```
public class Striver_DP{
//********************************PART II FINAL
CODES************************************************************** */
//*********************************DP-1 Introduction to Dynamic
Programming******************************************/
//*********************************DP-2 Climbing
Stairs*************************************************************/
class Solution{
    int main() {

        int n=3;
        vector<int> dp(n+1,-1);

        dp[0]=1;---------------------
        dp[1]=1;

        for(int i=2;i<=n;i++){
            dp[i]=dp[i-1]+dp[i-2];
        }
        cout<<dp[n];
        return 0;
      }
}
//*********************************DP-3 Frog
Jump********************************************************************/
class Solution{
    class TUF{
        static int solve(int ind,int[] height,int[] dp){
            if(ind==0) return 0;
            if(dp[ind]!=-1)return dp[ind];
            int jumpTwo=Integer.MAX_VALUE;
            int jumpOne=solve(ind-1, height,dp)+Math.abs(height[ind]-height[ind-1]);
            if(ind>1)
                jumpTwo=solve(ind-2, height,dp)+Math.abs(height[ind]-height[ind-2]);

            return dp[ind]=Math.min(jumpOne,jumpTwo);
        }
        class TUF{
            public static void main(String args[]) {

              int n=height.length;
              int dp[]=new int[n];
              Arrays.fill(dp,-1);
              dp[0]=0;
              for(int ind=1;ind<n;ind++){
                  int jumpTwo=Integer.MAX_VALUE;
                    int jumpOne=dp[ind-1]+Math.abs(height[ind]-height[ind-1]);
                    if(ind>1)
                        jumpTwo=dp[ind-2]+Math.abs(height[ind]-height[ind-2]);
                    dp[ind]=Math.min(jumpOne,jumpTwo);
              }
              System.out.println(dp[n-1]);
            }
}}}
//*********************************DP-4 Frog Jump with K
Distance**************************************************/
class Solution{
    class TUF{
        static int solveUtil(int ind,int[] height,int[] dp,int k){
            if(ind==0)return 0;
            if(dp[ind]!=-1)return dp[ind];
            int mmSteps=Integer.MAX_VALUE;
            for(int j=1;j<=k;j++){
                if(ind-j>=0){
               int jump=solveUtil(ind-j, height, dp, k)+ Math.abs(height[ind]-height[ind-j]);
                    mmSteps=Math.min(jump, mmSteps);
                }
            }
            return dp[ind]=mmSteps;
        }
        class TUF{
            static int solveUtil(int n,int[] height,int[] dp,int k){
                dp[0]=0;
                for(int i=1;i<n;i++){
                    int mmSteps=Integer.MAX_VALUE;
                    for(int j=1;j<=k;j++){
                        if(i-j>=0){
                            int jump=dp[i-j]+ Math.abs(height[i]-height[i-j]);
                            mmSteps= Math.min(jump, mmSteps);
                        }
                    }
                    dp[i]=mmSteps;
                }
                return dp[n-1];
                        }}          }
}
//*********************************DP-5 Maximum Sum of Non Adjacent
Elements*****************************************/
class Solution{
    public class Main{
        static int solveUtil(int ind,int[] arr,int[] dp){
            if(ind<0)return 0;
            if(ind==0)return arr[ind];
            if(dp[ind]!=-1)return dp[ind];
            int pick=arr[ind]+solveUtil(ind-2,arr,dp);
            int nonPick=0+solveUtil(ind-1,arr,dp);
            return dp[ind]=Math.max(pick, nonPick);
        }
        static int solveUtil(int n, int[] arr, int[] dp){
            dp[0]=arr[0];
            for(int i=1;i<n;i++){
                int pick=arr[i];
                if(i>1)
                    pick+=dp[i-2];
                int nonPick=0+dp[i-1];
                dp[i]= Math.max(pick, nonPick);
            }
            return dp[n-1];
        }
    }}
//********************************DP-6 House
Robber*****************************************************************/
class Solution{
    class TUF{
        static long solve(ArrayList<Integer> arr){
            int n=arr.size();
            long prev=arr.get(0);
            long prev2=0;
            for(int i=1;i<n;i++){
                long pick=arr.get(i);
                if(i>1)
                    pick+=prev2;
                long nonPick=0+prev;

                long cur_i=Math.max(pick,nonPick);
                prev2=prev;
                prev=cur_i;
            }
            return prev;
        }
        static long robStreet(int n,ArrayList<Integer> arr){
            ArrayList<Integer> arr1=new ArrayList<>();
            ArrayList<Integer> arr2=new ArrayList<>();
            if(n==1)
               return arr.get(0);
            for(int i=0;i<n;i++){

                if(i!=0) arr1.add(arr.get(i));
                if(i!=n-1) arr2.add(arr.get(i));
            }
            long ans1=solve(arr1);
            long ans2=solve(arr2);
            return Math.max(ans1,ans2);
        }
}}
//********************************DP-7 Ninja
```

```
Training***********************************************************/
class Solution{
   class TUF {
      static int f(int day,int last,int[][] points,int[][] dp){
         if (dp[day][last]!=-1)return dp[day][last];
         if (day==0){
            int maxi=0;
            for(int i=0;i<=2;i++) {
               if(i!=last)
                  maxi=Math.max(maxi,points[0][i]);
            }
            return dp[day][last]=maxi;
         }

         int maxi=0;
         for (int i=0;i<=2;i++) {
            if(i!=last){
               int activity=points[day][i]+f(day-1,i,points,dp);
               maxi=Math.max(maxi,activity);
            }
         }
         return dp[day][last]=maxi;
      }
   }
      class TUF {
         static int ninjaTraining(int n,int[][] points){
            int[][] dp=new int[n][4];
            dp[0][0]=Math.max(points[0][1],points[0][2]);
            dp[0][1]=Math.max(points[0][0],points[0][2]);
            dp[0][2]=Math.max(points[0][0],points[0][1]);
            dp[0][3]=Math.max(points[0][0],Math.max(points[0][1], points[0][2]));

            for(int day=1;day<n;day++){
               for(int last=0;last<4;last++){
                  dp[day][last]=0;
                  for(int task= 0;task<=2;task++){
                     if(task!=last){
                        int activity=points[day][task]+dp[day-1][task];
                        dp[day][last]=Math.max(dp[day][last],activity);
                     }
                  }
               }
            }
            return dp[n-1][3];
         }
      }
}
//*********************************DP-8 Grid Unique
Paths**********************************************************/
```

```
class Solution{
    class TUF{
        static int countWaysUtil(int i,int j,int[][] dp) {
          if(i==0&&j==0)return 1;
          if(i<0||j<0)return 0;
          if(dp[i][j]!=-1) return dp[i][j];
          int up=countWaysUtil(i-1,j,dp);
          int left=countWaysUtil(i,j-1,dp);
          return dp[i][j] = up+left;
        }
    }
    class TUF{
        static int countWaysUtil(int m, int n, int[][] dp) {
         for(int i=0;i<m;i++){
              for(int j=0;j<n;j++){
                  if(i==0&&j==0){
                      dp[i][j]=1;
                      continue;
                  }
                  int up=0;
                  int left = 0;
                  if(i>0)
                    up=dp[i-1][j];
                  if(j>0)
                    left=dp[i][j-1];
                  dp[i][j] = up+left;
              }
          }
          return dp[m-1][n-1];
        }
}}
//*********************************DP-9 Grid Unique Paths with
Obstacles********************************************/
class Solution{
    static int mazeObstaclesUtil(int i,int j,int[][] maze,int[][] dp){
        if(i>0&&j>0&&maze[i][j]==-1)return 0;
        if(i==0&&j==0)return 1;
        if(i<0||j<0)return 0;
        if(dp[i][j]!=-1)return dp[i][j];
        int up=mazeObstaclesUtil(i-1,j,maze,dp);
        int left=mazeObstaclesUtil(i,j-1,maze,dp);
        return dp[i][j]=up+left;
      }
      static int mazeObstaclesUtil(int n,int m,int[][] maze,int[][] dp){
        for(int i=0;i<n;i++){
             for(int j=0;j<m;j++){
                 if(i>0&&j>0&&maze[i][j]==-1){
                   dp[i][j]=0;
                   continue;
```

```
                }
                if(i==0&&j==0){
                    dp[i][j]=1;
                    continue;
                }
                int up=0;
                int left=0;
                if(i>0)up=dp[i-1][j];
                if(j>0)left=dp[i][j-1];
                dp[i][j]=up+left;
            }
        }
        return dp[n-1][m-1];
      }
}
//*********************************DP-10 Minimum Path Sum in
Grid***************************************************/
class Solution{
    static int minSumPathUtil(int i,int j,int[][] matrix,int[][] dp) {
        if(i==0&&j==0)
          return matrix[0][0];
        if(i<0||j<0)
          return (int)Math.pow(10,9);
        if(dp[i][j]!=-1) return dp[i][j];
        int up=matrix[i][j]+minSumPathUtil(i-1,j,matrix,dp);
        int left=matrix[i][j]+minSumPathUtil(i,j-1,matrix,dp);
