Fundamental Algorithms, Dynamic Programming, HW - 7

Tulsi Jain

1. Rod Cutting

Algorithm 1 Rod Cutting

```
function MaximumRevenus(p, n)                    ▷ where p price, n length
    int[] r = new int[n + 1];
    r[0] = 0;
    int i = 1;
    while i <= n do
        int j = 1;
        max = -∞;
        while j <= i do
            if i < j   And   p[j] + r[i − j] − c > max then
                max = p[j] + r[i-j] -c;
            else if i == j   And   p[j] > max then
                max = p[j];
            end if
            j = j + 1;
        end while
        r[i] = max;
        i = i + 1;
    end while
    return r[n];
end function
```

2. Coins with Denominations

Assumption c is sorted in increasing order and sum of all number upto n exists with given coins

Algorithm 2 CoinCount

```
function CoinCount(c, n, k)                    ▷ where c coins, k coinsNumber, n changeRequired
    int[] minCoins = new int[n + 1];
    int[] coinsType = new int[n + 1];
    minCoins[0] = 0;
    coinType[0] = 0;
    For all i form 0 to k-1 count[0][i] = 0;
    int i = 1;
    while i <= n do
        int j = 1;
        min = i;
        while j <= k   and   c[j] <= i do
            temp1 = i - c[j];
            temp2 = 1 + minCoins[temp1];
            if temp2 > min then
                min = temp2;
                coinsType[i] = c[j];
            end if
            j = j + 1;
        end while
        minCoins[i] = min;
        i = i + 1;
    end while
    return minCoins[n], coinType;
end function
```

Algorithm 3 CoinUsed

```
function CoinUsed(c, n, k)                     ▷ where w coinType, n changeRequired
    int[] coinsused;
    while n > 0 do
        coinsused.add( n - w[n]);
        n = n - w[n];
    end while
    return coinsused;
end function
```

Running time of complete algo is O(nk) as asked in question. CoinCount running time is O(nk) and CoinUsed running time is linear.

3. Neatly Printing

3.1

$$extras[i, j] = M - (\sum_{k=i}^{j} l_k + j - i)$$

Algorithm 4 ExtraSpace

```
function ExtraSpace(M, l, n)          ▷ where M max, l wordlength, n totalWord
    int[][] extras = new int[n][n];
    for i = 1 to n do
        extras[i, i] = M - l_i;
        for j = i + 1 to n do
            extras[i, j] = extras[i, j -1] - l_i - 1
        end for
    end for
    return extras;
end function
```

3.2

$$l_c[i,j] = \begin{cases} M - (\sum_{k=i}^{j} l_k + j - i) & \text{for all lines except last line} \\ \infty & \text{if element does not fit, } M - (\sum_{k=i}^{j} l_k + j - i) < 0 \\ 0 & \text{if } j = n \quad \text{last line and} \quad M - (\sum_{k=i}^{j} l_k + j - i) > 0 \end{cases}$$

Algorithm 5 LineCostExpression

```
function LineCostExpression(n, extras)          ▷ where n words, extras ExpraSpaces
    int[][] cost = new int[n][n];
    for i = 1 to n do
        for j = i to n do
            int extra = extras[i, j -1];
            if extra < 0 then
                cost[i, j] = ∞;
            else
                if j == n then
                    cost[i, j] = 0;
                else
                    cost[i, j] = extra ;
                end if
            end if
        end for
    end for
    return cost ;
end function
```

3.3

$c[0] = 0$
$c[1] = l_c[1, 1]$

$$c[j] = \begin{cases} 0 & \text{if j is 0} \\ \min \quad c[i-1] + l_c[i, j] \quad \text{for i from 1 to j} & \text{if j } > 0 \end{cases}$$

Algorithm 6 Cost

```
function COST(n, lc)                    ▷ where n words, lc linecost
    int cost = new int[n+1];
    cost[0] = 0;
    for i = 1 to n do
        cost[i] = ∞
        for j = 1 to i do
            if cost[j − 1] + lc[j, i] < cost[i] then
                cost[i] = cost[j -1] + lc[j, i];
            end if
        end for
    end for
    return cost[n];
end function
```

3.4

Algorithm 7 NeatlyPrinting

```
function NEATLYPRINTING(n, lc)                ▷ where n words, lc linecost
    int cost = new int[n+1];
    int firstWord = new int[n+1];
    cost[0] = 0;
    position[0] = 0;
    for i = 1 to n do
        cost[i] = ∞
        for j = 1 to i do
            if cost[j − 1] + lc[j, i] < cost[i] then
                cost[i] = cost[j -1] + lc[j, i];
                firstWord[i] = j;
                Comment: here j refer to the first word in the line i lies
            end if
        end for
    end for
    return cost[n];
end function
```

Algorithm 8 Line

```
function NEATLYPRINTING(fw, n)                ▷ where fw firstWord, n totalWords
    while n > 0 do
        print (fw[n], n);
        n = fw[n] - 1;
    end while
end function
```

Algorithms	Time Complexity	Space Complexity
ExtraSpace	$\Theta(n^2)$	$\Theta(n^2)$
LineCostExpression	$\Theta(n^2)$	$\Theta(n^2)$
Cost	$\Theta(n^2)$	$\Theta(n)$
Neatly Printing	$\Theta(n^2)$	$\Theta(n)$
Line	$\Theta(n)$	$\Theta(1)$
Full Algo	$\Theta(n^2)$	$\Theta(n^2)$

