



INSTITUTO POLITÉCNICO  
NACIONAL

ESCUELA SUPERIOR DE  
CÓMPUTO



ANALISIS DE ALGORITMOS

PROFESOR TITULAR: FRANCO MARTINEZ  
EDGARDO ADRIAN

EJERCICIOS #11:  
DISEÑO DE SOLUCIONES MEDIANTE  
PROGRAMACIÓN DINÁMICA

LEMUS RUIZ MARIANA ELIZABETH  
2020630211

GRUPO: 3CM12



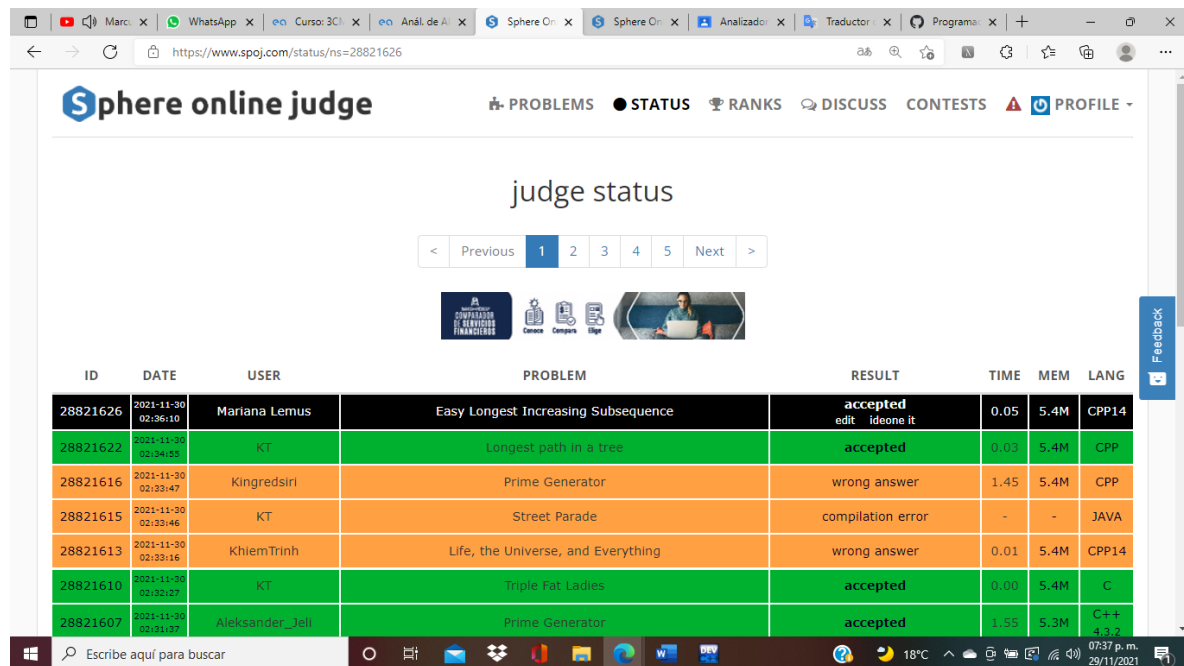
## EJERCICIOS 11: DISEÑO DE SOLUCIONES MEDIANTE PROGRAMACION DINAMICA

### INSTRUCCIONES:

De los siguientes 10 problemas que se plantean resolver al menos 4 problemas para completar el ejercicio.

### EJERCICIO 01: ELIS - Easy Longest Increasing Subsequence

#### Captura de aceptación



The screenshot shows the 'judge status' page on the Sphere Online Judge website. The page displays a table of problem-solving results for user Mariana Lemus. The table includes columns for ID, DATE, USER, PROBLEM, RESULT, TIME, MEM, and LANG. The first row shows a successful submission for 'Easy Longest Increasing Subsequence' (Problem 28821626) using C++14, with a time of 0.05s and memory of 5.4M. Other rows show various other problems and their results, including 'Longest path in a tree', 'Prime Generator', 'Street Parade', 'Life, the Universe, and Everything', 'Triple Fat Ladies', and another 'Prime Generator'.

