



Computación I

Curso 2025

Facultad de Ingeniería
Universidad de la República

Programas Recursivos

Redundancia

■ Función Fibonacci

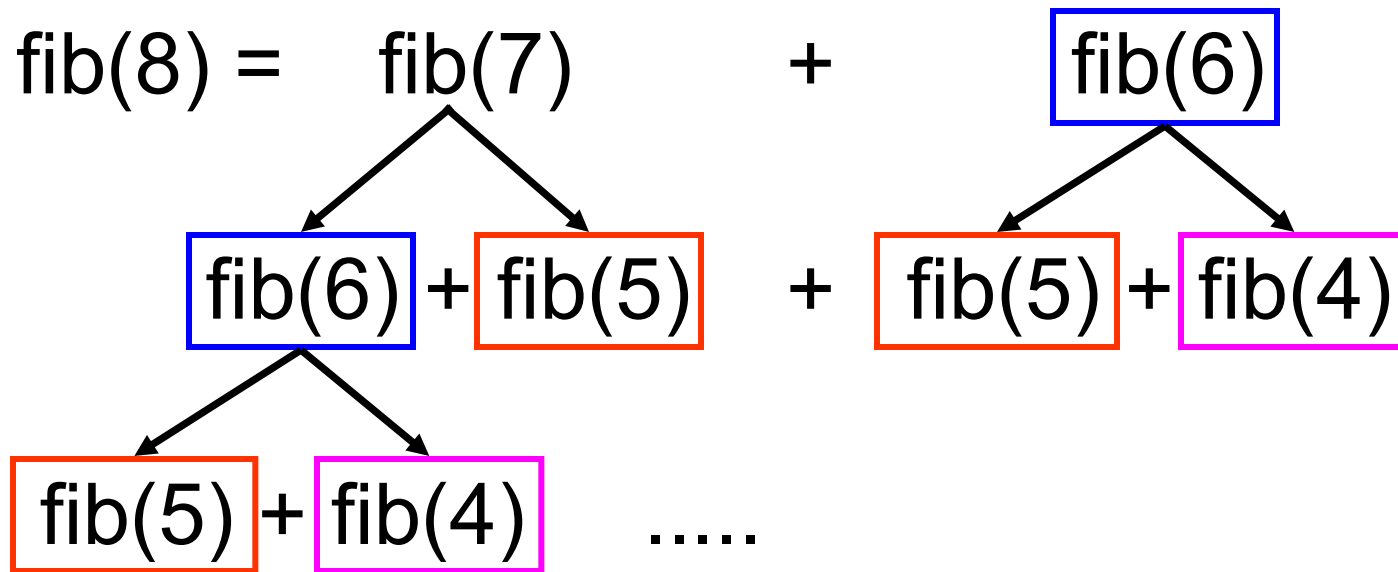
- $\text{Fib}(1) = 0$
- $\text{Fib}(2) = 1$
- $\text{Fib}(n) = \text{Fib}(n-1) + \text{Fib}(n-2)$

■ Programa Recursivo

```
function fn=fib(n)
if n == 1
    fn = 0;
elseif n == 2
    fn = 1;
else
    fn = fib(n-1) + fib(n-2);
end
```

Programas Recursivos

Redundancia



■ Redundancia en cálculos

Recursión vs Iteración

- No siempre es conveniente utilizar una solución recursiva.
 - Problemas de memoria.
 - Se podría cancelar la ejecución por falta de memoria.
 - Redundancia en cálculos.
 - Tiempo que crece exponencialmente.

Recursión vs Iteración

■ Recursiva

```
function fn=fibrec(n)
if n == 1
    fn = 0;
elseif n == 2
    fn = 1;
else
    fn = fibrec(n-1) + fibrec(n-2);
end
```

Recursión vs Iteración

■ Iterativa

```
function fn=fibiter(n)
if n==1
    fn=0;
elseif n==2
    fn=1;
else
    penult=0;
    ult=1;
    for i=3:n
        nue = penult + ult;
        penult = ult;
        ult = nue;
    end
    fn = ult;
end
```

■ Evita cálculos repetitivos

- Utiliza variables auxiliares para almacenar resultados intermedios

■ Menos clara.

