13 LIS, LCS AND EDIT DISTANCE PROBLEMS

Problem 1: Longest Increasing Subsequence (LIS)

Given an unsorted array *a[]* of integers, find the length of the *longest increasing subsequence*.

A **subsequence** is a sequence that appears in the same relative order, but not necessarily contiguous. So we can skip some elements in the array in order to make the longest subsequence.

Sample Input

```
8
10 9 2 5 3 7 101 18
```

Sample Output

```
4
```

Output Details

```
10 9 2 5 3 7 101 18
```

$$F(i) = \max \begin{cases} 1 \\ F(j) + 1 & \text{where } 0 <= j < i \text{ and } a[j] < a[i] \end{cases}$$

Sol 1: O(N*N) Time and O(N) Space

```cpp
// LIS. Print the length. Iterative O(N*N)
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main()
{
	vector<int> a = { 10, 9, 2, 5, 3, 7, 101, 18 };	//input array
	int N = a.size(); //number of elements

	vector<int> dp(N, 1);  //memo vector
	for (int i = 1; i < N; i++)
		for (int j = 0; j < i; j++)
			if (a[j] < a[i])
				dp[i] = max(dp[i], dp[j] + 1);

	//Max element of dp vector is the length of LIS
	cout << *max_element(dp.begin(), dp.end());
}
```

Output

```
4
```

Sol 2: Optimized DP. O(NlgN) Time and O(N) Space

```
// LIS. Print the length. Iterative DP with optimization O(NlgN).
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
	vector<int> a = { 10, 9, 2, 5, 3, 7, 101, 18 };     //input array
	int N = a.size();                                   //number of elements

	vector<int> tail;
	int res = 1;

	tail.push_back(a[0]);

	for (int i = 1; i < N; i++)
	{
		//find the positon of the first element in tail it is not smaller than a[i]
		auto it = lower_bound(tail.begin(), tail.end(), a[i]);

		if (it == tail.end())  //a[i] is the largest. Becomes the last element in tail.
			tail.push_back(a[i]);
		else                         //Replace lower bound element of tail with a[i]
			*it = a[i];
	}

	cout << tail.size() << endl;
}
```

Output

```
4
```

Exercise 1: Printing LIS

Given an unsorted array of integers. Print one of the *longest increasing subsequences* of the array.

Sample Input

```
8
10 9 2 5 3 7 101 18
```

Sample Output

```
2 3 7 18
```

Exercise 2: Counting LISs

Given an unsorted array of integers. Count number of all longest increasing subsequences of the array in O(NlgN) time.

Sample Input

```
8
10 9 2 5 3 7 101 18
```

Sample Output

```
4
```

Output Details

```
2 5 7 101
2 5 7 18
2 3 7 101
2 3 7 18
```

Variations of the LIS Problem

Building Bridges

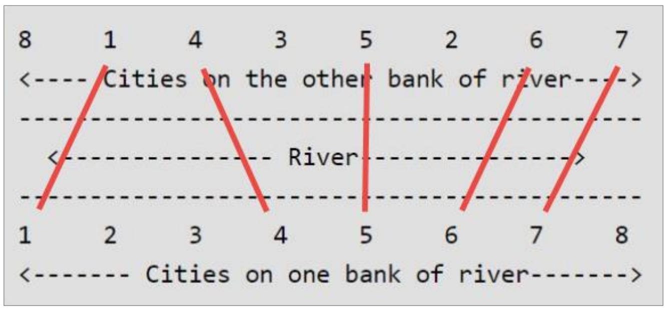
There is a river. There are N (1 <= N <= 100,000) cities on both sides of the river (not necessarily in the same order). Bridges are to be built on the river for every city (from one side of the river to the other side where the same city is located). But no two bridges should intersect. Find out the maximum number of cities for which the bridges can be built over the river.

Sample Input

```
8
8 1 4 3 5 2 6 7
1 2 3 4 5 6 7 8
```

Sample Output

```
5
```


8 1 4 3 5 2 6 7
<---- Cities on the other bank of river---->
<------------- River ------------->
1 2 3 4 5 6 7 8
<------- Cities on one bank of river------->

Maximum Sum Increasing Subsequence

Given an array of *N* (1 <= *N* <= 100,000) positive integers. Write a program to find the sum of maximum sum subsequence of the given array such that the integers in the subsequence are sorted in increasing order.

Sample Input

```
7
1 101 2 3 100 4 5
```

Sample Output

```
106  (1 + 2 + 3 + 100)
```

Largest Divisible Subset

Given a set of N (1 <= N <= 1000) distinct positive integers, find the largest subset such that every pair (Si, Sj) of elements in this subset satisfies:

`Si % Sj = 0` or `Sj % Si = 0.`

If there are multiple solutions, return any subset is fine.