ID	DATE	USER	PROBLEM	RESULT	TIME	MEM	LANG
28821626	2021-11-30 02:36:10	Mariana Lemus	Easy Longest Increasing Subsequence	accepted edit ideone it	0.05	5.4M	C++14
28821622	2021-11-30 02:34:55	KT	Longest path in a tree	accepted	0.03	5.4M	CPP
28821616	2021-11-30 02:33:47	Kingredsiri	Prime Generator	wrong answer	1.45	5.4M	CPP
28821615	2021-11-30 02:33:46	KT	Street Parade	compilation error	-	-	JAVA
28821613	2021-11-30 02:33:16	KhlemTrinh	Life, the Universe, and Everything	wrong answer	0.01	5.4M	C++14
28821610	2021-11-30 02:32:27	KT	Triple Fat Ladies	accepted	0.00	5.4M	C
28821607	2021-11-30 02:31:37	Aleksander_Jeli	Prime Generator	accepted	1.55	5.3M	C++14

### Explicación de solución y Análisis de Complejidad

#### Descripción

Dada una lista de números  $A$ , la longitud de la subsecuencia creciente más larga. Una subsecuencia creciente se define como un conjunto  $\{i_0, i_1, i_2, i_3, \dots, i_k\}$  tal que  $0 \leq i_0 < i_1 < i_2 < i_3 < \dots < i_k < N$  y  $A[i_0] < A[i_1] < A[i_2] < \dots < A[i_k]$ . Una subsecuencia creciente más larga es una subsecuencia con el máximo  $k$  (longitud).

Es decir, en la lista  $\{33, 11, 22, 44\}$

La subsecuencia  $\{33, 44\}$  y  $\{11\}$  son subsecuencias crecientes, mientras que  $\{11, 22, 44\}$  es la subsecuencia creciente más larga.

#### Entrada

La primera línea contiene un número N ( $1 \leq N \leq 10$ ) de la longitud de la lista A.

La segunda línea contiene N números ( $1 \leq \text{cada número} \leq 20$ ), los números de la lista A separados por espacios.

## Salida

Una línea que contiene la longitud de la subsecuencia creciente más larga en A.

## Código:

```
#include <iostream>
#include <vector>
using namespace std;

int maximoN(vector <int> DP, int
tam)
{
    int maximo = -1e9;
    for(int i = 0 ; i < tam ; i++)
        maximo = max(DP[i],
maximo);
    return maximo;
}

int lis(vector <int> numeros, int
tam)
{
    vector <int> DP(tam + 1);
    for(int i = 0 ; i <= tam ;
i++)
    {
        DP[i] = 1;

        for(int j = 0 ; j < i ;
j++)
            if(numeros[i] >
numeros[j] && DP[i] < DP[j] + 1)
                DP[i] = DP[j] + 1;
    }
    return maximoN(DP, tam);
}

int main(void)
{
    ios::sync_with_stdio(false);
    int num, auxI;
    cin >> num;
```

Primero se recibe el arreglo resultante y el tamaño del arreglo y se regresa el mayor de estos.

Después se recibe el arreglo de los números a los que se le quiere sacar la subsecuencia mayor y el tamaño de este arreglo.

Entonces se regresa el tamaño de la subsecuencia mayor donde todos los valores van incrementando.

Luego se cuenta la posición actual y se guarda en el arreglo que se usa para la DP.

Finalmente se van checando todos los valores antes y se va asegurando que el valor actual sea mayor que estos y se mantiene el máximo de la DP en el valor actual y la DP en los índices menores más uno y el actual.

