Shortest Common Supersequence

Mahdis Rahmani - Spring 2021

Contents:

-Defining LCS and SCS

-Considerations of the problem

-**Recursive (Brute Force) approach - TC: O(2^Min(m,n))**

-Dynamic Programming approach - TC: O(m*n)

-Printing SCS for multiple inputs

LCS: Longest Common Subsequence

Given two sequences, print all longest subsequence present in both of them.

X = "opengenus"
Y = "operagenes"

Z = "opegens"

X = "AGGTAB"
Y = "GXTXAYB"

Z = "GTAB"

SCS: Shortest Common Supersequence

Given two strings str1 and str2, the task is to find the length of the shortest string that has both str1 and str2 as subsequences.

X = "opengenus"
Y = "operagenes"

z = "openragenues"

X = "AGGTAB"
Y = "GXTXAYB"

Z = "AGXGTXAYB"

Considerations:

- Difference between subsequence and substring

  String 1 = Frankenstein

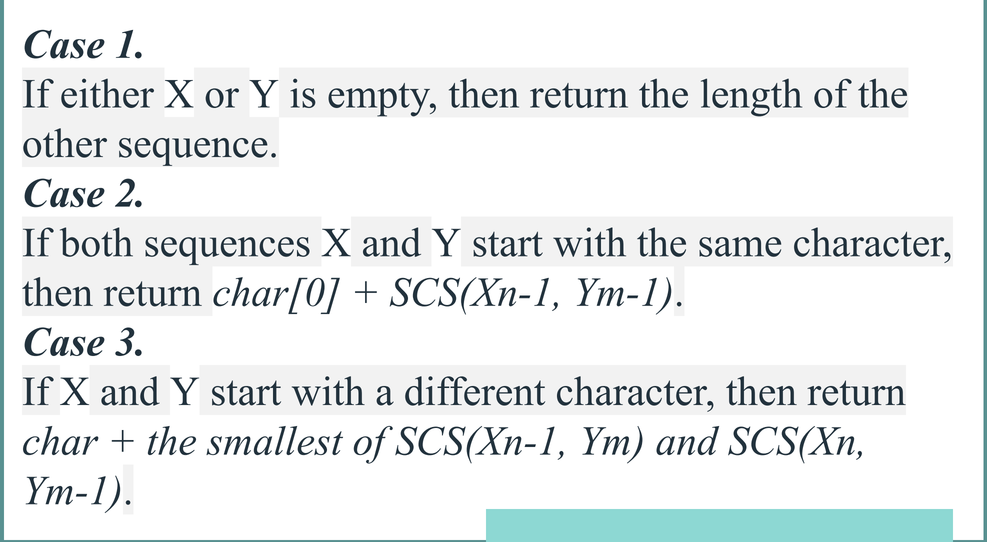
  Substring= rank, enstein
  Subsequence = rnk, esi

X = "AGGTAB"
Y = "GXTXAYB"

Z = "AGXGTXAYB"

SCSthLeng(X, Y) = m + n – LCSLength(X, Y)

-Recursive (Brute Force) approach - TC: O(2^Min(m,n))



Case 1.

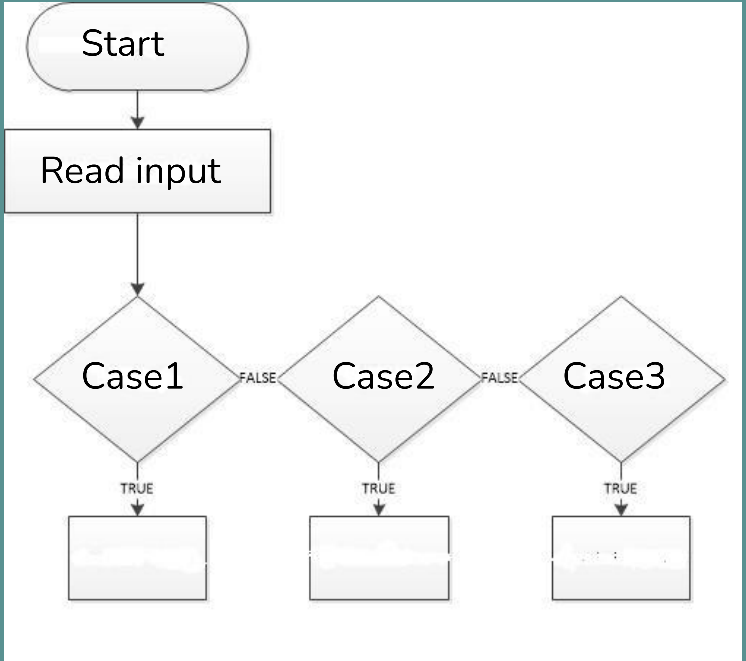
If either X or Y is empty, then return the length of the other sequence.

