Algorithm Implementation/Strings/Longest common substring

Note to reader: It is unavoidable for this algorithm that O(nm) time is used, but all of these implementations also use O(nm) storage. The astute reader will notice that only the previous column of the grid storing the dynamic state is ever actually used in computing the next column. Thus, these algorithm can be altered to have only an O(n) storage requirement. By reassigning array references between two 1D arrays, this can be done without copying the state data from one array to another. I may return later and update this page accordingly; for now, this optimization is left as an exercise to the reader.

For large n, faster algorithms based on rolling hashes exist that run in O(n log n) time and require O(n log n) storage.

Contents

C#

Length of Longest Substring

Given two non-empty strings as parameters, this method will return the length of the longest substring common to both parameters. A variant, below, returns the actual string.

```
public int LongestCommonSubstring(string str1, string str2)
{
        if (String.IsNullOrEmpty(str1) || String.IsNullOrEmpty(str2))
                return 0;
```

```
        int[,] num = new int[str1.Length, str2.Length];
        int maxlen = 0;

        for (int i = 0; i < str1.Length; i++)
        {
                for (int j = 0; j < str2.Length; j++)
                {
                        if (str1[i] != str2[j])
                                num[i, j] = 0;
                        else
                        {
                                if ((i == 0) || (j == 0))
                                        num[i, j] = 1;
                                else
                                        num[i, j] = 1 + num[i - 1, j - 1];

                                if (num[i, j] > maxlen)
                                {
                                        maxlen = num[i, j];
                                }
                        }
                }
        }
        return maxlen;
}
```

Retrieve the Longest Substring

This example uses the **out** keyword to pass in a string reference which the method will set to a string containing the longest common substring.

```
public int LongestCommonSubstring(string str1, string str2, out string sequence)
{
        sequence = string.Empty;
        if (String.IsNullOrEmpty(str1) || String.IsNullOrEmpty(str2))
                return 0;

        int[,] num = new int[str1.Length, str2.Length];
        int maxlen = 0;
        int lastSubsBegin = 0;
        StringBuilder sequenceBuilder = new StringBuilder();

        for (int i = 0; i < str1.Length; i++)
        {
                for (int j = 0; j < str2.Length; j++)
                {
                        if (str1[i] != str2[j])
                                num[i, j] = 0;
                        else
                        {
                                if ((i == 0) || (j == 0))
                                        num[i, j] = 1;
                                else
                                        num[i, j] = 1 + num[i - 1, j - 1];

                                if (num[i, j] > maxlen)
                                {
                                        maxlen = num[i, j];
                                        int thisSubsBegin = i - num[i, j] + 1;
                                        if (lastSubsBegin == thisSubsBegin)
                                        {//if the current LCS is the same as the last time this block ran
                                                sequenceBuilder.Append(str1[i]);
                                        }
                                        else //this block resets the string builder if a different LCS is
                                        {
                                                lastSubsBegin = thisSubsBegin;
                                                sequenceBuilder.Length = 0; //clear it
                                                sequenceBuilder.Append(str1.Substring(lastSubsBegin, (i +
                                        }
                                }
                        }
                }
```

```
        }
        sequence = sequenceBuilder.ToString();
        return maxlen;
}
```

The extra complexity in this method keeps the number of new String objects created to a minimum. This is important in C# because, since strings are immutable: every time a string field is assigned to, the old string sits in memory until the garbage collector runs. Therefore some effort was put into keeping the number of new strings low.

