Diseño II: Listas y árboles binarios

Román Gorojovsky

Algoritmos y Estructuras de Datos

22 de mayo de 2024

Plan del día

Plan del día

- Ejercicios con listas enlazadas
- Recursión
- Ejercicios con Árboles binarios
- Fantasmas del pasado (si sobra tiempo)
- Devolución parciales (aprox. 20:00hs)

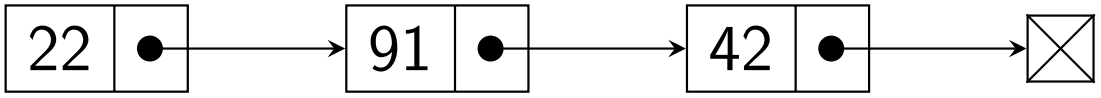
Listas Enlazadas

Introducción

- Familia de estructuras donde cada dato se guarda en un espacio "dedicado" de memoria dinámica
- Tamaño arbitrario
- Se agregan elementos en tiempo constante
- Acceso lineal a los elementos

Listas Enlazadas

Introducción



Lista simplemente enlazada

```
NodoLista<T> es struct<
    val: T,
    siguiente: NodoLista<T>,
  >
Modulo ListaEnlazada<T> implementa Secuencia<T> {
    var primero: NodoLista<T> // "puntero" al primer elemento
    var último: NodoLista<T> // "puntero" al último elemento
    var longitud: int // cantidad total de elementos
    ...
}
```

Listas enlazadas

Ejemplo de algoritmo

```
longitud(in l: ListaEnlazada<T>) : int
```

```
1  if(l.primero == null)
2    return 0
3  else
4    var ret = 1
5    var actual = l.primero
6
7    while (actual.siguiente != null)
8        actual = actual.siguiente
9        ret = ret + 1
10   endwhile
11 endif
12
13 return ret
```

Listas enlazadas

Ejercicios

- `pertenece(in l: ListaEnlazada<T>, in t: T) : bool`
- `obtener(in l: ListaEnlazada<T>, in i: int) : T`
- ```
  concatenar(
      inout l1: ListaEnlazada<T>,
      in l2: ListaEnlazada<T>
  )
  ```
  ```

Recursión: Introducción/Repaso
(tachar lo que no corresponda)

- Funciones que se llaman a si mismas (directa o indirectamente)
- Algunos problemas resulta más fácil expresarlos recursivamente
- Cada paso recursivo resuelve el mismo problema sobre una "porción" menor de la entrada
- Estructuralmente similar a Inducción
- Hay que tener cuidado para asegurar que terminen

- **Caso Base:** Paso no-recursivo que devuelve algún valor (según el problema)
- **Paso Recursivo:** Llamado a la propia función. Tiene que "acercarse" al caso base
- Puede haber más de un caso base y pasos recursivos

Recursión

Ejemplo

fibonacci(in n: int)

```
1 if (n <= 1)
2     return 1
3 else
4     return fibonacci(n - 1) + fibonacci(n - 2)
5 endif
```

- **Caso Base:** `return 1`
- **Paso Recursivo:**
```
return fibonacci(n - 1) +
       fibonacci(n - 2)
```

Recursión

Ejercicio

factorial(in n: int)

Recursión

Ejercicio

factorial(in n: int)