<pre>vector&lt;int&gt; numeros; for(int i = 0 ; i &lt; num ; i++) {     cin &gt;&gt; auxI;     numeros.push_back(auxI); } cout &lt;&lt; lis(numeros, num) &lt;&lt; "\n"; }</pre>	
--	--

## EJERCICIO 02: BACTERIAS

### Captura de aceptación

omegaUp Concursos Cursos Problemas Ranking Ayuda MLeMusRuiz

compiler.out/err logs.txt files.zip

1

Envíos

Enviado	GUID	Estatus	Porcentaje	Lenguaje	Memoria	Tiempo	Acciones
2021-11-29 17:01:22	14189d03	AC	100.00%	cpp17-gcc	3.61 MB	0.02 s	
Nuevo envío							

Opiniones de los usuarios

Calidad Dificultad

Muy

Escribe aquí para buscar

23°C Soleado 05:29 p.m. 29/11/2021

## Explicación de solución y Análisis de Complejidad

### Descripción

Un grupo de biólogos computacionales ha diseñado un experimento para decidir si una colonia de microbios es capaz de resolver problemas cuando se le estimula de ciertas formas específicas. En este experimento se construye un recipiente con la forma de una cuadrícula rectangular con  $m$  renglones y  $n$  columnas. En cada uno de los cuadritos de la cuadrícula se coloca cierta cantidad de un compuesto químico que le es muy desagradable a los microbios y que, por lo tanto, los microbios preferirían evitar lo más posible. El recipiente está inclinado de tal forma que los microbios solo pueden avanzar hacia el este o hacia el sur. Por supuesto, los microbios tampoco pueden salir del recipiente. Al principio la colonia de microbios está localizada en el cuadrito correspondiente al primer renglón y primera columna del recipiente. Al final se espera que la colonia de microbios termine en el cuadrito correspondiente al último renglón y última columna del recipiente. En términos de un sistema de coordenadas, los microbios comienzan en la coordenada  $(1, 1)$  y terminan en la coordenada  $(m, n)$ . Antes de llevar a cabo el experimento, los científicos desean calcular la cantidad  $c$  de unidades del compuesto químico que deberá soportar la colonia en su trayecto, esto es, la suma de todas las cantidades del compuesto químico que fueron depositadas en todos los cuadritos por los que pasen. En el ejemplo se muestra un recipiente donde el mínimo valor posible de  $c$  es 17.

## Entrada

Dos enteros  $m$  y  $n$  separados por un espacio, seguidos de  $m$  renglones cada uno con  $n$  enteros separados por espacios. Estos enteros representan las cantidades del compuesto químico. Puedes suponer que  $2 \leq m \leq 100$ ,  $2 \leq n \leq 100$  y que todas las demás cantidades están entre 1 y 9, incluyéndolos.

## Salida

Deberás escribir en pantalla un entero  $c$  el cual representa la cantidad mínima del compuesto químico al que puede estar expuesto la colonia durante su recorrido. Consideraciones Tu programa se evaluará con varios casos de prueba.

## Código:

```
#include <iostream>
#include <vector>
using namespace std;

int M, N, auxI;

int caminoMenor(vector
<vector<int> > DP, vector
<vector<int> > quimico)
{
    for(int i = 1 ; i < M ; i++)
        for(int j = 1 ; j < N ;
j++)

        DP[i][j] =
quimico[i][j] + min(DP[i - 1][j],
DP[i][j - 1]);
    return DP[M - 1][N - 1];
}

int main(void)
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cin >> M >> N;
    vector <vector <int> >
compuestoQ(M, vector <int> (N,
0));
    vector <vector <int> > DP(M,
vector <int> (N));
    int res;
    for(int i = 0 ; i < M ; i++)
    {
```

Se empieza recibiendo el arreglo de las cantidades del químico en la tabla y el arreglo DP con el que se hará la programación dinámica para sacar el camino menos costoso por el que pueden pasar las bacterias.

Después se va checando el valor entre la casilla de arriba y la casilla de la derecha ya que las bacterias solo pueden moverse hacia el este y hacia el sur.

Luego se empieza con la primera casilla que es donde empieza el recorrido.

