Clase 10 - Elección de estructuras 2

Algoritmos y Estructuras de Datos

1er Cuatrimestre 2024

Disclaimer: Este pdf tiene "anotaciones" y una solución propuesto en Clase (práctica turno mañana) por lo que podrían haber cosas que no se entiendan o incompletas si no participaron presencialmente de la clase

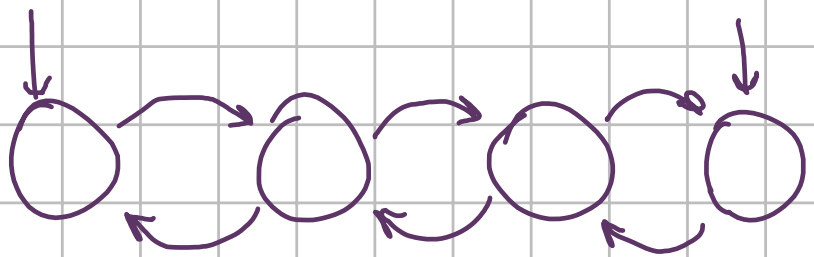
Plan del día

- Presentación del nuevo apunte **Módulos básicos**
- Ejercicio para recorrer un árbol
- Repaso de Complejidades
- Ejercicio tipo parcial

Conjunto Lineal

y

Diccionario Lineal

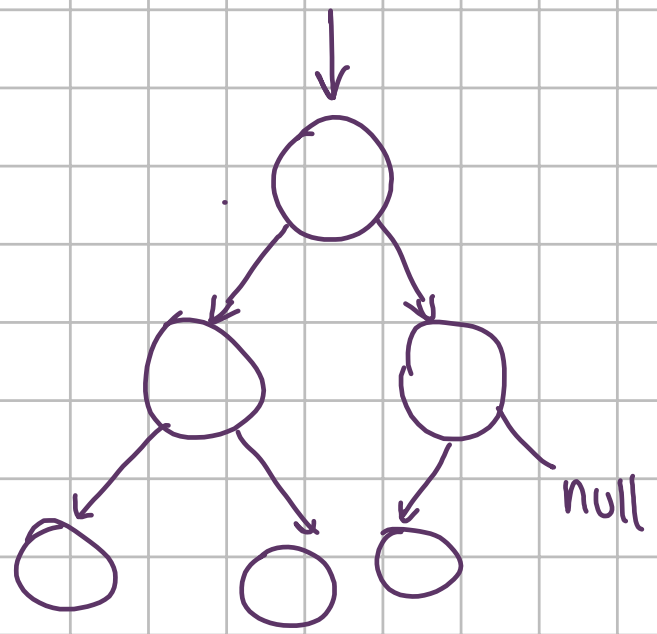


Lista Enlazada

Conjunto Log

y

Diccionario Log

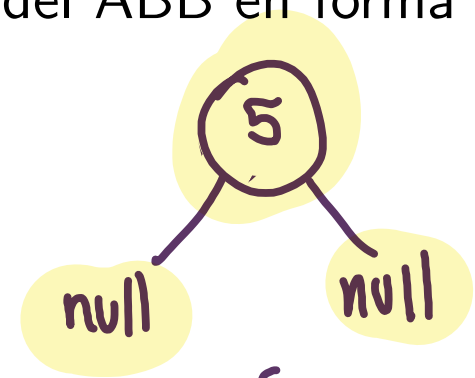


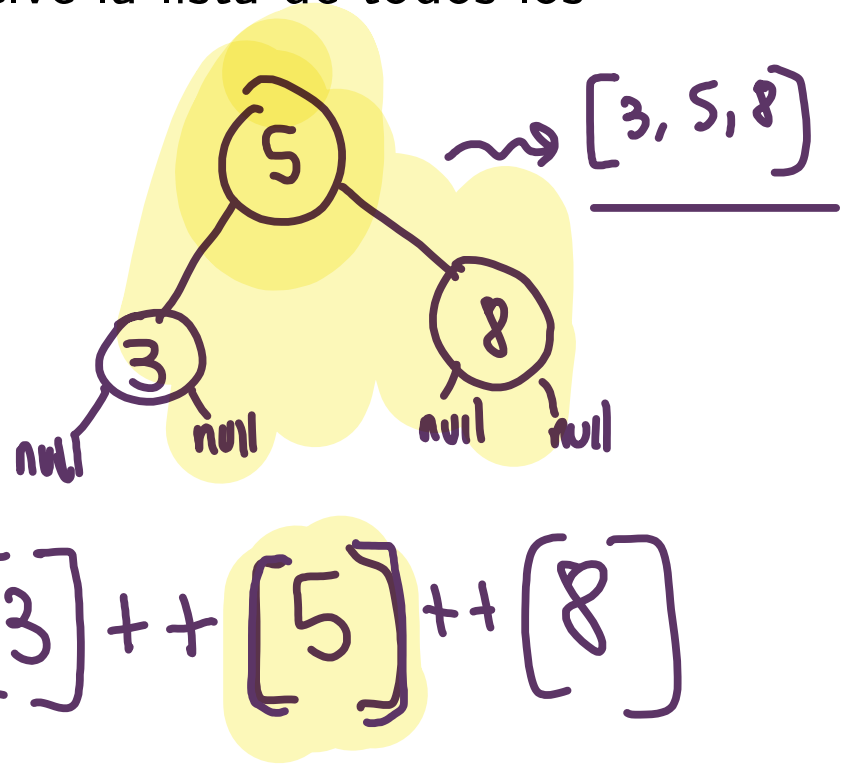
AVL

apunte módulos básicos

Inorder

Inorder: Dado un ABB, devuelve la lista de todos los elementos del ABB en forma ordenada.





```
proc inOrderNodo(in n: Nodo<T>, inout res: Secuencia<T>)
    if n != Null {
        inorderNodo(n.izquierda, res)
        res.AgregarAtras(n.dato)
        inorderNodo(n.derecho, res)
    }
```

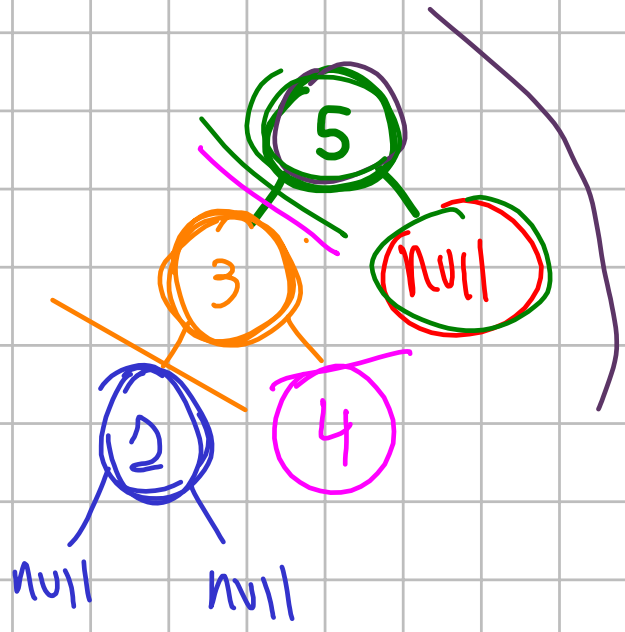
y

```
proc inorder(in a: ABB<T>): Secuencia<T>
    res := SecuenciaVacia()
    inorderNodo(a.raiz, res)
    return res
```

```
proc inOrderNodo (in n: Nodo<T>, inout res: res[T]):
  if n != Null then
    inorderNodo(n.izquierda, res)