Case 2.

If both sequences X and Y start with the same character, then return *char[0] + SCS(Xn-1, Ym-1)*.

Case 3.

If X and Y start with a different character, then return *char + the smallest of SCS(Xn-1, Ym) and SCS(Xn, Ym-1)*.

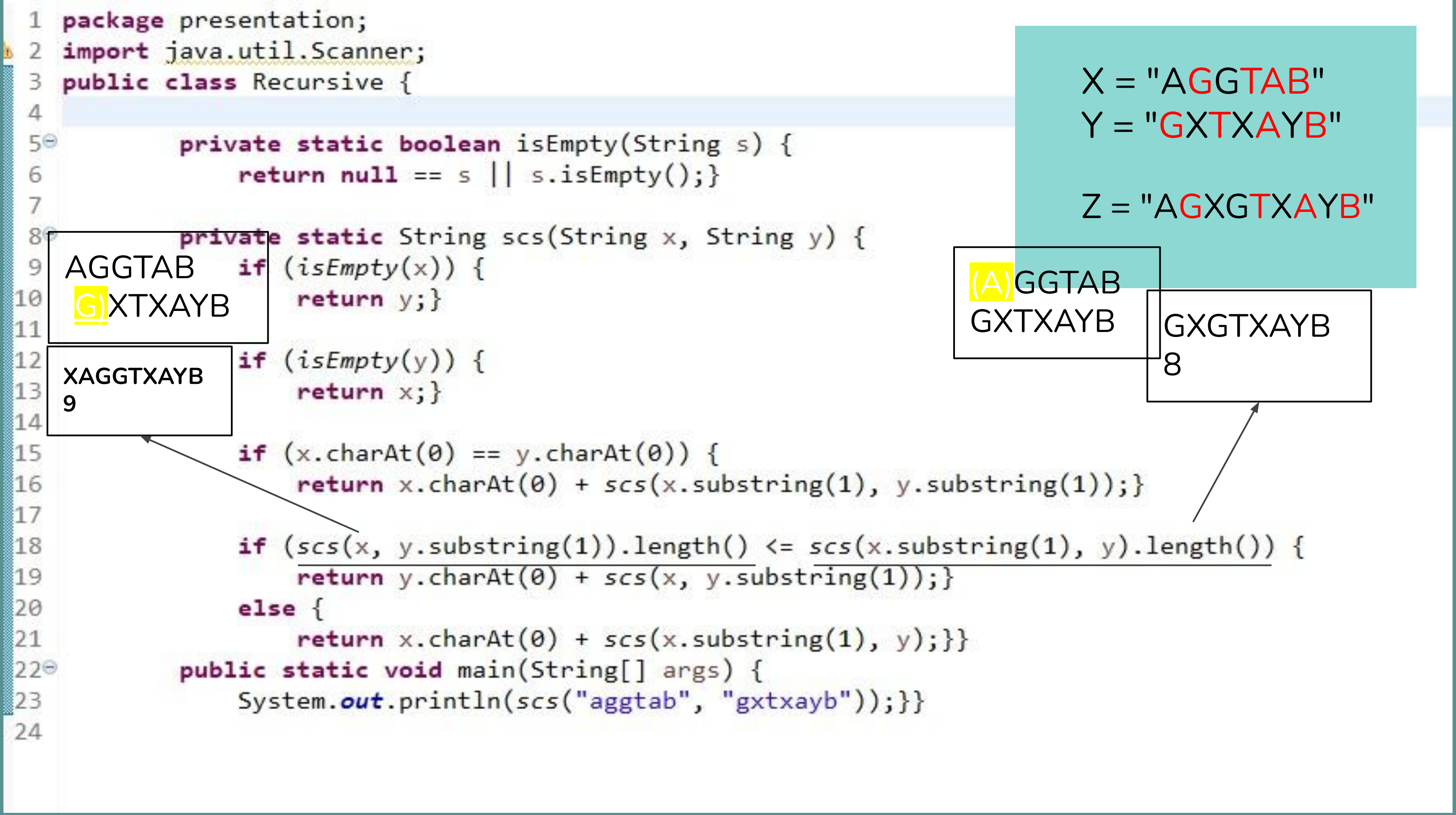


X = "AGGTAB"
Y = "GXTXAYB"

