TD3: Algoritmos y Estructuras de Datos

Prof. Agustín Garassino, Gervasio Pérez

Segundo Semestre de 2024

Clase Teórica 5
Recursión – Divide & Conquer

Resumen

En la clase de hoy veremos:

- Noción de recursión
- Arbol de recursión
- Patrón de Dividir y Conquistar

Recursión

Para escribir un algoritmo tenemos dos mecanismos básicos para realizar tareas repetitivas: iteración y recursión.

Iteración: ciclos `while` y `for`

- Código: Guarda y Cuerpo del ciclo
- Corrección: Invariante, P_c, Q_c
- Terminación: función variante

```
1  int sumatoria(vector<int> v) {
2      int res = 0, i = 0;                    // Pc: res = 0; i = 0
3      while (i<v.size()) {
4          // I: 0 <= i <= |v| ^ res = sumatoria(j=0,i-1) v[j]
5          res = res + v[i];
6          i = i + 1;                                  // fv: |v| - i
7      } // Qc: res = sumatoria(j=0,|v|-1) v[j]
8      return res;
9  }
```

Recursión

La función se llama a sí misma

- Estructura: Caso(s) base y recursivo(s)
  - **Casos base**: subproblemas resueltos sin hacer recursión
  - **Casos recursivos**: subproblemas resueltos con recursión pero disminuyendo de alguna manera el tamaño del problema
- Corrección: se prueba por inducción matemática, donde los casos base del algoritmo son los casos base de la demostración y los casos recursivos se prueban asumiendo como hipótesis inductiva que las llamadas recursivas devuelven resultados correctos.
- Terminación: se muestra que siempre se llega a un caso base

Sumatoria recursiva (1)

```
1  int sumatoria_rec(vector<int> v, int pos_desde = 0) {
2      if (desde >= v.size())  // Caso base
3          return 0;
4      else { // Llamado recursivo
5          int suma = sumatoria_rec(v, pos_desde + 1);
6          return v[pos_desde] + suma;
7      }
8  }
```

$Post : res = \sum_{i=pos\_desde}^{|v|-1} v[i]$

- Luego de la línea 5 sabemos que $suma = \sum_{i=pos\_desde+1}^{|v|-1} v[i]$.
- Sumarle **v[pos_desde]** resuelve la llamada actual.
- Siempre se incrementa **pos_desde** al hacer recursión

Sumatoria recursiva (2)

Puede haber múltiples casos base y múltiples llamados recursivos

```
1   int suma_rec2(vector<int> v, int desde, int hasta) {
2       if (hasta <= desde)  // Caso base 1: rango vacío
3           return 0;
4       else if (desde == hasta-1) // Caso base 2: 1 elemento
5           return v[desde];
6       else  { // 2 o más elementos: recursión
7           int medio = (desde+hasta)/2;
8           return suma_rec2(v,desde,medio)
9                + suma_rec2(v,medio,hasta);
10      }
11  }
```

$Post : res = \sum_{i=desde}^{hasta-1} v[i]$

La diferencia entre **desde** y **hasta** siempre disminuye en cada llamado recursivo.

Recursión: consejos

Metodología recomendada para escribir una función recursiva.

1. Escribir encabezado de la función (nombre y tipos de los parámetros)
2. Saber con claridad cuál va a ser la poscondición de la función (si les ayuda pueden escribirla).
3. Identificar y escribir el/los casos bases: ¿Con qué valores de entrada se puede dar una respuesta sin recursión?
4. Casos recursivos:
   - Identificar cómo "achicar" los valores de entrada para hacer el llamado recursivo.
   - Escribir el llamado recursivo pasando la instancia "achicada" como parámetro.
   - Asumir que luego del llamado recursivo vale la poscondición para la instancia "achicada".
   - Usar el resultado de la recursión para calcular el valor final [1] a devolver para cumplir la poscondición del problema completo.

[1] Si se trata de una función **`void`** no se debe devolver un valor final sino realizar el cómputo que falte para cumplir la post.

