EXPLORACIÓN EXHAUSTIVA Y VUELTA ATRÁS

EXPLORACIÓN EXHAUSTIVA

EXPLORACIÓN EXHAUSTIVA O BÚSQUEDA DE FUERZA BRUTA

- La búsqueda exhaustiva o por fuerza bruta consiste en enumerar (crear un árbol) todos los posibles candidatos para la solución de un problema y comprobar si el resultado esta entre todos los candidatos encontrados.

- La búsqueda por fuerza bruta es sencilla de implementar y, siempre que exista, encuentra una solución. Sin embargo, su coste de ejecución es proporcional al número de soluciones candidatas, el cual es exponencialmente proporcional al tamaño del problema.

- Por el contrario, la búsqueda por fuerza bruta se usa habitualmente cuando el número de soluciones candidatas no es elevado, o bien cuando este puede reducirse previamente usando algún otro método heurístico.

- Por ejemplo, un algoritmo de fuerza bruta para encontrar el divisor de un número natural ***n*** consistiría en enumerar todos los enteros desde **1** hasta ***n***, chequeando si cada uno de ellos divide ***n*** sin generar resto.

Divisores de 10: 1, 2, 5, 10.
Divisores de 20: 1, 2, 4, 5, 10, 20.
Divisores de 30: 1, 2, 5, 6, 10, 15, 30.
Divisores de 40: 1, 2, 4, 5, 8, 10, 20, 40.

- El método general consiste en:
  - Generar una lista de todas las soluciones potenciales del problema en una forma sistemática.
  - Evaluar las soluciones potenciales una a una, descalificando las no factibles y manteniendo un registro de la mejor encontrada hasta el momento (en problemas de optimización).
  - Cuando la búsqueda finalice, regresar todas las soluciones indicando cuál es la mejor solución encontrada.

- Este método puede ser fácilmente modificado de forma que:
  - Termine una vez encuentre la primera solución
  - Después de encontrar un determinado número de soluciones
  - Después de probar con un número específico de soluciones candidatas
  - Después de haber consumido una cantidad fija de tiempo de CPU.

Explosión combinacional

- La principal desventaja del método de fuerza bruta es que, para la mayoría de problemas reales, el número de soluciones candidatas es demasiado elevado.

- Por ejemplo, para buscar los divisores de un número ***n*** tal y como se describe anteriormente, el número de soluciones candidatas a probar será de ***n***.

- Por tanto, si ***n*** consta de 16 dígitos, la búsqueda requerirá de al menos 10^{15} comparaciones computacionales, tarea que puede tardar varios días en un ordenador personal.

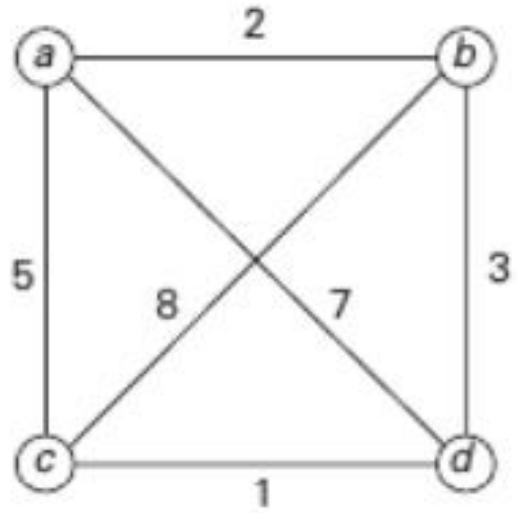
- Si ***n*** es un bit de 64 dígitos, que aproximadamente puede tener hasta 19 dígitos decimales, la búsqueda puede tardar del orden de 10 años.

- Este crecimiento exponencial en el número de candidatos, cuando crece el tamaño del problema ocurre en todo tipo de problemas. A este fenómeno no deseado se le denomina explosión combinacional.

PROBLEMA DEL AGENTE VIAJERO (TSP)

- Dada una lista de **n** ciudades y las distancias entre cada par de ellas, encontrar el recorrido (ruta) más corto posible que visita cada ciudad exactamente una vez y regresa a la ciudad origen.

- Es equivalente a resolver el problema de encontrar el ciclo hamiltoniano más corto en un grafo ponderado conexo.



Tour	Length	
$a \rightarrow b \rightarrow c \rightarrow d \rightarrow a$	$l = 2 + 8 + 1 + 7 = 18$	
$a \rightarrow b \rightarrow d \rightarrow c \rightarrow a$	$l = 2 + 3 + 1 + 5 = 11$	optimal
$a \rightarrow c \rightarrow b \rightarrow d \rightarrow a$	$l = 5 + 8 + 3 + 7 = 23$	
$a \rightarrow c \rightarrow d \rightarrow b \rightarrow a$	$l = 5 + 1 + 3 + 2 = 11$	optimal
$a \rightarrow d \rightarrow b \rightarrow c \rightarrow a$	$l = 7 + 3 + 8 + 5 = 23$	
$a \rightarrow d \rightarrow c \rightarrow b \rightarrow a$	$l = 7 + 1 + 8 + 2 = 18$	

- Para **n** ciudades, por lo tanto hay **n!** soluciones potenciales (número de permutaciones de n elementos).

- En varias de ellas sólo cambia el sentido del recorrido, por lo que existen $\frac{n!}{2}$ posibles soluciones distintas.

