



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO



ANÁLISIS DE ALGORITMOS

PROFESOR: CRISTHIAN ALEJANDRO ÁVILA
SÁNCHEZ

PROYECTO:
PROBLEMA NP-COMPLETO – MOCHILA (KNAPSACK)

ALUMNO: GARAYOA FLORES ROBERTO
ALESSANDRO

GRUPO: 3CM11

1. Introducción

¿Qué son los problemas NP-Completos?

El concepto de "NP-completo" fue introducido por Stephen Cook en un artículo titulado «The complexity of theorem-proving procedures» en las páginas 151-158 de Proceedings of the 3rd Annual ACM Symposium on Theory of Computing en 1971, aunque el término "NP-completo" como tal no aparece en el documento. En la conferencia de ciencias de la computación hubo un intenso debate entre los científicos de la computación sobre si los problemas NP-completos podían ser resueltos en tiempo polinómico o en una máquina de Turing determinista. John Hopcroft llevó a todos los asistentes de la conferencia a consenso concluyendo que el estudio sobre si los problemas NP-completos son resolubles en tiempo polinómico debería ser pospuesto ya que nadie había conseguido probar formalmente sus hipótesis ni en un sentido ni en otro.

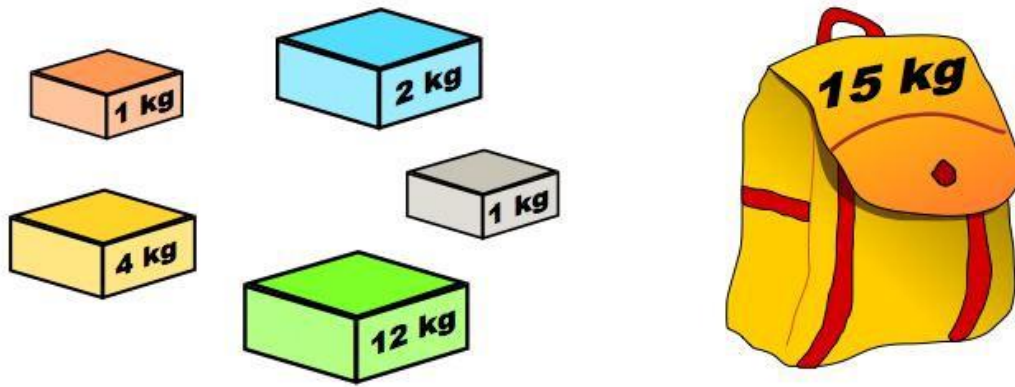
NP toma su nombre de "nondeterministic polynomial time", un término que remonta a las raíces de la Teoría de la Complejidad. Intuitivamente, significa que se puede encontrar y verificar una solución a cualquier problema de búsqueda en tiempo polinomial mediante un tipo de algoritmo especial (y bastante poco realista), llamado "Algoritmo no Determinista". Tal algoritmo tiene el poder de adivinar correctamente en cada paso.

KNAPSACK problem

En esencia, el problema de mochila, debes empaquetar un conjunto de elementos con valores y tamaños determinados (como pesos o volúmenes) en un contenedor con una capacidad máxima. Si el tamaño total de los elementos supera la capacidad, no puede empaquetarlos todos. En ese caso, el problema es elegir un subconjunto de los elementos de valor total máximo que cabe en el contenedor.

La investigación de operaciones (IO) se inició en Inglaterra durante la Segunda Guerra Mundial, cuando un equipo de científicos empezó a tomar decisiones respecto a la mejor utilización del material militar. Todas estas ideas se adaptaron para mejorar la eficiencia y productividad en la vida cotidiana.

La IO buscaba cuantificar un aspecto mejorable, donde la mejora va a estar condicionada por la toma de decisiones. Se pretende crear un modelo matemático que represente la realidad del problema y así poder obtener una solución factible y real para dicho problema. En este proyecto abarcaremos un modelo de optimización dinámica y combinatoria llamado Problema de la Mochila, "Knapsack Problem"(KP). Es uno de los 21 problemas NP-completos de Richard Karp, lo que quiere decir, de los 21 problemas no deterministas con dificultad para resolver. La formulación del problema es sencilla, pero la dificultad llega al intentar resolverlo computacionalmente.



2. Planteamiento del problema

Teniendo los "pesos" y "valores" (o se podría considerar como la "prioridad" del objeto) de N -artículos, coloque esos artículos en una mochila de capacidad "pesoMax" para obtener el máximo valor total en la mochila. En otras palabras, teniendo 2 arreglos de enteros "val[0..., N-1]" y "peso[0..., N-1]" los cuales representan los valores y pesos asociados con N artículos respectivamente. También teniendo un entero pesoMax el cual representa la capacidad de la mochila, encuentra el máximo valor dentro de val[], como la suma de pesos de los artículos ya sean pequeños o iguales a pesoMax. No puedes partir un objeto, se tiene que meter el objeto o no meterlo dentro de la mochila (0 ó 1).

