Límites de la Computación

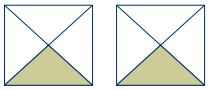
- ¿Hay problemas que un ordenador no puede resolver?
- ¿Existen límites a la capacidad de proceso de los ordenadores?
- ¿Pueden usarse esos límites para nuestro beneficio?

Límites de la Computación

- La siguiente cita está recogida de un número especial de la revista TIME del año 84:
  - *"Pongamos el tipo de software adecuado en un computador y hará lo que queramos. Puede haber límites en lo que podemos hacer con las máquinas, pero no lo hay en lo que podemos hacer con el software"*
- Más grave es la afirmación debida a Newell, (1958):
  - *"En un plazo de 15 años los computadores podrán hacer todas las tareas que hacen los humanos".*
- Estas proposiciones son absolutamente erróneas. De hecho el conjunto de problemas que un computador jamás podrá resolver (por rápido y grande que sea) es mucho más amplio que el conjunto de problemas "computables" (o decidibles).
- Así, en una primera aproximación, los problemas pueden clasificarse en:
  - Computables (o decidibles)
  - No computables (o no decidibles)

El problema de la teselación

- Una tesela (baldosa) es un cuadrado de 1x1, dividido en cuatro partes iguales por sus dos diagonales. Cada una de estas partes tiene un color. Para simplificar, supongamos que las baldosas tienen orientaciones fijas y no se pueden girar.



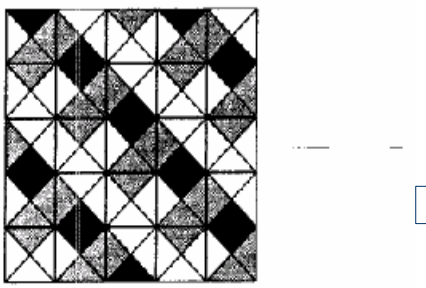
La entrada al problema es un número finito de descripciones de teselas (T). Cada tesela en T se define por sus cuatro colores en un cierto orden.

El problema es:

Dada cualquier superficie finita, de cualquier tamaño (con dimensiones enteras, por supuesto), ¿ puede recubrirse usando teselas de los tipos de T, de forma que cada dos teselas adyacentes tengan el mismo color en los lados que se tocan ?

Se supone que hay un número ilimitado de baldosas de cada tipo, pero el número total de tipos es finito.

Una solución a un problema de teselación

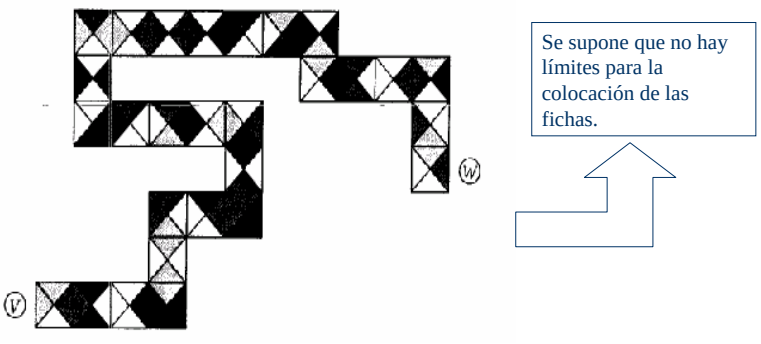


En este caso, que tiene solución, hay tres tipos de baldosas.

El problema de la teselación II

- Este es un ejemplo de un problema de decisión: la respuesta del "eventual" programa que lo resolviese debería ser "SI" o "NO".
- No obstante, es **imposible** diseñar un algoritmo (y, en consecuencia, un programa) que resuelva el problema de la teselación.
- El problema de la teselación es un caso particular de la clase de problemas de tipo *dominó*, y es el ejemplo más clásico de un problema "no decidible"
- Otro ejemplo clásico es el de la serpiente del domino:
  - *Dados dos puntos, V y W, del plano ¿es posible unirlos entre sí por teselas que cumplan la restricción de lados adyacente iguales, como las fichas del domino ?*
  - Si no se ponen límites a la porción del plano disponible para colocar teselas, el problema es decidible. Pero si el area de colocación se limita, el problema se hace no decidible.

