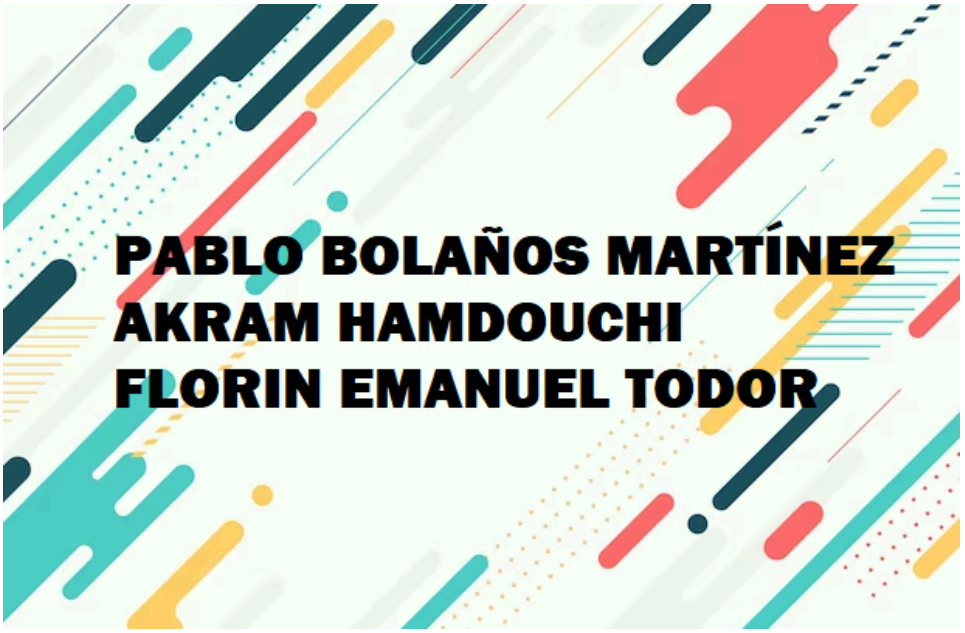
UNIVERSIDAD DE GRANADA

Algorítmica

2º Grado en Ingeniería Informática Y ADE

PRÁCTICA 4: Programación Dinámica



ÍNDICE

INTRODUCCIÓN

Este es un problema clásico conocido como el "**problema de la mochila**" que se puede resolver utilizando la técnica de programación dinámica. El objetivo es maximizar el valor de los artículos que se pueden colocar en una mochila con una capacidad de peso limitada.

EJERCICIO 1

(¿Se puede resolver con Programación Dinámica?)

Para determinar si el problema de la mochila cumple con los requisitos para ser resuelto mediante la técnica de programación dinámica, debemos verificar dos características fundamentales: estructura óptima y solapamiento de subproblemas.

1. **Estructura óptima:** El problema de la mochila cumple con la estructura óptima, lo que significa que una solución óptima global se puede construir a partir de soluciones óptimas de subproblemas. En este caso, al considerar el valor máximo de los artículos que se pueden agregar a la mochila, podemos dividir el problema en subproblemas más pequeños y combinar sus soluciones para obtener la solución óptima global.

2. **Solapamiento de subproblemas:** El problema de la mochila también cumple con el solapamiento de subproblemas, lo que implica que las soluciones a subproblemas se superponen entre sí. En este caso, al calcular el valor máximo de los artículos para cada capacidad de peso, es probable que se realicen cálculos repetidos para diferentes combinaciones de artículos y capacidades de peso. La programación dinámica permite evitar estos cálculos redundantes almacenando las soluciones parciales en una tabla.

Dado que el problema de la mochila cumple con ambas características, es adecuado para ser resuelto mediante la técnica de programación dinámica. Podemos proceder a plantear la ecuación recurrente, diseñar el algoritmo y aplicarlo para obtener la solución óptima.

EJERCICIO 2

(Plantear el problema como una ecuación recurrente)

Sea K(i, w) la función que devuelve el valor máximo que se puede obtener considerando los primeros i artículos y una capacidad de peso w.

