3. Técnicas de diseño de algoritmos

1. **Métodos Generales de Soluciones de Problemas**
2. **Técnicas de diseño de algoritmos**
   1. **Recursividad básica**
   2. **Divide y vencerás**
   3. ***Backtracking***

Bibliografía

- **Aho, Hopcroft y Ullman, Cap. 10**
- **Brassard y Bratley, Caps. 7 y 9**
- **Cairó y Guardati, Cap. 4**
- **Weiss, Caps. 1 y 10**
- **Wirth, Cap. 3**

3.1. Métodos Generales de Solución de Problemas (GPSM)

- Existen problemas tipo o clases de problemas genéricos:
  - dar el cambio, el problema de la mochila,... (optimización)
  - el problema del viajante, el problema de las N-reinas,... (búsqueda)

Buscamos algoritmos genéricos

- Existen técnicas o algoritmos eficientes para abordarlos:
  - divide y vencerás,
  - *backtracking*
  - algoritmos voraces (*greedy*),
  - programación dinámica

- **Solución**: ¿a qué clase pertenece el enunciado de un problema dado?

3. 2. 1. Recursividad básica como método de diseño de soluciones

- Definiciones recursivas sencillas
- Fácil obtener equivalente iterativo
- Ejemplos que ya hemos visto:
  - factorial,
  - términos de Fibonacci,…

Reglas fundamentales para resolver un problema mediante recursión

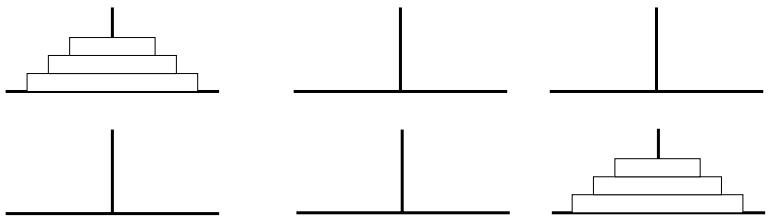
1. **CASOS BASE**
   Deben existir algún / algunos casos base que se puedan resolver sin recursión
2. **PROGRESO**
   Los casos que se resuelven de forma recursiva deben progresar hacia algún caso base
3. **REGLA DE DISEÑO**
   Supóngase que todas las llamadas recursivas van a funcionar
4. **REGLA DE INTERÉS COMPUESTO**
   El trabajo nunca se debe duplicar

3.2.2. Técnica *Divide y Vencerás*

Bibliografía (D*ivide and conquer):*

Weiss, Cap. 10,
Brassard y Bratley, Cap, 7,
Aho, Hopcroft y Ullman, Cap. 10**)**

3.2.2.1. Las Torres de Hanoi



Llevar todos los discos del poste 1 al 2;
sólo se puede mover un disco de cada vez;
nunca puede haber un disco más grande sobre uno más pequeño:
20 discos x 1 minuto x disco → 1048575 minutos → 7281 días → 19,95 años

¿Podemos especificar un algoritmo *iterativo* que nos diga cómo mover los discos?

A poste inicial,

B poste auxiliar,

C poste destino

Solución:

- disponer los postes: A →B → C → A, de tal forma que se puedan mover los bloques "circularmente"
- mover la pieza pequeña un número impar de veces en el sentido de las agujas del reloj
- realizar el único movimiento válido que no implique al disco más pequeño con un número de movimientos pares

3.2.2.2. Divide y Vencerás

Dado un problema arbitrariamente grande y un algoritmo *ad hoc* (denominado sub-algoritmo) que lo resuelve de forma eficiente para un caso (arbitrariamente) pequeño

```
Algoritmo DV (x, sol) es
x: T; {par. dato;}
sol:V;  {par. dato-resultado;}
INICIO
   si x es suficientemente pequeño o sencillo
   entonces
        ad hoc (x, sol);
   sino {dividir el problema x en r sub-problemas}
        descomponer x en x1, x2, ..., xr;
        para i desde 1 hasta r hacer
                DV (xi, soli);
        fin para;
        recombinar las soli para obtener sol;
   finsi
FIN
```

