# Recurrencias

# Método maestro

- Proporciona una manera de resolver recurrencias de la forma:

$$T(n) = aT(n/b) + f(n)$$

- donde a≥1 y b>1 son constantes y f(n) es una función asintóticamente positiva.

- Esta recurrencia describe el tiempo de un algoritmo que ...
  - divide un problema de tamaño n en a subproblemas, cada uno de tamaño n/b, donde a y b son constantes positivas.
  - los a subproblemas se resuelven recursivamente, cada uno en un tiempo T(n/b).
  - El costo de dividir el problema y combinar los resultados de los subproblemas está descrito por la función f(n).

# Método maestro

- El método maestro depende del teorema siguiente:
- Sean $a \geq 1$ y $b > 1$ constantes.
- Sea $f(n)$ una función.
- Sea $T(n)$ una función definida en los enteros positivos por la recurrencia:

$$T(n) = aT(n/b) + f(n)$$

- donde interpretamos que $n/b$ significa ya sea $\lfloor n/b \rfloor$ o $\lceil n/b \rceil$.
- Entonces $T(n)$ puede estar acotada asintóticamente de la forma siguiente:

# Teorema maestro: 3 casos

1. Si $f(n) = O(n^{\log_b a - \epsilon})$ para alguna constante $\epsilon > 0$, entonces

$$T(n) = \Theta(n^{\log_b a})$$

2. Si $f(n) = \Theta(n^{\log_b a})$, entonces

$$T(n) = \Theta(n^{\log_b a} \lg n)$$

3. Si $f(n) = \Omega(n^{\log_b a + \epsilon})$ para alguna constante $\epsilon > 0$, y si $af(n/b) \leq cf(n)$ para alguna constante $c < 1$ y una $n$ suficientemente grande, entonces

$$T(n) = \Theta(f(n))$$

# Teorema maestro

- En los tres casos estamos comparando la función $f(n)$ con la función $n^{\log_b a}$
- Intuitivamente, la solución a la recurrencia estará determinada por la mayor de estas funciones.
- En 1. la función $n^{\log_b a}$ crece más rápido polinomialmente que $f(n)$ por un factor $n^{\epsilon}$. La solución esta entonces dada por $n^{\log_b a}$.
- En 3. la función $f(n)$ crece polinomialmente más rápido que $n^{\log_b a}$ por un factor de $n^{\epsilon}$. Además $f(n)$ satisface la condición de regularidad que dice que $af(n/b) \leq cf(n)$ para alguna constante $c > 1$.
- Los tres casos no cubren todas las posibilidades, el método no puede aplicarse si las funciones no crecen polinomialmente mas rápido o si la condición de regularidad no se cumple.

# Estratégias generales de análisis y diseño de algorítmos

# Búsqueda exhaustiva (fuerza bruta)

- Probar todas las soluciones candidatas posibles hasta encontrar la solución al problema.
- Limite: **ineficiencia**
  - en general el número de candidatos a solución que se necesitan procesar crecen exponencialmente al tamaño del problema.
  - generación de candidatos a solución,
  - procesamiento y verificación de candidatos a solución.
- Ej. Magic Square Fill

# Recursión (tipo de reducción)

- Si la instancia dada del problema es pequeña o suficientemente simple, resolverla.
- Si no, reducir el problema a una o más instancias más simples del mismo problema.

- Ej. Torres de Hanoi

# Torres de Hanoi

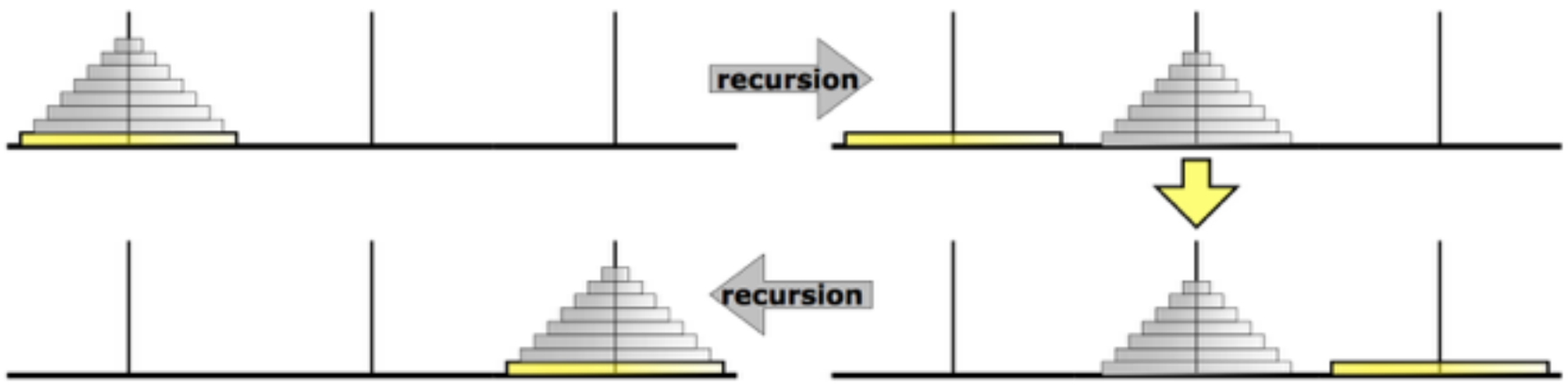
- Acertijo publicado por el matemático Francois Édouard Anatole Lucas en 1883 bajo el pseudónimo "N.Claus (de Siam)".
- El siguiente año, Henri de Parville describió el algoritmo con la siguiente historia:

