Introducción a la Programación

Prof. Agustín Gravano

Primer semestre de 2022

Clase teórica 23: Recursión algorítmica

Ejemplo: Factorial

Problema: Dado un entero $n \geq 0$, devolver $n! = 1 \cdot 2 \cdot \ldots (n-1) \cdot n$

```python
def fact(n:int) -> int:
    ''' Devuelve n!. Requiere: n>=0 '''
    vr:int = 1
    while n > 0:
        vr = vr * n
        n = n - 1
    return vr
```

Ejemplo: Factorial

Observación: $\texttt{fact}(n) = \big(1 \cdot 2 \cdot \ldots \cdot (n-1)\big) \cdot n$

$$= \texttt{fact}(n-1) \qquad \cdot n$$

$\texttt{fact}(n) = \texttt{fact}(n-1) \cdot n$ es una **definición recursiva**, pero todavía está incompleta.

```
1 def fact(n:int) -> int:
2     return fact(n-1) * n
```

Ejemplo: queremos ejecutar fact(3), pero esto requiere ejecutar fact(2) (en la línea 2), que a su vez requiere ejecutar fact(1), que a su vez requiere ejecutar fact(0), que a su vez requiere ejecutar fact(-1), que a su vez...

Ejemplo: Factorial

Nos faltó frenar la recursión: cuando llegamos a `fact(0)`, debemos parar y devolver el resultado: 1. A esto se lo llama caso base.

```python
def fact(n):
    ''' Devuelve n!. Requiere: n>=0'''
    if n==0:    # caso base
        return 1
    else:       # caso recursivo
        return fact(n-1) * n
```

Ejemplo: queremos ejecutar `fact(3)`, pero esto requiere ejecutar `fact(2)`, que a su vez requiere ejecutar `fact(1)`, que a su vez requiere ejecutar `fact(0)`, que devuelve 1, lo cual frena la recursión. Después, *a la vuelta de la recursión*, `fact(1)` devuelve 1; `fact(2)` devuelve 2 y por último `fact(3)` devuelve 6.

Importante: Notar que `fact(n-1)` es un problema más simple que `fact(n)`. Está más cerca del caso base `fact(0)`.

Ejemplo: Factorial

$$n! = 1 \cdot 2 \cdot \ldots \cdot (n-1) \cdot n$$

$$n! = \prod_{i=1}^{n} i$$

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ (n-1)! \cdot n & \text{si } n > 0 \end{cases}$$

```python
# algoritmo iterativo
def fact(n:int) -> int:
    vr:int = 1
    while n > 0:
        vr = vr * n
        n = n - 1
    return vr
```

```python
# algoritmo recursivo
def fact(n:int) -> int:
    if n==0:
        return 1
    else:
        return fact(n-1) * n
```

Recursión algorítmica

La solución a un problema depende de la solución a instancias de menor tamaño del mismo problema.

Recursión algorítmica

1. Resolver el problema para los casos base.

2. Suponiendo que se tiene resuelto el problema para instancias de menor tamaño, modificar dichas soluciones para obtener una solución al problema original.

La recursión ofrece otra forma de encarar la resolución de problemas.

No es mejor ni peor; es una herramienta más de programación.

```python
def fact(n:int) -> int:
    ''' Devuelve n!. Requiere: n>=0 '''
    if n==0:
        return 1
    else:
        return fact(n-1) * n
```

Ejercicio: Escribir una función recursiva suma que, dado un entero $n \geq 0$, calcule $1 + 2 + \cdots + n$.

Ejemplos:

- suma(0) $\rightarrow$ 0
- suma(1) $\rightarrow$ 1
- suma(3) $\rightarrow$ 6
- suma(10) $\rightarrow$ 55

Ejemplo: Sumar elementos de una lista

Problema: Dada una lista de enteros, devolver la suma.

Observación: `sumar([4,1,3,7]) = 4 + sumar([1,3,7])`

En general, podemos definir `sumar` en forma recursiva:

$$\texttt{sumar(xs)} = \begin{cases} 0 & \text{si } \texttt{xs} \text{ es vacía} \\ \texttt{xs[0] + sumar(xs[1:])} & \text{si no} \end{cases}$$

```python
def sumar(xs:List[int]) -> int:
    ''' Devuelve la suma de los elementos de la lista.
        Requiere: Nada. '''
    if len(xs) == 0:
        return 0
    else:
        return xs[0] + sumar(xs[1:])
```

Importante: Notar que `sumar(xs[1:])` es un problema más simple que `sumar(xs)`. Está más cerca del caso base `sumar([])`.

```python
def sumar(xs:List[int]) -> int:
    ''' Devuelve la suma de los elementos de la lista.
        Requiere: Nada. '''
    if len(xs) == 0:
        return 0
    else:
        return xs[0] + sumar(xs[1:])
```

Ejercicio: Escribir una función recursiva `maximo` que, dada una lista no vacía de enteros, devuelva su máximo elemento.

Ejemplos:

- `maximo([1])` $\rightarrow$ 1
- `maximo([10,20,15,40,1])` $\rightarrow$ 40

Repaso de la clase de hoy

- Recursión algorítmica
- Caso base y caso recursivo de las funciones recursivas
- Recursión sobre enteros y sobre listas.

Bibliografía complementaria:

- APPP2, secciones 4.9 a 4.11, 5.5 a 5.8.
- HTCSP3, capítulo 18 (la próxima clase veremos el módulo `turtle`).

Con lo visto, ya pueden resolver hasta el Ejercicio 3 (inclusive) de la Guía de Ejercicios 10.