        return dp[i][j]=Math.min(up,left);
      }
      static int minSumPath(int n, int m, int[][] matrix){
        int dp[][]=new int[n][m];
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(i==0&&j==0) dp[i][j]=matrix[i][j];
                else{
                    int up=matrix[i][j];
                    if(i>0) up+=dp[i-1][j];
                    else up+=(int)Math.pow(10,9);
                    int left=matrix[i][j];
                    if(j>0)left+=dp[i][j-1];
                    else left+=(int)Math.pow(10,9);
                    dp[i][j] = Math.min(up,left);
                }
            }
        }
        return dp[n-1][m-1];
    }
}
//*********************************DP-11 Triangle****************************************** */
class Solution{
    class TUF{
        static int minimumPathSumUtil(int i,int j,int[][] triangle,int n,int[][] dp) {
          if(dp[i][j]!=-1)return dp[i][j];
          if(i==n-1)return triangle[i][j];
          int down=triangle[i][j]+minimumPathSumUtil(i+1,j,triangle,n,dp);
          int diagonal=triangle[i][j]+minimumPathSumUtil(i+1,j+1,triangle,n,dp);
          return dp[i][j]=Math.min(down, diagonal);
        }
        static int minimumPathSum(int[][] triangle, int n){
            int dp[][]=new int[n][n];
            for(int j=0;j<n;j++){
                dp[n-1][j]=triangle[n-1][j];
            }
            for(int i=n-2;i>=0;i--){
                for(int j=i;j>=0;j--){
                    int down=triangle[i][j]+dp[i+1][j];
                    int diagonal=triangle[i][j]+dp[i+1][j+1];
                    dp[i][j]=Math.min(down,diagonal);
                }
            }
            return dp[0][0];
                }
            }
}
//**********************************DP-12 Minimum/Maximum Falling Path
Sum******************************************* */
class Solution{
    static int getMaxUtil(int i,int j,int m,int[][] matrix,int[][] dp){
        if(j<0||j>=m)return (int)Math.pow(-10,9);
        if(i==0)return matrix[0][j];
        if(dp[i][j]!=-1) return dp[i][j];
        int up = matrix[i][j]+getMaxUtil(i-1,j,m,matrix,dp);
        int leftDiagonal=matrix[i][j]+getMaxUtil(i-1,j-1,m,matrix,dp);
        int rightDiagonal=matrix[i][j]+getMaxUtil(i-1,j+1,m,matrix,dp);
        return dp[i][j]= Math.max(up,Math.max(leftDiagonal,rightDiagonal));
    }
    static int getMaxPathSum(int[][] matrix){
        int n=matrix.length;
        int m=matrix[0].length;
        int dp[][]= new int[n][m];
        for(int row[]: dp)
        Arrays.fill(row,-1);
        int maxi=Integer.MIN_VALUE;
        for(int j=0;j<m;j++){
            int ans=getMaxUtil(n-1,j,m,matrix,dp);
            maxi=Math.max(maxi,ans);
        }
        return maxi;
    }
    static int getMaxPathSum(int[][] matrix){
        int n=matrix.length;
        int m=matrix[0].length;
        int dp[][] = new int[n][m];
        for(int j=0;j<m;j++){
            dp[0][j]=matrix[0][j];
        }
        for(int i=1;i<n;i++){
            for(int j=0;j<m;j++){
                int up=matrix[i][j]+dp[i-1][j];
                int leftDiagonal=matrix[i][j];
                if(j-1>=0) leftDiagonal+=dp[i-1][j-1];
                else leftDiagonal+=(int)Math.pow(-10,9);
                int rightDiagonal=matrix[i][j];
                if(j+1<m) rightDiagonal+=dp[i-1][j+1];
                else rightDiagonal+=(int)Math.pow(-10,9);
                dp[i][j] = Math.max(up, Math.max(leftDiagonal,rightDiagonal));
            }
        }
        int maxi=Integer.MIN_VALUE;
        for(int j=0; j<m;j++){
            maxi=Math.max(maxi,dp[n-1][j]);
        }
        return maxi;
    }
}
//**********************************DP-13 Cherry PickUp -
II******************************************** */
class Solution{
    class TUF {
        static int maxChocoUtil(int i,int j1,int j2,int n,int m,int[][] grid,int[][][] dp){
          if(j1<0||j1>=m||j2<0||j2>=m)return (int)(Math.pow(-10, 9));

          if(i==n-1){
            if(j1==j2)return grid[i][j1];
            else return grid[i][j1]+grid[i][j2];
          }

          if(dp[i][j1][j2]!=-1)return dp[i][j1][j2];

          int maxi=Integer.MIN_VALUE;
          for (int di=-1;di<=1;di++){
            for (int dj=-1;dj<=1;dj++){
              int ans;
              if(j1==j2)
                ans=grid[i][j1]+maxChocoUtil(i+1,j1+di,j2+dj,n,m,grid,dp);
              else
                ans=grid[i][j1]+grid[i][j2]+maxChocoUtil(i+1,j1+di,j2+dj,n,m,grid,dp);
              maxi=Math.max(maxi, ans);
            }
          }
          return dp[i][j1][j2] = maxi;
        }
        class TUF {
            static int maximumChocolates(int n, int m, int[][] grid) {

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

              for (int j1=0;j1<m;j1++){
                for (int j2=0;j2<m;j2++){
                  if (j1 == j2)
                    dp[n-1][j1][j2]=grid[n-1][j1];
                  else
                    dp[n-1][j1][j2]=grid[n-1][j1]+grid[n-1][j2];
                }
              }

              for (int i=n-2;i>=0;i--){
                for (int j1=0;j1<m;j1++){
                  for (int j2=0;j2<m;j2++){
                    int maxi=Integer.MIN_VALUE;
                    //Inner nested loops to try out 9 options
                    for(int di=-1;di<=1;di++){
                      for(int dj=-1; dj<=1;dj++){
                        int ans;
                        if (j1==j2)
                          ans=grid[i][j1];
                        else
                          ans=grid[i][j1]+grid[i][j2];
                        if((j1+di<0||j1+di>= m)||(j2+dj<0||j2+dj>= m))
                          ans+=(int) Math.pow(-10,9);
                        else
                          ans+ dp[i+1][j1+di][j2+dj];

                        maxi=Math.max(ans,maxi);
                      }
                    }
                    dp[i][j1][j2] = maxi;
                  }
                }
              }

              return dp[0][0][m - 1];
            }
}}}
//*********************************DP-14 Subset Sum Equals
Target****************************************** */
class Solution{
    class TUF{
        static boolean subsetSumUtil(int ind,int target,int[] arr,int[][] dp){
            if(target==0)return true;
            if(ind==0)return arr[0]==target;
            if(dp[ind][target]!=-1)return dp[ind][target]==0?false:true;
            boolean notTaken = subsetSumUtil(ind-1,target,arr,dp);
            boolean taken = false;
            if(arr[ind]<=target)
                taken = subsetSumUtil(ind-1,target-arr[ind],arr,dp);
                dp[ind][target]=notTaken||taken?1:0;
            return notTaken||taken;
        }
    }
    class TUF{
        static boolean subsetSumToK(int n,int k,int[] arr){
            boolean dp[][]= new boolean[n][k+1];
            for(int i=0;i<n;i++){
                dp[i][0]=true;
            }
            if(arr[0]<=k)
                dp[0][arr[0]]=true;
            for(int ind=1;ind<n;ind++){
                for(int target=1;target<=k;target++){
                    boolean notTaken=dp[ind-1][target];
                    boolean taken = false;
                        if(arr[ind]<=target)
                            taken=dp[ind-1][target-arr[ind]];
                    dp[ind][target]= notTaken||taken;
                }
            }
            return dp[n-1][k];
        }
    }
}
//********************************DP-15 Partition Equal Subset
Sum****************************************** */
class Solution{
    class TUF{
        static boolean subsetSumUtil(int ind,int target,int arr[],int[][] dp){
            if(target==0)return true;
            if(ind==0)return arr[0]==target;
            if(dp[ind][target]!=-1)return dp[ind][target]==0?false:true;
            boolean notTaken=subsetSumUtil(ind-1,target,arr,dp);
            boolean taken=false;
            if(arr[ind]<=target)
                taken=subsetSumUtil(ind-1,target-arr[ind],arr,dp);
                dp[ind][target]=notTaken||taken?1:0;
            return notTaken||taken;
        }
        static boolean canPartition(int n,int[] arr){
            int totSum=0;
            for(int i=0; i<n;i++){
                totSum+= arr[i];
            }
            if (totSum%2==1) return false;
            else{
                int k = totSum/2;
                int dp[][]=new int[n][k+1];
                for(int row[]: dp)
                Arrays.fill(row,-1);
                return subsetSumUtil(n-1,k,arr,dp);
            }
        }
        }
        class TUF{
            static boolean canPartition(int n,int[] arr){
                int totSum=0;
                for(int i=0; i<n;i++){
                    totSum+= arr[i];
                }
                if (totSum%2==1) return false;
                else{
                    int k = totSum/2;
                    boolean dp[][]=new boolean[n][k+1];
                    for(int i=0; i<n; i++){
                        dp[i][0] = true;
                    }
                    if(arr[0]<=k)
                        dp[0][arr[0]] = true;
                    for(int ind=1;ind<n;ind++){
                        for(int target=1;target<=k;target++){
                            boolean notTaken=dp[ind-1][target];