3. Implementación

```

9  #include <stdio.h>
10
11
12 //Función que retorna el objeto máximo de 2 objetos/enteros
13 int max(int objct1, int objct2) { return (objct1 > objct2) ? objct1 : objct2; }
14
15 //Máximo valor que se puede poner en la mochila de capacidad pesoMax
16 //Necesitaremos el peso máximo de la mochila, el arreglo del peso y del valor (prioridad) y una constante n para el número de artículos
17 int mochila(int pesoMax, int peso[], int val[], int objcts){
18     /*Caso base:
19     Si no hay objetos para ingresar o no hay un peso máximo, retorna un 0 la función y ahí termina.*/
20     if (objcts == 0 || pesoMax == 0)
21         return 0;
22
23     /*Siguiente caso:
24     Si el N objeto es mayor a la capacidad de la mochila, entonces no será agregado a la mochila y regresará la mochila como está antes
25     del objeto.*/
26     if (peso[objcts - 1] > pesoMax)
27         return mochila(pesoMax, peso, val, objcts - 1);
28
29     /*Regresar alguno de los 2 casos...
30     Caso 1: El N-artículo SÍ es agregado
31     Caso 2: El N-artículo NO es agregado
32     */
33     else
34         return max(val[objcts - 1] + mochila(pesoMax - peso[objcts - 1], peso, val, objcts - 1), mochila(pesoMax, peso, val, objcts - 1));
35 }
36
37 //Main
38 int main(){
39     int val[] = {360, 83, 59, 130, 431, 67, 230, 52, 93, 125, 670, 892, 600, 38, 48, 147, 78, 256, 63, 17, 120, 164, 432, 35, 92, 110, 22, 42,
40     50, 323, 514, 28, 87, 73, 78, 15, 26, 78, 210, 36, 85, 189, 274, 43, 33, 10, 19, 389, 276, 312};
41     int peso[] = {7, 0, 30, 22, 80, 94, 11, 81, 70, 64, 59, 18, 0, 36, 3, 8, 15, 42, 9, 0, 42, 47, 52, 32, 26, 48, 55, 6, 29, 84, 2, 4, 18,
42     56, 7, 29, 93, 44, 71, 3, 86, 66, 31, 65, 0, 79, 20, 65, 52, 13};
43     int pesoMax = 90;
44     int objcts = sizeof(val)/sizeof(val[0]);
45     printf("%d", mochila(pesoMax, peso, val, objcts));
46     return 0;
47 }

```

```

#include <stdio.h>

//Función que retorna el objeto máximo de 2 objetos/enteros
int max(int objct1, int objct2) { return (objct1 > objct2) ? objct1 : objct2; }

//Máximo valor que se puede poner en la mochila de capacidad pesoMax
//Necesitaremos el peso máximo de la mochila, el arreglo del peso y del valor
(prioridad) y una constante n para el número de artículos
int mochila(int pesoMax, int peso[], int val[], int objects){
    /*Caso base:
        Si no hay objetos para ingresar o no hay un peso máximo, retorna un 0 la
función y ahí termina.*/
    if (objects == 0 || pesoMax == 0)
        return 0;

    /*Siguiente caso:
        Si el N objeto es mayor a la capacidad de la mochila, entonces no será
agregado a la mochila y regresará la mochila como está antes del objeto.*/
    if (peso[objects - 1] > pesoMax)
        return mochila(pesoMax, peso, val, objects - 1);

    /*Regresar alguno de los 2 casos...
        Caso 1: El N-artículo SÍ es agregado
        Caso 2: El N-artículo NO es agregado
    */
    else
        return max(val[objects - 1] + mochila(pesoMax - peso[objects - 1], peso, val,
objects - 1), mochila(pesoMax, peso, val, objects - 1));
}

//Main
int main(){
    int val[] = {360, 83, 59, 130, 431, 67, 230, 52, 93, 125, 670, 892, 600, 38, 48,
147, 78, 256, 63, 17, 120, 164, 432, 35, 92, 110, 22, 42, 50, 323, 514, 28, 87, 73,
78, 15, 26, 78, 210, 36, 85, 189, 274, 43, 33, 10, 19, 389, 276, 312};
    int peso[] = {7, 0, 30, 22, 80, 94, 11, 81, 70, 64, 59, 18, 0, 36, 3, 8, 15, 42,
9, 0, 42, 47, 52, 32, 26, 48, 55, 6, 29, 84, 2, 4, 18, 56, 7, 29, 93, 44, 71, 3, 86,
66, 31, 65, 0, 79, 20, 65, 52, 13};
    int pesoMax = 90;
    int objects = sizeof(val)/sizeof(val[0]);
    printf("%d", mochila(pesoMax, peso, val, objects));
    return 0;
}

```

4. Análisis de Complejidad

Comenzamos con el análisis de la complejidad con la primera línea del código:

```
//Función que retorna el objeto máximo de 2 objetos/enteros
int max(int objct1, int objct2) { return (objct1 > objct2) ? objct1 :
objct2; }
```

