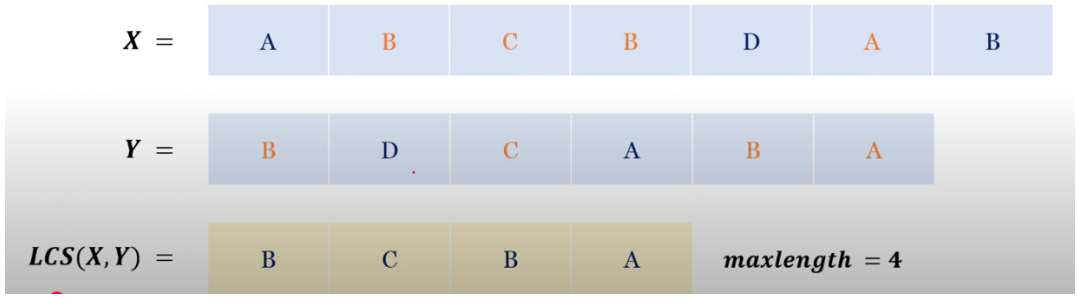
챕터 6. Dynamic Programming (동적 계획법) - (3) 최장공통부분서열 문제 (LCS)

LCS (Longest Common Subsequence)

- 두 개의 문자서열 X, Y가 주어졌을 때,
  - X와 Y에서 공통으로 나타나는 부분 문자서열을 찾고자 한다.
  - 부분 문자서열의 길이가 최대가 되도록 부분 문자서열을 찾는 방법은?



DNA 염기서열에서 공통 염기서열을 찾는 것은 굉장히 중요한 문제이다.

문제 정의

- 입력:
  - X = [x1, x2, --- xn]
  - Y = [y1, y2, --- yn]
- 출력:
  - LCS(X, Y)의 최장 길이

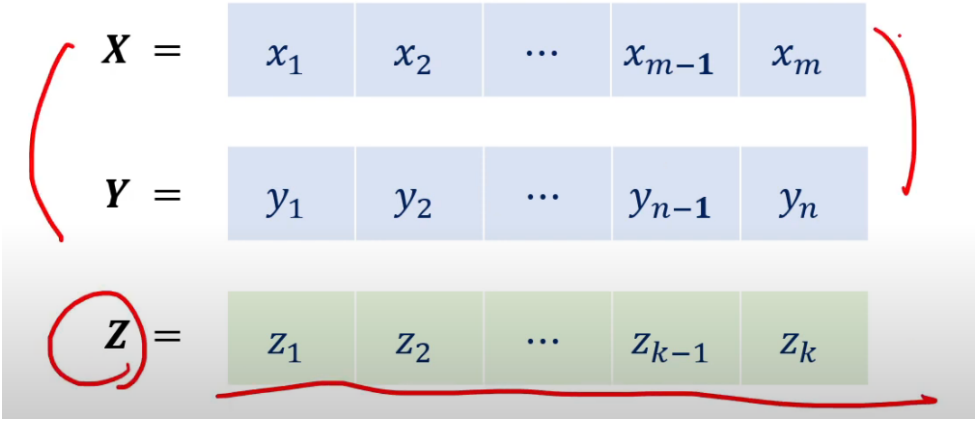
단순 무식하게 풀기: Brute-Force Approach

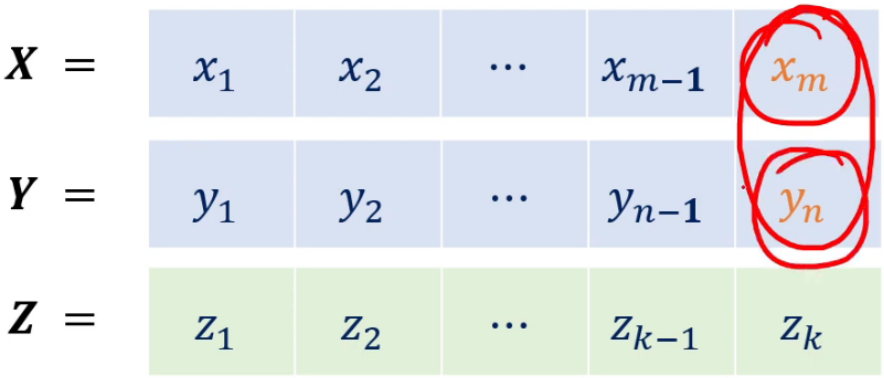
- X의 모든 부분 서열 중에서
  - Y의 부분 서열인 것들의 길이를 구한 뒤
  - 이 길이들 중에서 최대값을 찾는다.
- X의 모든 부분 서열의 개수는? 2^m
  - 지수 시간 복잡도를 가진 알고리즘: exponential time complexity