Z = "AGXGTXAYB"

X = "AGGTAB"
Y = "GXTXAYB"

Z = "AGXGTXAYB"



```
package presentation;
import java.util.Scanner;
public class Recursive {

        private static boolean isEmpty(String s) {
            return null == s || s.isEmpty();}

        private static String scs(String x, String y) {
            if (isEmpty(x)) {
                return y;}

            if (isEmpty(y)) {
                return x;}

            if (x.charAt(0) == y.charAt(0)) {
                return x.charAt(0) + scs(x.substring(1), y.substring(1));}

            if (scs(x, y.substring(1)).length() <= scs(x.substring(1), y).length()) {
                return y.charAt(0) + scs(x, y.substring(1));}
            else {
                return x.charAt(0) + scs(x.substring(1), y);}}
        public static void main(String[] args) {
            System.out.println(scs("aggtab", "gxtxayb"));}}
```

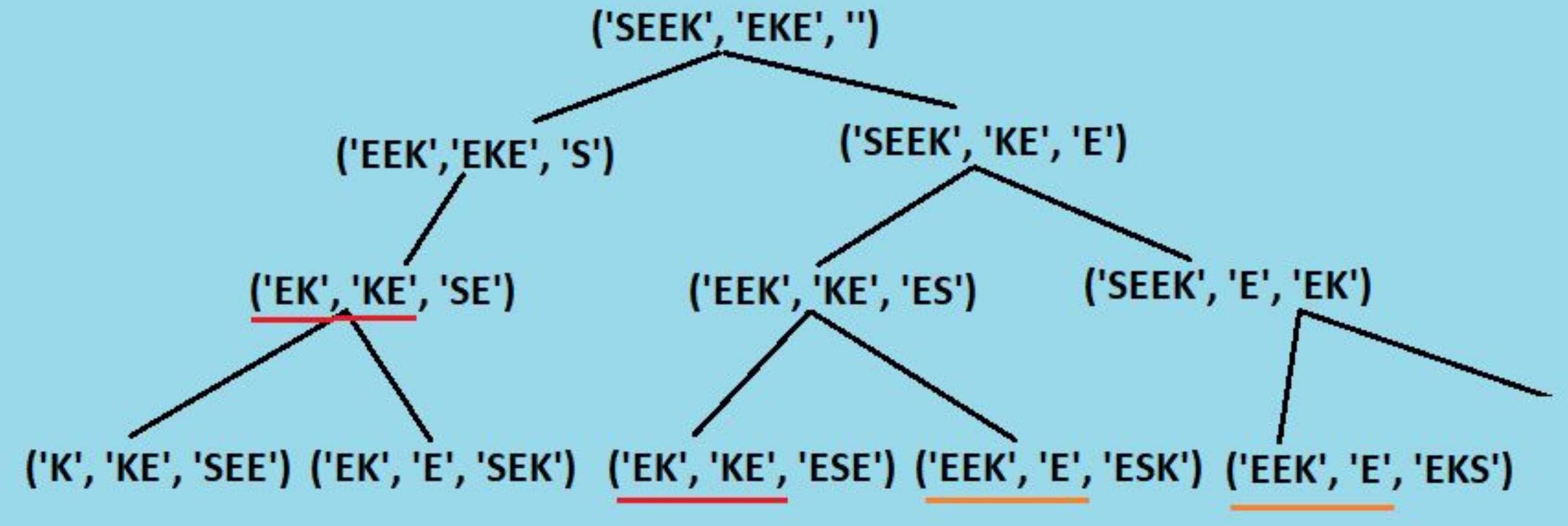
AGGTAB
G)XTXAYB

XAGGTXAYB
9

(A)GGTAB
GXTXAYB

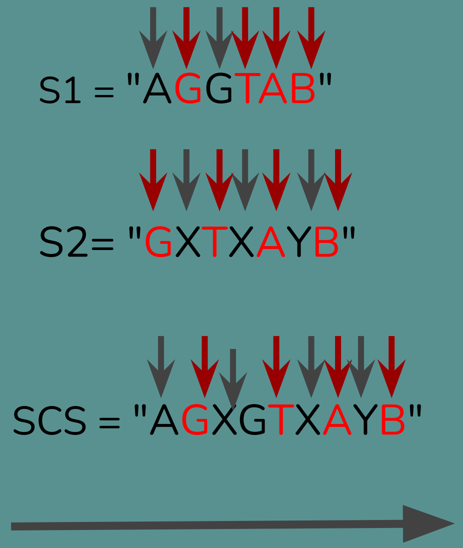
GXGTXAYB
8

Recursive approach leads to overlapping problem:



Partial Recursion Tree of "SEEK" and "EKE" with repeated subproblems

-Dynamic Programming approach - TC: O(m*n)



S1 = "AGGTAB"

S2= "GXTXAYB"

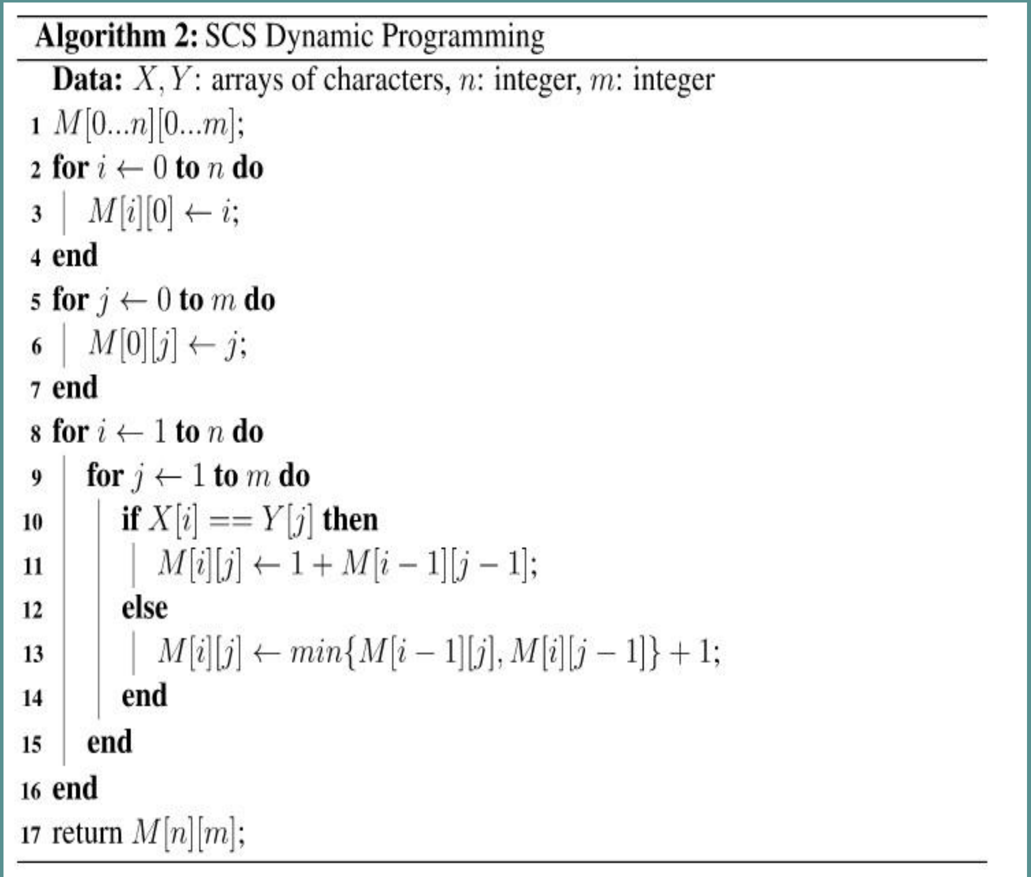
SCS = "AGXGTXAYB"

