Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

60009210033

Jhanvi Parekh

CSE(Data Science)

Department of Computer Science and Engineering (Data Science)

Experiment 8 (Dynamic Programming)

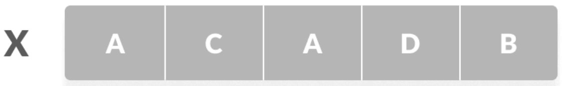
Aim: Implementation of Longest Common Subsequence (LCS).

Theory:

The longest common subsequence (LCS) is defined as the longest subsequence that is common to all the given sequences, provided that the elements of the subsequence are not required to occupy consecutive positions within the original sequences.

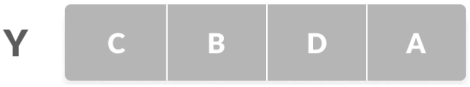
If S1 and S2 are the two given sequences then, Z is the common subsequence of S1 and S2 if Z is a subsequence of both S1 and S2. Furthermore, Z must be a strictly increasing sequence of the indices of both S1 and S2.
In a strictly increasing sequence, the indices of the elements chosen from the original sequences must be in ascending order in Z.
If: S1 = {B, C, D, A, A, C, D}
Then, {A, D, B} cannot be a subsequence of S1 as the order of the elements is not the same (ie. not strictly increasing sequence).

Example:



X	A	C	A	D	B

Figure 1. The first sequence

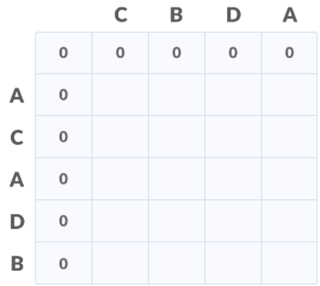


Y	C	B	D	A

Figure 2.Second Sequence

The following steps are followed for finding the longest common subsequence.

1. Create a table of dimension n+1*m+1 where n and m are the lengths of X and Y respectively. The first row and the first column are filled with zeros.

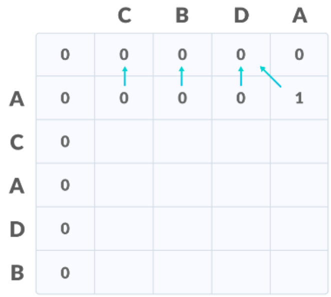


		C	B	D	A
	0	0	0	0	0
A	0				
C	0				
A	0				
D	0				
B	0				

Figure 3.Initialise a table

Department of Computer Science and Engineering (Data Science)

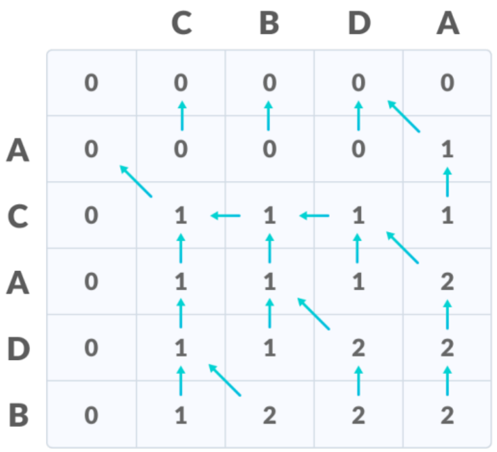
2. Fill each cell of the table using the following logic.
3. If the character correspoding to the current row and current column are matching, then fill the current cell by adding one to the diagonal element. Point an arrow to the diagonal cell.
4. Else take the maximum value from the previous column and previous row element for filling the current cell. Point an arrow to the cell with maximum value. If they are equal, point to any of them.



		C	B	D	A
	0	0	0	0	0
A	0	↑ 0	↑ 0	↑ 0	↖ 1
C	0				
A	0				
D	0				
B	0				

Figure 4.Fill the values 5.

Step 2 is repeated until the table is filled.

