A7:Solutions

Wednesday, May 5, 2021 10:51 AM

Edit Distance

A is of length n & B of length m. We want to transform A to B by adding/deleting/transforming (changing) characters. The min. no of ops needed is the edit distance. We assume that A sits in an array. When we delete the character at position i we replace A[i] with a deleted symbol. When we replace the character at posn. i we replace A[i] with the new character. When we insert a character at posn i we shift all characters to the right of (and including) i by 1 cell & set A[i] to the new character.

If is no loss of generality to assume that the operations done on A to transform it into B are done in a left to right manner. Further, any optimum sequence of operations would not be performing 2 ops at the same position. This can be checked by considering all 6 cases where both ops are performed at the same posn i

1) insert followed by delete is equivalent to retaining the original string
2) Insert followed by a replace could be combined into an insert
3) delete followed by an insert is equivalent to an insert at i & a delete at i+1
4) delete followed by replace could be combined into a replace
5) replace followed by an insert is equivalent to an insert at i and a replace at i+1
6) replace followed by a delete can be combined into a delete.

Thus a sequence of ops at posn i can be combined into a single op at posn i & perhaps an op at posn (i+1)

We define the subproblems as follows.

Let X(i,j) denote the edit distance between A[1..i] & B[1..j].

If A[i] = B[j] then X(i,j) is the same as the edit distance between A[1..i-1] & B[1..j-1] which is X(i-1, j-1)

If A[i] ≠ B[j] then consider the last operation in the optimal transformation from A[1..i] to B[1..j]. If this is an

If A[1..i] to B[1..j]. If this is an transformation from A[1..i] to B[1..j]. If this is an

1) insert operation then the previous operations transformed A[1..i] to B[1..j-1] & hence $X(i,j) = X(i,j-1) + 1$

2) delete operation then the previous ops transformed A[1..i-1] to B[1..j] & hence $X(i,j) = X(i-1,j) + 1$

3) replace operation then the previous ops transformed A[1..i-1] to B[1..j-1] & hence $X(i,j) = X(i-1,j-1) + 1$.

Thus if $A[i] \neq B[i]$ then

$$X(i,j) = \min \begin{cases} X(i-1,j) + 1 \\ X(i,j-1) + 1 \\ X(i-1,j-1) + 1 \end{cases}$$

Let X be a $n \times m$ matrix whose $(i,j)^{th}$ entry corresponds to $X(i,j)$. Since in computing $X(i,j)$ we only need values for $X(i-1,j), X(i,j-1), X(i-1,j-1)$ the matrix can be filled by moving row by row and filling each row left to right. computing each entry takes constant time and so it takes $O(mn)$ time to fill the table. The edit distance between A & B is $X(n,m)$ & the sequence of edits needed to transform A to B can be computed by tracing how $X(n,m)$ obtained its value.

Boolean Parenthesizations

We are given a sequence of n TRUE/FALSE symbols in an array B. Between two consecutive symbols B[i], B[i+1] we are given a boolean operation op[i] which is one of AND/OR/XOR. We need to determine the no. of different parenthesizations of this boolean expression which evaluate to true.

Let $T(i,j), j \geq i$, be the no. of parenthesizations of the boolean [illegible] evaluate to TRUE &

Let $T(i,j)$, $j \geq i$, be the no. of parenthesizations of the boolean expression defined over $B[i..j]$ which evaluate to TRUE & $F(i,j)$ the no. of parenthesizations which evaluate FALSE. Each parenthesization of the boolean expression defined over $B[i..j]$ can be viewed as a parenthesization of $B[i..k]$ & of $B[k+1, j]$ where $i \leq k \leq j-1$. Whether this parenthesization evaluates to TRUE / FALSE depends on the value of the expressions $B[i..k]$, $B[k+1, j]$ & the operation $op[k]$. Thus

$$T(i,j) = \sum_{k=i}^{j-1} T(i,k) * T(k+1,j) +$$

$op[k] = AND$

$$\sum_{k=i}^{j-1} \left[T(i,k)+F(i,k)\right] * \left[T(k+1,j)+F(k+1,j)\right] + - F(i,k) * F(k+1,j)$$

$op[k] = OR$

$$\sum_{k=i}^{j-1} T(i,k) * F(k+1,j) + F(i,k) * T(k+1,j)$$

$op[k] = XOR$

Similarly

$$F(i,j) = \sum_{k=i}^{j-1} F(i,k) * F(k+1,j) +$$

$op[k] = OR$

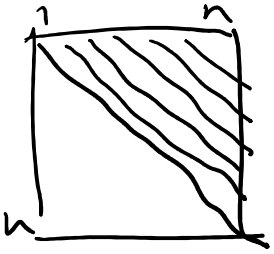
$$\sum_{k=i}^{j-1} \left[T(i,k)+F(i,k)\right] * \left[T(k+1,j)+F(k+1,j)\right] + - T(i,k) * T(k+1,j)$$

$op[k] = AND$

$$\sum_{k=i}^{j-1} T(i,k) * T(k+1,j) + F(i,k) * F(k+1,j)$$

Op[k] = XOR

To do this computation we have 2 nxn matrices T, F where T[i,j] (resp. F[i,j]) has the value T(i,j) (resp. F(i,j)). The computation is similar to that employed for matrix-chain multiplication. Only the upper triangular parts of T, F are filled (since $i \le j$). The entries can be filled diagonal by diagonal starting from the principal diagonal (of length n)



Computing entry T[i,j] takes time O(j-i) & hence total time to compute all entries of T, F is $O(n^3)$.

