Introduction to Algorithms HW6

109511207 蔡宗儒

Pseudo Code

```
LCS-LENGTH(X,Y)
m = X.length
n = Y.length
let b[1…m,1…n] and c[0…m,0…n] be new tables
for i = 1 to m
    c[i,0] = 0
for j = 0 to n
    c[0,j] = 0
for i = 1 to n
    for j = 1 to n
        if x_i == y_j
            c[i,j] = c[i-1,j-1] + 1
            b[i,j] = "↖"
        else if c[i-1,j] >= c[i,j-1]
            c[i,j] = c[i-1,j]
            b[i,j] = " ↑ "
        else c[i,j] = c[i,j-1]
            b[i,j] = "←"
return c and b
```

```
PRINT-LCS.(b,X,i,j)
if i == 0 or j == 0
    return
if b[i, j] == "↖"
    PRINT-LCS(b,X,i-1,j-1)
    print x_i
else if b[i,j] == " ↑ "
    PRINT-LCS(b,X,i-1,j)
else PRINT-LCS(b,X,i,j-1)
```

Introduction of Code

```
void LCS_Length(vector<double> X, vector<double> Y, vector<double> Z, vector<vector<vector<int>>> &C)
{
    int m = X.size();
    int n = Y.size();
    int p = Z.size();

    for(int i=1;i<=m;i++)
    {
        for(int j=1;j<=n;j++)
        {
            for(int k=1;k<=p;k++)
            {
                if(X[i-1] == Y[j-1] && X[i-1] == Z[k-1])
                    C[i][j][k] = C[i-1][j-1][k-1] + 1;
                else
                    C[i][j][k] = max(max(C[i-1][j][k],C[i][j-1][k]),C[i][j][k-1]);
            }
        }
    }
}
```

LCS_Length 可以建立 c table 並得出 LCS 的 length。

```
void findAllLCS(vector<vector<vector<int>>> C, vector<double> X, vector<double> Y, vector<double> Z, int i, int j, int k, vector<double> &cLCS, vector<vector<double>> &allLCS)
{
    // if the end of the sequence is reached, then we can return
    if(i == 0 || j == 0 || k == 0)
    {
        reverse(cLCS.begin(),cLCS.end());
        allLCS.push_back(cLCS);
        reverse(cLCS.begin(),cLCS.end());
        return;
    }

    if (X[i-1] == Y[j-1] && X[i-1] == Z[k-1])
    {
        cLCS.push_back(X[i-1]);
        findAllLCS(C,X,Y,Z,i-1,j-1,k-1,cLCS,allLCS);
        cLCS.pop_back();
    }

    else
    {
        // go to direct X
        if (C[i-1][j][k] >= C[i][j-1][k] && C[i-1][j][k] >= C[i][j][k-1])
            findAllLCS(C,X,Y,Z,i-1,j,k,cLCS,allLCS);

        // go to direct Y
        if (C[i][j-1][k] >= C[i-1][j][k] && C[i][j-1][k] >= C[i][j][k-1])
            findAllLCS(C,X,Y,Z,i,j-1,k,cLCS,allLCS);

        // go to direct Z
        if (C[i][j][k-1] >= C[i-1][j][k] && C[i][j][k-1] >= C[i][j-1][k])
            findAllLCS(C,X,Y,Z,i,j,k-1,cLCS,allLCS);
    }
}
```

findAllLCS，可以藉由 c table 的數值，一步一步遍歷回原點，並找出所有可能的 LCS。

執行結果

```
Input size(1-100) of Sequence X: 15
Input size(1-100) of Sequence Y: 15
Input size(1-100) of Sequence Z: 15
Input Sequence X: 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
Input Sequence Y: 1 3 5 2 4 2 4 5 3 1 1 2 3 4 5
Input Sequence Z: 5 3 1 2 4 3 4 5 1 2 1 2 3 4 5
LCS length is: 10
LCS is:
1 2 4 4 5 1 2 3 4 5
1 2 4 5 1 1 2 3 4 5
1 3 4 5 1 1 2 3 4 5
```

```
Input size(1-100) of Sequence X: 15
Input size(1-100) of Sequence Y: 15
Input size(1-100) of Sequence Z: 15
Input Sequence X: 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
Input Sequence Y: 5 3 1 2 4 2 4 1 3 5 1 2 3 4 5
Input Sequence Z: 3 5 4 1 2 2 5 3 1 4 1 2 3 4 5
LCS length is: 9
LCS is:
1 2 2 3 1 2 3 4 5
1 2 2 4 1 2 3 4 5
1 2 2 5 1 2 3 4 5
3 1 2 3 1 2 3 4 5
3 1 2 4 1 2 3 4 5
3 1 2 5 1 2 3 4 5
3 4 1 3 1 2 3 4 5
3 4 1 5 1 2 3 4 5
3 4 2 3 1 2 3 4 5
3 4 2 4 1 2 3 4 5
3 4 2 5 1 2 3 4 5
5 1 2 3 1 2 3 4 5
5 1 2 4 1 2 3 4 5
5 1 2 5 1 2 3 4 5
```

```
Input size(1-100) of Sequence X: 20
Input size(1-100) of Sequence Y: 20
Input size(1-100) of Sequence Z: 20
Input Sequence X: 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
Input Sequence Y: 1 3 5 7 9 2 4 6 8 0 9 6 3 0 1 4 7 2 5 8
Input Sequence Z: 2 4 6 8 0 1 3 5 7 9 2 6 8 4 1 7 5 9 3 0
LCS length is: 9
LCS is:
1 3 5 7 9 2 6 8 9
```

```
Input size(1-100) of Sequence X: 50
Input size(1-100) of Sequence Y: 50
Input size(1-100) of Sequence Z: 50
Input Sequence X: 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
Input Sequence Y: 2 1 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
Input Sequence Z: 3 2 1 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
LCS length is: 48
LCS is:
1 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
2 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
```

```
Input size(1-100) of Sequence X: 100
Input size(1-100) of Sequence Y: 100
Input size(1-100) of Sequence Z: 100
Input Sequence X: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 6
0 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
Input Sequence Y: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 0
Input Sequence Z: 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 0 1
LCS length is: 98
LCS is:
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67
 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
```

