

Homework 6

0811324 胡政煒

窮舉法(用二維舉例):

若沒有使用動態規劃或其他優化方法，而是使用窮舉法（也稱為暴力法）來解決最長共同子序列（LCS）問題，其時間複雜度將會非常高。

LCS 問題的窮舉法解法是列舉出所有可能的子序列，並逐一比較它們是否為兩個序列的共同子序列，然後找到其中最長的一個。假設兩個序列的長度分別為 m 和 n ，其中 $m \leq n$ 。接下來我們來計算使用窮舉法解決 LCS 問題的時間複雜度。

對於每個可能的子序列，我們需要檢查它是否為兩個序列的共同子序列。這需要進行兩個序列的遍歷，並且對於每個元素的比較，需要花費常數時間。因此，檢查一個子序列是否為共同子序列的時間複雜度為 $O(m + n)$ 。

由於有 2^m 個可能的子序列，我們需要對每個子序列進行檢查。因此，窮舉法的時間複雜度為 $O(2^m * (m + n))$ 。

現在舉個例子來說明窮舉法的時間複雜度。假設我們有兩個序列 'str1' 和 'str2'，分別為 "ABCDEF" 和 "XYZ"。

首先，來計算可能的子序列數量。對於 'str1'，長度為 6，有 $2^6 = 64$ 個可能的子序列。對於 'str2'，長度為 3，有 $2^3 = 8$ 個可能的子序列。

現在，需要對每個子序列進行檢查，並檢查它是否同時是 'str1' 和 'str2' 的子序列。對於每個子序列，需要進行兩個序列的遍歷，並進行元素的比較。因此，每個子序列的檢查需要花費 $O(m + n) = O(6 + 3) = O(9)$ 的時間。

總共有 $64 * 8 = 512$ 個子序列需要檢查，因此，窮舉法的時間複雜度為 $O(2^m * (m + n)) = O(2^6 * 9) = O(512 * 9) = O(4608)$ 。

從上述例子可以看出，使用窮舉法來解決 LCS 問題在較大的序列上會變得非常耗時，因為其時間複雜度是指數級增長的。這就是為什麼動態規劃等優化方法變得重要，因為它們可以在多項式時間內解決 LCS 問題，提高算法的效率。

Longest Common Subsequence (LCS) (二維舉例):

Longest Common Subsequence (LCS) 的時間複雜度可以通過使用動態規劃來改善。動態規劃利用先前計算的結果，以避免重複的計算，從而提高效率。

動態規劃解決 LCS 問題的基本思想是建立一個二維的表格，將兩個序列的每個元素進行比較，並填充表格。表格中的每個元素代表了兩個序列的部分匹配長度。通過填充表格，我們可以找到最長共同子序列的長度。

以下是動態規劃改善 LCS 問題時間複雜度的步驟：

1. 創建一個二維的表格，大小為 $(m+1) \times (n+1)$ ，其中 m 和 n 分別為兩個序列的長度。
2. 初始化表格的第一行和第一列為 0，表示空序列和任何序列的最長共同子序列長度為 0。
3. 遍歷表格的每個元素，從左上角開始。對於每個元素 (i, j) ，如果序列 1 的第 i 個元素和序列 2 的第 j 個元素相等，則該元素的值等於左上角元素的值加 1；否則，該元素的值等於左方元素和上方元素中的最大值。
4. 遍歷完成後，表格右下角的元素即為最長共同子序列的長度。
5. 可以根據填充表格的過程回溯，從右下角開始，根據表格中的數值和相鄰元素的關係，構建出最長共同子序列。

下面以例子來說明動態規劃改善 LCS 問題時間複雜度的過程。假設我們有兩個序列 'str1' 和 'str2'，分別為 "ABCD" 和 "ACDF"。

1. 創建一個 5x5 的表格，初始化第一行和第一列為 0：

	0	1	2	3	4
		A	C	D	F
0	X 0	X 0	X 0	X 0	X 0
1 A	X 0				
2 B	X 0				
3 C	X 0				
4 D	X 0				

2. 填充表格的過程如下：

$s1[1] == s2[1]$ ：以左上方之 LCS 長度設為此格的 LCS 長度。

$s1[1] != s2[2]$ ：以左方或上方之 LCS 中長度較高者設為此格的 LCS 長度。
(此處選擇為 '左' 方)

