Homework #4

Due date: 4/12 lab

Longest Common Subsequence

Given two sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$, find a subsequence common to both whose length is longest.

Comments

1 A subsequence doesn't have to be consecutive, but it has to be in order.
2 In case there is more than one longest common subsequence (lcs), you may find any one of them.

Example

$X =$ **p i o n e e r** length of an lcs $= 4$

$Y =$ **s p r i n g b r e a k** lcs: **pine**, **pinr**

Example

$X =$ **2 5 3 1 6 4** length of an lcs $= 3$

$Y =$ **1 2 3 4 5 6** lcs: **234**, **236**, **256**

Dynamic programming solution

Let $c(i,j) =$ the length of an lcs of $\langle x_1, x_2, \ldots, x_i \rangle$ and $\langle y_1, y_2, \ldots, y_j \rangle$

We have

$$c(i,j) = \begin{cases} 0 & i = 0 \text{ or } j = 0 \\ c(i-1, j-1) + 1 & i, j > 0 \text{ and } x_i = y_j \\ \max\big(c(i, j-1), c(i-1, j)\big) & i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

where $0 \leq i \leq m$ and $0 \leq j \leq n$.

What does this recursive definition mean?

Well, if you can grasp its meaning. Bravo!
This is a classic problem on dynamic programming and will be taught next year in the course "Analysis of Algorithms".
For now, you may simply take it for granted and enjoy turning it into a program.

How to turn it into a program?

Since this problem has a similar dynamic programming solution to the combination problem given in the lecture, the same procedure applies:

Step 1: Create an $(m+1) \times (n+1)$ table tbl
Step 2: Fill in the table, making $tbl[i][j] = c(i,j)$ for all i,j
Step 3: Output the value of $tbl[m][n]$, which is the length of an lcs
Step 4: Output an lcs according to the two input sequences and the table tbl

Example

Let $X = \langle 2,5,3,1,6,4 \rangle$ and $Y = \langle 1,2,3,4,5,6 \rangle$.
The entries of the 7×7 table tbl and the three lcs's are shown below.

		y_j					
		1	2	3	4	5	6
	0	0	0	0	0	0	0
2	0	0	1	1	1	1	1
5	0	0	1	1	1	2	2
3	0	0	1	2	2	2	2
1	0	1	1	2	2	2	2
6	0	1	1	2	2	2	3
4	0	1	1	2	3	3	3

x_i

blue: 2 5 6

red: 2 3 6

green: 2 3 4

length: 3

Notice that there are three possible red-colored paths, all yielding the same lcs.

How to construct an lcs in step 4?

Look at x_i and y_j

Case 1: $x_i = y_j$

x_i (or y_j) is an element of the lcs under construction. For the remaining elements of the lcs, look at x_{i-1} and y_{j-1} recursively.

Case 2: $x_i \neq y_j$

2.1 $tbl[i][j-1] > tbl[i-1][j] \Rightarrow$ look at x_i and y_{j-1} recursively

2.2 $tbl[i][j-1] < tbl[i-1][j] \Rightarrow$ look at x_{i-1} and y_j recursively

2.3 $tbl[i][j-1] = tbl[i-1][j] \Rightarrow$ arbitrarily choose 2.1 or 2.2

How to do steps 1 and 2?

You are asked to do them in two ways.

Version A

Use STL vector for the table.
Fill in the table by top-down recursion with memoization.
You shall write the following function templates in namespace A.

```
namespace A {
    // template A1: print out an lcs                    tbl is a constant
    template<typename T>
    void lcs(const T* x,int i,const T* y,int j,
                    const vector<vector<int> >& tbl);

    // template A2: fill in the table and return the length of an lcs
    template<typename T>                          tbl isn't a constant
    int lcs(const T* x,int i,const T* y,int j,
                          vector<vector<int> >& tbl);

    // template A3: use vector to create a table, call A2 to fill in the
    // table, call A1 to print out an lcs, and return the length of an lcs
    template<typename T>
    int lcs(const T* x,int m,const T* y,int n);
}
```

Version B

Use new and delete to maintain the table.
Fill in the table by bottom-up iteration.
You shall write the following function templates in namespace B.

```
namespace B {
    // template B1: print out an lcs
    template<typename T>
    void lcs(const T* x,int i,const T* y,int j,
                                const int*const* tbl);

    // template B2: use new to create a table, fill in the table, call B1 to
    // print out an lcs, delete the table, and return the length of an lcs
    template<typename T>
    int lcs(const T* x,int m,const T* y,int n);
}
```

Q: Why do we use two namespaces?
A: Because templates A3 and B2 can't be overloaded.

Q: Within template B2, we shall create a table
`int** tbl;`
and pass it into template B1 to print out an lcs. Since the functionality is in, the parameter **tbl** of template B1 may at first glance be declared as
`const int** tbl;`
saying that the table entries are constant and can't be modified. But this isn't enough. Why?

A: **int**** can't be implicitly converted to **const int****, since the conversion involves 3 steps:
int**
$\rightarrow^1$ **int*const*** $\rightarrow^2$ **const int*const*** $\rightarrow^3$ **const int****
where the first two are the standard qualification conversions, but the third casts away the blue-colored **const** qualifier, which requires a **const_cast**, as in
`const_cast<const int**>(tbl)`.
There are two reasons why the parameter **tbl** is so declared:
1 To get rid of the **const_cast**, and

2 to state correctly that the table is truly unmodifiable – not only its 2nd dimension, but also its 1st dimension.

Sample test

Suffice it to run the sample test given in file hw4test.cpp.

Sample run

It is OK if the lcs's found by your algorithms are different.

```
Sequence X: 2 5 3 1 6 4
Sequence Y: 1 2 3 4 5 6
Version A: Vector
Lcs: 2 5 6
Length of lcs = 3
Version B: new and delete
Lcs: 2 5 6
Length of lcs = 3

Sequence X: p i o n e e r
Sequence Y: s p r i n g b r e a k
Version A: Vector
Lcs: p i n e
Length of lcs = 4
Version B: new and delete
Lcs: p i n e
Length of lcs = 4

Sequence X: Snoopy Pluto Garfield Shrek Micky Lorax
Sequence Y: Garfield Doraamon Snoopy Micky Pluto Shrek Lorax
Version A: Vector
Lcs: Snoopy Pluto Shrek Lorax
Length of lcs = 4
Version B: new and delete
Lcs: Snoopy Pluto Shrek Lorax
Length of lcs = 4
```