Najdłuższy Wspólny Podciąg (Longest Common Subsequence)

Jest to przykład algorytmu ilustrującego metodę *programowania dynamicznego.*

Dane: ciągi $X = x_1, x_2, \ldots, x_m$ oraz $Y = y_1, y_2, \ldots, y_n$.

Wynik: ciąg $Z = z_1, z_2, \ldots, z_k$ który jest podciągiem zarówno ciągu X jak i Y i to najdłuższym podciągiem o tej własności (znajdujemy tylko jeden podciąg).

Przykład:

$X =$ `aabbcc`
$Y =$ `abacb`

Najdłuższy wspólny podciąg X i Y: `aab`

ale też: `abb abc aab`

Najprostsza strategia szukania NWP: wygenerować wszystkie podciągi X i zaczynając od najdłuższych sprawdzać, który jest również podciągiem Y. Złożoność pesymistyczna wykładnicza bo podciągów X jest 2^m.

Przeanalizujmy problem

Niech $X = x_1, x_2, \ldots, x_m$. Wtedy X_i oznacza ciąg $x_1, x_2, \ldots, x_i$, czyli początkowy fragment ciągu X do pozycji i włącznie.
(X_0 oznacza ciąg pusty a $X_n = X$)

Skrót NWP oznacza: "najdłuższy wspólny podciąg"

Stwierdzenie Niech $X = x_1, x_2, \ldots, x_m$, $Y = y_1, y_2, \ldots, y_n$ i niech $Z = z_1, z_2, \ldots, z_k$ będzie dowolnym NWP X i Y. Wtedy

1. jeżeli $x_m = y_n$ to $z_k = x_m = y_n$ i Z_{k-1} jest NWP X_{m-1} i Y_{n-1}.
2. jeżeli $x_m \neq y_n$ i $z_k \neq x_m$ to Z jest NWP X_{m-1} i Y.
3. jeżeli $x_m \neq y_n$ i $z_k \neq y_n$ to Z jest NWP X i Y_{n-1}.

