Comp 590-087/790-087: BioAlgorithms -- Fall 2011

Problem Set #2

Issued: 9/18/2011 Due: In class 10/6/2010

Homework Information: Some of the problems are probably too long to attempt the night before the due date, so plan accordingly. No late homework will accepted. However, your lowest homework will be dropped. Feel free to work with others, but the work you hand in should be your own.
Note: 590-087 students are not required to answer the starred (*) parts of questions.

Question 1. Write pseudocode for the TotalDistance(word, DNA) function used in various the MedianSearch algorithms given in section 4.9 and characterize its performance in terms of t, n, and l. In the BranchAndBoundMedianSearch algorithm the cost of computing TotalDistance seems unwarranted for small prefixes. For example, it is highly likely that all 2-mers appear in most strings over some length. This results in an *optimisticDistance* of 0, and provides no pruning. Suggest a heuristic for directly calling NextVertex() without testing the TotalDistance based on both the length of the prefix and, n, the sequence length, and write pseudocode that implements this algorithm variant.

The following Python function computes TotalDistance:

```
def TotalDistance(word, DNA):
    minTotalDist = 0
    for seq in DNA:
        minDist = len(word)
        for j in xrange(len(seq)-len(word)+1):
            dist = 0
            for k in xrange(len(word)):
                if (word[k] != seq[j+k]):
                    dist += 1
            if (dist < minDist):
                minDist = dist
            if (dist == 0):
                break
        minTotalDist += minDist
    return minTotalDist
```

It includes an early exit when a perfect match is found. Nonetheless, its worse-case performance is $O(t(n - l + 1) l) = O(tnl)$. Under the assumption of uniformly and independently distributed bases, a prefix of length k appears in a string of length N with a likelihood of $N/4^k$. We can therefore assume that a given prefix occurs whenever this likelihood is > 1.0. This is equivalent to the heuristic $4^k < N$. This impacts the code as follows:

```
nucleotide = ['a','c','g','t']

def nucleotideString(s):
    nString = ''
    for c in s:
        if (c == 0):
           break
        nString += nucleotide[c-1]
           return nString
```

```
def BranchAndBoundMedianSearch(DNA,t,n,l):
    s = [0 for i in xrange(l)]
    bestDistance = t*l
    i = 0
    while (i > 0 or bestDistance == t*l):
        if (i < l):
            prefix = nucleotideString(s)
            if (4**len(prefix) < n):
                s, i = NextVertex(s,i,l,4)
            else:
                optimisticDistance = TotalDistance(prefix, DNA)
                if (optimisticDistance > bestDistance):
                    s, i = Bypass(s,i,l,4)
                else:
                    s, i = NextVertex(s,i,l,4)
        else:
            word = nucleotideString(s)
            dist = TotalDistance(word, DNA)
            if (dist < bestDistance):
                print s, " = ", dist
                bestDistance = dist
                bestWord = word
            s, i = NextVertex(s,i,l,4)
    return bestWord
```

Question 2. Find a permutation with no decreasing strips for which there exists a reversal that reduces the number of breakpoints.
How many permutations are there of 6 elements? How many have a single breakpoint? How many permuations of 6 elements have exactly 2 breakpoints.

*Generalize your results from above for 6 elements to n elements.

The permutation [0,1,2|5,6|3,4|7] has no decreasing strips. Reversing the subsequence [5,6,3] results in [0,1,2,3|6,5,4|7], which eliminates one breakpoint. This reversal would not be selected by ImprovedBreakpointReversalSort.

There are 720 permutations of 6 elements. None have a single breakpoint, and 21 have two breakpoints. This generalizes to n! permutations, none have a single breakpoint, and $\binom{n}{2}$ have two breakpoints.

Question 3. In the *cookie game* two players begin with two stacks of cookies of heights n and m. In each turn a player eats two cookies from one stack and one cookie from the other. The player who cannot complete his turn loses. Who will win? Describe the winning strategy for any value of n and m (Hint: try dynamic programming).

If players use a perfect strategy, the winner is predetermined at the onset, as shown in the table below. Winning positions are denoted with W and losing ones with L. Superscripts indicate a winning move from the given position.

