CSCI-GA.1170-001/002 Fundamental Algorithms October 31, 2014

Solutions to Problem 1 of Homework 7 (6 points)

Name: Sahil Goel *Due: Wednesday, November 5*

Using dynamic programming, find the optimum printing of the text *"Not all those who wander are lost"*, i.e. $\ell_1 = 3, \ell_2 = 3, \ell_3 = 5, \ell_4 = 3, \ell_5 = 6, \ell_6 = 3, \ell_7 = 4$, with line length $L = 14$ and penalty function $P(x) = x^3$. Will the optimal printing you get be consistent with the strategy "print the word on as long as it fits, and otherwise start a new line"? Once again, you have to actually find the alignment, as opposed to only finding its penalty.

Solution: We will first initialize array m[7] with values infinity. Suppose m_i holds the optimal penalty up to i words. Initially, we will try to minimize it for m_1.
1
We will fill the first one into first row "Not _ _ _ _ _ _ _ _ _ _ _" Penalty will be $11^3 = 1331$
therefore array after first iteration will be [1331, $\infty, \infty, \infty, \infty, \infty, \infty$]

Now for m_2
There are two options
1. Include "all" in second row then m[2] = m[1] + p(2) = 1331 + $(11)^3$ = 2662
2. Include "all" in first row, then the row will be "Not all". There are 7 spaces after "all" Therefore penalty m[2] = 7*7*7 = 343 m[2] = min(2662,343)
Therefore array after second iteration will be [1331, 343, $\infty, \infty, \infty, \infty, \infty$]
and alignment is "Not All"

Now for m_3, there are again two options
1. Include "those" in second row, alignment will be
"Not all _ _ _ _ _ _ _"
"those _ _ _ _ _ _ _ _"
then m[3] = m[2] + p = $343 + 7^3 + 8^3 = 1198$
2. Include "those" in the first row itself
"Not all those _"
then m[3] = $1^3 = 1$
Therefore array after thirs iteration will be $[1331, 343, 1, \infty, \infty, \infty, \infty]$
and alignment is
"Not all those _"

Now for m_4, different cases will be
1. Include only "who" in new row
Alignment will be
"Not all those _"
"who _ _ _ _ _ _ _ _ _ _ _"
m[4] = m[3] + 11^3
m[4] = 1 + 1331 = 1332
2. Include "those who" in new row

Alignment will be
"Not all _ _ _ _ _ _ _"
"those who _ _ _ _ _"
m[4] = m[2] + 5^3 = 343 + 125 = 468
3. Include "all those who" in new row
Alignment will be
"Not _ _ _ _ _ _ _ _ _ _ _"
"all those who _"
m[4] = m[1] + 1^3 = 1331 + 1 = 1332

m[4] = min(1332,468,1332) = 468
optimal alignment is
"Not all _ _ _ _ _ _ _"
"those who _ _ _ _ _"
Array is $[1331, 343, 1, 468, \infty, \infty, \infty]$

Now for m_5, different cases will be
1. Include "wander" in new row
Alignment will be "Not all _ _ _ _ _ _ _"
"those who _ _ _ _ _"
""wander _ _ _ _ _ _ _ _ _ _"
m[5] = m[4] + 8^3 = 468 + 512 = 980

2. "Not all those _"
"who wander _ _ _ _ "
m[5] = m[3] + 4^3
m[5] = 1 + 64 = 65

m[5] = min (980,65) = 65
Alignment will be "Not all those _"
"who wander _ _ _ _"
Array will be
$[1331, 343, 1, 468, 65, \infty, \infty]$

Now for m_6, different cases will be
1. Include "are" in new line Alignment will be "Not all those _"
"who wander _ _ _ _ "
"are _ _ _ _ _ _ _ _ _ _ _"
m[6] = m[5] + 11^3 = 65 + 1331 = 1396
2. Include "wander are" in new line Alignment will be
"Not all _ _ _ _ _ _ _"
"those who _ _ _ _ _"
"wander are _ _ _ _"
m[6] = m[4] + 4^3 = 468 + 64 = 532

3. Include "who wander are" in new line Alignment will be
"Not all those _" "who wander are " m[6] = m[3] + 0 = 1+0 = 1

m[6] = min(1457,593,2) = 1
And alignment is
"Not all those _" "who wander are " Array is $[1331, 343, 1, 468, 65, 1, \infty]$

