

|  |
| --- |
| **Title: Implementation Of String Matching Algorithm** |



**Objective:** To compute longest common subsequence for the given two strings.



**CO to be achieved:**

|  |  |
| --- | --- |
| Sr. No | Objective |
| CO 1 | Compare and demonstrate the efficiency of algorithms using asymptotic complexity notations. |
| CO 2 | Analyze and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies. |
| CO 3 | Analyze and solve problems for   different string matching algorithms. |



**Books/ Journals/ Websites referred:**

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran,” Fundamentals of computer algorithm”, University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein,” Introduction to algortihtms”,2nd Edition ,MIT press/McGraw Hill,2001**
3. [**http://en.wikipedia.org/wiki/Longest\_common\_subsequence\_problem**](http://en.wikipedia.org/wiki/Longest_common_subsequence_problem)
4. [**http://www.columbia.edu/~cs2035/courses/csor4231.F11/lcs.pdf**](http://www.columbia.edu/~cs2035/courses/csor4231.F11/lcs.pdf)
5. **http://www-igm.univ-mlv.fr/~lecroq/seqcomp/node4.html**



**Pre Lab/ Prior Concepts:**

Data structures, Concepts of algorithm analysis



**Historical Profile:**

Given 2 sequences, *X* = *x*1 *, ..., xm*  and *Y* = *y*1 *, ... , yn* , find a subsequence common to both whose length is longest. A subsequence doesn’t have to be consecutive, but it has to be in order.



**New Concepts to be learned:**

String matching algorithm, Dynamic programming approach for LCS, Applications of LCS



**Recursive Formulation:**

Define *c*[*i, j* ] = length of LCS of *Xi* and *Y j* .

Final answer will be computed with *c*[*m, n*].

c[i, j]= 0 if i=0 or j=0.

c[i, j]= c[i − 1, j − 1] + 1 if i,j>0 and xi=yj

c[i, j]= max(c[i − 1, j ], c[i, j − 1]) if i, j > 0 and xi <> y j

**Algorithm: Longest Common Subsequence**

**Compute length of optimal solution-**

**LCS-LENGTH** *( X , Y, m, n)*

**for** *i* ← 1 **to** *m*

**do** *c*[*i,* 0] ← 0

**for** *j* ← 0 **to** *n*

**do** *c*[0*, j* ] ← 0

**for** *i* ← 1 **to** *m*

**do for** *j* ← 1 **to** *n*

**do if** *xi* = *y j*

**then** *c*[*i, j* ] ← *c*[*i* − 1*, j* − 1] + 1

*b*[*i, j* ] ← “≈”

**else if** *c*[*i* − 1*, j* ] ≥ *c*[*i, j* − 1]

**then** *c*[*i, j* ] ← *c*[*i* − 1*, j* ]

*b*[*i, j* ] ← “↑”

**else** *c*[*i, j* ] ← *c*[*i, j* − 1]

*b*[*i, j* ] ← “←”

**return** *c* and *b*

**Print the solution-**

**PRINT-LCS*(b, X , i, j )***

**if** *i* = 0 or *j* = 0

**then return**

**if** *b*[*i, j* ] = “≈”

**then** PRINT-LCS*(b, X , i* − 1*, j* − 1*)*

print *xi*

**elseif** *b*[*i, j* ] = “↑”

**then** PRINT-LCS*(b, X , i* − 1*, j )*

**else** PRINT-LCS*(b, X , i, j* − 1*)*

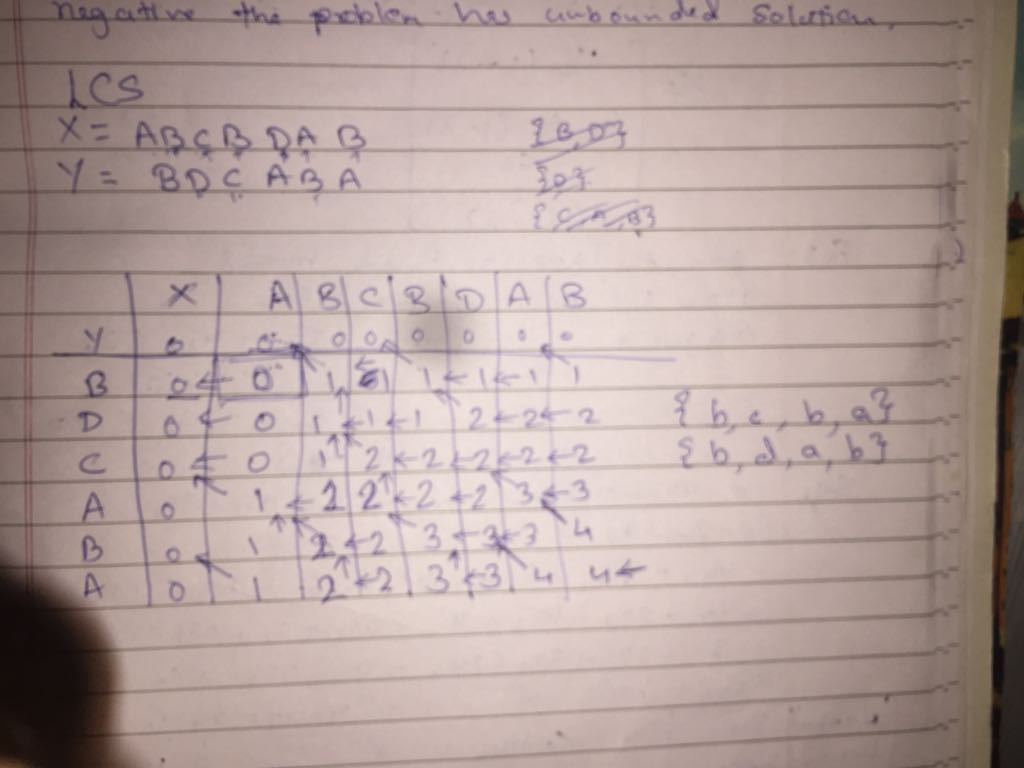
•Initial call is PRINT-LCS*(b, X , m, n)*.