Recursión: ejemplos conocidos

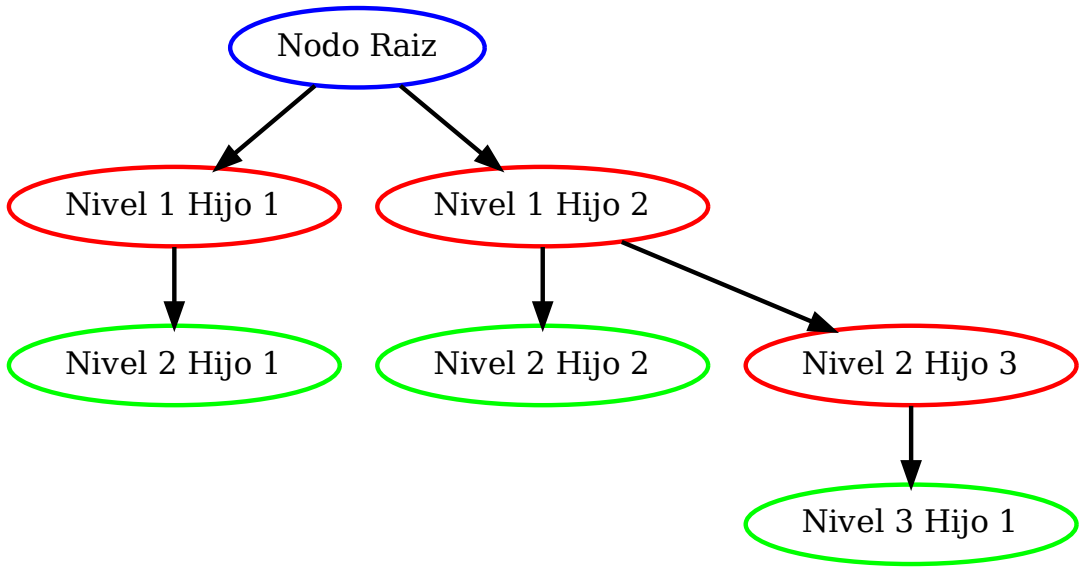
```
1   int factorial (int n) {
2       if (n <= 1) // Caso base
3           return 1;
4       else // Caso recursivo
5           return n * factorial(n-1);
6   }
```

```
1   int fib (int n) {
2       if (n == 0 || n == 1) // Casos base
3           return 1;
4       else // Caso recursivo
5           return fib(n-1) + fib(n-2);
6   }
```

Árboles en Computación

Un árbol representa información con relaciones estructurales

- **Raiz:** Nodo inicial
- **Nodo interno**: Con nodo padre y al menos un nodo hijo
- **Hoja**: Nodo terminal (sin hijos)
- **Altura:** Cantidad de niveles −1

