75.04 Algoritmos y Programación II

Práctica 2: recursividad

Notas preliminares

- El objetivo de esta práctica es familiarizar a los alumnos con el concepto de recursividad, fundamental en programación y sobre todo en técnicas de diseño de algoritmos.
- Los ejercicios marcados con el símbolo ♣ constituyen un subconjunto mínimo de ejercitación. No obstante, recomendamos fuertemente realizar todos los ejercicios.

Ejercicio 1

Dada

```
int sum_of_squares(int n)
{
        if (!n)
                return 0;
        return n * n + sum_of_squares(n - 1);
}
```

Si existe, proponer una implementación "más eficiente". Justificar.

Ejercicio 2 ♣

Los coeficientes binomiales se definen mediante la siguiente relación recurrente:

$$\begin{cases} C(n,0) = C(n,n) = 1, & n \geq 0 \\ C(n,k) = C(n-1,k) + C(n-1,k-1), & n > k > 0 \end{cases}$$

(a) Escribir un programa recursivo que calcule $C(n, k)$.

(b) Construir el árbol de recursividad para $C(6, 4)$.

Ejercicio 3

Dibujar el árbol de recursividad para ordenar los elementos $7, 4, 1, 8, 5, 2, 9, 6, 3, 0$

(a) usando merge sort;

(b) usando quicksort.

Ejercicio 4 ♣

Escribir un algoritmo recursivo que calcule el determinante de una matriz de $n \times n$ por definición. Dibujar el árbol de recursión cuando la entrada es

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & -1 \\ -1 & 1 & 0 \end{pmatrix}$$

Calcular la máxima cantidad de *stack frames*[1] usados por la implementación y estimar la cantidad de operaciones realizadas en función del tamaño de la entrada. Justificar.

[1] Cada nueva función o método llamado apila un *frame* en el stack del proceso, para almacenar, entre otras cosas, las variables locales a la función. Es por esto que, en cualquier instante de tiempo, un programa en ejecución puede ser caracterizado por la secuencia de *call frames* que causaron la invocación de la función actual. Esta secuencia se llama *backtrace*.

Ejercicio 5 ♣

Dada:

```
public static int mcCarthy(int n)
{
    if (n > 100)
        return n - 10;
    else
        return mcCarthy(mcCarthy(n + 11));
}
```

(a) Determinar el valor de `mcCarthy(50)`. ¿Cuántas llamadas recursivas hace `mcCarthy()` en este cómputo?

(b) Mostrar que el caso base es eventualmente alcanzado para todos los valores de `n`.

Ejercicio 6

En el problema de las torres de Hanoi, calcular la cantidad de movimientos de discos en la solución sugerida en el libro de Kruse, si la cantidad de discos a mover es n.

Ejercicio 7

Escribir una implementación recursiva del algoritmo merge sort,

```
template<typename T>
void merge_sort(vector<T> &v);
```

Evaluar la performance en función del tamaño n del vector a ordenar:

(a) Obtener una expresión de la mínima cantidad de comparaciones de claves en función de n.

(b) Obtener una expresión de la máxima cantidad de stack frames activos, en función de n.

Ejercicio 8

Comparar las siguientes implementaciones, señalando los puntos débiles y fuertes de cada una. Justificar.

```
int fibonacci1(int n)
{
        if (n == 0 || n == 1)
                return n;
        return fibonacci(n - 1) + fibonacci(n - 2);
}


int fibonacci2(int n)
{
        int prev = -1;
        int result = 1;
        for (int i = 0; i <= n; ++i) {
                int sum = result + prev;
                prev = result;
                result = sum;
```

```
        }
        return result;
}

// fibonacci3
template<int num> struct fibonacci
{
        static const int valor = fibonacci<num - 1>::valor +
                                 fibonacci<num - 2>::valor;
};

template<> struct fibonacci<1>
{
        static const int valor = 1;
};

template<> struct fibonacci<0>
{
        static const int valor = 0;
};
```

Ejercicio 9

Dado un conjunto S, el *conjunto de partes* de S, notado $\mathfrak{P}(S)$, es el conjunto de todos los subcojuntos de S. Por ejemplo,

$$\mathfrak{P}(\{2,3,1\}) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$$

Proponer un algoritmo recursivo para, dado S, calcular $\mathfrak{P}(S)$.

¿Cuál es la cantidad mínima de operaciones que este algoritmo puede realizar, expresada en función de $n = |S|$?

Ejercicio 10 ♣

Diseñar un algoritmo recursivo para, dada una secuencia de elementos s, computar la secuencia de todas las permutaciones de s en algún orden arbitrario.

Por ejemplo, si $s = \langle a, b, c \rangle$, el algoritmo debe devolver una secuencia como la siguiente:

$$\langle \langle a,b,c \rangle, \langle a,c,b \rangle, \langle b,a,c \rangle, \langle b,c,a \rangle, \langle c,a,b \rangle, \langle c,b,a \rangle \rangle$$

¿Cuál es la cantidad mínima de operaciones que este algoritmo puede realizar, expresada en función de $n = |s|$?