    res.AgregarAtras(n.dato)
    inorderNodo(n.derecho, res)
```

y

```
proc inorder(in a: ABB<T>): Sec[T]
  res := SecuenciaVacia()
  inorderNodo(a.raiz, res)
  return res
```



Estructuras que hasta hoy vimos

Confeccionemos una tabla comparativa de las complejidades de peor caso de las operaciones de pertenencia, inserción, borrado, búsqueda del mínimo y borrado del mínimo para conjuntos de naturales sobre las siguientes estructuras:

- Lista enlazada
- Lista enlazada ordenada
- Árbol binarios de búsqueda (ABB)
- ABB balanceado en altura (AVL)

Estructuras que hasta hoy vimos

	Pertenece	Insertar	Borrar	Buscar minimo	Borrar minimo
Lista Enlazada	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Lista Ordenada	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$
ABB	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
AVL	$\log(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$

AVL

$\log(n)+k$

insertar AVL: Buscar dónde insertar + rotaciones para balancear

$O(\log n)$ $\quad$ $O(\log n)$

Ejercicio de Parcial

Quiero hacer un sistema para registrar las notas de los alumnes en una facultad.

- Les alumnes se identifican con LU.
- Las materias tienen un nombre. Puede haber una cantidad no acotada de materias.
- Las notas están entre 0 y 10. Aprueban si la nota es mayor 7.