■ Más eficiente.

Recursión vs Iteración

- Ventajas de la Recursión.

- Soluciones simples y claras a problemas inherentemente recursivos.

- Desventajas de la Recursión: INEFICIENCIA

- Memoria

- Nueva asignación de memoria en cada llamada recursiva.

- Procesamiento

- Redundancia en cálculos.

- Claridad vs. Sobrecarga



Ejercicio

Escribir un programa recursivo que calcule los cuadrados de los enteros hasta N.

Ejercicio - Solución

```
function y = Cuadrados(n)
if n == 0
    y = [0];
else
    y = [Cuadrados(n-1) n*n];
end
```

Programas Recursivos

Ejemplo

- Búsqueda de un elemento en un vector
 - Posibles casos base
 - Vector vacío
 - Vector con un elemento
 - Posible caso recursivo
 - Vector con más de un elemento
 - En cada llamada recursiva reducir el tamaño del vector

Programas Recursivos

Ejemplo

```
function fn=Pertenece(n,a)
```

```
if isempty(a)  
    fn=0;
```

Casos Base

```
else
```

```
    if a(1)==n  
        fn=1;
```

Vector más Pequeño

```
    else
```

```
        fn=Pertenece(n,a(2:length(a))) ;
```

```
    end
```

```
end
```

Caso Recursivo



Ejercicio

Sumar todos los elementos de un vector.

Ejercicio

Solución

```
function y = Sumar(v)
if length(v)==0
    y = 0;
else
    y = v(1)+Sumar(v(2:length(v))) ;
end
```

Solución – partiendo equitativamente

```
function y = Sumar(v)
n = length(v) ;
if n == 0
    y = 0;
elseif n == 1
    y = v(1) ;
else
    y = Sumar(v(1:floor(n/2))) +
        Sumar(v(floor(n/2)+1:n)) ;
end
```



Ejercicio

Sumar todos los elementos de un vector,
devolver -1 si el vector es vacío.

Ejercicio

Solución

```
function y = Sumar(v)
if length(v)==0
    y = -1;
elseif length(v)==1
    y = v(1);
else
    y = v(1)+Sumar(v(2:length(v)));
end
```




Ejercicio

Buscar un elemento en un vector y devolver su posición. Si el elemento no pertenece al vector, devolver -1.

Ejercicio

Solución

```
function y = Buscar(elem,v)
if length(v)==0
    y = -1;
elseif v(1) == elem
    y = 1;
else
    posicion = Buscar(elem,v(2:length(v)));
    if posicion == -1
        y = - 1;
    else
        y = posicion + 1;
    end
end
```

Ejercicio

Una solución mejor

```
function y = Buscar(elem,v)
if length(v)==0
    y = -1;
elseif v(length(v))==elem
    y = length(v);
else
    y = Buscar(elem,v(1:length(v)-1));
end
```



Ejercicio

Escriba una función recursiva que separe los elementos pares e impares de un vector.

Ejercicios

Solución

```
function [vimpar,vpar] = separar(v)
    if length(v)==0
        vimpar = [];
        vpar = [];
    else
        [vimpar, vpar] = separar(v(2:length(v)));
        if mod (v(1),2) == 0
            vpar = [v(1), vpar];
        else
            vimpar = [v(1), vimpar];
        endif
    endif
endfunction
```



Ejercicio

Escriba una función recursiva que separe los 10 últimos elementos pares y los 10 últimos elementos impares de un vector.

