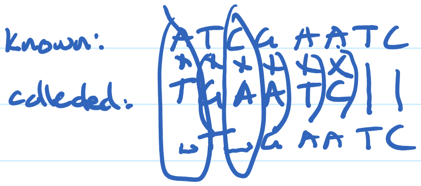
MORE DYNAMIC PROGRAMMING

Tuesday, November 6, 2018 5:26 PM

(1) DNA Matching

A DNA molecule can be thought of as a long string over alphabet $\{A, T, C, G\}$ (length $\sim O(10^6)$))

A labor-intensive and ERROR-PRONE process to sequence DNA



LONGEST COMMON SUBSEQUENCE PROBLEM

INPUT: two strings $X[0..n-1]$, $Y[0..m-1]$

OUTPUT: the length of a longest common subsequence of X, Y

Def: A subsequence S of a string X is obtained by removing a subset of characters in X.

Ex: $X = A \overset{\times}{B} C \overset{\times}{B} D A B$

Subsequences: A C D A B

B D A

␣

A B C B D A B

A subsequence is NOT a substring, which must be a single block

Def. Common subsequence

X = A B C B D A B BDAB

Y = B D C A B A BCAB

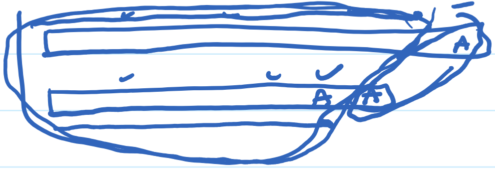
A DYNAMIC PROGRAMMING SOLUTION

Case 1

last letter match

X =

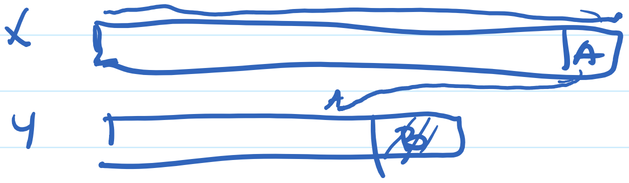
Y =



CLAIM: If last letter of X, Y match, there is a longest common match

CLAIM: If last letter of x, y match, there is a longest common subsequence that include these letters

Case 2: last letters do not match



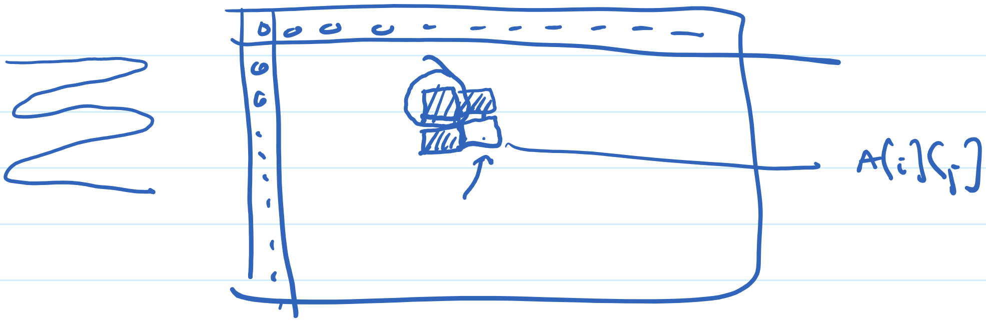
Recurrence

$$\overset{\text{lengths}}{A[m][n]} = \begin{cases} 1 + A[m-1][n-1] & \text{if } x[m-1] == y[n-1] \\ A[m-1][n] \\ A[m][n-1] \end{cases}$$

Subproblems $A[i][j]$ = length of longest common subsequence of $x[0..i-1]$ and $y[0..j-1]$

Base cases: $i = 0$ $A[0][j] = 0$

$j = 0$ $A[i][0] = 0$



$x[i-1] == y[j-1]$?

