[CSE3081(2반)] 알고리즘 설계와 분석

2020학년도 2학기

강의자료

(2020.10.29 목요일)

서강대학교 공과대학 컴퓨터공학과
임 인 성 교수

[주제 4]

Dynamic Programming

Longest Common Subsequence (LCS)

- Definitions
  - Given a sequence $X = <x_1, x_2, \ldots, x_m>$, another sequence $Z = <z_1, z_2, \ldots, z_k>$ is a ***subsequence*** of X if there exists a strictly increasing sequence $<i_1, i_2, \ldots, i_k>$ of indices of X such that for all $j = 1, 2, \ldots, k$, we have $x_{i_j} = z_j$.
    - A subsequence of a given sequence is just the given sequence with some elements (possibly none) left out.
    - Ex: $X = <A, B, C, B, D, A, B>$, $Z = <B, C, D, B>$ (<2, 3, 5, 7>)
  - Given two sequences X and Y, we say that a sequence Z is a **common subsequence** of X and Y if Z is a subsequence of both X and Y.
    - Ex: $X = <A, B, C, B, D, A, B>$, $Y = <B, D, C, A, B, A>$, $Z_1 = <B, C, A>$, $Z_2 = <B, C, B, A>$, $Z_3 = <B, D, A, B>$
  - Given a sequence $X = <x_1, x_2, \ldots, x_m>$, $X_i = <x_1, x_2, \ldots, x_i>$ is the ith ***prefix*** of X, for $i = 0, 1, \ldots, m$.
    - Ex: $X = <A, B, C, B, D, A, B>$, $X_4 = <A, B, C, B>$, X_0 = null sequence

- **Problem**
  - 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 *longest common subsequence* of X and Y.

X = TCCCCGCTCTGCTCTGTCCGGTCACAGGACTTTTTGCCCTCTGTTCCCGGGTCCCTCAGGCGGCCACCCA
GTGGGCACACTCCCAGGCGGCGCTCCGGCCCCGCGCTCCCTCCCTCTGCCTTTCATTCCCAGCTGTCAAC
ATCCTGGAAGCTTTGAAGCTCAGGAAAGAAGAGAAATCCACTGAGAACAGTCTGTAAAGGTCCGTAGTGC
TATCTACATCCAGACGGTGGAAGGGAGAGAAAGAGAAAGAAGGTATCCTAGGAATACCTGCCTGCTTAGA
CCCTCTATAAAAGCTCTGTGCATCCTGCCACTGAGGACTCCGAAGAGGTAGCAGTCTTCTGAAAGACTTC
AACTGTGAGGACATGTCGTTCAGATTTGGCCAACATCTCATCAAGCCCTCTGTAGTGTTTCTCAAAACAG
AACTGTCCTTCGCTCTTGTGAATAGGAAACCTGTGGTACCAGGACATGTCCTTGTGTGCCCGCTGCGGCC
AGTGGAGCGCTTCCATGACCTGCGTCCTGATGAAGTGGCCGATTTGTTTCAGACGACCCAGAGAGTCGGG
ACAGTGGTGGAAAAACATTTCCATGGGACCTCTCTCACCTTTTCCATGCAGGATGGCCCCGAAGCCGGAC
AGACTGTGAAGCACGTTCACGTCCATGTTCTTCCCAGGAAGGCTGGAGACTTTCACAGGAATGACAGCAT
CTATGAGGAGCTCCAGAAACATGACAAGGAGGACTTTCCTGCCTCTTGGAGATCAGAGGAGGAAATGGCA
GCAGAAGCCGCAGCTCTGCGGGTCTACTTTCAGTGACACAGATGTTTTTCAGATCCTGAATTCCAGCAAA
AGAGCTATTGCCAACCAGTTTGAAGACCGCCCCCCCGCCTCTCCCCAAGAGGAACTGAATCAGCATGAAA
ATGCAGTTTCTTCATCTCACCATCCTGTATTCTTCAACCAGTGATCCCCCACCTCGGTCACTCCAACTCC
CTTAAAATACCTAGACCTAAACGGCTCAGACAGGCAGATTTGAGGTTTCCCCCTGTCTCCTTATTCGGCA
GCCTTATGATTAAACTTCCTTCTCTGCTGCAAAAAAAAAAAAAAA

