

Mochila



19 de enero de 2023

Analisis y Diseño de Algoritmos

*Herrera Guadarrama Juan
Pablo*

0.NP Mochila

Escogí el problema de **mochila 0-1** que maximice V donde los objetos no se pueden repetir y no pueden fraccionarse, para completar un peso que no supere el máximo P . Se resuelve con **programación dinámica**, lo que nos ocupará este tema.

1.Análisis y diseño

El problema se establece sobre n objetos para llenar una mochila con peso máximo P . La primera parte del algoritmo tiene por objeto rellenar la tabla de resultados intermedios.

Para ello el array vacío cuya longitud es $P+1$.

La iteramos por el bucle doble $i=1..n \times j=1..P$. Entonces el coste de esta parte es $\text{Max}(P+1, n \times P) = n \times P$. En resumen, el coste de esta parte es $O(nP)$.

La segunda parte ejecuta el bucle voraz $i=n..1$, por lo que el coste final es el mismo $O(nP)$.

Una consideración con este algoritmo es que sólo sirve para pesos y peso máximo enteros.

2.Código fuente

```
#include <stdio.h>

// Una función de utilidad que devuelve
// máximo de dos enteros
int max(int a, int b)
{
    return (a > b) ? a : b;
}

// Devuelve el valor máximo que se puede
// poner en una mochila de capacidad W
int knapSack(int W, int wt[], int val[], int n)
{
    // Base Case
    if (n == 0 || W == 0)
        return 0;

    // Si el peso del enésimo artículo es mayor que
    // Capacidad de la mochila W, entonces este artículo no puede
    // estar incluido en la solución óptima
    if (wt[n - 1] > W)
    {
        printf("\n%d \n", wt[n - 1]);
        return knapSack(W, wt, val, n - 1);
    }

    // Devuelve el máximo de dos casos:
    // (1) enésimo artículo incluido
    // (2) no incluidos
    else
    {
        printf("\n w=%d peso =%d \n", W, wt[n - 1]);
```

```

        return max(val[n - 1] + knapSack(W - wt[n - 1], wt, val, n - 1), knapSack(W,
wt, val, n - 1));
    }
}


// Driver code
int main()
{
    int val[] = {10, 40, 30, 20 }; // costo$$
    int wt[] = { 4, 3, 5, 2 }; // peso
    int W = 8; // pesototal
    int n = sizeof(val) / sizeof(val[0]);
    printf("%d", knapSack(W, wt, val, n));
    return 0;
}

```

Evidencias

Para la evidencia este será los costos y precios y el valor máximo que soporta la mochila

4 kg 10 €	3 kg 40 €	5 kg 30 €	2 kg 20 €
--------------	--------------	--------------	--------------



8 kg

La resolución de dicho problema se ve en la siguiente tabla

i \ j	p	v	j → (P)								
			0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0	0	0
1	4	10	0	0	0	0	10	10	10	10	10
2	3	40	0	0	0	40	40	40	40	50	50
3	5	30	0	0	0	40	40	40	40	50	70
4	2	20	0	0	20	40	40	60	60	60	70

Corriendo el programa, al igual que en la tabla el coste mayor es de **70**

```

C:\Users\pablo\Documents\ESCOM\9_2022-2\Análisis de algoritmos\Prácticas\mochila.exe
w=8 peso =2
w=8 peso =5
w=8 peso =3
w=8 peso =4
w=5 peso =4
w=3 peso =3
4
w=6 peso =5
w=6 peso =3
w=6 peso =4
4
3
4
Valor maximo 70
-----
Process exited after 0.05649 seconds with return value 0
Presione una tecla para continuar

```

Referencias

Xavier Ochoa. (2015, 19 junio). *Programación Dinámica - Mochila 0-1* [Vídeo]. YouTube.

<https://www.youtube.com/watch?v=fVrPwSkSo0I>

Paz, A. de la. (2020, 15 abril). *El problema de la mochila discreta 0-1*. Copyright (c) 2020.

<https://www.wextensible.com/temas/programacion-dinamica/mochila.html>