La ecuación recurrente es:

K(i, w) = max{K(i-1, w), vi + K(i-1, w-wi)}

$$T[i][j] = \max\{T[i-1][j], b_i + T[i-1][j-w_i]\}$$

Explicación de la ecuación recurrente:

- *Si el peso del artículo i, wi,* ***es menor o igual a la capacidad de peso w****, tenemos dos opciones:*
    1. *No incluir el artículo i en la mochila, lo que nos lleva a considerar los i-1 artículos anteriores con la misma capacidad de peso w. Esto se representa mediante K(i-1, w).*

    2. *Incluir el artículo i en la mochila, lo que reduce la capacidad de peso disponible a w-wi y nos lleva a considerar los i-1 artículos anteriores. Agregamos el valor del artículo i, vi, al valor máximo obtenido en esa situación. Esto se representa mediante vi + K(i-1, w-wi). Tomamos el máximo entre las dos opciones anteriores para obtener el valor máximo que se puede obtener considerando los primeros i artículos y una capacidad de peso w.*

- *Si el peso del artículo i, wi,* ***es mayor que la capacidad de peso w****, no podemos incluir el artículo i en la mochila, ya que excedería la capacidad. En este caso, simplemente consideramos los i-1 artículos anteriores con la misma capacidad de peso w, lo cual se representa mediante K(i-1, w).*

EJERCICIO 3

(Representación de la ec. para almacenar las sol. parciales en tabla)

Para almacenar las soluciones parciales en forma de tabla, se utiliza una matriz bidimensional. La matriz se denomina comúnmente como una "tabla de programación dinámica" o "tabla DP".

¿CÓMO SE RELLENA?

El paso inicial es llenar la primera fila y la primera columna de la tabla con ceros, ya que cuando no se tienen artículos o no hay capacidad de peso, el valor máximo posible será cero.

A continuación, se procede a llenar el resto de la tabla siguiendo la ecuación recurrente mencionada anteriormente. Se inicia desde la celda (1, 1) y se continúa de izquierda a derecha y de arriba hacia abajo.

Para cada celda (i, w) de la tabla:

1. Si el peso del artículo i, wi, es mayor que la capacidad de peso w, se copia el valor de la celda superior, es decir, K(i, w) = K(i-1, w).
2. Si el peso del artículo i, wi, es menor o igual a la capacidad de peso w, se calcula el máximo valor posible considerando dos opciones: a. No incluir el artículo i: K(i, w) = K(i-1, w). b. Incluir el artículo i: K(i, w) = vi + K(i-1, w-wi). Se selecciona el máximo entre las dos opciones anteriores y se almacena en la celda (i, w).

El proceso de llenado continúa hasta que todas las celdas de la tabla se hayan completado. Al finalizar, la celda en la esquina inferior derecha (n, W) contendrá el valor máximo que se puede obtener considerando todos los artículos y la capacidad de peso máxima.

Tamaño de la mochila = 8											
Objeto	**Peso (kg)**	**Valor ($)**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Estado Inicial			0	0	0	0	0	0	0	0	0
A	1	2	0	2 A	2 A	2 A	2 A	2 A	2 A	2 A	2 A
B	2	5	0	2 A	5 B	7 A+B	7 A+B	7 A+B	7 A+B	7 A+B	7 A+B
C	4	6	0	2 A	5 B	7 A+B	7 A+B	8 A+C	11 B+C	13 A+B+C	13 A+B+C
D	5	10	0	2 A	5 B	7 A+B	7 A+B	10 D	12 A+D	15 B+D	17 A+B+D
E	7	13	0	2 A	5 B	7 A+B	7 A+B	10 D	12 A+D	15 B+D	17 A+B+D
F	8	16	0	2 A	5 B	7 A+B	7 A+B	10 D	12 A+D	15 B+D	17 A+B+D

EJERCICIO 4

(Comprobar el cumplimiento del P.O.B)

El P.O.B. (Principio de Optimalidad de Bellman) establece que una solución óptima a un problema se puede construir a partir de soluciones óptimas a subproblemas más pequeños.