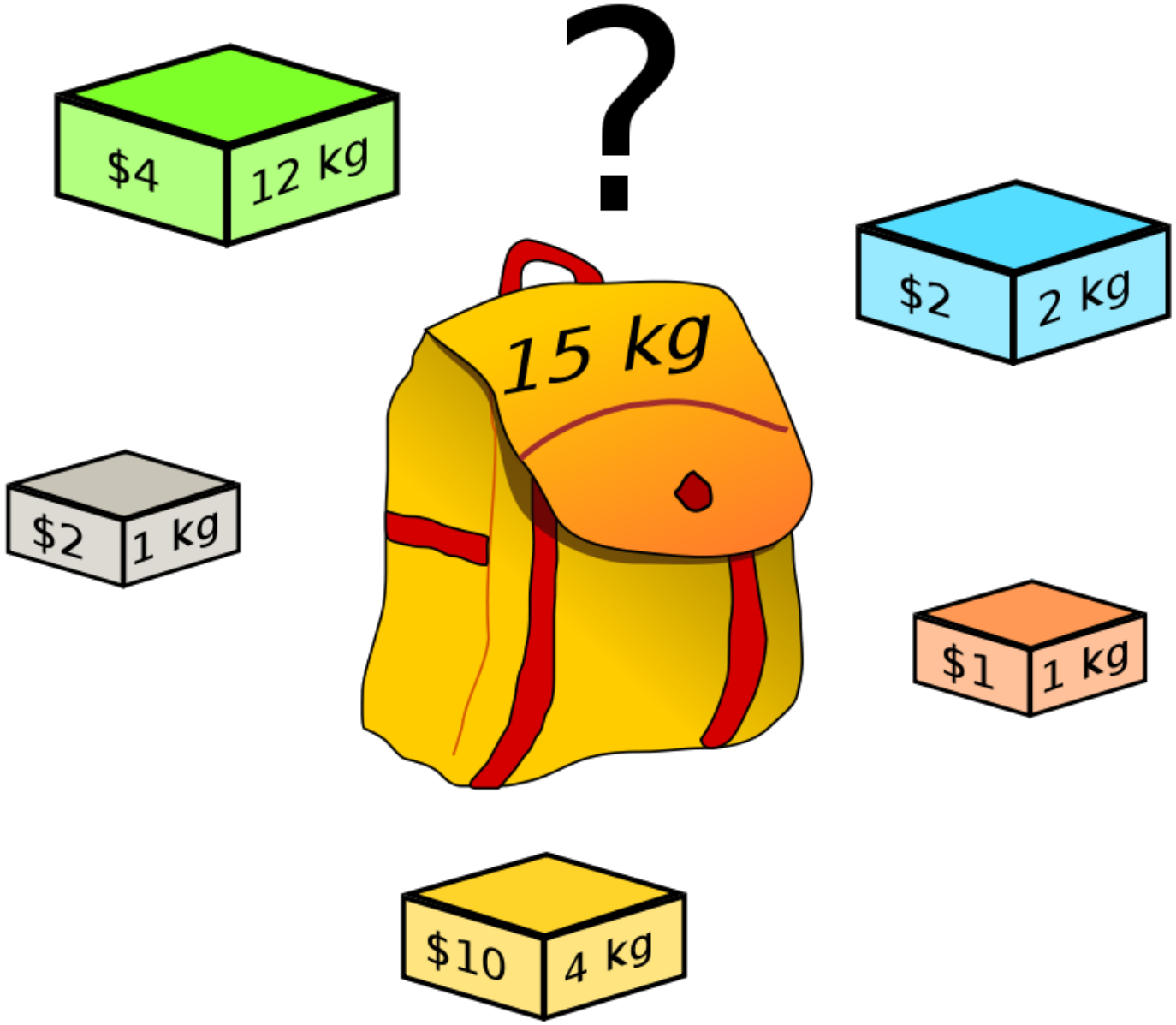
- El algoritmo de búsqueda exhaustiva pertenece entonces a la clase **O(n!)**

EL PROBLEMA DE LA MOCHILA

- Dados $\mathbf{n}$ distintos tipos de objetos, de los cuales se tienen $\mathbf{q_i}$ disponibles para cada tipo:

$$(\mathbf{1 \leq q_i \leq \infty}).$$

- Cada tipo de objeto $\mathbf{i}$ tiene un beneficio asociado $\mathbf{v_i}$ y un peso (o volumen) $\mathbf{w_i}$. $(\mathbf{v_i}, \mathbf{w_i} > \mathbf{0})$.
- Por otro lado se tiene una mochila, donde se pueden introducir los objetos, que soporta un peso máximo (o volumen máximo) $\mathbf{W}$.
- El problema consiste en meter en la mochila objetos de tal forma que se maximice el valor de los objetos que contiene y siempre que no se supere el peso máximo que puede soportar la misma.
- La solución al problema vendrá dada por la secuencia de variables $\mathbf{x_1, x_2, \ldots, xn}$ donde el valor de $\mathbf{x_i}$ indica cuantas copias se meterán en la mochila del objeto de tipo $\mathbf{i}$.


?
$4
12 kg
15 kg
$2
2 kg
$2
1 kg
$1
1 kg
$10
4 kg

- El problema se pude resolver con búsqueda exhaustiva generando todos los subconjuntos del conjunto de $\mathbf{n}$ objetos ($\mathbf{2^n}$), y evaluando cada uno de ellos para encontrar el mejor.

- Por lo tanto la búsqueda exhaustiva nos conduce a una algoritmo con complejidad $\boldsymbol{\Omega(2^n)}$

- Los algoritmos de búsqueda exhaustiva corren en tiempo razonable solamente para instancias muy pequeñas.

- En muchos casos, la búsqueda exhaustiva (o alguna de sus variantes) es la única forma conocida de obtener una solución exacta.