The value of T(1,n) is the no. of parenthesizations that evaluate to TRUE.

Value of Game

Let $V(i,j) = \sum_{k=i}^{j} v_k$ be the total value of coins from i to j ($j \ge i$)

& let F(i,j) be the value that the first player can win on the subarray A[i..j] (A[1..n] is the array containing the n coins)

To compute F(i,j) note that if the player moving first removes coin i then we are left with the subarray A[i+1,j] & the other player (which will be the first player on this subarray) will receive a value F(i+1,j). Thus the first player would receive value V(i+1,j) − F(i+1,j) & hence

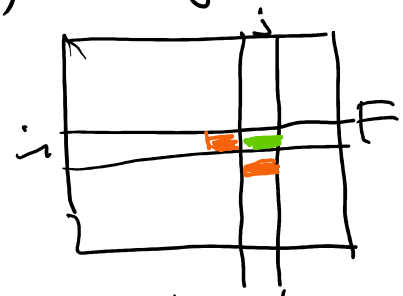
$$F(i,j) = v_i + V(i+1,j) - F(i+1,j).$$ Similarly if the first player removed v_j then

$$F(i,j) = v_j + V(i,j-1) - F(i,j-1)$$ & hence

$$F(i,j) = \max\{ v_i + V(i+1,j) - F(i+1,j),\ v_j + V(i,j-1) - F(i,j-1) \}$$

$\theta_j + V(i,j-1) - F(i,j-1)$]

Let F be a nxn array such that $F[i,j] = F(i,j)$. To compute $F(i,j)$ we need values $F(i+1,j)$ & $F(i,j-1)$. Since $F(i,j)$ is only defined for $i \le j$ we only need to compute the entries in the upper triangular half of F. These entries can be computed diagonal by diagonal starting from the principal diagonal $F[i,i] = \theta_i$. Since each entry can be computed in constant time the total time required to compute all entries is $O(n^2)$.



$F[1,n] = F(1,n)$ is the value of interest & gives the maximum value that the first player can obtain. The strategy to be followed by the first player to obtain this value can be determined by tracing how $F[1,n]$ is obtained.

Two person traversals

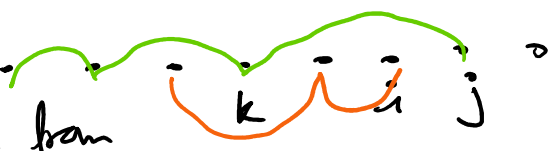
Let $d(i,j)$ be the distance between the i^{th} city & the j^{th} city. Let $X(i,j)$, $0 \le i < j$, denote the best way for 2 persons to traverse cities 1 to j such that one person ends at city i (& the other at city j). If $i = 0$ then one person has not started yet.

Note that if $i < j-1$ then the person who visits city j also visits $j-1$ & hence

$$X(i,j) = X(i,j-1) + d(j-1,j) \quad \text{if } i < j-1$$

We next consider the case when $i = j-1$. Let k be the penultimate city visited by the person who ends up at city j. Then

$$X(i,j) = X(k,i) + d(k,j)$$

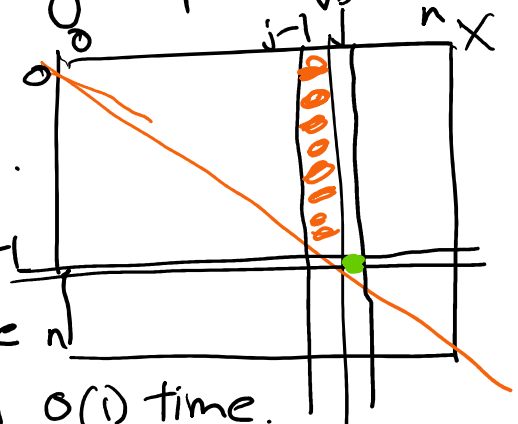


& since k could be any of the cities from

& since k could be any of the cities from 0 to j-2 we get

$$X(i,j) = \min_{0 \le k \le j-2} X(k,i) + d(k,j) \quad \text{for } i = j-1$$

Let X be a (n+1)x(n+1) matrix such that $X[i,j] = X(i,j)$. Since $i < j$, we will only be filling the upper triangular part of X. To compute the column j we only need entries of column j-1 & hence X can be computed column by column from left to right



Computing $X(j-1,j)$ takes $O(j)$ time while computing $X(i,j)$ $i \le j-2$ takes $O(i)$ time. Hence time to compute all entries of column j is $O(j)$ & hence total time to compute all entries of matrix X is $O(n^2)$.

Let $X[i,n]$ be the smallest entry in the n^{th} column; this gives us the shortest way for the 2-persons to traverse n cities. To compute the actual traversals we need to trace how $X[i,n]$ obtained its value. This can be accomplished in $O(n)$ time since in each step of our tracing procedure we would decrease the column number by 1.