Now for m_7, different cases will be
1. Include "lost" in new row
Alignment will be
"Not all those _" "who wander are " "lost _ _ _ _ _ _ _ _ _ _"
m[7] = m[6] + 10^3 = 1001
2. Include "are lost" in new row
Alignment will be
"Not all those _"
"who wander _ _ _ _"
"are lost _ _ _ _ _ _"
m[7] = m[5] + 6^3 = 65 + 216 = 281
m[7] = min(1001,281) = 281
Final array is $[1331, 343, 1, 468, 65, 1, 281]$

Optimal alignment is
"Not all those _"
"who wander _ _ _ _ "
"are lost _ _ _ _ _ _"

□

CSCI-GA.1170-001/002 Fundamental Algorithms October 31, 2014

Solutions to Problem 2 of Homework 7 (20 points)

Name: Sahil Goel *Due: Wednesday, November 5*

Let $X[1 \dots m]$ and $Y[1 \dots n]$ be two given arrays. A common supersequence of X and Y is an array $Z[1 \dots k]$ such that X and Y are both subsequences of $Z[1 \dots k]$. Your goal is to find the *shortest* common super-sequence (SCS) Z of X and Y, solving the following sub-problems.

(a) (4 points) First, concentrate on finding only the length k of Z. Proceeding similarly to the longest common subsequence problem, define the appropriate array $M[0 \dots m, 0 \dots n]$ (in English), and the write the key recurrence equation to recursively compute the values $M[i, j]$ depending on some relation between $X[i]$ and $Y[j]$. Do not forget to explicitly write the base cases $M[0, j]$ and $M[i, 0]$, where $1 \leq i \leq m, 1 \leq j \leq n$.

Solution: M will be a 2 dimensional array of size (m+1)X(n+1) where each entry in $M_{i,j}$ gives the length of shortest common supersequence of $X_1X_2 \dots X_i$ and $Y_1Y_2 \dots Y_j$
. i will vary from 1 to m and j will vary from 1 to n
M[0,j] gives the length of shortest common supersequence of $Y_1Y_2 \dots Y_j$ which is equal to j itself
M[i,0] gives the length of shortest common supersequence of $X_1Y_2 \dots X_j$ which is equal to i itself
Therefore for base case
M[0,j] = j for $1 \leq j \leq n$
M[i,0] = i for $1 \leq i \leq m$
and M[0][0] = 0 since length of common supersequence of two null words is 0
Now, **Recursive formula will be**
M[i,j] = 1 + M[i-1][j-1] **if** $X_i = Y_j$
M[i,j] = 1 + min (M[i-1,j] , M[i][j-1]) **if** X_i **is not equal to** Y_j

If $X_i = Y_j$ then two characters are equal and the character should be added to longest common subsequence
Otherwise, if two characters are not equal, then both the characters should be added but there will be two cases
(a). X_i before Y_j = total length will be 1 (for Y_j) + M[i,j-1]
(b). Y_j before X_i = total length will be 1 (for X_i) + M[i-1,j]
We will take the minimum of these two cases for the optimal solution

□

(b) (5 points) Translate this recurrence equation into an explicit bottom-up $O(mn)$ time algorithm that computes the length of the shortest common supersequence of X and Y.

Solution:

{
Let M be new array of size (m+1)X(n+1) all initialized to all zeros
Setting all the base cases
M[0][0] = 0
for i = 1 to m
M[i][0] = i
for j = 1 to n
M[0][n] = j

for(i = 1 to m)
{
for(j = 1 to n)
{
if($X_i = Y_j$)
M[i][j] = 1 + M[i-1][j-1]
else
M[i][j] = 1 + MINIMUM (M[i,j-1] , M[i-1, j])

} %End First For Loop
} %End Second For loop
} %End whole function

□

(c) (5 points) Find the SCS of $X = BARRACUDA$ and $Y = ABRACADABRA$. (Notice, you need to find the actual SCS, not only its length.)

