Computación 1
Curso 2022

Ingeniería Forestal

Universidad de la República

Introducción

Definiciones

- Recurrir
  - Volver una cosa al sitio de donde partió, retornar, repetirse.
  - Recurrir a algo -> hacer uso de ello.

Introducción

Definiciones

- Técnica algorítmica donde el algoritmo se llama a sí mismo para realizar una tarea.
- Un algoritmo es recursivo si se define en términos de sí mismo.

Introducción
Definiciones

- Matemáticas
  - El concepto de recursión es una herramienta básica
  - Principio de Inducción Completa
  - Definición de Conjuntos:
    - Números naturales:
      - 1 es un número natural
      - El siguiente de un número natural es un número natural
  - Definición de Funciones
    - La función factorial, n!
      - 0! = 1
      - Si n > 0 entonces n! = n * (n - 1)!

Introducción

Definiciones

- Programación
  - Técnica utilizada en lugar de la iteración cuando:
    - Solución compleja utilizando iteración
    - Solución poco clara al utilizar iteración
  - Problemas cuya solución se puede hallar resolviendo el mismo problema pero con un caso de menor tamaño.

Introducción
Definiciones

- Programa Directamente Recursivo
  - Se llama a sí mismo dentro de su cuerpo de sentencias
- Programa Indirectamente Recursivo
  - En su cuerpo de sentencias posee una invocación a otro programa que lo invoca nuevamente

Programas Recursivos
Definición

- Definidos en términos de si mismos.
  - Se invocan(llaman) a si mismos.
    - Caso Recursivo o General
  - Existe un caso (o varios) de menor tamaño que puede resolverse directamente sin necesidad de recurrencia.
    - Caso Base o Trivial

Programas Recursivos
Metodología

- Definición clara del problema
- Resolución de los casos simples (casos base).
  - Encontrar y resolver los casos de resolución simple, sin necesidad de recurrencia.
  - Condición de Parada o Salida
- Resolución del caso general en términos de casos más pequeños.
  - Llamada recursiva.

Programas Recursivos
Metodología

- Combinar las soluciones de los distintos casos conformando la solución al problema.
  - Utilizar estructuras de selección.

Programas Recursivos
Metodología

- Definición
  - **`factorial(0) = 1`**
  - **`factorial(n) = n*factorial(n–1)`**
- Caso Base
  - Si n = 0 entonces factorial(n) =1
  - No es necesario recurrir
- Caso Recursivo o General
  - Si n > 0 entonces factorial(n) = n*factorial(n-1)
  - Se recurre al factorial del natural anterior

Programas Recursivos

Metodología

```
function fn=factorial(n)
if n==0
    fn=1;
else
    fn=n*factorial(n-1);
end
```

Caso Base o Condición de Salida

Caso Recursivo

Caso más pequeño

Selección

Programas Recursivos

Errores comunes

- Ausencia del caso base.
  - Caso base = Condición de Parada.
  - Si falta, nunca termina la ejecución.
  - Utilizar estructuras de selección para garantizar la ejecución del caso base.
- Error en la llamada recursiva
  - Cada llamada recursiva se realiza con un valor de parámetro que hace el problema “de menor tamaño”.

Programas Recursivos

Errores comunes

- Utilizar estructuras iterativas en lugar de estructuras selectivas.
  - La llamada recursiva se realiza en una sentencia de selección.

Programas Recursivos
Ejecución

- Cálculo de factorial(3)

factorial(3) = 3 * factorial(2)

factorial(2) = 2 * factorial(1)

factorial(1) = 1* factorial(0)

factorial(0) =1 Caso Base

factorial(1) = 1 * 1

factorial(2) = 2*1

factorial(3) = 3*2

Programas Recursivos

Utilización de Memoria

- Cada paso de la recursión ejecuta una nueva versión del programa,
- Nueva asignación de espacio de memoria en cada paso.

