Problem Set #3 (Algorithms)

Department: 컴퓨터정보공학부

Student ID: 2015722025

Student Name: 정용훈

1. For a top-down memorized algorithm to compute the length of the LCS of x and y in Θ(mn) space →

(a) Write your program with your comments.

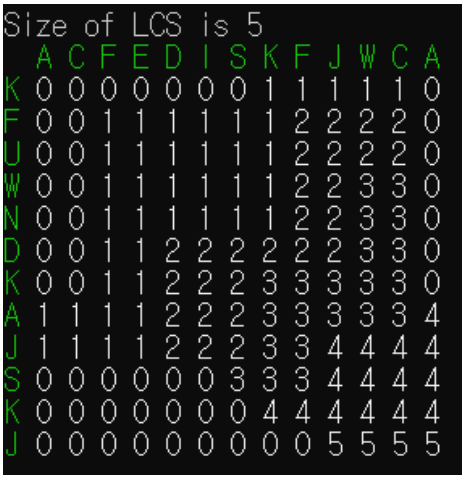
```
int LCS_top_down(string x, string y, int i, int j)
{
    if (i == -1 || j == -1) //For end working of function
        return 0;

    if (top_down_array[j][i] == 0) //checking visiting
    {
        if (x[i] == y[j]) //if text is equal
            top_down_array[j][i] = LCS_top_down(x, y, i - 1, j - 1) + 1;

        else if (x[i] != y[j]) //if text is not equal
            top_down_array[j][i] = //choose big value
            max(LCS_top_down(x, y, i, j - 1), LCS_top_down(x, y, i - 1, j));
    }
    return top_down_array[j][i]; //return size of LCS
}
```

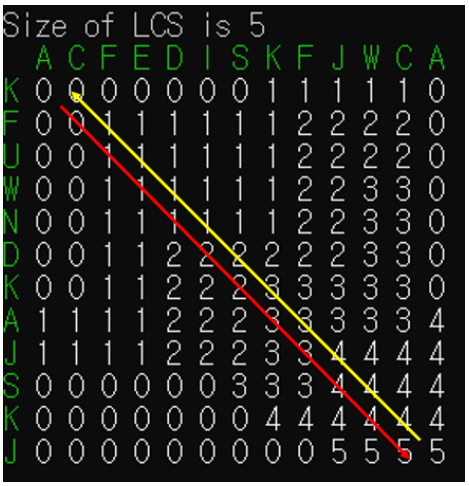
해당 code는 강의 자료를 참고하여 작성하였으며, top down방식의 동작을 한다. Top down이란 큰 문제를 작은 문제로 분할하여 계산하는 방법으로 재귀함수형태를 생각하면, 쉽게 이해할 수 있다. 문제를 작은 문제로 분할 하며, 호출의 끝을 알기 위해 I, j값이 음수가 되는 시점에 함수를 호출하지 않고 0을 반환해주며, **비교 문자가** 같은 경우는 **왼쪽 대각선 위 값을 구하는 함수를 호출하며, 더하기 1을** 해주는 모습이며, **다른 경우는 큰 값을 선택하여 해당 I, j 포인터를 다시 재귀함수를 통해 호출하는 것**을 확인할 수 있다. 그러면서 계산된 size를 계속 반환해주며, 최종적인 값을 도출할 수 있다.

(b) Show the results when you run your program for at least one example.



Program을 정상동작 시키면 다음과 같이 LCS에 대한 size를 구하며, 재귀함수를 실행하면서 구한 각각의 성분들을 모두 출력시킨 모습이다. 수업 이론과 강의자료 행과 열의 첫 번째 줄이 0으로 초기화 되어있는 것을 확인할 수 있는데 이는 bottom방식을 이용하였을 때 계산에 필요하기 때문에 존재하며, 재귀함수 방식을 사용하면, 조건에 따라 0을 반환하여 사용하면 되기 때문에 생략할 수 있다. 해당 항목은 size에 대한 출력이며, string을 구하는 항목은 3번에서 진행합니다.

