CS21003 Algorithms – I, Autumn 2012–13

Class test 2

Maximum marks: 20 | Time: 15-Nov-2012 | Duration: 1 hour

Roll no: ________________ **Name:** ______________________________________

[*Write your answers in the question paper itself. Be brief and precise. Answer all questions.*]

1. Let $S = a_0a_1a_2 \ldots a_{n-1}$ and $T = b_0b_1b_2 \ldots b_{m-1}$ be two strings of lengths n and m, respectively. The *Levenshtein distance* (or *edit distance*) $L(S,T)$ between S and T is the minimum number of elementary edit operations needed to convert S to T. Three types of elementary edit operations are permitted: insertion of a character (like $algorithm \Rightarrow alogorithm$), deletion of a character ($alogorithm \Rightarrow logorithm$), and replacing one character by another character ($logorithm \Rightarrow logarithm$).

For computing $L(S,T)$, build a two-dimensional table $L[i,j]$ for $-1 \leqslant i \leqslant n-1$ and $-1 \leqslant j \leqslant m-1$. The entry $L[i,j]$ stands for the Levenshtein distance between the prefixes $S[0\ldots i]$ and $T[0\ldots j]$. Write a $\Theta(nm)$-time algorithm to populate the entire table L in a suitable sequence. The entry $L[n-1,m-1]$ gives the desired distance $L(S,T)$. (**Hint:** Express $L[i,j]$ in terms of $L[i-1,j]$, $L[i,j-1]$ and $L[i-1,j-1]$.) **(10)**

Solution The boundary conditions are $L[i,-1] = i+1$ for all $i \geqslant -1$ (we need to make $i+1$ deletions in $S[0\ldots i]$), and $L[-1,j] = j+1$ for all $j \geqslant -1$ ($j+1$ insertions in $S[0\ldots -1] = \epsilon$). For $i,j \geqslant 0$, we have

$$L[i,j] = \min \begin{cases} L[i-1,j]+1, & \left[\text{Convert } a_0a_1\ldots a_{i-1}a_i \text{ to } b_0b_1\ldots b_ja_i\text{, and delete } a_i.\right] \\ L[i,j-1]+1, & \left[\text{Convert } a_0a_1\ldots a_i \text{ to } b_0b_1\ldots b_{j-1}\text{, and append } b_j.\right] \\ L[i-1,j-1]+t. & \left[\begin{array}{l}\text{If } a_i = b_j\text{, converting } a_0a_1\ldots a_{i-1}a_i \text{ to } b_0b_1\ldots b_{j-1}b_j \text{ is the same} \\ \text{as converting } a_0a_1\ldots a_{i-1} \text{ to } b_0b_1\ldots b_{j-1}\text{, so } t=0 \text{ in this case.} \\ \text{If } a_i \neq b_j\text{, then convert } a_0a_1\ldots a_{i-1}a_i \text{ to } b_0b_1\ldots b_{j-1}a_i\text{,} \\ \text{and replace } a_i \text{ by } b_j\text{, so } t=1 \text{ in this case.}\end{array}\right] \end{cases}$$

To start with, we populate the topmost row and the leftmost column of L using the boundary conditions. Subsequently, we populate the rest of the table in the row-major (or column-major) fashion. This ensures that when $L[i,j]$ is computed, the values $L[i-1,j]$, $L[i,j-1]$ and $L[i-1,j-1]$ are already available.

The pseudocode of an algorithm for computing $L(S,T)$ is given below.

1. Initialize $L[i,-1] = i+1$ for $i = -1,0,1,2,\ldots,n-1$.
2. Initialize $L[-1,j] = j+1$ for $j = 0,1,2,\ldots,m-1$.
3. For $i = 0,1,2,\ldots,n-1$, repeat: {
4.   For $j = 0,1,2,\ldots,m-1$, repeat: {
5.   If $(a_i = b_j)$, set $t = 0$, else set $t = 1$.
6.   Set $L[i,j] = \min\Big(L[i-1,j]+1,\ L[i,j-1]+1,\ L[i-1,j-1]+t\Big)$.
7.   } /* End of *for j* */
8. } /* End of *for i* */
9. Return $L[n-1,m-1]$.

2. Let S and T be strings as in Exercise 1. We are given a bound l on the number of errors. We want to compute all positions i in S, for which $S[i \ldots i+k]$ (for some $k \geqslant 0$) is at a Levenshtein distance $\leqslant l$ from T. This problem is known as *approximate string matching*, and has applications in spell checking, DNA sequence matching in computational biology, and identifying a multimedia file from a (possibly corrupted) snapshot.

Explain how you can modify the algorithm of Exercise 1 in order to find all the approximate matches (that is, matches with $\leqslant l$ errors) of T in S. The modified algorithm should run in $\Theta(nm)$ time. **(10)**

Solution The algorithm of Exercise 1 requires two modifications for solving the approximate string matching problem.

1. *Change in boundary conditions:* Since the approximate match of T can start from any location in S, the characters preceding any matching location do not count in the distance calculation, so we set the leftmost column as $L[i,-1] = 0$ (instead of $i+1$) for all i. The other boundary condition (the topmost row) remains the same.

2. *Remembering the edit sequences:* For $i, j \geqslant 0$, we need to remember which of the three arguments gives the minimum value during the computation of $L[i,j]$. We need to track back to the beginning of the match using these markers.

The modified algorithm is given below.

For $i = -1, 0, 1, 2, \ldots, n-1$, set $L[i,-1] = 0$.
For $j = 0, 1, 2, \ldots, m-1$, set $L[-1,j] = j+1$.
For $i = 0, 1, 2, \ldots, n-1$, repeat: $\{$
    For $j = 0, 1, 2, \ldots, m-1$, repeat: $\{$
        If $(a_i = b_j)$, set $t = 0$, else set $t = 1$.
        Let $u = L[i-1,j]+1$, $v = L[i,j-1]+1$ and $w = L[i-1,j-1]+t$.
        Set $L[i,j] = \min(u,v,w)$.
        If $(L[i,j] = u)$, set $E[i,j] = \uparrow$,
        else if $(L[i,j] = v)$, set $E[i,j] = \leftarrow$,
        else set $E[i,j] = \nwarrow$.
    $\}$ /* End of *for* j */
    If $(L[i,m-1] \leqslant l)$ $\{$
        Initialize $i' = i$ and $j' = m-1$.
        While $(L[i',j'] \neq 0)$, repeat: $\{$ /* Backtracking loop */
            If $(E[i',j'] = \uparrow)$, set $i' = i'-1$,
            else if $(E[i',j'] = \leftarrow)$, set $j' = j'-1$,
            else set $i' = i'-1$ and $j' = j'-1$.
        $\}$ /* End of *while* */
        Report the approximate match location $i' - j'$.
    $\}$ /* End of *if* */
$\}$ /* End of *for* i */

In this algorithm, the populating of L and E takes a total of $\Theta(nm)$ time. Each iteration in the backtracking loop for each approximate match reduces i' and/or j'. If only i' is reduced, then the value of $L[i',j']$ also reduces by 1. Therefore, the total number of iterations of each backtracking loop is $\max(m,l)$. We usually have $l \leqslant m-1$ (otherwise, every position in S is an approximate match position), so each backtracking loop runs in $O(m)$ time, and there are at most n executions of the backtracking loop.