Solution: In this case, along with the values of length, we will also store a direction character which will tell information about the previous character from where we have arrive to current position. For example, if we are filling the values of m[i][j], then three possible cells from where we would have arrived at m[i][j] are m[i-1][j-1], m[i-1][j] and m[i[j-1]. Possible characters are :
d: Diagnol (M[i-1][j-1] l: Left (M[i][j-1]) u: Up (M[i-1][j])
So the conditions now will be
if($X_i = Y_j$)
M[i][j] = 1 + M[i-1][j-1] & S[i][j] = d
else if($M[i][j-1] \leq M[i-1][j]$)
M[i][j] = 1 + M[i][j-1] & S[i][j] = l
else
M[i][j] = 1 + M[i-1][j] & S[i][j] = u
For base cases, S[i][j] is initialized as follows
For i=1 to m, S[i,0] = u ;
For j=1 to n, S[0,j] = l ;

The calculated table for two given words will be of size (9+1)X(11+1)
To show S[i][j] in more understandable form, I have concatenated it to the M[i][j] value

Initially length=M[m][n], i=m, j=n, scs = char[legth]
while($i \geq 0$ & $j \geq 0$)
ch = S[i][j];

if(ch=='d')
scs[length - -] = X[i];
i=i-1 & j=j-1

if(ch=='l')
scs[length - -] = Y[j];
j=j-1;

else scs[length - -] = X[i];
i=i-1;

end while

for i=1:length
print scs[i]
end for

This will print final word
SCS = **B A R B R A C U A D A B R A**

□

(d) (6 points) Show that the length k of the array Z computed in part (a) satisfies the equation $k = m + n - \ell$, where ℓ is the length of the longest common *subsequence* of X and Y. (**Hint**: Use the recurrence equation in part (a), then combine it with a similar recurrence equation for the LCS, and then use induction. There the following identity is very handy: $\min(a, b) + \max(a, b) = a + b$.)

Solution:

□

CSCI-GA.1170-001/002 Fundamental Algorithms October 31, 2014

Solutions to Problem 3 of Homework 7 (18 points)

Name: Sahil Goel *Due: Wednesday, November 5*

You are a CFO of a baby sitting company, and got a request to baby sit n children one day. You can hire several babysitters for a day for a fixed cost B per babysitter. Also, you can assign an arbitrary number of children $i \geq 1$ to a babysitter. However, each parent will only pay some amount $p[i]$ if his child is taken care of by a babysitter who looks after i children. For example, if $n = 7$ and you hire 2 babysitters who looks after 3 and 4 children, respectively, you revenue is $3p[3] + 4p[4] - 2B$.

Given $B, n, p[1], \ldots, p[n]$, your job is to assign children to babysitters as to maximize your total profit. Namely, you want to find an optimal number k and an optimal partition $n = n_1 + \ldots + n_k$ so as to maximize revenue $R = n_1 \cdot p[n_1] + \ldots + n_k \cdot p[n_k] - k \cdot B$.

(a) (5 points) Let $R[i]$ denote the optimum revenue you can get by looking after i children. E.g., $R[0] = 0$, $R[1] = p[1] - B$, $R[2] = \max(2p[1] - 2B, 2p[2] - B)$, etc. Write a top-down recursive formula for $R[n]$ in terms of values $R[j]$ for $j < n$.

Solution:
R[n] = maximum(R[n-1] + R[1] , R[n-2] + R[2], ... , R[1] + R[n-1], n*P[n] - B)

R[n] = maximum (R[n-q] + R[q], n*p[n] - B) Where q varies from 1 to n/2

□

(b) (5 points) Write a top-down recursive procedure with memorization which will compute $R[n]$. Analyze the running time of your procedure in the $\Theta(\cdot)$ notation.

Solution: OptimalBSTopDown
{
Initialize an array r[n] with all values = -∞
OptimalBSTopDown-AUX(p,n,r)
}

OptimalBSTopDown-AUX(p,n,r)
{
if($r[n] \geq 0$)
return r[n]
else
if (n==0) temp=0;
else {
$temp = n * p[n] - B$

for q=1 to n/2
temp = max(temp, OptimalBSTopDown-AUX(p,n-q,r) + OptimalBSTopDown-AUX(p,q,r))
end for
r[n]=temp
return temp
}

Analysis of running time
Initially when OptimalBSTopDown-AUX is called, it is called with n
To calculate r[n], the inner loop will be run at least n/2 times

Now different branches will be
T(1) + T(n-1), T(2) + T(n-2), T(3) + T(n-3), ..., T(n/2) + T(n/2)
But we know that T(1) will be run only once so is T(2) and all other. As when it is again called for T(1), it will just return from the function form the first step and no work will be done
So you can see that for size n, loop runs n/2 times
for size n-1, loop runs (n-1)/2 times
Similarly for size 1, loop runs 0 times
Hence total work done or total complexity will be
$0 + 1 + \ldots + (n-1)/2 + n/2 = \Theta(n^2)$

Hence complexity of algorithm will be $\Theta(n^2)$

□

(c) (5 points) Write an iterative bottom-up variant of the same procedure.