LCS = “GTAB”

- Add LCS chars only one
- The first common char belongs to SCS
- Add non-lCS chars in order

	Y	A	B	C
X	0 (00)	1 (01)	2 (02)	3 (03)
A	1 (10)	1 (11)	2 (12)	3 (13)
D	2 (20)	2 (21)	3 (22)	4 (23)

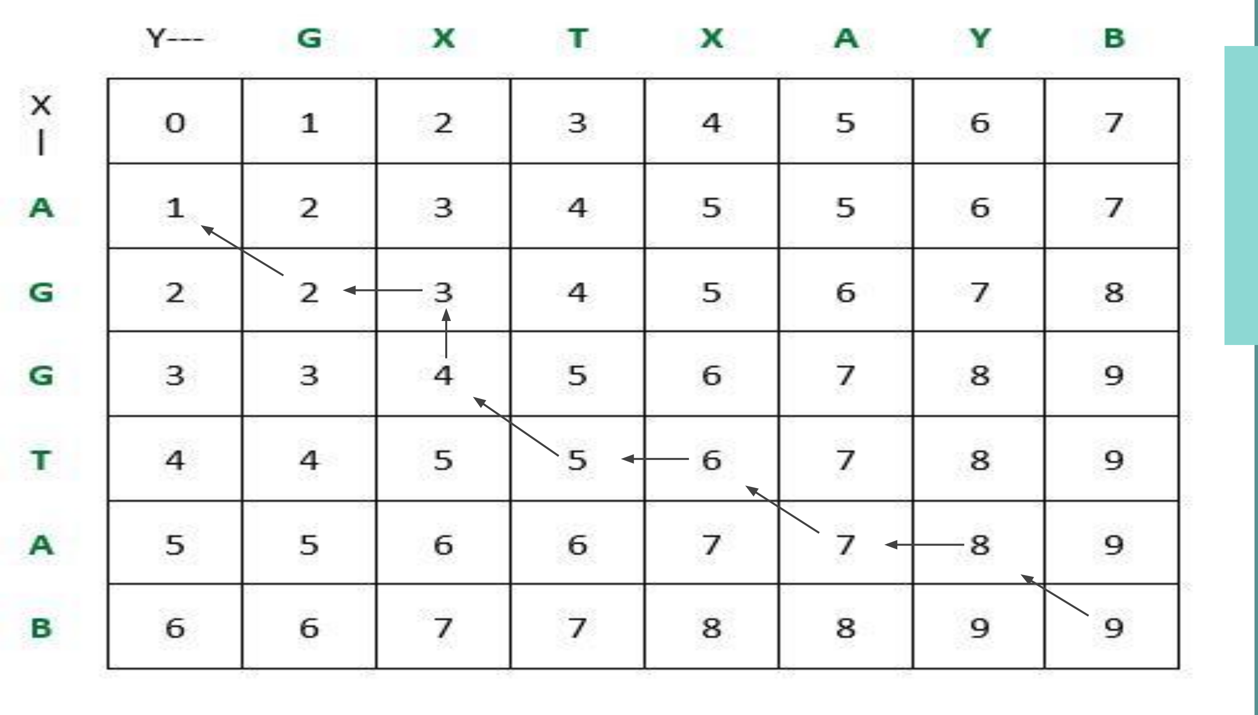
X = “AD”
Y = “ABC”
Output = “ADBC”



Algorithm 2: SCS Dynamic Programming

```
Data: $X, Y$: arrays of characters, $n$: integer, $m$: integer
$M[0...n][0...m]$;
for $i \leftarrow 0$ to $n$ do
    $M[i][0] \leftarrow i$;
end
for $j \leftarrow 0$ to $m$ do
    $M[0][j] \leftarrow j$;
end
for $i \leftarrow 1$ to $n$ do
    for $j \leftarrow 1$ to $m$ do
        if $X[i] == Y[j]$ then
            $M[i][j] \leftarrow 1 + M[i-1][j-1]$;
        else
            $M[i][j] \leftarrow min\{M[i-1][j], M[i][j-1]\} + 1$;
        end
    end
end
return $M[n][m]$;
```