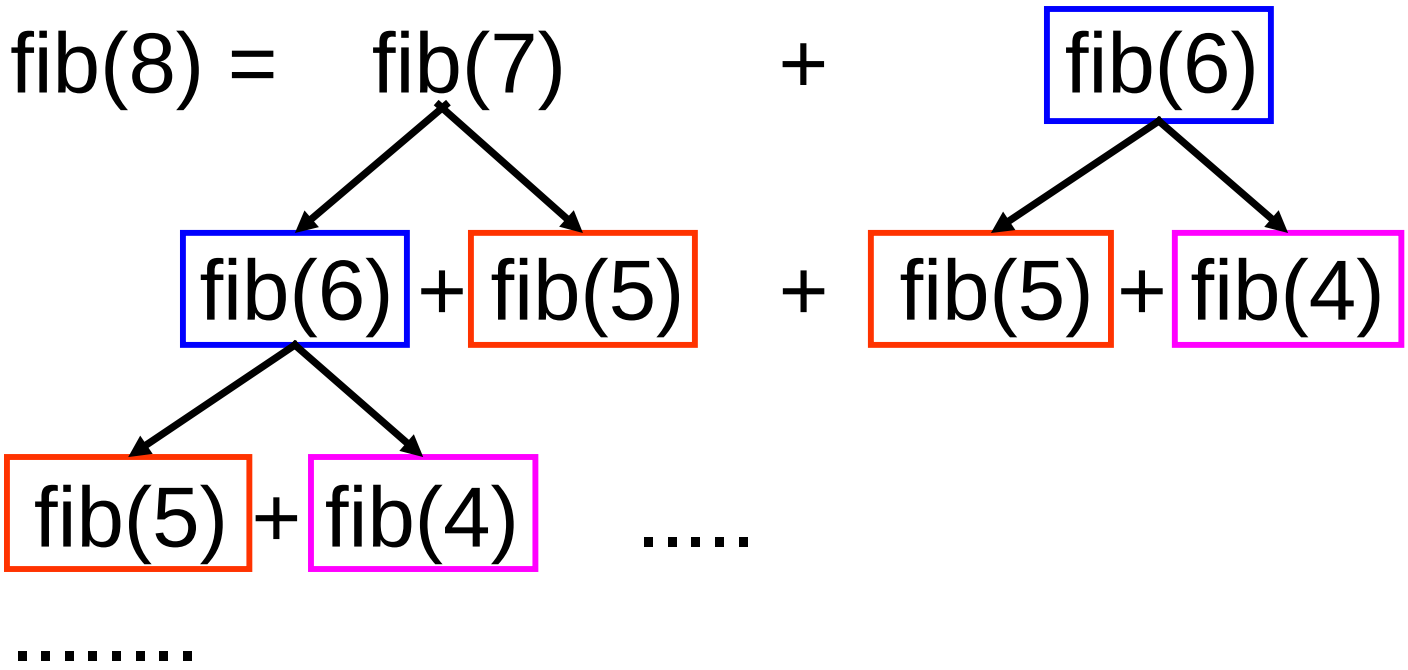
Programas Recursivos
Redundancia

- Función Fibonacci
  - **Fib(1) = 1**
  - **Fib(2) = 2**
  - **Fib(n) = Fib(n-1) + Fib(n-2)**
- Programa Recursivo

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

Programas Recursivos

Redundancia



- Redundancia en cálculos

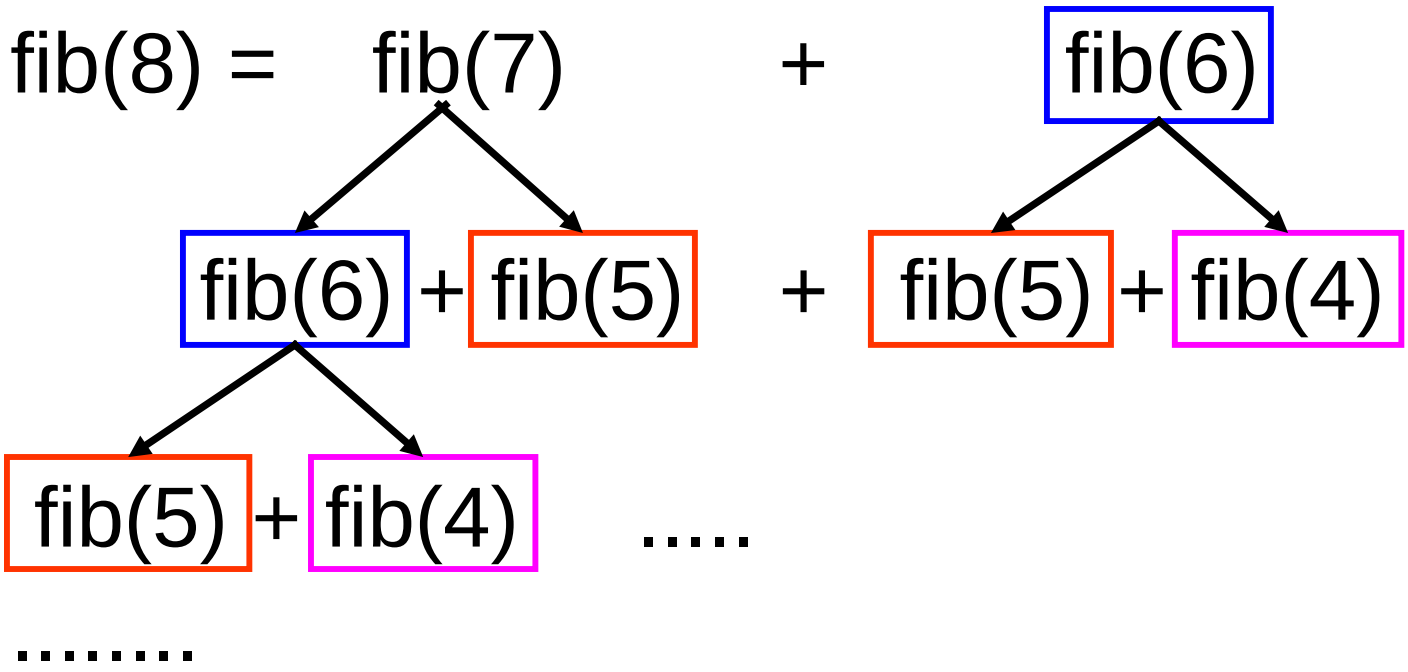
Programas Recursivos
Redundancia

- Función Fibonacci
  - **Fib(1) = 1**
  - **Fib(2) = 2**
  - **Fib(n) = Fib(n-1) + Fib(n-2)**
- Programa Recursivo

```
function fn=fib(n)
if n == 1
    fn = 1;
elseif n == 2
    fn = 2;
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.

Recursión vs Iteración

- Recursiva

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

Recursión vs Iteración

- Iterativa

```
function fn=fib(n)
if n==1
    fn=1;
elseif n==2
    fn=2;
else
    penult=1;
    ult=2;
    for i=3:n
        nue = penult + ult;
        penult = ult;
        ult = nu;
    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.

Programas Recursivos

Ejemplo

```
function fn=Pertenece(n,a)
if isempty(a)
    fn=0;
else
    if a(1)==n
        fn=1;
    else
        fn=Pertenece(n,a(2:length(a)));
    end
end
```

Casos Base

Vector más Pequeño

Caso Recursivo

Programas Recursivos
Ejemplo

- Búsqueda recursiva en un array
  - 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

Ejercicio - Solución

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