Practice Assignment 2

Due: Tuesday, Jan 17th at 2PM to canvas

To get credit, each student must submit their own solutions (which need not be typeset) to TEACH by the due date above – no late submissions are allowed. Note that while you must each submit solutions individually, you are free to work through the problem sets in groups. Solutions will be posted shortly after class and may be discussed in class. These will be not be graded on the basis of *completion* alone, not *correctness*.

1. Give an $O(nK)$ dynamic programming algorithm for the following task:

   Input: A list of n positive integers $a_1, a_2, \ldots, a_n$ and a positive integer K.
   Question: Does some subset of the a_i's add up to K? (You can use each a_i at most once.)

   You should:

   (a) Define the dynamic programming table.
   (b) Give a recursive formula for an entry in the dynamic programming table.
   (c) Describe in words how to fill the dynamic programming table.
   (d) Give pseudocode for the final algorithm.
   (e) Give the running time of your algorithm.

   (a) Let $T(i,k) = 1$ if there is a subset of the integers $a_1, a_2, \ldots, a_i$ that adds to k and 0 otherwise.

   (b) $T(i,0) = 1$ for $i = 0, 1, ..., n$
   $T(0,k) = 0$ for $k = 1, 2, ..., K$
   $T(i,k) = \max\{T(i-1,k), T(i-1,k-a_i)$ (if $a_i \leq k$)$\}$

   (c) Column-wise or row-wise.

   (d) initialize T as an $n+1 \times K+1$ array
   for $i = 0 \ldots n$
   $T(i,0) = 1$
   for $k = 1 \ldots K$
   $T(0,k) = 0$
   for $i = 1 \ldots n$
   for $k = 1 \ldots K$
   $T(i,k) = T(i-1,k)$
   if $a_i \leq k$
   if $T(i-1, k-a_i) = 1$, $T(i,k) = 1$
   If $T(n,K) = 0$, then there is no subset of the a_i's that add up to K. If $T(n,K) = 1$, then there is a subset of the a_i's that add up to K.

   (e) The running time is $O(nK)$.

2. Yuckdonald's is considering opening a series of restaurants along Quaint Valley Highway (QVH). The n possible locations are along a straight line, and the distances of these locations from the start of QVH are, in miles and in increasing order, $m_1, m_2, \ldots, m_n$. The constraints are as follows:

   - At each location, Yuckdonald's may open at most one restaurant. The expected profit from opening a restaurant at location i is p_i , where $p_i > 0$ and $i = 1, 2, \ldots, n$.
   - Any two restaurants should be at least k miles apart, where k is a positive integer.

   Give an efficient algorithm to compute the maximum expected total profit subject to the given constraints. You should:

(a) Define the dynamic programming table.

(b) Give a recursive formula for an entry in the dynamic programming table.

(c) Describe in words how to fill the dynamic programming table.

(d) Give pseudocode for the final algorithm.

(e) Give the running time of your algorithm.

(a) Let $T(i)$ be the maximum profit of the subproblem for $m_1, m_2, \ldots, m_i$ that includes location m_i

(b) $T(1) = p_1$
$T(i) = p_i + max_{j<i}\{T(j) : m_i - m_j \geq k\}$

(c) Fill in the table row-wise

(d) Input n possible locations with location i having distance from the starting point given by m_i initialize T as an n by 1 array.
$T(1) = p_1$
for $i = 2, \ldots, n$
$T(i) = max\{p_i, max_{j<i}\{T(j) + p_i : m_i - m_j \geq k\}\}$
return $\max T(i)$

(e) $O(n^2)$

Alternatively, we can also define $T(i)$ to be the maximum profit with $m_1, ..., m_i$ (without restricting the solution to must include m_i). In this case the recursion will be $T(i) = maxT(i-1), p_i + T(i^*)$ where i^* is the first location preceding i that is at least k apart. The final output in this case will be $T(n)$. If we use binary search to find i^*, this will lead to $O(n \log n)$ run time.

3. A subsequence is palindromic if it is the same whether read left to right or right to left. For instance, the sequence

$$A, C, G, T, G, T, C, A, A, A, A, T, C, G$$

has many palindromic subsequences, including A, C, G, C, A and A, A, A, A (on the other hand, the subsequence A, C, T is not palindromic). Devise an algorithm that takes a sequence $x[1 \ldots n]$ and returns the (length of the) longest palindromic subsequence. Its running time should be $O(n^2)$. You should:

(a) Define the dynamic programming table.

(b) Give a recursive formula for an entry in the dynamic programming table.

(c) Describe in words how to fill the dynamic programming table.