	0	1	2	3	4	5	6	7	8	9	10
0	L	L	L	L	L	L	L	L	L	L	L
1	L	L	$W^{0,0}$	$W^{1,0}$	$W^{2,0}$	$W^{3,0}$	$W^{4,0}$	$W^{5,0}$	$W^{6,0}$	$W^{7,0}$	$W^{8,0}$
2	L	$W^{0,0}$	$W^{1,0}$	$W^{2,0}$	$W^{3,0}$	$W^{4,0}$	$W^{5,0}$	$W^{6,0}$	$W^{7,0}$	$W^{8,0}$	$W^{9,0}$
3	L	$W^{0,1}$	$W^{0,2}$	L	L	L	L	L	L	L	L
4	L	$W^{0,2}$	$W^{0,3}$	L	L	$W^{3,3}$	$W^{4,3}$	$W^{5,3}$	$W^{6,3}$	$W^{7,3}$	$W^{8,3}$
5	L	$W^{0,3}$	$W^{0,4}$	L	$W^{3,3}$	$W^{4,3}$	$W^{5,3}$	$W^{6,3}$	$W^{7,3}$	$W^{8,3}$	$W^{9,3}$
6	L	$W^{0,4}$	$W^{0,5}$	L	$W^{3,4}$	$W^{3,5}$	L	L	L	L	L
7	L	$W^{0,5}$	$W^{0,6}$	L	$W^{3,5}$	$W^{3,6}$	L	L	$W^{6,6}$	$W^{7,6}$	$W^{8,6}$
8	L	$W^{0,6}$	$W^{0,7}$	L	$W^{3,6}$	$W^{3,7}$	L	$W^{6,6}$	$W^{6,7}$	$W^{8,6}$	$W^{9,6}$
9	L	$W^{0,7}$	$W^{0,8}$	L	$W^{3,7}$	$W^{3,8}$	L	$W^{6,7}$	$W^{6,8}$	L	L
10	L	$W^{0,8}$	$W^{0,9}$	L	$W^{3,8}$	$W^{3,9}$	L	$W^{6,8}$	$W^{6,9}$	L	L

Winning moves have the property that the opponent is left in a losing position. Thus, after each move attempt to make the smaller of the two stack heights a multiple of 3, or make the two stacks equal in height and have a remainder of either 0 or 1 when divided by 3. If you are in a losing position, try to make the stack heights more equal. This leads to more possible ways of winning if the opponent makes a mistake.

Question 4. Fill the global alignment dynamic programming matrix for strings TA and TTCA with affine scoring function defined by match premium 0, mismatch penalty -1, gap opening penalty -1, and gap extension penalty -1. Find *all* optimal global alignments.

If one assumes that gap penalties only apply within the interior of a sequence, the following scoring function results, which leads to two alignments.

	ε	T	T	C	A
ε	0	0	0	0	0
T	0	0	0	-1	-1
A	0	-1	-1	-1	-1

```
TTCA      TTCA
T_A        TA
```

If one assumes that gap penalties prior to the sequence the following scoring function results, which also leads to two alignments.

	ε	T	T	C	A
ε	0	-1	-2	-3	-4
T	-1	0	-2	-2	-3
A	-2	-1	-1	-2	-2

```
TTCA      TTCA
T__A      _T_A
```

Question 5. Consider the Manhattan Tourist problem that we discussed in class. Assume that we need to go back to the source (0,0) after we reach the sink (m, n), by going either to the north or to the west (from (m, n) to (0,0)). And we want to maximize the total number of distinct attractions along the combination of these two paths. Design a dynamic programming algorithm to find this pair of paths.
*How many total paths are there from (0,0) to (m,n)?