(c) Explain your program and the results at least four lines.



우선 **노란색 화살표**는 호출되는 함수의 순서로써 가장 마지막 성분에서 string text에 따라 i, j값을 순차적으로 줄이며, 값을 return 받기 위해 왼쪽 위로 올라가며, 함수를 호출합니다. 그 후 text상태에 따라 i, j값과 return 되는 값이 결정되며, **빨간색 화살표** 방향으로 값들이 순차적으로 return 되며, 최종적으로 마지막 성분에는 LCS의 size가 저장되며, 이를 출력하면서 LCS의 사이즈가 도출된다. 반환되는 값은 string의 text를 비교하였을 때 같은 경우 왼쪽 대각선 위 값에서 +1을 반환하여 table에 저장하며, text가 다른 경우는 저장되는 성분 위치 기준으로 왼쪽 위쪽 성분을 비교하여 큰 값을 채택하여 반환 받는다.

2. For a bottom-up dynamic-programming algorithm to compute the length of LCS of x and y in Θ(mn) space →

(a) Write your pseudocode. Then, explain your pseudocode at least four lines.

Bottom_up function pseudocode

```
// Create table for saving value  //

For i ← 0 to length of string y
        For j ← to length of string x
        {
                If i or j is 0
                        table (i, j) ← 0;
                Else if x[j-1] is equal y[i-1]
                        table (i, j) ← Above the left diagonal  + 1;
                Else
                        table (i, j) ← Left or upper, whichever is greater;
        }

//              Print table             //
```

다음은 이번 과제에 사용된 의사코드로 동작은 다음과 같다. 우선 값들을 저장할 수 있는 table을 생성한 후 반복 조건을 통하여, 한 줄씩 계산을 순차적으로 하는데 i, j 값이 0이면, 해당 성분은 0으로 초기화 시켜주면 아닌 경우는 각 x, y string비교에 따라 값을 순차적으로 채워준다. 만약 i, j에 따라 x와 y의 text가 같다면 왼쪽 위 대각선 값에 1을 더한 값을 table i, j pointer에 해당하는 위치에 값을 저장시켜주며, 그렇지 않은 경우 save되는 위치 기준에서 왼쪽 값과 위쪽 값을 비교하여, 큰 값을 해당 위치에 저장시킨다. 후에 print함수를 통하여 table을 출력한다. 이렇게 x, y string에 대한 모든 비교와 조건에 맞게 table을 채우면 마지막 성분에는 LCS에 size가 저장되며, 정확한 결과는 아래 항목에서 결과 콘솔과 설명으로 확인할 수 있다.

(b) Write your program with your comments.

```
int LCS_bottom_up(string x, string y, int i, int j)
{
    ///////////////  Create space  ///////////////
    int** c = new int* [y.size()+1];
    for (int i = 0; i <= y.size(); i++)
        c[i] = new int[x.size()+1];
    ////////////////////////////////////////////

    ////////    Fill table with value    ////////
    for (int i = 0; i <= y.size(); i++)
        for (int j = 0; j <= x.size(); j++)
        {
            if (i == 0 || j == 0) //Initialization
                c[i][j] = 0;
            else if (x[j-1] == y[i-1]) //If text is equal
                c[i][j] = c[i - 1][j - 1] + 1;
            else if (x[j-1] != y[i-1]) //If text is not equal
                c[i][j] = max(c[i - 1][j], c[i][j - 1]); //choose max
        }
    ////////////////////////////////////////////
    print_array(x,y,c,2);

    int ans = c[j][i]; //return size of LCS
    delete[] c; //delete table

    return ans;
}
```

Bottom up이란 작은 문제를 풀어 큰 문제에 대한 최종적인 풀이를 도출하는 것이다. 우선 인자로 받는 각 string size만큼의 공간을 생성해주며, 한 칸씩 LCS를 구하는 알고리즘을 적용하여, 순차적으로 모든 table을 채워 table마지막 칸에는 LCS의 size를 도출하도록 되어있다. LCS를 구하는 알고리즘은 강의 자료 table과 1번 문항에서 사용한 알고리즘을 적용하여, i와 j에 의해 선정된 칸의 string 문자를 각각 비교하여 같으면 왼쪽 대각선 위 값에 1을 더한 값을 저장해주고, 다르면 왼쪽, 위 칸의 값을 비교하여, 더 큰 값을 저장하게 된다.