Teniendo en cuenta que la función solamente se encarga de comparar 2 objetos y retornar el de mayor valor, esta función siempre funcionará 1 sola vez; por lo consiguiente se puede decir que tiene una complejidad:

$O(1)$

```
int main(){
    int val[] = {360, 83, 59, 130, 431, 67, 230, 52, 93, 125, 670, 892, 600, 38, 48,
147, 78, 256, 63, 17, 120, 164, 432, 35, 92, 110, 22, 42,50, 323, 514, 28, 87, 73,
78, 15, 26, 78, 210, 36, 85, 189, 274, 43, 33, 10, 19, 389, 276,
312}; //O(1)
    int peso[] = {7, 0, 30, 22, 80, 94, 11, 81, 70, 64, 59, 18, 0, 36, 3, 8, 15, 42,
9, 0, 42, 47, 52, 32, 26, 48, 55, 6, 29, 84, 2, 4, 18, 56, 7, 29, 93, 44, 71, 3, 86,
66, 31, 65, 0, 79, 20, 65, 52, 13}; //O(1)
    int pesoMax = 90; //O(1)
    int objcts = sizeof(val)/sizeof(val[0]);
```

Siguiendo la misma lógica estas declaraciones de variables y de arreglos únicamente se ejecutarán 1 vez, por lo que de igual manera se puede entender que tienen una complejidad:

$O(1)$

Iniciando con el caso base inicial tenemos:

```
/*Caso base:
    Si no hay objetos para ingresar o no hay un peso máximo, retorna un 0 la
función y ahí termina.*/
    if (objcts == 0 || pesoMax == 0) //O(1)
        return 0;
```

Partiendo de la lógica de que si no hay objetos para meter a la mochila o no se especifica un peso máximo, va a retornar un 0 la función, esto lo hará 1 sola vez de igual forma, por lo que también tiene una complejidad:

$O(1)$

Para este caso tenemos la siguiente lógica

```
/*Siguiente caso:  
    Si el N objeto es mayor a la capacidad de la mochila, entonces no será  
    agregado a la mochila y regresará la mochila como está antes del objeto.*/  
if (peso[objcts - 1] > pesoMax)  
    return mochila(pesoMax, peso, val, objcts - 1);
```

Se tomaron 2 casos iniciales para considerar en esta condición

1: La mochila está por llenarse y tenemos que agregar 1 artículo más para llenarla por completo, si tenemos que alguno de los objetos que sobran sobrepasan el peso máximo que resta de la mochila, este no será agregado.

2: Si 1 sólo objeto pesa más que el peso máximo de la mochila no se agrega.

Como resumen, si el peso del objeto antes de llenarse la mochila (`peso[objcts - 1]`) es mayor al peso máximo de la mochila se retornará el peso máximo de la mochila antes de que se llene la mochila.

Esta condición puede ser:

$$T(n) = T(n - 1) + O(1)$$

Continuando con la 2da parte del condicional

```
else  
    return max(val[objcts - 1] + mochila(pesoMax - peso[objcts - 1], peso, val,  
objcts - 1), mochila(pesoMax, peso, val, objcts - 1));
```

Continuando con el condicional anterior pero ahora en la parte de qué pasa cuando el peso del último objeto por agregar NO es mayor que el peso máximo de la mochila y Sí se puede agregar. Bueno, en este caso lo que se debe hacer es retornar la suma del valor de los objetos con el peso que tiene la mochila en ese momento.