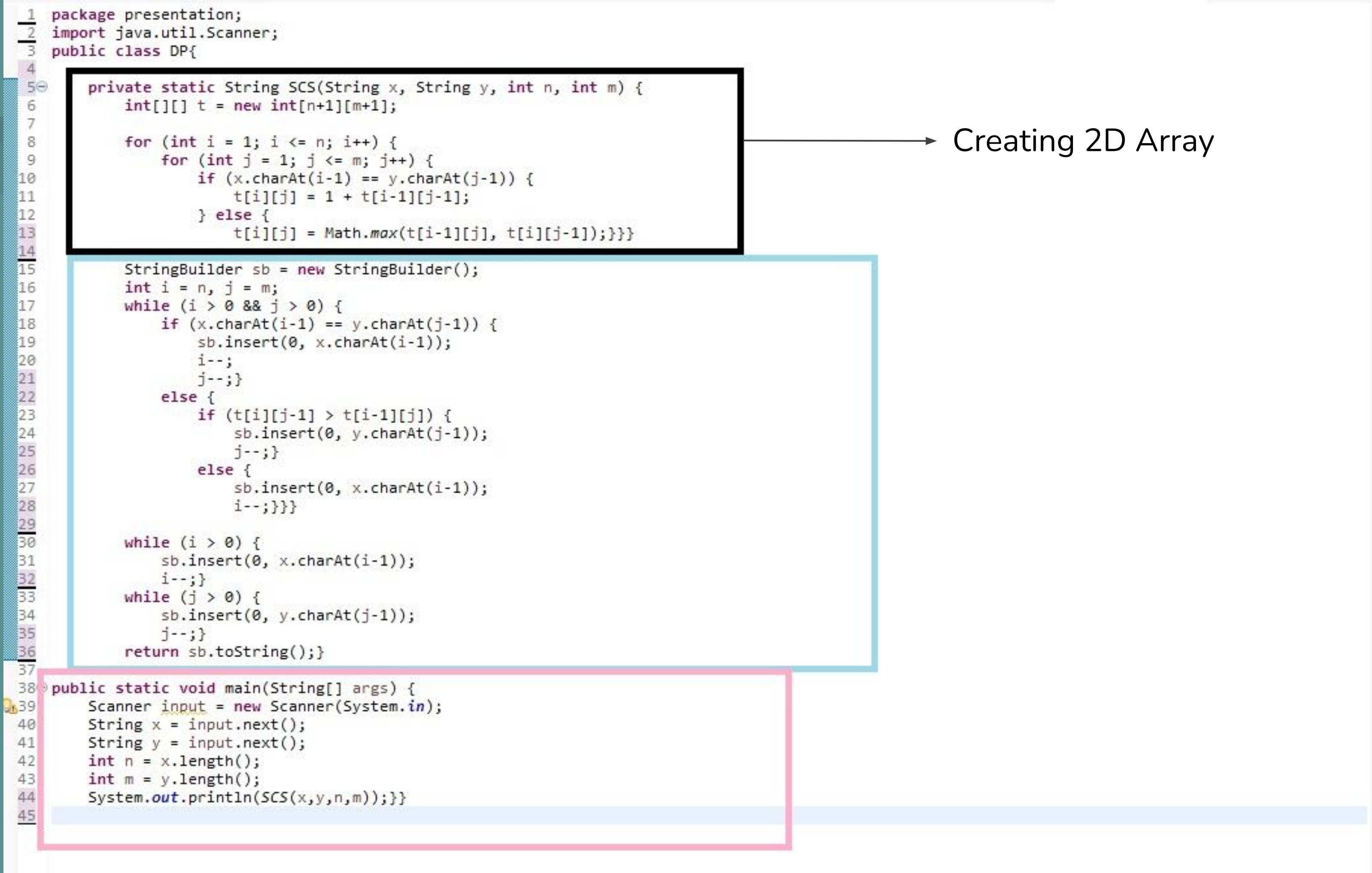
-Dynamic Programming approach - TC: O(m*n)



	Y---	G	X	T	X	A	Y	B
X |	0	1	2	3	4	5	6	7
A	1	2	3	4	5	5	6	7
G	2	2	3	4	5	6	7	8
G	3	3	4	5	6	7	8	9
T	4	4	5	5	6	7	8	9
A	5	5	6	6	7	7	8	9
B	6	6	7	7	8	8	9	9