                            boolean taken=false;
                                if(arr[ind]<=target)
                                    taken=dp[ind-1][target-arr[ind]];
                            dp[ind][target]= notTaken||taken;
                        }
                    }
                    return dp[n-1][k];
                }
            }
    }
}
//**********************************DP-16 Partition A Set into Two Subsets with minimum
Difference********************************************** */
class Solution{
    bool subsetSumUtil(int ind,int target,vector<int>&arr,vector<vector<int>>&dp){
        if(target==0)return dp[ind][target]=true;
        if(ind==0)return dp[ind][target]=arr[0]==target;
        if(dp[ind][target]!=-1)return dp[ind][target];
        bool notTaken=subsetSumUtil(ind-1,target,arr,dp);
        bool taken=false;
        if(arr[ind]<=target)
        taken=subsetSumUtil(ind-1,target-arr[ind],arr,dp);
        return dp[ind][target]=notTaken||taken;
    }
    int minSubsetSumDifference(vector<int>&arr,int n){
        int totSum=0;
        for(int i=0;i<n;i++){
            totSum+=arr[i];
        }
        vector<vector<int>> dp(n,vector<int>(totSum+1,-1));
        for(int i=0;i<=totSum;i++){
            bool dummy=subsetSumUtil(n-1,i,arr,dp);
        }
        int mini=1e9;
        for(int i=0;i<=totSum;i++){
            if(dp[n-1][i]==true){
            int diff=abs(i-(totSum-i));
            mini=min(mini, diff);
            }
        }
        return mini;
    }
    int minSubsetSumDifference(vector<int>&arr,int n){
        int totSum=0;
        for(int i=0;i<n;i++){
          totSum+=arr[i];
        }
        vector<vector<bool>>dp(n,vector<bool>(totSum+1,false));
        for(int i=0;i<n;i++){
          dp[i][0]=true;
        }
        if(arr[0]<=totSum)
          dp[0][totSum]=true;
        for(int ind=1;ind<n;ind++){
          for(int target=1;target<=totSum;target++){
            bool notTaken=dp[ind-1][target];
            bool taken=false;
            if(arr[ind]<=target)
              taken=dp[ind-1][target-arr[ind]];
            dp[ind][target]=notTaken||taken;
          }
        }
        int mini=1e9;
        for(int i=0;i<=totSum;i++){
          if(dp[n-1][i]==true){
            int diff=abs(i -(totSum-i));
            mini=min(mini, diff);
          }
        }
        return mini;
      }

}
//**********************************DP-17 Count Subsets with Sum equals
K******************************************** */
class Solution{
    class TUF{
        static int findWaysUtil(int ind,int target,int[] arr,int[][] dp){
            if(target==0)return 1;
            if(ind==0)return arr[0]==target?1:0;
            if(dp[ind][target]!=-1)return dp[ind][target];
            int notTaken=findWaysUtil(ind-1,target,arr,dp);
            int taken = 0;
            if(arr[ind]<=target)
                taken=findWaysUtil(ind-1,target-arr[ind],arr,dp);
            return dp[ind][target]= notTaken + taken;
        }
        static int findWays(int[] num, int k){
            int n=num.length;
            int dp[][]=new int[n][k+1];
            for(int row[]: dp)
            Arrays.fill(row,-1);
            return findWaysUtil(n-1,k,num,dp);
        }}
    class TUF{
            static int findWays(int[] num,int k){
                int n=num.length;
                int[][] dp=new int[n][k+1];
                for(int i=0;i<n;i++){
                    dp[i][0]=1;
                }
                if(num[0]<=k)
                    dp[0][num[0]]=1;
                for(int ind=1;ind<n;ind++){
                    for(int target=1;target<=k;target++){
                        int notTaken=dp[ind-1][target];
                        int taken=0;
                            if(num[ind]<=target)
                                taken=dp[ind-1][target-num[ind]];
                        dp[ind][target]= notTaken+taken;
                    }
                }
                return dp[n-1][k];
                }
        }
}
//*********************************DP-18 Count Partitions with given
difference****************************************** */
class Solution{
    class TUF{
        static int mod =(int)(Math.pow(10,9)+7);
        static int countPartitionsUtil(int ind,int target,int[] arr, int[][] dp){
             if(ind==0){
                if(target==0&&arr[0]==0)
                    return 2;
                if(target==0||target==arr[0])
                    return 1;
                return 0;
            }
            if(dp[ind][target]!=-1)return dp[ind][target];
            int notTaken=countPartitionsUtil(ind-1,target,arr,dp);
            int taken=0;
            if(arr[ind]<=target)
                taken=countPartitionsUtil(ind-1,target-arr[ind],arr,dp);
            return dp[ind][target]=(notTaken+taken)%mod;
        }

        static int countPartitions(int d,int[] arr){
            int n = arr.length;
            int totSum = 0;
            for(int i=0; i<arr.length;i++){
                totSum += arr[i];
            }
            if(totSum-d<0) return 0;
            if((totSum-d)%2==1) return 0;
            int s2 = (totSum-d)/2;
            int dp[][] = new int[n][s2+1];
            for(int row[]: dp)
            Arrays.fill(row,-1);
            return countPartitionsUtil(n-1,s2,arr,dp);
        }}
        class TUF{
            static int mod =(int)(Math.pow(10,9)+7);
            static int findWays(int[] num,int tar){
                 int n=num.length;
                int dp[][]=new int[n][tar+1];
                if(num[0]==0)dp[0][0] =2;  // 2 cases -pick and not pick
                else dp[0][0]=1;  // 1 case - not pick
                if(num[0]!=0&&num[0]<=tar) dp[0][num[0]]=1;  // 1 case -pick
                for(int ind=1;ind<n;ind++){
                    for(int target= 0;target<=tar;target++){
                        int notTaken=dp[ind-1][target];
                        int taken=0;
                    if(num[ind]<=target)
                        taken=dp[ind-1][target-num[ind]];
                dp[ind][target]=(notTaken + taken)%mod;
            }
        }
        return dp[n-1][tar];
    }

    static int countPartitions(int n,int d,int[] arr){
        int totSum=0;
        for(int i=0;i<n;i++){
            totSum+=arr[i];
        }
        if(totSum-d<0||(totSum-d)%2==1) return 0;
        return findWays(arr,(totSum-d)/2);
    }
}}
//*********************************DP-19 0/1
KnapSack****************************************** */
class Solution{
    class TUF{
        static int knapsackUtil(int[] wt,int[] val, int ind, int W,int[][] dp){
            if(ind == 0){
                if(wt[0] <=W) return val[0];
                else return 0;
            }
            if(dp[ind][W]!=-1)return dp[ind][W];
            int notTaken=0+knapsackUtil(wt,val,ind-1,W,dp);
            int taken=Integer.MIN_VALUE;
            if(wt[ind]<=W)
                taken=val[ind]+knapsackUtil(wt,val,ind-1,W-wt[ind],dp);

            return dp[ind][W]=Math.max(notTaken,taken);
        }
    }
        class TUF{
            static int knapsack(int[] wt,int[] val, int n, int W){
                int dp[][]=new int[n][W+1];
                for(int i=wt[0];i<=W;i++){
                    dp[0][i]=val[0];
                }
                for(int ind=1;ind<n;ind++){
                    for(int cap=0;cap<=W;cap++){
                        int notTaken=0+dp[ind-1][cap];
                        int taken=Integer.MIN_VALUE;
                        if(wt[ind]<=cap)
                            taken=val[ind]+dp[ind-1][cap-wt[ind]];
                        dp[ind][cap]=Math.max(notTaken,taken);
                    }
            }
            return dp[n-1][W];
        }
    }
}
//********************************DP-20 Minimum
Coins****************************************** */
class Solution{
    class TUF{
        static int minimumElementsUtil(int[] arr,int ind,int T,int[][] dp){
            if(ind==0){
                if(T%arr[0]==0) return T/arr[0];
                else return (int)Math.pow(10,9);
            }
            if(dp[ind][T]!=-1)
                return dp[ind][T];
            int notTaken=0+minimumElementsUtil(arr,ind-1,T,dp);
            int taken=(int)Math.pow(10,9);
            if(arr[ind]<=T)
                taken=1+minimumElementsUtil(arr,ind,T-arr[ind],dp);
            return dp[ind][T]=Math.min(notTaken,taken);
        }
    }
        class TUF{
            static int minimumElements(int[] arr,int T){
                int n=arr.length;
                int dp[][]=new int[n][T+1];
                for(int i=0;i<=T;i++){
                    if(i%arr[0]==0)
                        dp[0][i]=i/arr[0];
                    else dp[0][i]=(int)Math.pow(10,9);
                }
                for(int ind=1;ind<n;ind++){
                    for(int target=0;target<=T;target++){

                        int notTake=0+dp[ind-1][target];
                        int take=(int)Math.pow(10,9);
                        if(arr[ind]<=target)
                            take=1+ dp[ind][target-arr[ind]];
                         dp[ind][target]=Math.min(notTake, take);
                    }
                }
                int ans=dp[n-1][T];
                if(ans>=(int)Math.pow(10,9)) return -1;