재귀적 관계를 파악하기: Recurrence Relation

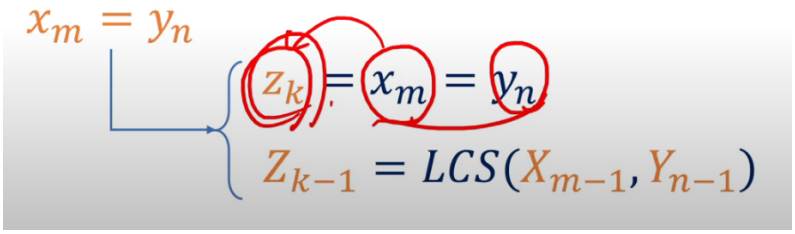
- X = [x1, x2, --- xm], Y = [y1, y2, --- yn]

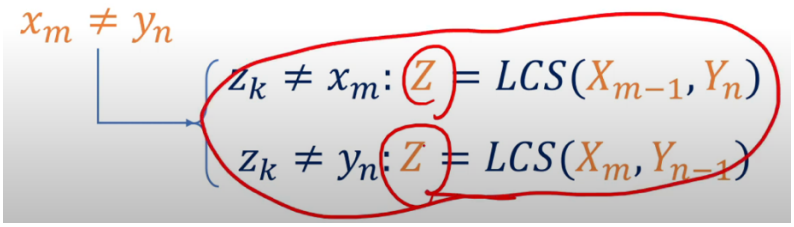
- Z = LCS(X, Y) = [z1, z2, --- zk],





-> 제일 끝에 있는 두 가지의 원소를 가지고 고려를 해보자.