The algorithm might be simplified (left as an exercise to the reader) by tracking only the *start* position (in, say str1, or both str1 and str2) of the string, and leaving it to the caller to extract the string using this and the returned length. Such a variant may prove more useful, too, as the actual locations in the subject strings would be identified.

```
#include "iostream"

using namespace std;

char **A;

int main(int argc, char* argv[])
{
        int satir, sira=1;

        cin >> satir;

        A = new char *[satir];

        for(int i=0; i<satir; i++)
                *(A+i)= new char [sira];

        for(int j=1; j<=satir; j++)
                cin >> A[0][j];

        for(int j=1; j<=satir; j++)
                cout << A[0][j];

        cin >> sira;

        for(int i=1; i<=sira; i++)
                cin >> A[i][0];

        for(int j=1; j<=satir; j++)
                cout << A[0][j];

        for(int i=1;i<=sira;i++){
                for(int j=1; j<=satir; j++){
                        if(A[0][j]==A[i][0]){
                                cout << A[i][0];
                        }
                }
        }

        return 0;
}
```

Python

```
def longest_common_substring(s1, s2):
    m = [[0] * (1 + len(s2)) for i in xrange(1 + len(s1))]
    longest, x_longest = 0, 0
    for x in xrange(1, 1 + len(s1)):
```

```
        for y in xrange(1, 1 + len(s2)):
            if s1[x - 1] == s2[y - 1]:
                m[x][y] = m[x - 1][y - 1] + 1
                if m[x][y] > longest:
                    longest = m[x][y]
                    x_longest = x
            else:
                m[x][y] = 0
    return s1[x_longest - longest: x_longest]
```

Perl

```
sub lc_substr {
  my ($str1, $str2) = @_;
  my $l_length = 0; # length of longest common substring
  my $len1 = length $str1;
  my $len2 = length $str2;
  my @char1 = (undef, split(//, $str1)); # $str1 as array of chars, indexed from 1
  my @char2 = (undef, split(//, $str2)); # $str2 as array of chars, indexed from 1
  my @lc_suffix; # "longest common suffix" table
  my @substrings; # list of common substrings of length $l_length

  for my $n1 ( 1 .. $len1 ) {
    for my $n2 ( 1 .. $len2 ) {
      if ($char1[$n1] eq $char2[$n2]) {
        # We have found a matching character. Is this the first matching character, or a
        # continuation of previous matching characters? If the former, then the length of
        # the previous matching portion is undefined; set to zero.
        $lc_suffix[$n1-1][$n2-1] ||= 0;
        # In either case, declare the match to be one character longer than the match of
        # characters preceding this character.
        $lc_suffix[$n1][$n2] = $lc_suffix[$n1-1][$n2-1] + 1;
        # If the resulting substring is longer than our previously recorded max length ...
        if ($lc_suffix[$n1][$n2] > $l_length) {
          # ... we record its length as our new max length ...
          $l_length = $lc_suffix[$n1][$n2];
          # ... and clear our result list of shorter substrings.
          @substrings = ();
        }
        # If this substring is equal to our longest ...
        if ($lc_suffix[$n1][$n2] == $l_length) {
          # ... add it to our list of solutions.
          push @substrings, substr($str1, ($n1-$l_length), $l_length);
        }
      }
    }
  }

  return @substrings;
}
```

VBA

```
Function LongestCommonSubstring(S1 As String, S2 As String) As String
  MaxSubstrStart = 1
  MaxLenFound = 0
  For i1 = 1 To Len(S1)
    For i2 = 1 To Len(S2)
      X = 0
      While i1 + X <= Len(S1) And _
            i2 + X <= Len(S2) And _
            Mid(S1, i1 + X, 1) = Mid(S2, i2 + X, 1)
        X = X + 1
      Wend
      If X > MaxLenFound Then
        MaxLenFound = X
```

```
        MaxSubstrStart = i1
      End If
    Next
  Next
  LongestCommonSubstring = Mid(S1, MaxSubstrStart, MaxLenFound)
End Function
```

VB.NET

```
Public Function LongestCommonSubstring(ByVal s1 As String, ByVal s2 As String) As Integer
    Dim num(s1.Length - 1, s2.Length - 1) As Integer   '2D array
    Dim letter1 As Char = Nothing
    Dim letter2 As Char = Nothing
    Dim len As Integer = 0
    Dim ans As Integer = 0
    For i As Integer = 0 To s1.Length - 1
        For j As Integer = 0 To s2.Length - 1
            letter1 = s1.Chars(i)
            letter2 = s2.Chars(j)
            If Not letter1.Equals(letter2) Then
                num(i, j) = 0
            Else
                If i.Equals(0) Or j.Equals(0) Then
                    num(i, j) = 1
                Else
                    num(i, j) = 1 + num(i - 1, j - 1)
                End If
                If num(i, j) > len Then
                    len = num(i, j)
                    ans = num(i, j)
                End If
            End If
        Next j
    Next i
    Return ans
End Function
```