**"En el gran templo de Benarés, debajo de la cúpula que marca el centro del mundo, yace una base de bronce, en donde se encuentran acomodadas tres agujas de diamante, cada una del grueso del cuerpo de una abeja y de una altura de 50 cm aproximadamente. En una de estas agujas, Dios, en el momento de la Creación, colocó sesenta y cuatro discos de oro, el mayor sobre la base de bronce y el resto de menor tamaño conforme se va ascendiendo. Día y noche, incesantemente, los sacerdotes del templo se turnan en el trabajo de mover los discos de una aguja a otra de acuerdo con las leyes impuestas e inmutables de Brahma, que requieren que siempre haya algún sacerdote trabajando, que no muevan más de un disco a la vez y que deben colocar cada disco en alguna de las agujas de modo que no cubra a un disco de radio menor. Cuando los sesenta y cuatro discos hayan sido transferidos de la aguja en la que Dios los colocó en el momento de la Creación a otra aguja, el templo y los brahmanes se convertirán en polvo y, junto con ellos, el mundo desaparecerá"**

http://web.engr.illinois.edu/~jeffe/teaching/algorithms/all-recursion.pdf

# Torres de Hanoi

- El truco para resolver este acertijo es pensar de forma recursiva en lugar de intentar resolver todo de una vez.
- Intentemos mover el disco más grande:
  - No podemos moverlo al inicio porque tiene discos más chicos sobre él.
  - Tenemos que mover los n-1 discos al tercer palo antes de mover el n-ésimo.
  - Después de mover los n discos, tenemos que regresar los n-1 discos sobre el n-ésimo disco.
- Redujimos el problema de una Torre de Hanoi de n discos a uno de (n-1) discos.




The Tower of Hanoi algorithm; ignore everything but the bottom disk

http://web.engr.illinois.edu/~jeffe/teaching/algorithms/all-recursion.pdf

# Torres de Hanoi

- Nuestra reducción recursiva hace una suposición sutil pero importante: Hay un disco más grande.
- En otras palabras, nuestro algoritmo funciona para cualquier n mayor a 1 pero ya no funciona cuando n=0. Tenemos que manejar el caso base directamente.
- Nuestra única tarea es reducir el problema a instancias más simples y resolver el problema directamente cuando una reducción ya no sea posible.
- Nuestro algoritmo es trivialmente correcto cuando n=0.
- Para instancias mayores, haremos recurrencia.

http://web.engr.illinois.edu/~jeffe/teaching/algorithms/all-recursion.pdf

# Torres de Hanoi

- El algoritmo mueve una pila de n discos de su palo fuente (src) a su palo de destino (dst) utilizando el tercer palo temporalmente (tmp).

```
HANOI(n, src, dst, tmp):
  if n > 0
      HANOI(n − 1, src, tmp, dst)
      move disk n from src to dst
      HANOI(n − 1, tmp, dst, src)
```

- Sea T(n) el número de movimientos necesarios para transferir n discos, el tiempo de cálculo de nuestro algoritmo.
- El caso base implica T(0) = 0.
- El caso recursivo más general implica que T(n) = 2T(n-1)+1 para cualquier n mayor a 1.
- La forma cerrada (por método de sustitución) es $2^n$-1.
- En particular, moviendo una torre de 64 discos, requiere $2^{64}$-1 movimientos: 18,446,744,073,551,615.
- Haciendo 1 movimiento por segundo, tomaría a los monjes 585 billones de años.