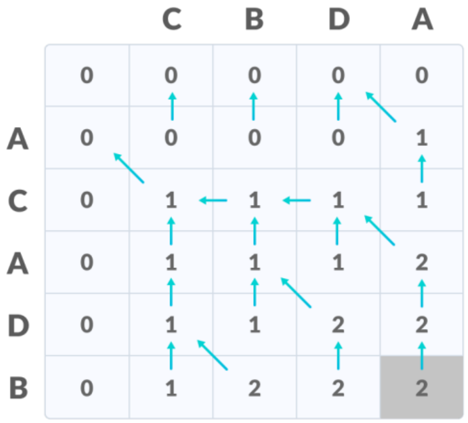


		C	B	D	A
	0	0	0	0	0
A	0	↑ 0	↑ 0	↑ 0	↖ 1
C	0	↖ 1	← 1	← 1	↑ 1
A	0	↑ 1	↑ 1	↑ 1	↖ 2
D	0	↑ 1	↑ 1	↖ 2	↑ 2
B	0	↑ 1	↖ 2	↑ 2	↑ 2

Figure 5.Fill all the values

6. The value in the last row and the last column is the length of the longest common subsequence.

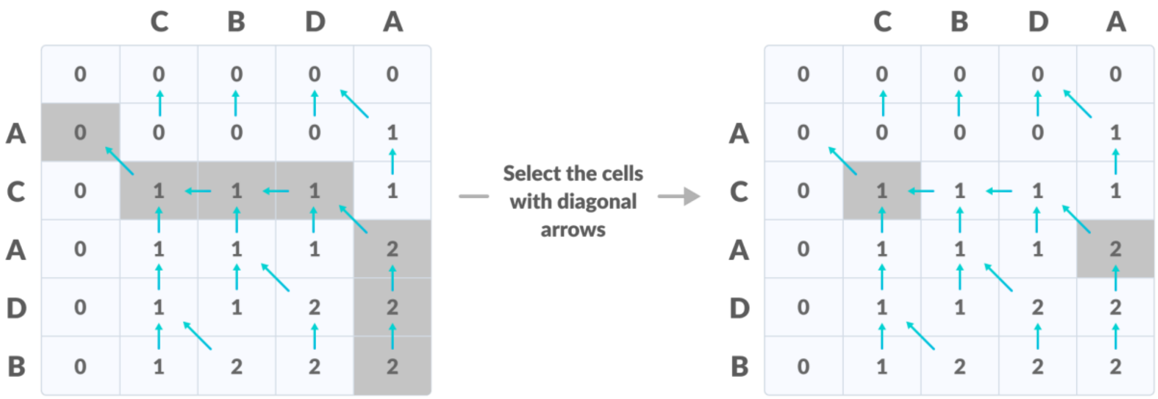
Department of Computer Science and Engineering (Data Science)



		C	B	D	A
	0	0	0	0	0
A	0	↑ 0	↑ 0	↑ 0	↖ 1
C	0	↖ 1	← 1	← 1	↑ 1
A	0	↑ 1	↑ 1	↑ 1	↖ 2
D	0	↑ 1	↑ 1	↖ 2	↑ 2
B	0	↑ 1	↖ 2	↑ 2	↑ 2

Figure 6.The bottom right corner is the length of the LCS

7. In order to find the longest common subsequence, start from the last element and follow the direction of the arrow. The elements corresponding to () symbol form the longest common subsequence.



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

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

Figure 7.Create a path according to the arrows

Thus, the longest common subsequence is CA.