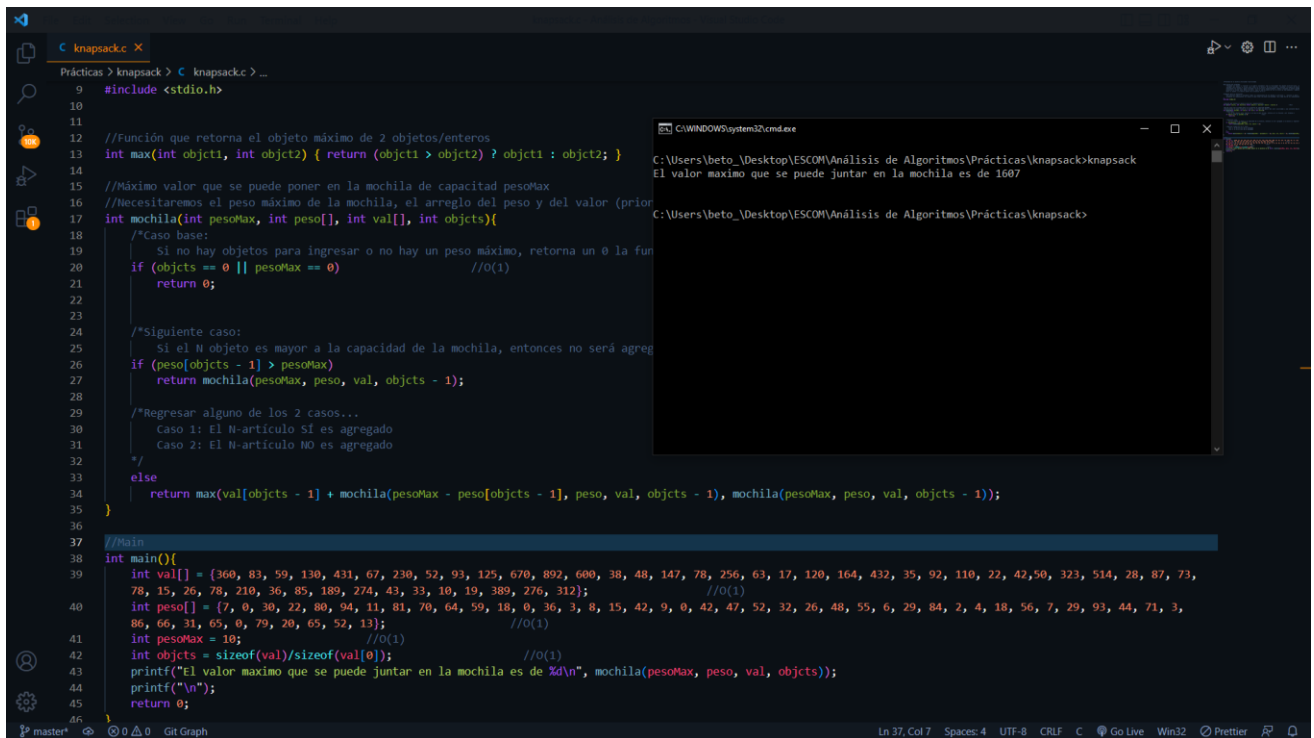
Para esta línea de código podemos encontrar una complejidad de:

$$\Sigma T(n) = T(n - 1) + O(1)$$

Ya que como bien sabemos, esta línea contiene al menos 2 llamadas recursivas a la función "mochila".

5. Pruebas

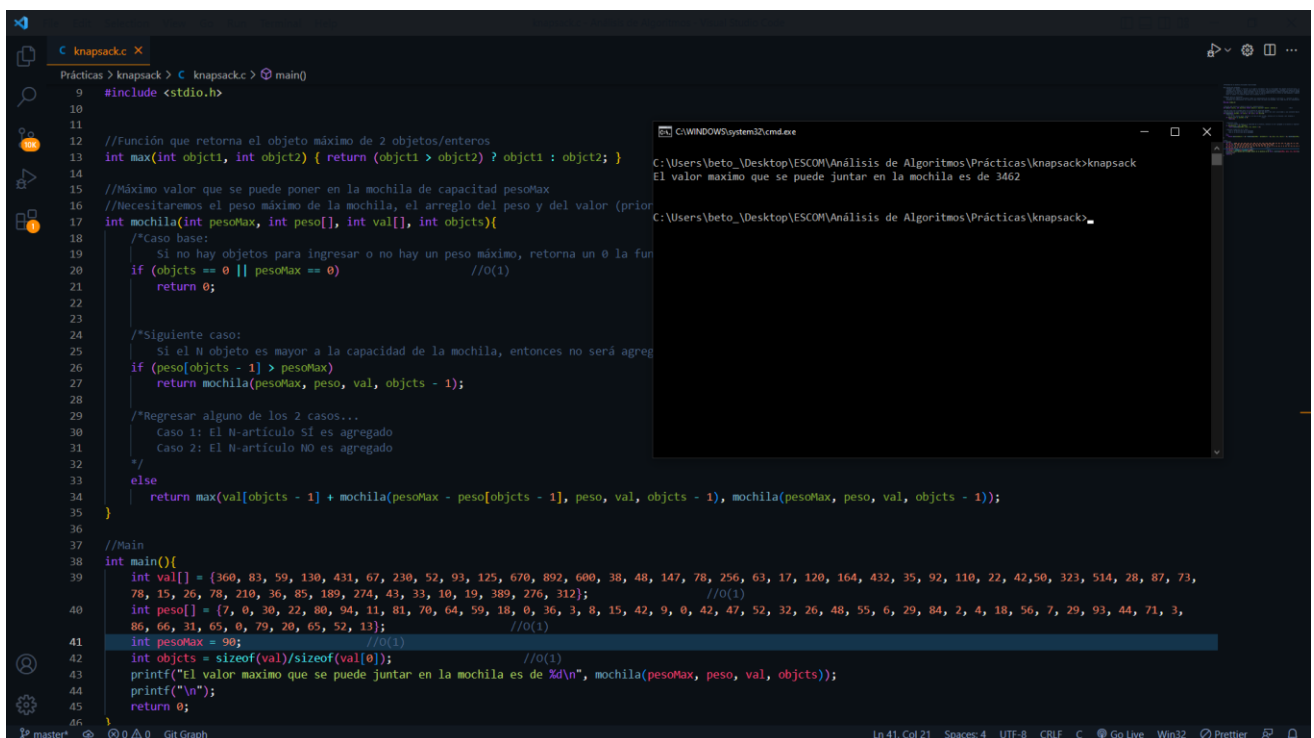
1er prueba:



```
1  #include <stdio.h>
2
3  //Función que retorna el objeto máximo de 2 objetos/enteros
4  int max(int objct1, int objct2) { return (objct1 > objct2) ? objct1 : objct2; }
5
6  //Máximo valor que se puede poner en la mochila de capacidad pesoMax
7  //Necesitaremos el peso máximo de la mochila, el arreglo del peso y del valor (prior
8  int mochila(int pesoMax, int peso[], int val[], int objs){
9      /*Caso base:
10         Si no hay objetos para ingresar o no hay un peso máximo, retorna un 0 la fun
11         if (objs == 0 || pesoMax == 0) //O(1)
12             return 0;
13
14         /*Siguiente caso:
15         Si el N objeto es mayor a la capacidad de la mochila, entonces no será agreg
16         if (peso[objcs - 1] > pesoMax)
17             return mochila(pesoMax, peso, val, objcs - 1);