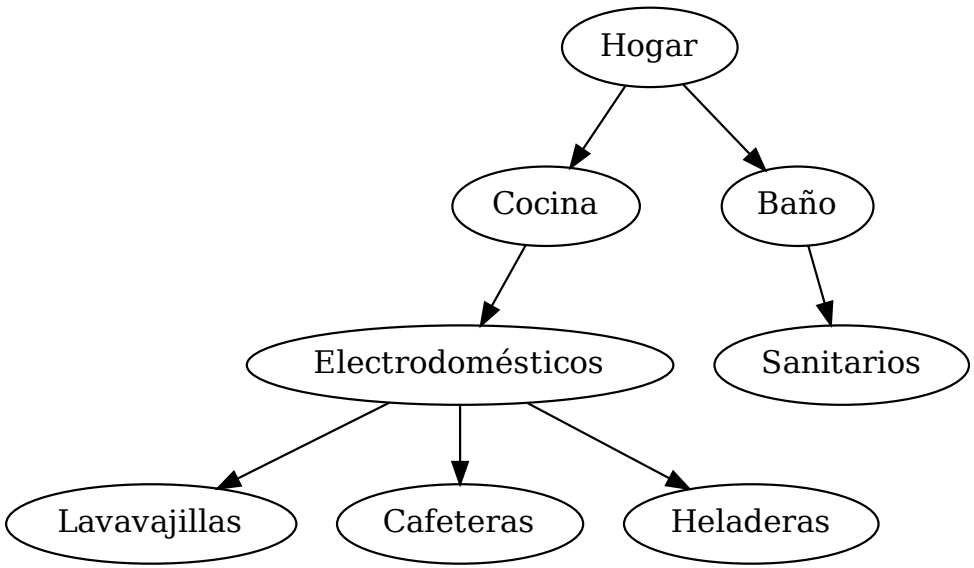


Un árbol de altura 3

Árboles en la Computación

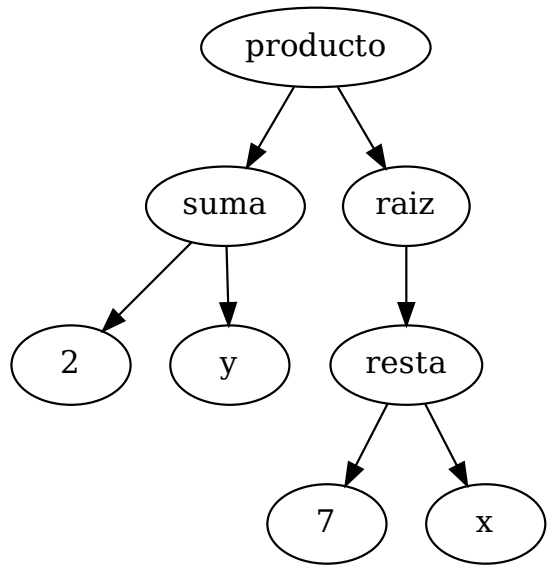
En Ciencias de la Computación el concepto de árbol es muy utilizado para representar muchas cosas distintas

Categorías de productos



Expresiones aritméticas

$(2+y) \times \sqrt{7-x}$



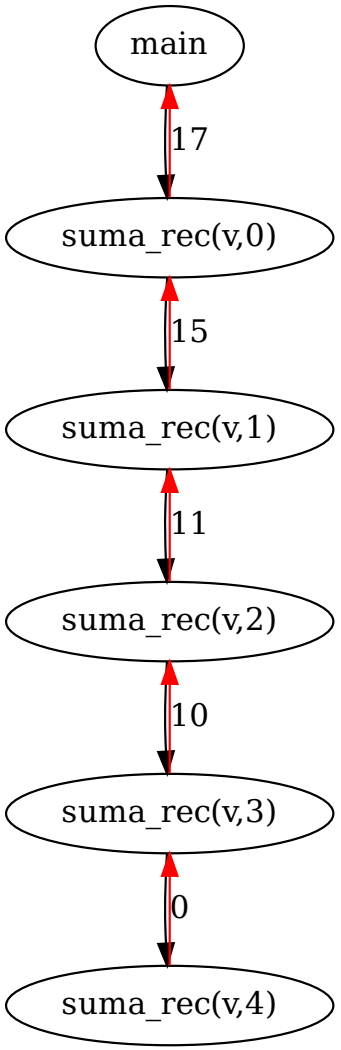
Visualizando la recursión: Arbol de recursión

Diagrama de ejecución de las diversas llamadas recursivas de un programa

- Nodo: llamada a la función
- ⟶ **nuevas llamadas generadas**
- ⟶ valores devueltos
- **Altura:** cantidad de niveles del arbol igual a la "profundidad" de la recursión

