DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Guia 7

2do cuatrimestre 2024 Algoritmos y Estructuras de Datos I

Integrante	LU	Correo electrónico
Federico Barberón	112/24	`jfedericobarberonj@gmail.com`

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - Pabellón I
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Argentina
Tel/Fax: (54 11) 4576-3359
http://exactas.uba.ar

Índice

1. Guia 7

1.1. Ejercicio 1

Implementamos un **Árbol Binario** (AB)

- Escriba en castellano el invariante de representación para este módulo
- Escriba en lógica el invrep usando preds recursivos
- Escriba los algoritmos para los siguientes procs y, de ser posible, calcule su complejidad
    - altura(in ab: ArbolBinario<T>): int
    - cantidadHojas(in ab: ArbolBinario<T>): int
    - está(in ab: ArbolBinario<T>, in t: T): bool
    - cantidadApariciones(in ab: ArbolBinario<T>, in t: T): int

Nodo<T>es `struct`$\langle dato : T, izq : Nodo, der : Nodo\rangle$

```
Módulo ArbolBinario<T> implementa Arbol Binario<T> {
    var raiz: Nodo<T>

    InvRep: No tiene ciclos y la raiz es null o el subarbol derecho y el izquierdo son AB
    pred esAB (r: Nodo<T>) {
        $r = null \vee_L (esAB(r.der) \wedge esAB(r.izq))$
    }
    pred InvRep (ab: $ArbolBinario\langle T\rangle$ ) {
        $sinCiclos(ab.raiz) \wedge esAB(ab.raiz)$
    }
    proc altura (in ab: $ArbolBinario\langle T\rangle$ ) : int
          Complejidad: $O(n) \leftarrow n = cantNodos$

        return altura(ab.raiz);

    proc alturaAux (in r: Nodo<T>) : int

        if r == null:
            return 0;
        endif

        return 1 + max(alturaAux(r.izq), alturaAux(r.der));

    proc cantidadHojas (in ab: $ArbolBinario\langle T\rangle$ ) : int
          Complejidad: $O(n)$

        return cantidadHojasAux(ab.raiz);

    proc cantidadHojasAux (in r: Nodo<T>) : int

        if r == null
            return 0;
        else if r.izq == null && r.der == null
            return 1;
        endif

        return cantidadHojasAux(r.izq) + cantidadHojasAux(r.der);
```

```
proc esta (in ab: ArbolBinario⟨T⟩ ) : Bool
        Complejidad: $O(n)$

    return estaAux(r.raiz, e);

proc estaAux (in r: Nodo<T>, in e: T) : Bool

    if r == null
        return false;
    else if r.dato == e
        return true;
    else
        return estaAux(r.izq, e) || estaAux(r.der, e);
    endif

proc cantidadApariciones (in ab: ArbolBinario⟨T⟩ , in e: T) : int
        Complejidad: $O(n)$

    return cantidadAparicionesAux(ab.raiz, e);

proc cantidadAparicionesAux (in r: Nodo<T>, in e: T) : int

    var cant: int;
    cant := 0;

    if r == null
        return 0;
    else if r.dato == e:
        cant := 1;
    endif

    return cant + cantidadAparicionesAux(r.izq, e) + cantidadAparicionesAux(r.der, e);

}
```

1.2. Ejercicio 2

Un **Árbol Binario de Búsqueda** (ABB) es un árbol binario que cumple que para cualquier nodo N, todos los elementos del árbol a la izquierda son menores o iguales al valor del nodo y todos los elementos del árbol a la derecha son mayores al valor del nodo, es decir

`pred esABB` (a: Nodo<T>) {
$a = null \lor ($
$(\forall e : T)\ (e \in elems(a.izq) \rightarrow e \leq a.dato) \land (\forall e : T)\ (e \in elems(a.der) \rightarrow e > a.dato) \land$
$esABB(a.izq) \land esABB(a.der)$
$)$
}

`aux elems` (a: Nodo<T>) : conj$\langle T \rangle$ = `IfThenElse`$(a = null,\ \emptyset,\ \{a.dato\} \cup elems(a.izq) \cup elems(a.der))$ **;**

- Implemene los algoritmos para los siguientes procs y calcule su complejidad en mejor y peor caso
    1. está(in ab: ABB<T>, in t: T): bool
    2. cantidadApariciones(in ab: ABB<T>, in t: T): int
    3. insertar(inout ab: ABB<T>, in t: T)
    4. eliminar(inout ab: ABB<T>, in t: T)
    5. inOrder(in ab: ABB<T>): Array<T>
- Asumiendo que el árbol está balanceado, recalcule, si es necesario, las complejidads en peor caso de los algoritmos del ítem anterior