18
19         /*Regresar alguno de los 2 casos...
20         Caso 1: El N-artículo SI es agregado
21         Caso 2: El N-artículo NO es agregado
22         */
23         else
24             return max(val[objcs - 1] + mochila(pesoMax - peso[objcs - 1], peso, val, objcs - 1), mochila(pesoMax, peso, val, objcs - 1));
25     }
26
27 //Main
28 int main(){
29     int val[] = {360, 83, 59, 130, 431, 67, 238, 52, 93, 125, 670, 892, 600, 38, 48, 147, 78, 256, 63, 17, 120, 164, 432, 35, 92, 110, 22, 42, 50, 323, 514, 28, 87, 73,
30     78, 15, 26, 78, 210, 36, 85, 189, 274, 43, 33, 10, 19, 389, 276, 312}; //O(1)
31     int peso[] = {7, 0, 30, 22, 80, 94, 11, 81, 70, 64, 59, 18, 0, 36, 3, 8, 15, 42, 9, 0, 42, 47, 52, 32, 26, 48, 55, 6, 29, 84, 2, 4, 18, 56, 7, 29, 93, 44, 71, 3,
32     86, 66, 31, 65, 0, 79, 20, 65, 52, 13}; //O(1)
33     int pesoMax = 10; //O(1)
34     int objs = sizeof(val)/sizeof(val[0]); //O(1)
35     printf("El valor maximo que se puede juntar en la mochila es de %d\n", mochila(pesoMax, peso, val, objs));
36     printf("\n");
37     return 0;
38 }
```

