Algorithms Fall 2016 Homework 1 Solution

September 28, 2016

PROBLEM 1: (20 POINTS)

Since each cut incurs a fixed cost of c, we have the following recursive formula (note when there is no cut, no cut cost is incurred):

$$r_n = max(p_1 + r_{n-1} - c, p_2 + r_{n-2} - 1, ..., p_{n-1} + r_1 - c, p_n)$$

The MODIFIED-MEMOIZED-CUT-ROD should be the same as MEMOIZED-CUT-ROD. Only the MEMOIZED-CUT-ROD-AUX should be modified.

MODIFIED-MEMOIZED-CUT-ROD-AUX(p, n, r)

```
if r[n] >= 0
        return r[n]
if n == 0
        q = 0
   else q = −∞
        for i = 1 to n
              if i ≠ n
                    q = max(q, p[i] + MODIFIED−MEMOIZED−CUT−ROD−AUX(p, n − i, r) − c)
                else
                    q = max(q, p[n])
r[n] = q
return q
```

MODIFIED-BOTTOM-UP-CUT-ROD(p, n, c)

```
let r[0...n] be a new array
r[0] = 0
for j = 1 to n
        q = p[j]
        for i = 1 to j − 1
              q = max(q, p[i] + r[j − i] − c)
        r[i] = q
return r[n]
```

PROBLEM 2: (20 POINTS)

We define $score[i, j]$ to be the similarity score of the sequence X_i and Y_j. We have the following recursive formula for the String Score Problem:

$$score[i,j] = \begin{cases} -j & \text{if } i = 0 \\ -i & \text{if } j = 0 \\ score[i-1, j-1] + 3 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max(score[i, j-1], score[i-1, j]) - 1 & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

Based on the above recursive formula, the pseudo code is as follows:

STRING-SIMILARITY-SCORE(X, Y)

```
m = X.length
n = Y.length
let score[0...m, 0...n] be new tables
for i = 1 to m
        score[i, 0] = −i
for j = 1 to n
        score[0, j] = −j
for i = 1 to m
        for j = 1 to n
                if x_i == y_i
                        score[i, j] = score[i − 1, j − 1] + 3
                else if score[i − 1, j] >= score[i, j − 1]
                        score[i, j] = score[i − 1, j] − 1
                else
                        score[i, j] = score[i, j − 1] − 1
return score[m, n]
```

PROBLEM 3: (20 POINTS)

This problem does exist optimal substructure property. Suppose the highest level parenthesization splits the matrix chain into two sub chains. The parenthesization within each sub-chain must be such that they maximize the number of scalar multiplications involved for each such chain. If that is not the case (e.g the sub-chain could be the parenthesized in another way that increases the multiplications for that sub chain), we could obtain a higher number of total multiplications for the entire chain.

PROBLEM 4: (40 POINTS)

We will first prove the optimal substructure of LPS problem. The proof will help us derive the recursive formula and prove the correctness of the bottom-up algorithm implemented.
Theorem(Optimal substructure of a LPS)
Let $S =< s_1, s_2, ..., s_n >$ be the input sequence and $X =< x_1, x_2, ..., x_n >$ be a LPS of S, then for $n > 2$, we have:

(a) If $s_1 = s_n$, then $x_1 = x_m = s_1 = s_n$, and $X_{2,m-1}$ is a LPS of $S_{2,n-1}$

(b) If $s_1 \neq s_n$, then $X_{1,m}$ is either a LPS of $S_{2,n}$ or a LPS of $S_{1,n-1}$

Proof. In the theorem, we only consider the format of LPS of S when $n > 2$. Determining the LPS of S when $n <= 2$ is trivial. We prove conclusion (a)(b) respectively as follows:

(a) Suppose $x_1 \neq s_1$, then we can prepend and append s_1 to X to get a palindrome sequence of S with greater length than X which contradicts that X is an LPS of S. Thus $x_1 = s_1 = s_n = x_m$. Then $X_{2,m-1}$ is a palindrome subsequence of $S_{2,n-1}$. We claim that is also must be a LPS of $S_{2,n-1}$. Suppose it is not a LPS of $S_{2,n-1}$, then there must exist a palindrome subsequence Y which is longer than $X_{2,m-1}$. By prepending and appending s_1 to Y, we get a palindrome subsequence of S which is longer than X which is a contradiction to that X is a LPS of S.

(b) Since X is a palindrome subsequence of s and $s_1 \neq s_n$, it must be the case that either $x_1 \neq s_1$ or $x_m \neq s_n$. If $x_1 \neq s_1$, then X is a palindrome subsequence of $S_{2,n}$. If there is a paindrome subsequence Y of $S_{2,n}$ which is longer than X, Y would also be a palindrome subsequence of S contradicting that X is a LPS of S. Thus X is a LPS of $S_{2,n}$. When $x_m \neq s_n$, using similar proof, we can prove that X is a LPS of $S_{1,n-1}$.

Recursive formula
For a sequence $S =< s_1, s_2, ..., s_n >$, $l[i, j]$ denotes the length of an LPS(Longest Palindrome Subsequence) of the subsequence S_{ij}. We have following recursive formula(the first three cover the base cases) to calculate $l[i, j]$:

- **If** $i = j$, $l[i, j] = 1$
- **If** $j = i+1$ and $s_i = s_j$, $l[i][j] = 2$
- **If** $j = i+1$ and $s_i \neq s_j$, $l[i][j] = 1$
- **If** $j > i+1$ and $s_i = s_j$, $l[i][j] = l[i+1][j-1] + 2$
- **If** $j > i+1$ and $s_i \neq s_j$, $l[i][j] = max(l[i+1][j], l[i][j-1])$

Pseudo-code for computing the length of LPS

```
LONGEST-PALINDROME-SUBSEQUENCE(S)
1   n = S.length
2   let d[1...n, 1...n] be new tables
3   let l[1...n, 1...n] be new tables
4   for i = 1 to n - 1
5           l[i, i] = 1
6           if s_i == s_{i+1}
7                   l[i, j] = 2
8                   d[i, j] = "B"
9             else
10                  l[i, j] = 1
11                  d[i, j] = "L"
12  l[n, n] = 1
13  for k = 3 to n
14          for i = 1 to n - k + 1
15                  if s[i] == s[i + k - 1]
16                          l[i, i + k - 1] = l[i + 1, i + k - 2] + 2
17                          d[i, i + k - 1] = "B"
18                    else  if l[i, i + k - 2] >= l[i + 1, i + k - 1]
19                          l[i, i + k - 1] = l[i, i + k - 2]
20                          d[i, i + k - 1] = "L"
21                    else
22                          l[i, i + k - 1] = l[i + 1, i + k - 1]
23                          d[i, i + k - 1] = "R"
24  return l and d
```

The running time for this algorithm is $O(n^2)$ (two levels of loops from line 13 to 23)

Pseudo-code for constructing LPS

```
CONSTRUCT-LPS(S, i, j, d)
if i > j
        return ""
   else  if i == j
        return s_i
   else  if d[i, j] = "B"
        return s_i + CONSTRUCT-LPS(S, i + 1, j - 1, d) + s_j
   else  if d[i, j] == "L"
        return CONSTRUCT-LPS(S, i, j - 1, d)
   else
        return CONSTRUCT-LPS(S, i + 1, j, d)
```