Y = ATGTTAACCAAGGAATGGATCTGTGTCGTTCCACGTTCGAAGGCCTTTTCTGATGAAATGAAGATAGGTT
TCAACTCCACAGGTTATTGTGGTATGATCTTAACCAAAAATGATGAAGTTTTCTCCAAGATTACTGAAAA
ACCTGAATTGATTAACGATATCTTATTGGAATGTGGTTTCCCAAACACTTCTGGTCAAAAACCAAACGAA
TACAACTATTGAGTCCTTCACAAGTACAAGTATACACGTATACATGTATGTATATATATATGTATTTAAA
TGATAGAAGTAATTTCTATATGTATATGTCTATTCAATTTTTATTTCTAATGACTTTGAAATTTTATATT
TATTATTCTACACTTTATTATTAAAACTACATGCAATCAATGCCGCTAAGGTTACGATTCACCTTTTTAT
TACTTATTATTAATATTGTTGTATTACTTCCTTGAAAAAATATGTCTAAAGAGTCCTAATTTGGATTTTC
TTTTCCTCCTAACTTCACTGTCCTGCGCCTGCTTTCCTAACGCACCATCGCTAATACACCAGCTTTCATT
GCTTGTTGCTGCGCTATTGCTCGCATGGAACGTTTTCAGTGCGTCATCATCCTGGGATAAACTAAAGACT
AAGTCACCAGTTTCATTTGAGGCTTTTCTCTGCGTGTTGACAAAAGAGGACAGATCAACCATATCACCTG
GTTCACCATGAGAATGTCCTTACTTAGTTGCGAATTTGGTTCTGATCTGCCTTCAGACTTGGAATCATTC
ATTACTGTCATTACTTTTAGCCTATTCATTTTCTTCCTTGCCATCAGGTACAGGGATTTGACCACAGAGT
GTTGAAGGGGTGCATCGTCCGCTCTCGTAATAACCCTCCGATACTATTTTCATTGTTGGCACGTTGCACT
GAAAAGGGCACTTGGCACTGTGCACTTTTAATGTTTTCATTTTTCATCATATCATCATATGGCATTTCAT
AATGTTGACTCTTGTAGTTGAAGATTGAGTTTATCGTGTTACTGTTTCGTCTACCTTTCATATTATCAAT
CAGGTTGCGGTGTTGCATGTGGGAGATAGGACTGCCCGATCTTCTTCTTCTCGATTCTCCACTAAAATCC
TGTCTTTTATCGCTATCCAGCACACGATTGAGCTGTGAATTGCCGCACTTTTTAGGATAACCATCCTTGG

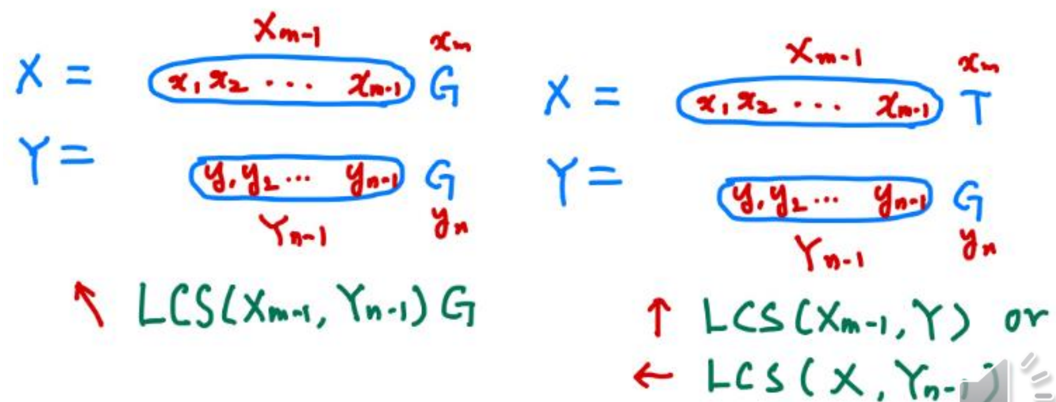
- Naïve approach
  - Enumerate all subsequences of X and check each subsequence to see if it is also a subsequence of Y, keeping track of the longest subsequence found.

