Submission Link: https://forms.gle/TFAJ4BfkCPu2R6fT9

Submission Guidelines:

1. You can code all of them either in Python or Java. But you should choose 1 language for all tasks.
2. For each task write separate python files like task1.py, task2.py and so on.
3. For each task include the input files(if any) in the submission folder.
4. All files (input and code) **MUST** be put in a folder. The folder **MUST** be named following the format **sec_ID_labNo**. Zip this folder. This will result in a zip format of **sec_ID_labNo.zip**. (Example: 2_20101234_6.zip). Submit this zip in the above mentioned submission link.
5. You **MUST** follow all the guidelines, naming/file/zipping convention stated above. Failure to follow instructions will result in straight 50% mark deduction.

Problem Description:

The following tasks needed to be solved by using dynamic Programming. Dynamic programming is a way of finding the optimal solution of a problem where it reduces the call of functions by storing their results in every sequence. It compares the results of every step and from the comparison finds out the optimal solution. Longest common subsequence (LCS) is such an algorithm that functions as a dynamic approach. It basically finds out the longest subsequence which is common to the given sequences.
Read the task descriptions carefully and implement the algorithm using either Java or Python or any language you are comfortable with. The output format **MUST** match exactly as shown in each task.

Problem 1 [5 marks]

We would like to find the minimum number of steps required to get 0 from any number, when you can only subtract a digit present in that number in a single step.

For example, if you are given 25. Then you can either subtract 2 or 5 from 25. Let's subtract 5 then we will get 20. Then we can subtract 2 and get 18 and so on. The minimum number of steps to get from 25 to 0 is shown below:

25→20→18→10→9→0.

Now you need to write a DP program to find the minimum number of steps to solve the problem for any input. Input range will be from 0 to 999.

Pseudo-code is not given because it is quite a basic dp problem!

Problem 2 (LCS) [10 Marks]:

PUBGM (Player Unknown's BattleGrounds Mobile) is one of the most popular online battle royale games. PMPL (PUBGM Pro League) is the biggest tournament of south asia and Future Station a team from Bangladesh has qualified for the final round of the tournament. MagneT also known as "The zone magnet" for his accuracy of in-game zone predictions is the IGL (Team leader) of the team and he predicted the zone sequence before a match in the finals. For the match of map Erangel his prediction was,

Zone number	Zone center	Keyword
1	Yasnaya	Y
2	Pochinki	P
3	School	S
4	Rozhok	R
5	Farm	F
6	Mylta	M
7	Shelter	H
8	Prison	I

On that match the team had a very good result and actual zone sequence of that match was:

Zone number	Zone center	Keyword
1	Yasnaya	Y
2	Rozhok	R
3	School	S
4	Pochinki	P
5	Farm	F
6	Mylta	M
7	Shelter	H
8	Prison	I

Now, you have to find out the longest common zone sequence from MagneT's prediction and actual match by using LCS (Longest Common Subsequence) algorithm. Then verify the correctness of MagneT's prediction. For correctness you will use the formula given below.

Correctness = (Length of longest common zone sequence × 100) ÷ Number of zones

Note: Zone meta given here is completely fictional. Please do not match it with the real game zone meta.

Hint: You have to store zone center values according to keywords first. You can use the following pseudo code of LCS algorithm.

```
LCS(X, Y):
    m <- length of X + 1
    n <- length of Y + 1
    for i <- 1 to m:
        c[i,0] <- 0
        t[i,0] <- null/None
    for j <- 1 to n:
        c[0,j] <- 0
        t[0,j] <- null/None
    for i <- 1 to m:
        for j <-1 to n:
            if X[i] = Y[j]:
```

```
            c[i,j] <- c[i-1,j-1]+1
            t[i,j] <- diagonal
    else if c[i-1,j] >= c[i,j-1]:
            c[i,j] <- c[i-1,j]
            t[i,j] <- up
    else:
            c[i,j] <- c[i,j-1]
            t[i,j] <- left
```

Sample Input:
8 //Number of zones
YRSPFMHI //Zone sequence of the match
YPSRFMHI //Zone sequence of MagneT's prediction

Sample Output:
Yasnaya Pochinki Farm Mylta Shelter Prison
Correctness of prediction: 75%

Problem 3 (LCS for 3 string) [10 Marks]:

You know about how to find the Longest Common Subsequence (LCS) between two strings using dynamic programming. Let's make it a bit challenging. I will give you three strings instead of two, and now you need to find out the Longest Common Subsequence (LCS) among these three strings. Sounds interesting, right? I know you can do it!