X = "AGGTAB"
Y = "GXTXAYB"

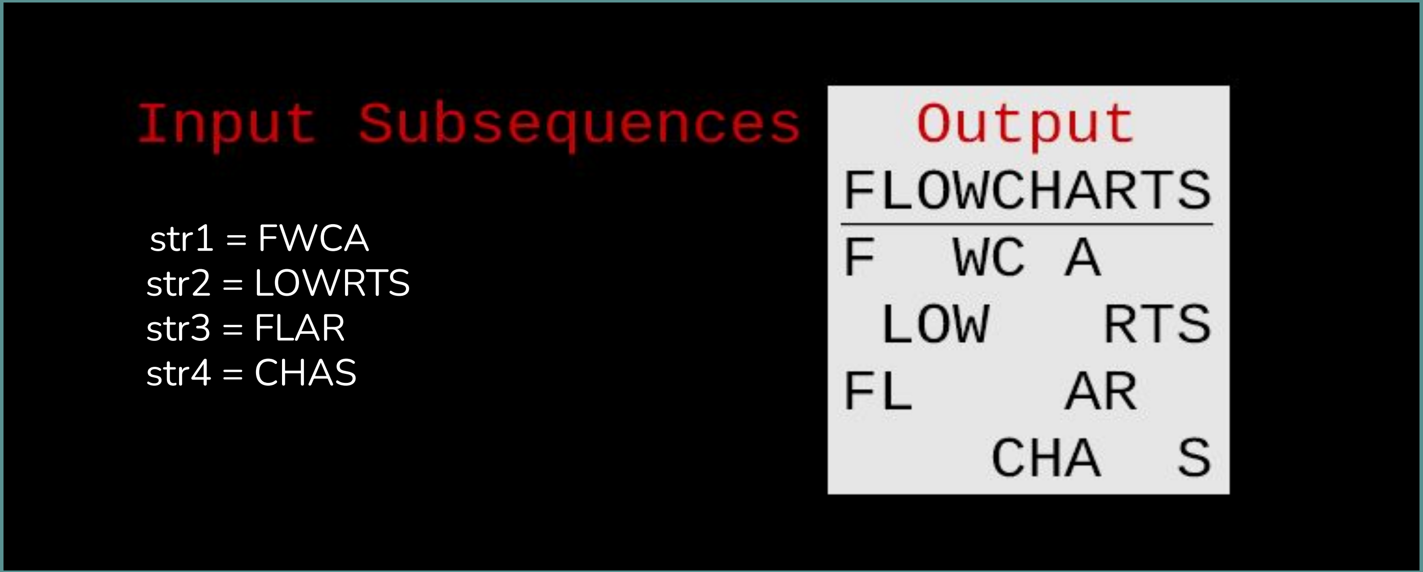
Z = "AGXGTXAYB"



```java
package presentation;
import java.util.Scanner;
public class DP{

    private static String SCS(String x, String y, int n, int m) {
        int[][] t = new int[n+1][m+1];

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

        StringBuilder sb = new StringBuilder();
        int i = n, j = m;
        while (i > 0 && j > 0) {
            if (x.charAt(i-1) == y.charAt(j-1)) {
                sb.insert(0, x.charAt(i-1));
                i--;
                j--;}
            else {
                if (t[i][j-1] > t[i-1][j]) {
                    sb.insert(0, y.charAt(j-1));
                    j--;}
                else {
                    sb.insert(0, x.charAt(i-1));
                    i--;}}}

        while (i > 0) {
            sb.insert(0, x.charAt(i-1));
            i--;}
        while (j > 0) {
            sb.insert(0, y.charAt(j-1));
            j--;}
        return sb.toString();}

public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    String x = input.next();
    String y = input.next();
    int n = x.length();
    int m = y.length();
    System.out.println(SCS(x,y,n,m));}}
```

Creating 2D Array

Finding SCS of more than two strings:



Input Subsequences

str1 = FWCA
str2 = LOWRTS
str3 = FLAR
str4 = CHAS

Output

```
FLOWCHARTS
----------
F  WC A
 LOW   RTS
FL    AR
    CHA  S
```

Thank you for your ATTENTION.