

       }

       for(int i=1;i<m+1;i++){

           for(int j=1;j<n+1;j++){

               if(x.charAt(i-1)==y.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]);

               }

           }

       }

       if(dp[m][n]==x.length()){

        return true;

       }

       return false;

    }

}