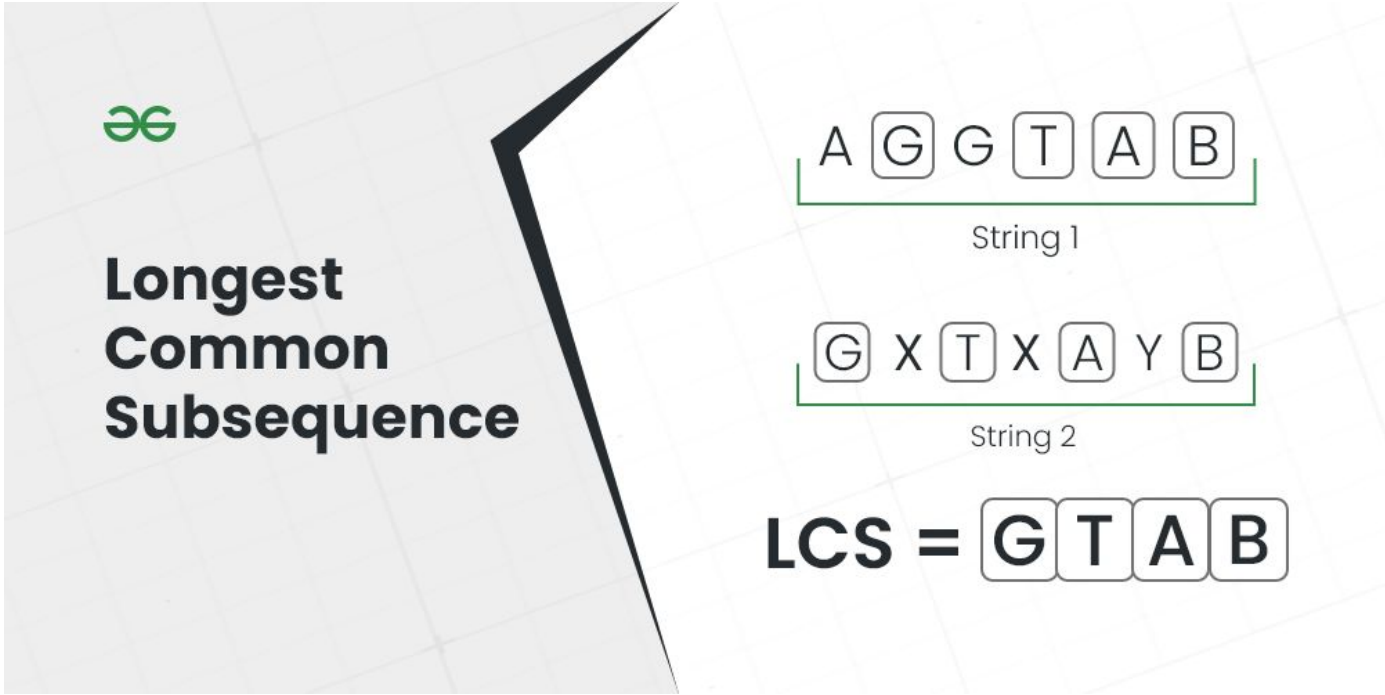
AEL-5

Md. Atiqur Rahman

Longest Common Subsequence

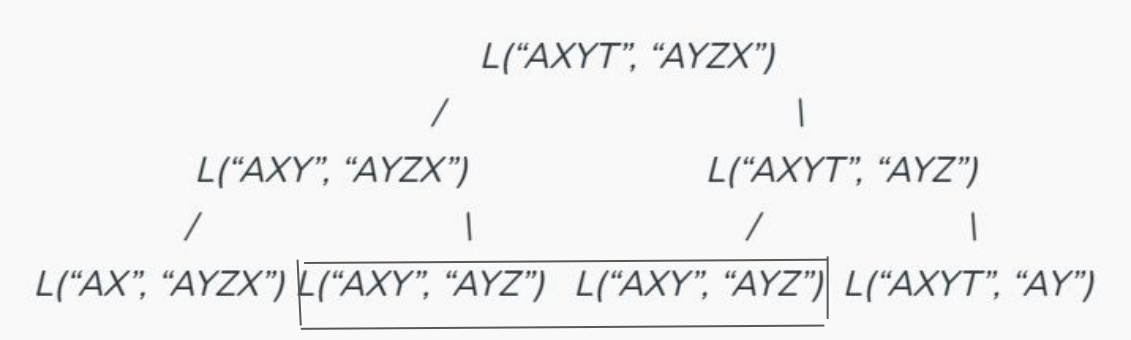
Given two strings, **S1** and **S2**, the task is to find the length of the Longest Common Subsequence, i.e. longest subsequence present in both of the strings.