4. Edit Distance

This problem is similar to problem LCS. X and Y are given and target character array of size m and n respectively.
Let's say after performing a set of operation S which contains a last operation S_l and we have traverse first i characters of X and building a character array of Z. First j characters of Z is equal to first j characters of Y.
c[i,j] denotes the changing of first i characters of X into first j characters of Y. Possible choice of S_l is as follows:
Possible case:
Possible case 1: Last operation is copy, if x[i] != y[j] because if they are same replacing them would not yield anything

$$c[i,j] = c[i-1, j-1] + cost(copy)$$

Possible case 2: Last operation is replace, if x[i] != y[j] because if they are same replacing them would not yield anything

$$c[i,j] = c[i-1, j-1] + cost(replace)$$

Possible case 3: Last operation is delete

$$c[i,j] = c[i-1, j] + cost(delete)$$

Possible case 4: Last operation is insert

$$c[i,j] = c[i, j-1] + cost(insert)$$

Possible case 5: Last operation is twiddle, if x[i] != y[j], if x[i] == y[j - 1], if x[i -1] == y[j]

$$c[i,j] = c[i-2, j-2] + cost(twiddle)$$

Possible case 6: Last operation is kill If last step is kill then Y has change to X. So i is equal to m and n is equal to j = n

$$c[m,n] = \min_{1<=i<=n} c[m,i]) + cost(kill)$$

Cost[0,0] = 0, as converting a empty string into a empty string does not requires any operation.
Cost[1, 0] = 1*delete, as converting a empty string into a 1 character string requires 1 insertion.
Cost[2, 0] = 2*delete, as converting a empty string into a 1 character string requires 2 insertions.
And so on
Cost[n, 0] = n*delete, as converting a empty string into a n characters string requires n insertions.
Cost[0, 1] = 1*insert, as converting a single string character into a empty string requires 1 deletion.
Cost[0, 2] = 2*insert, as converting a single string character into a empty string requires 2 deletion.
Cost[0, 3] = 3*insert, as converting a single string character into a empty string requires 3 deletion.
And so on
Cost[0, n] = n*insert, as converting a single string character into a empty string requires n deletion.

$$c[i,j] = min \begin{cases} c[i-1, j-1] & \text{if x[i] == y[j]} \\ c[i-1, j-1] + cost(copy) & \text{if x[i] != y[j]} \\ c[i-1, j-1] + cost(replace) & \text{if x[i] != y[j]} \\ c[i, j-1] + cost(insert) & \text{if x[i] != y[j]} \\ c[i-1, j] + cost(delete) & \text{if x[i] != y[j]} \\ c[i-2, j-2] + cost(twiddle) & \text{if i, j >= 2, x[i] != y[j], x[i] == y[j - 1], x[i -1] == y[j]} \\ \min_{1<=i<=n} c[m, i]) + cost(kill) & \text{if i = m, j = n} \end{cases}$$

Space complexity is $\Theta(n^2)$ and time complexity is $\Theta(n^2)$. Algorithms is presented on next **two** pages.

Algorithm 9 EditDistance

```
function EditDistance(X, Y, m, n)                ▷ where X target, Y given, m , n
    int[][] cost = new int[m + 1, n + 1];
    int[][] op = new int[m + 1, n + 1];
    cost[0][0] = 0;
    for i = 1 to m do
        cost[i][0] = i*cost(delete);
    end for
    for i = 1 to n do
        cost[0][i] = i*cost(insert);
    end for
    for i = 1 to m do
        for i = 1 to n do
            cost[i][j] = ∞;
            if X[i] == Y[i] then
                cost[i][j] = cost[i - 1][j - 1] ;
            end if
            if X[i] != Y[i] and cost[i - 1][j - 1] + cost(copy) < cost[i][j] then
                cost[i][j] = cost[i - 1][j - 1] + cost(copy);
                op[i][j] = copy;
            end if
            if X[i] != Y[i] and cost[i - 1][j - 1] + cost(replace) < cost[i][j] then
                cost[i][j] = cost[i - 1][j - 1] + cost(replace);
                op[i][j] = replace;
            end if
            if i,j >= 2 and X[i -1] == Y[j] and X[i] == Y[j - 1] and cost[i - 2][j - 2] + cost(twiddle)
< cost[i][j] then
                cost[i][j] = cost[i - 2][j - 2] + cost(twiddle);
                op[i][j] = twiddle;
            end if
            if cost[i - 1][j] + cost(delete) < cost[i][j] then
                cost[i][j] = cost[i - 1][j] + cost(delete);
                op[i][j] = delete;
            end if
            if cost[i][j - 1] + cost(insert) < cost[i][j - 1] then
                cost[i][j] = cost[i - 1][j - 1] + cost(insert);
                op[i][j] = insert;
            end if
        end for
    end for
    for i = 1 to m do
        if cost[m][n] < cost[i][n] + cost(kill) then
            cost[m][n] = cost[i][n] + cost(kill);
        end if
    end for
    return cost, op;
end function
```

Algorithm 10 PrintOps

```
function PrintOps(ops, i, j)                    ▷ where ops PrintOps, i, j
    Stack st
    while i! = 0||j! = 0 do
        int a;
        int b;
        if ops[i][j] == copy) then
            a = i - 1;
            b = j - 1;
        end if
        if ops[i][j] == replace) then
            a = i - 1;
            b = j - 1;
        end if
        if ops[i][j] == insert) then
            a = i;
            b = j - 1;
        end if
        if ops[i][j] == delete) then
            a = i - 1;
            b = j;
        end if
        if ops[i][j] == twiddle) then
            a = i - 2;
            b = j - 1;
        end if
        if ops[i][j] == kill) then
            Comment: means i is m and j is n
            a = k;
            b = j;
        end if
        st.push(op[i, j])
        i = a;
        j = b;
    end while
    while st.size() > 0 do
        print(st.pop());
    end while
end function
```