Pseudocode:

```
X and Y be two given sequences
Initialize a table LCS of dimension X.length * Y.length
X.label = X
Y.label = Y


LCS[0][] = 0
LCS[][0] = 0
Start from LCS[1][1]
Compare X[i] and Y[j]
```

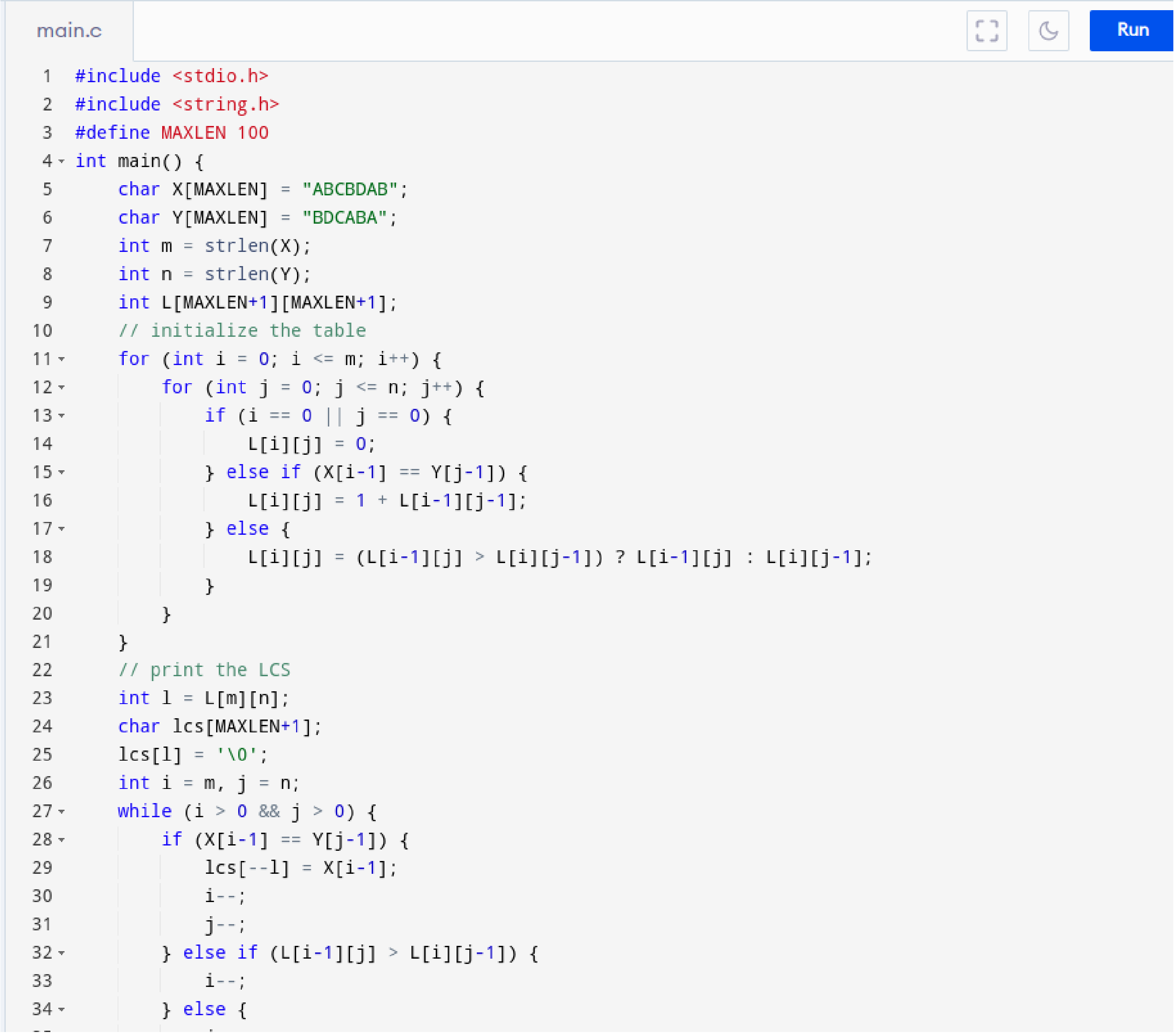
```
  If X[i] = Y[j]
     LCS[i][j] = 1 + LCS[i-1, j-1]
     Point an arrow to LCS[i][j]
  Else
     LCS[i][j] = max(LCS[i-1][j], LCS[i][j-1])
     Point an arrow to max(LCS[i-1][j], LCS[i][j-1])
```