COBOL

This algorithm uses no extra storage, but it runs in O(mnl) time. The 2 strings to compare should be placed in WS-TEXT1 and WS-TEXT2, and their lengths placed in WS-LEN1 and WS-LEN2, respectively. The output of this routine is MAX-LEN, the length of the largest common substring, WS-LOC1, the location within WS-TEXT1 where it starts, and WS-LOC2, the location within WS-TEXT2 where it starts.

```
01 MAX-LEN       PIC 9999 COMP.
01 WS-IX1        PIC 9999 COMP.
01 WS-IX2        PIC 9999 COMP.
01 WK-LEN        PIC 9999 COMP.
01 WS-LOC1       PIC 9999 COMP.
01 WS-LOC2       PIC 9999 COMP.
01 WS-FLAG       PIC X.
   88 NO-DIFFERENCE-FOUND VALUE 'N'.
   88 DIFFERENCE-FOUND    VALUE 'Y'.

MOVE ZERO TO MAX-LEN.
PERFORM VARYING WS-IX1 FROM 1 BY 1 UNTIL WS-IX1 > WS-LEN1
  PERFORM VARYING WS-IX2 FROM 1 BY 1 UNTIL WS-IX2 > WS-LEN2
    SET NO-DIFFERENCE-FOUND TO TRUE
    PERFORM VARYING WK-LEN FROM MAX-LEN BY 1 UNTIL
                    WS-IX1 + WK-LEN > WS-LEN1 OR
                    WS-IX2 + WK-LEN > WS-LEN2 OR
                    DIFFERENCE-FOUND
      IF WS-TEXT1(WS-IX1 : WK-LEN + 1) = WS-TEXT2(WS-IX2 : WK-LEN + 1)
         COMPUTE MAX-LEN = WK-LEN + 1
```

```
          MOVE WS-IX1 TO WS-LOC1
          MOVE WS-IX2 TO WS-LOC2
       ELSE
          SET DIFFERENCE-FOUND TO TRUE
       END-IF
     END-PERFORM
   END-PERFORM
END-PERFORM.
```

C++

```
#include <string>

using std::string;

int LongestCommonSubstring(const string& str1, const string& str2)
{
     if(str1.empty() || str2.empty())
     {
          return 0;
     }

     int *curr = new int [str2.size()];
     int *prev = new int [str2.size()];
     int *swap = nullptr;
     int maxSubstr = 0;

     for(int i = 0; i<str1.size(); ++i)
     {
          for(int j = 0; j<str2.size(); ++j)
          {
               if(str1[i] != str2[j])
               {
                    curr[j] = 0;
               }
               else
               {
                    if(i == 0 || j == 0)
                    {
                         curr[j] = 1;
                    }
                    else
                    {
                         curr[j] = 1 + prev[j-1];
                    }
                    //The next if can be replaced with:
                    //maxSubstr = max(maxSubstr, curr[j]);
                    //(You need algorithm.h library for using max())
                    if(maxSubstr < curr[j])
                    {
                         maxSubstr = curr[j];
                    }
               }
          }
          swap=curr;
          curr=prev;
          prev=swap;
     }
     delete [] curr;
     delete [] prev;
     return maxSubstr;
}
```

Ruby

```
  def self.find_longest_common_substring(s1, s2)
```

```
    if (s1 == "" || s2 == "")
      return ""
    end
    m = Array.new(s1.length){ [0] * s2.length }
    longest_length, longest_end_pos = 0,0
    (0 .. s1.length - 1).each do |x|
      (0 .. s2.length - 1).each do |y|
        if s1[x] == s2[y]
          m[x][y] = 1
          if (x > 0 && y > 0)
            m[x][y] += m[x-1][y-1]
          end
          if m[x][y] > longest_length
            longest_length = m[x][y]
            longest_end_pos = x
          end
        end
      end
    end
    return s1[longest_end_pos - longest_length + 1 .. longest_end_pos]
  end
```