•*b*[*i, j* ] points to table entry whose subproblem we used in solving LCS of *Xi*

and *Y j* .

• When *b*[*i, j* ] = ≈, we have extended LCS by one character. So longest com- mon subsequence = entries with ≈ in them.

**Example: LCS computation**



**LCS computation**

**Code:**

**import java.util.\*;**

**public class LCS1**

**{**

**public static void main(String args[])**

**{**

**Scanner sc=new Scanner(System.in);**

**int i,j;**

**System.out.println("enter the 1st string");**

**String s1=sc.nextLine();**

**System.out.println("enter the 2nd string" );**

**String s2=sc.nextLine();**

**char[] x=s1.toCharArray();**

**int a=s1.length();**

**int b=s2.length();**

**int[][] table=new int[a+2][b+2];**

**char[][] track=new char[a+2][b+2];**

**table[0][0]=0;**

**table[1][0]='X';**

**table[0][1]='Y';**

**for(i=0;i<a;i++)**

**{**

**for(j=0;j<b;j++){**

**if(s1.charAt(i)==s2.charAt(j)){**

**table[i+2][j+2]=table[i+1][j+1]+1;**

**System.out.print(table[i+2][j+2]);**

**track[i+2][j+2]='D';**

**System.out.print(track[i+2][j+2]+"\t");**

**}**

**else{**

**table[i+2][j+2]=Math.max(table[i+1][j+2],table[i+2][j+1]);**

**System.out.print(table[i+2][j+2]);**

**if(table[i+1][j+2]>table[i+2][j+1]){**

**track[i+2][j+2]='U';**

**System.out.print(track[i+2][j+2]+"\t");**

**}**

**else{**

**track[i+2][j+2]='L';**

**System.out.print(track[i+2][j+2]+"\t");**

**}**

**}**

**}**

**System.out.println();**

**}/\***

**for(i=0;i<=a+1;i++)**

**{**

**for(j=0;j<=b+1;j++){**

**System.out.print(i+" "+j+" "+track[i][j]+"\t");**

**}**

**System.out.println();**

**}\*/**

**printlcs(track,x,a,b);**

**}**

**static void printlcs(char track[][],char x[],int i,int j){**

**if(i==0 || j==0)**

**return;**

**if(track[i+1][j+1]=='D'){**

**printlcs(track,x,i-1,j-1);**

**System.out.print(x[i-1]+"\t");**

**}**

**else if(track[i+1][j+1]=='U')**

**printlcs(track,x,i-1,j);**

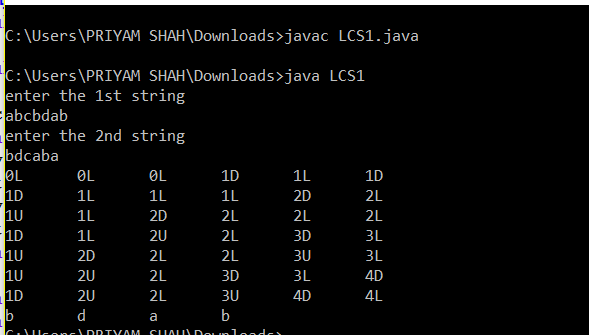
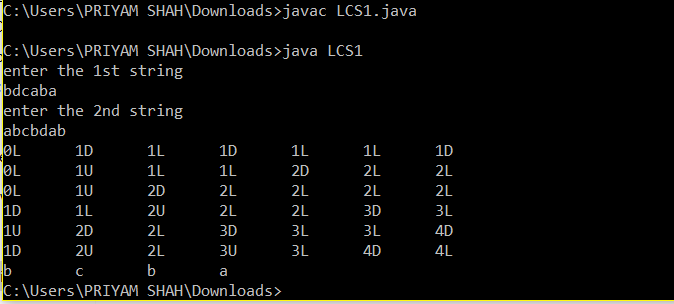
**else**

**printlcs(track,x,i,j-1);**

**}**

**}**

**Output:**

****

**CONCLUSION:**

**Implementation Of String Matching Algorithm has been studied.**