    → *Exponential algorithm!*
  - ✓ The LCS problem can be solved efficiently using dynamic programming.

- **Optimal substructure of an LCS:** Let $X = <x_1, x_2, \ldots, x_m>$ and $Y = <y_1, y_2, \ldots, y_n>$ be sequences, and let $Z = <z_1, z_2, \ldots, z_k>$ 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}.

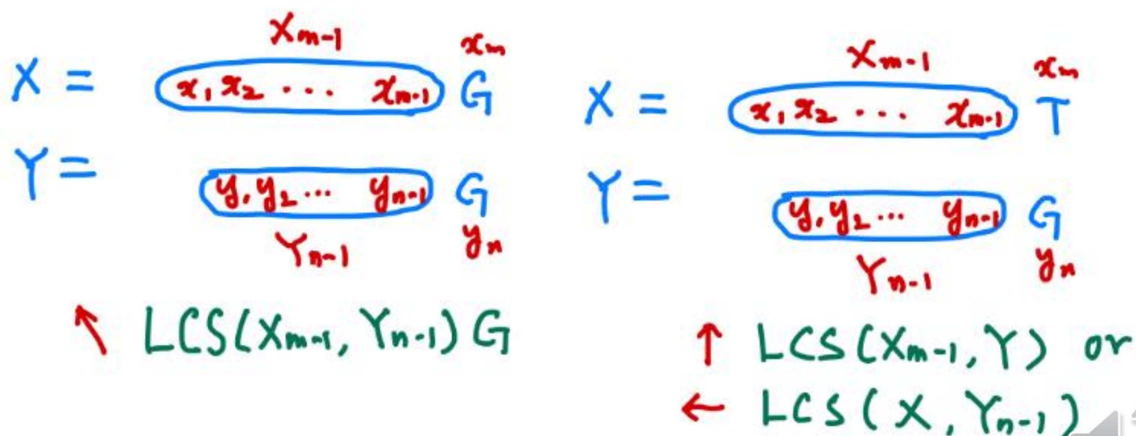
  ② If $x_m \neq y_n$, then an LCS of X and Y is either <u>an LCS of X_{m-1} and Y</u> or <u>an LCS of X and Y_{n-1}</u>.



- Let $c[i, j]$ be the length of an LCS of the sequences X_i and Y_j.

- Optimal substructure for computing $c[i, j]$

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



$O(mn)$ Algorithm

- **Filling the table**

$X = \overbrace{x_1, x_2 \cdots x_{m-1}}^{X_{m-1}} \overset{x_m}{G}$

$Y = \underbrace{y_1, y_2 \cdots y_{n-1}}_{Y_{n-1}} \underset{y_n}{G}$

↖ $LCS(X_{m-1}, Y_{n-1})\ G$

$X = \overbrace{x_1, x_2 \cdots x_{m-1}}^{X_{m-1}} \overset{x_m}{T}$

$Y = \underbrace{y_1, y_2 \cdots y_{n-1}}_{Y_{n-1}} \underset{y_n}{G}$

↑ $LCS(X_{m-1}, Y)$ or

← $LCS(X, Y_{n-1})$

```
LCS-LENGTH(X, Y)
m ← length[X]
n ← length[Y]
for i ← 1 to m
      do c[i, 0] ← 0
for j ← 0 to n
      do c[0, j] ← 0
for i ← 1 to m
      do for j ← 1 to n
              do if x_i = y_j
                     then c[i, j] ← c[i − 1, j − 1] + 1
                          b[i, j] ← "↖"
                     else if c[i − 1, j] ≥ c[i, j − 1]
                             then 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
```