                return ans;
            }
}
}
//********************************DP-21 Target
```

```
Sum****************************************** */
class Solution{
    class TUF{
        static int countPartitionsUtil(int ind, int target, int[] arr,int[][] dp){
             if(ind==0){
                if(target==0&&arr[0]==0)
                    return 2;
                if(target==0||target==arr[0])
                    return 1;
                return 0;
            }

            if(dp[ind][target]!=-1)
                return dp[ind][target];

            int notTaken=countPartitionsUtil(ind-1,target,arr,dp);

            int taken=0;
            if(arr[ind]<=target)
                taken=countPartitionsUtil(ind-1,target-arr[ind],arr,dp);

            return dp[ind][target]=(notTaken+taken);
        }

        static int targetSum(int n,int target,int[] arr){
            int totSum=0;
            for(int i=0;i<arr.length;i++){
                totSum+=arr[i];
            }
            if(totSum-target<0) return 0;
            if((totSum-target)%2==1) return 0;
            int s2=(totSum-target)/2;
            int dp[][]=new int[n][s2+1];
            for(int row[]: dp)
            Arrays.fill(row,-1);
            return countPartitionsUtil(n-1,s2,arr,dp);
        }}
    static int findWays(int []num, int tar){
       int n=num.length;
       int[][] dp=new int[n][tar+1];
       if(num[0]==0) dp[0][0] =2;  // 2 cases -pick and not pick
       else dp[0][0]=1;  // 1 case - not pick
       if(num[0]!=0&&num[0]<=tar) dp[0][num[0]] = 1;  // 1 case -pick
       for(int ind=1;ind<n;ind++){
           for(int target=0;target<=tar;target++){
               int notTaken=dp[ind-1][target];
               int taken=0;
                   if(num[ind]<=target)
                       taken=dp[ind-1][target-num[ind]];
               dp[ind][target]=(notTaken+taken)%mod;
           }
       }
       return dp[n-1][tar];
    }

    static int targetSum(int n,int target,int[] arr){
       int totSum=0;
       for(int i=0;i<n;i++){
           totSum+=arr[i];
       }
       if(totSum-target <0||(totSum-target)%2==1) return 0;
       return findWays(arr,(totSum-target)/2);
    }
 }
 //*********************************DP-22 Coin Change
 ||***************************************** */
 class Solution{
    class TUF{
        static long countWaysToMakeChangeUtil(int[] arr,int ind, int T,long[][] dp){
            if(ind == 0){
                if(T%arr[0]==0)
                return 1;
                else
                return 0;
            }
            if(dp[ind][T]!=-1)return dp[ind][T];
            long notTaken=countWaysToMakeChangeUtil(arr,ind-1,T,dp);
            long taken=0;
            if(arr[ind]<=T)
                taken=countWaysToMakeChangeUtil(arr,ind,T-arr[ind],dp);

            return dp[ind][T]=notTaken+taken;
        }
    }
        class TUF{
            static long countWaysToMakeChange(int[] arr,int n,int T){
                long dp[][]=new long[n][T+1];
                for(int i=0;i<=T;i++){
                    if(i%arr[0]==0)
                       dp[0][i]=1;
                }
                for(int ind=1;ind<n;ind++){
                    for(int target=0;target<=T;target++){
                       long notTaken=dp[ind-1][target];
                       long taken=0;
                       if(arr[ind]<=target)
                          taken=dp[ind][target-arr[ind]];
                       dp[ind][target] = notTaken + taken;
```

```
                }
            }
            return dp[n-1][T];
        }
    }
}
//*********************************DP-23 Unbounded
KnapSack****************************************** */
class Solution{
    class TUF{
        static int knapsackUtil(int[] wt,int[] val, int ind, int W,int[][] dp){
            if(ind == 0){return ((int)(W/wt[0]))*val[0];}
            if(dp[ind][W]!=-1)return dp[ind][W];
            int notTaken=0+knapsackUtil(wt,val,ind-1,W,dp);
            int taken=Integer.MIN_VALUE;
            if(wt[ind]<=W)
                taken=val[ind]+knapsackUtil(wt,val,ind,W-wt[ind],dp);
            return dp[ind][W]=Math.max(notTaken,taken);
        }
    }
        class TUF{
            static int unboundedKnapsack(int n,int W, int[] val,int[] wt){
                int[][] dp=new int[n][W+1];
                for(int i=wt[0];i<=W;i++){
                    dp[0][i]=((int)i/wt[0])*val[0];
                }
                for(int ind=1;ind<n;ind++){
                    for(int cap=0;cap<=W;cap++){
                        int notTaken=0+dp[ind-1][cap];
                        int taken=Integer.MIN_VALUE;
                        if(wt[ind]<=cap)
                            taken=val[ind]+dp[ind][cap-wt[ind]];
                        dp[ind][cap]=Math.max(notTaken,taken);
                    }
                }
                return dp[n-1][W];
            }}
}
//*********************************DP-24 Rod Cutting
Problem****************************************** */
class Solution{
    class TUF{
        static int cutRodUtil(int[] price,int ind,int N,int[][] dp){
            if(ind==0){return N*price[0];}
            if(dp[ind][N]!=-1)return dp[ind][N];
            int notTaken=0+cutRodUtil(price,ind-1,N,dp);
            int taken=Integer.MIN_VALUE;
            int rodLength=ind+1;
            if(rodLength<=N)
                taken=price[ind]+cutRodUtil(price,ind,N-rodLength,dp);
            return dp[ind][N] = Math.max(notTaken,taken);
        }
    }
        class TUF{
            static int cutRod(int[] price,int N) {
                int dp[][]=new int[N][N+1];
                for(int row[]:dp)
                Arrays.fill(row,-1);
                for(int i=0; i<=N; i++){
                    dp[0][i] = i*price[0];
                }
                for(int ind=1;ind<N;ind++){
                    for(int length=0;length<=N;length++){
                        int notTaken=0+dp[ind-1][length];
                        int taken=Integer.MIN_VALUE;
                        int rodLength=ind+1;
                        if(rodLength<=length)
                            taken=price[ind]+dp[ind][length-rodLength];
                        dp[ind][length]=Math.max(notTaken,taken);
                    }
                }
                return dp[N-1][N];
            }
        }
    }
    //*********************************DP-25 Longest Common
Subsequence******************************************* */
class Solution{
        //****************************Memoization**************************** */
        class TUF{
        static int lcsUtil(String s1,String s2,int ind1,int ind2,int[][] dp){
            if(ind1<0||ind2<0)return 0;
            if(dp[ind1][ind2]!=-1)return dp[ind1][ind2];
            if(s1.charAt(ind1)==s2.charAt(ind2))
                return dp[ind1][ind2]=1+lcsUtil(s1,s2,ind1-1,ind2-1,dp);
            else
                return dp[ind1][ind2]=0+Math.max(lcsUtil(s1,s2,ind1,ind2-1,dp),lcsUtil(s1,s2,
ind1-1,ind2,dp));
        }
        static int lcs(String s1,String s2) {
            int n=s1.length();
            int m=s2.length();
            int dp[][]=new int[n][m];
            for(int rows[]: dp)
            Arrays.fill(rows,-1);
            return lcsUtil(s1,s2,n-1,m-1,dp);
        }
        }
    //***************************Tabulation*********************************** */
    class TUF{
        static int lcs(String s1,String s2) {
            int n=s1.length();
            int m=s2.length();
            int dp[][]=new int[n+1][m+1];
            for(int rows[]: dp)
            Arrays.fill(rows,-1);
            for(int i=0;i<=n;i++){
                dp[i][0] = 0;
            }
            for(int i=0;i<=m;i++){
                dp[0][i] = 0;
            }
            for(int ind1=1;ind1<=n;ind1++){
                for(int ind2=1;ind2<=m;ind2++){
                    if(s1.charAt(ind1-1)==s2.charAt(ind2-1))
                        dp[ind1][ind2]=1+dp[ind1-1][ind2-1];
                    else
                        dp[ind1][ind2]=0+Math.max(dp[ind1-1][ind2],dp[ind1][ind2-1]);
                }
            }
            return dp[n][m];
        }
}
}
//*********************************DP-26 Printing Longest Common
Subsequence****************************************** */
class Solution{
    class TUF{
        static void lcs(String s1,String s2) {
            int n=s1.length();
            int m=s2.length();
            int dp[][]=new int[n+1][m+1];
            for(int i=0;i<=n;i++){
                dp[i][0] = 0;
            }
            for(int i=0;i<=m;i++){
                dp[0][i] = 0;
            }
            for(int ind1=1;ind1<=n;ind1++){
                for(int ind2=1;ind2<=m;ind2++){
                    if(s1.charAt(ind1-1)==s2.charAt(ind2-1))
                        dp[ind1][ind2]=1+dp[ind1-1][ind2-1];
                    else
                        dp[ind1][ind2]=0+Math.max(dp[ind1-1][ind2],dp[ind1][ind2-1]);
                }
            }
            int len=dp[n][m];
            int i=n;
            int j=m;
            int index = len-1;
            String str="";
            for(int k=1; k<=len;k++){
                str +="$"; // dummy string
            }
            StringBuilder ss= new StringBuilder(s1);