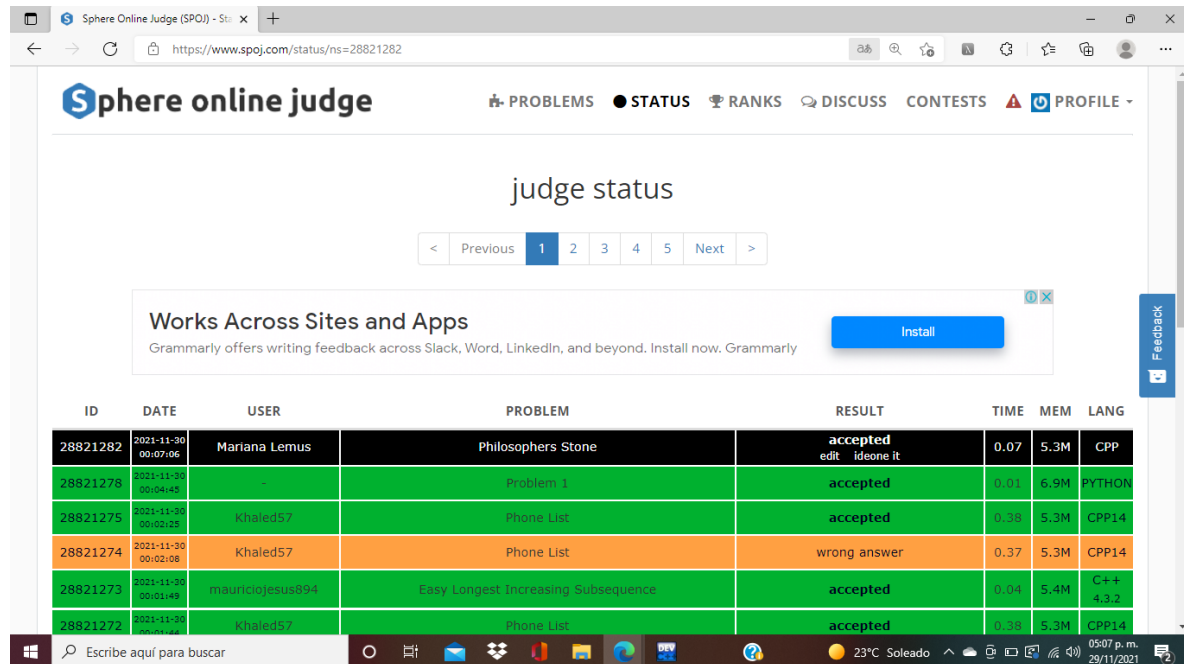
Entonces se va haciendo un barrido acumulado sobre la primera columna para poder saber cuánto cuesta moverse hacia abajo.

Y finalmente se va haciendo un barrido acumulado sobre la primera columna para poder saber cuánto cuesta moverse hacia la derecha.

```
        for(int j = 0 ; j < N ;  
j++)  
        {  
            cin >> auxI;  
            compuestoQ[i][j] =  
auxI;  
        }  
  
        DP[0][0] = compuestoQ[0][0];  
  
        for(int i = 1 ; i < M ; i++)  
            DP[i][0] = DP[i - 1][0] +  
compuestoQ[i][0];  
  
        for(int i = 1 ; i < N ; i++)  
            DP[0][i] = DP[0][i - 1] +  
compuestoQ[0][i];  
        res = caminoMenor(DP,  
compuestoQ);  
        cout << res << "\n";  
    }
```

## EJERCICIO 03: BYTESM2 - PHILOPHERS STONE

### Captura de aceptación



ID	DATE	USER	PROBLEM	RESULT	TIME	MEM	LANG
28821282	2021-11-30 00:07:06	Mariana Lemus	Philosophers Stone	accepted edit ideone it	0.07	5.3M	CPP
28821278	2021-11-30 00:04:49		Problem 1	accepted	0.01	6.9M	PYTHON
28821275	2021-11-30 00:02:25	Khaled57	Phone List	accepted	0.38	5.3M	CPP14
28821274	2021-11-30 00:02:08	Khaled57	Phone List	wrong answer	0.37	5.3M	CPP14
28821273	2021-11-30 00:01:49	mauriciojesus994	Easy Longest Increasing Subsequence	accepted	0.04	5.4M	C++ 4.3.2
28821272	2021-11-30 00:01:49	Khaled57	Phone List	accepted	0.38	5.3M	CPP14