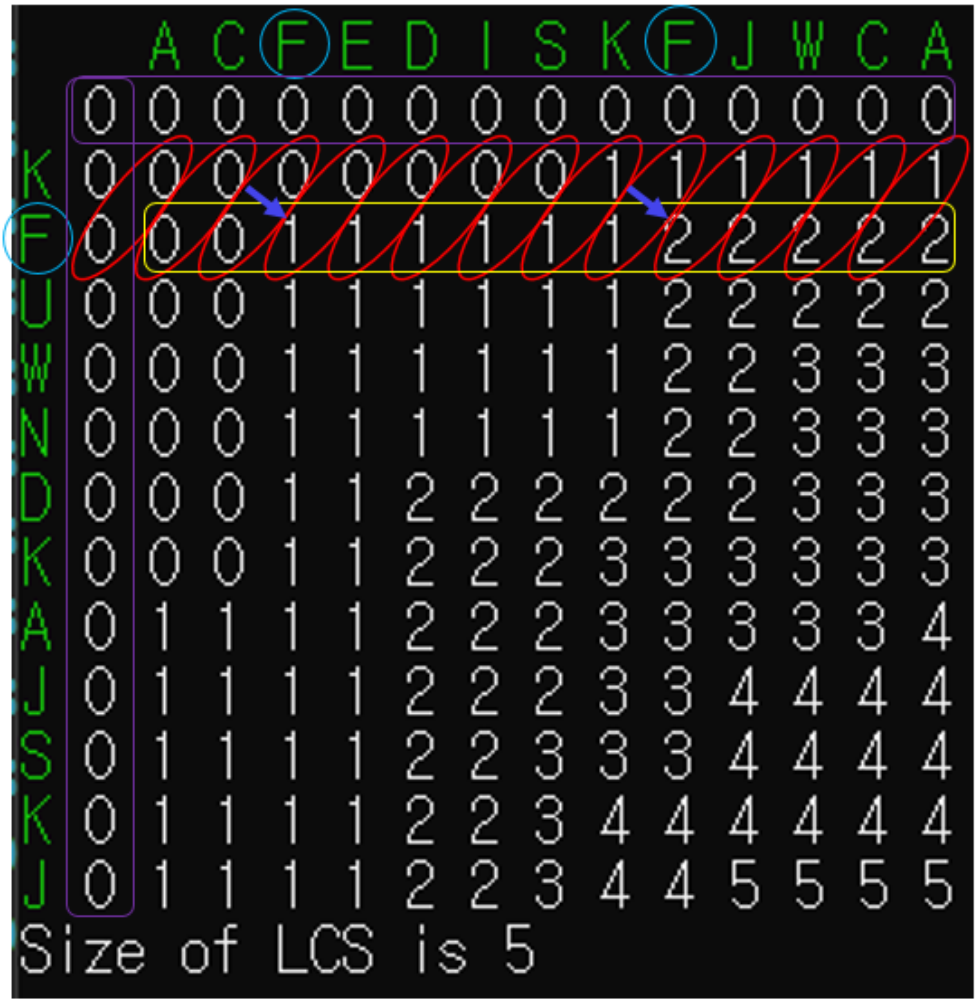
(c) Show the results when you run your program for at least one example.

```
    A C F E D I S K F J W C A
  0 0 0 0 0 0 0 0 0 0 0 0 0
K 0 0 0 0 0 0 0 0 1 1 1 1 1
F 0 0 0 1 1 1 1 1 1 2 2 2 2
U 0 0 0 1 1 1 1 1 1 2 2 2 2
W 0 0 0 1 1 1 1 1 1 2 2 3 3
N 0 0 0 1 1 1 1 1 1 2 2 3 3
D 0 0 0 1 1 2 2 2 2 2 2 3 3
K 0 0 0 1 1 2 2 2 3 3 3 3 3
A 0 1 1 1 1 2 2 2 3 3 3 3 4
J 0 1 1 1 1 2 2 2 3 3 4 4 4
S 0 1 1 1 1 2 2 3 3 3 4 4 4
K 0 1 1 1 1 2 2 3 4 4 4 4 4
J 0 1 1 1 1 2 2 3 4 4 5 5 5
Size of LCS is 5
```