Subsequence can be derived from the given sequence by deleting some or no elements without changing the order of the remaining elements.



Naive Approach

Generate all the possible subsequences and find the longest among them that is present in both strings using recursion.




```cpp
#include <bits/stdc++.h>
using namespace std;

// Returns length of LCS for X[0..m-1], Y[0..n-1]
int lcs(string X, string Y, int m, int n)
{
    if (m == 0 || n == 0)
        return 0;
    if (X[m - 1] == Y[n - 1])
        return 1 + lcs(X, Y, m - 1, n - 1);
    else
        return max(lcs(X, Y, m, n - 1),
                   lcs(X, Y, m - 1, n));
}

// Driver code
int main()
{
    string S1 = "AGGTAB";
    string S2 = "GXTXAYB";
    int m = S1.size();
    int n = S2.size();

    cout << "Length of LCS is " << lcs(S1, S2, m, n);

    return 0;
}

// This code is contributed by rathbhupendra
```

Properties suitable for Dynamic Programming

1. **Optimization problem**, like longest, minimum, maximum.
2. **Optimal Substructure**, See for solving the structure of L(X[0, 1, . . ., m-1], Y[0, 1, . . . , n-1]) we are taking the help of the substructures of X[0, 1, …, m-2], Y[0, 1,…, n-2], depending on the situation (i.e., using them optimally) to find the solution of the whole.
3. **Overlapping Subproblems**, If we use the above recursive approach for strings “**BD**” and “**ABCD**“, we will get a partial recursion tree as shown below. Here we can see that the subproblem L(“BD”, “ABCD”) is being calculated more than once. If the total tree is considered there will be several such overlapping subproblems.

Memoization

```
int lcs(char* X, char* Y, int m, int n,
        vector<vector<int> >& dp)
{
    if (m == 0 || n == 0)
        return 0;
    if (X[m - 1] == Y[n - 1])
        return dp[m][n] = 1 + lcs(X, Y, m - 1, n - 1, dp);

    if (dp[m][n] != -1) {
        return dp[m][n];
    }
    return dp[m][n] = max(lcs(X, Y, m, n - 1, dp),
                          lcs(X, Y, m - 1, n, dp));
}
```