## Explicación de solución y Análisis de Complejidad

### Descripción

Una de las cámaras secretas de Hogwarts está llena de piedras filosofales. El piso de la cámara está cubierto por baldosas cuadradas  $h \times w$ , donde hay  $h$  filas de baldosas desde la parte delantera (primera fila) hasta la parte posterior (última fila) y  $w$  columnas de baldosas de izquierda a derecha. Cada baldosa tiene de 1 a 100 piedras. Harry tiene que agarrar tantas piedras filosofales como sea posible, sujeto a las siguientes restricciones:

- Comienza eligiendo cualquier azulejo en la primera fila, y recoge las piedras filosofales en ese azulejo. Luego, se mueve a una baldosa en la siguiente fila, recoge las piedras filosofales en la baldosa, y así sucesivamente hasta llegar a la última fila.
- Cuando se mueve de una baldosa a una baldosa en la siguiente fila, solo puede moverse a la baldosa justo debajo de ella o diagonalmente a la izquierda o a la derecha.

Dados los valores de  $h$  y  $w$ , y el número de piedras filosofales en cada baldosa, escriba un programa para calcular el máximo número posible de piedras filosofales que Harry puede agarrar en un solo viaje desde la primera fila hasta la última fila.

### Entrada



La primera línea consiste en un solo entero  $T$ , el número de casos de prueba. En cada uno de los casos de prueba, la primera línea tiene dos enteros. El primer entero  $h$  ( $1 \leq h \leq 100$ ) es el número de filas de baldosas en el piso. El segundo entero  $w$  ( $1 \leq w \leq 100$ ) es el número de columnas de baldosas en el piso. A continuación, hay líneas  $h$  de entradas. La  $i$ -ésima línea de estos, especifica el número de piedras filosofales en cada baldosa de la  $i$ -ésima fila desde el frente. Cada línea tiene  $w$  enteros, donde cada entero  $m$  ( $0 \leq m \leq 100$ ) es el número de piedras filosofales en esa baldosa. Los enteros están separados por un carácter de espacio.

### Salida

La salida debe consistir en líneas  $T$ , ( $1 \leq T \leq 100$ ), una para cada caso de prueba. Cada línea consiste en un solo entero, que es el número máximo posible de piedras filosofales que Harry puede agarrar, en un solo viaje desde la primera fila hasta la última fila para el caso de prueba correspondiente.

### Código:

```
#include <iostream>
#include <vector>
using namespace std;

int columns, rows;

int sacaMax(vector <int>
valoresFin)
{
    int maximo = -1e9;
    for(auto a : valoresFin)
        maximo = max(maximo, a);
    return maximo;
}

int piedrasMaximas(vector
<vector<int> > piedras, vector
<vector<int> > DP)
{
    int maximo;
    for(int i = 1 ; i < rows ;
i++)
    {
        for(int j = 1 ; j <=
columns ; j++)
        {
            maximo = -1e9;
            maximo = max(maximo,
DP[i - 1][j - 1]);
            maximo = max(maximo,
DP[i - 1][j]);
            maximo = max(maximo,
DP[i - 1][j + 1]);
```

La Función recibe el arreglo donde están los resultados de la DP y de realizar el proceso para sacar el valor máximo de piedras que se pueden recolectar.

Después se va checando cuáles son los mayores valores directamente arriba o a la diagonal derecha y la diagonal izquierda para poder escoger el máximo número de piedras, y posteriormente se suma con el valor del número de piedras en la casilla actual.