由執行結果可看出，只要 LCS 的長度在一定的長度之上，input sequence 的 size 是可以達到 100 的。

Analysis

對於窮舉法來說，假設 Sequence X 的長度為 m，Sequence Y 的長度為 n，Sequence Z 的長度為 p，當要取 subsequence 時，每個元素都有選或不選兩種選擇，所以 Sequence X 總共有 2^m 種 subsequence，Sequence Y 總共有 2^n 種 subsequence，Sequence Z 總共有 2^p 種 subsequence，用窮舉法配對的話總共有 $2^m*2^n*2^p = 2^{m+n+p}$ 配對，所以時間複雜度為 $\Theta(2^{m+n+p})$。

而對於老師上課(及課本上)所講的使用 Dynamic programming 的演算法來說，可以利用以下的二維的 optimal structure 來大致構想三維的 optimal structure

Theorem 15.1 (Optimal substructure of an LCS)

Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y.

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.

所以，X, Y, Z 的 LCS 可以根據 x_m 是否等於 y_n 和 z_p 得知，如下。

1. If $x_m = y_n = z_p$, then $w_q = x_m = y_n = z_p$ and Wq-1 is an LCS of Xm-1 and Yn-1.
2. If not($x_m = y_n = z_p$), then W is an LCS of (Xm-1 and Y and Z) or (X and Yn-1 and Z) or (X and Y and Zp-1).

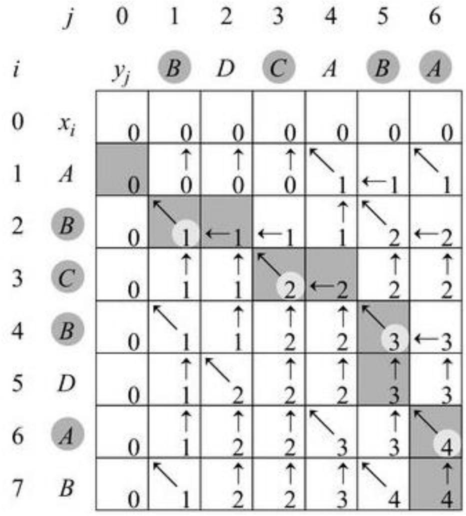
因此我們可以把各 i, j, k (where i <=m, j<=n, k<=p)所產生的 LCS_Length 存入一個 3D 的 table C，關係式如下。

1. C[i,j,k] = 0 , if i = 0 or j = 0 or k = 0
2. C[i,j,k] = C[i-1,j-1,k-1] + 1 , if i, j, k> 0 and $x_i = y_j = z_k$
3. C[i,j,k] = max(C[i-1,j,k], C[i,j-1,k], C[i,j,k-1]), if i, j, k> 0 and not($x_i = y_j = z_k$)

所以構建 table C，產生 LCS 的 length 所需的時間複雜度應為 Θ(m*n*p)，如下我所寫的程式碼，可看出三層迴圈時間複雜度應為 Θ(m*n*p)。

```
void LCS_Length(vector<double> X, vector<double> Y, vector<double> Z, vector<vector<vector<int>>> &C)
{
    int m = X.size();
    int n = Y.size();
    int p = Z.size();
    for(int i=1;i<=m;i++)
    {
        for(int j=1;j<=n;j++)
        {
            for(int k=1;k<=p;k++)
            {
                if(X[i-1] == Y[j-1] && X[i-1] == Z[k-1])
                    C[i][j][k] = C[i-1][j-1][k-1] + 1;
                else
                    C[i][j][k] = max(max(C[i-1][j][k],C[i][j-1][k]),C[i][j][k-1]);
            }
        }
    }
}
```