Bottom up을 통해 문자열 "ACFEDISKFJWCA"와 "KFUWNDKAJSKJ"의 LCS size를 구하기 위해 table을 채운 후 구현한 print함수를 통해 table을 출력한 결과 화면이며, 최종적으로 배열의 마지막 성분을 보면, 5라는 결과를 확인할 수 있다.

(d) Explain your program and the results at least four lines.

Bottom up이란 위에서 언급한 것과 같이 각각 작은 문제를 풀어 큰 문제의 답을 도출하는 방법으로, program의 동작은 table을 통해 다음과 같이 설명할 수 있다.



위에 표시한 것과 같이 보라색 네모는 table의 첫째 줄들을 0으로 초기화한 상태로 확인할 수 있으며, 실질적인 계산은 그 안쪽 table부터 시작된다. 계산은 계산 대상이 되는 노란색 네모원소들로 기준을 잡았을 때 왼쪽에서부터 해당하는 순서의 **문자를 비교**하여 **다르면 왼쪽 원소와, 위쪽 원소를 비교하여 큰 값을 Save**하며, **문자가 같으면 왼쪽 대각선 위 성분에 +1을 해준 값**을 Save하게 된다. 결과의 예시를 확인하면, y축 **F**와 각각의 문자 x축 A, C, F, E D, I, S, K, F, J, W, C, A를 비교하는데 같은 경우 왼쪽 대각선 위에서 값이 1이 더해진 것을 확인할 수 있으며, 아닌 경우 비교하여 큰 값이 순차적으로 저장되는 것을 확인할 수 있다.

3. For an algorithm to print an LCS of x and y in Θ(m+n) time by using the results of Problem 2 →

(a) Write your pseudocode. Then, explain your pseudocode at least four lines.

For printing LCS string

```
//   Same as problem 2 for making table   //

While i is not 0 and j is not 0
{
        Create list for history of path;

        If y's text is equal x's text
                → save text;
                  move pointer to Above the left diagonal;
        Else
        {
                If above value is bigger than left value or equal
                        → move pointer to above;
                else
                        → move pointer to left;
        }
}
//          Print table          //
```