En algunos casos es necesario modificar el esquema anterior, e ir obteniendo progresivamente la solución general, sol, a partir de las soluciones parciales, sol_i

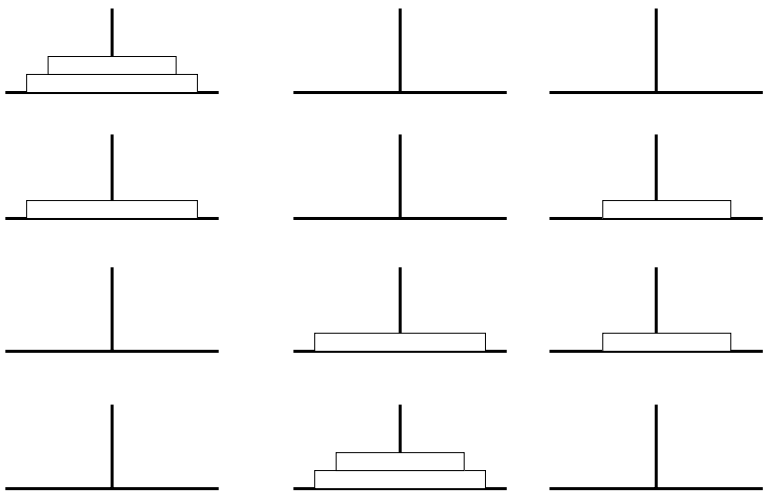
```
Algoritmo DV (x, sol) es
x: T; {par. dato;}
sol: V; {par. dato-resultado;}
INICIO
    si x es suficientemente pequeño o sencillo entonces
        ad hoc (x, sol);
    sino {dividir el problema en r sub-problemas}
        descomponer x en x1, x2, ..., xr;
        para i desde 1 hasta r hacer
            DV (xi, soli);
            recombinar soli para obtener sol;
            {sol es una solución parcial}
        fin para;
        {sol es la solución final}
    fin si
FIN
```

- El *para* puede no ser necesario (p.e. si hay 2 sub-ejemplares)

Torres de Hanoi (solución recursiva):

Mover 2 discos desde Origen hasta Destino usando Aux como auxiliar

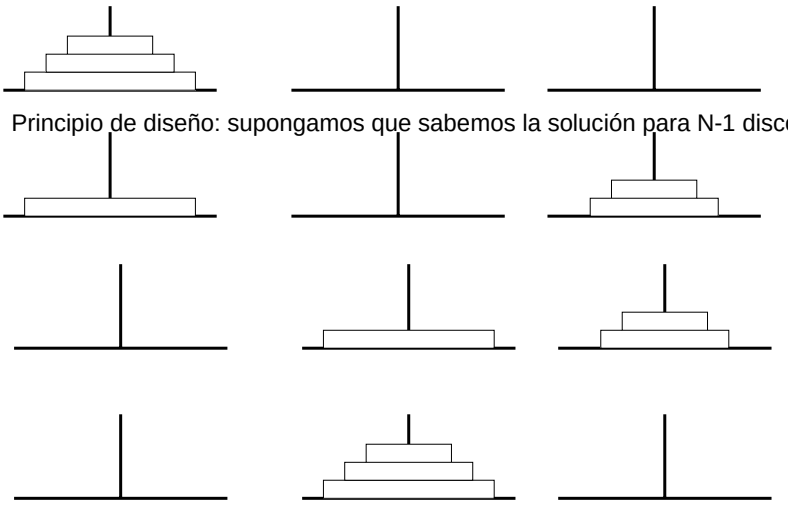
ORIGEN **DESTINO** **AUX**



Torres de Hanoi (solución recursiva):

Mover 3 discos desde Origen hasta Destino usando Aux como auxiliar

ORIGEN **DESTINO** **AUX**



Principio de diseño: supongamos que sabemos la solución para N-1 discos

Torres de Hanoi: Mover n discos desde Origen hasta Destino usando Aux como auxiliar

Solución recursiva:

- **Disponer los postes:**
  Origen → Destino → Aux
- **Los discos se numeran**
  1, 2, ..., n-1, n de menor a mayor

Caso básico:

Sólo se mueve un disco desde "origen" a "destino"

Caso general:

1. **Mover n-1 discos más pequeños desde Origen hasta el auxiliar, Aux**
2. **Mover el disco más grande, n, desde Origen hasta Destino**
3. **Mover los n-1 discos desde Aux hasta Destino**

- El número de sub-ejemplares x_r debe ser r > 1

  Si r = 1, no se habla de Divide y Vencerás, sino de Reducción
- Ejemplos de Reducción:
  - Búsqueda binaria o dicotómica
  - Exponenciación (a^n)

  Cálculo de la potencia a^n de un número

  Solución iterativa

```
Algoritmo pot_it (a, n, pot) es
a, n: {p. dato}numérico;
pot {p. resultado}: numérico;
i: numérico;
INICIO
  pot := 1;
  para i desde 1 hasta n hacer
        pot := pot * a;
  fin para
FIN
```

- Exponenciación (a^n), solución recursiva

solución de pot (a, n)

{casos básicos}

es 1 si n = 0

es a si n = 1

{casos generales: sub-ejemplares}

es $(a^{n/2})\cdot(a^{n/2})$ si n es par

es $a\cdot(a^{n-1})$ si n es impar

Cada sub-ejemplar se soluciona con una llamada recursiva

Comprueba cuál es el número de multiplicaciones que se hace cuando:

- n=7
- n=15

¿qué método hace menos multiplicaciones?

¿cuándo podría ser útil?

Solución recursiva

```
Algoritmo pot_r (a, n, pot) es
a, n: numérico; {p. dato}
pot: numérico; {p. resultado}
INICIO
{casos básicos}
si n = 0 entonces  pot := 1;
sino
  si n = 1 entonces pot := a;
  sino {casos generales}
    si n mod 2 = 0 entonces
          pot_r (a, n div 2, pot);
          pot := pot * pot;
    sino {a es impar}
          pot_r (a, n-1, pot);
          pot := a * pot;
    finsi {casos generales}
  finsi {caso básico 2}
finsi {caso básico 1}
FIN
```

3.2.3. Técnica de Retroceso o *Backtracking*

Bibliografía:

- **Wirth, Cap. 3.4**
- **Russell y Norvig, Bloque II. Cap. 3**
- **Brassard y Bratley, Cap. 9.6**

- Una de las técnicas de solución de problemas es la búsqueda (relacionado con temas de Grafos y Árboles)

- Hemos visto casos muy sencillos: búsqueda asociativa y búsqueda binaria, aplicadas sobre estructuras de datos lineales

La solución de problemas mediante búsqueda consiste en:

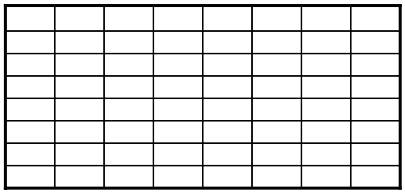
- representar el problema mediante un estado (configuración),
- definir el conjunto de operadores o acciones que nos permiten modificar el estado (configuración),
- especificar una estrategia de búsqueda de la secuencia de operaciones que nos lleven del estado inicial (configuración inicial en la definición del problema) a un estado meta (que se corresponda con una solución al problema)

