Dibujo en blanco y negro

Descripción generada automáticamente con confianza baja**INSTITUTO POLITÉCNICO NACIONAL**

**ESCUELA SUPERIOR DE CÓMPUTO**

**--------------------ANÁLISIS DE ALGORITMOS------------------**

**ACTIVIDAD**

Diseño de soluciones mediante DP

**PROFESOR:**

Franco Martínez Edgardo Adrián

**ALUMNO:**

Meza Vargas Brandon David – 2020630288

**GRUPO:**

3CM13

Joven con camiseta negra

Descripción generada automáticamente

**índice**

[**Problema: Longest Common Subsequence** 3](#_Toc88063413)

[**Redacción** 3](#_Toc88063414)

[**Captura de aceptación por juez** 3](#_Toc88063415)

[**Explicación Algoritmo** 3](#_Toc88063416)

[**Análisis de complejidad en cota O()** 5](#_Toc88063417)

[**Código de solución completo** 6](#_Toc88063418)

[**Problema: ELIS – Easy Longest Increasing Subsequence** 7](#_Toc88063419)

[**Redacción** 7](#_Toc88063420)

[**Captura de aceptación por juez** 7](#_Toc88063421)

[**Explicación Algoritmo** 7](#_Toc88063422)

[**Análisis de complejidad en cota O()** 8](#_Toc88063423)

[**Código de solución completo** 9](#_Toc88063424)

[**Problema: The Knapsack Problem** 10](#_Toc88063425)

[**Redacción** 10](#_Toc88063426)

[**Captura de aceptación por juez** 10](#_Toc88063427)

[**Explicación Algoritmo** 10](#_Toc88063428)

[**Análisis de complejidad en cota O()** 12](#_Toc88063429)

[**Código de solución completo** 13](#_Toc88063430)

[**Problema: AIBOHP - Aibohphobia** 14](#_Toc88063431)

[**Redacción** 14](#_Toc88063432)

[**Captura de aceptación por juez** 14](#_Toc88063433)

[**Explicación Algoritmo** 15](#_Toc88063434)

[**Análisis de complejidad en cota O()** 15](#_Toc88063435)

[**Código de solución completo** 17](#_Toc88063436)

# **Problema: Longest Common Subsequence**

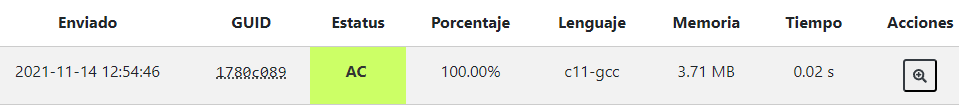
## **Redacción**

Al finalizar su viaje por cuba, Edgardo se puso a pensar acerca de problemas más interesantes que sus alumnos podrían resolver.

En esta ocasión tu trabajo es el siguiente:

Dadas 2 cadenas A y B, debes de encontrar la subsecuencia común más larga entre ambas cadenas.

## **Captura de aceptación por juez**



Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

## **Explicación Algoritmo**

A continuación se presenta la captura del algoritmo:

Texto

Descripción generada automáticamente

Primeramente se usa un arreglo bidimensional para ir guardando la longitud de la subsecuencia común mas larga de la cadena A y la cadena de B.

Posteriormente van los for anidados que recorren ambas longitudes. Si nos encontramos con que i y j no avanzan, es decir, se quedan en 0, es por que las cadenas son vacías y no existe una lcs.

Pero si el último carácter de las cadenas coincide se retira ese carácter y se encuentra el lcs de los caracteres restantes, por ejemplo:

cadena A : ABCDE

cadena B: XYZE

La E coincide, se retira y se calcula el lcs de las anteriores, al no tener ninguna subsecuencia común queda el resultado final con **1.**

Si el último carácter no coincide se encuentra la cadena más larga entre las subacadenas de todos los elementos anteriores de la cadena A o las subcadenas de todos los elementos anteriores a la cadena B, por ejemplo:

A: ABCDE

B: ABCDEA

El último carácter no coindice, si se retira la E de la cadena A, queda una subcadena de 4 y si se retira la A de la cadena B queda una subcadena de 5, por lo tanto se almacena la cadena B.