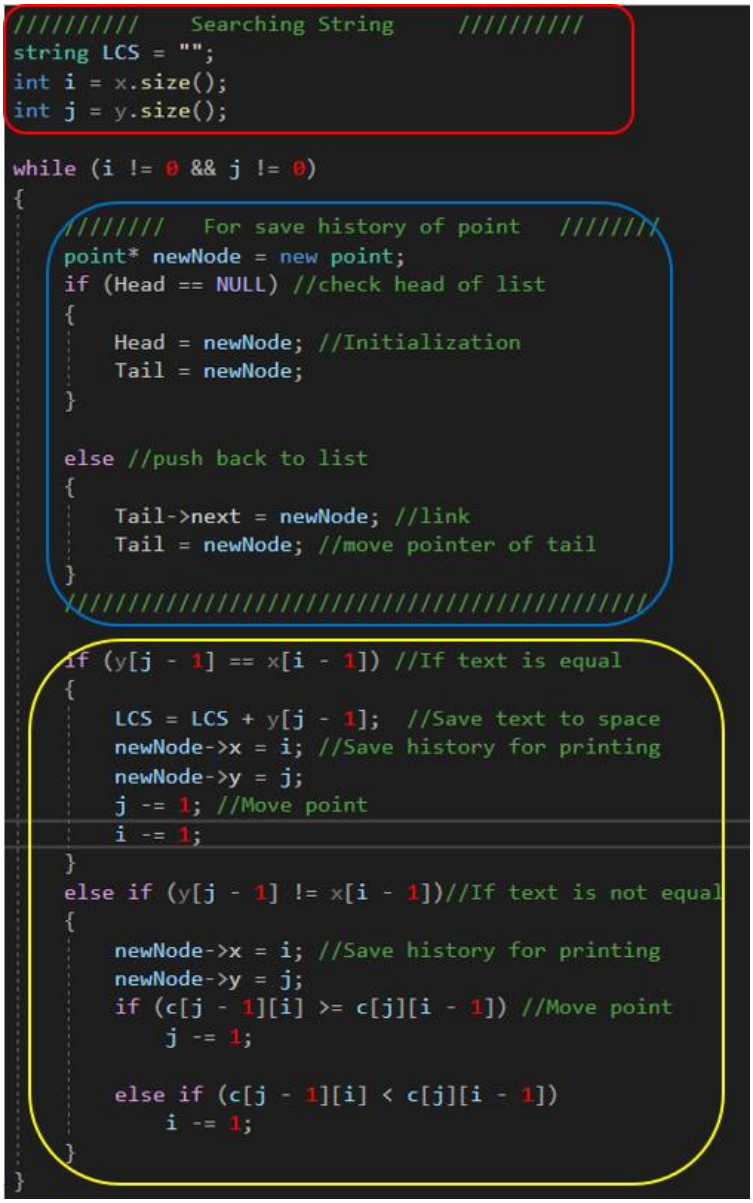
위 의사 code는 problem2를 통해 생성된 table을 기반으로 실제 LCS string에 대한 one example에 해당하는 문자열 출력을 위한 코드입니다. 우선 problem2에서 와 마찬가지로 table생성과 value를 채워 준비한 후 i, j값이 0이 아닌 경우 까지만 반복하여, detection을 실행합니다. (i, j 값은 table 가장 마지막 원소에서부터 시작합니다.) table에 값을 저장 할 때와 같은 방법으로 함수가 실행되며, 가장 마지막 원소부터 시작하여, 비교 text가 같은 경우는 왼쪽 대각선 위로 pointer를 이동시키며, **해당 text를 출력하기 위한 string에 따로 저장**합니다. (문자열이 같은 경우 LCS string의 성분이 된다.) 만약 다르다면, 왼쪽과 위쪽 값을 비교하여 큰 값으로 pointer를 이동시키며, 임의 적으로 본인은 같은 값인 경우 위쪽으로 pointer를 이동시켰습니다. i, j값에 의해 모든 detection이 끝났다면 의사 code에 의해 history정보를 기반으로 table을 출력하며, 함수를 종료합니다.

(b) Write your program with your comments.

```
void print_LCS(string x, string y)
{
    ///////////////  Create space  ///////////////
    int** c = new int* [y.size() + 1];
    for (int i = 0; i <= y.size(); i++)
        c[i] = new int[x.size() + 1];
    ////////////////////////////////////////////

    /////////////    Create table    /////////////
    for (int i = 0; i <= y.size(); i++)
        for (int j = 0; j <= x.size(); j++)
        {
            if (i == 0 || j == 0) //Initialization
                c[i][j] = 0;
            else if (x[j - 1] == y[i - 1]) //If text is equal
                c[i][j] = c[i - 1][j - 1] + 1;
            else if (x[j - 1] != y[i - 1]) //If text is not equal
                c[i][j] = max(c[i - 1][j], c[i][j - 1]);
        }
    ////////////////////////////////////////////
```

String을 출력하는 처음 동작은 problem2에서 table을 만드는 과정과 똑같으며, 해당 과정을 거치고 나면 string을 출력하기 위한 back trace동작을 하는 부분으로 넘어 갑니다. 정확한 detection동작은 아래와 같습니다.



