**Time: 1-2 hours; open book**

**Please answer the following questions, complete using vb.net/c# (Visual Studio) or other programming languages.**

**Q1:** Design a product catalogue with products (name, price, description), and n-level and multiple categories and manufacturer (name, logo). Draw normalized table structure with primary & foreign keys and write SQL to retrieve all n-level category products recursively, expected output:

Books – Philosophy – Metaphysics

Books – Philosophy – Confucianism - Mencius

Books – Literature – Lin Yutang

Software – Utilities – File Management

**A1:**

**Q2. Write an ASP.net web site (or in other programming languages) with a "contact us" web form to send email to hr@reasonables.com. Use web@reasonables.com as SENDER, get FROM from contact us form. Use local IIS SMTP as SMTP server. The following email header is expected:**

**Sender: <web@reasonables.com>**

Return-Path: <web@reasonables.com>

…....

Date: Sun, 22 Mar 2009 16:16:12 +0800

**From: John Smith<johnsmith@reasonables.com>**

Reply-To:<web@reasonables.com>

**To: <hr@reasonables.com>**

Subject: Realizing Value from IT Investment

**Q3. Write a Windows form application or iOS/Xamarin/Android that returns the difference of two given files (up to 1GB) using Levenshtein distance (edit distance).**

A commonly-used bottom-up [dynamic programming](http://en.wikipedia.org/wiki/Dynamic_programming) algorithm for computing the Levenshtein distance involves the use of an (n + 1) × (m + 1) matrix, where n and m are the lengths of the two strings. Here is [pseudocode](http://en.wikipedia.org/wiki/Pseudocode) for a function *LevenshteinDistance* that takes two strings, *s* of length *m*, and *t* of length *n*, and computes the Levenshtein distance between them:

**int** LevenshteinDistance(**char** s[1..m], **char** t[1..n])

*// d is a table with m+1 rows and n+1 columns*

**declare** **int** d[0..m, 0..n]

**for** i **from** 0 **to** m

d[i, 0] := i

**for** j **from** 1 **to** n

d[0, j] := j

**for** i **from** 1 **to** m

**for** j **from** 1 **to** n

**if** s[i] = t[j] **then** cost := 0

**else** cost := 1

d[i, j] := minimum(

d[i-1, j] + 1, *// deletion*

d[i, j-1] + 1, *// insertion*

d[i-1, j-1] + cost *// substitution*

)

**return** d[m, n]

Two examples of the resulting matrix (the minimum steps to be taken are highlighted):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  |  | **k** | **i** | **t** | **t** | **e** | **n** | |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | | **s** | 1 | 1 | 2 | 3 | 4 | 5 | 6 | | **i** | 2 | 2 | 1 | 2 | 3 | 4 | 5 | | **t** | 3 | 3 | 2 | 1 | 2 | 3 | 4 | | **t** | 4 | 4 | 3 | 2 | 1 | 2 | 3 | | **i** | 5 | 5 | 4 | 3 | 2 | 2 | 3 | | **n** | 6 | 6 | 5 | 4 | 3 | 3 | 2 | | **g** | 7 | 7 | 6 | 5 | 4 | 4 | 3 | | |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  |  | **S** | **a** | **t** | **u** | **r** | **d** | **a** | **y** | |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | **S** | 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | **u** | 2 | 1 | 1 | 2 | 2 | 3 | 4 | 5 | 6 | | **n** | 3 | 2 | 2 | 2 | 3 | 3 | 4 | 5 | 6 | | **d** | 4 | 3 | 3 | 3 | 3 | 4 | 3 | 4 | 5 | | **a** | 5 | 4 | 3 | 4 | 4 | 4 | 4 | 3 | 4 | | **y** | 6 | 5 | 4 | 4 | 5 | 5 | 5 | 4 | 3 | |

The [invariant](http://en.wikipedia.org/wiki/Invariant_%28mathematics%29) maintained throughout the algorithm is that we can transform the initial segment s[1..i] into t[1..j] using a minimum of d[i,j] operations. At the end, the bottom-right element of the array contains the answer.