Algoritmo botes de pintura

José María Ramírez González
Ana García Muñoz
Manuel Sánchez Pérez
Víctor Gutiérrez Rubiño

Nomenclatura

- n: nº de tipos diferentes de botes de pintura.
- V: litros de pintura del pedido a satisfacer.
- l(i): capacidad (litros) del bote de pintura tipo i.
- u(i): unidades disponibles del bote de pintura tipo i.
- c(i): coste del bote de pintura tipo i.
- k: variable auxiliar que hemos usado para hallar el nº de recipientes que usamos.
- m: tabla en la que se guardan los costes.
- r: tabla en la que se guardan el nº de recipientes que usamos.

El problema

Este problema consistía en: dados nº tipos de botes de pintura, cada uno su capacidad, coste y número de unidades, satisfacer un pedido de V litros, realizando una selección de los botes de pintura de forma en que se tenga el menor coste posible satisfaciendo dicho pedido.

	capacidad l_i	número u_i	coste c_i
e_1	3	3	5
e_2	5	3	9
e_3	7	2	12

Modelización

Para solucionar este problema, hemos usado programación dinámica. Así, hemos planteado este problema como una sucesión de decisiones, dividiendo dicho problema en subproblemas.

La sucesión de decisiones ha consistido en decidir cuál es el mínimo coste, satisfaciendo el pedido, cuando tengo botes de tipo 1, botes de tipo 2, etc, hasta tener botes de tipo n. Dicho esto, tenemos que m(i,j) es el mínimo coste que se obtiene al utilizar botes de tipo 1, tipo 2, ..., tipo i para satisfacer un problema de j litros. El problema original es m(n,V).

Por último, para la representación de la solución hemos usado un vector x, en el que i se refiere al tipo de recipiente y x[i] al número de recipientes de tipo i que he usado para satisfacer el pedido.

Principio de optimalidad

Vamos a demostrarlo mediante reducción al absurdo:

Sea m[i,j] el mínimo coste usando i tipos de envases diferentes para satisfacer un pedido de j litros y x1, x2, x3, ..., xn el número de envases que utilizamos de tipo 1, tipo 2, etc. Si la secuencia x1, .. , xn es óptima, entonces x1, ..., xn-1 también lo será para m[n-1, V-xn*l(n)].

Si esto no fuera así, entonces existiría y1, .., yn-1 tal que $\sum_{i=1}^{n-1} xi * c(i) > \sum_{i=1}^{n-1} yi * c(i)$. Por ello, se tendría que cumplir que $\sum_{i=1}^{n} yi * l(i) >= V - (xn * l(n))$.

Si hacemos yn = xn, tendríamos que $\sum_{i=1}^{n} yi * l(i) >= V,$ luego podríamos decir que y1, ..., yn es una solución óptima. Sin embargo, como $\sum_{i=1}^{n} xi * c(i) > \sum_{i=1}^{n} yi * c(i)$, entonces x1, ..., xn ya no sería la solución óptima, llegando a una contradicción.

Recurrencia

En programación dinámica, se debe realizar una definición recursiva de la solución optimal. Nosotros hemos planteado la siguiente, teniendo en cuenta todos los casos posibles:

$$
i = 1 \quad \begin{cases} m[i,j] = m[i,j-1] + c_i \;\; ; \;\; r[i,j] = r[i,j-1] + 1 & \text{si} \quad j > \left(\left(m[i,j-1] \,/\, c_i \right) \cdot \ell_i \right) \\ m[i,j] = m[i,j-1] \;\; ; \;\; r[i,j] = r[i,j-1] & \text{si} \quad j \leq \left(\left(m[i,j-1] \,/\, c_i \right) \cdot \ell_i \right) \end{cases}
$$

Caso base

$m[i,j] = \emptyset \quad \text{si} \quad j = \emptyset$

$$
i > 1 \quad \begin{cases} m[i,j] = \min\left(m[i-1,j], c_i \right) \;\; ; \;\; r[i,j] = \begin{cases} 1 & \text{si} \;\; m[i,j] = c_i \\ \emptyset & \text{si} \;\; m[i,j] \neq c_i \end{cases} & \text{si} \quad j < \ell_i \\ m[i,j] = \min\left\{ \emptyset \leq K \leq u_i \;\; \& \;\; j \geq K \cdot \ell_i \right\} \left(m[i-1, j-(\ell_i \cdot K)] + c_i \cdot K \right) \;\; ; \;\; r[i,j] = K & \text{si} \quad \ell_i \leq j \leq \ell_i \cdot u_i \\ m[i,j] = \infty \;\; ; \;\; r[i,j] = \emptyset & \text{si} \quad j > \ell_i \cdot u_i \end{cases}
$$

Cálculo y determinación de la solución óptima

Hemos utilizado dos tablas, ambas con enfoque ascendente, para almacenar la información que vamos obteniendo conforme ejecutamos el algoritmo. Estas tablas comparten la misma dimensión: tenemos V (litros del pedido) columnas y n (tipos de botes) filas. Rellenaremos estas tablas según la recurrencia definida anteriormente.