- ¿Qué pasa en un ABB cuando se insertan valores repetidos? Proponga una modificación del módulo que resuelva este problema

```
Módulo ABB<T> implementa Arbol Binario De Busqueda<T> {
    var raiz: Nodo<T>

    proc esta (in ab: $ABB\langle T\rangle$ , in t: T) : Bool
        Mejor caso: $\Theta(1)$
        Peor caso: $\Theta(n)$

        return estaAux(ab.raiz, t);

    proc estaAux (in r: Nodo<T>, in t: T) : Bool

        if r.dato == t
            return true;
        else if t > r.dato
            return estaAux(r.der, t);
        else
            return estaAux(r.izq, t);
        endif

    proc cantidadApariciones (in ab: $ABB\langle T\rangle$ , in t: T) : int
        Mejor caso: $\Theta(1)$
        Peor caso: $\Theta(n)$

        return cantidadAparicionesAux(ab.raiz, t);

    proc cantidadAparicionesAux (in r: Nodo<T>, in t: T) : int

        if r == null
            return 0;
        else if r.dato == t
            return 1 + cantidadAparicionesAux(r.izq, t);
        else if r.dato > t
            return cantidadAparicionesAux(r.der, t);
        else
            return cantidadAparicionesAux(r.izq, t);
        endif

    proc insertar (inout ab: $ABB\langle T\rangle$ , in t: T)
        Mejor caso: $\Theta(1)$
        Peor caso: $\Theta(n)$

        ab.raiz := insertarAux(ab.raiz, t);

    proc insertarAux (inout r: Nodo<T>, in t: T) : Nodo<T>

        if r == null
            r := new Nodo < T >;
            r.dato := t;
        else if t <= r.dato
            r.izq := insertarAux(r.izq, t);
        else
            r.der := insertarAux(r.der, t);
        endif

        return r;
```

```
proc eliminar (inout ab: $ABB\langle T\rangle$ , in t: T)
    Mejor caso: $\Theta(1)$
    Peor caso: $\Theta(n)$

  ab.raiz := eliminarAux(ab.raiz, t);
```

```
proc eliminarAux (inout r: Nodo<T>, in t: T) : Nodo<T>

  if r == null
      return null
  else if r.dato == t
      if r.izq != null && r.der != null
          r.dato = minimo(r.der);
          r.der = eliminarAux(r.der,r.dato);
      else if r.izq != null
          return r.izq;
      else
          return r.der;
      endif
  else if t > r.dato
      r.der := eliminarAux(r.der, t);
  else
      r.izq := eliminarAux(r.izq, t);
  endif

  return r;
```

```
proc minimo (in r: Nodo<T>) : T

  while r.izq != null do
      r := r.izq;
  endwhile

  return r.dato;
```

```
proc inOrder (in ab: $ABB\langle T\rangle$ ) : Array < T >
    Mejor caso: $\Theta(1)$
    Peor caso: $\Theta(n)$

  var cola: ColaSobreLista < T >;
  cola := colaVacia();

  inOrderAux(cola, ab.raiz);

  return colaAArray(cola, cantidadNodos(ab.raiz));
```

```
proc inOrderAux (inout c: ColaSobreLista<T>,in r: Nodo<T>)

  if r == null
      return;
  endif

  inOrderAux(c, r.izq);
  c.encolar(r.dato);
  inOrderAux(c, r.der);
```

```
proc colaAArray (inout c: ColaSobreLista<T>, in size: int) : Array < T >
    var res: Array < T >;
    var i: int;
    res := new Array < T >(size);
    i := 0;

    while !c.vacia() do
        res[i] := c.desencolar();
        i := i + 1;
    endwhile

    return res;

proc cantidadNodos (in r: Nodo<T>) : int
    if r == null
        return 0;
    else
        return 1 + cantidadNodos(r.der) + cantidadNodos(r.izq);
    endif

}
```

Si el arbol está balanceado entonces la complejidad en el peor caso de los algoritmos está, insertar y eliminar pasa a ser $\Theta(\log n)$

Si se insertar valores repetidos, según el enunciado, estos se insertaran en el subarbol izquierdo del nodo con ese mismo valor (no se cual sería el problema)