Esto se puede resumir de la siguiente manera:

- El P.O.B. se cumple en el problema de la mochila resuelto mediante programación dinámica.
- El P.O.B. establece que una solución óptima a un problema se puede construir a partir de soluciones óptimas a subproblemas más pequeños.
- En el problema de la mochila, la ecuación recurrente utilizada para calcular el valor máximo de los artículos en la mochila se basa en la elección óptima en cada paso. Cada artículo se considera individualmente y se elige la opción que maximiza el valor total. Esto se hace tomando en cuenta las soluciones óptimas obtenidas previamente para subproblemas más pequeños.
- Al seguir esta estrategia y tomar decisiones óptimas en cada paso, se garantiza que no se pierdan oportunidades para obtener la solución óptima global.
- *En resumen, el P.O.B. se cumple en el problema de la mochila resuelto mediante programación dinámica, ya que se toman decisiones óptimas en cada paso basadas en soluciones óptimas previas, lo que conduce a una solución global óptima.*

EJERCICIO 5

(Diseño del algoritmo)

El algoritmo de programación dinámica para resolver el problema de la mochila se puede diseñar de la siguiente manera:

1. Definir una función MochilaPD que toma como parámetros el número de artículos n, la capacidad de peso máxima W, y las listas de valores y pesos de los artículos.
2. Crear una matriz tabla de dimensiones (n + 1) x (W + 1) para almacenar las soluciones parciales.
3. Llenar la primera fila y columna de la matriz con ceros.
4. Para cada i de 1 a n:
    - Para cada w de 1 a W:
        - Si el peso del artículo i, pesos[i], es menor o igual a w:
            - Calcular el máximo entre no incluir el artículo i (valor en tabla[i-1][w]) e incluirlo (valor en valores[i] + tabla[i-1][w-pesos[i]]).
            - Almacenar el resultado en tabla[i][w].
        - Si el peso del artículo i es mayor que w, copiar el valor de la celda superior tabla[i-1][w] en tabla[i][w].
5. Devolver el valor en la celda tabla[n][W], que representa el valor máximo que se puede obtener considerando todos los artículos y la capacidad de peso máxima.

El algoritmo sigue una estrategia iterativa, llenando la tabla de programación dinámica en orden creciente de i y w, evitando recalcular los mismos subproblemas y almacenando las soluciones parciales para su uso posterior.

Este diseño de algoritmo de programación dinámica permite resolver eficientemente el problema de la mochila y encontrar la combinación óptima de artículos que maximiza el valor total dentro de la capacidad de peso establecida.

PSEUDOCÓDIGO:

```
Función MochilaPD(n, W, valores[], pesos[]):
    Crear una matriz tabla con dimensiones (n + 1) x (W + 1)

    // Paso inicial: Llenar la primera fila y columna con ceros
    Para cada i de 0 a n:
        tabla[i][0] = 0
    Para cada w de 0 a W:
        tabla[0][w] = 0

    // Llenar el resto de la tabla
    Para cada i de 1 a n:
        Para cada w de 1 a W:
            Si pesos[i] <= w:
                // Calcular el máximo entre no incluir el artículo i o incluirlo
                tabla[i][w] = max(tabla[i-1][w], valores[i] + tabla[i-1][w-pesos[i]])
            Sino:
                tabla[i][w] = tabla[i-1][w]

    Devolver tabla[n][W]
Fin de la función
```

MEJORA CON REESCALADO.

Hemos pensado en mejorar la eficiencia de nuestro programa pensando en el caso de que los valores introducidos por el usuario sean valores elevados.
Para ello, hemos hecho una suposición inicial en la que suponemos que los valores introducidos por el usuario siempre serán en gramos.

En este caso, establecemos un umbral superior y otro inferior, de manera que si los valores introducidos son superiores a los límites marcados, se realiza un reescalado de los valores y se devuelven con la unidad correspondiente.