http://web.engr.illinois.edu/~jeffe/teaching/algorithms/all-recursion.pdf

# Backtracking

- Da un método conveniente para generar candidatos a solución y evitar generar candidatos innecesarios.
- Construir soluciones un componente cada vez y evaluar las soluciones parcialmente construidas como sigue:
  - si puede desarrollarse más sin violar las restricciones del problema, expandir tomando la primera opción legítima posible.
  - si no hay opción legítima para el siguiente componente no se necesita seguir expandiendo y regresamos al primer nodo posible.
  - en el peor caso puede ser como búsqueda exhaustiva pero casi nunca pasa esto.
- Se puede pensar como un árbol de decisiones.

http://web.engr.illinois.edu/~jeffe/teaching/algorithms/all-recursion.pdf

# Backtracking (vuelta atrás)

- Un algoritmo de backtracking intenta construir una solución a un problema computacional de manera incremental.

- Cuando el algoritmo necesita decidir entre dos opcionen para el siguiente componente de la solución, trata ambas recursivamente.

- El término backtrack fue acuñado por el matemático estadounidense D.H. Lehmer en los años 1950s.

http://web.engr.illinois.edu/~jeffe/teaching/algorithms/all-recursion.pdf

# Problema de las n-reinas

- Problema prototipo que se resuelve por backtracking.
- Propuesto por el entusiasta del ajedrez Max Bezzel en 1848 para un tablero estándar de 8x8.
- Resuelto y generalizado a tableros más grandes por Franz Nauck en 1950.
- Poner a n reinas en un tablero de nxn de tal forma que ningún par de reinas se puedan atacar.
  - ningún par de reinas pueden estar en la misma columna, renglón o diagonal.
- En cualquier solución al problema de las n-reinas habrá solamente 1 reina en cada columna del tablero.

http://web.engr.illinois.edu/~jeffe/teaching/algorithms/all-recursion.pdf

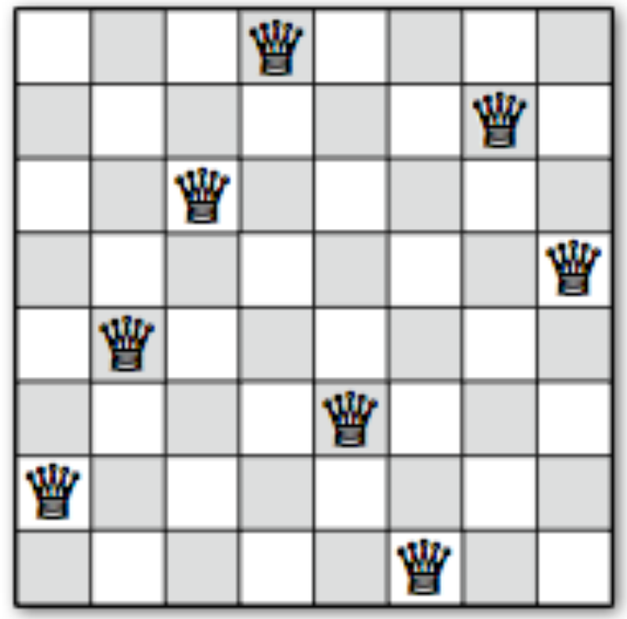
# Problema de las n-reinas

- Las soluciones posibles se representan usando un arreglo Q[1,...,n] donde Q[i] indica la casilla en la columna i que contiene a la reina o 0 si no se ha colocado reina en la columna i.

- Para resolver el problema se colocan las reinas renglón por renglón empezando de arriba.

- Una solución parcial es un arreglo Q[1,...,n] cuyas primeras r-1 entradas son positivas y cuyas últimas n-r+1 entradas son ceros, para un entero dado r.