            StringBuilder str2=new StringBuilder(str);
            while(i>0&&j>0){
                if(ss.charAt(i-1)==s2.charAt(j-1)){
                    str2.setCharAt(index,ss.charAt(i-1) );
                    index--;
                    i--;
                    j--;
                }
                else if(ss.charAt(i-1)>s2.charAt(j-1)){
                    i--;
                }
                else j--;
            }
            System.out.println(str2);
        }
    }
}
//*********************************DP-27 Longest Common
Substring******************************************* */
class Solution{
    class TUF{
        static int lcs(String s1,String s2){
            int n=s1.length();
            int m=s2.length();
            int[][] dp=new int[n+1][m+1];
            int ans = 0;
            for(int i=1;i<=n;i++){
                for(int j=1;j<=m;j++){
                    if(s1.charAt(i-1)==s2.charAt(j-1)){
                        int val=1+dp[i-1][j-1];
                        dp[i][j]=val;
                        ans=Math.max(ans,val);
                    }
                    else
                        dp[i][j] = 0;
                }
            }
            return ans;
        }
    }
}
```

```
//*********************************DP-28 Longest Palindromic
Subsequence******************************************** */
class Solution{
    class TUF{
        static int lcs(String s1,String s2) {
            int n=s1.length();
            int m=s2.length();
            int dp[][]=new int[n+1][m+1];
            for(int rows[]:dp)
            Arrays.fill(rows,-1);
            for(int i=0;i<=n;i++){
                dp[i][0] = 0;
            }
            for(int i=0;i<=m;i++){
                dp[0][i] = 0;
            }
            for(int ind1=1;ind1<=n;ind1++){
                for(int ind2=1;ind2<=m;ind2++){
                    if(s1.charAt(ind1-1)==s2.charAt(ind2-1))
                        dp[ind1][ind2]=1+dp[ind1-1][ind2-1];
                    else
                        dp[ind1][ind2]=0+Math.max(dp[ind1-1][ind2],dp[ind1][ind2-1]);
                }
            }
            return dp[n][m];
        }

        static int longestPalindromeSubsequence(String s){
            String t = s;
            String ss=new StringBuilder(s).reverse().toString();
            return lcs(ss,t);
        }

        }
    }
//*********************************DP-29 Minimum Insertions to make string
palindrome******************************************** */
class Solution{
    class TUF{
        static int lcs(String s1,String s2){
            int n=s1.length();
            int m=s2.length();
            int dp[][]=new int[n+1][m+1];
            for(int rows[]:dp)
            Arrays.fill(rows,-1);
            for(int i=0;i<=n;i++){
                dp[i][0] = 0;
            }
            for(int i=0;i<=m;i++){
                dp[0][i] = 0;
            }
            for(int ind1=1;ind1<=n;ind1++){
                for(int ind2=1;ind2<=m;ind2++){
                    if(s1.charAt(ind1-1)==s2.charAt(ind2-1))
                        dp[ind1][ind2]=1+dp[ind1-1][ind2-1];
                    else
                        dp[ind1][ind2]=0+Math.max(dp[ind1-1][ind2],dp[ind1][ind2-1]);
                }
            }
            return dp[n][m];
        }
        static int longestPalindromeSubsequence(String s){
            String t = s;
            String ss=new StringBuilder(s).reverse().toString();
            return lcs(ss,t);
        }
        static int minInsertion(String s){
            int n = s.length();
            int k = longestPalindromeSubsequence(s);
            return n-k;
        }
}
}
//*********************************DP-30 Minimum Insertions/Deletions to convert one
string to another******************************************* */
class Solution{
    class TUF{
        static int lcs(String s1,String s2){
            int n=s1.length();
            int m=s2.length();
            int dp[][]=new int[n+1][m+1];
            for(int rows[]: dp)
            Arrays.fill(rows,-1);
            for(int i=0;i<=n;i++){
                dp[i][0] = 0;
            }
            for(int i=0;i<=m;i++){
                dp[0][i] = 0;
            }
            for(int ind1=1;ind1<=n;ind1++){
                for(int ind2=1;ind2<=m;ind2++){
                    if(s1.charAt(ind1-1)==s2.charAt(ind2-1))
                        dp[ind1][ind2]=1+dp[ind1-1][ind2-1];
                    else
                        dp[ind1][ind2]=0+Math.max(dp[ind1-1][ind2],dp[ind1][ind2-1]);
                }
            }
            return dp[n][m];
        }
        static int canYouMake(String str1, String str2){
            int n = str1.length();
            int m = str2.length();
            int k = lcs(str1,str2);
            return (n-k)+(m-k);
        }

        }
    }
//*********************************DP-31 Shortest Common
SuperSequence****************************************** */
class Solution{
    static String Recursion{
        //Str1.length()+Str2.length()-LCS;
    }
    static String shortestSupersequence(String s1, String s2){
        int n=s1.length();
        int m=s2.length();
        int[][] dp =new int[n+1][m+1];
        for (int i =0;i<=n;i++) {
          dp[i][0]=0;
        }
        for (int i=0;i<=m;i++) {
          dp[0][i]=0;
        }
        for (int ind1=1;ind1<=n;ind1++) {
          for (int ind2=1;ind2 <= m; ind2++) {
            if (s1.charAt(ind1-1)==s2.charAt(ind2-1))
              dp[ind1][ind2]=1+dp[ind1-1][ind2-1];
            else
              dp[ind1][ind2]=0  Math.max(dp[ind1-1][ind2],dp[ind1][ind2-1]);
          }
        }
        int len=dp[n][m];
        int i=n;
        int j=m;
        int index=len-1;
        String ans="";

        while(i>0&&j> 0){
          if(s1.charAt(i-1)==s2.charAt(j-1)){
            ans+=s1.charAt(i-1);
            index--;
            i--;
            j--;
          } else if(dp[i-1][j]> dp[i][j-1] {
              ans+=s1.charAt(i-1);
              i--;
          } else{
             ans+=s2.charAt(j-1);
             j--;
          }
        }
        while(i>0){
           ans+=s1.charAt(i-1);
           i--;
        }
        while(j>0){
           ans+=s2.charAt(j-1);
           j--;
        }
        String ans2=new StringBuilder(ans).reverse().toString();
        return ans2;
      }
}
//**********************************DP-32 Distinct
Subsequences****************************************** */
class Solution{
   static int countUtil(String s1, String s2, int ind1, int ind2,int[][] dp){
      if(ind2<0)return 1;
      if(ind1<0)return 0;
      if(dp[ind1][ind2]!=-1)return dp[ind1][ind2];
      if(s1.charAt(ind1)==s2.charAt(ind2)){
         int leaveOne=countUtil(s1,s2,ind1-1,ind2-1,dp);
         int stay=countUtil(s1,s2,ind1-1,ind2,dp);
         return dp[ind1][ind2]=(leaveOne+stay)%prime;
      }
      else{
         return dp[ind1][ind2]=countUtil(s1,s2,ind1-1,ind2,dp);
      }
   }
   static int subsequenceCounting(String s1,String s2,int n,int m){
      int dp[][]=new int[n+1][m+1];
      for(int i=0;i<n+1;i++){
         dp[i][0]=1;
      }
      for(int i=1;i<m+1;i++){
         dp[0][i]=0;
      }
      for(int i=1;i<n+1;i++){
         for(int j=1;j<m+1;j++){
            if(s1.charAt(i-1)==s2.charAt(j-1))
               dp[i][j]=(dp[i-1][j-1]+dp[i-1][j])%prime;
            else
               dp[i][j]=dp[i-1][j];
         }
      }
```

```
        return dp[n][m];
    }

}
//*********************************DP-33 Edit
Distance******************************************* */
class Solution{
    static int editDistanceUtil(String S1,String S2,int i,int j,int[][] dp){
        if(i<0)return j+1;
        if(j<0)return i+1;
        if(dp[i][j]!=-1) return dp[i][j];
        if(S1.charAt(i)==S2.charAt(j))
            return dp[i][j]=0+editDistanceUtil(S1,S2,i-1,j-1,dp);
        // Minimum of three choices
        else return dp[i][j]=1+Math.min(editDistanceUtil(S1,S2,i-1,j-1,dp),Math.
min(editDistanceUtil(S1,S2,i-1,j,dp),editDistanceUtil(S1,S2,i,j-1,dp)));
    }
    static int editDistance(String S1, String S2){
        int n=S1.length();
        int m=S2.length();
        int[][] dp=new int[n+1][m+1];
         for(int i=0;i<=n;i++){
            dp[i][0]=i;
        }
        for(int j=0;j<=m;j++){
            dp[0][j]=j;
        }
        for(int i=1;i<n+1;i++){
            for(int j=1;j<m+1;j++){
                if(S1.charAt(i-1)==S2.charAt(j-1))
                    dp[i][j]=0+dp[i-1][j-1];
                else dp[i][j]=1+Math.min(dp[i-1][j-1],Math.min(dp[i-1][j],dp[i][j-1]));
            }
        }
        return dp[n][m];
    }
}
//*********************************DP-34 Wild Card
Matching******************************************* */
class Solution{
    static boolean isAllStars(String S1,int i){
        for (int j=0;j<=i;j++){
          if (S1.charAt(j)!='*')
            return false;
        }
        return true;
      }
    static int wildcardMatchingUtil(String S1, String S2, int i, int j, int[][] dp) {
        //Base Conditions
        if(i<0&&j<0)return 1;
        if(i<0&&j>=0)return 0;
        if(j<0&&i>=0)return isAllStars(S1,i)?1:0;
        if(dp[i][j]!=-1) return dp[i][j];
        if(S1.charAt(i)==S2.charAt(j)||S1.charAt(i)=='?')
