Lindsey Bieda and Joe Frambach
Dynamic Programming Problems
10.05.2011

20. The input to this problem is two sequences $T = t_1, \ldots, t_n$ and $P = p_1, \ldots, p_k$ such that $k = n$, and a positive integer cost c_i associated with each t_i. The problem is to find a subsequence of T that matches P with maximum aggregate cost. That is, find the sequence $i_1 < \ldots < i_k$ such that for all j, $1 = j = k$, we have $t_{i_j} = p_j$ and $\sum_{j=1}^{k} c_{i_j}$ is maximized.
So for example, if $n = 5$, $T = XY\ XXY$, $k = 2$, $P = XY$, $c_1 = c_2 = 2$, $c_3 = 7$, $c_4 = 1$ and $c_5 = 1$, then the optimal solution is to pick the second X in T and the second Y in T for a cost of $7 + 1 = 8$.

(a) Give a recursive algorithm to solve this problem. Then explain how to turn this recursive algorithm into a dynamic program.

A function, Weighted Sub Sequence (WSS), is defined:
The recursive algorithm is called initally as wss(n,k) with T, P, and C being globally accessable.
The algorithm works by examining substrings of both of the given sequences and then determining where the values at a given position are equal and maximizes the values at these positions.
wss(i,j):
    if i = 0 or j = 0: *outside the bounds of either string*
        return 0 *no value here*
    if i ¡ j: *if the length of P is less than the length of T there is no solution*
        return $-\infty$
    else if $T_i = P_j$: *The last characters are equal. Either use it or ignore and continue.*
        return max(v_i + wss(i-1,j-1), wss(i-1,j)) *check if there is a better location elsewhere in the string*
    else:
        return wss(i-1,j) *check the rest of the string*

Given the above recursive definition we can draw a call tree and then determine what pruning rules to apply. From there we map the tree to an array based on these pruning rules.
The pruning rules are based on i and j passed into the WSS call, so the complexity is polynomial in terms of n and k.

(b) Give a dynamic programming algorithm based on enumerating subsequences of T and using the pruning method.

Tree Defintion:
Every node in the tree is a different substring of T. Each of the levels of the tree are created by either choosing to add the next character of the T or not. We represent these states as [level,number of matching characters] in the array, with the total cost being stored at that location. The tree is rooted at [0,0] representing the empty string with a total cost of 0.
The children of $A[x, y]$ will be $A[x + 1, y]$ and $A[x + 1, y + 1]$ (if $T_{x+1} == P_{y+1}$).

Pruning Rules (Ruling Prunes):

(1) For every node on the same level with the same number of matching characters, prune all but the one with the maximum cost.
(2) At every level, ℓ, if the next added character to the substring does not increase the amount of matching characters, prune that node.

Algorithm:
wss:

A[0 to n, 0 to k] = 0. *Initialize costs to zero.*
A[0,0] = 0. *T's empty substring matches zero characters of P with a cost of zero*
for $\ell = 0$ to $n-1$: *For each character in T*
    for $m = 0$ to $k-1$: *For each amount of matching, to $k-1$, don't go past the end of P*
        $A[\ell+1, m] = \max(A[\ell+1, m],\ A[\ell, m])$
        if $T_{\ell+1} == P_{m+1}$ then
            $A[\ell+1, m+1] = \max(A[\ell+1, m+1],\ c_{\ell+1} + A[\ell, m]))$
The solution is given at $A[n, k]$.

(c) Give a dynamic programming algorithm based on enumerating subsequences of P and using the pruning method.

Tree Defintion:
Every node in the tree is a different substring of T. The nodes are represented as [level, last matching character index] in the array, with the total cost being stored at that location. At each level a different substring of P is being considered. With the level, ℓ, representing the first ℓ characters of P. The tree is rooted at [0,0] representing the empty string.

Pruning rules:

(1) At every level remove the generated substrings of T that do not match the current considered substring of P as defined above.
(2) At every level for nodes with the same last matching character index keep only the one with the maximum cost.

Algorithm:
wss:
A[0 to k, 0 to n] = 0. *Initalize costs to zero.*
A[0,0] = 0. *The empty string does not match anyting and has a cost of zero.*
for $\ell = 0$ to $k-1$: *For every level*
    for $i = 0$ to n: *For every character in T*
        if $P_{\ell+1} == T_i$:
            A[$\ell+1$, i] = max(A[$\ell+1$, i], c_i + A[ℓ, i])

The solution is given at max(A[k, *]).