## **Análisis de complejidad en cota O()**

Texto

Descripción generada automáticamente

longB

longA

En el algoritmo se presentan dos loops, uno que recorre toda la longitud de la cadena A y otro que recorre la cadena B, por lo tanto la complejidad será:

**O(longA\*longB)**

## **Código de solución completo**

Texto

Descripción generada automáticamente

# **Problema: ELIS – Easy Longest Increasing Subsequence**

## **Redacción**

Given a list of numbers A output the length of the longest increasing subsequence. An increasing subsequence is defined as a set {i0 , i1 , i2 , i3 , ... , ik} such that 0 <= i0 < i1 < i2 < i3 < ... < ik < N and A[ i0 ] < A[ i1 ] < A[ i2 ] < ... < A[ ik ]. A longest increasing subsequence is a subsequence with the maximum k (length).

i.e. in the list {33 , 11 , 22 , 44}

the subsequence {33 , 44} and {11} are increasing subsequences while {11 , 22 , 44} is the longest increasing subsequence.

**Input**

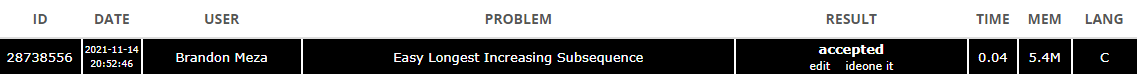
First line contain one number N (1 <= N <= 10) the length of the list A.

Second line contains N numbers (1 <= each number <= 20), the numbers in the list A separated by spaces.

**Output**

One line containing the lenght of the longest increasing subsequence in A.

## **Captura de aceptación por juez**



## **Explicación Algoritmo**

Texto

Descripción generada automáticamente

Este problema nos pide encontrar la subsecuencia creciente más larga, por ejemplo, si tenemos:

**[10,9,2,5,3,7,101,18]**

Podemos ver fácilmente que a partir del 2 hasta el 101 hay una subsecuencia creciente, sin embargo el 3 nos esta estorbando ahí, al quitarlo obtenemos la LIS, siendo esta de 4: 2,5,7,101.

Para lograr lo anterior podemos simplificar este problema al primer problema mostrado que obtiene la LCS, para esto debemos tener dos arreglos, uno que será el original y el otro que será el original pero ordenado, de esta forma si obtenemos la subsecuencia común más larga, nos dará automáticamente la subsecuencia creciente más larga. Haciéndose el mismo procedimiento que el primero problema mostrado en el presente documento.

## **Análisis de complejidad en cota O()**

Texto

Descripción generada automáticamente

N

M

Como podemos ver se realizan dos ciclos que van recorriendo la longitud del arreglo original, por lo tanto la complejidad será:

**O(N2)**

## **Código de solución completo**

Texto

Descripción generada automáticamente

# **Problema: The Knapsack Problem**

## **Redacción**

The famous knapsack problem. You are packing for a vacation on the sea side and you are going to carry only one bag with capacity S (1 <= S <= 2000). You also have N (1<= N <= 2000) items that you might want to take with you to the sea side. Unfortunately you can not fit all of them in the knapsack so you will have to choose. For each item you are given its size and its value. You want to maximize the total value of all the items you are going to bring. What is this maximum total value?

**Input**

On the first line you are given S and N. N lines follow with two integers on each line describing one of your items. The first number is the size of the item and the next is the value of the item.

**Output**

You should output a single integer on one like - the total maximum value from the best choice of items for your trip.

## **Captura de aceptación por juez**

## **Explicación Algoritmo**

A continuación se presenta la captura del algoritmo usado para resolver este problema:

Texto

Descripción generada automáticamente

En primer lugar establecemos un arreglo bidimensional que nos servirá para representar la tabla, este será del tamaño de los artículos y de la capacidad que tiene la mochila más uno.

Posteriormente viene un ciclo for anidado, el primer for recorre cada item de la mochila (N) y el segundo la capacidad que puede tener la mochila (S).

