DYNAMIC PROGRAMMING

Content

11.1. 0/1 knapsack Problem

Statement : You are given weights and values of **N** items, put these items in a knapsack of capacity **W** to get the maximum total value in the knapsack. Note that we have only **one quantity of each item**.
In other words, given two integer arrays **val[0..N-1]** and **wt[0..N-1]** which represent values and weights associated with **N** items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of **val[]** such that sum of the weights of this subset is smaller than or equal to **W.** You cannot break an item, **either pick the complete item, or don't pick it (0-1 property)**.

```
 static int knapSack(int W, int wt[], int val[], int n)
   {
      int i, w;
      int K[][] = new int[n+1][W+1];

      // Build table K[][] in bottom up manner
      for (i = 0; i <= n; i++)
      {
         for (w = 0; w <= W; w++)
         {
            if (i==0 || w==0)
               K[i][w] = 0;
            else if (wt[i-1] <= w)
                K[i][w] = Math.max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w]);
            else
                K[i][w] = K[i-1][w];
         }
      }

      return K[n][W];
   }
      public static void main (String[] args) {
               Scanner sc = new Scanner(System.in);
               int t = sc.nextInt();
               for(int i=0;i<t;++i){
                  int n = sc.nextInt();
                  int w = sc.nextInt();
                  int val[] = new int[n];
                  for(int j=0;j<n;j++){
                     val[j] = sc.nextInt();
                  }
                  int wt[] = new int[n];
                  for(int j=0;j<n;++j){
                    wt[j] = sc.nextInt();
                }
                System.out.println(knapSack(w,wt,val,n));
            }
        }
```

11.2. Minimum no. of Jumps

Statement : Given an array of integers where each element represents the max number of steps that can be made forward from that element. The task is to find the minimum number of jumps to reach the end of the array (starting from the first element). If an element is **0**, then cannot move through that element.

```
static int minimumSteps(int arr[],int n){
        if(n==0 || arr[0]==0){
            return -1;
        }
        int steps[] = new int[n];
        for(int i=1;i<n;i++){
            steps[i] = -1;
            for(int j=0;j<i;j++){
                if(i<=j+arr[j] && steps[j]!=-1){
                    steps[i] = 1+steps[j];
                    break;
                }
            }
        }
        return steps[n-1];
    }
        public static void main (String[] args) {
                Scanner sc = new Scanner(System.in);
                int t = sc.nextInt();
                for(int i=0;i<t;i++){
                    int n = sc.nextInt();
                    int arr[] = new int[n];
                    for(int j=0;j<n;j++){
                        arr[j] = sc.nextInt();
                    }
                    System.out.println(minimumSteps(arr,n));
                }
        }
```

11.3. Shortest common SuperSequence of two string

Statement : Given two strings **str1** and **str2**, find the length of the smallest string which has both, str1 and str2 as its sub-sequences.
Note: str1 and str2 can have both uppercase and lowercase letters.

Input:	**Output :**
abcd xycd	6
efgh jghi	6

```
static int lcs(char X[],char Y[],int m,int n){
    int dp[][] = new int[m+1][n+1];
    for(int i=0;i<=m;i++){
        for(int j=0;j<=n;j++){
            if(i==0 || j==0){
                dp[i][j] = 0;
            }else if(X[i-1]==Y[j-1]){
                dp[i][j] = dp[i-1][j-1]+1;
            }else{
                dp[i][j] = Math.max(dp[i-1][j],dp[i][j-1]);
            }
        }
    }
    return dp[m][n];
}
      public static void main (String[] args) {
              Scanner sc = new Scanner(System.in);
              int t = sc.nextInt();
              for(int i=0;i<t;i++){
                  String a = sc.next();
                  String b = sc.next();
                  char X[] = a.toCharArray();
                  char Y[] = b.toCharArray();
                  int m = X.length;
                  int n = Y.length;
                  int lcs = lcs(X,Y,m,n);
                  System.out.println(m+n-lcs);
              }
      }
```

11.4. Longest Increasing Subsequence : O(n^2) and O(nlogn) is also exits

Statement : Given a sequence **A** of size **N**, find the **length** of the **longest increasing subsequence** from a given sequence .
The longest increasing subsequence means to find a subsequence of a given sequence in which the subsequence's elements are in sorted order, lowest to highest, and in which the subsequence is as long as possible. This subsequence is not necessarily contiguous, or unique.

