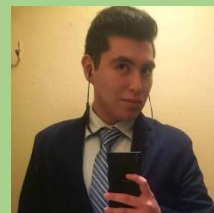


Análisis de algoritmos

Those who cannot remember the past
are condemned to repeat it.

-Dynamic Programming



Erick Efrain Vargas Romero

Prof. Franco Martínez Edgardo Adrián

Diseño de soluciones de programación
dinámica

Longest Common Subsequence

Descripción

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

En esta ocasión tu trabajo es el siguiente:

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

Entrada

La primera linea contendra la cadena A. En la segunda linea vendra la cadena B.

Salida

La longitud de la subsecuencia común mas larga.

Ejemplo

Entrada	Salida	Descripción
AGCT AMGXTP	3	La subsecuencia común más larga es: AGT

Solución

Podemos aplicad programación dinámica generando una especie de tabla, donde las filas serán la cadena que vamos a comparar y las columnas la cadena de la cual vamos a obtener la subsecuencia. Si lo analizamos tendríamos algo así

	(J)	A	G	C	T
(I)	0	0	0	0	0
A	0	1	1	1	1
M	0	1	1	1	1
G	0	1	2	2	2
X	0	1	2	2	2
T	0	1	2	2	3
P	0	1	2	2	3

La solución como podemos ver es generar una tabla, como la que se muestra arriba, vamos a ir iterando hasta encontrar una letra que coincida, si coinciden ambas letras entonces vamos a colocar en la celda actual el valor que tenemos en la celda (i-1,j-1) más uno. Si no coinciden las letras vamos a tener que cubrir dos casos, el primero es que nos movamos (i-1, j) o (i , j-1) y de estos dos casos tomamos el máximo, eso lo colocaremos en la celda (i, j)

Código

```
#include <bits/stdc++.h>

using namespace std;

string a, b;
```

```
int LCS() {
    int a_size = a.size();
    int b_size = b.size();
    int DP[a_size + 1][b_size + 1];


    for(int i = 0; i <= a_size; i++){
        for(int j = 0; j <= b_size; j++){
            if(i == 0 || j == 0)
                DP[i][j] = 0;
            else if(a[i-1] == b[j-1])
                DP[i][j] = DP[i-1][j-1] + 1;
            else
                DP[i][j] = max(DP[i-1][j], DP[i][j-1]);
        }
    }
    return DP[a_size][b_size];
}

int main() {
    cin >> a >> b;
    cout << LCS() << endl;
    return 0;
}
```

Como es evidente la solución de este ejercicio se encuentra en la función LCS, la cual construye la tabla gracias a las dos cadenas, podemos pensar que la solución es similar a realizar una suma de acumulados en dos dimensiones, la solución de nuestro problema se encontrará en la última celda de la tabla, es decir en la celda inferior derecha.

Finalmente, el modelo de este algoritmo es Bottom Up, partimos de los casos más pequeños hasta los más grandes.

Captura

2018-11-17 23:37:40	6dd25725	Respuesta correcta	100.00%	cpp11	5.21 MB	0.02 s	
---------------------	----------	--------------------	---------	-------	---------	--------	---

ELIS – Easy Longest Increasing Subsequence

Descripción

Given a list of numbers A output the length of the longest increasing subsequence. An increasing subsequence is defined as a set $\{i_0, i_1, i_2, i_3, \dots, i_k\}$ such that $0 \leq i_0 < i_1 < i_2 < i_3 < \dots < i_k < N$ and $A[i_0] < A[i_1] < A[i_2] < \dots < A[i_k]$. 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.

Entrada

First line contain one number N ($1 \leq N \leq 10$) the length of the list A.

Second line contains N numbers ($1 \leq \text{each number} \leq 20$), the numbers in the list A separated by spaces.

Salida

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

Ejemplo

Entrada	Salida
5 1 4 2 4 3	3

Solución

Para este ejercicio a diferencia del anterior, no es necesario generar una tabla, bastará un arreglo o vector para llegar a la solución. Realizamos iteraciones nuevamente sobre el vector que contiene todos los números que son ingresados, desde la posición 1 hasta la n-ésima posición. Además anidamos otro ciclo for, que parte de la posición j hasta la posición i, ahí encontraremos todos los números que son menores que lo que hay en la i-ésima posición del vector. Si el i-ésimo número es mayor que el j-ésimo número entonces en la i-ésima posición de nuestro arreglo que usaremos para hacer DP vamos a colocar el máximo entre lo que hay en la i-ésima posición de nuestro arreglo para DP y la j-ésima posición más uno. El realizar esto nos asegura que obtendremos siempre la mayor cantidad de números mayores a la i-ésima posición. Finalmente solo es necesario obtener el número más grande de nuestro arreglo que nos ayuda a hacer la DP

1	4	2	4	3
---	---	---	---	---

Inicialmente nuestro arreglo que utilizaremos para hacer DP esta lleno con unos, ya que un número por si mismo ya es una subsecuencia.

