

# ALGORITMOS VORACES

Método algorítmico donde las soluciones a problemas de optimización se construyen de forma muy directa, dando pasos avariciosos que seleccionan cada vez la mejor alternativa posible con la esperanza de que una secuencia de decisiones óptimas localmente produzcan una solución óptima globalmente para todo el problema.



UNIVERSIDAD  
**COMPLUTENSE**  
MADRID

**ALBERTO VERDEJO**

# Problema del cambio de monedas

- ▶ Pagar de forma exacta cierta cantidad, utilizando el menor número de monedas.



- ▶ ¿Cómo pagarías 1,47 €?

Pensando en cómo devolver 1'47 con el menor número de monedas estamos haciendo un algoritmo voraz.



Cada paso hemos añadido la moneda de mayor valor que NO excedía la cantidad a pagar

# Problema del cambio de monedas

- ▶ El método voraz construye una solución a través de una secuencia de pasos, hasta que se alcanza una solución completa al problema.
- ▶ En cada paso, la elección debe ser: *Esta es la clave de los algoritmos voraces.*
  - ▶ *factible*, es decir, tiene que satisfacer las restricciones del problema
  - ▶ *óptima localmente*, es decir, la mejor opción local entre todas las disponibles en ese paso
  - ▶ *irrevocable*, es decir, no se puede cambiar en los pasos posteriores del algoritmo

# Problema del cambio de monedas, no siempre funciona

- ▶ La estrategia voraz no funciona para cualquier sistema monetario.

Funciona para nuestro sistema monetario.

Si solo tenemos estas monedas..



- ▶ Si queremos pagar 30, la solución voraz sería esta, con 6 monedas:



Pero queriendo pagar con el menor número de monedas, podemos usar solamente 3 monedas de 30.

# Problema del cambio de monedas, no siempre funciona

- ▶ La estrategia voraz no funciona para cualquier sistema monetario.



- ▶ Mientras que la solución óptima utiliza solamente 3 monedas:



Se podría resolver utilizando programación dinámica. Pero con un coste mayor que si se pudiera utilizar un algoritmo voraz, ya que pretendemos resolver un problema más general.

# Método voraz

Las características generales de los algoritmos voraces son:

- ▶ Para construir la solución se dispone de un *conjunto de candidatos*. Se van formando dos conjuntos: los candidatos seleccionados (formarán parte de la solución), y los rechazados definitivamente. Dos conjuntos CONCEPTUALES.
- ▶ Existe una *función de selección* que indica cuál es el candidato más prometedor de entre los aún no considerados. Esta es la ESTRATEGIA VORAZ.
- ▶ Existe un *test de factibilidad* que comprueba si un candidato es compatible con la solución parcial construida hasta el momento.

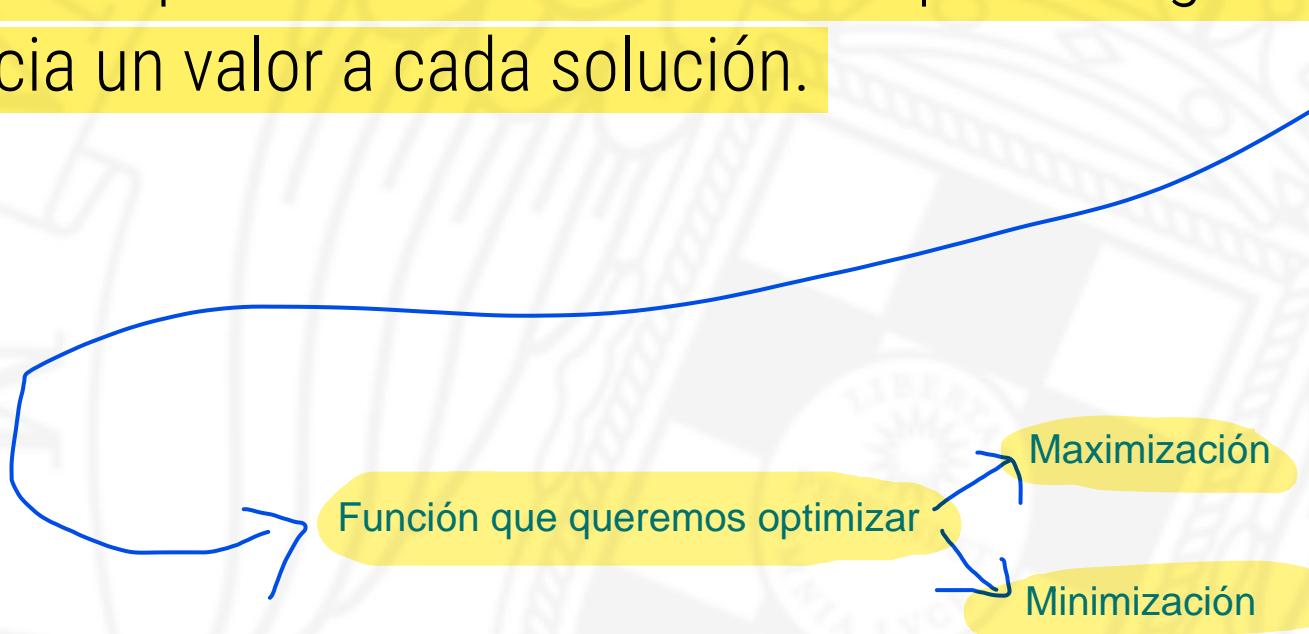
En qué orden se tienen que ir considerando los candidatos para encontrar la mejor solución.