Lab Assignment to Complete:

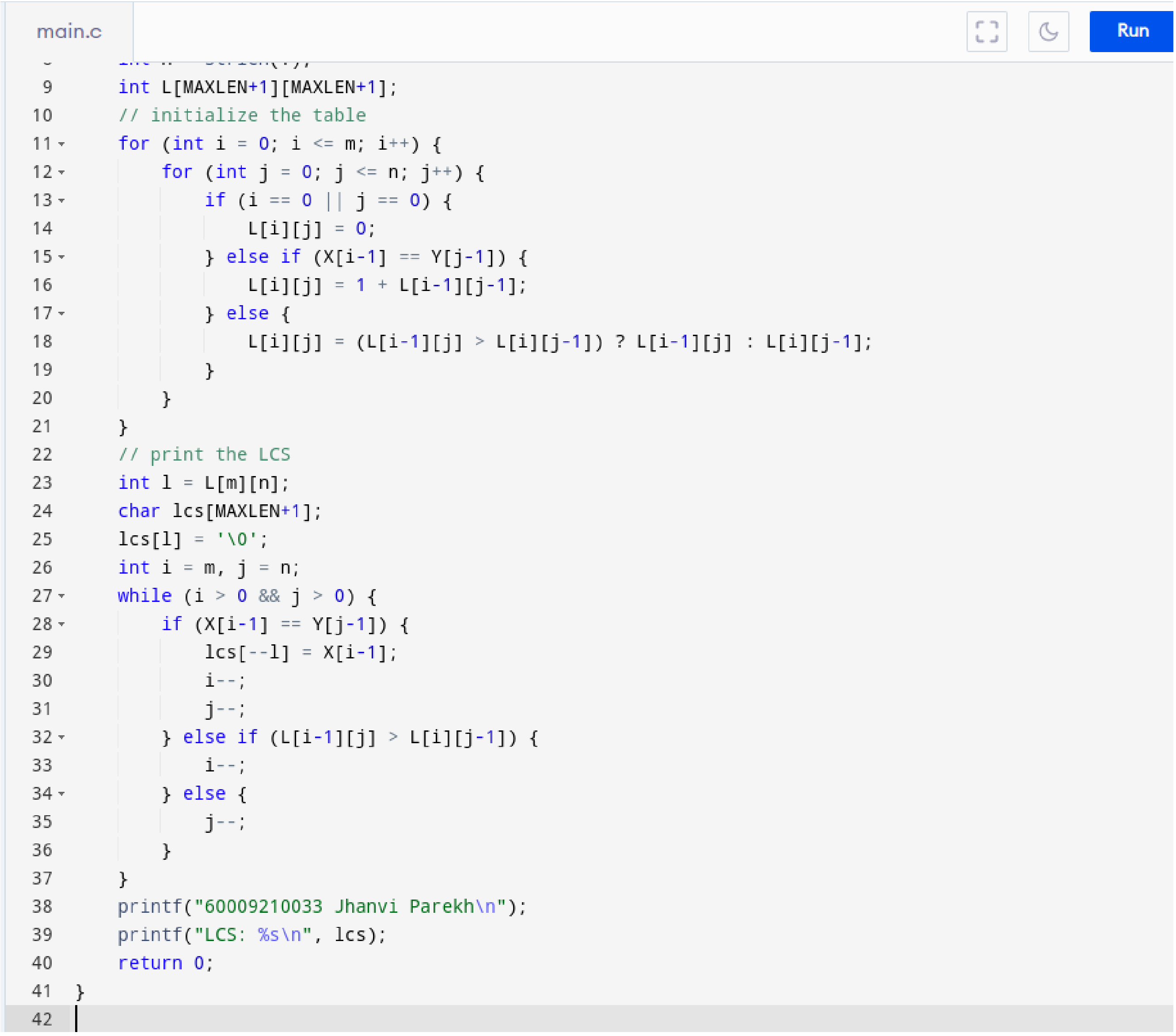
Consider the two following sequences, X and Y:

X: ABCBDAB

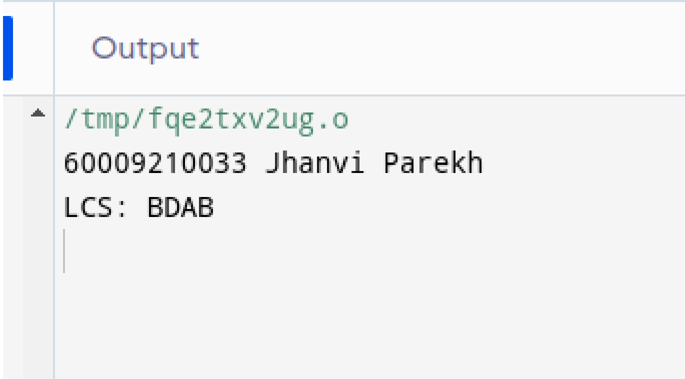
Y:BDCABA



```c
#include <stdio.h>
#include <string.h>
#define MAXLEN 100
int main() {
    char X[MAXLEN] = "ABCBDAB";
    char Y[MAXLEN] = "BDCABA";
    int m = strlen(X);
    int n = strlen(Y);
    int L[MAXLEN+1][MAXLEN+1];
    // initialize the table
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0 || j == 0) {
                L[i][j] = 0;
            } else if (X[i-1] == Y[j-1]) {
                L[i][j] = 1 + L[i-1][j-1];
            } else {
                L[i][j] = (L[i-1][j] > L[i][j-1]) ? L[i-1][j] : L[i][j-1];
            }
        }
    }
    // print the LCS
    int l = L[m][n];
    char lcs[MAXLEN+1];
    lcs[l] = '\0';
    int i = m, j = n;
    while (i > 0 && j > 0) {
        if (X[i-1] == Y[j-1]) {
            lcs[--l] = X[i-1];
            i--;
            j--;
        } else if (L[i-1][j] > L[i][j-1]) {
            i--;
        } else {
```