Cabe aclarar que se usó la siguiente estructura para representar al artículo:

Pantalla de video juego

Descripción generada automáticamente con confianza media

Dentro del segundo for se pregunta si i o j son 0, en caso de ser 0 el beneficio será 0, pues no habrá ningún artículo en la mochila.

En caso contrario y si el articulo que se quiere meter a la mochila tiene un tamaño menor al volumen que puede contener la mochila vamos a poner en la casilla actual [i,j] el valor del beneficio que maximice este beneficio de meter o no cierto artículo. Para esto se encuentra el máximo de dos beneficios; el primer beneficio será la suma del beneficio del articulo actual más el beneficio obtenido en la casilla al recorrer hacia la izquierda el tamaño o volumen del articulo actual, el segundo beneficio será el de meter el articulo anterior. El mayor beneficio de estos dos es el que se almacena.

En caso de que el tamaño del elemento sea mayor al soportado por la mochila se coloca el mismo beneficio en las casillas restantes de la tabla.

## **Análisis de complejidad en cota O()**

Texto

Descripción generada automáticamente

N

S

Podemos ver que la complejidad radica en el llenado de la tabla, haciéndose dos ciclos for, uno recorriendo la cantidad de elementos y otro que recorre la capacidad de la mochila, siendo la complejidad:

**O(N\*S)**

## **Código de solución completo**

Texto

Descripción generada automáticamente

# **Problema: AIBOHP - Aibohphobia**

## **Redacción**

BuggyD suffers from AIBOHPHOBIA - the fear of Palindromes. A palindrome is a string that reads the same forward and backward.

To cure him of this fatal disease, doctors from all over the world discussed his fear and decided to expose him to large number of palindromes. To do this, they decided to play a game with BuggyD. The rules of the game are as follows:

BuggyD has to supply a string **S**. The doctors have to add or insert characters to the string to make it a palindrome. Characters can be inserted anywhere in the string.

The doctors took this game very lightly and just appended the reverse of **S** to the end of **S**, thus making it a palindrome. For example, if **S** = "fft", the doctors change the string to "ffttff".

Nowadays, BuggyD is cured of the disease (having been exposed to a large number of palindromes), but he still wants to continue the game by his rules. He now asks the doctors to insert the minimum number of characters needed to make **S** a palindrome. Help the doctors accomplish this task.

For instance, if **S** = "fft", the doctors should change the string to "tfft", adding only 1 character.

**Input**

The first line of the input contains an integer **t**, the number of test cases. **t** test cases follow.

Each test case consists of one line, the string **S**. The length of **S** will be no more than 6100 characters, and **S** will contain no whitespace characters.

**Output**

For each test case output one line containing a single integer denoting the minimum number of characters that must be inserted into **S** to make it a palindrome.

## **Captura de aceptación por juez**

## **Explicación Algoritmo**

El algoritmo empleado es el mismo que el algoritmo para obtener la subsecuencia común más larga con una pequeña modificación al inicio. Esta modificación radica en tener otra cadena igual a la original ingresada, con la característica de estar invertida, de esta manera podemos obtener fácilmente el lcs que crea un palíndromo, al final hacemos una resta de la longitud de la cadena original menos la longitud del lcs de esta manera el resultado nos dará los caracteres necesarios para formar el palíndromo. El código del algoritmo se presenta a continuación.

Texto

Descripción generada automáticamente

## **Análisis de complejidad en cota O()**

Texto

Descripción generada automáticamente

longB

longA

En el algoritmo se presentan dos loops, uno que recorre toda la longitud de la cadena A y otro que recorre la cadena B, por lo tanto la complejidad será:

**O(longA\*longB)**

Podemos darnos cuenta de que al ser el mismo algoritmo que usamos para obtener el lcs la complejidad es la misma, sin embargo, al tener ambas cadenas la misma longitud la complejidad termina siendo:

**O(lonS2)**

## Texto Descripción generada automáticamente**Código de solución completo**

Texto

Descripción generada automáticamente