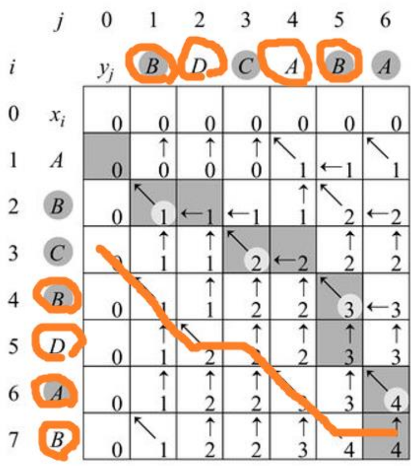
而若要產生 LCS 的解的話，課本跟講義給的 pseudo code 是用了一個 b table，依照箭頭的指示遍歷 table，用這種方法可以以相反的順序遇到 LCS 的所有 elements。這個過程只會花 O(m+n+p)的時間，因為每次 recursive call 中，i、j、k 至少會有一個變數少 1，所以最多走 m+n+p 步就會走回原點(0,0,0)，所以複雜度為 O(m+n+p)，如下為課本二維的 b table 的圖，三維的概念則類似。



遇到的問題與想法

這次作業我遇到的問題便是要輸出全部的 LCS，除了看了課本的 pseudo code 以外，我也參考了網路上的一些做法，但網路上只看到輸出兩個 Sequence 全部的 LCS 或是僅輸出三個 Sequence 的其中一個 LCS 解，並沒有看到有人做出可以輸出三個 Sequence 全部的 LCS 的實作。

於是我便將課本二維輸出全部 LCS 的做法自己修成三維的做法，要注意因為需要能找到全部的 LCS，按照課本原本的做法會少考慮很多 Case，以下面課本的圖為例，當 b table back-trace 時，當他遇到可以同時往左跟往上走的時候，也要考慮往左走的情況，如此一來可以找到有一個 LCS 是 BDAB，而在三維的 b table 的情況下也就三個方向都要考慮。



然後我又將 b table 刪掉，僅用 c table 的數值判定要往哪些方向走。我的程式碼如下。

```cpp
void findAllLCS(vector<vector<vector<int>>> C, vector<double> X, vector<double> Y, vector<double> Z, int i, int j, int k, vector<double> &cLCS, vector<vector<double>> &allLCS)
{
    // if the end of the sequence is reached, then we can return
    if(i == 0 || j == 0 || k == 0)
    {
        reverse(cLCS.begin(),cLCS.end());
        allLCS.push_back(cLCS);
        reverse(cLCS.begin(),cLCS.end());
        return;
    }

    if (X[i-1] == Y[j-1] && X[i-1] == Z[k-1])
    {
        cLCS.push_back(X[i-1]);
        findAllLCS(C,X,Y,Z,i-1,j-1,k-1,cLCS,allLCS);
        cLCS.pop_back();
    }

    else
    {
        // go to direct X
        if (C[i-1][j][k] >= C[i][j-1][k] && C[i-1][j][k] >= C[i][j][k-1])
            findAllLCS(C,X,Y,Z,i-1,j,k,cLCS,allLCS);

        // go to direct Y
        if (C[i][j-1][k] >= C[i-1][j][k] && C[i][j-1][k] >= C[i][j][k-1])
            findAllLCS(C,X,Y,Z,i,j-1,k,cLCS,allLCS);

        // go to direct Z
        if (C[i][j][k-1] >= C[i-1][j][k] && C[i][j][k-1] >= C[i][j-1][k])
            findAllLCS(C,X,Y,Z,i,j,k-1,cLCS,allLCS);
    }
}
```

但這麼做的話，在三維的情況下，可能會要走同時走 X 方向、Y 方向、Z 方向的其中兩個方向甚至三個方向都走，用 recursive back trace 回去會花超級多時間，這種情況會發生在越少直接往左上的方向走的 table 下(即 LCS 長度較短或是 Sequence Size 和 LCS 長度的比值較大時)，這種情況下我僅能輸出 LCS 的長度，當要輸出 LCS 所有的解時會無法跑完，因為要不斷地找尋 X 方向、Y 方向和 Z 方向的解。我自己用比較小的 Case 時(Sequence X 長度為 15，Sequence Y 長度為 15，Sequence Z 長度為 2，LCS 長度為 2)，跑了 50 分鐘才出來，再把 Size 跟 LCS 的長度比值拉更大、更極端的話，即便跑好幾個小時也跑不出來。若是在 Size 和 LCS 長度比值較小時我的程式碼便可以正常輸出了，而對於這個問題，即便多了一個禮拜的時間，我依然沒有找到解決的方式，我認為若是用類似課本的演算法的話，這個問題是無法避免的，recursive 就是需要跑這麼多次，或許要想想不用 recursive 的方法便可以完整遍歷回去?!

也於是我在這次作業交了兩份 Code，一份為 10951127_蔡宗儒_HW6.cpp，這份是可以找所有的 LCS 的，但可能會跑很久或是跑不出來；另一份是 10951127_蔡宗儒_HW6_v2.cpp，這是只找一個 LCS 的解的。