          return dp[i][j]=wildcardMatchingUtil(S1,S2,i-1,j-1,dp);

        else {
          if(S1.charAt(i)=='*')
            return (wildcardMatchingUtil(S1,S2,i-1,j,dp)==1||wildcardMatchingUtil(S1,S2,i,j-
1,dp)==1)?1:0;
          else return 0;
        }
      }

}
//*********************************DP-35 Best Time To Buy and Sell Stocks
I****************************************** */
class Solution{
    class TUF{
        static int maximumProfit(int []Arr){
            // Write your code here.
            int maxProfit=0;
            int mini=Arr[0];

            for(int i=1;i<Arr.length;i++){
                int curProfit=Arr[i]-mini;
                maxProfit=Math.max(maxProfit,curProfit);
                mini=Math.min(mini,Arr[i]);
                }
            return maxProfit;
        }
}}
//*********************************DP-36 Best Time To Buy and Sell Stocks
II****************************************** */
class Solution{
    long getAns(long *Arr, int ind, int buy, int n, vector<vector<long>> &dp ){
        if(ind==n) return 0; //base case
        if(dp[ind][buy]!=-1)return dp[ind][buy];
        long profit;
        if(buy==0){// We can buy the stock
            profit=max(0+getAns(Arr,ind+1,0,n,dp),-Arr[ind]+getAns(Arr,ind+1,1,n,dp));
        }
        if(buy==1){// We can sell the stock
            profit=max(0+getAns(Arr,ind+1,1,n,dp),Arr[ind]+getAns(Arr,ind+1,0,n,dp));
        }
        return dp[ind][buy] = profit;
    }
    static long getMaximumProfit(long Arr[], int n)
{
    long dp[][]=new long[n+1][2];
    for(long row[]: dp)
    Arrays.fill(row,-1);
    dp[n][0]=dp[n][1]=0;
    long profit=0;
    for(int ind=n1;ind>=0;ind--){
        for(int buy=0;buy<=1;buy++){
            if(buy==0){// We can buy the stock
                profit=Math.max(0+dp[ind+1][0], -Arr[ind] + dp[ind+1][1]);
            }
            if(buy==1){// We can sell the stock
                profit=Math.max(0+dp[ind+1][1], Arr[ind] + dp[ind+1][0]);
            }

            dp[ind][buy]=profit;
        }
    }
    return dp[0][0];
}
}
//*********************************DP-37 Best Time To Buy and Sell Stocks
III****************************************** */
class Solution{
    int getAns(vector<int>& Arr,int n,int ind,int buy,int cap,vector<vector<vector<int>>>&
dp){

        if(ind==n||cap==0) return 0; //base case
        if(dp[ind][buy][cap]!=-1)return dp[ind][buy][cap];
        int profit;
        if(buy==0){// We can buy the stock
            profit=max(0+getAns(Arr,n,ind+1,0,cap,dp),-Arr[ind]+getAns(Arr,n,ind+1,1,cap,
dp));
        }
        if(buy==1){// We can sell the stock
            profit=max(0+getAns(Arr,n,ind+1,1,cap,dp),Arr[ind]+getAns(Arr,n,ind+1,0,cap-1,
dp));
        }
        return dp[ind][buy][cap]=profit;
    }
    int maxProfit(vector<int>& Arr, int n)
    {
    vector<vector<vector<int>>> dp(n+1,vector<vector<int>>(2,vector<int>(3,0)));
    for(int ind=n-1;ind>=0;ind--){
        for(int buy=0;buy<=1;buy++){
            for(int cap=1;cap<=2;cap++){
                if(buy==0){// We can buy the stock
                    dp[ind][buy][cap]=max(0+dp[ind+1][0][cap],-Arr[ind]+dp[ind+1][1][cap]);
            }
           if(buy==1){// We can sell the stock
               dp[ind][buy][cap]=max(0+dp[ind+1][1][cap],Arr[ind]+dp[ind+1][0][cap-1]);
                   }
               }
           }
       }
   }
}}
//*********************************DP-38 Best Time To Buy and Sell Stocks
IV******************************************* */
class Solution{
   int getAns(vector<int>& Arr, int n, int ind, int buy, int cap,
vector<vector<vector<int>>>& dp ){

       if(ind==n||cap==0) return 0; //base case
       if(dp[ind][buy][cap]!=-1)return dp[ind][buy][cap];
       int profit;
       if(buy==0){// We can buy the stock
           profit=max(0+getAns(Arr,n,ind+1,0,cap,dp), -Arr[ind]+getAns(Arr,n,ind+1,1,cap,
dp));
       }
       if(buy==1){// We can sell the stock
           profit = max(0+getAns(Arr,n,ind+1,1,cap,dp),Arr[ind]+getAns(Arr,n,ind+1,0,cap-
1,dp));
       }
       return dp[ind][buy][cap]=profit;
   }
   int maximumProfit(vector<int>& Arr, int n, int k)
{
   // Creating a 3d - dp of size [n+1][2][k+1] initialized to 0
   vector<vector<vector<int>>> dp(n+1,vector<vector<int>>(2,vector<int>(k+1,0)));
   for(int ind=n-1;ind>=0;ind--){
       for(int buy=0;buy<=1;buy++){
           for(int cap=1; cap<=k; cap++){
               if(buy==0){// We can buy the stock
                   dp[ind][buy][cap]=max(0+dp[ind+1][0][cap],-Arr[ind]+dp[ind+1][1][cap]);
                }
               if(buy==1){// We can sell the stock
                   dp[ind][buy][cap]=max(0+dp[ind+1][1][cap],Arr[ind]+dp[ind+1][0][cap-1]);
               }
           }
       }
   }
   return dp[0][0][k];
}}
}
//*********************************DP-39 Best Time To Buy and Sell Stocks with
Cooldown******************************************* */
```

```
class Solution{
    static int getAns(int[] Arr, int ind, int buy, int n, int[][] dp ){
        if(ind>=n) return 0; //base case
        if(dp[ind][buy]!=-1)return dp[ind][buy];
        int profit=0;
        if(buy==0){// We can buy the stock
        profit=Math.max(0+getAns(Arr,ind+1,0,n,dp),-Arr[ind]+getAns(Arr,ind+1,1,n,dp));
        }
        if(buy==1){// We can sell the stock
        profit=Math.max(0+getAns(Arr,ind+1,1,n,dp),Arr[ind]+getAns(Arr,ind+2,0,n,dp));
        }
        return dp[ind][buy] = profit;
    }
    static int stockProfit(int[] Arr)
    {
    int n = Arr.length;
    int dp[][]=new int[n+2][2];
    for(int ind=n-1;ind>=0;ind--){
        for(int buy=0;buy<=1;buy++){
            int profit=0;
            if(buy==0){// We can buy the stock
                profit=Math.max(0+dp[ind+1][0],-Arr[ind]+dp[ind+1][1]);
            }
            if(buy==1){// We can sell the stock
                profit=Math.max(0+dp[ind+1][1],Arr[ind]+dp[ind+2][0]);
            }
            dp[ind][buy]=profit;
        }
    }
    return dp[0][0];
}
}
//**********************************DP-40 Buy and Sell Stock with Transaction
Fee******************************************** */
class Solution{
    static int getAns(int[] Arr, int ind, int buy, int n, int fee, int[][] dp ){
        if(ind==n) return 0; //base case
        if(dp[ind][buy]!=-1)return dp[ind][buy];
        int profit=0;
        if(buy==0){// We can buy the stock
            profit=Math.max(0+getAns(Arr,ind+1,0,n,fee,dp),-Arr[ind]+getAns(Arr,ind+1,1,n,
fee,dp));
        }
        if(buy==1){// We can sell the stock
            profit=Math.max(0+getAns(Arr,ind+1,1,n,fee,dp),Arr[ind]-fee+getAns(Arr,ind+1,
0,n,fee,dp));
        }
        return dp[ind][buy] = profit;
    }
    static int maximumProfit(int n, int fee, int[] Arr)
{
      if(n==0) return 0;
    int dp[][]=new int[n+1][2];
      for(int ind=n-1;ind>=0;ind--){
        for(int buy=0;buy<=1;buy++){
            int profit=0;
            if(buy==0){// We can buy the stock
                profit=Math.max(0+dp[ind+1][0],-Arr[ind]+dp[ind+1][1]);
            }
            if(buy==1){// We can sell the stock
                profit=Math.max(0+dp[ind+1][1],Arr[ind]-fee+dp[ind+1][0]);
            }
            dp[ind][buy]=profit;
        }
    }
    return dp[0][0];
}

}
//**********************************DP-41 Longest Increasing
Subsequence|Memoization******************************************** */
class Solution{
    class TUF{
        static int getAns(int arr[], int n,  int ind, int prev_index,int[][] dp){
            if(ind==n) return 0;
            if(dp[ind][prev_index+1]!=-1)return dp[ind][prev_index+1];
            int notTake=0+getAns(arr,n,ind+1,prev_index,dp);
            int take=0;
            if(prev_index==-1||arr[ind]>arr[prev_index]){
                take=1+getAns(arr,n,ind+1,ind,dp);
            }
            return dp[ind][prev_index+1]=Math.max(notTake,take);
        }
    }

}
//**********************************DP-42 Printing Longest Increasing
Subsequence******************************************** */
class Solution{
    class TUF{
        static int longestIncreasingSubsequence(int arr[],int n){
            int dp[][]=new int[n+1][n+1];
            for(int ind=n-1;ind>=0;ind --){
                for (int prev_index=ind-1;prev_index>=-1;prev_index --){
                    int notTake=0+dp[ind+1][prev_index +1];
                    int take=0;
                    if(prev_index==-1||arr[ind] > arr[prev_index]){
                        take=1+dp[ind+1][ind+1];
                    }
                    dp[ind][prev_index+1]=Math.max(notTake,take);
                }
            }
            return dp[0][0];
        }
        static int longestIncreasingSubsequence(int arr[],int n){
            int dp[]=new int[n];
            Arrays.fill(dp,1);
            for(int i=0;i<=n-1;i++){
                for(int prev_index=0;prev_index<=i-1;prev_index ++){
                    if(arr[prev_index]<arr[i]){
                        dp[i]=Math.max(dp[i],1+dp[prev_index]);
                    }
                }
            }
            int ans=-1;
            for(int i=0;i<=n-1;i++){
                ans=Math.max(ans, dp[i]);
            }
            return ans;
        }