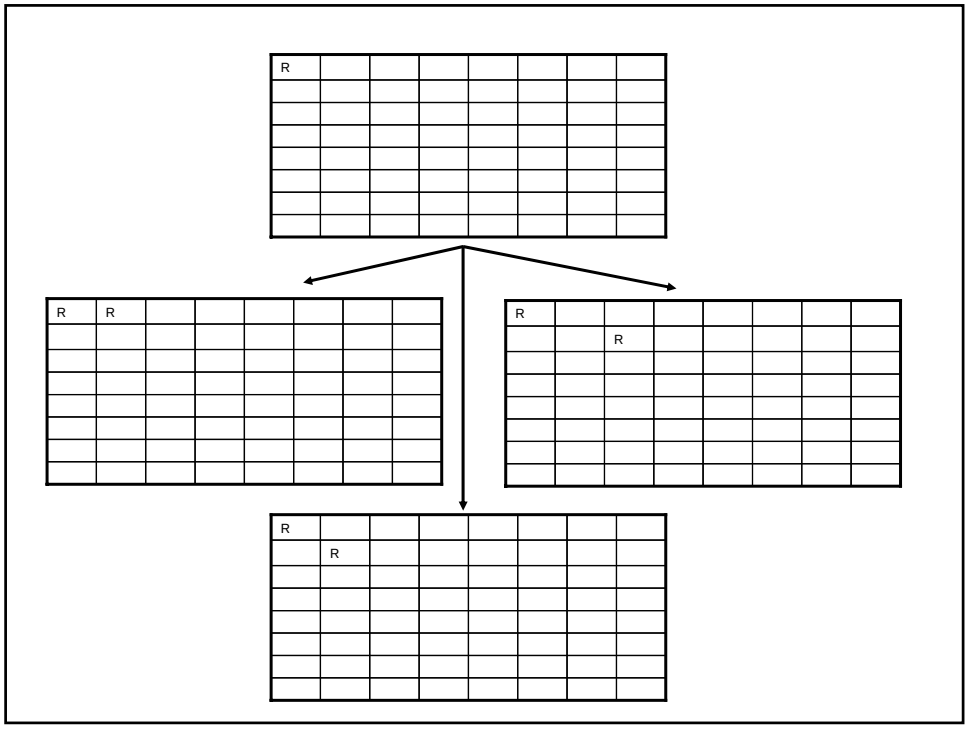
Formulando los problemas de esta manera definimos (implícitamente) un grafo (particularmente un árbol)

- **Encontrar la solución al problema consiste en recorrer el árbol o grafo buscando un nodo (o conjunto de nodos) meta (solución)**

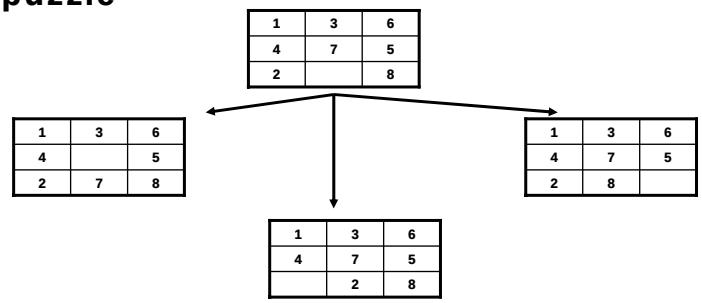
Ejemplos de problemas tipo que se solucionan mediante búsqueda

- N Reinas: ¿se pueden disponer N reinas/damas en un tablero de ajedrez sin que se ataquen?





- **8 puzzle**



Los métodos deben ser exhaustivos, pero además pueden ser:

- **no informados o informados**
- **irrevocables** (anchura o profundidad) que recorren TODA la estructura sin replantearse decisiones intermedias **o tentativos**, como el retroceso

3.2.3.2. Búsqueda con retroceso

- Modificación de la búsqueda en profundidad
- Método tentativo: prueba y error
- Explora sub-problemas, intentando podar el espacio de búsqueda

Ejemplo: Amueblar una casa nueva

Disponemos de un conjunto de muebles

Solución exhaustiva:

Prueba todas las combinaciones posibles

Solución tentativa:

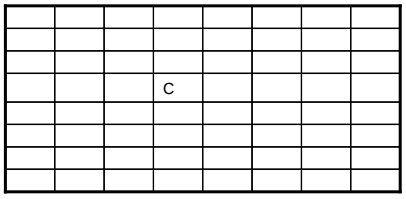
Sabemos que existen disposiciones que no pertenecen a la solución final deseada:

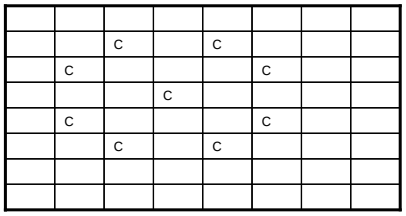
Frigorífico en el salón
Sofá en la cocina
Cama en el baño

No es necesario estudiar configuraciones que las contengan

3.2.3.3. LOS MOVIMIENTOS DEL CABALLO DE AJEDREZ

¿Puede recorrerse exhaustivamente un tablero de ajedrez usando los movimientos del caballo y pasando una única vez por cada casilla?