Java

```
public static int longestSubstr(String first, String second) {
    if (first == null || second == null || first.length() == 0 || second.length() == 0) {
        return 0;
    }

    int maxLen = 0;
    int fl = first.length();
    int sl = second.length();
    int[][] table = new int[fl+1][sl+1];

    for(int s=0; s <= sl; s++)
      table[0][s] = 0;
    for(int f=0; f <= fl; f++)
      table[f][0] = 0;



    for (int i = 1; i <= fl; i++) {
        for (int j = 1; j <= sl; j++) {
            if (first.charAt(i-1) == second.charAt(j-1)) {
                if (i == 1 || j == 1) {
                    table[i][j] = 1;
                }
                else {
                    table[i][j] = table[i - 1][j - 1] + 1;
                }
                if (table[i][j] > maxLen) {
                    maxLen = table[i][j];
                }
            }
        }
    }
    return maxLen;
}
```

- Java-Adaptation of C# code for retrieving the longest substring

```
private static String longestCommonSubstring(String S1, String S2)
{
    int Start = 0;
    int Max = 0;
    for (int i = 0; i < S1.length(); i++)
```

```
    {
        for (int j = 0; j < S2.length(); j++)
        {
            int x = 0;
            while (S1.charAt(i + x) == S2.charAt(j + x))
            {
                x++;
                if (((i + x) >= S1.length()) || ((j + x) >= S2.length())) break;
            }
            if (x > Max)
            {
                Max = x;
                Start = i;
            }
         }
    }
    return S1.substring(Start, (Start + Max));
}
```

Java - O(n) storage

```
        public static int longestSubstr(String s, String t) {
                if (s.isEmpty() || t.isEmpty()) {
                        return 0;
                }

                int m = s.length();
                int n = t.length();
                int cost = 0;
                int maxLen = 0;
                int[] p = new int[n];
                int[] d = new int[n];

                for (int i = 0; i < m; ++i) {
                        for (int j = 0; j < n; ++j) {
                                // calculate cost/score
                                if (s.charAt(i) != t.charAt(j)) {
                                        cost = 0;
                                } else {
                                        if ((i == 0) || (j == 0)) {
                                                cost = 1;
                                        } else {
                                                cost = p[j - 1] + 1;
                                        }
                                }
                                d[j] = cost;

                                if (cost > maxLen) {
                                        maxLen = cost;
                                }
                        } // for {}

                        int[] swap = p;
                        p = d;
                        d = swap;
                }

                return maxLen;
        }
```

JavaScript

```
function longestCommonSubstring(string1, string2){
        // init max value
        var longestCommonSubstring = 0;
        // init 2D array with 0
```

```
        var table = [],
            len1 = string1.length,
            len2 = string2.length,
            row, col;
        for(row = 0; row <= len1; row++){
                table[row] = [];
                for(col = 0; col <= len2; col++){
                        table[row][col] = 0;
                }
        }
        // fill table
        var i, j;
        for(i = 0; i < len1; i++){
                for(j = 0; j < len2; j++){
                        if(string1[i]==string2[j]){
                                if(table[i][j] == 0){
                                        table[i+1][j+1] = 1;
                                } else {
                                        table[i+1][j+1] = table[i][j] + 1;
                                }
                                if(table[i+1][j+1] > longestCommonSubstring){
                                        longestCommonSubstring = table[i+1][j+1];
                                }
                        } else {
                                table[i+1][j+1] = 0;
                        }
                }
        }
        return longestCommonSubstring;
}
```