Por ejemplo: Si el usuario introduce 1000 gramos, esto sería equivalente a guardar en la tabla 1 kg.
En el caso contrario de que el usuario introduzca 0.001 gramos esto sería equivalente a guardar en la tabla 1 miligramo.

Al final según la escala empleada, se devuelve el resultado en gramos, es decir el resultado * la escala empleada. A continuación presentamos el código de esta mejora:

```
Función MochilaPD(n, W, valores[], pesos[], umbralSuperior, umbralInferior):
    escala = 100  // Factor de escala inicial

    // Verificar si se supera el umbral superior o inferior
    si W > umbralSuperior o algún peso en pesos[] > umbralSuperior:
        escala = ceil(max(W, max(pesos[])) / umbralSuperior)  // Calcular el factor de escala superior
    sino si W < umbralInferior y todos los pesos en pesos[] < umbralInferior:
        escala = floor(umbralInferior / min(W, min(pesos[])))  // Calcular el factor de escala inferior

    // Aplicar el reescalado si es necesario
    si escala > 1:
        W_reescalado = W / escala
        pesos_reescalados[] = pesos[] / escala

        // Resto del algoritmo utilizando W_reescalado y pesos_reescalados
```

```
        // Llenar la tabla y calcular el resultado

            Devolver resultado * escala  // Multiplicar el resultado final por el factor de escala

    sino:
        // Resto del algoritmo sin cambios

        // Llenar la tabla y calcular el resultado

        Devolver resultado
Fin de la función
```

EJERCICIO 6

(Implementación del algoritmo)

CÓDIGO

```
int MochilaPD(int n, int W, const vector<int>& valores, const vector<int>&
pesos) {
    vector<vector<int>> tabla(n + 1, vector<int>(W + 1, 0));

    // Paso inicial: Llenar la primera fila y columna con ceros
    for (int i = 0; i <= n; i++) {
        tabla[i][0] = 0;
    }
    for (int w = 0; w <= W; w++) {
        tabla[0][w] = 0;
    }

    // Llenar el resto de la tabla
    for (int i = 1; i <= n; i++) {
        for (int w = 1; w <= W; w++) {
            if (pesos[i - 1] <= w) {
                //Calcular el máx entre no incluir el artículo i o incluirlo
                tabla[i][w] = max(tabla[i - 1][w], valores[i - 1] + tabla[i
- 1][w - pesos[i - 1]]);
            } else {
                tabla[i][w] = tabla[i - 1][w];
            }
        }
    }

    return tabla[n][W];
}
vector<int> getSelectedItems(const vector<vector<int>>& table, const
vector<int>& valores, const vector<int>& pesos) {
    vector<int> selectedItems; //Almacena los indices de los elementos
seleccionados para dar el resultado
    int i = table.size() - 1; //Tamaño de la tabla de filas
    int w = table[0].size() - 1; //Tamaño de la tabla de columnas

    while (i > 0 && w > 0) {
        if (table[i][w] != table[i - 1][w]) {
            selectedItems.push_back(i);
            w -= pesos[i - 1]; //Se resta la posición i porque se ha
incluido como solucion, esto se utiliza para pasar a la siguiente columna
        }
        i--; //Pasamos a la fila anterior de la tabla y continuamos buscando
    }

    return selectedItems;}
```

EJERCICIO 7

(Análisis de las Eficiencias)

EFICIENCIA TEÓRICA

Aquí hay un análisis de la eficiencia teórica de cada parte del código:

1. Llenar la tabla inicial con ceros: Este paso tiene una complejidad de tiempo de O(n * W), donde "n" es el número de elementos y "W" es la capacidad máxima de la mochila.
2. Llenar el resto de la tabla: Este paso tiene una complejidad de tiempo de O(n * W), ya que se recorren todas las celdas de la tabla y se realizan operaciones constantes en cada celda.
3. Imprimir la tabla generada: Este paso tiene una complejidad de tiempo de O(n * W), ya que se deben imprimir todas las celdas de la tabla.
4. Leer los valores y pesos del archivo: Este paso tiene una complejidad de tiempo de O(n), ya que se deben leer "n" valores y "n" pesos del archivo.