Sample Input

```
6
5 3 2 18 9 6
```

Sample Output

```
3 6 18
```

The Longest Chain

You are given *N* (1 <= *N* <= 100,000) pairs of numbers. In every pair, the first number is always smaller than the second number. A pair (*c*, *d*) can follow (*a*, *b*) if and only if *b* is less than *c*. Chains of pairs can be formed in this manner. Find the longest chain of pairs formed.

Sample Input

```
(15, 40) (5, 8) (1, 10) (6, 8) (9, 20) (2, 4) (36, 37) (34, 35) (9, 14) (30, 31)
```

Sample Output

```
(2, 4) (6, 8) (9, 14) (30, 31) (34, 35) (36, 37)
```

Box Stacking

There are *N* (1 <= *N* <= 100,000) cuboidal boxes. The dimensions of these boxes (length, breadth and height) are given. The objective is to stack the boxes to achieve maximum height. But you can place a box on top of another box only if its base dimensions are strictly lower than the other box. You can rotate the boxes. Also, you can use multiple instances of the same box.

Sample Input

```
n=3
Dimensions (l, b, h)
Box 1 - (1,2,3)
Box 2 - (2,3,4)
Box 3 - (3,4,5)
```

Sample Output

```
15
```

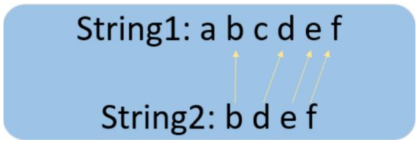
Output Details

```
Stacking from top to bottom
```

```
1, 2, 3
2, 3, 4
3, 4, 5
4, 5, 3
```

Problem 2: Longest Common Subsequence (LCS)

Given two sequences, find the length of longest subsequence present in both of them.



$$F(n, m) = \begin{cases} 0 & \text{if } n == 0 \text{ or } m == 0 \\ 1 + F(n-1, m-1) & \text{if } s1[n] == s2[m] \\ \max(F[n-1, m], F[n, m-1]) & \text{otherwise} \end{cases}$$

Implementation with Memoization O(nm)

```
// Longest Common SubString with memoization.
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

string str1 = "abcdef", str2 = "bdef";
vector<vector<int>> memo;

int go(int n, int m)
{
      if (n < 0 || m < 0)
            return 0;

      if (memo[n][m] > 0)
            return memo[n][m];

      if (str1[n] == str2[m])
            memo[n][m] = 1 + go(n - 1, m - 1);
      else
            memo[n][m] = max(go(n - 1, m), go(n, m - 1));

      return memo[n][m];

}
int main()
{
      int n = str1.length(), m= str2.length();
      memo.resize(n, vector<int>(m, 0));
      cout << go(n-1, m-1) << endl;
}
```

```
Output

4
```

Implementation with Tabulation O(nm)

```
// Longest Common SubString with tabulation.
#include <iostream>
#include <string>
#include <algorithm>
#include <vector>

using namespace std;

string s1 = "BCDAACD";
string s2 = "ACDBAC";
//----------------------------------------------------
int go()
{
	int n = s1.length();
	int m = s2.length();
	vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));

	for (int i = 1; i <= n; i++)
		for (int j = 1; j <= m; j++)
			if (s1[i - 1] == s2[j - 1])
				dp[i][j] = 1 + dp[i - 1][j - 1];
			else
				dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
	return dp[n][m];
}
//----------------------------------------------------
int main()
{
	cout << "Length of a LCS is " << go() << endl;
}
```

Output

```
Length of a LCS is 4
```

Problem 3: Longest Common Substring (LCS)

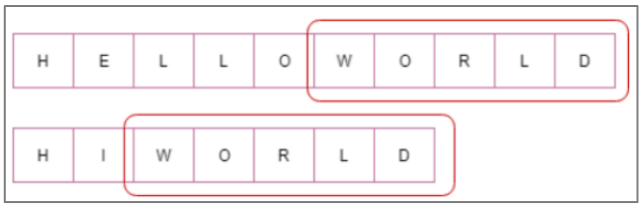
Given two strings, find the length of longest substring present in both of them. A **substring** is a contiguous sequence of characters within a string.

Sample Input

```
HELLOWORLD
HIWORLD
```

Sample Output

```
5
```



Implementation with Memoization O(nm).

```
// Longest Common Substring with memoization.
#include <iostream>
#include <string>
#include <algorithm>
#include <vector>

using namespace std;

string s1 = "BCDAACDB";
string s2 = "BCDDAACDCB";
vector<vector<int>> dp;
int res = 0;
//---------------------------------------------------
void go(int n, int m)
{
	if (n == 0 || m == 0)
		return;
	if (dp[n][m] >= 0)
		return;

	go(n - 1, m);
	go(n, m - 1);

	if (s1[n-1] == s2[m-1])
	{
		go(n - 1, m - 1);
		dp[n][m] = dp[n - 1][m - 1] + 1;
		res = max(res, dp[n][m]);
	}
	else
		dp[n][m] = 0;
}
//---------------------------------------------------
int main()
{
	int n = s1.length();
	int m = s2.length();
	dp.resize(n+1, vector<int>(m+1, -1));
	go(n , m);
	cout << res << endl;
}
```

```
Output

5
```