$s1[1] != s2[3]$ ：以左方或上方之 LCS 中長度較高者設為此格的 LCS 長度。
(此處選擇為 '左' 方)

$s1[1] != s2[4]$ ：以左方或上方之 LCS 中長度較高者設為此格的 LCS 長度。
(此處選擇為 '左' 方)

$s1[2] != s2[1]$ ：以左方或上方之 LCS 中長度較高者設為此格的 LCS 長度。
(此處選擇為 '上' 方)

$s1[2] != s2[2]$ ：以左方或上方之 LCS 中長度較高者設為此格的 LCS 長度。
(此處選擇為 '上' 方)

$s1[2] != s2[3]$ ：以左方或上方之 LCS 中長度較高者設為此格的 LCS 長度。
(此處選擇為 '上' 方)

$s1[2] != s2[4]$ ：以左方或上方之 LCS 中長度較高者設為此格的 LCS 長度。
(此處選擇為 '上' 方)

$s1[3] != s2[1]$ ：以左方或上方之 LCS 中長度較高者設為此格的 LCS 長度。
(此處選擇為 '上' 方)

$s1[3] == s2[2]$ ：以左上方之 LCS 長度設為此格的 LCS 長度。

$s1[3] != s2[3]$ ：以左方或上方之 LCS 中長度較高者設為此格的 LCS 長度。
(此處選擇為 '左' 方)

$s1[3] != s2[4]$ ：以左方或上方之 LCS 中長度較高者設為此格的 LCS 長度。
(此處選擇為 '左' 方)

$s1[4] != s2[1]$ ：以左方或上方之 LCS 中長度較高者設為此格的 LCS 長度。
(此處選擇為 '上' 方)

$s1[4] != s2[2]$ ：以左方或上方之 LCS 中長度較高者設為此格的 LCS 長度。
(此處選擇為 '上' 方)

$s1[4] == s2[3]$ ：以左上方之 LCS 長度設為此格的 LCS 長度。

$s1[4] != s2[4]$ ：以左方或上方之 LCS 中長度較高者設為此格的 LCS 長度。
(此處選擇為 '左' 方)

填表完成，開始取得 LCS。

完成！LCS 為：ACD。

3. 遍歷完成後，表格的狀態如下：

		0	1	2	3	4
		A C D F				
0		X 0	X 0	X 0	X 0	X 0
1	A	X 0	↖ 1	← 1	← 1	← 1
2	B	X 0	↑ 1	↑ 1	↑ 1	↑ 1
3	C	X 0	↑ 1	↖ 2	← 2	← 2
4	D	X 0	↑ 1	↑ 2	↖ 3	← 3

4. 表格右下角的元素的值為 3，表示最長共同子序列的長度為 3。

5. 通過回溯填充表格的過程，我們可以構建出最長共同子序列。從右下角開始，根據表格中的數值和相鄰元素的關係，我們可以得到一個最長共同子序列為“ACD”。

這個例子展示了動態規劃如何改善 LCS 問題的時間複雜度。使用動態規劃，我們只需填充一個二維表格一次，而不需要遞迴地比較所有可能的子序列。這大大減少了計算的時間複雜度，提高了算法的效率。

參考資料：<https://web.ntnu.edu.tw/~algo/Subsequence2.html>

作業程式碼：

```
1  #include <iostream>
2  #include <vector>
3  #include <set>
4  #include <algorithm>
5  #include <ctime>
6  using namespace std;
7
8  void generateLCS(vector<double>& X, vector<double>& Y, vector<double>& Z, int i, int j, int k,
9      vector<vector<vector<int>>>& lookup, vector<double>& currentLCS, set<vector<double>>& result);
10
11 set<vector<double>> findLCS(vector<double>& X, vector<double>& Y, vector<double>& Z);
12
13 // Function to find all longest common subsequences of three vectors
14 set<vector<double>> findLCS(vector<double>& X, vector<double>& Y, vector<double>& Z)
15 {
16     int m = X.size();
17     int n = Y.size();
18     int p = Z.size();
19
20     // Create a 3D lookup table to store the length of LCS
21     vector<vector<vector<int>>> lookup(m + 1, vector<vector<int>>(n + 1, vector<int>(p + 1, 0)));
22
23     // Compute the length of LCS using bottom-up dynamic programming approach
24     for (int i = 1; i <= m; i++) {
25         for (int j = 1; j <= n; j++) {
26             for (int k = 1; k <= p; k++) {
27                 if (X[i - 1] == Y[j - 1] && Y[j - 1] == Z[k - 1]) {
28                     lookup[i][j][k] = lookup[i - 1][j - 1][k - 1] + 1;
29                 }
30                 else {
31                     lookup[i][j][k] = max(max(lookup[i - 1][j][k], lookup[i][j - 1][k]), lookup[i][j][k - 1]);
32                 }
33             }
34         }
35     }
36
37     // Backtrack and generate all LCS subsequences
38     set<vector<double>> lcsSet;
39     vector<double> currentLCS;
40     generateLCS(X, Y, Z, m, n, p, lookup, currentLCS, lcsSet);
41
42     return lcsSet;
43 }
```