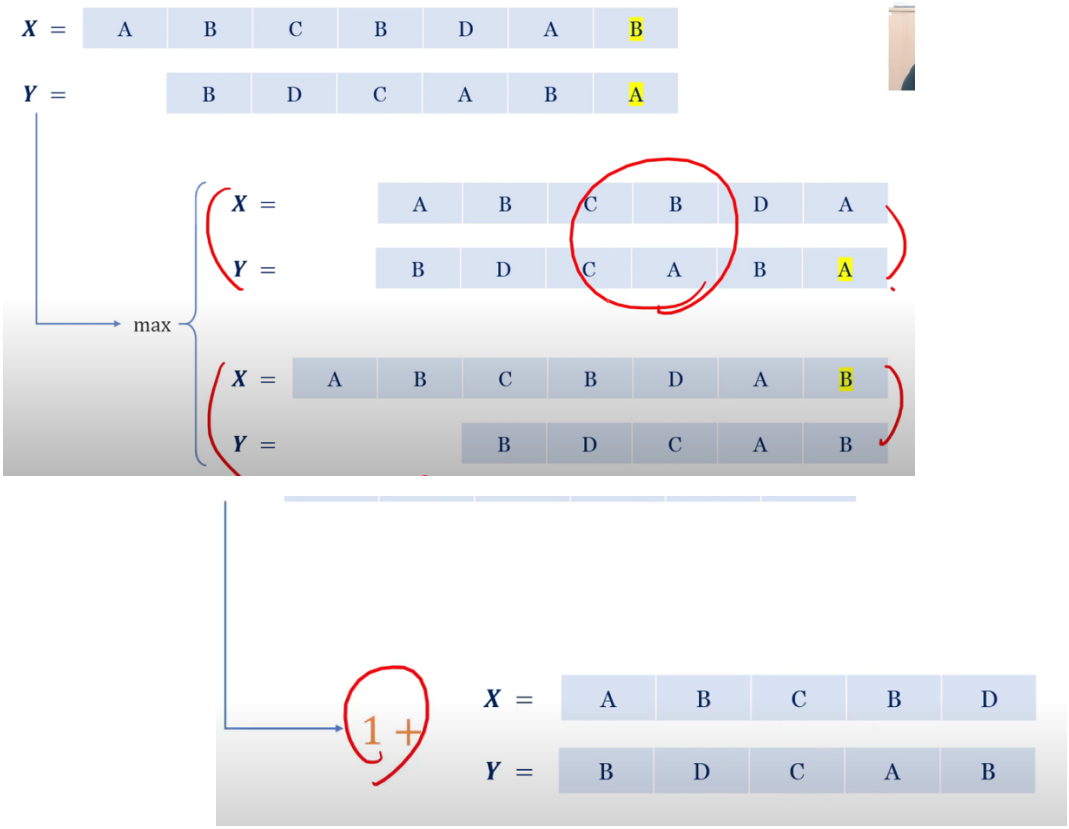


재귀적으로 정의하기: Recurrence Equation

- c(i, j): 수열 Xi와 Yj의 최장공통부분서열(LCS)의 길이로 정의
- 재귀식:
  - 종료조건: i = 0 또는 j = 0이면 c(i, j) = 0
  - 재귀조건:
    - $X_i = Y_i$ 이면 C(i, j) = C(i - 1, j - 1) + 1
    - X_i != Y_i 이면 C(i, j) = max{C(i, j - 1), C(i - 1, j)}

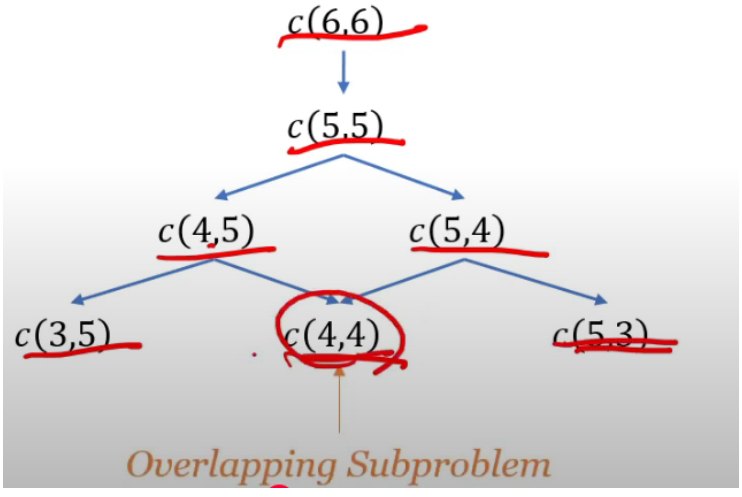
```python
def lcs(x, y):
    m, n = len(x), len(y)
    if m == 0 or n == 0:
        return 0
    else:
        if x[-1] == y[-1]:
            return lcs(x[:(m-1)], y[:(n-1)]) + 1
        else:
            return max(lcs(x[:m], y[:(n-1)]),
                       lcs(x[:(m-1)], y[:n]))
```

```cpp
int lcs(string x, string y) {
    int m = x.length();
    int n = y.length();

    if (m == 0 || n == 0) {
        return 0;
    }

    if (x[m - 1] == y[n - 1]) {
        return lcs(x.substr(0, m - 1), y.substr(0, n - 1)) + 1;
    } else {
        return max(
            lcs(x, y.substr(0, n - 1)),
            lcs(x.substr(0, m - 1), y)
        );
    }
}
```



중복 부분 문제: Overlapping Subproblems

- c(i, j) = 0
- c(i, j) = c(i - 1, j - 1) + 1
- c(i, j) = max{c(i, j - 1), c(i - 1, j)}



동적계획법의 적용:

- 메모이제이션(memoization): 이미 계산한 값을 테이블에 저장

- 하향식(top-down) 해법에서 상향식(bottom-up) 해법으로 전환

```cpp
vector<vector<int>> lcs(string x, string y) {
    // 공백 문자 추가 (파이썬의 [' '] + x 와 동일)
    x = ' ' + x;
    y = ' ' + y;

    int m = x.length();
    int n = y.length();

    vector<vector<int>> c(m, vector<int>(n, 0)); // m x n 2차원 DP 테이블

    for (int i = 1; i < m; ++i) {
        for (int j = 1; j < n; ++j) {
            if (x[i] == y[j]) {
                c[i][j] = c[i - 1][j - 1] + 1;
            } else {
                c[i][j] = max(c[i][j - 1], c[i - 1][j]);
            }
        }
    }

    return c;
}
```