Note: Duplicate numbers are not counted as increasing subsequence.

```
Input : 6              output : 3
5 8 3 7 9 1             5 7 9
```

```
class GFG {
    static int lis(int arr[],int n){
        int max = 1;
        int dp[] = new int[n];
        dp[0] = 1;
        for(int i=1;i<n;i++){
            dp[i] = 1;
            for(int j=0;j<i;j++){
                if(arr[j]<arr[i] && dp[j]+1>dp[i]){
                    dp[i] = dp[j] + 1;
                }
            }
            if(dp[i]>max)
                max = dp[i];
        }
        return max;
    }
          public static void main (String[] args) {
                    Scanner sc = new Scanner(System.in);
                    int test = sc.nextInt();
                    for(int t=0;t<test;t++){
                        int n = sc.nextInt();
                        int arr[] = new int[n];
                        for(int i=0;i<n;i++){
                            arr[i] = sc.nextInt();
                        }
                        System.out.println(lis(arr,n));
                    }
          }
}
```

11.5. Longest Common Subsequence of two string

Statement : Given two sequences, find the length of longest subsequence present in both of them. Both the strings are of uppercase.

```
Input : 6 6          Output : 3
ABCDGH              ADH
AEDFHR
```

```
static int LCS(char X[],char Y[],int m,int n){
    int dp[][] = new int[m+1][n+1];
    for(int i=0;i<=m;i++){
        for(int j=0;j<=n;j++){
            if(i==0 || j==0){
                dp[i][j]=0;
            }else if(X[i-1]==Y[j-1]){
                dp[i][j] = dp[i-1][j-1]+1;
            }else{
                dp[i][j] = Math.max(dp[i-1][j],dp[i][j-1]);
            }
        }
    }
    return dp[m][n];
}
      public static void main (String[] args) {
              Scanner sc = new Scanner(System.in);
              int t = sc.nextInt();
              for(int i=0;i<t;i++){
                  int m = sc.nextInt();
                  int n = sc.nextInt();
                  String a = sc.next();
                  String b = sc.next();
                  char X[] = a.toCharArray();
                  char Y[] = b.toCharArray();
                  System.out.println(LCS(X,Y,m,n));
              }
      }
```

11.6. Maximum sum increasing subsequence

Statement : Given an array **A** of **N** positive integers. Find the **sum** of **maximum sum increasing subsequence** of the given array.

Input : 7   Output : 106
1 101 2 3 100 4 5

All the increasing subsequences are : (1,101); (1,2,3,100); (1,2,3,4,5). Out of these (**1, 2, 3, 100**) has maximum sum,i.e., 106.

```
private static int sumOfIncresingSequence(int n,int arr[]){
        int temp[] = new int[n];
        for(int i=0;i<n;i++){
            temp[i] = arr[i];
        }
        for(int i=1;i<n;i++){
            for(int j=0;j<i;j++){
                if(arr[j]<arr[i]){
                    temp[i] = Math.max(temp[j]+arr[i],temp[i]);
                }
            }
        }
        int max = temp[0];
        for(int i=1;i<n;i++){
            if(temp[i]>max){
                max = temp[i];
            }
        }
        return max;
    }

          public static void main (String[] args) {
                    Scanner sc = new Scanner(System.in);
                    int t = sc.nextInt();
                    for(int i=0;i<t;i++){
                        int n = sc.nextInt();
                        int arr[] = new int[n];
                        for(int j=0;j<n;j++){
                            arr[j] = sc.nextInt();
                        }
                        System.out.println(sumOfIncresingSequence(n,arr));
                    }
          }
```

11.7. Edit Distance

Statement : Given two strings **str1** and **str2** and below operations that can performed on str1. Find minimum number of edits (operations) required to convert ‘str1′ into ‘str2′.

1. Insert
2. Remove
3. Replace

All of the above operations are of cost=1.
Both the strings are of lowercase.

Input : 4 5 Output : 1
geek gesek

One operation is required to make 2 strings same i.e. removing 's' from str2.

```
 static int editDistDP(String str1, String str2, int m, int n)
   {
        // Create a table to store results of subproblems
        int dp[][] = new int[m+1][n+1];

        // Fill d[][] in bottom up manner
        for (int i=0; i<=m; i++)
        {
                for (int j=0; j<=n; j++)
                {
                   // If first string is empty, only option is to
                   // insert all characters of second string
                   if (i==0)
                       dp[i][j] = j;  // Min. operations = j

                   // If second string is empty, only option is to
                   // remove all characters of second string
                   else if (j==0)
                       dp[i][j] = i; // Min. operations = i

                    // If last characters are same, ignore last char
                    // and recur for remaining string
                    else if (str1.charAt(i-1) == str2.charAt(j-1))
                        dp[i][j] = dp[i-1][j-1];

                    // If the last character is different, consider all
                    // possibilities and find the minimum
```

```
            else
                dp[i][j] = 1 + min(dp[i][j-1],  // Insert
                                   dp[i-1][j],  // Remove
                                  / dp[i-1][j-1]); // Replace
        }
    }

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

11.8. Coin Change

Statement : Given a value N, find the number of ways to make change for N cents, if we have infinite supply of each of S = { S1, S2, .. , Sm} valued coins. The order of coins doesn't matter. For example, for N = 4 and S = {1,2,3}, there are four solutions: {1,1,1,1},{1,1,2},{2,2},{1,3}. So output should be 4. For N = 10 and S = {2, 5, 3, 6}, there are five solutions: {2,2,2,2,2}, {2,2,3,3}, {2,2,6}, {2,3,5} and {5,5}. So the output should be 5.