C:\WINDOWS\system32\cmd.exe
C:\Users\beto\Desktop\ESCOM\Análisis de Algoritmos\Prácticas\knapsack>knapsack
El valor maximo que se puede juntar en la mochila es de 1607
C:\Users\beto\Desktop\ESCOM\Análisis de Algoritmos\Prácticas\knapsack>

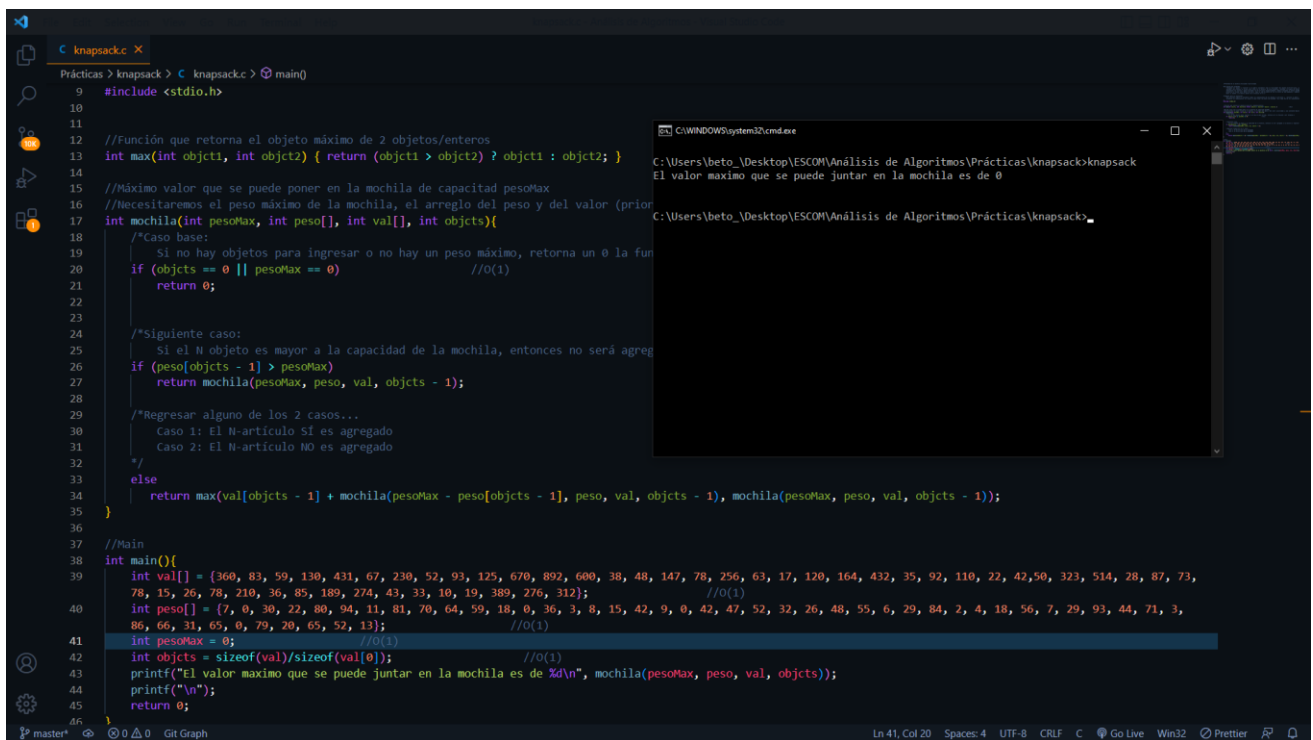
Como primer prueba tenemos que el peso Máximo de la mochila será de 10, (da igual si son kg, lb, m³, gr, etc.) y entre los 50 objetos que se encuentran en el arreglo inicial, se debe de llenar una mochila de peso 10 y retornar el valor de dicha mochila.



```
1  #include <stdio.h>
2
3  //Función que retorna el objeto máximo de 2 objetos/enteros
4  int max(int objct1, int objct2) { return (objct1 > objct2) ? objct1 : objct2; }
5
6  //Máximo valor que se puede poner en la mochila de capacidad pesoMax
7  //Necesitaremos el peso máximo de la mochila, el arreglo del peso y del valor (prior
8  int mochila(int pesoMax, int peso[], int val[], int objs){
9      /*Caso base:
10         Si no hay objetos para ingresar o no hay un peso máximo, retorna un 0 la fun
11         if (objs == 0 || pesoMax == 0) //O(1)
12             return 0;
13
14         /*Siguiente caso:
15         Si el N objeto es mayor a la capacidad de la mochila, entonces no será agreg
16         if (peso[objcs - 1] > pesoMax)
17             return mochila(pesoMax, peso, val, objcs - 1);
18
19         /*Regresar alguno de los 2 casos...
20         Caso 1: El N-artículo SI es agregado
21         Caso 2: El N-artículo NO es agregado
22         */
23         else
24             return max(val[objcs - 1] + mochila(pesoMax - peso[objcs - 1], peso, val, objcs - 1), mochila(pesoMax, peso, val, objcs - 1));
25     }
26
27 //Main
28 int main(){
29     int val[] = {360, 83, 59, 130, 431, 67, 238, 52, 93, 125, 670, 892, 600, 38, 48, 147, 78, 256, 63, 17, 120, 164, 432, 35, 92, 110, 22, 42, 50, 323, 514, 28, 87, 73,
30     78, 15, 26, 78, 210, 36, 85, 189, 274, 43, 33, 10, 19, 389, 276, 312}; //O(1)
31     int peso[] = {7, 0, 30, 22, 80, 94, 11, 81, 70, 64, 59, 18, 0, 36, 3, 8, 15, 42, 9, 0, 42, 47, 52, 32, 26, 48, 55, 6, 29, 84, 2, 4, 18, 56, 7, 29, 93, 44, 71, 3,
32     86, 66, 31, 65, 0, 79, 20, 65, 52, 13}; //O(1)
33     int pesoMax = 90; //O(1)
34     int objs = sizeof(val)/sizeof(val[0]); //O(1)
35     printf("El valor maximo que se puede juntar en la mochila es de %d\n", mochila(pesoMax, peso, val, objs));
36     printf("\n");
37     return 0;
38 }
```

C:\WINDOWS\system32\cmd.exe
C:\Users\beto\Desktop\ESCOM\Análisis de Algoritmos\Prácticas\knapsack>knapsack
El valor maximo que se puede juntar en la mochila es de 3462
C:\Users\beto\Desktop\ESCOM\Análisis de Algoritmos\Prácticas\knapsack>

