Introducción

Los integrantes del grupo y su porcentaje de participación son los siguientes:

- Ana García Muñoz - 40%
- José María Ramírez González - 40%
- Víctor Gutiérrez Rubiño - 10%
- Manuel Sánchez Pérez - 10%

El problema a realizar fue el problema de los botes de pintura. 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.

Resolución del problema

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).

Hemos usado la siguiente representación del problema:

- 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.

Para guardar la información hallada por el algoritmo, hemos utilizado dos tablas de datos:

- m: tabla en la que se guardan los costes.
- r: tabla en la que se guardan el nº de recipientes que usamos.

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

Antes de proceder con la recurrencia, vamos a demostrar que se cumple el Principio de Optimalidad de Belman para el planteamiento que hemos escogido. Para ello, reduciremos 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:

Caso base: si j=0 -> m[i,j] = 0
-> r[i,j] = 0.

Para i = 1:

- si j > ((m[i,j-1] / c(i)) * l(i))
  -> m[i,j] = m[i, j-1] + c(i)
  -> r[i,j] = r[i, j-1] + 1

- si j <= ((m[i,j-1] / c(i)) * l(i))
  -> m[i,j] = m[i, j-1]
  -> r[i,j] = r[i, j-1]

Para i > 1:

- si j < l(i)
  -> m[i,j] = min(m[i-1,j], c(i))
  -> si m[i,j] = c(i) -> r[i,j] = 1
  -> si m[i,j] != c(i) -> r[i,j] = 0

- si l(i) <= j <= l(i) * u(i)
  -> m[i,j] = min{0 <= k <= u(i) && j >= k * l(i)} (m[i-1, j-(l(i)*k)] + c(i)*k)
  -> r[i,j] = k

- si j > l(i) * u(i)
  -> m[i,j] = ∞

-> r[i,j] = 0

Cálculo del valor óptimo

Como he mencionado anteriormente, 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.

Determinación de la solución óptima

Una vez tenemos las tablas rellenas, debemos buscar la solución óptima. Para ello, 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 i > 0 y j > 0:

- si r[i,j] = 0 -> i = i - 1

- si r[i,j] > 0 -> x[i,j] = r[i,j]
  j = j - (r[i,j] * l(i))
  i = i - 1

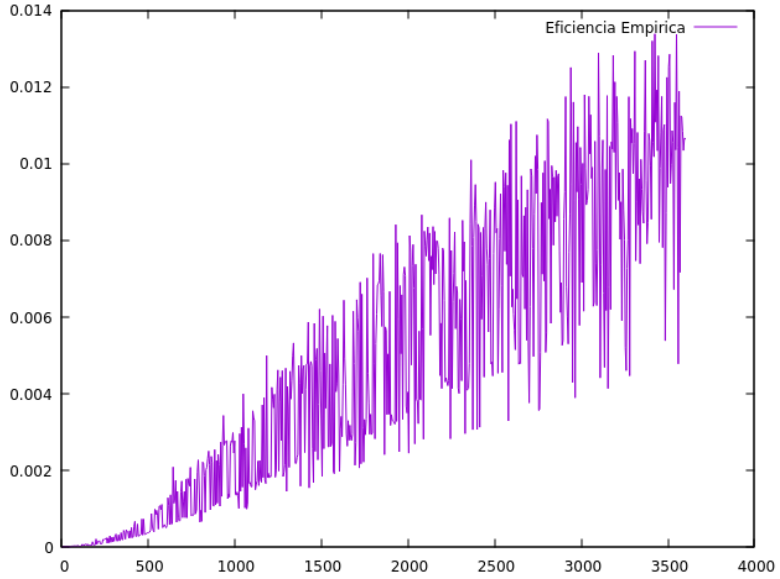
Eficiencia Teórica

La eficiencia teórica se calcula teniendo en cuenta el total de iteraciones que realizamos por los distintos bucles.
Si nos fijamos en el pseudocódigo de abajo, podemos ver que iteramos por todos los botes de pintura distintos que tenemos (lo llamaremos l), por todos los números de litros hasta llegar al que nos piden (lo llamaremos n) y también por el máximo número de botes de pintura existentes en un tipo de bote de pintura concreto, es decir, como tenemos limitada la cantidad máxima de botes que podemos usar para cada bote concreto, pues el máximo valor posible será el que tengamos en cuenta para nuestra eficiencia teórica (llamada k).

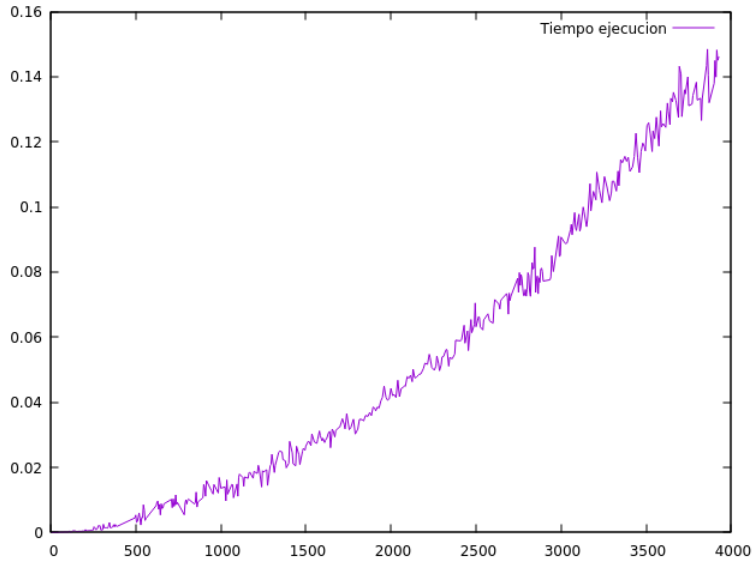
Así pues, la eficiencia teórica del algoritmo es del orden $O(n \cdot l \cdot k)$.

Eficiencia Empírica

Como vemos, al aumentar el número de litros y no aumentar ni el número de botes ni el número de botes de cada tipo, seguimos una función que simplemente se multiplica por un valor, aunque tiene mucha dispersión entre el mejor y peor caso, se puede ver el patrón que sigue.



Si alteramos el programa para poder también modificar el número de botes y la cantidad de recipientes de cada tipo, obtenemos una gráfica bastante más definida, a la vez que también aumenta el tiempo de ejecución debido al aumento del número de botes y cantidad máxima de los mismos. En esta gráfica se puede apreciar con bastante más claridad la función que sigue nuestro algoritmo.



Pseudocódigo

```
for each TipoBotePintura:
    for each Numero ⩽ LitrosPedidos:
        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
                    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
```

Como vemos, iteramos por los distintos botes de pintura y por los litros que nos piden, así hasta llegar al final de todos los valores, comprobando las distintas condiciones expresadas en la modelización y rellenando la tabla con los valores óptimos de cada recipiente y el coste asociado esos valores óptimos.

Esto se realiza creando un struct Botes (visible en el archivo de código adjunto) y 2 matrices dinámicas (r y m) para el número de recipientes y el coste óptimo para cada número de litros y tipos distintos de recipientes.

Ejemplo

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:

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	∅
2	0	0	0	0	1	1	0	1	1	0	2
3	0	0	0	0	0	0	0	1	0	0	1

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.