TÉCNICA DE DISEÑO

GREEDY

Características de la técnica

- Se aplica a **problemas de optimización**, donde

- se tienen **n entradas**

- encontrar un subconjunto que satisfaga **ciertas restricciones**

- optimice alguna **función objetivo** (maximizar o minimizar)

Características de la técnica

- ***Solución factible*** es cualquier subconjunto que satisface las restricciones.

- ***Solución objetivo*** es una solución factible que **maximice o minimice la *función objetivo***. Una solución factible que logra este objetivo es llamada ***óptima***.

Características de la técnica

```
void GREEDY (A, n)
        { //A(1..n) contiene n entradas
        solución = ø;
        for (i=1; i <= n; i++)
                {
                x = SELECCIONAR (A);
                A = A-{x};
                if ( FACTIBLE (solución, x) )
                        solución = solución  U {x};
                }
        }
```

Características de la técnica

El costo del algoritmo depende:

- del número de iteraciones, que depende del tamaño de la entrada.

- del costo de las funciones SELECCIONAR y FACTIBLE.

FACTIBLE → tiempo constante (generalmente),

SELECCIONAR → explora el conjunto de candidatos y obtiene el mejor en ese momento. Mejora: preparar el conjunto de candidatos antes de entrar al bucle (por ejemplo, ordenar por algún criterio).

Características de la técnica

Propiedades de los problemas resolubles por greedy:

- **1°) elección greedy (greedy-choice):** una solución globalmente óptima puede ser alcanzada tomando alternativas localmente óptimas.

Características de la técnica

Propiedades de los problemas resolubles por greedy:

- **1°) elección greedy (greedy-choice):** una solución globalmente óptima puede ser alcanzada tomando alternativas localmente óptimas.
  - En un momento dado, realizamos la mejor alternativa y luego resolvemos el subproblema que queda.
  - Debemos probar que la elección greedy en cada paso conduce a la solución óptima global.

Características de la técnica

Propiedades de los problemas resolubles por greedy:

- **2°) subestructura óptima:** un problema exhibe una subestructura óptima si una solución óptima al problema contiene soluciones óptimas para los subproblemas.

Problema de la mochila

(con fracciones de objetos)

Problema de la mochila

- Se tienen **n objetos**, cada objeto i tiene → **peso** p_i
  → **beneficio** asociado b_i.
- y una mochila con **capacidad M,**
- se quiere colocar objetos enteros o fraccionados en la mochila,
- si colocamos en la mochila una **fracción x_i** de un objeto i

  → **ganancia = x_i b_i**

El **objetivo** es llenar la mochila de manera de maximizar la ganancia total.

Problema de la mochila

Definición formal del problema:

Restricción $\sum_{i=1}^{n} x_i \, pi \;\leq\; M$ (1)

Función objetivo $\sum_{i=1}^{n} x_i \, bi$ **a maximizar** (2)

con $0 \leq x_i \leq 1$ y $1 \leq i \leq n$ (3)

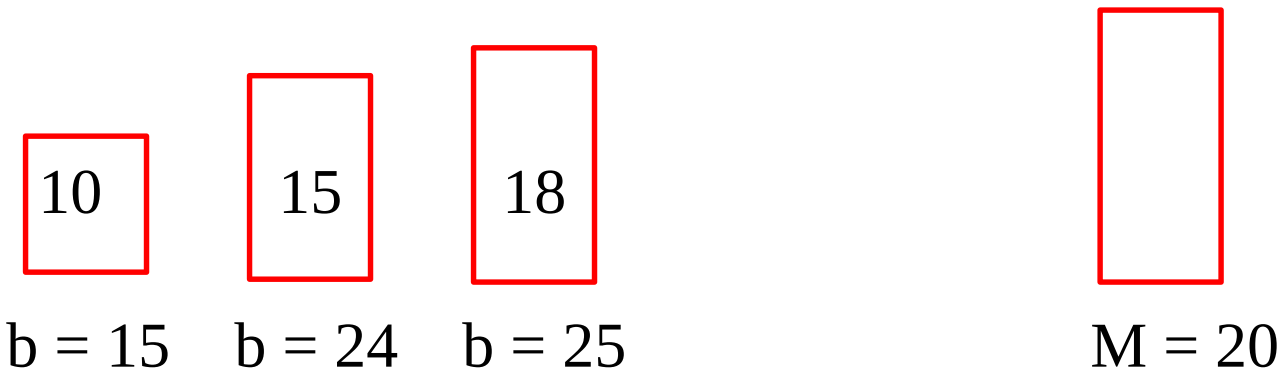
Solución factible → conjunto ($x_1, \ldots, x_n$) que satisface (1) y (3)

Solución óptima es una solución factible para la cual (2) es máximo.

Problema de la mochila

Ejemplo

Una instancia del problema de la mochila:



¿Con qué criterio seleccionamos los objetos que colocaremos en la mochila?