Programming Problem. Modify the Longest Common Subsequence Algorithm compnents LCS(v,w) and PrintLCS(b,v,i,j) given on page 176 to return *all* answers. Turn in a a code listing and your program's output for the following sequences:

```
v = "tcagggctgggcataaaaggaagaacagggacagctgctgcttacatttgcttctgaaacaaccgtgttc\
     actagcaaccacaaacagacaccatggtgcatctgactgctgaagagaagagtcttgtctccggcctgtg\
     gggcaaggtgaatgtggacgaagttggcggtgaggccctgggcaggctgctgattgtctacccctggact\
     cagaggttctttgactcctttggggacctgtccactcctgatgctgttatgagcaatgctaaagtgaagg\
     cccatggcaagaaggtgctgaactcctttagtgatggcctgaaaaacctggacaacctcaagggcacctt\
     tgctaagctcagtgagcttcactgtgacaagctgcacgtggatcccgagaacttcaagctcctgggcaac\
     gtgcttgtgtgtgtgctggctcaccactttggcaaagaattcacccctcaggtgcaggctgcctatcaga\
     aggtggtggctggtgtggccaatgccctggctcacaagtaccattgagatcctggcctgtttcctggtga\
     ccactggaagaccctgtctccctaaattctttcttctgaactgggggaaataatgtccaccatcaagggt\
     atggcttctgcctaataaagaaccttcagctcaactttctgattcattc"
w = "acatttgcttctgacacaactgtgttcactagcaacctcaaacagacaccatggtgcatctgactcctgag\
      gagaagtctgccgttactgccctgtggggcaaggtgaacgtggatgaagttggtggtgaggccctgggcag\
      gctgctggtggtctacccttggacccagaggttctttgagtccttggggatctgtccactcctgatgctg\
      ttatgggcaaccctaaggtgaaggctcatggcaagaaagtgctcggtgcctttagtgatggcctggctcac\
      ctggacaacctcaagggcacctttgccacactgagtgagctgcactgtgacaagctgcacgtggatcctga\
      gaacttcaggctcctgggcaacgtgctggtctgtgtgctggcccatcactttggcaaagaattcaccccac\
      cagtgcaggctgcctatcagaaagtggtggctggtgtggctaatgccctggcccacaagtatcactaagct\
      cgctttcttgctgtccaatttctattaaaggttcctttgttccctaagtccaactactaaactgggggata\
      ttatgaagggccttgagcatctggattctgcctaataaaaaacatttatttttcattgc"
```

The following changes how the backtracking array values are saved

```
from numpy import *

def LCS(v, w):
    s = zeros((len(v)+1,len(w)+1), dtype=int32)
    b = zeros((len(v)+1,len(w)+1), dtype=int32)
    for i in xrange(1,len(v)+1):
        for j in xrange(1,len(w)+1):
            if (v[i-1] == w[j-1]):
                s[i,j] = max(s[i-1,j], s[i,j-1], s[i-1,j-1] + 1)
            else:
                s[i,j] = max(s[i-1,j], s[i,j-1])
            if (s[i,j] == s[i,j-1]):
               b[i,j] += 1
            if (s[i,j] == s[i-1,j]):
               b[i,j] += 2
            if (v[i-1] == w[j-1]) and (s[i,j] == s[i-1,j-1] + 1):
               b[i,j] += 4
    return (s[i,j], b)
```

The following changes are to backtracking itself

```
answers = dict()

def PrintLCS(b,v,w,i,j,result=[]):
    global answers
    if ((i == 0) or (j == 0)):
        lcs = "".join(result)
        t = answers.get(lcs, 0)
        answers[lcs] = t + 1
        if (t == 0):
            print "LCS =", lcs
        return
    if (b[i,j] & 4 == 4):
        result = [v[i-1]] + result
        PrintLCS(b,v,w,i-1,j-1,result)
        result = result[1:]
    if (b[i,j] & 2 == 2):
        PrintLCS(b,v,w,i-1,j,result)
    if (b[i,j] & 1 == 1):
        PrintLCS(b,v,w,i,j-1,result)
```

This modified version of LCS finds the following six distinct LCSs with scores of 538 with the variable parts highlighted:

LCS =
acatttgcttctgcacaacgtgttcactagcaacccaaacagacaccatggtgcatctgactctgagagaagtctttctccctgtggggcaagg
tgaagtggagaagttggggtgaggccctgggcaggctgctgtgtctaccctggaccagaggttctttgatcctttggggactgtccactcctga
tgctgttatggcaactaagtgaaggccatggcaagaagtgctgtcctttagtgatggcctgacctggacaacctcaagggcacctttgcaacta
gtgagctcactgtgacaagctgcacgtggatccgagaacttcagctcctgggcaacgtgctgttgtgtgctggccacactttggcaaagaattc
acccccagtgcaggctgcctatcagaagtggtggctggtgtggcaatgccctggccacaagtaccaagtccttttctgtccactaagcctgtt
ccctaatcctctaactgggggaaataagccacatcggattctgcctaataaaaactttattttcattc