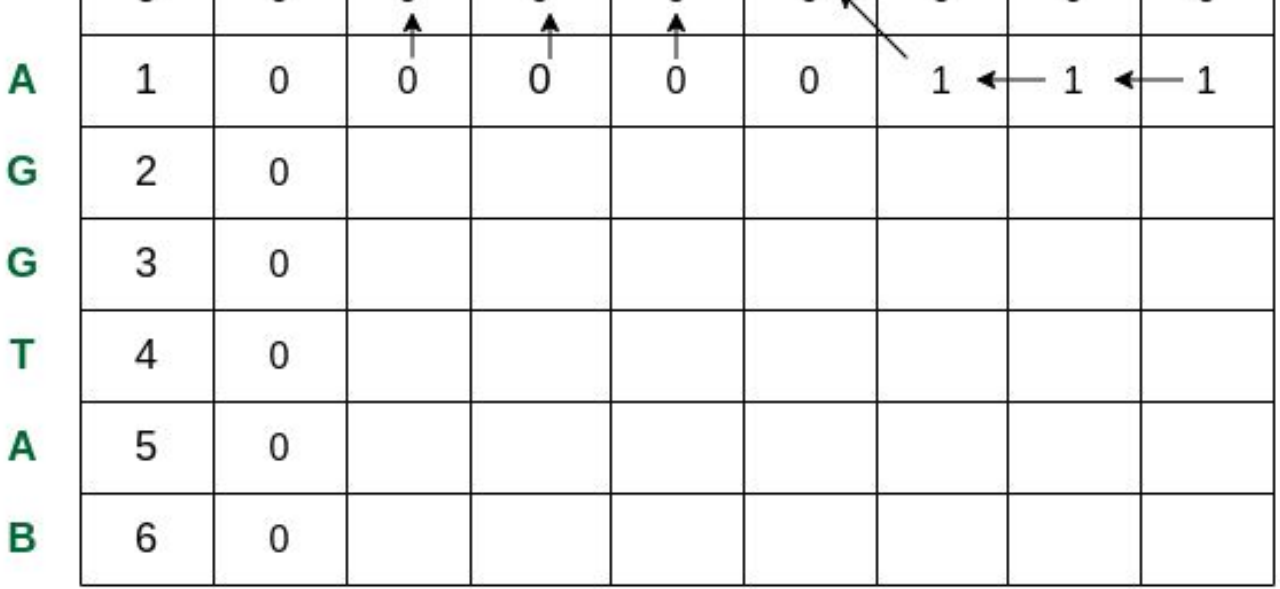
Bottom-Up or Tabulation

			G	X	T	X	A	Y	B
		0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
A	1	0							
G	2	0							
G	3	0							
T	4	0							
A	5	0							
B	6	0							

Creating dp table and filling 0 in 0th row and column

Continued

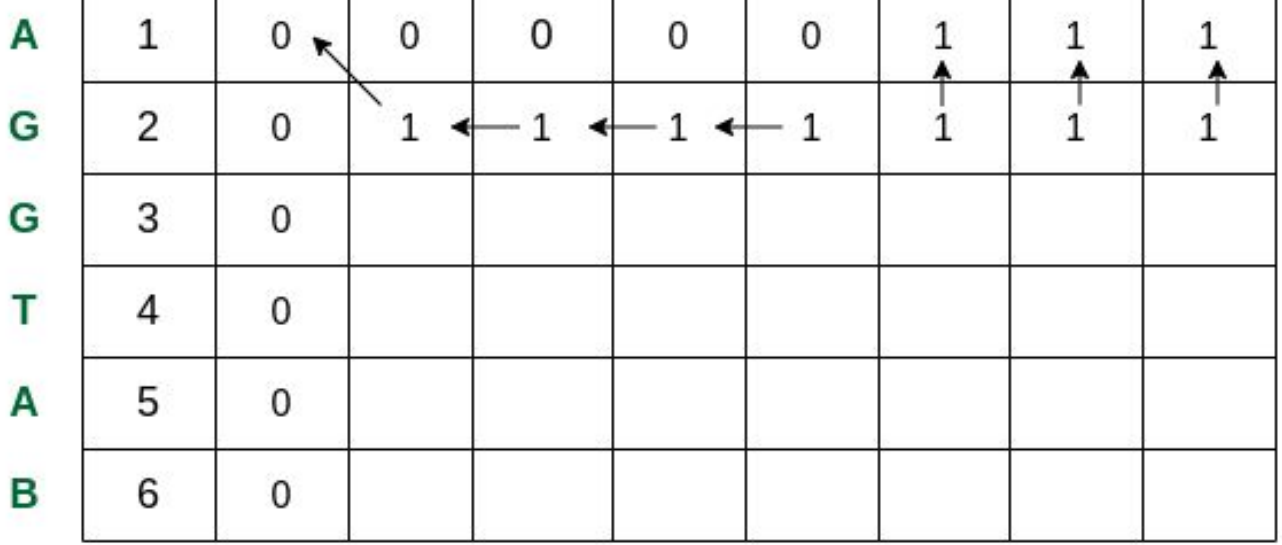
			G	X	T	X	A	Y	B
		0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
A	1	0	0	0	0	0	1	1	1
G	2	0							
G	3	0							
T	4	0							
A	5	0							
B	6	0							