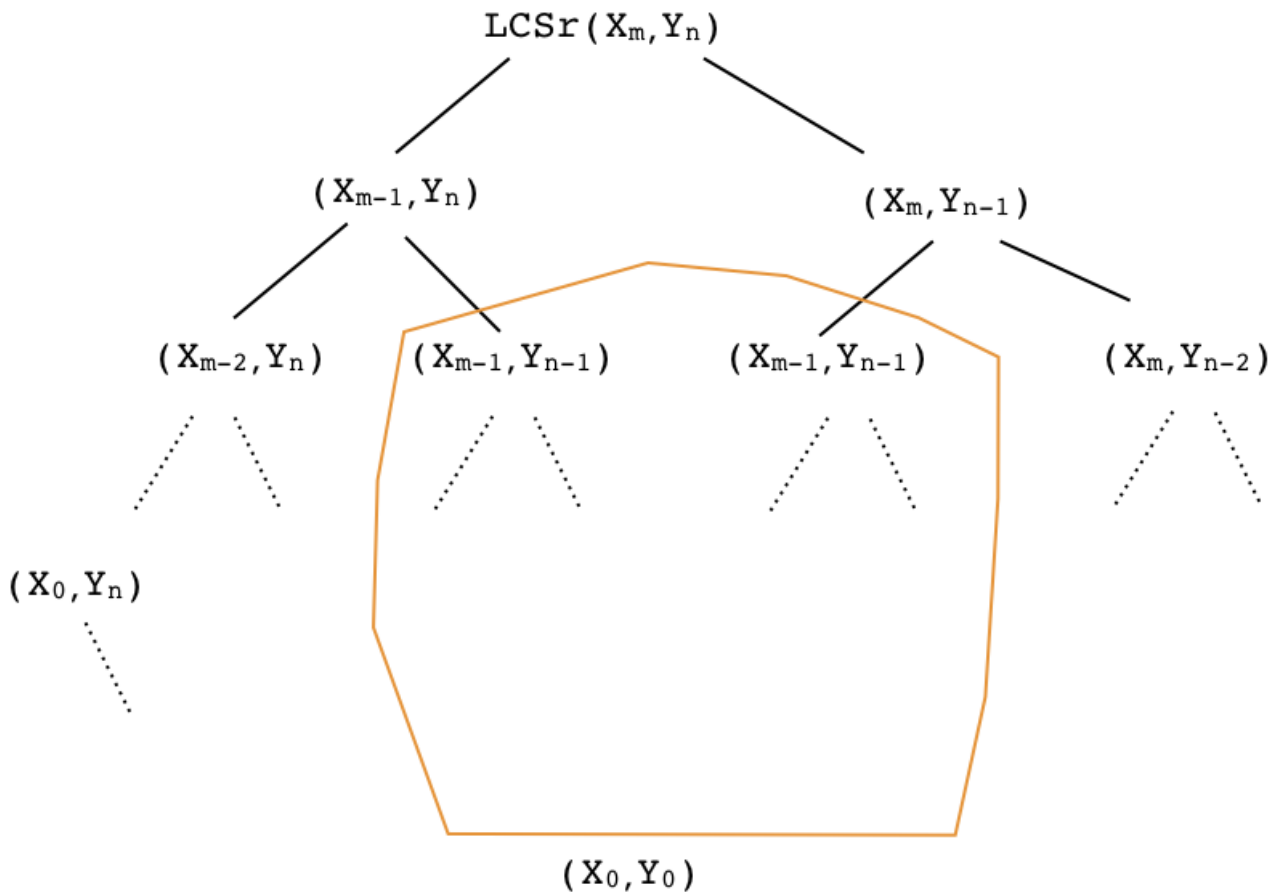
Dowód

$$X = X_m = x_1, x_2, \ldots, x_{m-1}, x_m$$
$$Y = Y_n = y_1, y_2, \ldots, y_{n-1}, y_n$$
$$Z = Z_k = z_1, z_2, \ldots, z_{k-1}, z_k$$

To uzasadnia następujący algorytm

```
LCSr(X_m,Y_n)
   if m==0 or n==0
     return pusty ciąg
   if x_m == y_n
     return LCSr(X_{m-1},Y_{n-1}) + x_m
   else
     Z_1 = LCSr(X_{m-1},Y_n)
     Z_2 = LCSr(X_m,Y_{n-1})
     return dłuższy spośród Z_1, Z_2
```

Złożoność pesymistyczna `LCSr` jest niestety wykładnicza.



Drzewo rekursji do poziomu $\min(m, n)$ jest pełnym drzewem binarnym, jeżeli ciągi X i Y nie mają wspólnych znaków. Wywołań rekurencyjnych jest $\Theta(2^{\min(m,n)})$. Dzieje się tak, bo wielokrotnie powtarzamy te same obliczenia.

Idea efektywnego rozwiązania. Organizujemy obliczenie tak, żeby tlko raz liczyć powtarzające się obliczenia i spamiętywać ich wyniki do ponownego wykorzystania. Takie organizowanie obliczenia nazywamy właśnie *programowaniem dynamicznym*.

Dla `i = 0,1, ..., m` oraz `j = 0,1, ..., n` wyliczymy długość $NWP(X_i, Y_j)$ i zapamiętujemy tę wartość w tablicy (dwuwymiarowej) `c[i,j]` .

Ażeby wyliczyć `c[i,j]` wystarczy wiedzieć jakie są x_i i y_j oraz znać wartości `c[i-1,j-1]` , `c[i-1,j]` , `c[i,j-1]` .

$i \setminus j$	(0)	y_1 (1)	y_2 (2)	y_j (j)	…	y_n (n)
	0	0	0	0	0	0
x_1 (1)	0					
x_2 (2)	0					
x_i (i)	0			NWP (X_i, Y_j)		
…	0					
x_m (m)	0					

Najdłuższy Wspólny Podciąg (wersja iteracyjna)

Ciągi wejściowe indeksowane od pozycji `1` , czyli
`x[1], x[2], ... , x[m]`
`y[1], y[2], ... , y[n]` .
Tablice `c, b` indeksowane od `0` .

w `c[i,j]` wyliczymy długość $NWP(X_i, Y_j)$

```
LCS-Length(x,y)
  m = x.length
  n = y.length
  for i = 0 to m
    c[i,0] = 0
  for j = 1 to n
    c[0,j] = 0
  for i = 1 to m
    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 b,c
```

Drukowanie wyniku

```
PrintLCS(x,b,i,j)
  if i==0 lub j==0
      return
  if b[i,j] == "\"
     PrintLCS(x,b,i-1,j-1)
     drukuj x[i]
  else if b[i,j] == "|"
          PrintLCS(x,b,i-1,j)
       else
          PrintLCS(x,b,i,j-1)
```

Stwierdzenie Złożoność pesymistyczna znalezienia i wypisania najdłuższego wspólnego podciągu ciągów o długościach m i n, przy zastosowaniu metody programowania dynamicznego jest $\Theta(m\,n)$.

Uwaga Można nie używać pomocniczej tablicy `b` a wyliczać na bieżąco jej zawartości na podstawie `x` i `y`

Wersja rekurencyjna ze spamiętywaniem

```
LCS-Rec(x,y)
  m = x.length
  n = y.length
  for i = 1 to m
    for j = 1 to n
        c[i,j]= INFINITY
  for i = 1 to m
     c[i,0] = 0
  for j = 0 to n
     c[0,j] = 0
  LCS-Lookup(m,n,c,b)
  return b,c

LCS-Lookup(i,j,c,b)
  if c[i,j]<INFINITY
     return c[i,j]
  if x[i]=y[j]
     c[i,j] = LCS-Lookup(i-1,j-1,c,b)+1
     b[i,j] = "\"
   else c1 = LCS-Lookup(i-1,j,c,b)
        c2 = LCS-Lookup(i,j-1,c,b)
        if c1>=c2 // czyli c[i-1,j] >= c[i,j-1]
           c[i,j] = c1
           b[i,j] = "|"
        else
            c[i,j] = c2
            b[i,j] = "-"
  return c[i,j]
```