此函數 findLCS 將三個向量 X、Y 和 Z 作為輸入，並使用維度 $(m + 1) \times (n + 1) \times (p + 1)$ 初始化查找表 lookup，其中 m、n 和 p 是大小分別為向量 X、Y 和 Z。查找表將存儲 LCS 的長度。

此嵌套循環結構使用自下而上的動態規劃為所有可能的子問題計算 LCS 的長度。它遍歷索引 i、j 和 k，分別表示向量 X、Y 和 Z 中的位置。如果這些位置的元素相等，它會根據先前位置的 LCS 長度加一來更新查找表。否則，它從前三個位置獲取最大 LCS 長度。

在計算查找表後，此函數初始化一個空集 lcsSet 以存儲所有生成的 LCS 子序列。它還初始化一個空向量 currentLCS 以保存正在生成的當前 LCS。然後，它調用 generateLCS 函數，傳遞向量 X、Y 和 Z、維度 m、n 和 p、查找表 lookup、當前 LCS currentLCS 和集合 lcsSet。最後，它返回包含所有 LCS 子序列的集合 lcsSet。

```

// Function to backtrack and generate all LCS subsequences
void generateLCS(vector<double>& X, vector<double>& Y, vector<double>& Z, int i, int j, int k,
vector<vector<vector<int>>>& lookup, vector<double>& currentLCS, set<vector<double>>& result)
{
    // If we have reached the end of any vector, add the current LCS subsequence to the result
    if (i == 0 || j == 0 || k == 0) {
        result.insert(currentLCS);
        return;
    }

    // If the current elements in all three vectors are equal, include it in the current LCS subsequence
    // and move to the previous indices in all vectors
    if (X[i - 1] == Y[j - 1] && Y[j - 1] == Z[k - 1]) {
        currentLCS.insert(currentLCS.begin(), X[i - 1]);
        generateLCS(X, Y, Z, i - 1, j - 1, k - 1, lookup, currentLCS, result);
        currentLCS.erase(currentLCS.begin());
    }

    // If the value in the lookup table for the current position came from the top cell, move up
    if (i > 0 && lookup[i][j][k] == lookup[i - 1][j][k]) {
        generateLCS(X, Y, Z, i - 1, j, k, lookup, currentLCS, result);
    }

    // If the value in the lookup table for the current position came from the left cell, move left
    if (j > 0 && lookup[i][j][k] == lookup[i][j - 1][k]) {
        generateLCS(X, Y, Z, i, j - 1, k, lookup, currentLCS, result);
    }

    // If the value in the lookup table for the current position came from the top-left cell, move diagonally
    if (k > 0 && lookup[i][j][k] == lookup[i][j][k - 1]) {
        generateLCS(X, Y, Z, i, j, k - 1, lookup, currentLCS, result);
    }
}

```

提供的代碼實現了 generateLCS 函數，該函數負責根據查找表回溯並生成所有 LCS（最長公共子序列）子序列。

此函數採用三個向量 X、Y 和 Z，以及它們的當前索引分別為 i、j 和 k。它還採用查找表檢查，當前 LCS 被生成 currentLCS，set 結果存儲生成的 LCS 子序列。

第一個 IF:

此條件檢查索引 i、j 或 k 中的任何一個是否已到達末尾（即到達 0）。如果是，則意味著我們已到達至少一個向量的末尾，並且當前 LCS 子序列已完成。它將當前的 LCS 子序列插入到結果集中並返回。

第二個 IF:

此條件檢查所有三個向量 X、Y 和 Z 中的當前元素是否相等。如果它們相等，則意味著當前元素是 LCS 子序列的一部分。它將當前元素添加到 currentLCS 向量的開頭，並使用所有三個向量 (i - 1、j - 1、k - 1) 中的先前索引遞歸調用 generateLCS。這樣做是為了通過回溯生成 LCS 子序列。在遞歸調用之後，它從 currentLCS 向量中刪除添加的元素。

第三個 IF:

此條件檢查當前位置的查找表中的值是否來自頂部單元格。如果該值等於上面單元格中的值 ($\text{lookup}[i - 1][j][k]$)，則表示 LCS 長度沒有因包含向量 X 中的當前元素而改變。因此，它遞歸調用 `generateLCS` 索引 i 減 1，其他索引不變。

第四個 IF:

此條件檢查當前位置的查找表中的值是否來自左側單元格。如果該值等於左側單元格中的值 ($\text{lookup}[i][j - 1][k]$)，則表示 LCS 長度沒有因包含向量 Y 中的當前元素而改變。因此，它遞歸調用 `generateLCS`，索引 j 減 1，其他索引不變。

第五個 IF:

此條件檢查當前位置的查找表中的值是否來自左上角的單元格。如果該值等於左上角單元格中的值 ($\text{lookup}[i][j][k - 1]$)，則表示 LCS 長度沒有因包含向量 Z 中的當前元素而改變。因此，它遞歸調用 `generateLCS`，索引 k 減 1，其他索引不變。

`generateLCS` 函數根據上述條件遞歸地探索查找表中的所有可能路徑。它通過包含或排除三個向量中的元素來回溯並生成所有 LCS 子序列。生成的 LCS 子序列存儲在結果集中，然後返回。

在主程式中

```
// // Read the length and elements of sequence X from the user
// cout << "Enter the length of sequence X: ";
// cin >> lx;
// cout << "Enter the elements of sequence X: ";
// for (int i=0; i<lx; i++) {
//     cin >> k;
//     X.push_back(k);
// }

// // Read the length and elements of sequence Y from the user
// cout << "Enter the length of sequence Y: ";
// cin >> ly;
// cout << "Enter the elements of sequence Y: ";
// for (int i=0; i<ly; i++) {
//     cin >> k;
//     Y.push_back(k);
// }

// // Read the length and elements of sequence Z from the user
// cout << "Enter the length of sequence Z: ";
// cin >> lz;
// cout << "Enter the elements of sequence Z: ";
// for (int i=0; i<lz; i++) {
//     cin >> k;
//     Z.push_back(k);
// }
```

這裡註解的部分可以拿來用手動輸入(輸入長度及數值)

下面部分則是我用 RANDOM 隨機產生數值的程式碼(輸入長度即可)

```
// Generate random elements for sequence X
cout << "Enter the length of sequence X: ";
cin >> lx;
cout << "Sequence X: ";
srand(time(0)); // seed the random number generator with current time
for (int i = 0; i < lx; i++) {
    double randomElement = rand() % 10; // generate random number between 0 and 9
    X.push_back(randomElement);
    cout << randomElement << " ";
}
cout << endl;

// Generate random elements for sequence Y
cout << "Enter the length of sequence Y: ";
cin >> ly;
cout << "Sequence Y: ";
for (int i = 0; i < ly; i++) {
    double randomElement = rand() % 10; // generate random number between 0 and 9
    Y.push_back(randomElement);
    cout << randomElement << " ";
}
cout << endl;

// Generate random elements for sequence Z
cout << "Enter the length of sequence Z: ";
cin >> lz;
cout << "Sequence Z: ";
for (int i = 0; i < lz; i++) {
    double randomElement = rand() % 10; // generate random number between 0 and 9
    Z.push_back(randomElement);
    cout << randomElement << " ";
}
cout << endl;

set<vector<double>> lcsSet = findLCS(X, Y, Z);

// Print all LCS subsequences
for (const vector<double>& lcs : lcsSet) {
    for (const double& num : lcs) {
        cout << num << " ";
    }
    cout << endl;
}
```

```
Enter the length of sequence X: 12
Sequence X: 2 3 2 3 5 1 2 9 8 7 1 3
Enter the length of sequence Y: 12
Sequence Y: 8 4 0 3 2 9 8 3 6 5 9 9
Enter the length of sequence Z: 12
Sequence Z: 3 6 3 1 1 4 7 9 6 6 6 3
3 3 9
3 9 3
```

最高只能跑到長度 12 左右，再高所需的時間就會非常多，才會有數據。