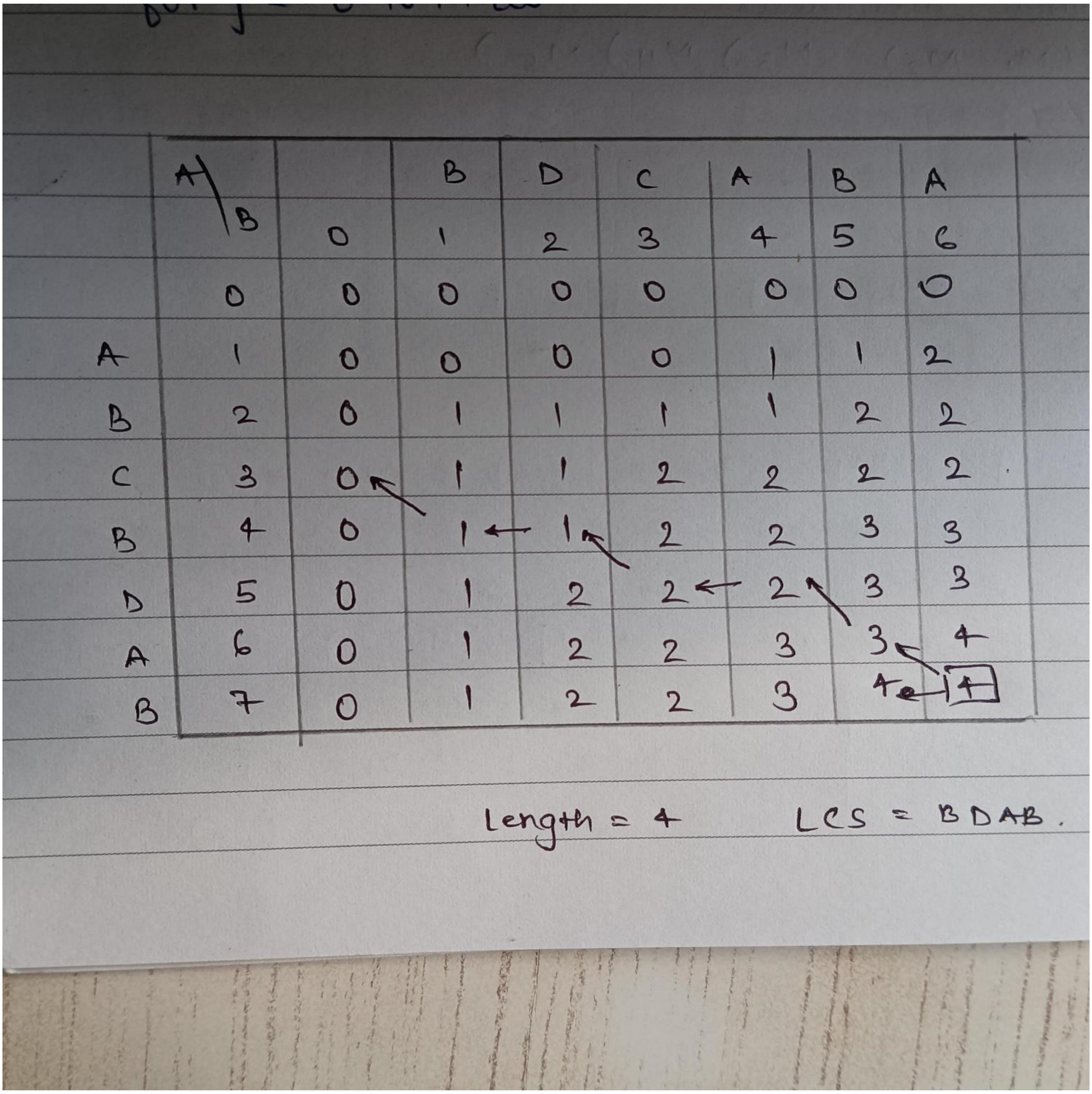


```c
    int n = strlen(Y);
    int L[MAXLEN+1][MAXLEN+1];
    // initialize the table
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0 || j == 0) {
                L[i][j] = 0;
            } else if (X[i-1] == Y[j-1]) {
                L[i][j] = 1 + L[i-1][j-1];
            } else {
                L[i][j] = (L[i-1][j] > L[i][j-1]) ? L[i-1][j] : L[i][j-1];
            }
        }
    }
    // print the LCS
    int l = L[m][n];
    char lcs[MAXLEN+1];
    lcs[l] = '\0';
    int i = m, j = n;
    while (i > 0 && j > 0) {
        if (X[i-1] == Y[j-1]) {
            lcs[--l] = X[i-1];
            i--;
            j--;
        } else if (L[i-1][j] > L[i][j-1]) {
            i--;
        } else {
            j--;
        }
    }
    printf("60009210033 Jhanvi Parekh\n");
    printf("LCS: %s\n", lcs);
    return 0;
}
```



```
/tmp/fqe2txv2ug.o
60009210033 Jhanvi Parekh
LCS: BDAB
```

WRITTEN:



	A\B		B	D	C	A	B	A
		0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	2
B	2	0	1	1	1	1	2	2
C	3	0 ↖	1	1	2	2	2	2
B	4	0	1 ←	1 ↖	2	2	3	3
D	5	0	1	2	2 ←	2 ↖	3	3
A	6	0	1	2	2	3	3 ↖	4
B	7	0	1	2	2	3	4 ←	$\boxed{4}$

Length = 4 LCS = BDAB.