```
class GFG {
   static int coinChange(int arr[],int m,int n){
      int dp[] = new int[n+1];
      dp[0] = 1;
      for(int i=0;i<m;i++){
         for(int j=arr[i];j<=n;j++){
            dp[j] += dp[j-arr[i]];
         }
      }
      return dp[n];
   }
        public static void main (String[] args) {
             Scanner sc = new Scanner(System.in);
             int test = sc.nextInt();
             for(int t=0;t<test;t++){
                int m = sc.nextInt();
                int arr[] = new int[m];
                for(int i=0;i<m;i++){
                   arr[i] = sc.nextInt();
                }
                int n = sc.nextInt();
                System.out.println(coinChange(arr,m,n));
             }
        }
}
```

11.9. subset sum problem : O(n*sum)

Statement : Given a set of numbers, check whether it can be partitioned into two subsets such that the sum of elements in both subsets is same or not.

Input : 4 output: YES
1 5 11 5

There exists two subsets such that {1, 5, 5} and {11}.

```
class GFG {
    static boolean subset(int arr[], int n,int sum){
        boolean dp[][] = new boolean[n+1][sum+1];
        for(int i=0;i<n+1;i++){
            dp[i][0] = true;
        }
        for(int i=1;i<sum+1;i++){
            dp[0][i] = false;
        }
        for(int i=1;i<=n;i++){
            for(int j=1;j<=sum;j++){
                dp[i][j] = dp[i-1][j];
                if(arr[i-1]<=j){
                    dp[i][j] |= dp[i-1][j-arr[i-1]];
                }
            }
        }
        return dp[n][sum];
    }
        public static void main (String[] args) {
                Scanner sc = new Scanner(System.in);
                int test = sc.nextInt();
                for(int t=0;t<test;t++){
                    int n = sc.nextInt();
                    int arr[] = new int[n];
                    int sum = 0;
                    for(int i=0;i<n;i++){
                        arr[i] = sc.nextInt();
                        sum += arr[i];
                    }
                    String ans = "NO";
                    if(sum%2==0 && subset(arr,n,sum/2)){
                        ans = "YES";
                    }
                    System.out.println(ans);
                }
            }
}
```

11.10. Egg Droping Puzzle

Statement : Suppose you have N eggs and you want to determine from which floor in a K-floor building you can drop an egg such that it doesn't break. You have to determine the minimum number of attempts you need in order find the critical floor in the worst case while using the best strategy.There are few rules given below.

- An egg that survives a fall can be used again.
- A broken egg must be discarded.
- The effect of a fall is the same for all eggs.
- If the egg doesn't break at a certain floor, it will not break at any floor below.
- If the eggs breaks at a certain floor, it will break at any floor above.

```
static int trialsForDropEggs(int n,int k){
        int dp[][] = new int[n+1][k+1];
        for(int i=0;i<=n;i++){
            dp[i][0] = 0;
            dp[i][1] = 1;
        }
        for(int j=0;j<=k;j++){
            dp[0][j] = 0;
            dp[1][j] = j;
        }
        for(int i=2;i<=n;i++){
            for(int j=2;j<=k;j++){
                dp[i][j] = Integer.MAX_VALUE;
                for(int x=1;x<=j;x++){
                    int res = 1 + Math.max(dp[i-1][x-1],dp[i][j-x]);
                    if(res<dp[i][j]){
                        dp[i][j] = res;
                    }
                }
            }
```

```
    }
    return dp[n][k];
}
     public static void main (String[] args) {
              Scanner sc = new Scanner(System.in);
              int t = sc.nextInt();
              for(int i=0;i<t;i++){
                 int n = sc.nextInt();
                 int k = sc.nextInt();
                 System.out.println(trialsForDropEggs(n,k));
              }
      }
```