위 과정에서 table을 작성 후 만들어진 table에서 수업시간에 배운 내용을 기반으로 back trace방법을 사용하여, LCS에 해당하는 one example을 출력하기 위한 동작이다. 간단하게 설명하면, **빨간색 네모** 부분은 LCS string을 받기 위한 공간 선언과 반복 동작을 하기 위한 string의 size를 변수로 받으며 동작에 대한 준비를 하는 과정이며, **파란색 네모**는 후에 출력을 위한 history를 저장하기 위해 linked list를 사용하기 위한 선언이다. 다음으로 **노란색 네모**는 해당 과제에서 요구하는 LCS string을 직접 찾는 함수이다. 노란색 네모를 보면 과제에서 요구하는 $\Theta(m+n)$만큼 걸리는데 이에 해당하는 정확한 설명은 아래 항목에서 진행하겠습니다.

```
    ////////////////////////////////////////////
    cout << "----View of LCS----" << endl;
    print_array(x, y, c, 3);
    cout << "-------------------" << endl;
    reverse(LCS.begin(), LCS.end());
    cout << "LCS is --> " << LCS << endl;
    ////////////////////////////////////////////

    delete[] c;
    return;
}
```

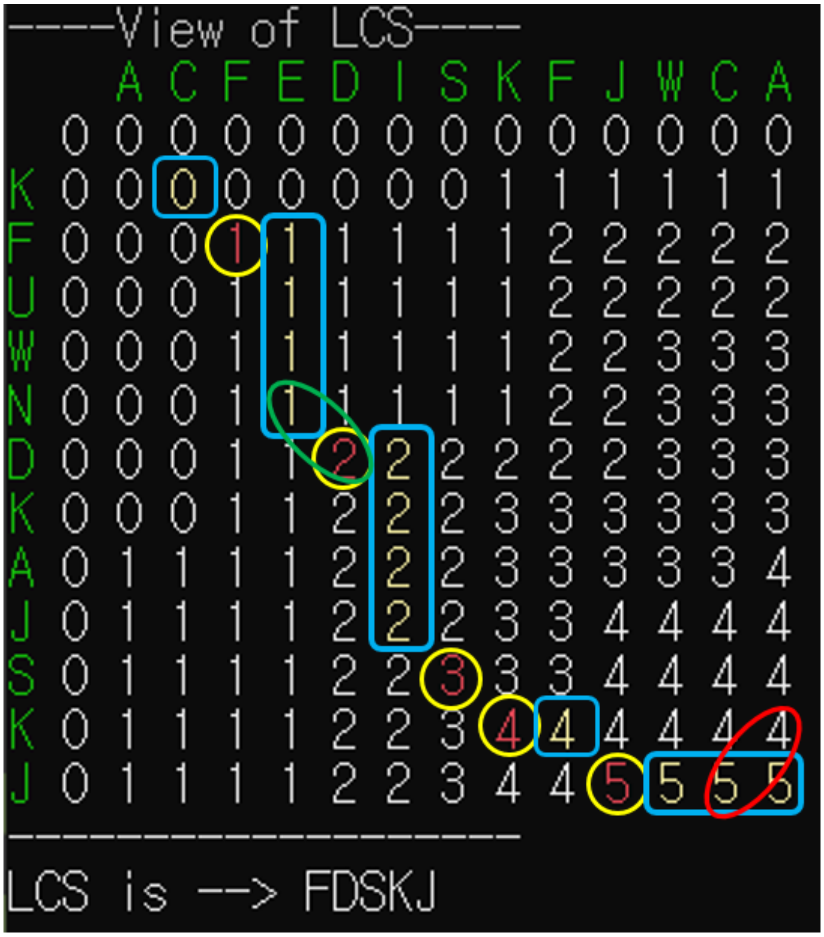
다음은 LCS를 구하는 과정을 table을 통해 나타내기 위한 출력 부분이며, print array 함수를 통해 table을 출력하고, string이 저장된 변수를 뒤에서부터 저장했으므로 반전시킨 후 출력하면 string을 도출할 수 있습니다.

