2. Recursividad

1. La recursividad
2. Programación recursiva
   1. Recursión directa como método de diseño de soluciones
   2. Ejemplos
3. Funcionamiento de la recursión
   1. ¿Cómo se comporta un algoritmo recursivo?
   2. Criterios para usar soluciones recursivas
4. Versiones iterativas para evitar la recursividad

Bibliografía

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

2. 1. Recursividad

Algoritmos + EE.DD. = Programas

Hasta ahora sólo hemos visto algoritmos con tratamiento secuencial

2.1.1. Definiciones y ejemplos

- **Recurrir / Recurrente (DRAE)**
  Recurrir: "Dicho de una cosa: volver al lugar de donde salió".
  Recurrente: Dicho de un proceso: Que se repite
- Recursividad o recursión vs definición circular
- **Concepto de definición recursiva**
  la definición recursiva hace referencia a un caso similar más sencillo, no a sí mismo
  - Permite dar definiciones finitas de conjuntos infinitos (ejemplo: conjunto números naturales por inducción, lenguaje de la lógica de proposiciones,...)

- **Fórmula o función recursiva**
  aquella que se define en función de sí misma

  Ejemplos:
- **Factorial de n:**
  - n! ó fact(n) = 1 si n = 0 ó n = 1
  - n! ó fact(n) = n *(n-1)! = n * fact(n-1)

  Existe una definición no recursiva:

  $n ! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1$
- Término n-ésimo de la **sucesión de Fibonacci**:
  fibo(n) := fibo(n-1) + fibo(n-2)
- **Capital acumulado:**

  C_0 := Capital inicial

  C_1 := (1+ % intereses anuales) $\cdot C_0$

  ...

  C_i := (1+ % intereses anuales) $\cdot C_{i-1}$
- **Estructuras de datos recursivas**

  Árboles, listas, grafos, conjuntos,... (Veremos estas definiciones en el Bloque 2)

2. 2. Programación recursiva

- La recursividad es una técnica de diseño de algoritmos que se base en solucionar versiones más pequeñas de un problema, para obtener la solución general del mismo
- Permiten realizar un número (posiblemente infinito) de cálculos sin especificar el número de repeticiones de forma explícita
- Algoritmos recursivos:
  - **Recursión directa**
  - **Recursión indirecta**

  Pueden conseguirse versiones iterativas de soluciones recursivas, como por ejemplo: búsqueda binaria

  Vamos a ver en el tema siguiente técnicas de diseño de programas que utilizan recursión:
  - Divide y vencerás,
  - Retroceso o *backtracking*

2. 2. 1. Recursión directa como método de diseño de soluciones

- Ejemplos:
  - factorial,
  - términos de Fibonacci,
  - visualizar un número cifra a cifra
  - búsqueda binaria o dicotómica

- Se pueden obtener fácilmente equivalentes iterativos

Es necesario formular:

- **Caso base**: del cual se conoce directamente la solución
- **Caso general**: que debe poder resolverse en función del caso base y un caso general de menor tamaño (que progrese hacia una solución más sencilla → hacia el caso base)

Reglas fundamentales para resolver un problema mediante recursión directa

1. **CASO/S 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 siempre 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, resolviendo el mismo ejemplar de un problema en llamadas recursivas separadas (→ programación dinámica)

2.2.2. Ejemplos

- Cálculo del factorial de un número mediante fact_r(),
- Cálculo del n-ésimo término de la sucesión de Fibonacci mediante fibo_r(),
- Búsqueda binaria
- Visualización de los dígitos de un número, visualiza_número(), utilizando sólo una función visualiza_dígito () que escriba números de una sóla cifra

2.3. Funcionamiento de la recursión

2.3.1. ¿Qué ocurre al usar programación recursiva?

Es necesario guardar el estado de cada programa antes de cada llamada (variables locales, parámetros y punto de ejecución) para que sepa seguir después de la llamada

Debemos saber:

- la recursión no es más rápida que la iteración (ya que implica guardar entornos),
- consume muchos recursos (memoria)

2.3.2. Criterios para usar soluciones recursivas

- la definición del problema o su solución es inherentemente recursiva, y su equivalente iterativa es enrevesada
- manejamos estructuras de datos recursivas y existen algoritmos recursivos

Cómo debe utilizarse

- Garantizando que el programa termina en un número finito de pasos
- El número de llamadas a realizar no sólo es finito, sino también pequeño

Cuándo no debe utilizarse

- Existen soluciones iterativas sencillas