En general, la eficiencia teórica del algoritmo de la Mochila implementado es de O(n * W), donde "n" es el número de elementos y "W" es la capacidad máxima de la mochila. Esto significa que el tiempo de ejecución del algoritmo aumentará linealmente con el tamaño de entrada.

```
MochilaPD(n, W, valores, pesos)
    tabla = crearTabla(n + 1, W + 1, 0)  // O(n * W)

    // Paso inicial: Llenar la primera fila y columna con ceros
    para i desde 0 hasta n  // O(n)
        tabla[i][0] = 0
    fin para
    para w desde 0 hasta W  // O(W)
        tabla[0][w] = 0
    fin para

    // Llenar el resto de la tabla
    para i desde 1 hasta n  // O(n)
        para w desde 1 hasta W  // O(W)
            si pesos[i - 1] <= w entonces
                tabla[i][w] = max(tabla[i - 1][w], valores[i - 1]
+ tabla[i - 1][w - pesos[i - 1]])
            sino
                tabla[i][w] = tabla[i - 1][w]
            fin si
        fin para
    fin para

    imprimir "Tabla generada:"  // O(1)
    imprimirTabla(tabla)  // O(n * W)
    imprimir una línea en blanco  // O(1)

    retornar tabla[n][W]  // O(1)
fin función

main(argc, argv)
    n, W = leer valores desde argv  // O(1)
    valores = crear vector vacío  // O(1)
    pesos = crear vector vacío  // O(1)

    archivo = abrir archivo(argv[1])  // O(1)
    si archivo está abierto entonces
        leer n y W desde archivo  // O(1)

        // Leer los valores
        para i desde 0 hasta n  // O(n)
            leer valor desde archivo  // O(1)
```

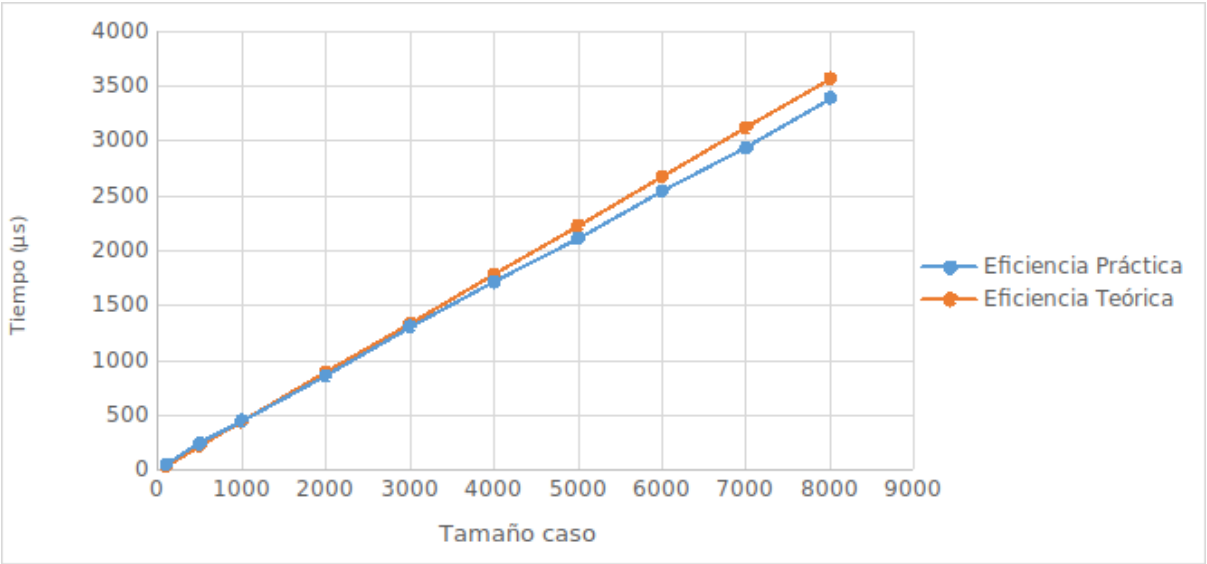
```
            agregar valor a pesos  // O(1)
        fin para

        // Leer los pesos
        para i desde 0 hasta n  // O(n)
            leer peso desde archivo  // O(1)
            agregar peso a valores  // O(1)
        fin para

        cerrar archivo  // O(1)

        maxValor = MochilaPD(n, W, valores, pesos)  // O(n * W)

        imprimir "El valor máximo que se puede obtener es:",
maxValor  // O(1)
    sino
        imprimir "No se pudo abrir el archivo."  // O(1)

    retornar 0  // O(1)
fin función
```