(c) Show the results when you run your program for at least one example.

```
----View of LCS----
    A C F E D I S K F J W C A
  0 0 0 0 0 0 0 0 0 0 0 0 0 0
K 0 0 0 0 0 0 0 0 1 1 1 1 1 1
F 0 0 0 1 1 1 1 1 1 2 2 2 2 2
U 0 0 0 1 1 1 1 1 1 2 2 2 2 2
W 0 0 0 1 1 1 1 1 1 2 2 3 3 3
N 0 0 0 1 1 1 1 1 1 2 2 3 3 3
D 0 0 0 1 1 2 2 2 2 2 2 3 3 3
K 0 0 0 1 1 2 2 2 3 3 3 3 3 3
A 0 1 1 1 1 2 2 2 3 3 3 3 3 4
J 0 1 1 1 1 2 2 2 3 3 4 4 4 4
S 0 1 1 1 1 2 2 3 3 3 4 4 4 4
K 0 1 1 1 1 2 2 3 4 4 4 4 4 4
J 0 1 1 1 1 2 2 3 4 4 5 5 5 5
-------------------
LCS is --> FDSKJ
```

다음 결과는 3번문제에 해당하는 함수를 실행시켰을 때 도출되는 화면이다. 마지막으로 출력되는 부분에서 back trace를 시작해 문자열의 공집합 부분이 되기 전 까지 text를 비교하며, LCS를 찾아가는 것을 확인할 수 있다. 프로그램의 정확한 동작은 아래 항목에서 설명하겠습니다.

(d) Explain your program and the results at least four lines.



Back trace는 마지막 text에서 시작하며, 다음과 같은 과정을 진행합니다. 우선 **파란색 네모**는 비교 string의 text가 같지 않은 경우로 해당경우에는 기준이 되는 원소에서 **빨간색 동그라미**와 같은 방향으로 값을 비교하여 큰 값의 원소로 pointer(I, j값에 의한 위치)로 이동하게 됩니다. (one example에 대한 예시이므로 값이 같은 경우에는 Pointer가 위로 이동하게 구현하였습니다.) **노란색 동그라미**는 비교 문자열의 text가 같은 경우로 해당 경우에는 **초록색 동그라미**와 같이 왼쪽 대각선 위로 Pointer가 이동하도록 하며, String의 공집합 부분인 맨 앞에 도달하게 되면 동작이 끝이 납니다. LCS같은경우 비교 문자열의 text가 동일할 때 string에 저장함으로 각 노란색 동그라미가 최종적으로 도출되는 LCS string에 해당이 되며, 이는 위 항목에서 설명한 함수를 이용하여 반전을 통해 출력할 수 있습니다.

마지막으로 Θ(m+n)의 시간 복잡도를 도출할 수 있는데 이는 다음과 같은 예시로 설명할 수 있습니다.

문자열이 다음과 같은 경우, 위 항목에서 제시된 one example에 관련된 도출방법은 한가지 case에 대한 결과를 도출하는 것이므로, 모든 LCS를 구해도 되지 않는 경우이기 때문에 대각선 값을 비교하여 같은 **경우 위로 올라가게 본인이 스스로 다음 코드와 같이 구현한 것입니다.**

```
if (c[j - 1][i] >= c[j][i - 1]) //Move point
    j -= 1;
```

하지만 좀 더 정확히 LCS를 구현하기 위해서는 대각선의 값이 같게 된다면, **두 가지 path의 경우가 생기는 것**이므로 여러 가지 case에 대한 LCS string이 나오기 때문에 실질적으로는 동작을 분리시켜줘야 합니다. 그렇기에 모든 case를 고려한다면, worst case같은 경우 위와 같이 m+n번의 비교 절차를 거쳐야 하며, 이는 string을 출력하기 위한 시간 복잡도가 Θ(m+n)이라는 것을 쉽게 알 수 있습니다.

4. Reference

- ✓ Dynamic programming

  강의 자료 & 온라인 강의

- ✓ Bottom up 과 Top down에 대한 개념과 예시

  https://semaph.tistory.com/16

강의자료와 수업 참고를 통한 구현이 수월한 과제였습니다.