Input:
Each input will consist of three strings in each line. The length of each string will be no greater than 100.

Output:
Output the **length of the longest common subsequence** of the given three strings.

Sample Input 1:
hell
hello
bella

Sample Output 1:
3

Sample Input 2:
abbcdab
daccbadb
abccdaab

Sample Output 2:
4

Hint: For this problem you have to construct LCS for 3 parameters. You can use the following pseudo code to find out the longest common subsequence of three strings.

The followings are for your understanding:

```
LCS(X, Y, Z):
    m <- length[X] + 1
    n <- length[Y] + 1
    o <- length[Z] + 1
    c[m][n][o]
    t[m][n][o]
    for i <- 1 to m:
        for j <- 1 to n:
            for k <-1 to o:
                if i = 0 Or j = 0 Or k = 0:
                    c[i][j][k] <- 0
                    t[i][j][k] <- null/None
                else:
                    if x[i] = y[j] And x[i] = z[k]:
                        c[i][j][k] <- 1 + c[i-1][j-1][k-1]
                        t[i][j][k] <- diagonal
                    else:
                        if c[i-1][j][k] >= c[i][j-1][k]:
                            max <- c[i-1][j][k]
                            if max >= c[i][j][k-1]:
```

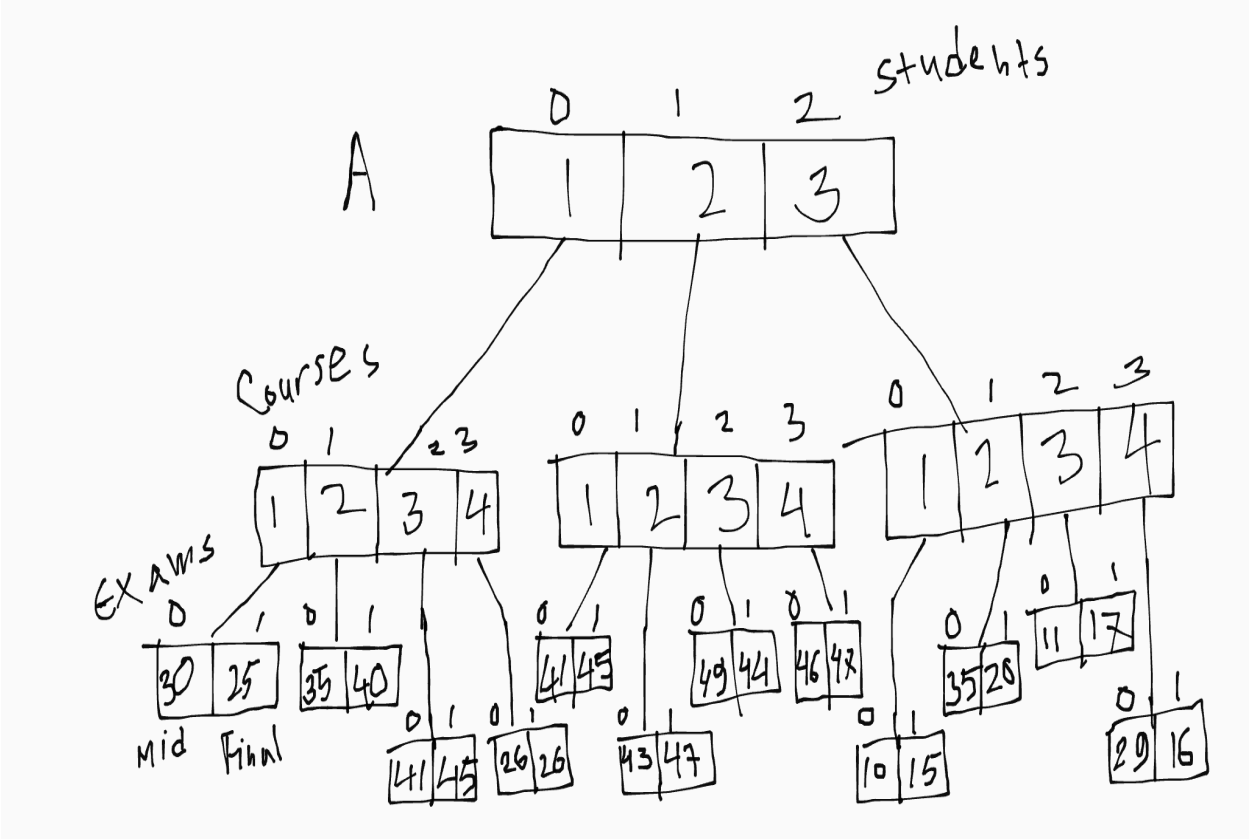
```
                    c[i][j][k] <- max
                    t[i][j][k] <- up-up-left

                else:
                    max <- c[i][j][k-1]
                    c[i][j][k] <- max
                    t[i][j][k] <- left-up-up
        else:
            max <- c[i][j-1][k]
            if max >= c[i][j][k-1]:
                c[i][j][k] <- max
                t[i][j][k] <- up-left-up
            else:
                max <- c[i][j][k-1]
                c[i][j][k] <- max
                t[i][j][k] <- left-up-up
```