Variant to return the longest common substring and offset along with the length

```
function longestCommonSubstring(str1, str2){
        if (!str1 || !str2)
                return {
                        length: 0,
                        sequence: "",
                        offset: 0
                };

        var sequence = "",
                str1Length = str1.length,
                str2Length = str2.length,
                num = new Array(str1Length),
                maxlen = 0,
                lastSubsBegin = 0;

        for (var i = 0; i < str1Length; i++) {
                var subArray = new Array(str2Length);
                for (var j = 0; j < str2Length; j++)
                        subArray[j] = 0;
                num[i] = subArray;
        }
        var thisSubsBegin = null;
        for (var i = 0; i < str1Length; i++)
        {
                for (var j = 0; j < str2Length; j++)
                {
                        if (str1[i] !== str2[j])
                                num[i][j] = 0;
                        else
                        {
                                if ((i === 0) || (j === 0))
                                        num[i][j] = 1;
                                else
                                        num[i][j] = 1 + num[i - 1][j - 1];

                                if (num[i][j] > maxlen)
                                {
                                        maxlen = num[i][j];
                                        thisSubsBegin = i - num[i][j] + 1;
                                        if (lastSubsBegin === thisSubsBegin)
                                        {//if the current LCS is the same as the last time this block ran
```

```
                                        sequence += str1[i];
                                }
                                else //this block resets the string builder if a different LCS is
                                {
                                        lastSubsBegin = thisSubsBegin;
                                        sequence= ""; //clear it
                                        sequence += str1.substr(lastSubsBegin, (i + 1) - lastSubsB
                                }
                        }
                }
        }
        return {
                length: maxlen,
                sequence: sequence,
                offset: thisSubsBegin
        };
}
```

PHP

```
function get_longest_common_subsequence($string_1, $string_2)
{
        $string_1_length = strlen($string_1);
        $string_2_length = strlen($string_2);
        $return          = '';

        if ($string_1_length === 0 || $string_2_length === 0)
        {
                // No similarities
                return $return;
        }

        $longest_common_subsequence = array();

        // Initialize the CSL array to assume there are no similarities
        $longest_common_subsequence = array_fill(0, $string_1_length, array_fill(0, $string_2_length, 0));

        $largest_size = 0;

        for ($i = 0; $i < $string_1_length; $i++)
        {
                for ($j = 0; $j < $string_2_length; $j++)
                {
                        // Check every combination of characters
                        if ($string_1[$i] === $string_2[$j])
                        {
                                // These are the same in both strings
                                if ($i === 0 || $j === 0)
                                {
                                        // It's the first character, so it's clearly only 1 character long
                                        $longest_common_subsequence[$i][$j] = 1;
                                }
                                else
                                {
                                        // It's one character longer than the string from the previous cha
                                        $longest_common_subsequence[$i][$j] = $longest_common_subsequence[

                                }

                                if ($longest_common_subsequence[$i][$j] > $largest_size)
                                {
                                        // Remember this as the largest
                                        $largest_size = $longest_common_subsequence[$i][$j];
                                        // Wipe any previous results
                                        $return       = '';
                                        // And then fall through to remember this new value
                                }

                                if ($longest_common_subsequence[$i][$j] === $largest_size)
                                {
                                        // Remember the largest string(s)
                                        $return = substr($string_1, $i - $largest_size + 1, $largest_size)
                                }
```

```
                        }
                        // Else, $CSL should be set to 0, which it was already initialized to
                }
        }

        // Return the list of matches
        return $return;
}
```

VFP

```
function GetLongestSubstring(lcString1, lcString2)
Local lnLenString1, lnLenString2, lnMaxlen, lnLastSubStart, lnThisSubStart, i, j
Local lcLetter1, lcLetter2, lcSequence

Store Space(0) TO lcLetter1, lcLetter2, lcSequence
Store 0 TO lnLenString1, lnLenString2, lnMaxlen, lnLastSubStart, lnThisSubStart, i, j, laNum

lnLenString1 = Len(lcString1)
lnLenString2 = Len(lcString2)
Dimension laNum(lnLenString1,lnLenString2)

For i = 1 To lnLenString1
        For j = 1 To lnLenString2
                lcLetter1 = Substr(lcString1,i,1)
                lcLetter2 = Substr(lcString2,j,1)

                If !lcLetter1 == lcLetter2
                        laNum(i, j) = 0
                Else
                        If i=1 OR j=1
                                laNum(i, j) = 1
                        Else
                                laNum(i, j) = 1 + laNum(i - 1, j - 1)
                        Endif

                        If laNum(i, j) > lnMaxlen
                                lnMaxlen = laNum(i, j)
                                lnThisSubStart = i - laNum[i, j] + 1
                                If (lnLastSubStart == lnThisSubStart)
                                        lcSequence = lcSequence + lcLetter1
                                Else
                                        lnLastSubStart = lnThisSubStart
                                        lcSequence = Space(0)
                                        lcSequence = Substr(lcString1,lnLastSubStart,(i + 1) - lnLastSubSt
                                Endif
                        Endif
                Endif
        Next
Next
Return(lcSequence)
```