Una serpiente del dominó



Un algoritmo indecidible

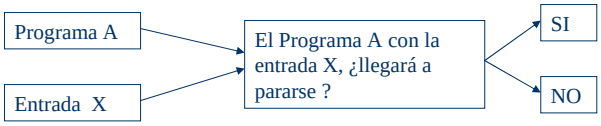
- Sea el siguiente algoritmo, debido a Lagarias (1985) :

```
Entrar X
Mientras X≠1 hacer:
        Si X es par, X = X/2
        Si no, X = 3X +1
Parar
```

- Por ejemplo, si el valor inicial de X es 7, va tomando los valores:
  - 7,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1
- Sin embargo, no se puede demostrar que este algoritmo llegue siempre a pararse, para cualquier número positivo de entrada. Aunque tampoco puede demostrarse lo contrario: que exista algún número para el cual el algoritmo no se para nunca.

El problema de la parada

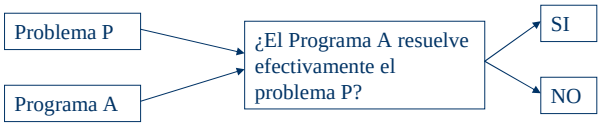
- El ejemplo anterior nos sirve para plantear otro problema no decidible:
  - *Dado un programa (o algoritmo) A y un valor de la entrada X, ¿podemos saber siempre si A se parará o no?*



  - El problema de la parada es ***no decidible*** : no hay manera de decir, en general y en un tiempo finito si la ejecución de un programa dado, con una entrada dada, terminará o no.

El problema de la verificación.

- Es más complicado que el de la parada:
  - *Dada la descripción formal de un problema, P, y un programa A que supuestamente lo resuelve, ¿es posible construir un programa que verifique si A resuelve, efectivamente, P ?*



- Este problema, mucho más general que el de la parada o el de la teselación, se dice que es un ***fuertemente no computable***.
- De forma que el mundo de los problemas se divide, al menos, en tres clases:
  - Computables (o decidibles)
  - No computables (teselación, parada, ...)
  - Fuertemente no computables (verificación)

Complejidad

- Una vez que sabemos que un problema es computable se plantea un segundo problema : ¿cuántos recursos de computación va a emplear el programa que implementa el algoritmo de solución ?
- Al hablar de recursos se consideran dos:
  - Tiempo : medido por el número de acciones elementales que se efectúan durante la ejecución del programa.
  - Espacio : medido por la cantidad de memoria necesaria.
- Así, se habla de complejidad temporal y complejidad espacial.
  - Obviamente la complejidad que casi siempre recibe mayor atención es la temporal.
- La complejidad dependerá del "tamaño" de la entrada al programa, que se denota de forma genérica por N.
  - N puede ser la dimensión de una matriz, el número de nodos en un grafo, el número de elementos de una lista, etc.

Tipos de complejidad

Complejidad de un cálculo

Se refiere a una MT y una entrada concretas

Son valores numéricos *ct, ce*

Complejidad de un algoritmo

Para una cierta MT compute un dato de entrada genérico

Son funciones *t(n)* y *s(n)* de la longitud del dato *n* en el *peor caso*

Complejidad un problema

Se definiría como la complejidad de su *algoritmo más sencillo* (¿?)