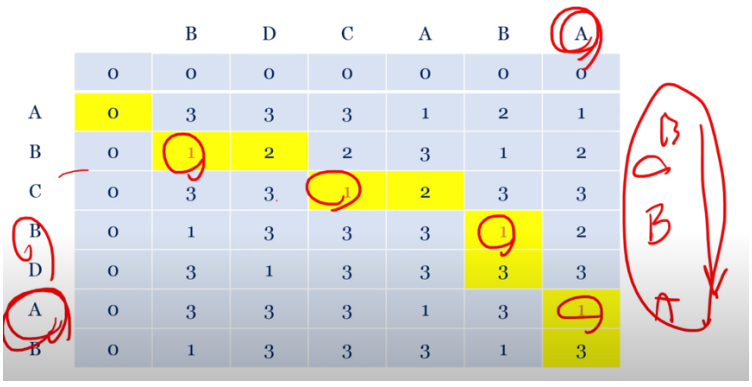
		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	0	0	0	1	1	1
B	0	1	1	1	1	2	2
C	0	1	1	2	2	2	2
B	0	1	1	2	2	3	3
D	0	1	2	2	2	3	3
A	0	1	2	2	3	3	4
B	0	1	2	2	3	4	4

최적해의 재구성:

- LCS의 최장 길이는 알겠는데,
    - 최장 길이를 가진 LCS는 어떻게 알지?
- 최적해를 계산하는 과정을 별도로 기록
    - 하향식(top-down)으로 최적해를 재구성할 수 있음

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

```cpp
// LCS 문자열을 재귀적으로 복원하는 함수
string get_lcs(int i, int j, const vector<vector<int>>& b, const string& x) {
    if (i == 0 || j == 0) {
        return "";
    }

    if (b[i][j] == 1) {
        return get_lcs(i - 1, j - 1, b, x) + x[i];  // 대각선 방향, 문자 포함
    } else if (b[i][j] == 2) {
        return get_lcs(i, j - 1, b, x);             // 왼쪽
    } else if (b[i][j] == 3) {
        return get_lcs(i - 1, j, b, x);             // 위쪽
    }

    return ""; // 혹시 모를 fallback
}
```



```cpp
// LCS (Longest Common Subsequence)
/* input case
ABCBDAB
BDCABA
*/

#include <iostream>
#include <vector>
#include <string>
using namespace std;

void lcs(const string& x, const string& y, vector<vector<int> >& c, vector<vector<int> >& b) {
    int m = x.length();
    int n = y.length();
    c.assign(m + 1, vector<int>(n + 1, 0));
    b.assign(m + 1, vector<int>(n + 1, 0));

    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (x[i - 1] == y[j - 1]) {
                c[i][j] = c[i - 1][j - 1] + 1;
                b[i][j] = 1;
            } else {
                if (c[i][j - 1] > c[i - 1][j]) {
                    c[i][j] = c[i][j - 1];
                    b[i][j] = 2;
                } else {
                    c[i][j] = c[i - 1][j];
                    b[i][j] = 3;
                }
            }
        }
    }
}

string get_lcs(int i, int j, const vector<vector<int> >& b, const string& x) {
    if (i == 0 || j == 0) return "";
    if (b[i][j] == 1)
        return get_lcs(i - 1, j - 1, b, x) + x[i - 1];
    else if (b[i][j] == 2)
        return get_lcs(i, j - 1, b, x);
    else
        return get_lcs(i - 1, j, b, x);
}

int main() {
    string x, y;
    cin >> x >> y;

    vector<vector<int> > c, b;
    lcs(x, y, c, b);

    string lcs_str = get_lcs(x.length(), y.length(), b, x);
    int LCS_len = lcs_str.length();

    if (LCS_len == 0) cout << 0 << endl;
    else {
        cout << LCS_len << endl;
        cout << lcs_str << endl;
    }

    return 0;
}
```