- No sabemos, a priori, si existe solución
- 8 x 8 posiciones iniciales
- ≤ 8 posibles desplazamientos en cada nueva jugada
- Complejidad exponencial
- Solución más sencilla:
  - Probar una nueva jugada
  - Anotarla, como parte de la posible solución, si se puede avanzar
  - Descartar la jugada si no puede avanzar, quitándola de la posible solución

RETROCESO:
desandamos el camino, paso a paso, cuando no podemos avanzar

3.2.3.4. Algoritmo genérico de retroceso

Número de movimientos (jugadas) posibles es finito en cada paso

```
Algoritmo Ensayar_Retroceso es
Inicio
  Inicializar conjunto posibles movimientos; {para nivel actual de la solución}
  repetir
    seleccionar siguiente movimiento; {de este nivel}
    si aceptable entonces
      anotarlo; {como parte solución hasta ahora}
      si solución incompleta entonces
        Ensayar_Retroceso; {siguiente nivel}
        si no hubo éxito entonces
            cancelar anotación;
            {retroceso, se elimina última anotación de la solución}
        finsi;
      fin si; {¿solución incompleta?}
    fin si; {¿aceptable?}
  hasta que éxito o no haya más posibilidades
Fin
```

- Puede acotarse la profundidad de búsqueda
- Pueden indicarse número máximo de candidatos por nivel
- Esta versión ya NO ES EXHAUSTIVA

Algoritmo Ensayar_Retroceso_2 (nivel) es

```
nivel: numérico; {p. dato-resultado}
k: numérico; {número de candidatos por nivel}
Inicio
      Inicializar conjunto posibles movimientos;
      k := 0;
      repetir
         k := k + 1:
         seleccionar movimiento k-ésimo;
         si aceptable entonces
           anotarlo; {como parte solución hasta ahora}
           si solución incompleta AND  nivel < n entonces
                    Ensayar_Retroceso_2 (nivel + 1);
                    si no hubo éxito entonces
                              cancelar anotación;
                    finsi;
           fin si {¿solución incompleta?}
         fin si {¿aceptable?}
  hasta que éxito o k = m {nº máximo candidatos}
Fin
```

3.2.3.5. Recorrido del tablero con el caballo

Estructuras básicas

{tipo}

Tablero = vector[1..8,1..8] de numérico;

{Variables}

T: Tablero;

{ T(x,y) = 0; si no ha sido visitada

T(x,y) = i; si ha sido visitada en la jugada i ($1 \le i \le n^2$)}

resp: lógico; {Cierto=éxito, Falso= fracaso}

Siguiente jugada o movimiento

Coordenadas (u, v) a partir de las actuales (x,y)

i := i + 1

Jugada aceptable

T(u,v) = 0 AND $1 \le u \le n$ y $1 \le v \le n$

Éxito

i = n^2

```
Algoritmo ensayar_caballo_R (i, x, y, resp, T) es
i, x, y: numérico; {p. dato-resultado}
resp: lógico; {p. resultado}
T: Tablero; {p. dato-resultado}
J: Jugadas; {variable interna}
u, v: numérico; {variable interna}
resp2: lógico; {variable interna}
Inicio
 Calcular_jugadas_posibles (J, x, y);
 repetir
     Calcular_siguiente (J, u, v);
     si 1 ≤ u and u ≤ n and 1 ≤ v and v ≤ n and T(u, v) = 0
     entonces
        T(u, v) := i; {provisionalmente}
```

```
        si i < n² entonces
         ensayar_caballo_R(i+1,u, v, resp2, T);
         si resp2 = falso entonces
           T(u,v) := 0; {retroceso}
         finsi; {not resp2}
        sino
          resp2 := cierto;
         finsi; {¿llegó solución?}
    finsi; {¿jugada aceptable?}
   hasta que resp2 = cierto OR no haya más jugadas;
   resp := resp2;
Fin
```

Ejercicio:

Plantead una solución utilizando retroceso para el problema de las 8-reinas