Updating dp table for row 1

Continued

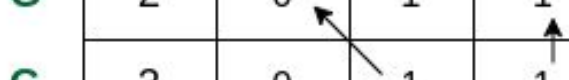
			G	X	T	X	A	Y	B
		0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
A	1	0	0	0	0	0	1	1	1
G	2	0	1	1	1	1	1	1	1
G	3	0							
T	4	0							
A	5	0							
B	6	0							



Updating dp table for row 2

Continued

			G	X	T	X	A	Y	B
		0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
A	1	0	0	0	0	0	1	1	1
G	2	0	1	1	1	1	1	1	1
G	3	0	1	1	1	1	1	1	1
T	4	0							
A	5	0							
B	6	0							



Updating dp table for row 3

Continued

			G	X	T	X	A	Y	B
		0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
A	1	0	0	0	0	0	1	1	1
G	2	0	1	1	1	1	1	1	1
G	3	0	1	1	1	1	1	1	1
T	4	0	1	1	2	2	2	2	2
A	5	0							
B	6	0							

Updating dp table for row 4

Continued

			G	X	T	X	A	Y	B
		0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
A	1	0	0	0	0	0	1	1	1
G	2	0	1	1	1	1	1	1	1
G	3	0	1	1	1	1	1	1	1
T	4	0	1	1	2	2	2	2	2
A	5	0	1	1	2	2	3	3	3
B	6	0							

Updating dp table for row 5

Continued

			G	X	T	X	A	Y	B
		0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
A	1	0	0	0	0	0	1	1	1
G	2	0	1	1	1	1	1	1	1
G	3	0	1	1	1	1	1	1	1
T	4	0	1	1	2	2	2	2	2
A	5	0	1	1	2	2	3	3	3
B	6	0	1	1	2	2	3	3	4

Updating dp table for row 6

Greedy Approach

Greedy Choice Property : says that the globally optimal solution can be obtained by making a locally optimal solution (Greedy). The choice made by a Greedy algorithm may depend on earlier choices but not on the future. It iteratively makes one Greedy choice after another and reduces the given problem to a smaller one.

Characteristics of Greedy approach:

- There is an ordered list of resources(profit, cost, value, etc.)
- Maximum of all the resources(max profit, max value, etc.) are taken.
- For example, in the fractional knapsack problem, the maximum value/weight is taken first according to available capacity.

The local decisions (or choices) must possess three characteristics as mentioned below:

1. **Feasibility**: The selected choice must fulfil local constraints.
2. **Optimality**: The selected choice must be the best at that stage (locally optimal choice).
3. **Irrevocability**: The selected choice cannot be changed once it is made.

Applications of Greedy Approach

(1) Make a change problem

(2) Knapsack problem

(3) Minimum spanning tree

(4) Single source shortest path

(5) Activity selection problem

(6) Job sequencing problem

(7) Huffman code generation.

(8) Dijkstra's algorithm

(9) Greedy coloring

(10) Minimum cost spanning tree

(11) Job scheduling

(12) Interval scheduling

(13) Greedy set cover

(14) Knapsack with fractions

Advantages

- The greedy approach is easy to implement.
- Typically have less time complexity.
- Greedy algorithms can be used for optimization purposes or finding close to optimization in case of Hard problems.
- Greedy algorithms are often faster than other optimization algorithms, such as dynamic programming or branch and bound, because they require less computation and memory.
- The greedy approach can be used to solve problems in real-time, such as scheduling problems or resource allocation problems, because it does not require the solution to be computed in advance.

Disadvantages

- The local optimal solution may not always be globally optimal.
- Greedy algorithms do not always guarantee to find the optimal solution, and may produce suboptimal solutions in some cases.
- The greedy approach relies heavily on the problem structure and the choice of criteria used to make the local optimal choice. If the criteria are not chosen carefully, the solution produced may be far from optimal.
- Greedy algorithms may require a lot of preprocessing to transform the problem into a form that can be solved by the greedy approach.
- Greedy algorithms may not be applicable to problems where the optimal solution depends on the order in which the inputs are processed.