1	1	1	1	1
---	---	---	---	---

Arranca el programa y colocamos dos índices i será el número que vamos a comparar para saber si hay una subsecuencia, j iniciará siempre desde cero, e irá moviéndose hasta llegar a $i - 1$, como podemos ver, en efecto 4 es más grande que 1 por tanto colocamos en 4 el máximo entre nuestro arreglo para la DP en i y nuestro arreglo en j , más uno, en este caso nos conviene el segundo caso por tanto, tenemos lo siguiente

1	4	2	4	3
↑ j	↑ i			

1	2	1	1	1
---	---	---	---	---

Ahora movemos i y nuevamente, j se moverá desde $j = 0$ mientras $j < i$, en este caso tenemos nuevamente una subsecuencia al iterar j . $2 > 1$, es verdad, pero $2 > 4$ es falso, por tanto el $\max(DP[i], DP[j] + 1)$ será 2 nuevamente

1	4	2	4	3
↑ j		↑ i		

1	2	2	1	1
---	---	---	---	---

Nuevamente i se mueve, y j itera hasta $i - 1$, como podemos ver realizaremos las comparaciones $4 > 1$, $4 > 4$, $4 > 2$, y podemos ver que solo se cumplió la condición dos veces por tanto tenemos que en la posición i tomaremos el $\max(DP[i], DP[j] + 1)$

$4 > 1$? si

1	4	2	4	3
↑ j			↑ i	

1	2	2	2	1
---	---	---	---	---

$4 > 4$? No

1	4	2	4	3
	↑ j		↑ i	

1	2	2	2	1
---	---	---	---	---

4 > 2? Si

1	4	2	4	3
		↑ j	↑ i	

1	2	2	3	1
---	---	---	---	---

Finalmente i se mueve a la derecha y nuevamente j itera hasta $i - 1$,

3 > 1? Si

1	4	2	4	3
↑ j				↑ i

1	2	2	3	2
---	---	---	---	---

3 > 4? No

1	4	2	4	3
	↑ j			↑ i

1	2	2	3	2
---	---	---	---	---

3 > 2? Si

1	4	2	4	3
		↑ j		↑ i

1	2	2	3	3
---	---	---	---	---

3 > 4? No

1	4	2	4	3
			↑ j	↑ i

1	2	2	3	3
---	---	---	---	---

Código

```
#include <bits/stdc++.h>

using namespace std;

vector<int> numbers;

int ELIS() {

    int size = numbers.size();
    int ans = -1;
    vector< int > DP(size, 1);

    for(int i = 1; i < size; i++){
        for(int j = 0; j < i; j++){
            if(numbers[i] > numbers[j])
                DP[i] = max(DP[i], DP[j] + 1);
        }
    }
    for(int i = 0; i < size; i++)
        ans = max(ans, DP[i]);

    return ans;
}

int main(){
    int n;
    cin >> n;
    while(n--){
        int a;
        cin >> a;
        numbers.push_back(a);
    }
    cout << ELIS() << endl;
    return 0;
}
```

Nuevamente el modelo utilizado para solucionar este ejercicio es Botton up, ya que partimos de problemas sencillos hasta los más complejos

Captura

22727487		2018-11-18 22:54:21	Easy Longest Increasing Subsequence	accepted edit ideone it	0.00	16M	C++14
----------	---	------------------------	---	-----------------------------------	------	-----	-------

The knapsack problema

Descripción

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 ($1 \leq S \leq 2000$). You also have N ($1 \leq N \leq 2000$) 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?

Entrada

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.

Salida

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

Ejemplo

Entrada	Salida
4 5 1 8 2 4 3 0 2 5 2 3	13

Solución

Este problema lo podemos tratar similar al ejercicio del LCS, realizando una tabla, donde tengamos el número de elementos que se nos dan como entrada y el tamaño tope que tiene nuestra mochila. Lo que haremos es lo siguiente, si i o j son cero rellenamos con cero la posición (i, j) , si el elemento $i - 1$ es menor que una mochila de tamaño j , entonces en la posición (i, j) , vamos a guardar el máximo entre el i -ésimo objeto menos 1 más el valor que tenga nuestra tabla en su posición $(i-1, j - \text{el tamaño del objeto}[i-1])$, es decir el tamaño restante de nuestra mochila actual, y el otro valor es el que esté en la posición $(i-1, \text{el tamaño actual de la mochila "j"})$, si no se cumplió nada de lo anterior entonces el valor actual será el que esté en $(i-1, j)$, finalmente, nuestra respuesta se encontrará en la celda inferior derecha de nuestra tabla.

Código

```
#include <bits/stdc++.h>

using namespace std;

struct Item{
    int S;
    int V;
};

vector< Item > items;

int KNAPSACK(int S){
    int size = items.size();
    int DP[size + 1][S + 1];
```



```

//From the 0 to n item iterate
for(int i = 0; i <= size; i++){
    //From the 0 size to S size iterate
    for(int curr_s = 0; curr_s <= S; curr_s++){
        if(i == 0 || curr_s == 0)
            DP[i][curr_s] = 0;
        //I can put an item in the backpack
        else if(items[i - 1].S <= curr_s){
            DP[i][curr_s] = max(items[i - 1].V + DP[i-1][curr_s-items[i-1].S], DP[i-1][curr_s]);
        } else {
            DP[i][curr_s] = DP[i-1][curr_s];
        }
    }
}
return DP[size][S];
}

int main(){
    int S, N;
    cin >> S >> N;
    while(N--){
        Item item;
        cin >> item.S >> item.V;
        items.push_back(item);
    }
    cout << KNAPSACK(S) << endl;
    return 0;
}

```

Captura

22727694		2018-11-19 01:50:24	The Knapsack Problem	accepted edit ideone it	0.00	19M	CPP14
----------	--	------------------------	-------------------------	--	------	-----	-------