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

i \ j		0	1	2	3	4	5	6
	y_j		B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3
6	A	0	↑1	↑2	↑2	↖3	↑3	↖4
7	B	0	↖1	↑2	↑2	↑3	↖4	↑4

- **Printing the LCS**

$X = \underbrace{x_1, x_2 \cdots x_{m-1}}_{X_{m-1}}\ G$ (x_m)

$Y = \underbrace{y_1, y_2 \cdots y_{n-1}}_{Y_{n-1}}\ G$ (y_n)

↖ $LCS(X_{m-1}, Y_{n-1})\ G$

$X = \underbrace{x_1, x_2 \cdots x_{m-1}}_{X_{m-1}}\ T$ (x_m)

$Y = \underbrace{y_1, y_2 \cdots y_{n-1}}_{Y_{n-1}}\ G$ (y_n)

↑ $LCS(X_{m-1}, Y)$ or

← $LCS(X, Y_{n-1})$

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

j		0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3
6	A	0	↑1	↑2	↑2	↖3	↑3	↖4
7	B	0	↖1	↑2	↑2	↑3	↖4	↑4

C Implementation

Courtesy of
http://www.bioalgorithms.info/downloads/code/

```c
/** Copyright (C) 2005 Neil Jones. **/

#include <stdio.h>
char* LCS(char* a, char* b);

#define NEITHER 0
#define UP 1
#define LEFT 2
#define UP_AND_LEFT 3

int main(int argc, char* argv[]) {
  printf("%s\n", LCS(argv[1],argv[2]));
}

char* LCS(char* a, char* b) {
  int n = strlen(a); int m = strlen(b);
  int** S; int** R; int ii; int jj;
  int pos; char* lcs;

  S = (int **)malloc( (n+1) * sizeof(int *) );
  R = (int **)malloc( (n+1) * sizeof(int *) );

  for (ii = 0; ii <= n; ++ii) {
    S[ii] = (int*) malloc( (m+1) * sizeof(int) );
    R[ii] = (int*) malloc( (m+1) * sizeof(int) );
  }
```

```c
  for (ii = 0; ii <= n; ++ii) {
    S[ii][0] = 0; R[ii][0] = UP;
  }
  for (jj = 0; jj <= m; ++jj) {
    S[0][jj] = 0; R[0][jj] = LEFT;
  }
  for (ii = 1; ii <= n; ++ii) {
    for (jj = 1; jj <= m; ++jj) {
      if ( a[ii-1] == b[jj-1] ) {
        S[ii][jj] = S[ii-1][jj-1] + 1;
        R[ii][jj] = UP_AND_LEFT;
      }
      else {
        S[ii][jj] = S[ii-1][jj-1] + 0;
        R[ii][jj] = NEITHER;
      }
      if ( S[ii-1][jj] >= S[ii][jj] ) {
        S[ii][jj] = S[ii-1][jj];
        R[ii][jj] = UP;
      }
      if ( S[ii][jj-1] >= S[ii][jj] ) {
        S[ii][jj] = S[ii][jj-1];
        R[ii][jj] = LEFT;
      }
    }
  }
```

```c
  ii = n;
  jj = m;
  pos = S[ii][jj];
  lcs = (char *) malloc( (pos+1) * sizeof(char) );
  lcs[pos--] = (char)NULL;

  while ( ii > 0 || jj > 0 ) {
    if( R[ii][jj] == UP_AND_LEFT ) {
      ii--; jj--;
      lcs[pos--] = a[ii];
    }
    else if ( R[ii][jj] == UP ) {
      ii--;
    }
    else if ( R[ii][jj] == LEFT ) {
      jj--;
    }
  }
  for (ii = 0; ii <= n; ++ii ) {
    free(S[ii]); free(R[ii]);
  }
  free(S);
  free(R);
  return lcs;
}
```