LCS =
acatttgcttctgaacaacgtgttcactagcaacccaaacagacaccatggtgcatctgactctgagagaagtctttctccctgtggggcaagg
tgaagtggagaagttggggtgaggccctgggcaggctgctgtgtctaccctggaccagaggttctttgatcctttggggactgtccactcctga
tgctgttatggcaactaagtgaaggccatggcaagaagtgctgtcctttagtgatggcctgacctggacaacctcaagggcacctttgcaacta
gtgagctcactgtgacaagctgcacgtggatccgagaacttcagctcctgggcaacgtgctgttgtgtgctggccacactttggcaaagaattc
acccccagtgcaggctgcctatcagaagtggtggctggtgtggcaatgccctggccacaagtaccaagtccttttctgtccactaagcctgtt
ccctaatcctctaactgggggaaataagccacatcggattctgcctaataaaaactttattttcattc

LCS =
acatttgcttctgcacaacgtgttcactagcaacccaaacagacaccatggtgcatctgactctgaggaaagtctttctccctgtggggcaagg
tgaagtggagaagttggggtgaggccctgggcaggctgctgtgtctaccctggaccagaggttctttgatcctttggggactgtccactcctga
tgctgttatggcaactaagtgaaggccatggcaagaagtgctgtcctttagtgatggcctgacctggacaacctcaagggcacctttgcaacta
gtgagctcactgtgacaagctgcacgtggatccgagaacttcagctcctgggcaacgtgctgttgtgtgctggccacactttggcaaagaattc
acccccagtgcaggctgcctatcagaagtggtggctggtgtggcaatgccctggccacaagtaccaagtccttttctgtccactaagcctgtt
ccctaatcctctaactgggggaaataagccacatcggattctgcctaataaaaactttattttcattc

LCS =
acatttgcttctgaacaacgtgttcactagcaacccaaacagacaccatggtgcatctgactctgaggaaagtctttctccctgtggggcaagg
tgaagtggagaagttggggtgaggccctgggcaggctgctgtgtctaccctggaccagaggttctttgatcctttggggactgtccactcctga
tgctgttatggcaactaagtgaaggccatggcaagaagtgctgtcctttagtgatggcctgacctggacaacctcaagggcacctttgcaacta
gtgagctcactgtgacaagctgcacgtggatccgagaacttcagctcctgggcaacgtgctgttgtgtgctggccacactttggcaaagaattc
acccccagtgcaggctgcctatcagaagtggtggctggtgtggcaatgccctggccacaagtaccaagtccttttctgtccactaagcctgtt
ccctaatcctctaactgggggaaataagccacatcggattctgcctaataaaaactttattttcattc

LCS =
acatttgcttctgcacaacgtgttcactagcaacccaaacagacaccatggtgcatctgactctgaggagagtctttctccctgtggggcaagg
tgaagtggagaagttggggtgaggccctgggcaggctgctgtgtctaccctggaccagaggttctttgatcctttggggactgtccactcctga
tgctgttatggcaactaagtgaaggccatggcaagaagtgctgtcctttagtgatggcctgacctggacaacctcaagggcacctttgcaacta
gtgagctcactgtgacaagctgcacgtggatccgagaacttcagctcctgggcaacgtgctgttgtgtgctggccacactttggcaaagaattc
acccccagtgcaggctgcctatcagaagtggtggctggtgtggcaatgccctggccacaagtaccaagtccttttctgtccactaagcctgtt
ccctaatcctctaactgggggaaataagccacatcggattctgcctaataaaaactttattttcattc

LCS =
acatttgcttctgaacaacgtgttcactagcaacccaaacagacaccatggtgcatctgactctgaggagagtctttctccctgtggggcaagg
tgaagtggagaagttggggtgaggccctgggcaggctgctgtgtctaccctggaccagaggttctttgatcctttggggactgtccactcctga
tgctgttatggcaactaagtgaaggccatggcaagaagtgctgtcctttagtgatggcctgacctggacaacctcaagggcacctttgcaacta
gtgagctcactgtgacaagctgcacgtggatccgagaacttcagctcctgggcaacgtgctgttgtgtgctggccacactttggcaaagaattc
acccccagtgcaggctgcctatcagaagtggtggctggtgtggcaatgccctggccacaagtaccaagtccttttctgtccactaagcctgtt
ccctaatcctctaactgggggaaataagccacatcggattctgcctaataaaaactttattttcattc