# Método voraz

(continuación)

Las características generales de los algoritmos voraces son:

- ▶ Existe un *test de solución* que determina si una solución parcial forma una solución “completa”.
- ▶ Se tiene que obtener una solución óptima según una *función objetivo* que asocia un valor a cada solución.



# Método voraz

```
fun voraz(datos : conjunto) dev S : conjunto
var candidatos : conjunto
    S :=  $\emptyset$  { en S se va construyendo la solución }
    candidatos := datos
    mientras candidatos  $\neq \emptyset$   $\wedge$   $\neg$ es-solución?(S) hacer
        x := seleccionar(candidatos)
        candidatos := candidatos - {x}
        si es-factible?(S  $\cup$  {x}) entonces S := S  $\cup$  {x} fsi
    fmientras
ffun
```

Considera el primer candidato

Forma parte de los seleccionados.

Cada candidato se toma una sola vez.

# Demostración de corrección

Los algoritmos voraces crean soluciones directas y eficientes. Sin embargo necesitamos demostrar que la estrategia voraz es correcta, es decir, que siempre encuentra una solución óptima.

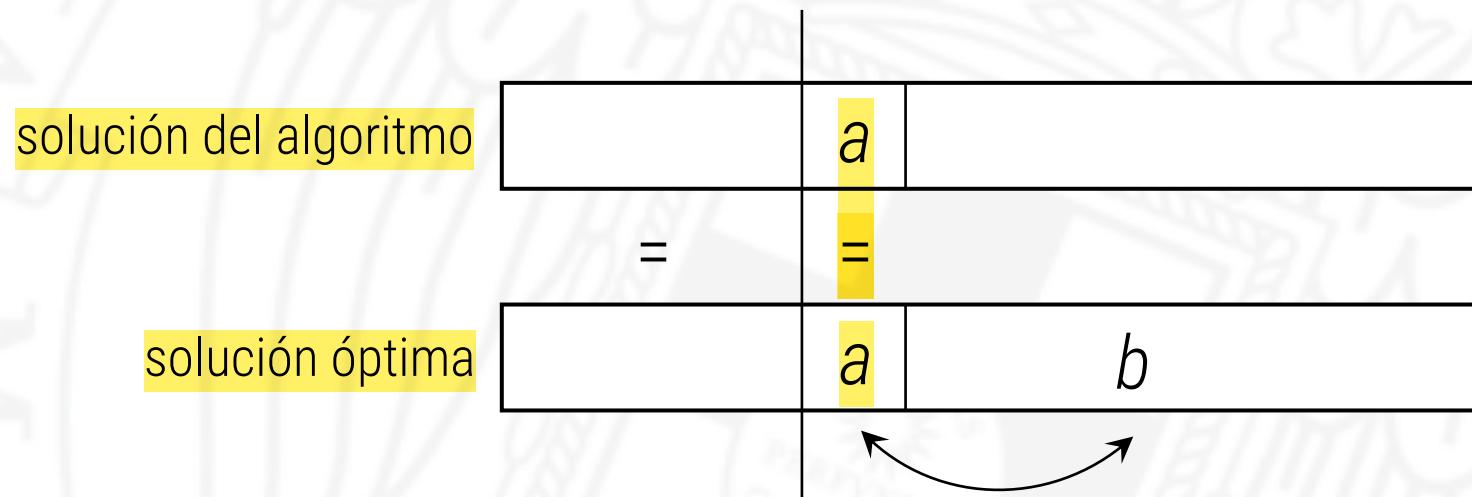
- Método de *reducción de diferencias*: comparar una solución óptima con la solución obtenida por el algoritmo voraz. Si ambas soluciones no son iguales, se va transformando la solución optima de partida de forma que continúe siendo óptima, pero siendo más parecida a la del algoritmo voraz.