Solution: Bootom-Up-Baby-Sitter(p,n)
{
let r[n] be a new array
r[0]=0
for j = 1 to n
temp = n*p[n] - B
for q = 1 to j/2
temp = max (r[j-q] + r[q],temp)
end for
r[j] = temp
end for
return r[n]
}

Time complexity of this solution will be $\Theta(n^2)$ as
For j=1, inner loop will run 0 times
For j=2, inner loop will run 1 time
.
.

For j=n, inner loop will run n/2 times
Total T(n) = 0 + 1 + 2 + 3 + ... + n/2 = $(n^2 + n)/4 = \Theta(n^2)$

□

(d) (3 points) Explain how to augment your procedure (either in part (b) or (c)) to also compute the optimal number of babysitters k and the actual partition of children. Either English or pseudocode will work.

Solution: For storing the partition info, we will have to create a new array say s[n]
We will make some changes in the code
1. When we initialized temp, we have to add one line in the initializisation it as
temp = n*p[n] - B s[n] = n

2. Now inside the second nested for loop in which q varies from 1 to n/2
We will re write the lines as
if (temp ¡ r[n-q] + r[q])
temp = r[n-q] + r[q]
s[n] = q

Now for retrieving back the actual partition of children we will add the below-mentioned function and call it as print-optimalset(s,n)

```
print-optimalset(s,n)
{
if(n≤0)
print("Done printing")
else if (s[n] = n)
print(s[n]
else
print-optimalset(s,n-s[n])
print-optimalset(s,s[n])
}
```

□

CSCI-GA.1170-001/002 Fundamental Algorithms October 31, 2014

Solutions to Problem 4 of Homework 7 (10 points)

Name: Sahil Goel *Due: Wednesday, November 5*

You have $m \times n$ chocolate bar. You are also given a matrix $\{p[i,j] \mid 1 \leq i \leq m, 1 \leq j \leq n\}$ telling you the price of the $i \times j$ chocolate bar. You are allowed to repeat the following procedure any number of times, starting initially with the single big $m \times n$ piece you have. Take one of the pieces you have and split it into two pieces by cutting it either vertically or horizontally. Say, $m = 5$, $n = 4$. You may first choose to split it into two pieces of size 3×4 and 2×4. Then you may take the 3×4 piece and split it into two pieces 3×2 and 1×2. Finally, you may take the previous 2×4 piece and split it into two 1×4 pieces. If you stop, you have four pieces of sizes 3×2, 1×2, 1×4 and 1×4, which you can sell for $p[3,2] + p[1,2] + 2p[1,4]$. You goal is to find a partition maximizing your total profit.

(a) (5 points) Let $C[i,j]$ be the largest profit you can get by splitting an $i \times j$ piece, where $0 \leq i \leq m$, $0 \leq j \leq n$ and we set $C[i,0] = C[0,j] = 0$. Write a recursive formula for $C[m,n]$ in terms of values $C[i,j]$, where either $i < m$ or $j < n$.

Solution:
C[m][n] = MAX(MAX(C[i,n] + C[m-i,n]) where $1 \leq i \leq m$, MAX (C[m,j] + C[m,n-j]) where $1 \leq j \leq n$, P[m,n])
In short
C[m,n] = MAX (C[i,n] + C[m-i,n] , C[m,j] + C[m,n-j], P[m][n]) where
$1 \leq i < m$ and $1 \leq j < n$
To remove redundancy we can optimize this formula by changing the limits of i and j
$1 \leq i \leq m/2$ and $1 \leq j \leq n/2$

□

(b) (5 points) Write a bottom-up procedure to compute $C[m,n]$ and analyze its running time as a function of m and n.

Solution: Bottom-up-chocolate(p,m,n)
{
let c[m,n] be new array
% base cases
for i = 1 to m
c[i][0] = 0
end for
for j = 1 to n
c[0][j] = 0
end for

```
for i = 1 to m
for j = 1 to n
temp = p[i][j];
for k = 1 to m/2
temp = max ( temp, c[k,n] + c[m-k,n])
end for
for k = 1 to n/2
temp = max (temp, c[m,k] + c[m,n-k])
end for
c[i][j] = temp
end for
end for
return c[m,n]
}
```

To analyze the Time Complexity
Outermost loop runs m times
Second to Outermost loop runs n times
third to innermost loop runs m/2 + n/2 times
Therefore total complexity will be m*n*(m/2+n/2) = mn(m+n) = $O(mn(m+n))$

□