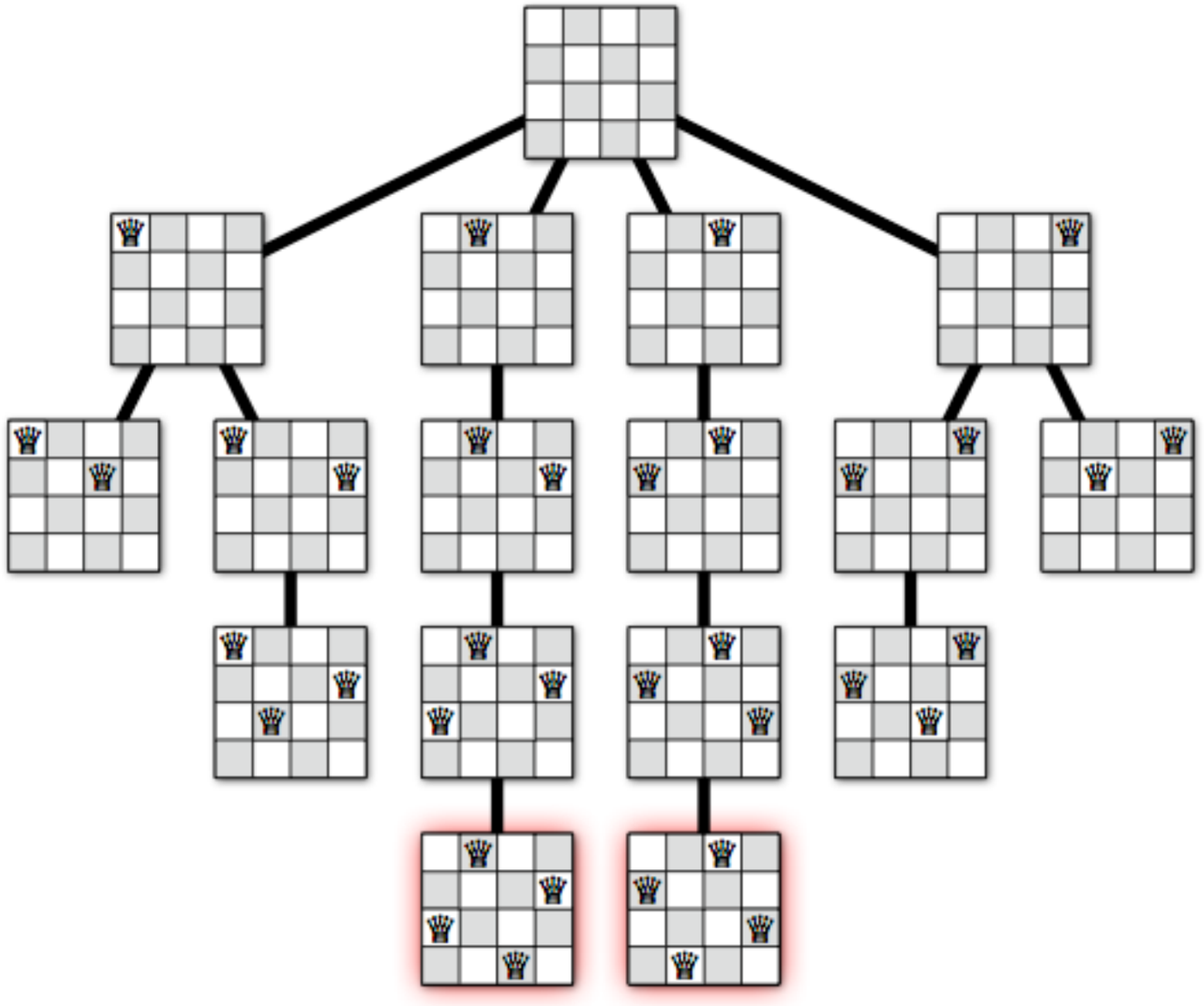
- El algoritmo recursivo enumera todas las soluciones al problema de las n-reinas que sean consistentes con una solución parcial dada.
  - r → primer renglón r vacío.
  - para resolver el problema se llama al procedimiento RECURSIVEQUEENS( Q[1,...,n],1 ).



[7,5,3,1,6,8,2,4]

http://web.engr.illinois.edu/~jeffe/teaching/algorithms/all-recursion.pdf

```
RecursiveNQueens(Q[1..n], r):
  if r = n + 1
      print Q
  else
      for j ← 1 to n
          legal ← True
          for i ← 1 to r − 1
              if (Q[i] = j) or (Q[i] = r − j + i) or (Q[i] = r + i − j)
                  legal ← False
          if legal
              Q[r] ← j
              RecursiveNQueens(Q[1..n], r + 1)
```



http://web.engr.illinois.edu/~jeffe/teaching/algorithms/all-recursion.pdf

# Suma de subconjuntos

- Dado un conjunto de X enteros positivos y un entero meta T, ¿existe un subconjunto de elementos en X que sumen T?

- Notemos que puede haber más de un subconjunto.