        static int longestIncreasingSubsequence(int arr[],int n){
            int[] dp=new int[n];
            Arrays.fill(dp,1);
            int[] hash=new int[n];
            Arrays.fill(hash,1);

            for(int i=0;i<=n-1;i++){
                hash[i]=i; // initializing with current index
                for(int prev_index=0;prev_index<=i-1;prev_index ++){
                    if(arr[prev_index]<arr[i]&&1+dp[prev_index]>dp[i]){
                        dp[i]=1+dp[prev_index];
                        hash[i]=prev_index;
                    }
                }
            }
            int ans = -1;
            int lastIndex =-1;
            for(int i=0;i<=n-1;i++){
                if(dp[i]>ans){
                    ans=dp[i];
                    lastIndex=i;
                }
            }
            ArrayList<Integer> temp=new ArrayList<>();
            temp.add(arr[lastIndex]);
            while(hash[lastIndex]!=lastIndex){ // till not reach the initialization value
                lastIndex=hash[lastIndex];
```

```
                temp.add(arr[lastIndex]);
            }
            for(int i=temp.size()-1; i>=0; i--){
                System.out.print(temp.get(i)+" ");
            }
            return ans;
        }
}}
//*********************************DP-43 Longest Increasing Subsequence|Binary
Search******************************************* */
class Solution{
    int longestIncreasingSubsequence(int arr[], int n){
        vector<int> temp;
        temp.push_back(arr[0]);
        int len=1;
        for(int i=1;i<n;i++){
            if(arr[i]>temp.back()){
               temp.push_back(arr[i]);
               len++;
            }
            else{
                int ind=lower_bound(temp.begin(),temp.end(),arr[i])-temp.begin();
                temp[ind]=arr[i];
            }
        }
        return len;
    }
}
//*********************************DP-44 Largest Divisible
Subset******************************************* */
class Solution{
    vector<int> divisibleSet(vector<int>& arr){
        int n=arr.size();
        sort(arr.begin(),arr.end());
        vector<int> dp(n,1);
        vector<int> hash(n,1);
        for(int i=0;i<=n-1;i++){
            hash[i]=i; // initializing with current index
            for(int prev_index=0;prev_index <=i-1;prev_index ++){
                if(arr[i]%arr[prev_index]==0&&1+dp[prev_index]>dp[i]){
                    dp[i]=1+dp[prev_index];
                    hash[i]=prev_index;
                }
            }
        }
        int ans=-1;
        int lastIndex=-1;
        for(int i=0;i<=n-1;i++){
            if(dp[i]>ans){
                ans=dp[i];
                lastIndex=i;
            }
        }
        vector<int> temp;
        temp.push_back(arr[lastIndex]);
        while(hash[lastIndex]!=lastIndex){ // till not reach the initialization value
            lastIndex=hash[lastIndex];
            temp.push_back(arr[lastIndex]);
        }
        reverse(temp.begin(),temp.end());
        return temp;
    }}
}
//*********************************DP-45 Longest String
chain******************************************** */
class Solution{

bool compare(string& s1, string& s2){
    if(s1.size()!=s2.size()+1) return false;
    int first=0;
    int second=0;
    while(first<s1.size()){
        if(second<s2.size()&&s1[first]==s2[second]){
            first++;
            second++;
        }
        else first++;
    }
    if(first==s1.size()&&second == s2.size()) return true;
    else return false;
}
bool comp(string& s1,string& s2){
    return s1.size()<s2.size();
}
int longestStrChain(vector<string>& arr){
    int n=arr.size();
    sort(arr.begin(),arr.end(),comp);
    vector<int> dp(n,1);
    int maxi=1;
    for(int i=0;i<=n-1;i++){
        for(int prev_index=0;prev_index<=i-1;prev_index++){
            if(compare(arr[i],arr[prev_index])&&1+dp[prev_index]>dp[i]){
                dp[i]=1+dp[prev_index];
            }
        }
        if(dp[i]>maxi)
            maxi=dp[i];
    }
    return maxi;
}}
}
//**********************************DP-46 Longest Bitonic
Sequence******************************************** */
class Solution{
    class TUF{
        static int longestBitonicSequence(int[] arr, int n){
            int[] dp1=new int[n];
            int[] dp2=new int[n];
            Arrays.fill(dp1,1);
            Arrays.fill(dp2,1);
            for(int i=0;i<=n-1;i++){
                for(int prev_index=0;prev_index<=i-1;prev_index++){
                    if(arr[prev_index]<arr[i]){
                        dp1[i]=Math.max(dp1[i],1+dp1[prev_index]);
                    }
                }
            }
            for(int i=n-1;i>=0;i--){
                for(int prev_index=n-1;prev_index >i;prev_index--){
                    if(arr[prev_index]<arr[i]){
                        dp2[i]=Math.max(dp2[i],1+dp2[prev_index]);
                    }
                }
            }
            int maxi=-1;
            for(int i=0;i<n;i++){
                maxi=Math.max(maxi,dp1[i]+dp2[i]-1);
            }
            return maxi;
         } }
}
//**********************************DP-47 Number of Longest Increasing
Subsequence******************************************** */
class Solution{
    class TUF{
        static int findNumberOfLIS(int[] arr){
            int n=arr.length;
            int[] dp=new int[n];
            int[] ct=new int[n];
            Arrays.fill(dp,1);
            Arrays.fill(ct,1);
            int maxi=1;
            for(int i=0;i<=n-1;i++){
                for(int prev_index=0;prev_index<=i-1;prev_index++){
                    if(arr[prev_index]<arr[i]&&dp[prev_index]+1>dp[i]){
                        dp[i]=dp[prev_index]+1;
                        ct[i]=ct[prev_index];
                    }
                    else if(arr[prev_index]<arr[i]&&dp[prev_index]+1==dp[i]){
                        ct[i]=ct[i]+ct[prev_index];
                    }
                }
                 maxi=Math.max(maxi,dp[i]);
            }
            int nos =0;
            for(int i=0;i<=n-1;i++){
               if(dp[i]==maxi) nos+=ct[i];
            }
            return nos;
           }   }
}
//*********************************DP-48 Matrix Change
Multiplication******************************************* */
class Solution{
    static int f(int arr[],int i,int j,int[][] dp){
        if(i==j)return 0;
        if(dp[i][j]!=-1)return dp[i][j];
        int mini=Integer.MAX_VALUE;
        for(int k=i;k<=j-1;k++){
        int ans=f(arr,i,k,dp)+f(arr,k+1,j,dp)+arr[i-1]*arr[k]*arr[j];
        mini=Math.min(mini,ans);
        }
        return mini;
    }
}
//*********************************DP-49 Matrix Change Multiplication|Bottom
Up******************************************* */
class Solution{
    static int matrixMultiplication(int[] arr,int N){
        int [][] dp=new int[N][N];
        for(int row[]: dp)
        Arrays.fill(row,-1);
        for(int i=1;i<N;i++){
            dp[i][i]=0;
        }
        for(int i=N-1;i>=1;i--){
            for(int j=i+1;j<N;j++){
                int mini=Integer.MAX_VALUE;
                for(int k=i;k<=j-1;k++){
                    int ans=dp[i][k]+dp[k+1][j]+arr[i-1]*arr[k]*arr[j];
                    mini=Math.min(mini,ans);
                }
                dp[i][j] = mini;
            }
        }
        return dp[1][N-1];
    }
}
//*********************************DP-50 Minimum Cost to Cut the
Rod****************************************** */
class Solution{
    int f(int i, int j, vector<int> &cuts,  vector<vector<int>> &dp){
        if(i>j)return 0;
        if(dp[i][j]!=-1)
            return dp[i][j];
        int mini=INT_MAX;
        for(int ind=i;ind<=j;ind++){
            int ans=cuts[j+1]-cuts[i-1]+f(i,ind-1,cuts,dp)+f(ind+1,j,cuts,dp);
            mini=min(mini, ans);
        }
        return dp[i][j]=mini;
    }


    int cost(int n, int c, vector<int> &cuts){
        cuts.push_back(n);
        cuts.insert(cuts.begin(),0);
        sort(cuts.begin(),cuts.end());
        vector<vector<int>> dp(c+1,vector<int>(c+1,-1));
        return f(1,c,cuts,dp);
    }

    int cost(int n, int c, vector<int> &cuts){
        cuts.push_back(n);
        cuts.insert(cuts.begin(),0);
        sort(cuts.begin(),cuts.end());
        vector<vector<int>> dp(c+2,vector<int>(c+2,0));
        for(int i=c;i>=1;i--){
            for(int j=1;j<=c;j++){
                if(i>j) continue;
                int mini=INT_MAX;
                for(int ind=i;ind<=j;ind++){
                    int ans=cuts[j+1]-cuts[i-1]+dp[i][ind-1]+dp[ind+1][j];
                    mini=min(mini, ans);
                }
                dp[i][j]=mini;
            }
        }
        return dp[1][c];
    }
}}
//*********************************DP-51 Burst
Balloons****************************************** */
class Solution{
    int f(int i,int j,vector<int>& a,vector<vector<int>> &dp){
        if(i>j)return 0;
        if(dp[i][j]!=-1)return dp[i][j];
        int maxi=INT_MIN;
        for(int ind=i;ind<=j;ind++){
            int cost=a[i-1]*a[ind]*a[j+1]+f(i,ind-1,a,dp)+f(ind+1,j,a,dp);
            maxi=max(maxi,cost);
        }
        return dp[i][j]=maxi;
    }
    int maxCoins(vector<int>& a)
    {
        // Write your code here.