Finalmente se guarda en el arreglo para la programación dinámica la primera fila del arreglo de entrada ya que esta es la que empieza todo el proceso.

```
        DP[i][j] = maximo +
pedras[i][j];
    }

    }

    return sacaMax(DP[rows - 1]);
}

int main(void)
{
    int cases, auxI;
    cin >> cases;
    while(cases--)
    {
        cin >> rows >> columns;
        vector <vector<int> >
pedras(rows, vector<int> (columns
+ 2, 0));
        vector <vector<int> >
DP(rows, vector<int> (columns + 2,
0));
        for(int i = 0 ; i < rows ;
i++)
            for(int j = 1 ; j <=
columns ; j++)
                cin >>
pedras[i][j];

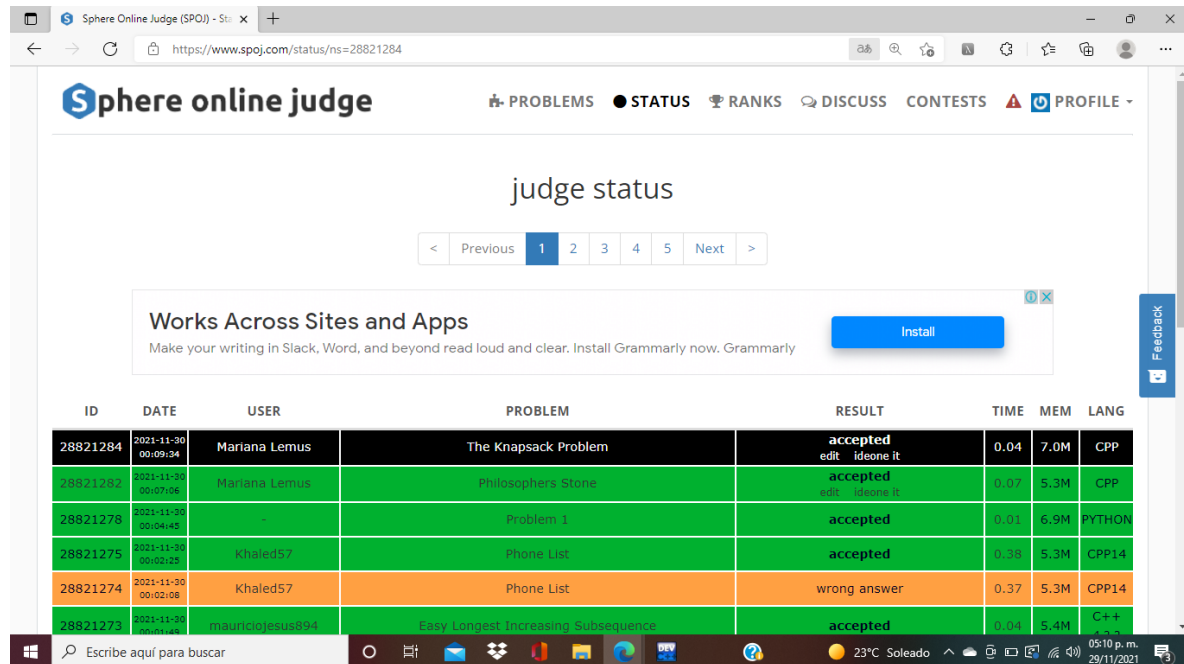
                for(int i = 1 ; i <=
columns ; i++)
                    DP[0][i] =
pedras[0][i];
                cout <<
pedrasMaximas(pedras, DP) <<
"\n";
    }

}
```

$$T(n) = \begin{cases} 0, & \text{si } n=1 \\ 2T\left(\frac{n}{2}\right) + n, & \text{si } n > 1 \end{cases}$$
$$T(n) \in O(n \log(n))$$

