

CSE 3101: Assignment 3

Worth 15%. Due 5pm July 20, 2006. Please put your assignment in the dropbox.

The work you submit must be your own. You may discuss problems with each others; however, you should prepare written solutions alone. Copying assignments is a serious academic offence, and will be dealt with accordingly.

Question 1 [From 15-3 in the text]

In order to transform one source string of text $X[1 \dots m]$ to a target string $Y[1 \dots n]$, we can perform various transformation operations. Our goal is, given X and Y , to produce a series of transformations that change X to Y . We use an array Z —assumed to be large enough to hold all the characters it will need—to hold the intermediate results. Initially, Z is empty, and at termination, we should have $Z[j] = Y[j]$ for $j = 1, 2, \dots, n$. We maintain current indices i into X and j into Z , and the operations are allowed to alter Z and these indices. Initially, $i = j = 1$. We are required to examine every character in X during the transformation, which means that at the end of the sequence of transformation operations, we must have $i = m + 1$.

There are six transformation operations:

Copy a character from X to Z by setting $Z[j] \leftarrow X[i]$ and then incrementing both i and j . This operation examines $Z[i]$.

Replace a character from X by another character c , by setting $Z[j] \leftarrow c$, and then incrementing both i and j . This operation examines $X[i]$.

Delete a character from X by incrementing i but leaving j alone. This operation examines $X[i]$.

Insert the character c into Z by setting $Z[j] \leftarrow c$ and then incrementing j , but leaving i alone. This operation examines no characters of X .

Twiddle (i.e., exchange) the next two characters by copying them from X to Z but in the opposite order; we do so by setting $Z[j] \leftarrow X[i + 1]$ and $Z[j + 1] \leftarrow X[i]$ and then setting $i \leftarrow i + 2$ and $j \leftarrow j + 2$. This operation examines $X[i]$ and $X[i + 1]$.

Kill the remainder of X by setting $i \leftarrow m + 1$. This operation examines all characters in X that have not yet been examined. If this operation is performed, it must be the final operation.

As an example, one way to transform the source string **algorithm** to the target string **altruistic** is to use the following sequence of operations, where the underlined characters are $X[i]$ and $Z[j]$ after the operation:

Operation	X	Z
<i>Initial strings</i>	<u>a</u> lgorithm	_
copy	<u>a</u> lgorithm	a_
copy	al <u>a</u> lgorithm	al_
replace by t	al <u>a</u> lgorithm	alt_
delete	algor <u>a</u> lgorithm	alt_
copy	algor <u>a</u> lgorithm	altr_
insert u	algor <u>a</u> lgorithm	altru_
insert i	algor <u>a</u> lgorithm	altrui_
insert s	algor <u>a</u> lgorithm	altruiss_
twiddle	algor <u>a</u> lgorithm	altruisti_
insert c	algor <u>a</u> lgorithm	altruistic_
kill	algor <u>a</u> lgorithm_	altruistic_

Note that there are several other sequences of transformation operations that transform **algorithm** to **altruistic**.

Each of the transformation operations has an associated cost. The cost of an operation depends on the specific application, but we assume that each operation's cost is a constant that is known to us. We also assume that the individual costs of the copy and replace operations are less than the combined costs of the delete and insert operations; otherwise, the copy and replace operations would not be used. The cost of a given sequence of transformation operations is the sum of the costs of the individual operations in the sequence. The cost of the sequence above is

$$3 \cdot \text{cost}(\text{copy}) + \text{cost}(\text{replace}) + \text{cost}(\text{delete}) + 4 \cdot \text{cost}(\text{insert}) + \text{cost}(\text{twiddle}) + \text{cost}(\text{kill})$$

Given two sequences $X[1 \dots m]$ and $Y[1 \dots n]$ and set of transformation-operation costs, the *edit distance* from X to Y is the cost of the least expensive operation sequence that transforms X to Y . Describe a dynamic-programming algorithm that finds the edit distance from $X[1 \dots m]$ to $Y[1 \dots n]$ and prints an optimal operation sequence. Analyze the running time and space requirement of your algorithm. [Clearly indicate the four steps, and give comments to any code you provide.]

Question 2

Let $G = (V, E)$ be a directed graph with nodes v_1, \dots, v_n . We say that G is an *ordered graph* if it has the following properties:

1. Each edge goes from a node with a lower index to a node with a higher index. That is, every directed edge has the form (v_i, v_j) with $i < j$.
2. Each node except v_n has at least one edge leaving it. That is, for every node v_i , $1 \leq i \leq n-1$, there is at least one edge of the form (v_i, v_j) .

The length of a path is the number of edges in it. The goal in this question is to solve the following problem (see Figure for an example):

Given an ordered graph G , find a longest path that begins at v_1 and ends at v_n .

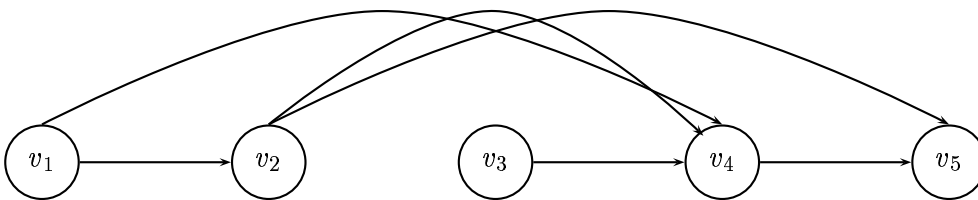


Figure 1: The longest path is v_1, v_2, v_4, v_5 . The output array is $[1, 1, 0, 1, 1]$

Give a dynamic-programming algorithm that solve the problem. Give the output in the form of an array B of length n , where $B[i] \in \{0, 1\}$ for $i = 1, \dots, n$, and $B[i] = 1$ if and only if v_i is in the computed path.

Question 3 [From question 22.2-6 in the text]

There are two types of professional wrestlers: “good guys” and “bad guys”. Between any pair of professional wrestlers, there may or may not be a rivalry. Suppose we have n professional wrestlers

and we have a list of r pairs of wrestlers for which there are rivalries. Give an $\mathcal{O}(n + r)$ -time algorithm that determines whether it is possible to designate some of the wrestlers as good guy and the remainder as bad guys such that each rivalry is between a good guy and a bad guy. If it is possible to perform such a designation, your algorithm should produce it.

Question 4

Suppose we are given an undirected graph $G = (V, E)$ and two vertices s, t of G . Give an algorithm that computes the number of shortest st paths in G . (The algorithm should not list all the paths; just the number suffices.) The running time of your algorithm should be $\mathcal{O}(m + n)$, where $n = |V|$ and $m = |E|$. (Hint: for any shortest st path, the distances from s to the nodes are $0, 1, \dots, \ell$ as we go along the path, where ℓ is the distance between s and t .)