        int n=a.size();
        a.push_back(1);
        a.insert(a.begin(),1);
        vector<vector<int>> dp(n+1,-1);
        return f(1,n,a,dp);
    }
    int maxCoins(vector<int>& a)
    {
      // Write your code here.
    int n=a.size();
    a.push_back(1);
    a.insert(a.begin(),1);
    vector<vector<int>> dp(n+1,0);
    for(int i=n-1;i>=1;i--){
        for(int j=1;j<=n;j++){
            if(i>j)continue;
            int maxi=INT_MIN;
            for(int ind=i;ind<=j;ind++){
                    int cost=a[i-1]*a[ind]*a[j+1]+dp[i][ind-1]+dp[ind+1][j];
                    maxi=max(maxi,cost);
            }
            dp[i][j]=maxi;

        }
    }
    return dp[1][n];
       }   }
}
//**********************************DP-52 Evaluate Boolean Expression to
True******************************************* */
class Solution{
    int f(int i,int j,boolean isTrue,String str){
        if(i>j)return 0;
        if(i==j){
            if(isTrue==1){
                return str.charAt(i)=='T';
            }
            else return str.charAt(i)=='F';
        }
        if(dp[i][j][isTrue]!=-1)return dp[i][j][isTrue];
        int ways=0;
        for(int ind=i+1;ind<=j-1;ind=ind+2){
            int LeftTrue=f(i,ind-1,1,str);
            int LeftFalse=f(i,ind-1,0,str);
            int RightTrue=f(ind+1,j,1,str);
            int RightFalse=f(ind+1,j,0,str);


        if(str.charAt(ind)=='&'){
            if(isTrue==true) ways=ways+(LeftTrue*RightTrue);
            else ways=ways+(LeftFalse*RightTrue)+(LeftTrue*RightFalse)
+(LeftFalse*RightFalse);
        }
        else if(str.charAt(ind)=='|'){
            if(isTrue==true) ways=ways+(LeftFalse*RightTrue)+(LeftTrue*RightFalse)
+(LeftTrue*RightTrue);
            else ways=ways+(LeftFalse*RightFalse);
        }
        else if(str.charAt(ind)=='^'){
            if(isTrue==true) ways=ways+((LeftFalse*RightTrue)+(LeftTrue*RightFalse))
            else ways=ways+(LeftTrue*RightTrue)+(LeftFalse*RightFalse);
        }
    }
    }
    return dp[i][j][isTrue]=ways;
}
//*********************************DP-53 Palindrome Partitioning
II****************************************** */
class Solution{
    int f(int i,String str){
        if(i==str.length())return 0;
        if(dp[i]!=-1)return dp[i];
        String temp="";
        int minCost=Integer.MAX_VALUE;
        for(int j=i;j<str.length();j++){
            temp=temp+str.charAt(j);
            if(isPalindrome(temp)==true){
                int cost=1+f(j+1,str);
            }
            minCost=Math.min(minCost,cost);
        }

        return dp[i]=minCost;
    }

    int f(int i,String str){
        int dp[]=new int[n+1];
        for(int i=1;i<n;i++){
            dp[i]=0;
        }
        int n=str.length();
        for(int i=n-1;i>=1;i--){
            int minCost=Integer.MAX_VALUE;
            for(int j=i;j<n;j++){
                if(isPalindrome(i,j,str)==true){
                    int cost=1+dp[j+1];
                    minCost=Math.min(minCost,cost);
                }
            }
            dp[i]=minCost;
        }
        return dp[0]-1;
}}
//*********************************DP-54 Partition Array for Maximum
Sum****************************************** */
class Solution{
    f(int i,int arr[],int k){
        if(i==n)return 0;
        if(dp[i]!=-1)return dp[i];
        int maxAns=Integer.MIN_VALUE,len=0,maxi=Integer.MIN_VALUE;
        for(int j=i;j<Math.min(j+k,n);j++){
            len++;
            maxi=Math.max(arr[j],maxi);
            int sum=len*maxi+f(j+1,arr,k);
            maxAns=Math.max(sum,maxAns);
        }
        return dp[i]=maxAns;
    }

    int f(int i,int arr[]){
        int dp[]=new int[n+1];
        for(int i=1;i<n;i++){
            dp[i]=0;
        }
        dp[n]=0;
        int n=arr.length;
        for(int i=n-1;i>=1;i--){
            int maxAns=Integer.MIN_VALUE,len=0,maxi=Integer.MIN_VALUE;
            for(int j=i;j<Math.min(j+k,n);j++){
                len++;
                maxi=Math.max(arr[j],maxi);
                int sum=len*maxi+dp[j+1];
                maxAns=Math.max(sum,maxAns);
            }
            dp[i]=maxAns;
        }
            return dp[0];
}}
//*********************************DP-55 Maximum Rectangle Area with All
1's****************************************** */
class Solution{
    //Area of Largest Histogram
    public static int maximalAreaOfSubMatrixOfAll1(int[][] mat,int n,int m){
        int maxArea=0;
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(mat[i][j]==1)height[j]+=mat[i][j];
                else height[j]=0;
                int area=getMaxAreainHistogram(height);
                maxArea=Math.max(maxArea,area);
            }
        }
        return maxArea;
      }
}
//*********************************DP-56 Count Square SubMatrices with All
1's****************************************** */
class Solution{
    public static int countSquares(int n, int m, int[][] arr) {
                // Write your code here
        int dp[][]=new int[n][m];
        for(int j=0;j<m;j++)dp[0][j]=arr[0][j];
        for(int i=0;i<n;i++)dp[i][0]=arr[i][0];
        for(int i=1;i<n;i++){
            for(int j=1;j<m;j++){
                if(arr[i]==1)dp[i][j]=Math.min(Math.min(dp[i-1][j],dp[i][j-1]),dp[i-1][j-1])+1;
                else dp[i][j]=0;
            }
        }
        int sum=0;
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                sum+=dp[i][j];
            }
        }
        return sum;
    }
}

}
//Format
//********************** */
public class Solution {
    public static int findWays(int arr[], int k) {
        // Write your code here..
        int n=arr.length;
        int dp[][]=new int[n+1][k+1];
        for(int rows[]: dp){Arrays.fill(rows,0);}
        for(int i=0;i<n;i++){dp[i][0]=1;}
        if(arr[0]<k)dp[0][arr[0]]=1;

        for(int ind=1;ind<n;ind++){
            for(int target=0;target<=k;target++){
                int notTake=dp[ind-1][target];
                int take=0;
                if(arr[ind]<=target){
                    take=dp[ind-1][target-arr[ind]];
                }
                dp[ind][target]=take+notTake;
            }
        }
        return dp[n-1][k];
        //return f(n-1,k,arr,dp);
    }
    public static int f(int ind,int target,int arr[],int dp[][]){
        if(target==0){return 1;}
        if(ind==0){return arr[0]==target?1:0;}
        if(dp[ind][target]!=-1){return dp[ind][target];}
        int notTake=f(ind-1,target,arr,dp);
        int take=0;
        if(arr[ind]<=target){
            take=f(ind-1,target-arr[ind],arr,dp);
        }
        return dp[ind][target]=take+notTake;
    }
}
int dp[][]=new int[][];
for(int rows[]: dp){Arrays.fill(rows,-1);}
return f();

if(){}
if(dp[ind][target]!=-1){return dp[ind][target];}
int notTake=f();
int take=0;
if(){
    take=f()
}
return dp[][]=
```