		Nuestra solución
solución del algoritmo		a
	=	≠
solución óptima	b	a

De forma que ahora sea más parecida a la del algoritmo voraz

# Demostración de corrección

- Método de *reducción de diferencias*: comparar una solución óptima con la solución obtenida por el algoritmo voraz. Si ambas soluciones no son iguales, se va transformando la solución optima de partida de forma que continúe siendo óptima, pero siendo más parecida a la del algoritmo voraz.



■ De forma que ahora sea más parecida a la del algoritmo voraz

# Maximizar número de ficheros en disco

Vamos a realizar un problema.

- ▶ Tenemos  $n$  ficheros  $F_1, F_2, \dots, F_n$  que queremos almacenar en un disco.
- ▶ El fichero  $F_i$  requiere  $s_i$  megabytes de espacio de disco y la capacidad del disco es  $D$  megabytes.
- ▶ Queremos maximizar el número de ficheros almacenados en el disco.
- ▶ Supondremos que  $D < \sum_{i=1}^n s_i$ . Capacidad del disco es insuficiente para contener a todos los discos.
- ▶ Consideraremos los programas de menor a mayor tamaño.

Cogeremos primero los de menor tamaño en megabytes y los meteremos, para maximizar el número de ficheros tenemos que coger los que menos tamaño ocupan.

Siempre debe devolver el fichero más pequeño aun no considerado.

Si llegamos a un fichero y no podemos meterlo en el disco lo descartamos y podemos terminar.

# Maximizar número de ficheros en disco

```
//  $S[0] \leq S[1] \leq \dots \leq S[n - 1] \wedge D < \sum_{i=0}^{n-1} S[i]$ 
vector<bool> ficheros_en_disco(vector<int> const& S, int D) {
    vector<bool> solucion(S.size(), false); Ficheros seleccionados para meter en el disco.
    int acumula = 0;
    for (int i = 0; acumula + S[i] <= D; ++i) {
        solucion[i] = true;
        acumula += S[i];
    }
    return solucion;
}
```

Primero ordenaríamos los candidatos según algún criterio.

En cada iteración preguntamos si cabe.

Se selecciona y acumula su tamaño.

Lineal respecto al número de ficheros

$O(n)$

# Maximizar número de ficheros en disco, corrección

solución del algoritmo



solución óptima



Hay que demostrar que la solución así construida es óptima. Que no hay otra mejor (que consiga almacenar más ficheros).

# Maximizar número de ficheros en disco, corrección

$F_1 < F_2 \dots$

USAMOS EL MÉTODO DE REDUCCIÓN DE DIFERENCIAS.

solución del algoritmo

$F_1$	$F_2$	...	$F_{i-1}$	$F_i$	...
-------	-------	-----	-----------	-------	-----

=

$\neq$

solución óptima

$F_1$	$F_2$	...	$F_{i-1}$	$F_{\bar{F}_j}$	...
-------	-------	-----	-----------	-----------------	-----

Ordenar los ficheros en la solución óptima por tamaño de menor a mayor

►  $F_i$  no está en la solución óptima

Nos paramos en la primera posición donde las soluciones difieren.

