# DYNAMIC PROGRAMMING 1
# EDIT DISTANCE

DR. GODFREY MUGANDA

## 1. Edit Distance

This is section 6.3 in our text.

The problem is to edit a string $X[1..m]$ and transform it into a second string $Y[1..n]$ with minimum cost, using only three edit operations Insert (I), Delete (D), and Edit (E).

The idea is to work backwards from the end of the two strings. You start out at position $i = m$ in the string $X[]$ and position $j$ in the string $Y[]$:

$$\begin{array}{cccccc} X[1] & X[2] & \cdots & X[m-1] & X[m] \\ & & & & \uparrow \\ & & & & i = m \end{array}$$

$$\begin{array}{cccccc} Y[1] & Y[2] & \cdots & Y[n-1] & Y[n] \\ & & & & \uparrow \\ & & & & j = n \end{array}$$

The idea is that we perform one of the three edit operations at positions $i$ in $X[]$ and $j$ in $Y[]$ to ensure that the last character in the edited $X[]$ matches the last character in $Y[]$. We then move backwards through the two strings until eventually, the edited $X[]$ becomes $Y[]$.

Following the book, let $E(i, j)$ be the minimum cost to transform the string $X[1..i]$ into $Y[1..j]$.

When working at positions $i$ and $j$ in $X[]$ and $Y[]$ respectively.

$$\begin{array}{cccccc} X[1] & X[2] & \cdots & X[i-1] & X[i] \\ Y[1] & Y[2] & \cdots & Y[j-1] & Y[j] \end{array} \tag{1}$$

1.1. **Insertion Operation.** We can choose to perform an Insert operation (I). We insert the character $Y[j]$ at the end of $X[1..i]$ to get

$$\begin{array}{ccccccc} X[1] & X[2] & \cdots & X[i-1] & X[i] & Y[j] \\ & Y[1] & Y[2] & \cdots & Y[j-1] & Y[j] \end{array} \tag{2}$$

Once we have performed this step, we are left with the subproblem $E(i, j-1)$ of transforming $X[1..i]$ into $Y[1..j-1]$. (Why?)

Because an insert has a cost of 1, the final cost will be

$$1 + E(i, j-1) \tag{3}$$

if we commit to insert $Y[j]$ at the end of $X[1..i]$.

1.2. **Deletion Operation.** Instead of inserting $Y[j]$ at the end of $X[1..i]$ in (1), we can delete $X[i]$. After deletion we get

$$\begin{array}{cccccc} X[1] & X[2] & \cdots & X[i-1] & \\ Y[1] & Y[2] & \cdots & Y[j-1] & Y[j] \end{array} \tag{4}$$

which we means we are left with the subproblem $E(i-1,j)$ of transforming $X[1..i-1]$ into $Y[1..j]$. (Why?)

Because a deletion has a cost of 1, the final cost will be

$$1 + E(i-1,j) \tag{5}$$

if we commit to delete $X[i]$ at the end of $X[1..i]$.

1.3. **Edit Operation.** Finally, we can just replace $X[i]$ with $Y[j]$ in (1) getting

$$\begin{array}{cccccc} X[1] & X[2] & \cdots & X[i-1] & Y[j] \\ Y[1] & Y[2] & \cdots & Y[j-1] & Y[j] \end{array} \tag{6}$$

and leaving us with the subproblem $E(i-1,j-1)$ of transforming $X[1..i-1]$ into $Y[1..j-1]$. (Why?)

The cost of an edit is 0 if $X[i] = Y[j]$, and is 1 otherwise. We define

$$\text{diff}(X[i], Y[j]) = \begin{cases} 0 & \text{if } X[i] = X[j] \\ 1 & \text{otherwise} \end{cases} \tag{7}$$

Therefore, if we commit to an edit operation (D) then the minimum final cost will be

$$\text{diff}(X[i], Y[j]) + E(i-1, j-1). \tag{8}$$

Now if we put together the final costs from (3), (5) and (8) we see that the minimum final cost must be given by the

$$E(i,j) = \min\{1 + E(i,j-1), 1 + E(i-1,j), \text{diff}(X[i], Y[j]) + E(i-1,j-1)\}. \tag{9}$$

1.4. **Boundary Conditions.** Notice that $E(i,0) = i$ and $E(0,j) = j$ (Why?).

The algorithm is now straightforward, and you can find it on page 163 of the class text.

Actually, to solve $E(m,n)$, you should have two solution tables, and array

$$E[1..m, 1..n]$$

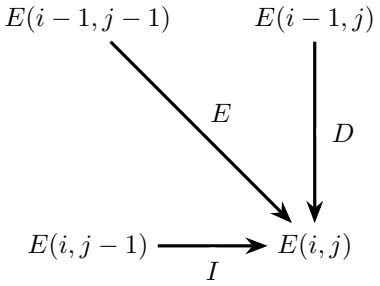
and an array

$$S[1..m, 1..n]$$

. The entries $S[i,j]$ record one of the characters 'I', 'D', 'E' according to whether the minimum in (9) is achieved by an insertion(I), deletion (D), or edit(E).

## 2. Order of Computation

Note that the dependence DAG for the subproblems looks like the figure below




This allows us to fill in the solution tables by first filling in rows 0 and column 0 according to the boundary conditions, and then filling in the rest of the tables by increasing rows, and within each row, left to right.

## 3. Example

We do an example similar to Figure 6.4 in the class text book, with the examples $X[]$ is SNOWY and $Y[]$ is SUNNY.

We combine the $E[i,j]$ and $S[i,j]$ tables, and moreover, we store a list of all edit operations that tie for the minimum in the recurrence relation (9).

| | | | S | U | N | N | Y |
|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 |
| | 0 | 0 | $1_I$ | $2_I$ | $3_I$ | $4_I$ | $5_I$ |
| S | 1 | $1_D$ | $0_E$ | $1_I$ | $2_I$ | $3_I$ | $4_I$ |
| N | 2 | $2_D$ | $1_D$ | $1_E$ | $1_E$ | $2_{EI}$ | $3_I$ |
| O | 3 | $3_D$ | $2_D$ | $2_{DE}$ | $2_{DE}$ | $2_E$ | $3_{EI}$ |
| W | 4 | $4_D$ | $3_D$ | $3_{DE}$ | $3_{DE}$ | $3_{DE}$ | $3_E$ |
| Y | 5 | $5_D$ | $4_D$ | $4_{DE}$ | $4_{DE}$ | $4_{DE}$ | $3_E$ |

Here is how the solution tables can be used. The edit operation appears on top of the arrow, and the cost of the operation appears below the arrow.

$$
\begin{aligned}
(SNOWY, SUNNY) &\xrightarrow[0]{E} (SNOW, SUNN) \\
&\xrightarrow[1]{D} (SNO, SUNN) \\
&\xrightarrow[1]{E} (SNN, SUNN) \to (SN, SUN) \\
&\xrightarrow[0]{E} (S, SU) \\
&\xrightarrow[1]{I} (SU, SU) \to (S, S) \\
&\xrightarrow[0]{E} (,)
\end{aligned}
$$

You should be able to take any two strings, including strings of unequal length, and fill in a table like this.