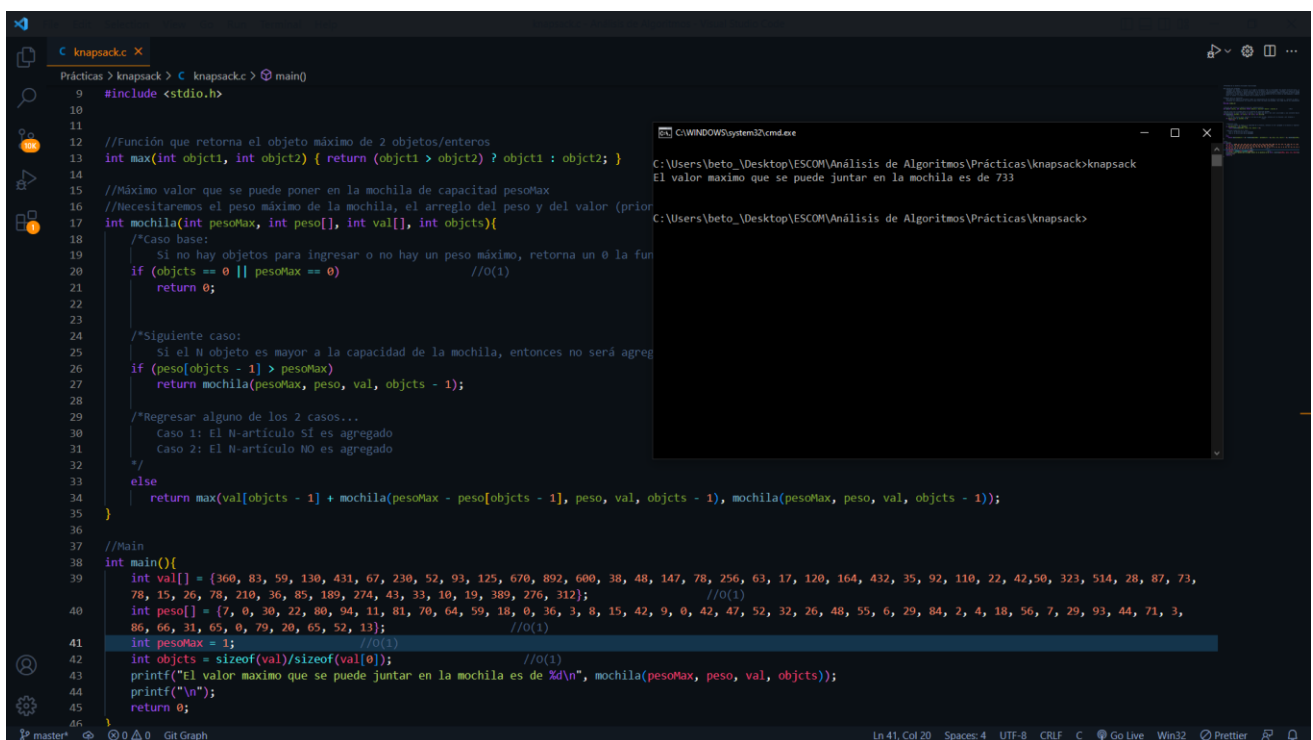
Tenemos un peso de 90 y el valor máximo que podemos obtener al llenar la mochila es de 3462.



```
Prácticas > knapsack > C knapsack.c > main()
9 #include <stdio.h>
10
11
12 //Función que retorna el objeto máximo de 2 objetos/enteros
13 int max(int objct1, int objct2) { return (objct1 > objct2) ? objct1 : objct2; }
14
15 //Máximo valor que se puede poner en la mochila de capacidad pesoMax
16 //Necesitaremos el peso máximo de la mochila, el arreglo del peso y del valor (prior
17 int mochila(int pesoMax, int peso[], int val[], int objects){
18     /*Caso base:
19     Si no hay objetos para ingresar o no hay un peso máximo, retorna un 0 la fun
20     if (objects == 0 || pesoMax == 0) //O(1)
21         return 0;
22
23     /*Siguiente caso:
24     Si el N objeto es mayor a la capacidad de la mochila, entonces no será agreg
25     if (peso[objects - 1] > pesoMax)
26         return mochila(pesoMax, peso, val, objects - 1);
27
28     /*Regresar alguno de los 2 casos...
29     Caso 1: El N-artículo SI es agregado
30     Caso 2: El N-artículo NO es agregado
31     */
32     else
33         return max(val[objects - 1] + mochila(pesoMax - peso[objects - 1], peso, val, objects - 1), mochila(pesoMax, peso, val, objects - 1));
34 }
35
36
37 //Main
38 int main(){
39     int val[] = {360, 83, 59, 130, 431, 67, 230, 52, 93, 125, 670, 892, 600, 38, 48, 147, 78, 256, 63, 17, 120, 164, 432, 35, 92, 110, 22, 42, 50, 323, 514, 28, 87, 73,
40     78, 15, 26, 78, 210, 36, 85, 189, 274, 43, 33, 10, 19, 389, 276, 312}; //O(1)
41     int peso[] = {7, 0, 30, 22, 80, 94, 11, 81, 70, 64, 59, 18, 0, 36, 3, 8, 15, 42, 9, 0, 42, 47, 52, 32, 26, 48, 55, 6, 29, 84, 2, 4, 18, 56, 7, 29, 93, 44, 71, 3,
42     86, 66, 31, 65, 0, 79, 20, 65, 52, 13}; //O(1)
43     int pesoMax = 0; //O(1)
44     int objects = sizeof(val)/sizeof(val[0]); //O(1)
45     printf("El valor maximo que se puede juntar en la mochila es de %d\n", mochila(pesoMax, peso, val, objects));
46     printf("\n");
47     return 0;
48 }
```