Con las siguientes restricciones, siendo **m** la cantidad de materias y **n** la cantidad de alumnes:

- RegistrarMateria en O(log m)
- RegistrarNota en O(log n + log m)
- NotaDeAlumno en O(log n + log m)
- CantAlumnosConNota y CantAlumnosAprobados en O(log m)

Ejercicio de Parcial

```
TAD Sistema {
      obs ...
      proc NuevoDc() y acá declaramos dm y cm vacías
      proc RegistrarMateria(inout s: Sistema, in m:
materia) O(log (m))
      proc RegistrarNota(inout s: Sistema, in m: materia,
in a: alumno, in n: nota) O(logn + log m)
      proc NotaDeAlumno(in s: Sistema, in a: alumno, m:
materia): nota O(log n + log m)
      proc CantAlumnosConNota(in s: Sistema, in m: materia,
n: nota): nat O(log m)
      proc CantAlumnosAprobados(in s: Sistema, in m:
materia): nat O(log m)
}
```

```
    proc RegistrarMateria(inout s: Sistema, in m:
materia) O(log (m))
    proc RegistrarNota(inout s: Sistema, in m: materia,
in a: alumno, in n: nota) O(log n + log m)
    proc NotaDeAlumno(in s: Sistema, in a: alumno, m:
materia): nota O(log n + log m)
    proc CantAlumnosConNota(in s: Sistema, in m: materia,
n: nota): nat O(log m)
    proc CantAlumnosAprobados(in s: Sistema, in m:
materia): nat O(log m)
}
```

materia, alumno } son ordenables

1er idea

llamamos da

var dm: dicc log < materia, dicc log < alumno, nota: int > >

veamos "a ojo" que funciona para cada proc

- RegistrarMateria en O(log m)
- RegistrarNota en O(log n + log m)
- NotaDeAlumno en O(log n + log m)
- CantAlumnosConNota y CantAlumnosAprobados en O(log m)

llamamos da

var dm: dicc Log < materia , dicc Log < alumno, nota: int > >

Registrar Materia: insertar m: materia en dm: diccionarioLog ⇝ O(log m) ✓

Registrar Nota:

buscar materia + { buscar (si a: alumno esta) en da: diaLog
                 { insertar (si a no está)

⇝ O(log m) + O(log n) ✓

Nota De Alumno:

busar m en dm + busar a en da ⇝ log(m) + log(n) ✓

Cant Alumnos Con Nota

busaor m en dm + para cada a ver la nota (recorrer todo da)

O(log m) + O(n) = max{O(log m), O(n)} = O(n) ✗

asumiendo que hay más alumnos que materias (m ≤ n)

Vemos que con solo dicc no alcanza, proponemos también:

var cm : diccLog < materias , notas: array [11] >

en cada pos tengo la cant de alumnes con esa nota

cantAlumnosConNota: buscar m materia + ver una pos de un array
$\rightsquigarrow$ $O(\log m) + O(1) = O(\log m)$ ✓

cantAlumnosAprobados buscar m materia + contar la pos 7, 8, 9 y 10 del array
$\rightsquigarrow$ $O(\log m) + O(1) = O(\log m)$ ✓

Veamos si las anteriores siguen OK:

- RegistrarMateria en O(log m) → insertar m materia en cm → $O(\log m) + O(\log m)$ (anterior) ✓
- RegistrarNota en O(log n + log m) → buscar m en cm + modificar nota
  $O(\log m) + O(1) + O(\log n + \log m)$ ✓
- NotaDeAlumno en O(log n + log m)
  ↳ no modifico ni uso cm

```
dm : diccLog < materia , diccLog < alumno , nota : int > >

var cm : diccLog < materias , notas : array [11] >

proc iniciarDC ()
    dm = new diccLog< materia, new diccLog< alumno, nota >>
    cm = new diccLog< materia , nota[] >

proc RegistrarMateria(inout s: Sistema, in m: materia)
    dm . definir ( m , new diccLog< alumno, nota > )
    cm . definir ( m, new nota[11] )

proc RegistrarNota (inout s: Sistema, in m: materia , in a: alumno, in n:nota):
    dm.obtener(m) . definir (a , n)
    cm . definir ( m, cm.obtener ( m ) [n] = +1 )

proc NotaDeAlumno( in s: Sistema , in m:materia , in a: alumno):nota
    nota res= dm . obtener (m) . obtener (a)
    return res
```

```
proc CantAlumosConNota (in s: Sistema, in m:materia, in n:nota) : int
    nota res = cm.obtener(m)[n]
    return res

proc CantAlumosAprobados (in s: Sistema, in m:materia) : int

    nota[] aux = cm.obtener(m)
    int res = aux[7] + aux[8] + aux[9] + aux[10]
    return res
```

Otra posible sol (turno tarde)

dicc Log < materia, array < conjLog <LU>>>