3D Array:

We already know how to operate 2D arrays. Now we have to use 3D arrays in order to construct LCS with 3 parameters. Multidimensional array means multiple arrays of different dimensions or sizes can be held by a single array. We will learn about 3D arrays with an example.

Suppose we want to store midterm and final marks separately of four different courses of three students in a single array. We can easily do it by using a 3D array.



This is the pictorial view of the given scenario.Normally we can denote an array of 3 dimensions by this notation A[ ][ ][ ]. If we initialize this array empirically it will be like,
A[3][4][2]
Here, A is a 3D array where it has a size of 3 which represents 3 students. Each of 3 indices can hold an array of size 4 which indicates the different courses. And indices of these arrays can hold arrays of size 2 where each index represents the marks of mid and final.
For extracting the specific values from array A, we can simply use these instructions,

1. print(A[0][1][0])
2. print (A[2][2][1])
3. print(A[1][3][0])

First command will give 35 as output because first it will access the 0 th index of A. Then it will go through the index number 1 of it's connected array and finally index number 0 of the array which is connected with the index number 1 of 2nd array will be accessed and give its value as output which is 35. Same procedure will be applied for the rest of the instructions. The second and third commands will give outputs 17 and 46 respectively. For traversing this array you can use the given codes.

Python:

```
for i in range(len(A)):
    print("Student: ",(i+1))
    for j in range(len(A[i])):
        print("Course: ",(j+1))
        print("Marks of Mid and Final: ")
        for k in range(len(A[i][j])):
            print(A[i][j][k],end=" ")
        print()
    print()
```

Java:

```
for (int i = 0; i < A.length; i++) {
        System.out.println("Student: "+(i+1));
        for (int j = 0; j < A[i].length; j++) {
                System.out.println("Course: "+(j+1));
                System.out.println("Marks of Mid and Final: ");
                for (int k = 0; k < A[i][j].length; k++) {
                        System.out.print(A[i][j][k]+" ");
                }
                System.out.println();
        }
        System.out.println();
}
```

Output:

```
Student:  1
Course:  1
Marks of Mid and Final:
30 25
Course:  2
Marks of Mid and Final:
35 40
Course:  3
Marks of Mid and Final:
```

41 45
Course: 4
Marks of Mid and Final:
26 26

Student: 2
Course: 1
Marks of Mid and Final:
41 45
Course: 2
Marks of Mid and Final:
43 47
Course: 3
Marks of Mid and Final:
49 44
Course: 4
Marks of Mid and Final:
46 47

Student: 3
Course: 1
Marks of Mid and Final:
10 15
Course: 2
Marks of Mid and Final:
35 20
Course: 3
Marks of Mid and Final:
11 17
Course: 4
Marks of Mid and Final:
29 16