- Por ejemplo:

  - si X = {8,6,7,5,3,10,9} y T=15, es VERDADERO gracias a los subconjuntos {8,7} o {7,5,3} o {6,9} o {5,10}.
  - si X = {11,6,5,1,7,13,12} y T = 15 la respuesta es FALSO.

http://web.engr.illinois.edu/~jeffe/teaching/algorithms/all-recursion.pdf

# Suma de subconjuntos

- Hay dos casos triviales:
  - Si el valor meta T=0, podemos regresar TRUE inmediatamente porque los elementos del conjunto vacío suman cero.
  - Si T<0, o si T≠0 pero el conjunto X está vacío, podemos regresar FALSE inmediatamente.
- En el caso general, consideramos el elemento arbitrario x∈X.
- Hay un subconjunto de X que suma T si y solo si uno de los siguientes enunciados es cierto:
  - Existe un subconjunto de X que incluye a x y cuya suma es T.
  - Existe un subconjunto de X que excluye a x y cuya suma es T.

http://web.engr.illinois.edu/~jeffe/teaching/algorithms/all-recursion.pdf

# Suma de subconjuntos

- En el primer caso debe haber un subconjunto de X \ {x} que sume T-x.
- En el segundo caso debe haber un subconjunto de X \ {x} que sume T.

- Podemos resolver SUBSETSUM(X,T) reduciendole a dos instancias más pequeñas:

  - SUBSETSUM(X\{x}, T-x) y SUBSETSUM(X\{x}, T).

- Algoritmo recursivo si X está almacenado en un arreglo:

```
SUBSETSUM(X[1..n], T):
  if T = 0
      return TRUE
  else if T < 0 or n = 0
      return FALSE
  else
      return (SUBSETSUM(X[2..n], T) ∨ SUBSETSUM(X[2..n], T − X[1]))
```

http://web.engr.illinois.edu/~jeffe/teaching/algorithms/all-recursion.pdf

# Suma de subconjuntos

- Exactitud:
  - Si T=0, los elementos del conjunto vacío suman T, la respuesta es TRUE, que es la respuesta correcta.
  - Si T<0 o el conjunto X es vacío, ningún subconjunto de X sumará T, la respuesta es FALSO, que es la respuesta correcta.
  - De modo general, si hay un subconjunto que suma T, entonces contiene a X[i] o no lo contiene y la recursión verifica correctamente cada una de esas posibilidades.

- Tiempo de cálculo:
  - El tiempo de cálculo T(n) satisface la recurrencia $T(n) \leq 2T(n-1) + O(1)$.
  - $T(n) = O(2^n)$.

http://web.engr.illinois.edu/~jeffe/teaching/algorithms/all-recursion.pdf

# Decrease-and-conquer

- Consiste en encontrar una relación entre una solución a un problema de tamaño dado y una solución a una instancia más pequeña.
- Lleva naturalmente a un algoritmo recursivo que reduce el problema a una secuencia de estas instancias más pequeñas hasta que se hace tan pequeño como para solucionarlo trivialmente.

http://web.engr.illinois.edu/~jeffe/teaching/algorithms/all-recursion.pdf

# Celebrity problem

- Entre un grupo de personas n, uno es una celebridad si:
  - no conoce a nadie en el grupo
  - es conocido por todos los del grupo.

- Se determina si un miembro del grupo es celebridad con la pregunta: ¿Conoces a esta persona?

- Suponiendo que en el grupo hay un celebridad, si n=1 la solución es trivial y la persona es una celebridad.
- Si n>1, seleccionamos a dos personas en el grupo, digamos A y B.
  - preguntamos a A si conoce a B.
  - si A conoce a B entonces eliminamos a A de las posibles celebridades.
  - si A no conoce a B eliminamos a B de las posibles celebridades.
  - solucionar recursivamente.

http://web.engr.illinois.edu/~jeffe/teaching/algorithms/all-recursion.pdf

# Decrease-and-conquer

- Algoritmos más rápidos mientras se puedan eliminar a más miembros del grupo más rápido.
- Ej. Adivinar un número (veinte preguntas)..
- Adivinar un número entre 1 y n inclusive haciendo preguntas con respuesta sí o no.
- ¿El número es más grande a n/2?
  - se reduce a la mitad el problema.
  - para un n=1,000,000 encontrar la solución no tomará más de 20 preguntas.
  - sería más rápido si se puede reducir a una tercera parte de su tamaño original.

http://web.engr.illinois.edu/~jeffe/teaching/algorithms/all-recursion.pdf

# Otros métodos

- Algoritmos glotones o greedy.
- Programación dinámica.

http://web.engr.illinois.edu/~jeffe/teaching/algorithms/all-recursion.pdf