Ejercicios

Solución

```
function [vimpar,vpar] = separar_ultimos10(v)
    if length(v)==0
        vimpar = [];
        vpar = [];
    else
        [vimpar, vpar] = separar_ultimos10 (v(2:length(v)));
        if mod (v(1),2) == 0 & length(vpar) < 10
            vpar = [v(1), vpar];
        endif
        if mod (v(1),2) == 1 & length(vimpar) < 10
            vimpar = [v(1), vimpar];
        endif
    endif
endfunction
```



Ejercicio 1

Buscar la n -ésima ocurrencia de un elemento en un vector y devolver su posición. Si el elemento no pertenece al vector o está menos de n veces, devolver -1.

Ejercicio 1

Solución

```
function res = n_ocurr(n,elem,v)
largo = length(v)
if largo == 0
    res = -1;
elseif n == 1 & v(1) == elem
    res = 1;
else
    if v(1) == elem
        res = n_ocurr(n-1, elem,v(2:largo));
    else res = n_ocurr(n, elem,v(2:largo));
    end
    if res ~= -1
        res = res + 1;
    end
end
```

Ejercicio 1

Solución “optimizada”

```
function res = n_ocurr(n,elem,v)
largo = length(v)
if largo < n || largo == 0
    res = -1;
elseif n == 1 & v(1) == elem
    res = 1;
else
    if v(1) == elem
        res = n_ocurr(n-1, elem,v(2:largo));
    else res = n_ocurr(n, elem,v(2:largo));
    end
    if res ~= -1
        res = res + 1;
    end
end
```

Ejercicio 1

Solución “optimizada” considerando caso n 0

```
function res = n_ocurr(n,elem,v)
largo = length(v)
if n == 0
    res = 0;
elseif largo < n || largo == 0
    res = -1;
else
    if v(1) == elem
        res = n_ocurr(n-1, elem,v(2:largo));
    else res = n_ocurr(n, elem,v(2:largo));
    end
    if res ~= -1
        res = res + 1;
    end
end
```



Ejercicio 2

Intercalar ordenadamente los elementos de
dos vectores ordenados.

Cada vector representa un conjunto de números, y el resultado de intercalar es también un conjunto.

No hay elementos repetidos en un conjunto.

Ejercicio 2

```
function v = intercalar(v1,v2)
if length(v1)==0
    v = v2;
elseif length(v2)==0
    v = v1;
elseif v1(1) < v2(1)
    v = [v1(1), intercalar(v1(2:length(v1)),v2)] ;
elseif v2(1) < v1(1)
    v = [v2(1), intercalar(v1,v2(2:length(v2)))] ;
else v = [v1(1), intercalar(v1(2:length(v1)),
                             v2(2:length(v2)))]
end
```

Ejercicio 3

Implementar en Octave la suma de los elementos de una matriz en forma recursiva.

Se puede procesar de a una fila o de a una columna. En este caso lo resolveremos procesando de a una fila, usando una función auxiliar también recursiva.

Ejercicio 3

Solución

```
function y=sumaMatriz(M)
    [m,n]=size(M) ;
    if m==0
        y=0;
    else
        y=sumaVector(M(1,1:n)) +
            sumaMatriz(M(2:m,1:n)) ;
    end
```

Ejercicio 3

Solución

```
function y=sumaVector(v)
    n=length(v);
    if n==0
        y=0;
    else
        y=v(1) + sumaVector(v(2:n));
    end
```


Ejercicio 4

Implementar en Octave la suma de los elementos de una matriz en forma recursiva.

Si la matriz es cuadrada y su lado es potencia de 2, se puede partir la matriz en cuatro cuadrados en cada paso.

Ejercicio 4

Solución

```
function y=sumaMatriz(M)
    [m,n]=size(M);
    if m==0
        y=0;
    elseif m==1
        y=M(1,1);
    else
        y=sumaMatriz(M(1:m/2,1:n/2)) +
            sumaMatriz(M(1:m/2,n/2+1:n)) +
            sumaMatriz(M(m/2+1:m,1:n/2)) +
            sumaMatriz(M(m/2+1:m,n/2+1:n));
    end
```