CSC721 Dynamic Programming Examples Fall 2017

Reference URL: `https://people.cs.clemson.edu/~bcdean/dp_practice`

Maximum Value Contiguous Subsequence

Input: Array $A = [a_1, a_2, a_3, ..., a_n]$ of real numbers.

Output: $J(i,j) = \max_{i,j} \sum_{\ell=i}^{j} a_\ell$

Note: The problem is only interesting if the array A contains negative numbers.

As in all dynamic programming solutions, we define a memoization table. Let:

$$M_j = \max_i \sum_{\ell=i}^{j} a_\ell$$

Table M_j is the maximum subsequence sum over all subsequences ending at position j.

To complete the dynamic programming strategy, we need to define an update rule for the table M_j. Our rule is:

$$M_j = \max(M_{j-1} + a_j , a_j) \tag{1}$$

To make sense of the update rule in equation (1), we consider the following assertion:

> **The maximal subsequence ending at position j either extends a subsequence ending at position $j-1$ or consists of a subsequence length one starting and ending at position j.**

The logic of the assertion above is complete. The maximal subsequence either extends a previous subsequence or it doesn't.

- If a maximal subsequence ending at position j extends a subsequence ending at position $j-1$ then the best subsequence to extend is the optimal one, with value M_{j-1}. The sum of the extended sequence is obviously $M_{j-1} + a_j$.
- If a maximal subsequence ending at position j is length one, then the sum value is a_j.

Algorithm:

```
M[1] = A[1] ;       // Initialize table.
for  j = 2 to n do
    M[j] = max( M[j-1] + A[j] ,  A[j] ) ;  // Update table.
end for

r = M[1] ;          // Candidate for the largest.
for  k = 2 to n do
    if ( r < M[k] ) r = M[k] ;  //  Update candidate.
end for

return r ;
```

The asymptotic complexity of this algorithm is $\mathcal{O}(n)$.

Edit Distance

Input: Strings $A = [a_1, a_2, a_3, ..., a_n]$ and $B = [b_1, b_2, b_3, ..., a_m]$.
Positive real costs C_i, C_d, and C_r.

Given a position p in a string, the costs C_i, C_d, and C_r are the (fixed constant) costs of inserting, deleting, and replacing a character at position p respectively.

Output: The minimal cost of transforming string A to string B.

As before, we define a memoization table. Let:

$$T_{i,j} \;=\; \text{minimum cost to transform substring } a_1, a_2, ..., a_i \text{ to substring } b_1, b_2, ..., b_j$$

Also as before, we need to define an update rule for the table $T_{i,j}$. Our update rule is:

$$T_{i,j} \;=\; \min \begin{cases} C_d + T_{i-1,j} \\ T_{i,j-1} + C_i \\ \begin{cases} T_{i-1,j-1} & \text{if } a_i = b_j \\ T_{i-1,j-1} + C_r & \text{if } a_i \neq b_j \end{cases} \end{cases} \tag{2}$$

The update rule in equation (2) is based on the idea that the transformation of string A to string B consists of a sequence of insert, delete, and replace operations. We consider each of the sub-expressions in equation (2).

$\mathbf{C_d + T_{i-1,j}}$ quantifies a delete operation. To optimally transform the string $a_1, a_2, ..., a_i$ to string $b_1, b_2, ..., b_j$, we delete symbol a_i and optimally convert string $a_1, a_2, ..., a_{i-1}$ to string $b_1, b_2, ..., b_j$. The cost of this operation is the cost C_d of deleting symbol a_i, plus the optimal cost $T_{i-1,j}$ of transforming string $a_1, a_2, ..., a_{i-1}$ to string $b_1, b_2, ..., b_j$.

$\mathbf{T_{i,j-1} + C_i}$ quantifies an insert operation. To optimally transform the string $a_1, a_2, ..., a_i$ to string $b_1, b_2, ..., b_j$, we first convert string $a_1, a_2, ..., a_i$ to string $b_1, b_2, ..., b_{j-1}$, and insert symbol b_j. The cost of this operation is the conversion cost $T_{i,j-1}$ plus the cost C_i to insert one symbol.

The replace operation has two sub-cases;

- If $a_i = b_j$, then the cost of optimally transform string $a_1, a_2, ..., a_i$ to string $b_1, b_2, ..., b_j$ is equal to the cost of optimally transforming string $a_1, a_2, ..., a_{i-1}$ to string $b_1, b_2, ..., b_{j-1}$; there is nothing to do for symbols a_i and b_j.
- If $a_i \neq b_j$, then the cost of optimally transform string $a_1, a_2, ..., a_i$ to string $b_1, b_2, ..., b_j$ is equal to the cost of optimally transforming string $a_1, a_2, ..., a_{i-1}$ to string $b_1, b_2, ..., b_{j-1}$, plus the cost C_r to replace symbol a_i by b_j.

Exercise: (Hat Problem)

1. Use equation (2) to construct an algorithm to solve the edit distance problem.
2. Give an asymptotic upper bound on the time complexity of your algorithm.
3. What quantity should be output to solve the original problem?