C:\WINDOWS\system32\cmd.exe
C:\Users\beto\Desktop\ESCOM\Análisis de Algoritmos\Prácticas\knapsack>knapsack
El valor maximo que se puede juntar en la mochila es de 0
C:\Users\beto\Desktop\ESCOM\Análisis de Algoritmos\Prácticas\knapsack>

En dado caso de que tengamos un peso de “0” en la entrada, en automático el programa al condicional del “Caso Base”, ya que no hay un peso máximo por agregar.



```
Prácticas > knapsack > C knapsack.c > main()
9 #include <stdio.h>
10
11
12 //Función que retorna el objeto máximo de 2 objetos/enteros
13 int max(int objct1, int objct2) { return (objct1 > objct2) ? objct1 : objct2; }
14
15 //Máximo valor que se puede poner en la mochila de capacidad pesoMax
16 //Necesitaremos el peso máximo de la mochila, el arreglo del peso y del valor (prior
17 int mochila(int pesoMax, int peso[], int val[], int objects){
18     /*Caso base:
19     Si no hay objetos para ingresar o no hay un peso máximo, retorna un 0 la fun
20     if (objects == 0 || pesoMax == 0) //O(1)
21         return 0;
22
23     /*Siguiente caso:
24     Si el N objeto es mayor a la capacidad de la mochila, entonces no será agreg
25     if (peso[objects - 1] > pesoMax)
26         return mochila(pesoMax, peso, val, objects - 1);
27
28     /*Regresar alguno de los 2 casos...
29     Caso 1: El N-artículo SI es agregado
30     Caso 2: El N-artículo NO es agregado
31     */
32     else
33         return max(val[objects - 1] + mochila(pesoMax - peso[objects - 1], peso, val, objects - 1), mochila(pesoMax, peso, val, objects - 1));
34 }
35
36
37 //Main
38 int main(){
39     int val[] = {360, 83, 59, 130, 431, 67, 230, 52, 93, 125, 670, 892, 600, 38, 48, 147, 78, 256, 63, 17, 120, 164, 432, 35, 92, 110, 22, 42, 50, 323, 514, 28, 87, 73,
40     78, 15, 26, 78, 210, 36, 85, 189, 274, 43, 33, 10, 19, 389, 276, 312}; //O(1)
41     int peso[] = {7, 0, 30, 22, 80, 94, 11, 81, 70, 64, 59, 18, 0, 36, 3, 8, 15, 42, 9, 0, 42, 47, 52, 32, 26, 48, 55, 6, 29, 84, 2, 4, 18, 56, 7, 29, 93, 44, 71, 3,
42     86, 66, 31, 65, 0, 79, 20, 65, 52, 13}; //O(1)
43     int pesoMax = 1; //O(1)
44     int objects = sizeof(val)/sizeof(val[0]); //O(1)
45     printf("El valor maximo que se puede juntar en la mochila es de %d\n", mochila(pesoMax, peso, val, objects));
46     printf("\n");
47     return 0;
48 }
```

C:\WINDOWS\system32\cmd.exe
C:\Users\beto\Desktop\ESCOM\Análisis de Algoritmos\Prácticas\knapsack>knapsack
El valor maximo que se puede juntar en la mochila es de 733
C:\Users\beto\Desktop\ESCOM\Análisis de Algoritmos\Prácticas\knapsack>

Bueno, y qué pasa si mi peso máximo de la mochila = 1 y no tengo un objeto de peso 1 en mi arreglo inicial. Lo que hará el código es sumar el valor de todos los objetos de peso

0 y retornarlos; La razón por la que no se agrega otro objeto de un peso mayor es porque no lo permite nuestro segundo condicional, por lo tanto, la mochila regresará su valor antes de que se agregue algún otro objeto en caso de que no haya un objeto de peso equivalente o igual al peso máximo de la mochila.

6. Conclusiones

El problema de la mochila se considera como un problema NP ya que toma en cuenta muchos factores y uno de ellos es la complejidad, que para terminar, la complejidad de este problema es de n^2 . Puede parecer un problema fácil, pero una vez que consideras que otros factores como el valor de los objetos, comienza a tornarse bastante complejo.

7. Bibliografías

- ✓ (Dasgupta, July 18, 2006, p. 318)
- ✓ https://developers.google.com/optimization/bin/knapsack#c++_3
- ✓ <https://es.wikipedia.org/wiki/NP-completo>
- ✓ <https://www.cs.us.es/~marper/docencia/TCC-2019-2020/temas/tema-5-trans.pdf>