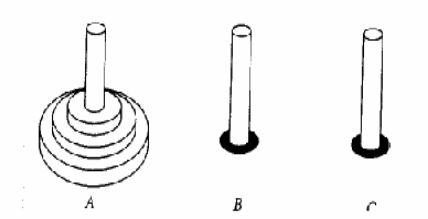
Normalmente se establece una cota superior, un valor medio o se delimita una clase de funciones (*tasa u orden de crecimiento*)

Complejidad II

- La complejidad puede depender de N de forma lineal (kN), polinómica (kN^m), logarítmica (logN), exponencial (k^N), etc
- Dado un algoritmo que resuelve un problema, automáticamente establece una *cota máxima* en su complejidad: cualquier otro algoritmo, para ser más eficiente, deberá ser "menos complejo".
  - Por ejemplo, si conocemos un algoritmo de clasificación de N elementos con complejidad temporal (kN^3), para que otro algoritmo sea mejor deberá mejorar esta complejidad.
- Sería interesante disponer de una cota mínima, es decir, alguna demostración formal de que problema dado no puede ser resuelto con menos de una cierta complejidad.
  - Por ejemplo, está demostrado que para los algoritmos de búsqueda sobre listas ordenadas, la cota mínima de complejidad es (logN) y no es posible encontrar ningún algoritmo nuevo que la mejore: es un **problema cerrado**.

Las torres de Hanoi

- Un problema de gran interés es el juego de las torres de Hanoi:



El juego consiste en mover los N anillos de la torre A a la torre B o C, usando la otra como ayuda, pero sin que haya un disco de mayor diámetro sobre otro de menor.

- La cota mínima de complejidad para este problema es (2^N).
  - Esto significa que un ordenador capaz de hacer 1 millón de operaciones por segundo, tardaría 1 msg en resolver el juego con 10 anillos y casi 36 años si le colocamos 50 anillos !

Torres de Hanoi. Caso práctico

Número de discos	Tiempo de CPU (sg)[1]	Tiempo de CPU (sg)[2]	Tiempo de CPU (sg)[3]	Tiempo de CPU (sg)[4]
14	0.96	0.15		
15	1.92	0.26		
16	3.86	0.52		
17	7.74	0.99	0.57	0.187
18	15.57	1.98	1.11	0.312
19	31.27	3.89	2.16	0.64
20	62.61	7.82	4.29	1.312
21	124.76	15.62	8.8	2.73

[1] HP 9000/D250 (tejo)

[2] SGI Origin 200 (lisisu02)

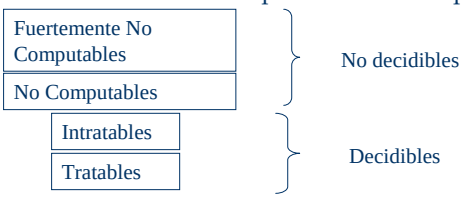
[3]Sun Enterprise 250 (encina)

[4] Pentium 4 a 2,8 MHz

•Tiempos calculados con un programa en Fortran 90

Problemas intratables

- Los problemas que admiten como solución algoritmos de complejidad polinómica (kN^m),se llaman **tratables**.
- Los problemas que solamente admiten como solución algoritmos cuya complejidad es super-polinómica (exponencial, por ejemplo) se llaman **intratables**.
  - Las torres de Hanoi es un ejemplo típico de un problema intratable.
- Notemos que los problemas intratables son computables.
- Por consiguiente, nuestra clasificación de los problemas ahora queda:

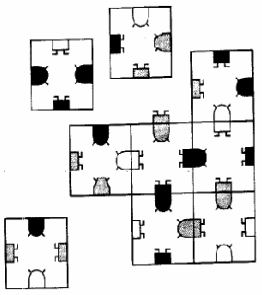


Problemas aún más intratables

- Las torres de Hanoi (2^N), el juego de ajedrez (3^N), son problemas intratables.
- Pero hay problemas todavía peores, cuya complejidad es del orden de 2exp(2(expN)).
  - Uno de los más estudiados es la aritmética de Presburger: se trata de establecer mecanismos de validación lógica sobre proposiciones planteadas sobre números enteros con la operación "+"
    - Si X=15 $\Rightarrow \neg\exists$ Y:X=Y+Y
    - N, en este caso, es el número de operadores (+,=, $\Rightarrow$,$\neg$,$\exists$,...)
- Existen problemas que son triplemente exponenciales.
- Si en la aritmética de Presburger incluimos la operación "*", el problema se convierte en no-computable.

