

Problem A is for those who have not done it last week.

If you did it already, go on to problem B.

## PROBLEM A: EDIT DISTANCE

The edit distance between two words—sometimes also called the *Levenshtein* distance—is the *minimum* number of letter insertions, letter deletions, and letter substitutions required to transform one word into another.

For example, the edit distance between FOOD and MONEY is at most four:

FOOD → MOOD → MON\_D → MONED → MONEY

Given two strings, find the edit distance between them.

INPUT:

Line 1: the first string, A

Line 2: the second string, B

OUTPUT:

Edit distance between the two strings

- 1) **We are transforming string A to string B.** Assume that string  $A[0] \dots A[i-1]$  have been transformed to be identical to  $B[0] \dots B[j-1]$ , and the consideration now is on  $A[i]$  and  $B[j]$ .

The table below lists all possible scenarios at state  $(i, j)$  and edit operations that can be performed. What is the consequential state for each combination of condition and operation ?

condition	edit operation	next state to consider
$A[i] == B[j]$	None	$(i+1, j+1)$
$A[i] != B[j]$	Insert $B[j]$ in front of $A[i]$	$(i, j+1)$
$A[i] != B[j]$	Delete $A[i]$	$(i+1, j)$
$A[i] != B[j]$	Change $A[i]$ to $B[j]$	$(i+1, j+1)$ but add one more

- 2) What is the beginning state?
- 3) If A runs out, but B has not yet, in other words,  $i == \text{len}(A)$ , but  $j < \text{len}(B)$ , what is the additional edit distance required to complete the transformation?
- 4) If B runs out, but A has not yet, what is the additional edit distance required to complete the transformation?
- 5) Use the concepts obtained from step 1 to 4 above in write a recursive brute-force solution for this problem. The zipped test case file is downloadable from Class Materials.
- 6) Given that a string can be up to 1000 letters long, improve the brute-force solution so that the program will finish in no more than 2.5 seconds (CPU processing time).

## PROBLEM B: Dynamic Programming for Minimum Coin Change

INPUT:

Line 1 : the list of coin denominator

Line 2 : the amount of change

OUTPUT: The minimum number of coins required for the change

EXAMPLE

INPUT	OUTPUT
1 3 4 5 7	2
1 2 5 10 13 3377	260

The following code is a memoized minimum coin change function.

```
mm = [-1]*(V+1)

def mincoin(v):
    global coin, mm

    if mm[v] == -1:
        if v == 0:
            mm[v] = 0
        else:
            minc = 10000000000
            for c in coin:
                if c <= v:
                    minc = min(minc, 1 + mincoin(v-c))
            mm[v] = minc
    return mm[v]
```

1. Given that  $v_1 \geq v_2$ ,
  - 1.1 which recursive call, to `mincoin(v1)` or to `mincoin(v2)`, is made first?
  - 1.2 which recursive function, `mincoin(v1)` or `mincoin(v2)`, returns first?
  - 1.3 which `mm`'s entry, `mm[v1]` or `mm[v2]`, obtains its final value first?

According, if items of `mm` are computed in a certain order, the function call "`mincoin(v-c)`" can always retrieve value from the pre-computed `mm` entry. Thus virtually eliminate chains of recursive calls.

2. Develop a *non-recursive* minimum coin change solution i.e. does not utilize recursive function, by iterating through `mm`'s indices with an appropriate sequence, computing value of corresponding `mm`'s entry along the way.