NORTHWESTERN UNIVERSITY

EECS 336 – Homework 4

Weihan Chu

May 5, 2016

1 QUESTION 1

This algorithm is not right. For example, if $X = BCDAAAAAAA$, $Y = AAAAAAABCD$ from those two strings, we can easily get that the longest common sub-sequence is $AAAAAAA$, but if we use the algorithm this article provided, we start from the beginning. We can find that B fits the requirement. So B must be one element of the result, which contracts the fact. So this algorithm is not right.

2 QUESTION 2

I think those two methods provide same optimal trees. Suppose that the lecture cost model is $c_1(i, j)$, and the book cost model is $c_2(i, j)$, where i, j is the binary search tree from $k_i, k_i + 1, ...k_j$.Assume they have different result that: $c_2(i, j) - c_1(i, j) = \Sigma_{i \le m \le j} a_m$
if k_o is the point that we can get the optimal binary search tree. for any $k \in [i, j]$, we have that :

$$c_1(i, k-1) + c_1(k, j) \ge c_1(i, k_o - 1) + c_1(k_o, j)$$

from above we can get that:

$$c_1(i,k-1)+\Sigma_{i \le m \le (k-1)} a_m + c_1(k,j) + \Sigma_{k \le m \le j} a_m \ge c_1(i,k_o-1) + \Sigma_{i \le m \le (k-1)} a_m + c_1(k_o,j) + \Sigma_{k \le m \le j} a_m$$

from above we can get that:

$$c_2(i, k-1) + c_2(k, j) \geq c_2(i, k_o - 1) + c_2(k_o, j)$$

so this equation tells us that k_o is the result,which means k_o is also the optimal binary search tree point for c_2. So those two cost model actually provide the same answer.

3 QUESTION 3

3.1 PART A

In this problem, we use C represent the length of common sub-sequence and we use W to represent the weight. So we use $C[i, j]$ to represent the length of longest common sub-sequence till now, and we use the $W[i, j]$ to represent the total weight till now.

Algorithm 1 shortest maximum-weight common sub-sequence

m=X.length, n=Y.length
let c[1..m,1..n] and w[1..m,1..n] be new tables
for $i \in [1, m]$ **do**
  $[c[i,0], w[i,0]] = [0,0]$
end for
for $j \in [0, n]$ **do**
  $[c[0,j], w[0,j]] = [0,0]$
end for
for $i \in [i, m]$ **do**
  for $j \in [1, n]$ **do**
    if $x_i == y_i$ **then**
      $c[i,j] = c[i-1, j-1] + 1$, $w[i,j] = w[i-1, j-1] + w_i$
      then compare $w[i,j], w[i,j-1], w[i-1,j]$
      $w[i,j] = \max(w[i,j], w[i,j-1], w[i-1,j])$
    else if $w[i,j-1] > w[i-1,j]$ **then**
      $c[i,j] = c[i,j-1]$, $w[i,j] = w[i,j-1]$
    else if $w[i-1,j] > w[i,j-1]$ **then**
      $c[i,j] = c[i-1,j]$, $w[i,j] = w[i-1,j]$
    else if $w[i-1,j] == w[i,j-1]$ **then**
      if $c[i-1,j] < c[i,j-1]$ **then**
        $c[i,j] = c[i-1,j]$, $w[i,j] = w[i-1,j]$
      else
        $c[i,j] = c[i,j-1]$, $w[i,j] = w[i,j-1]$
      end if
    end if
  end for
end for
return w and c

To prove this algorithm, we just need to go through the whole process. There are three major cases:
Case 1:: if $i = 0 || j = 0$,this means one of the alphabet's length is 0. so the total common sub-sequence's length and the weight should be zero.
Case 2:: if $x_i == y_i$,this means that this character is useful for us to get the maximum weight because it can increase the total weight. So if this case happens, we should include this into our maximum weight calculation.
Case 3:: if $x_i != y_i$, there are also three small cases:
3.1:if $w[i, j-1] > w[i-1, j]$, this means the above weight is bigger than the left weight, so this place's value should inherit the above value.
3.2:if $w[i-1, j] > w[i, j-1]$, this means the left weight is bigger than the above weight, so this place's value should inherit the left value.
3.3:if $w[i-1, j] == w[i, j-1]$, this means the left weight is equal to the above value, so we should check whose length is less. If left length is less than the above length, we should choose the left length. If above length is less than left length, we should choose the above length.
By this method, we can find the final result.
For the time complexity, this algorithm takes $O(m * n)$ time complexity because it create a $m * n$ size table and to get every element in the table, it need $O(1)$.