Haskell

```
import Data.List
import Data.Function

lcstr xs ys = maximumBy (compare `on` length) . concat $ [f xs' ys | xs' <- tails xs] ++ [f xs ys' | ys' <
  where f xs ys = scanl g [] $ zip xs ys
        g z (x, y) = if x == y then z ++ [x] else []
```

Common Lisp

```
(defun longest-common-substring (a b)
  (let ((L (make-array (list (length a) (length b)) :initial-element 0))
        (z 0)
        (result '()))
    (dotimes (i (length a))
      (dotimes (j (length b))
        (when (char= (char a i) (char b j))
          (setf (aref L i j)
                (if (or (zerop i) (zerop j))
                    1
                    (1+ (aref L (1- i) (1- j)))))
          (when (> (aref L i j) z)
            (setf z (aref L i j)
                  result '()))
          (when (= (aref L i j) z)
            (pushnew (subseq a (1+ (- i z)) (1+ i))
                     result :test #'equal)))))
    result))
```

Objective-C

```
+ (NSString *)longestCommonSubstring:(NSString *)substring string:(NSString *)string {
    if (substring == nil || substring.length == 0 || string == nil || string.length == 0) {
        return nil;

    }
    NSMutableDictionary *map = [NSMutableDictionary dictionary];
    int maxlen = 0;
    int lastSubsBegin = 0;
    NSMutableString *sequenceBuilder = [NSMutableString string];

    for (int i = 0; i < substring.length; i++)
    {
        for (int j = 0; j < string.length; j++)
        {
            unichar substringC = [[substring lowercaseString] characterAtIndex:i];
            unichar stringC = [[string lowercaseString] characterAtIndex:j];

            if (substringC != stringC) {
                [map setObject:[NSNumber numberWithInt:0] forKey:[NSString stringWithFormat:@"%i%i",i,j]];
            }
            else {
                if ((i == 0) || (j == 0)) {
                    [map setObject:[NSNumber numberWithInt:1] forKey:[NSString stringWithFormat:@"%i%i",i,
                }
                else {
                    int prevVal = [[map objectForKey:[NSString stringWithFormat:@"%i%i",i-1,j-1]] intValue
                    [map setObject:[NSNumber numberWithInt:1+prevVal] forKey:[NSString stringWithFormat:@'
                }
                int currVal = [[map objectForKey:[NSString stringWithFormat:@"%i%i",i,j]] intValue];
                if (currVal > maxlen) {
                    maxlen = currVal;
                    int thisSubsBegin = i - currVal + 1;
                    if (lastSubsBegin == thisSubsBegin)
                    {//if the current LCS is the same as the last time this block ran
                        NSString *append = [NSString stringWithFormat:@"%C",substringC];
                        [sequenceBuilder appendString:append];
                    }
                    else //this block resets the string builder if a different LCS is found
                    {
                        lastSubsBegin = thisSubsBegin;
                        NSString *resetStr = [substring substringWithRange:NSMakeRange(lastSubsBegin, (i +
                        sequenceBuilder = [NSMutableString stringWithFormat:@"%@",resetStr];
                    }
                }
            }
        }
    }
    return sequenceBuilder;
}
```

Retrieved from "http://en.wikibooks.org/w/index.php?title=Algorithm_Implementation/Strings/Longest_common_substring&oldid=2685205"