```
1 if (n == 1)
2     return 1
3 else
4     return n * factorial(n - 1)
5 endif
```

Recursión

Ejercicio

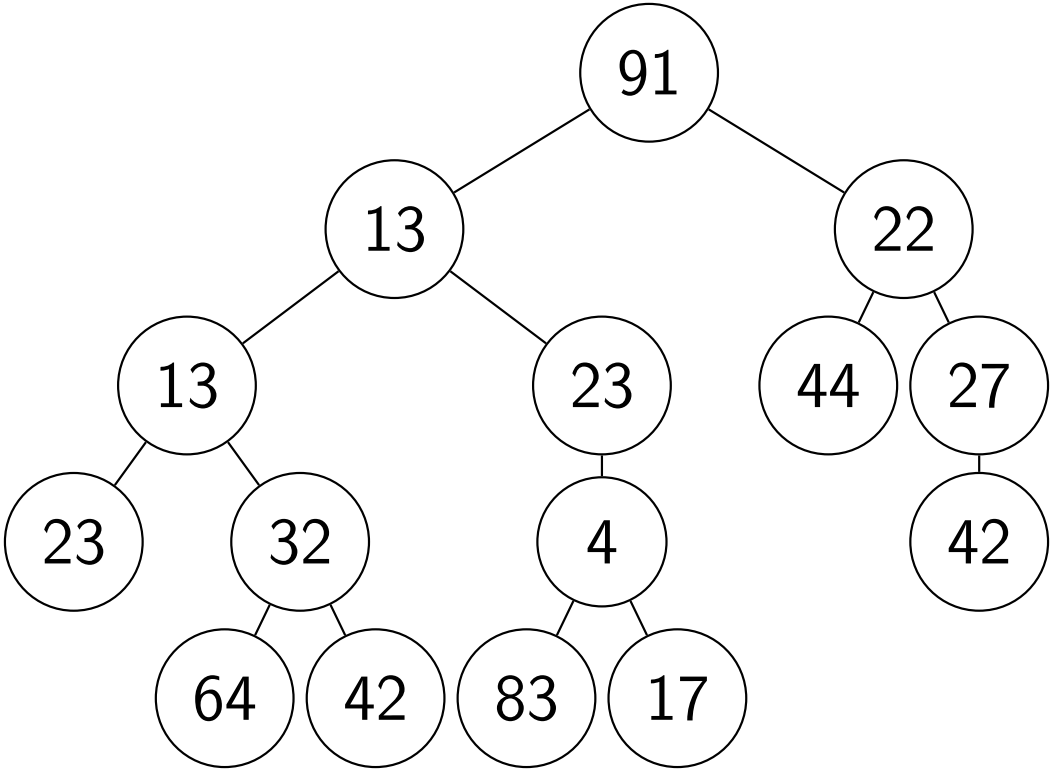
factorial(in n: int)

```
1 if(n == 1)
2     return 1
3 else
4     return n * factorial(n - 1)
5 endif
```

- **Caso Base:** `return 1`
- **Paso Recursivo:** `return n * factorial(n - 1)`

Árboles Binarios

Introducción



Árboles Binarios

Introducción

Árbol Binario

```
NodoAB<T> es struct<
    val: T,
    izquierda: NodoAB<T>,
    derecha: NodoAB<T>,
  >
Modulo ArbolBinario<T> implementa ArbolBinario<T> {
    var raíz: NodoAB<T> // "puntero" a la raíz del árbol
    esVacío(in ab: ArbolBinario<T>): bool
    ...
}
```

esVacío(in ab: ArbolBinario<T>): bool

```
return ab.raíz == null
```

Árboles Binarios

Introducción

¿Cómo recorremos un árbol binario?

Árboles Binarios

Introducción

¿Cómo recorremos un árbol binario?

¡Recursivamente!

- **Caso Base:** $esVacío(ab) = true$
- **Paso Recursivo:** $esVacío(ab) = false$

Árboles Binarios

Introducción

¿Cómo recorremos un árbol binario?

¡Recursivamente!

- **Caso Base:** $esVacío(ab) = true$
- **Paso Recursivo:** $esVacío(ab) = false$

```
1 cantidadDeNodos(in ab: ArbolBinario<T>): int
2     return cantidadDeNodos(ab.raiz)
3
4 cantidadDeNodos(in n: NodoAB<T>): int
5 if(n == null)
6     return 0
7 else
8     return 1 + cantidadDeNodos(n.izquierda) + cantidadDeNodos(n.derecha)
9 endif
```

Árboles Binarios

Ejercicios

- `altura(in ab: ArbolBinario<T>): int`
- `está(in ab: ArbolBinario<T>, int t: T): bool`

¿Qué hora es?

¿Qué hora es?
¿Quieren más ejercicios?

Rosetree

Árbol con una cantidad arbitraria de hojas por nodo

Lista simplemente enlazada

```
NodoRosetree<T> es struct<
    val: T,
    hijos: Array<NodoRosetree<T>>,
  >
Modulo Rosetree<T> {
    var raíz: NodoRosetree<T> // "puntero" a la raíz del árbol
    ...
}
```

Ejercicios

- `altura(in rt: Rosetree<T>): int`
- `está(in rt: Rosetree<T>, in t: T): bool`