INDIAN STATISTICAL INSTITUTE

Assignment 11

Data and File Structures Laboratory, M. Tech (CS) - I Year, 2014-2015 (Semester - I)

Total marks : 40 (Problem 1) + 40 (Problem 2) + 20 (Good programming habits) = 100

Uploading date: 24.11.2014 Clarification deadline: 01.12.2014

Submission deadline: 02.12.2014

Problem 1 (Edit distance): The following problem is known as the *edit distance* problem. Given two strings S_1 of n_1 characters and S_2 of n_2 characters, the problem is to modify S_1 and convert it into S_2 through a sequence of character edits. The character edits can be insertion, deletion and overwrite. Each character edit takes unit amount of time. Implement in Java, a dynamic programming based algorithm that minimizes the number of edits required. The characters of S_1 and S_2 belong to the same alphabet set. The alphabet set consists of the 26 upper case, 26 lower case English alphabets and numbers between 0 to 9. The two strings are to be taken as inputs.

Can you find appropriate overlapping subproblems?

The *edit distance* between two strings S_1 and S_2 is the minimum number of character insertions, character deletions and character substitutions required to transform S_1 to S_2. Let S_1 be of m characters and S_2 be of n characters. Let $C(i,j)$, $(i \leq m)$ and $(j \leq n)$, be the *edit distance* between the first i characters of S_1 and the first j characters of S_2. Thus, the final answer is surely $C(m,n)$. As to the overlapping sub-problem, note that $C(i,j)$ can be built from subproblems as was in the case of *Longest Common Subsequence (LCS)*.

We would hope for, again as in *LCS*, filling up a table of m rows and n columns. In writing the recursion, first, look at the base cases, i.e. the first row and the first column. $C(i,0) = i$ and $C(0,j) = j$. Now, we treat insertion, deletion and overwrite separately.

For insertion, we insert one character into the i^{th} position of S_1 thus generating i characters of S_1. So, for the cost, we have $C(i,j) = C(i-1,j) + 1$.

For deletion, we delete one character from the j^{th} position of S_2. So, for the cost, we have $C(i,j) = C(i,j-1) + 1$; as the j^{th} character in S_2 is deleted.

For substitution, if the i^{th} character of S_1 is the same as the j^{th} character of S_2, then there is no cost involved, i.e. $C(i,j) = C(i-1,j-1)$; else $C(i,j) = C(i-1,j-1) + 1$.

Obviously, $C(i,j)$ is the minimum of these three quantities. Thus, we have the following recurrence.

$$C(i,j) = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ \min & \begin{cases} C(i-1,j)+1 & \\ C(i,j-1)+1 & \\ C(i-1,j-1) & \text{if } S_1[i] = S_2[j] \\ C(i-1,j-1)+1 & \text{if } S_1[i] \neq S_2[j] \end{cases} \end{cases}$$

Filling up the $m \times n$ table using this recurrence takes $O(mn)$ time.

[Base case (10) + Solution (30) = 40]

Problem 2 (Integer 0-1 Knapsack): This is a well known problem called 0-1 integer knapsack; which is NP-complete. You will learn both the problem and NP-completeness in your algorithms course.

Let $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$ be a set of n items to be packed in a knapsack of size B; $B \in \mathbb{Z}^+$. Each item $u_i \in \mathcal{U}$ has a size s_i and a profit p_i; $s_i, p_i \in \mathbb{Z}^+$. The objective is to find a subset of $\mathcal{U}$, say $\mathcal{U}'$ whose total size is bounded by B and total profit is maximized. Formally, we need to find $\mathcal{U}'$, such that

$$\sum_{u_i \in \mathcal{U}'} p_i \text{ is maximized subject to the constraint } \sum_{u_i \in \mathcal{U}'} s_i \leq B$$

Try to formulate a dynamic programming based solution and implement an efficient code in Java. The input to the program will be the number of items n, their respective sizes and profits and the knapsack capacity.

Hints: Try to find a recursive formula for $V[i,j]$ that denotes the profit obtained by filling a knapsack of size j ($0 \leq j \leq B$) with items taken from the first i ($0 \leq i \leq n$) items $u_1, u_2, \ldots, u_i$ in an optimal way. What are the subproblems, in terms of which, you can find out $V[i,j]$? Are they: $V[i-1,j]$, $V[i-1,j-s_i]+p_i$?

Find the base cases properly. $V[n,B]$ should give you the optimal solution.

[Base case (10) + Solution (30) = 40]