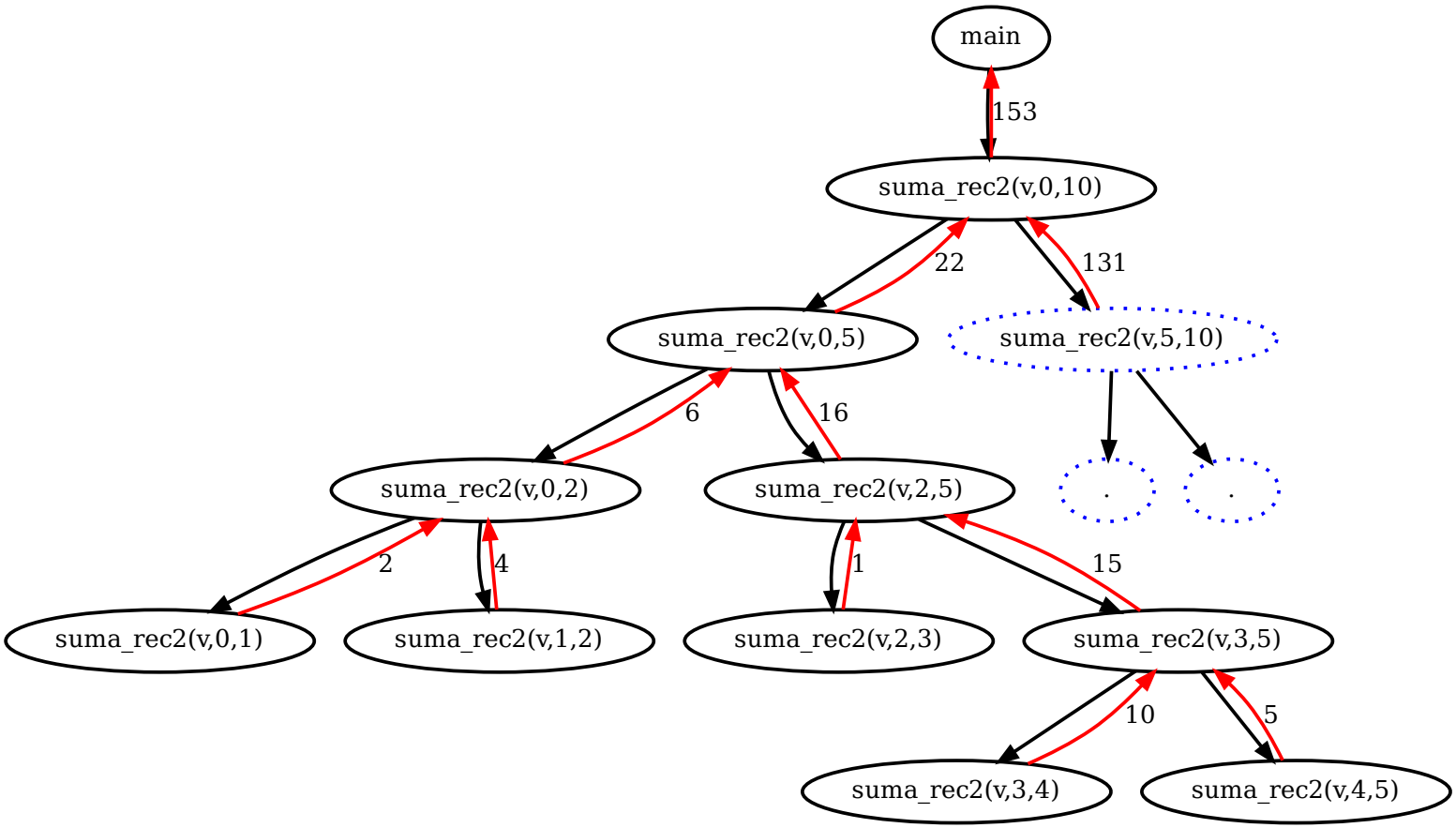
```
1  int suma_rec(vector<int> v, int pos_desde = 0)
2  {
3    if (desde >= v.size())
4        return 0;
5    else
6        int suma = suma_rec(v,pos_desde + 1);
7        return v[pos_desde] + suma;
8  }
```

v = {2,4,1,10}



Recursión de **suma_rec2**

v = {2,4,1,10,5,3,11,9,8,100}



Esquema recursivo Divide & Conquer

Divide & Conquer es una técnica de diseño de algoritmos que estructura la solución de un problema de tamaño N así:

1. Caso base: Si el problema es suficientemente pequeño, resolverlo directamente; si no...
2. Divide: Partir el problema a resolver en a subproblemas de tamaño N/b
3. Conquer: Resolver los subproblemas (en general recursivamente)
4. Combine: Combinar los resultados de los subproblemas para resolver el problema original

Esquema recursivo Divide & Conquer

```
1   int suma_rec2(vector<int> v, int desde, int hasta) {
2     if (hasta <= desde)
3     // Caso base 1: rango vacío
4       return 0;
5     else if (desde == hasta-1)
6     // Caso base 2: 1 elemento
7       return v[desde];
8     else  { // 2 o más elementos
9       // "Divide"
10      int medio = (desde+hasta)/2;
11      // "Conquer": Resuelvo 2 subproblemas recursivamente
12      int a = suma_rec2(v,desde,medio);
13      int b = suma_rec2(v,medio,hasta);
14      // "Combine": combino resultados
15      return a + b;
16    }
17  }
```

Ejercicios de recursión

Para cada uno usar el patrón de Dividir y Conquistar, señalando en el algoritmo cada una de las etapas .

- Hallar el mínimo de un vector **no vacío**
- Buscar un elemento en un vector **ordenado**
- Multiplicar por k a cada elemento de un vector.
- Multiplicar por k a cada elemento de un vector y al mismo tiempo devolver su productoria (del vector original).
- Hallar la *subsecuencia de suma máxima* de un vector, es decir, el subvector que maximiza el resultado de la sumatoria.

Repaso de la clase

Hoy vimos

- Repaso de recursión
- Árbol de recursión
- Técnica de Divide & Conquer.

Guia 4

- Pueden hacer la primera sección: **Recursión**.

Bibliografía:

- Divide & Conquer
  - **"Introduction to Algorithms, Fourth Edition" por Thomas Cormen**, capítulo 2.4.