EFICIENCIA PRÁCTICA

```
void generarNumerosRandom(int n, int& W, vector<int>& valores, vector<int>&
pesos) {
    srand(static_cast<unsigned int>(time(0)));

    W = rand() % 101;

    valores.clear();
    pesos.clear();

    for (int i = 0; i < n; i++) {
        valores.push_back(rand() % 101);
        pesos.push_back(rand() % 101);
    }
}

int main() {
    ofstream datosFile("../tests/datos.txt"); // Contendrá toda la información
que se usa, n, w, pesos y valores
    ofstream muestraFile("../tests/muestra.txt"); //Contendrá el tamaño de
prueba y sus microsegundos
    ofstream tiemposFile("../tests/tiempos.txt");// Contendrá unicamente los
microsegundos, para realizar la gráfica

    if (!datosFile.is_open() || !muestraFile.is_open() ||
!tiemposFile.is_open()) {
        cout << "Error al abrir los archivos." << endl;
        return 1;
    }

    vector<int> tamaños = {100, 500, 1000, 2000, 3000,4000, 5000, 6000, 7000,
8000};

    for (int i =0; i<tamaños.size(); ++i) {
        int n = tamaños[i];
        int W;
        vector<int> valores, pesos;

        generarNumerosRandom(n, W, valores, pesos);

        datosFile << "Tamaño: " << n << endl;
        datosFile << "W: " << W << endl;
        datosFile << "Valores: ";
        for (int i =0;  i< valores.size(); ++i) {
            datosFile << valores[i] << " ";
        }
        datosFile << endl;
        datosFile << "Pesos: ";
        for (int i=0; i<pesos.size(); ++i) {
            datosFile << pesos[i] << " ";
        }
        datosFile << endl;
```

```
        auto start = high_resolution_clock::now();
        int maxValor = MochilaPD(n, W, valores, pesos);
        auto end = high_resolution_clock::now();

        auto duration = duration_cast<microseconds>(end - start);

        muestraFile << "Tamaño: " << n;
        muestraFile << " -> Tiempo: " << duration.count() << " microsegundos"
<< endl;

        tiemposFile << duration.count() << endl;

        datosFile << "Valor máximo obtenido: " << maxValor << endl;
        datosFile << endl;
    }

    datosFile.close();
    muestraFile.close();
    tiemposFile.close();

    return 0;
}
```

	A	B	C	D	E
	K =	**35**			K (media)
	Tam. Caso (N)	**f(N) = N*K**	**T(n) (µs)**	**K=T(n)/f(n)**	**Tiempo teórico estimado = Kmedia*f(n)**
	100	3500	53	0.01514285714	44.6280595238095
	500	17500	246	0.01405714286	223.140297619048
	1000	35000	454	0.01297142857	446.280595238095
	2000	70000	860	0.01228571429	892.56119047619
	3000	105000	1311	0.01248571429	1338.84178571429
	4000	140000	1716	0.01225714286	1785.12238095238
	5000	175000	2114	0.01208	2231.40297619048
	6000	210000	2545	0.01211904762	2677.68357142857
	7000	245000	2938	0.01199183673	3123.96416666667
	8000	280000	3393	0.01211785714	3570.24476190476
		K media	0.01275087415		



EJERCICIO 8

(Dos ejemplos de problema concretos)

Para la realización de los ejemplos, los ficheros de prueba deben de tener la siguiente estructura:

1. ***Valor de n //Nº de objetos que hay***
2. ***Valor de W // Tamaño máximo de la mochila***
3. ***Vector de enteros (pesos) // los pesos de cada objeto***
4. ***Vector de enteros (Valores) // los valores o beneficios de cada objeto***

EJEMPLO I

Tamaño de la mochila = 8											
Objeto	**Peso (kg)**	**Valor ($)**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Estado Inicial			0	0	0	0	0	0	0	0	0
A	1	2	0	2 A	2 A	2 A	2 A	2 A	2 A	2 A	2 A
B	2	5	0	2 A	5 B	7 A+B	7 A+B	7 A+B	7 A+B	7 A+B	7 A+B
C	4	6	0	2 A	5 B	7 A+B	7 A+B	8 A+C	11 B+C	13 A+B+C	13 A+B+C
D	5	10	0	2 A	5 B	7 A+B	7 A+B	10 D	12 A+D	15 B+D	17 A+B+D
E	7	13	0	2 A	5 B	7 A+B	7 A+B	10 D	12 A+D	15 B+D	17 A+B+D
F	8	16	0	2 A	5 B	7 A+B	7 A+B	10 D	12 A+D	15 B+D	17 A+B+D

Fichero: *ejemplo1.txt*:

```
6
8
1 2 4 5 7 8
2 5 6 10 13 16
```

Salida del código:

```
/home/flo/Desktop/ALG/ALG_PRACTICA_4/cmake-build-debug/ALG_PRACTICA_4 ../tests/ejemplo1.txt
Tabla generada:
0       0       0       0       0       0       0       0       0
0       2       2       2       2       2       2       2       2
0       2       5       7       7       7       7       7       7
0       2       5       7       7       8       11      13      13
0       2       5       7       7       10      12      15      17
0       2       5       7       7       10      12      15      17
0       2       5       7       7       10      12      15      17

Valores seleccionados:
Objeto 1: Valor = 2, Peso = 1
Objeto 2: Valor = 5, Peso = 2
Objeto 4: Valor = 10, Peso = 5

El valor máximo que se puede obtener es: 17
```

EJEMPLO II

Tamaño de la mochila = 6									
Objeto	**Peso (kg)**	**Valor ($)**	**0**	**1**	**2**	**3**	**4**	**5**	**6**
Estado Inicial			0	0	0	0	0	0	0
A	2	1	0	0	1	1	1	1	1
B	3	2	0	0	1	2	2	3	3
C	4	5	0	0	1	2	5	5	6

Fichero: *ejemplo2.txt*:

```
3
6
2 3 4
1 2 5
```

Salida del código:

```
/home/flo/Desktop/ALG/ALG_PRACTICA_4/cmake-build-debug/ALG_PRACTICA_4 ../tests/ejemplo2.txt
Tabla generada:
0       0       0       0       0       0       0
0       0       1       1       1       1       1
0       0       1       2       2       3       3
0       0       1       2       5       5       6

Valores seleccionados:
Objeto 1: Valor = 1, Peso = 2
Objeto 3: Valor = 5, Peso = 4

El valor máximo que se puede obtener es: 6

Process finished with exit code 0
```

EJEMPLO III

Fichero: *ejemplo3.txt*:

```
5
11
1 2 5 6 7
1 6 18 22 28
```

Salida del código:

```
/home/flo/Desktop/ALG/ALG_PRACTICA_4/cmake-build-debug/ALG_PRACTICA_4 ../tests/ejemplo3.txt
Tabla generada:
0       0       0       0       0       0       0       0       0       0       0       0
0       1       1       1       1       1       1       1       1       1       1       1
0       1       6       7       7       7       7       7       7       7       7       7
0       1       6       7       7       18      19      24      25      25      25      25
0       1       6       7       7       18      22      24      28      29      29      40
0       1       6       7       7       18      22      28      29      34      35      40

Valores seleccionados:
Objeto 3: Valor = 18, Peso = 5
Objeto 4: Valor = 22, Peso = 6

El valor máximo que se puede obtener es: 40
```