Taller de preparación para la OCI - PUC

Recursión

1. Introducción a recursión

Hasta ahora, hemos resuelto muchos problemas usando el concepto de iteración. Por ejemplo, para calcular la suma de todos los elementos de un arreglo de números, usamos palabras claves como `while` y `for`. La iteración consiste en repetir una porción de código algún número de veces para lograr un objetivo de cómputo. La iteración a primera vista parece necesaria en los lenguajes de programación, porque hay problemas cuya solución es repetitiva. Volviendo al ejemplo anterior, para sumar todos los elementos de un arreglo, necesariamente debemos tener acceso a cada elemento del arreglo, y si no sabemos con anticipación cuál es el tamaño del arreglo (podría ser muy grande), debemos expresar la solución como un algoritmo repetitivo.

La iteración, sin embargo, no es la única forma de resolver un problema que requiere repetición. Existe una forma alternativa: la recursión. Como veremos, recursión es una técnica extremadamente poderosa. De hecho, existen clases de problemas de computación cuyas soluciones requieren de repetición y para los cuales sólo es posible dar una solución elegante usando la técnica de recursión. Recursión, cuando es bien utilizada, generalmente resulta en soluciones muy elegantes, de pocas líneas de código.

Nótese lo radical de lo que hemos dicho arriba. La iteración no es necesaria si tenemos recursión a nuestro alcance. Dicho de otro modo, el `for` y el `while` son dispensables si uno conoce bien la técnica de recursión.

2. La repetición descrita como recursión

La recursión es una técnica natural. Esto es porque las personas, incluso cuando no tienen conocimientos de programación, la utilizan. Imagina que un niño pequeño se acerca a preguntarte **"¿como puedo pintar este árbol?"**.

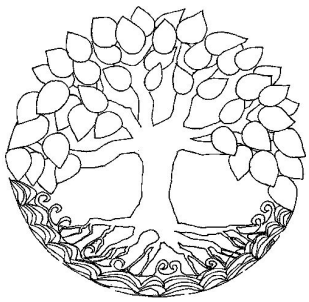

Figura 1: Mandala.

Una respuesta sencilla (recordemos que el niño es pequeño) podría ser la siguiente: "Primero, pinta el tronco de color café. Luego haz lo siguiente: escoge una hoja, píntala verde, y luego sigue con el resto de las hojas. Habrás terminado cuando no queden hojas por pintar". Esta forma de describir cómo pintar el árbol es recursiva. De hecho, le hemos explicado al niño un procedimiento para pintar el árbol completo. Este procedimiento pinta una hoja y luego se invoca a sí mismo al "seguir con el resto". El

procedimiento termina cuando no existe una hoja sin pintar.

3. Funciones Recursivas: Más Allá de los Ejemplos Simples

Las siguientes son dos características de las funciones recursivas.

1. Definen una solución concreta para uno o varios problemas pequeños. Esto se conoce como **el/los caso(s) base**.
2. Definen la solución a un cierto problema P en términos de la solución de uno o varios problemas más pequeños que P. La implementación de esto se hace mediante lo que se conoce como un **llamado recursivo**, el que usualmente se implementa mediante un llamado a la misma función.

Hay muchas funciones matemáticas que naturalmente se expresan de manera recursiva. Por ejemplo, la serie de Fibonacci, que corresponde a los números que resultan de sumar los dos anteriores (partiendo con 1 y 1 en la secuencia), es la siguiente: $1, 1, 2, 3, 5, 8, 13, 21, ...$ (hasta el infinito). Obtener el número de Fibonacci en la posición n se puede definir mediante una función **f** que satisface:

$$fib(n) = \begin{cases} 0 & \text{si n} = 0 \text{ o n} = 1 \\ fib(n-1) + fib(n-2) & \text{en otro caso} \end{cases} \tag{1}$$

Por otro lado, la función factorial, que es el resultado de multiplicar todos los números entre 1 y n (posición que estamos calculando) se puede definir como:

$$factorial(n) = \begin{cases} 1 & \text{si n} = 0 \\ n \cdot factorial(n-1) & \text{en otro caso} \end{cases} \tag{2}$$

Ejercicio: Escribe versiones recursivas e iterativas para ambas funciones.

En este punto tú podrías sentir algo de decepción. Todos los ejemplos que hemos visto se pueden resolver usando `for` y `while`. Además, recursión tal vez no te parece ser la técnica más natural del mundo, pero te podemos asegurar que hay ciertos casos en donde es más simple programar así.

4. Ejemplos

Ejemplo 4.1. Haga un método que entregue todas las permutaciones de un `String`. Por ejemplo las permutaciones de `'abc'` son `'abc'`, `'acb'`, `'bac'`, `'bca'`, `'cab'`, `'cba'`. Para esto use una función recursiva.

```
1  public static void eliminar(String s, int i) {
2    if (s.length == 0) {
3        return s.substring(1, s.length);
4      }
5      else if (i == s.length - 1){
6        return s.substring(0, s.length - 1);
7      }
8      else {
9        return s.substring(0, i) + s.substring(i+1, s.length);
10     }
11 }
12
```

```
13 public static void permutar(String s, String perm) {
14   // Si el largo de perm es 1 la recursión termina
15   if (perm.length == 1) {
16       System.out.println(s + perm);
17     }
18     else if {
19       for (int i = 0; i < perm.length; i ++) {
20             String s1 = s + perm.charAt(i);
21             String s2 = eliminar(perm, i);
22             permutar(s1, s2);
23         }
24     }
25 }
26
27 public static void llamar_permutar(String s) {
28   permutar("", s);
29 }
```