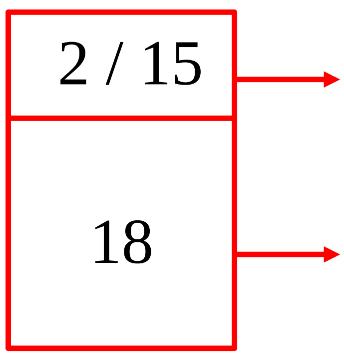
Problema de la mochila

Ejemplo: $M = 20$

$p_1=10$, $p_2=15$, $p_3=18$

$b_1=15$, $b_2=24$, $b_3=25$

1° criterio: Elegimos primero los objetos más valiosos



ganancia = 2 / 15 * 24 = 3,2

ganancia = 25

ganancia total = **28,2**

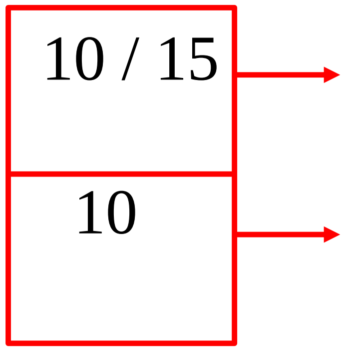
Problema de la mochila

Ejemplo: M = 20

$p_1=10,\quad p_2=15,\quad p_3=18$

$b_1=15,\quad b_2=24,\quad b_3=25$

2° criterio: Elegimos primero los objetos menos pesados para tratar de llenar la mochila lo más tarde posible.



ganancia = 10/15 * 24 = 16

ganancia = 15

ganancia total = **31**

Problema de la mochila

Ejemplo: M = 20

$p_1=10, \quad p_2=15, \quad p_3=18$

$b_1=15, \quad b_2=24, \quad b_3=25$

3° criterio: Elegimos primero aquel objeto con mayor ganancia por unidad de peso.

b_i/ p_i	1.5	1.6	1.3

5 / 10 → **ganancia = 5/10 * 15 = 7,5**

15 → **ganancia = 24**

ganancia total = **31,5** ⇒ **Solución óptima**

Problema de la mochila

```
void GREEDY_MOCHILA (float M , int n)
{         // V[n] y P [n] son arreglos de valores y pesos
          // ordenados según su V[i]/ P[i] en orden decreciente            O ( n log n)

          // X[n] es el arreglo solución, donde en x_i tendré la fracción
          // del objeto i que es colocado en la mochila, inicializado en 0.     O ( n )

    float resto = M;            // resto = capacidad de la mochila restante

    for ( i = 1, (i < n) && ( P[i] <= resto) ; i++ )                  O ( n )
        { X[i] = 1;
        resto = resto - P[i]; }

    if ( i < = n )
    X[i] = resto / P[i];
}
```

$T_{\text{Mochila (n)}} \in O(n \log n)$

Problema de la mochila

Teorema:
El algoritmo greedy para el problema de la mochila con selección por mayor relación *valor/peso* siempre encuentra una solución óptima.

Demostración:

Sea **X = (x1,x2,...,xn)** → solución encontrada por el algoritmo.
Sea **Y = (y1,y2,...,yn)** → otra solución

Sea **B(X) = Σ xi bi** → beneficio de la solución X
Sea **B(Y) = Σ yi bi** → beneficio de la solución Y

Se prueba que **B(X)** – **B(Y)** ≥ 0, luego X es una solución óptima.

Problema de la mochila

Teorema:
El algoritmo greedy para el problema de la mochila con selección por mayor relación *valor/peso* siempre encuentra una solución óptima.

Demostración:
Suponemos sin perder generalidad que los elementos ya están ordenados de esta forma, es decir, que $b_i / p_i \geq b_j / p_j$ si $i < j$.

Sea **X = (x1,x2,...,xn)** → solución encontrada por el algoritmo.

- Si **X = (1,1,...,1)** , la solución es óptima.
- En otro caso, sea j el menor índice tal que $x_j < 1$, es decir, **X = (1,1,..,0.x,...0,0)**

  $x_i = 1$ para todo $i < j$,

  $x_i = 0$ para todo $i > j$, y además $\Sigma\, x_i p_i = M$.

Sea **B(X) = Σ xi bi** → beneficio de la solución X.

Problema de la mochila

Consideremos **Y = (y1,y2,...,yn)** otra solución,

B(Y) = Σ yi bi → beneficio de la solución Y
y además cumple que Σ yi pi ≤ M.

Entonces, restando ambas capacidades, podemos afirmar que

$$\Sigma(xi\ pi - yi\ pi) = \Sigma\ pi\ (xi - yi) \geq 0$$

Calculemos entonces la diferencia de beneficios:

$$B(X) - B(Y) = \Sigma(xi - yi)\ bi = \Sigma(xi - yi)\ \underbrace{pi\ (bi/pi)}_{\text{multiplico y divido por pi.}}.$$

Problema de la mochila

Con esto, para el índice j escogido anteriormente sabemos que:

- Si $i < j$ entonces $x_i = 1$ $\left\{ \begin{array}{l} (x_i - y_i) \geq 0 \\ (b_i/p_i) \geq (b_j/p_j) \text{ por la ordenación decreciente} \end{array} \right.$

- Si $i > j$ entonces $x_i = 0$ $\left\{ \begin{array}{l} (x_i - y_i) \leq 0. \\ (b_i/p_i) \leq (b_j/p_j) \text{ por la ordenación decreciente} \end{array} \right.$

- Si $i = j$ entonces $(b_i/p_i) = (b_j/p_j)$.

En consecuencia, podemos afirmar que:

$$(x_i - y_i)\,(b_i/p_i) \geq (x_i - y_i)\,(b_j/p_j) \qquad \text{para todo } i \text{, por lo tanto:}$$

$B(X) - B(Y) = \Sigma(x_i - y_i)\, p_i\, (b_i/p_i) \geq (b_j/p_j)\Sigma(x_i - y_i)p_i \geq 0$, esto es,

$B(X) \geq B(Y)$ → lo que queríamos demostrar

BIBLIOGRAFÍA

- Cormen, T.; Lieserson, C.; Rivest, R. **Introduction to Algorithms** Ed. The MIT Press. 2009.

- Horowitz, E.; Sahni, S.; Rajasekaran, S. **Computer Algorithms.** Computer Science Press.1998.

- Brassard, G.; Bratley, P. Prentice-Hall. **Fundamentos de Algoritmia**. 1997.