Una vez tenemos las tablas rellenas, usando la tabla r (tabla en al que hemos guardado los recipientes usados) y el vector x (donde se almacena la solución), seguiremos el siguiente procedimiento:

Mientras que $j>0$ e $i>0$:

- si $r[i,j]=0 \rightarrow i = i-1$
- si $r[i,j]>0 \rightarrow$
  - $x[i,j] = r[i,j]$
  - $j = j-(K[i,j]\cdot e_i)$
  - $i = i-1$

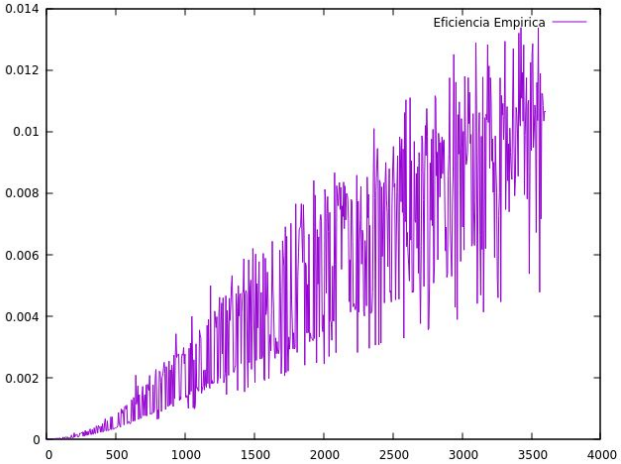
Pseudocódigo y eficiencia teórica

```
for each TipoBotePintura:                                                                                    N
    for each Numero ≤ LitrosPedidos:                                                                         L
        if col = 0:
            m[TipoBotePintura][Numero] = 0
            r[TipoBotePintura][Numero] = 0
        else if TipoBotePintura = Primero:
            if Numero > TipoBotePintura.capacidad * TipoBotePintura.NumeroDisponible:
                m[TipoBotePintura][Numero] = Infinito
                r[TipoBotePintura][Numero] = 0
            else if Numero > m[TipoBotePintura][Numero-1]/TipoBotePintura.coste * TipoBotePintura.capacidad:
                m[TipoBotePintura][Numero] = m[TipoBotePintura][Numero-1] + TipoBotePintura.coste
                r[TipoBotePintura][Numero] = r[TipoBotePintura][Numero-1] + 1
            else
                m[TipoBotePintura][Numero] = m[TipoBotePintura][Numero-1]
                r[TipoBotePintura][Numero] = r[TipoBotePintura][Numero-1]
        else
            if Numero < TipoBotePintura.capacidad
                if m[TipoBotePintura-1][Numero] > TipoBotePintura.coste
                    m[TipoBotePintura][Numero] = TipoBotePintura.coste
                    r[TipoBotePintura][Numero] = 1
                else
                    m[TipoBotePintura][Numero] = m[TipoBotePintura-1][Numero]
                    r[TipoBotePintura][Numero] = 0
            else
                for k ≤ TipoBotePintura.NumeroDisponible && k * TipoBotePintura.capacidad ≤ Numero                K
                    if m[TipoBotePintura-1][Numero - k*TipoBotePintura.capacidad] + TipoBotePintura.capacidad*k < m[TipoBotePintura][Numero]:
                        m[TipoBotePintura][Numero] = m[TipoBotePintura-1][Numero - k*TipoBotePintura.capacidad] + TipoBotePintura.capacidad*k
                        r[TipoBotePintura][Numero] = k
```

Puesto que el resto de operaciones son O(1), nuestro algoritmo resulta en

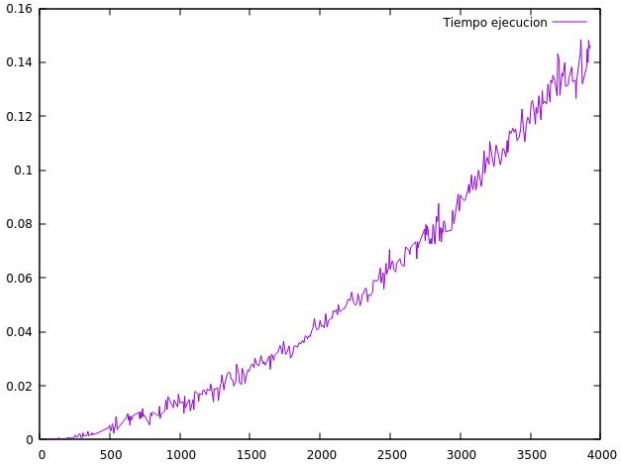
O(N·L·K)

Eficiencia empírica



Variando solo el número de litros pedidos

Aumento del tiempo de ejecución, pero menor difusión de los tiempos, se aprecia mejor la función que la define



Variando también el número de botes y el número máximo de los mismos

Ejemplo resuelto

Queremos satisfacer un pedido de 10 litros con los siguientes botes de pintura:

	capacidad l_i	número u_i	coste c_i
e_1	3	3	5
e_2	5	3	9
e_3	7	2	12

Tras aplicar el algoritmo que hemos definido, obtenemos las siguientes tablas:

Usando la tabla de recipientes y siguiendo el procedimiento definido nos quedamos con la siguiente solución:

x [1, 0 , 1]

Esto significa que hemos utilizado un bote de tipo 1 y un bote de tipo 3, teniendo esta elección un coste de 17.

Tabla costes

m	0	1	2	3	4	5	6	7	8	9	10
1	0	5	5	5	10	10	10	15	15	15	∞
2	0	5	5	5	9	9	10	14	14	15	18
3	0	5	5	5	9	9	10	12	14	15	17

Tabla recipientes

	0	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	2	2	2	3	3	3	0
2	0	0	0	0	1	1	0	1	1	0	2
3	0	0	0	0	0	0	0	1	0	0	1