(d) Give pseudocode for the final algorithm.

(e) Give the running time of your algorithm.

(a) Let $T(i, j)$ be the length of the longest palindromic subsequence of the sequence $s_i, s_{i+1}, \ldots, s_j$.

(b) $T(i, j) = 0$ *if* $i > j$
$T(i, j) = 1$ *if* $i = j$
$T(i, j) = \max\{T(i+1, j), T(i, j-1), 2 + T(i+1, j-1)\}$ *if* $i < j$ *and* $s_i = s_j$
$T(i, j) = \max\{T(i+1, j), T(i, j-1)\}$ *if* $i < j$ *and* $s_i \neq s_j$
Note that we could replace the second last case with the following case: if $i < j$ and $s_i = s_j$, then $T(i, j) = 2 + T(i+1, j-1)$. To see why see the note at the end.

(c) Fill in the table along the diagonals beginning with the main diagonal and ending with the top right entry of the table.

(d) *Initialize* T *as a* $n \times n$ *array*
for $i = 1, \ldots, n$
for $j = 1, \ldots, i-1$
$T(i,j) = 0$
$T(i,i) = 1$
for $k = 2, \ldots, n$
for $i = 1, \ldots, n-k+1$
$j = i + k - 1$
if $s_i = s_j$
$T(i,j) = 2 + T(i+1, j-1)$
else
$T(i,j) = \max\{T(i+1,j), T(i,j-1)\}$
return $T(1,n)$

(e) $O(n^2)$

Here is a proof that if $s_i = s_j$ *then* $T(i,j) = 2 + T(i+1, j-1)$*:*

Let $s_1, s_2, \ldots, s_n$ *be a sequence and let* i *and* j *be indices where* $i \leq j$*. Then we will say that a matching of* $s_i, s_{i+1}, \ldots, s_j$ *is a matching of the indices between* i *and* j *such that for each match* (p,q)*, we have* $p \leq q$*, the characters at index* p *and* q *are equal, i.e.,*$s_p = s_q$*,* p *and* q *are not matched with any other indices, for each match* $(p,q) \neq (p',q')$*, we do not have* $p < p' < q < q'$*, and if* $p = q$*, then we have* $p' < p = q < q'$*. We will say that the size of a matching is the number of unique indices in the matching. So, the length of a longest palindrome subsequence of* $s_i, s_{i+1}, \ldots, s_j$ *is equal to the size of a largest matching of* $s_i, s_{i+1}, \ldots, s_j$*.*

Let I *be a largest matching of* $s_i, s_{i+1}, \ldots, s_j$ *and let* $T(i,j)$ *denote the size of* I*. We will show* $T(i,j) = 1$ *if* $i = j$*,* $T(i,j) = 2 + T(i+1, j-1)$ *if* $i < j$ *and the characters at index* i *and* j *are equal, and* $T(i,j) = \max\{T(i+1,j), T(i,j-1)\}$ *if* $i < j$ *and the characters at index* i *and* j *are not equal, where we define* $T(i,j) = 0$ *whenever* $i > j$*. If* $i = j$*, then it is clear that* $T(i,j) = 1$ *and we're done. So, assume that* $i < j$*. We will consider two cases: the case where the characters at index* i *and* j *are equal and the case where the characters at index* i *and* j *are not equal.*

Case 1: Assume the characters at index i *and* j *are equal. If* (i,k) *is not matched in* I*, for any* k *with* $i < k \leq j$*, then the matching* I' *obtained from* I *by adding* (i,j) *is a larger matching of* $s_i, s_{i+1}, \ldots, s_j$ *than* I*, which is impossible. So, it must be the case that* (i,k) *is a match in* I*, for some* k *with* $i < k \leq j$*. If* (i,k) *is a match in* I*, for some* k *with* $i < k < j$*, then the matching* I' *obtained from* I *by removing* (i,k) *and adding* (i,j) *is a matching of* $s_i, s_{i+1}, \ldots, s_j$ *that is the same size as* I*. So, we may assume* (i,j) *is a match in* I*. This means* $T(i,j) = 2 + T(i+1, j-1)$*.*

Case 2: Assume the characters at index i *and* j *are not equal. Since characters at index* i *and* j *are not equal, it follows that* (i,j) *is not a match in* I*. This means either* i *is not matched in* I *or* j *is not matched in* I*. Without loss of generality, assume* i *is not matched in* I*. This means* $T(i,j) = T(i+1,j) \geq T(i,j-1)$*, which means* $T(i,j) = \max\{T(i+1,j), T(i,j-1)\}$*.*