## EJERCICIO 02: KNAPSACK - THE KNAPSACK PROBLEM

### Captura de aceptación



ID	DATE	USER	PROBLEM	RESULT	TIME	MEM	LANG
28821284	2021-11-30 00:09:34	Mariana Lemus	The Knapsack Problem	accepted edit ideone it	0.04	7.0M	CPP
28821283	2021-11-30 00:07:06	Mariana Lemus	Philosophers Stone	accepted edit ideone it	0.07	5.3M	CPP
28821278	2021-11-30 00:04:45	-	Problem 1	accepted	0.01	6.9M	PYTHON
28821275	2021-11-30 00:02:25	Khaled57	Phone List	accepted	0.38	5.3M	CPP14
28821274	2021-11-30 00:02:08	Khaled57	Phone List	wrong answer	0.37	5.3M	CPP14
28821273	2021-11-30 00:01:49	mauriciojesus894	Easy Longest Increasing Subsequence	accepted	0.04	5.4M	C++

## Explicación de solución y Análisis de Complejidad

### Descripción

El famoso problema de la mochila. Está empacando para unas vacaciones en el lado del mar y va a llevar solo una bolsa con capacidad  $S$  ( $1 \leq S \leq 2000$ ). También tiene  $N$  ( $1 \leq N \leq 2000$ ) que es posible que desee llevar con usted al lado del mar. Desafortunadamente no puedes caber todos ellos en la mochila por lo que tendrás que elegir. Para cada artículo se le da su tamaño y su valor. Desea maximizar el valor total de todos los artículos que va a traer. ¿Cuál es este valor total máximo?

### Entrada

En la primera línea se le dan  $S$  y  $N$ .  $N$  líneas siguen con dos enteros en cada línea que describen uno de sus elementos. El primer número es el tamaño del artículo y el siguiente es el valor del artículo.

### Salida

Debe generar un solo entero en uno como: el valor máximo total de la mejor opción de artículos para su viaje.

### Código:

```
#include <iostream>
#include <vector>
#include <utility>
#include <algorithm>
using namespace std;

int maxValor(vector <pair<int,
int> > objetos, int capacidad)
{
    vector <vector<int> >
DP(objetos.size() + 1, vector
<int> (capacidad + 1,-1));
    for(int i = 0 ; i <=
objetos.size() ; i++)
    {
        for(int j = 0 ; j <=
capacidad ; j++)
        {

            if(j == 0 || i == 0)
                DP[i][j] = 0;

            else if(objetos[i -
1].first <= j)

                DP[i][j] =
max(objetos[i - 1].second + DP[i -
1][j - objetos[i - 1].first],
                DP[i - 1][j]);

            else
                DP[i][j] = DP[i -
1][j];
        }
    }
    return
DP[objetos.size()][capacidad];
}

int main(void)
{
    int capacidad, objetos,
auxValor, auxPeso;
    cin >> capacidad >> objetos;

    vector <pair<int, int> >
cosas;
    for(int i = 0 ; i < objetos ;
i++)
    {
        cin >> auxPeso >>
auxValor;

        cosas.push_back(make_pair(auxPeso,
auxValor));
    }
}
```

Primero se va llevando a cabo la DP en donde se tiene como base en donde no se tiene ningún objeto en la mochila.

Entonces si el peso del objeto actual es menor que la capacidad actual de la mochila se debe de actualizar y evaluar la tabla en la posición actual.

Si no se saca el máximo entre el valor del objeto anterior más DP para el peso actual y el valor de la DP en el objeto anterior. En pocas palabras, entre tomar el objeto actual y no tomarlo.

También si el valor actual es mayor que la capacidad de la mochila, entonces no se toma.

Para la capacidad se guardan el peso y el valor de los objetos que se pueden seleccionar para llenar la mochila. Se guarda en un arreglo de pares en donde el primer valor del par va a ser el peso y el segundo cuánto vale este objeto.

Finalmente se ordena el arreglo con respecto al peso, esto para saber poder llenar la tabla de la programación dinámica.

<pre>    }      sort(cosas.begin(), cosas.end());     cout &lt;&lt; maxValor(cosas, capacidad) &lt;&lt; "\n"; }</pre>	
---	--