Implementation with Tabulation O(nm).

```
// Longest Common Substring with tabulation.
#include <iostream>
#include <string>
#include <algorithm>
#include <vector>

using namespace std;

string s1 = "BCDAACDB";
string s2 = "BCDDAACDCB";
//---------------------------------------------------
int go()
{
	int res = 0;

	int n = s1.length();
	int m = s2.length();
	vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));
	for(int i=1; i<=n; i++)
		for(int j=1; j<=m; j++)
			if (s1[i - 1] == s2[j - 1])
			{
				dp[i][j] = dp[i - 1][j - 1] + 1;
				res = max(res, dp[i][j]);
			}

	return res;
}
//-------------------------------------------------------
int main()
{
	cout << go() << endl;
}
```

Output

```
5
```

Exercise 3: Longest Common Substring

Modify the longest common substring program so that is runs in O(nm) time and in O(n) space. Your program should print the length of the longest common substring and print one of them.

Sample Input

```
BCDDAACDCB
BCDAACDB
```

Sample Output

```
5
DAACD
```

Problem 4: Edit (Levenshtein) Distance

Given two words *word1* and *word2*, find the minimum number of operations required to convert *word1* to *word2*.

You have the following three operations permitted on a word:

- Insert a character
- Delete a character
- Replace a character

Sample Input

```
Horse
ros
```

Sample Output

```
3
```

Output Details

```
horse -> rorse (replace 'h' with 'r')
rorse -> rose (remove 'r')
rose -> ros (remove 'e')
```

$$F(n, m) = \begin{cases} \max(n, m) & \text{if } n == 0 \text{ or } m == 0 \\ \min \begin{cases} F(n-1, m) + 1 & \\ F(n, m-1) + 1 & \\ F(n-1, m-1) + 0 & \text{if } (s1[n] == s2[m] \\ F(n-1, m-1) + 1 & \text{if } (s1[n] \mathrel{!=} s2[m] \end{cases} & \end{cases}$$

Implementation with Memoization O(nm).

```
// Edit Distance with memoization O(nm).
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

string s1 = "counter", s2 = "computer";
vector<vector<int>> memo;
//----------------------------------
int go(int n, int m)
{
      if (n == 0 || m == 0)
            return n + m - 1;

      if (memo[n][m] >= 0)
            return memo[n][m];

      int res1 = min(go(n - 1, m), go(n, m - 1)) + 1;

      int res2 = go(n - 1, m - 1);
      if (s1[n - 1] != s2[m - 1])
            res2++;

      return memo[n][m] = min(res1, res2);}

//----------------------------------
int main()
{
    int n = s1.length(), m = s2.length();
      memo.resize(n + 1, vector<int>(m + 1, -1));
      cout << go(n, m);
}
```

```
Output

3
```

Implementation with Tabulation O(nm).

```
// Edit Distance with tabulation O(nm).
# #include <iostream>
#include <vector>
#include <string>
#include <algorithm>

using namespace std;
```

```
string s1 = "ABBDADACD", s2 = "BABBBADADCD";
vector<vector<int>> memo;
//---------------------------------
int go()
{
	int n = s1.length(), m = s2.length();
	vector<vector<int>> memo(n + 1, vector<int>(m + 1));

	//s2 is empty
	for (int i = 0; i <= n; i++)
		memo[i][0] = i;

	//s1 is empty
	for (int i = 0; i <= m; i++)
		memo[0][i] = i;

	for(int i=1; i<=n; i++)
		for (int j = 1; j <= m; j++)
		{
			memo[i][j] = min(memo[i - 1][j], memo[i][j - 1]) + 1;
			int k = 0;
			if (s1[i - 1] != s2[j - 1])
				k++;
			memo[i][j] = min(memo[i][j], memo[i - 1][j - 1] + k);
		}
	return memo[n][m];
}

//---------------------------------
int main()
{
	cout << go() << endl;;
}
```

```
Output

3
```

Exercise 3: Weighted Edit Distance

Modify the Minimum Edit Distance program so that it calculates the minimum edit distance when the three operations have different costs.

Useful Links and References

https://www.geeksforgeeks.org/ugly-numbers/
http://ghcimdm4u.weebly.com/3-binomial-theorem.html
https://www.geeksforgeeks.org/variations-of-lis-dp-21/
https://leetcode.com/problems/largest-divisible-subset/
https://www.sanfoundry.com/dynamic-programming-solutions-building-bridges-problem/
https://leetcode.com/problems/largest-divisible-subset/
https://stackoverflow.com/questions/17530303/longest-chain-of-pairs
https://www.sanfoundry.com/dynamic-programming-solutions-box-stacking-problem/
https://leetcode.com/problems/edit-distance/