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

The Gapped Alignment Problem

$A = \varepsilon$
$B = ATCG$

```
- - - -
| | | |
A T C G
```

- **Problem:** Given two sequences, find a gapped alignment that maximize the score!
  - Compare two sequences if they are similar (related).
  - Gapped alignment
    - Example: A = **ATCGGATCT**, B = **ACGGACT**

```
A T C G G A T _ C T     A T C G G A T C T
|   |   |       | |     |   | | |     | |
A - C - G G - A C T     A - C G G - A C T
```

5 * 2 +1*(-1) + 4*(-2) = 1   6 * 2 + 1*(-1) + 2*(-2) = 7

$A_i = a_1 a_2 \cdots a_{i-1}\ a_i$
$B_j = b_1 b_2 \cdots b_{j-1}\ b_j$

$A_i = a_1 a_2 \cdots a_{i-1}\ a_i\ -$
$B_j = b_1 b_2 \cdots b_{j-1}\ b_j$

$A_i = a_1 a_2 \cdots a_{i-1}\ a_i$
$B_j = b_1 b_2 \cdots b_{j-1}\ b_j\ -$

    - A possible alignment scoring scheme
      - Ex: match score = 2, mismatch penalty = -1, gap penalty = -2
- **Optimal substructure**

$$S(i,j) = \begin{cases} -2j, & \text{if } i = 0 \\ -2i, & \text{if } j = 0 \\ \max\{\, S(i-1,j-1) + s(a_i,b_j), S(i,j-1) - 2, S(i-1,j) - 2 \,\}, & \text{if } i > 0 \text{ and } j > 0 \end{cases}$$

Longest Increasing Subsequence (LIS)

- **Problem:** Given a sequence $A = (a[0], a[1], \cdots, a[n-1])$, find the length of the longest subsequence such that all elements of the subsequence are sorted increasing order.

- **Example**
  - (10, 22, 9, 33, 21, 50, 41, 60, 80) → (10, 22, 33, 50, 60, 80)
  - (0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15) → (0, 2, 6, 9, 11, 15)

    (0, 4, 6, 9, 11, 15)

    …

- **Algorithm**

  Let $d[i]$ be the length of the longest increasing subsequence that ends in the element at index i. Then, the answer to the LIS problem is the maximum value of $d[i]$, $i = 0, 1, \cdots, n-1$.

- **Optimal substructure**

i

$(\underline{0}, \underline{8}, \underline{4}, 12, \underline{2}, 10, \underline{6}, 14, \underline{1}, 9$

$a[i]$

$$d[i] = \max \left(1, \max_{\substack{j=0 \ldots i-1 \\ a[j] < a[i]}} (d[j] + 1) \right), i = 0, 1, \cdots, n-1$$

```c
int LIS( int* a, int N ) {
  int *best, *prev, i, j, max = 0;

  best = (int*) malloc ( sizeof( int ) * N );
  prev = (int*) malloc ( sizeof( int ) * N );
  for ( i = 0; i < N; i++ ) best[i] = 1, prev[i] = i;

  for ( i = 1; i < N; i++ )
    for ( j = 0; j < i; j++ )
      if ( a[i] > a[j] && best[i] < best[j] + 1 )
        best[i] = best[j] + 1, prev[i] = j;

  for ( i = 0; i < N; i++ )
    if ( max < best[i] ) max = best[i];
  // Print the LIS using prev[] here.
  free( best );  free( prev );
  return max;
}
```

i

(0, 8, 4, 12, 2, 10, 6, 14, 1, 9

a[i]

$$d[i] = \max\left(1, \max_{\substack{j=0\dots i-1 \\ a[j]<a[i]}} (d[j]+1)\right), i = 0, 1, \cdots, n-1$$