Problemas NP completos

- Consideremos el rompecabezas del mono:



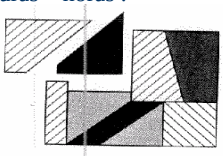
Se dispone de N (un cuadrado perfecto) piezas de un rompecabezas, cada una con figuras en los lados de formas y colores diferentes. Se trata de decidir si se pueden colocar las piezas en un cuadrado haciendo coincidir los colores y las figuras

Un algoritmo de fuerza bruta necesita probar (4^N)* N! Combinaciones. No se conoce ningún algoritmo mejor

Es un tipo de problemas de los que se conoce una cota superior, no tratable, pero no se ha demostrado que sean no-tratables : **NP-completos**

Más Problemas NP completos

- Problema del viajante (TSP):
  - Se dispone de un mapa de N ciudades, con las distancias entre ellas. ¿Es posible realizar una ruta que pase por todas las ciudades y cuya longitud total sea menor de una cierta cantidad ? ¿O sea mínima?
    - Este problema es inabordable para N mayor de 200.
    - Se puede generalizar fácilmente: como mover la cabeza de un robot de taladrar agujeros sobre un circuito impreso (cientos de agujeros, y miles de circuitos)
- Problema del horario:
  - Asignar profesores – aulas – asignaturas – horas .
  - Puzzles de “lineas aéreas” :
  - Tetris
  - Sudoku...



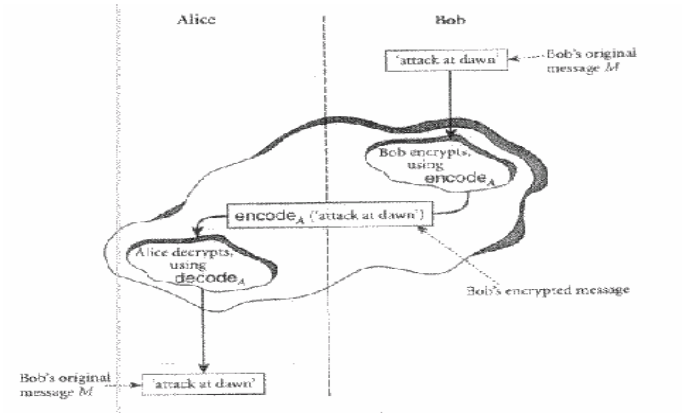
Posibles soluciones

- Está claro que los problemas no computables seguirán siéndolo aunque cambiemos el modelo de computación.
- No obstante, los problemas intratables pueden resolverse con modelos alternativos:
  - Sistemas paralelos.
  - Computación cuántica.
  - Computación molecular.
  - Computación estocástica.

Aprovechando lo malo

- La no tratabilidad de algunos problemas puede aprovecharse en algunos casos. El más popular es la criptografía, empleada para encriptar información.
- Encriptar un mensaje M (texto plano) consiste en convertirlo a otro mensaje M^* (texto cifrado) por un procedimiento que sea reversible.
  - M^* = encripta (M) y M = desencripta (M^*)
  - M = desencripta (encripta(M))
  - M^* = encripta (desencripta (M^*))
- El método más empleado es el llamado criptografía de clave pública, debido a Diffie y Hellman (1976)
  - La idea básica es que exista una clave para encriptar y otra para desencriptar.
  - La clave de encriptación de cada usuario es pública, pero la de desencriptación es privada.

Criptografía de clave pública



RSA

- La implementación de la criptografía de clave pública más extendida es la llamada RSA (Rivest, Shamir, Adleman, 1978).
- Se basa en la asimetría entre encontrar un número primo y encontrar los factores de un número dado.
  - Cada usuario (Alice) elige al azar dos números primos, P y Q, muy grandes (de unos 200 dígitos), y encuentra el producto P*Q (que tendrá unos 400 dígitos).
  - El usuario hace público el producto, pero se guarda los factores P y Q.
  - La función de encriptado se basa en el producto P*Q y por tanto, es pública. Pero la función de desencriptado necesita los factores P y Q, que son privados (y el problema de encontrarlos es no tratable)

Autentificación: Firma electrónica