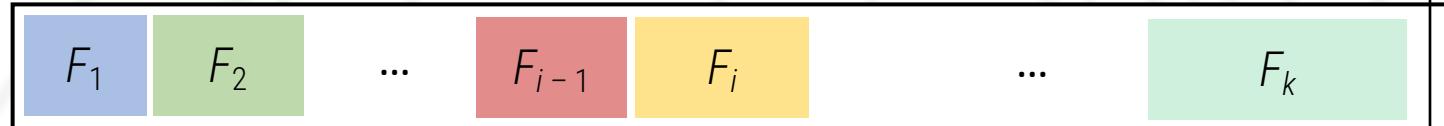
►  $j > i \Rightarrow s_i \leq s_j$

Jota es mayor que i.

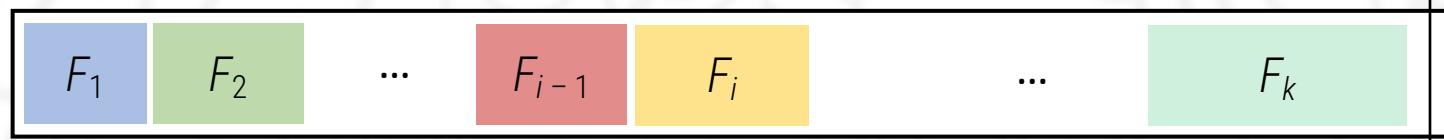
Sustituimos  $F_j$  por  $F_i$

# Maximizar número de ficheros en disco, corrección

solución del algoritmo



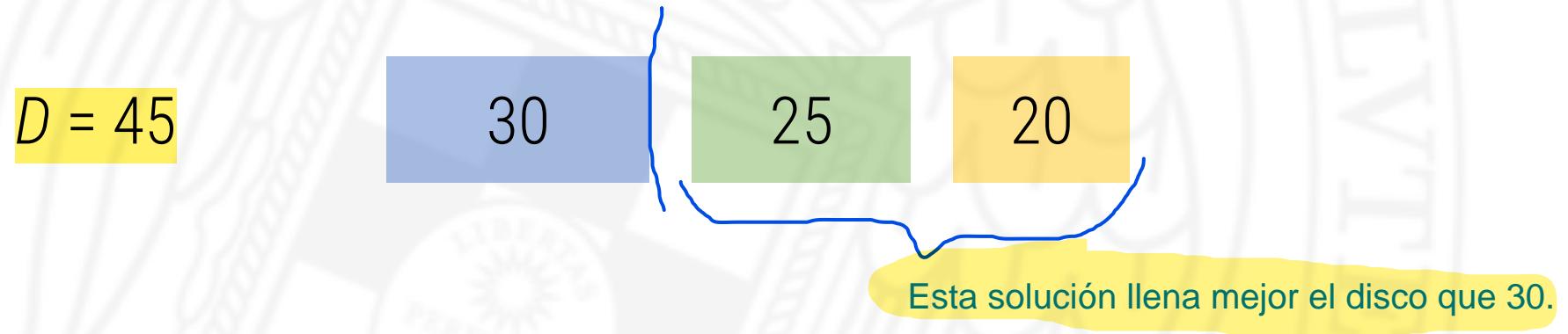
solución óptima



- ▶  $F_i$  no está en la solución óptima
- ▶  $j > i \implies s_i \leq s_j$
- ▶ Podemos seguir haciendo transformaciones con los  $k$  ficheros de la solución del algoritmo

# Maximizar ocupación del disco

- ▶ No existe estrategia voraz para este problema.
- ▶ Buscar contraejemplos que demuestren que una estrategia no funciona en todos los casos. *Para cualquier tipo de algoritmo que planteáramos para este problema.*
- ▶ Contraejemplo para la estrategia que considera los programas de mayor a menor tamaño:



# Maximizar ocupación del disco

- ▶ No existe estrategia voraz para este problema.
- ▶ Buscar contraejemplos que demuestren que una estrategia no funciona en todos los casos.
- ▶ Contraejemplo para la estrategia que considera los programas de menor a mayor tamaño:

$$D = 30$$

10

15

20