3.2 PART B

This problem is totally same as the the above problem, we only need to change some characters and conditions to get the final answer. To get the problem solved clear. I'll write the whole answer. In this problem, we use C represent the length of common sub-sequence and we use W to represent the weight. So we use $C[i, j]$ to represent the length of longest common sub-sequence till now, and we use the $W[i, j]$ to represent the total weight till now.

Algorithm 2 maximum-weight longest common sub-sequence

```
m=X.length, n=Y.length
let c[1..m,1..n] and w[1..m,1..n] be new tables
for i ∈ [1, m] do
    [c[i,0], w[i,0]] = [0,0]
end for
for j ∈ [0, n] do
    [c[0, j], w[0, j]] = [0,0]
end for
for i ∈ [i, m] do
    for j ∈ [1, n] do
        if x_i == y_i then
            c[i, j] = c[i − 1, j − 1] + 1, w[i, j] = w[i − 1, j − 1] + w_i
            then compare c[i, j], c[i, j − 1], c[i − 1, j]
            c[i, j] = max(c[i, j], c[i, j − 1], c[i − 1, j])
        else if c[i, j − 1] > c[i − 1, j] then
            c[i, j] = c[i, j − 1], w[i, j] = w[i, j − 1]
        else if c[i − 1, j] > c[i, j − 1] then
            c[i, j] = c[i − 1, j], w[i, j] = w[i − 1, j]
        else if c[i − 1, j] == c[i, j − 1] then
            if w[i − 1, j] > w[i, j − 1] then
                c[i, j] = c[i − 1, j], w[i, j] = w[i − 1, j]
            else
                c[i, j] = c[i, j − 1], w[i, j] = w[i, j − 1]
            end if
        end if
    end for
end for
return w and c
```

To prove this algorithm, we just need to go through the whole process. There are three major cases:
Case 1:: if $i = 0 || j = 0$,this means one of the alphabet's length is 0. so the total common sub-sequence's length and the weight should be zero.
Case 2:: if $x_i == y_i$,this means that this character is useful for us to get the longest common sub-sequence because it can increase the total length. So if this case happens, we should include this into our longest length calculation.
Case 3:: if $x_i != y_i$, there are also three small cases:
3.1:if $c[i, j-1] > c[i-1, j]$, this means the above common length is bigger than the left common length, so this place's value should inherit the above value.
3.2:if $c[i-1, j] > c[i, j-1]$, this means the left common length is bigger than the common length, so this place's value should inherit the left value.

3.3:if $c[i-1,j] == c[i,j-1]$, this means the left common length is equal to the above value, so we should check whose weight is bigger. If left weight is bigger than the above weight, we should choose the left weight. If above weight is bigger than left weight, we should choose the above weight.
By this method, we can find the final result.
For the time complexity, this algorithm takes $O(m*n)$ time complexity because it create a $m*n$ size table and to get every element in the table, it need $O(1)$.

4 QUESTION 4

For each renting, $m(i,j)$ =total minimum total rental cost from post i to post j. $f(i,j)$=renting fees from post i to post j. So we can get the recurrence:

$$m(i,j) = Min\{f(i,j), m(i,k)+m(k+1,j)\}, i \le j-1, k = i+1, ..., j-2$$

Algorithm 3 total minimum total rental cost

Require: $i \le j-1, k = i+1, ..., j-2$
  let $m[1..n, 1..n]$ and $s[1..n-1, 2..n]$ be new tables
  for $i \in [1, n-1]$ **do**
    $m[i, i+1] = f[i, i+1]$
  end for
  for $l \in [3, n]$ **do**
    for $i \in [1, n-l+1]$ **do**
      $j = i+l-1$
      $m[i,j] = f[i,j]$
      for $k \in [i+1, j-1]$ **do**
        $q = m[i,k] + m[k,j]$
        if $q < m[i,j]$ **then**
          $m[i,j] = q$
          $s[i,j] = k$
        end if
      end for
    end for
  end for

To prove this is correct, consider the solution for $m[1,n]$,There are two cases: either $j = i+1$ or $j > i+1$.