$x[i-1] \; != \; y[j-1]$

Ex.: $x =$ A B C B D A B

$y =$ B D C A B A

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2

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

BCAB

BC BA

X columns

Y rows

```
LCS ( X[0..n-1], Y[0..m-1] )
{
    for (i = 0; i ≤ n; ++i)
        S[0][i] = 0;
    for (i = 0; i ≤ m; ++i)
        S[i][0] = 0;
    for (r = 1; r ≤ m; ++r)
      for (c = 1; c ≤ n; ++c)
      {
          S[r][c] = max (S[r-1][c], S[r][c-1]);
          if (X[c-1] == Y[r-1])
              S[r][c] = 1+ S[r-1][c-1];
      }
    return S[m][n];
}
```

② MATRIX CHAIN MULTIPLICATION

Matrix

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{2\times3} \quad B = \begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix}_{2\times2}$$

$$\underset{2\times2\ \ 2\times3}{BA} \qquad \underset{2\times\boxed{3}\ \neq\ \boxed{2}\times2}{AB} \text{ not define}$$

$$BA = \begin{bmatrix} 1 & 3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 13 & 17 & 21 \end{bmatrix}$$

$$BA = \underbrace{\begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix}}_{2\times 2} \underbrace{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}}_{2\times 3} = \begin{bmatrix} 13 & 17 & 21 \\ 33 & 45 & 57 \end{bmatrix}_{2\times 3}$$

$B_{m\times n} \, A_{n\times p}$ $\boxed{\Theta(m\times p\times n)}$

MATRIX CHAINED MATRICES

Input: $M_{d_0\times d_1} \cdot M_{d_1\times d_2} \cdot M_{d_2\times d_3} \cdot \cdots \; M_{d_{n-1}\times d_n}$

Output: the least number of integer multiplications required to compute this product of matrices.

Ex: $M_{10\times 100} \cdot M_{100\times 5} \cdot M_{5\times 50}$

(A) $\underbrace{(M_{10\times 100} \cdot M_{100\times 5})_{10\times 5} \cdot M_{5\times 50}} = (\quad)_{10\times 50}$

$5000 + 2500 = \boxed{7500}$

(B) $M_{10\times 100} \cdot (M_{100\times 5} \cdot M_{5\times 50})_{100\times 50}$

$100\times 5\times 50 = 25{,}000$

$10\times 100\times 50 = 50{,}000$

$\boxed{75{,}000}$

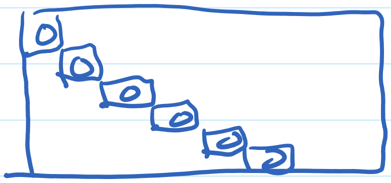
Subproblems: $\underbrace{M_1 \quad M_2} \times \underbrace{\cdots \quad M_n}$

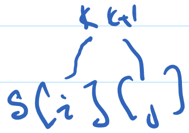
$\underbrace{(M_1 \quad M_2) \times (M_3 \quad M_4 \quad M_5)}$

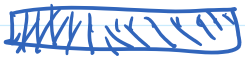
$S[i][j] =$ least answer for multiplying $M_i \cdots M_j$

Base cases $S[i][i] = 0$

Base cases $S[i][i] = 0$

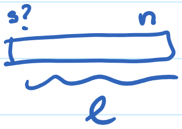






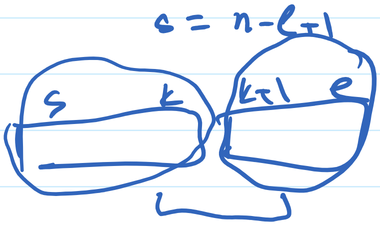
$M_{d[0]d[1]}$ $M_{d[1]d[2]}$ . . . . . $M_{d[n-1]d[n]}$

```
MCM ( d[0..n] )
{
    for (i = 1; i ≤ n; ++i)
        S[i][i] = 0;          // chains of length 1
    for ( l = 2 ; l ≤ n ; ++l )
    {
        for ( s = 1; s ≤ n-l+1 ; ++s )
        {
            e = s+l-1; M[s][e] = ∞ ;
            for ( k = s; k < e; ++k )
            {
                s[s][e] = min( S[s][e], S[s][k] + S[k+1][e]
                                        + d[s-1] * d[k] * d[e] ))
            }
        {
    }
    return S[1][n];
}
```



~~s~~ $n - s + 1 = l$

$s = n - l + 1$



$d[s] \times d[k] \times d[e]$