Case 1: if $j = i+1$, it is easy to get that $m[i,i+1] = f[i,i+1]$ because there are only one way to get from i to j.
Case 2: if $j > i+1$, we need to use $m(i,j) = Min\{f(i,j), m(i,k)+m(k+1,j)\}, i \le j-1, k = i+1, ..., j-2$ to find the min value, which means we search from beginning to end. For example, we want to get the min value of $m[1,4]$, so we will compare the value of $f[1,4]$(means directly go to the end without stop), $(m[1,2]+m[2,4])$ and $(m[1,3]+m[3,4])$ to find the min value from those three values; And we already calculate the value of $m[1,2], m[2,4], m[1,3], m[3,4]$ in formal steps and remember them. So we can get the min value of $m[1,4]$ now
For the time complexity, this algorithm takes $O(n^3)$ because it has three layer loops. In first layer loop, the l has $n-3+1$ different values. In the second layer loop, the i has $n-l+1$ different values. In the third layer loop, the k has $j-i-1$ different values. So totally this algorithm takes $O(n^3)$ time complexity. What's more, this algorithm also take $O(n^2)$ space complexity.

5 QUESTION 5

To solve this problem, we need first sort all points according to their x coordinate in increasing order, where 1 denotes the starting point which has the smallest x and n is one of the turning point which has the smallest x coordinate . Then in the 4-tonic tour, we have a start point and three tuning point. This includes a n and two other points. We call those two points a and b. and we assume that $x_a < x_b$,now we have four paths in this 4-tonic tour:

$$P1 : [n \leftarrow 1]$$

$$P1 : [n \leftarrow a]$$

$$P1 : [b \leftarrow a]$$

$$P1 : [b \leftarrow 1]$$

then we create a 4-dimensional table to do our algorithms:
a new notation $L(i,j,p,q)$, in which i is the farthest point on P1, j is the farthest point on P4, p is the farthest point on P2, and q is the farthest point on P3. this $L(i,j,p,q)$ means the current minimum length in every path. To get the answer , we need to update the table like this.
denote r is the next point and initialize the $L(1,1,a,a) = 0$
Case 1: if $1 < r < a$:

$$r = 1 + \max(i,j)$$

$$L(r,j,a,a) = \min L(i,j,a,a) + d(r,i), L(r,j,a,a)$$

$$L(i,r,a,a) = \min L(i,j,a,a) + d(r,j), L(i,r,a,a)$$

Case 2: if $a < r < b$:

$$r = 1 + max(i, j, p, q)$$

$$L(r, j, p, q) = \min L(i, j, p, q) + d(r, i), L(r, j, p, q)$$

$$L(i, r, p, q) = \min L(i, j, p, q) + d(r, j), L(i, r, p, q)$$

$$L(i, j, r, q) = \min L(i, j, p, q) + d(p, r), L(i, j, r, q)$$

$$L(i, j, p, r) = \min L(i, j, p, q) + d(q, r), L(i, j, p, r)$$

Case 3: if $r = b$:

$$L(i, b, p, b) = \min L(i, j, p, q) + d(b, j) + d(b, q), L(i, b, p, b)$$

Case 4: if $n > r > b$:

$$r = 1 + maxi, p$$

$$L(r, b, p, b) = \min L(i, b, p, b) + d(i, r)$$

$$L(i, b, r, b) = \min L(i, b, p, b) + d(r, p)$$

Case 5: if $r = n$:

$$L(n, b, n, b) = \min L(i, b, p, b) + d(n, i) + d(p, n)$$

For the correctness,because this algorithm traversal from beginning to end, and it uses the dynamic programming method to compute the best solution at every stage. because in every state it is a best solution. So this algorithm can get best shortest length at last. For the time complexity, this algorithm holds and updates a four dimension table and then it choose another a,b which is n^2. So the final complexity is n^7