1

1.1

El invrep debe asegurar que para todos los elementos del arbol haya algun camino que los conecte con la raiz, excepto que sean la raiz en si.

1.2

`pred` invRep (ab:$ArbolBinario < T >$){$(\forall e : T)(e \in ab \wedge ab.raiz \neq null \leftrightarrow elemInAB(e, ab.raiz))$}

`pred` elemInAB (e: T,raiz: $Nodo < T >$){$raiz \neq \wedge_L (e = raiz.dato \vee (raiz.izq \neq \texttt{null} \wedge_L elemInAB(e, raiz.izq)) \vee (raiz.der \neq \texttt{null} \wedge_L elemInAB(e, raiz.der)))$}

1.3

`modulo` $ArbolBinario < T >$ `implementa` $ArbolBinario < T >$ {

```
proc alturaRama (in raiz: Nodo<T>): int {
    if (raiz!=null) then
        altura := 1
        altIzq := 0
        altDer := 0
        if (raiz.izq!=null) then
            altIzq := alturaRama(raiz.izq)
        endif
        if (raiz.der!=null) then
            altDer := alturaRama(raiz.der)
        endif
        if (altIzq>altDer) then
            res := altura+altIzq
        else
            res := altura+altDer
    else
        res := 0
    endif
    return res
}

proc hojasRama (in raiz: Nodo<T>): int {
    res := 0
    if (raiz=null) then
        res++
    if (raiz.izq!=null) then
        res := res+hojasRama(raiz.izq)
    else
        res++
    endif
```

```
        if (raiz.der!=null) then
            res := res+hojasRama(raiz.der)
        else
            res++
        endif
    }

    proc altura (in ab:ArbolBinario<T>): int {
        return alturaRama(ab.raiz)
    }

    proc cantidadHojas (in ab:ArbolBinario<T>): int {
        return hojasRama(ab.raiz)
    }

}
```

$altura \in O(a)$ [altura del arbol]
$cantidadHojas \in O(a)$ [altura del arbol]

2 Hacer

3

3.1 Hacer

3.2

Supongo que las key son elementos comparables
$elem < K, V >$ es $struct < key : K, val : V >$

```
modulo dictABB<K,V> implementa Diccionario<K,V> {
        var data : ABB<elem<K,V>>
        var keys : ABB<K>

    proc diccionarioVacio (): dictABB {
        datos := new ABB<elem<K,V>>
        datos.raiz := null
        claves := new ABB<K>
        claves.raiz := null
        res := new dictABB<K,V>
        res.data := datos
        res.keys := claves
        return res
    }

    proc pertenece (in d: dictABB<K,V>, in k: K): bool {
        var res : bool
        res := esta(d.keys,k)
        return res
    }
```

```
    proc definir (inout d: dictABB < K, V >, in k: K,in v:V): {
        elemento := new elem < K, V >
        elemento.key := k
        elemento.val := v
        insertar(d.data,elemento)
        insertar(d.keys,k)
    }

}
```

- $diccionarioVacio \in O(1)$
- $esta \in O(n)$ [ABB]
- $esta \in O(log_2(n))$ [AVL (altura)]

4 HACER

5

```
    proc esMaxHeap (a:AB): Bool {
        return a ≥ a.izq & a ≥ a.der & esMaxHeap(a.izq) & esMaxHeap(a.der)
    }
```

6

6.1

pred invRep (cp:$colaprioridad\langle T\rangle$){$(\forall i : \mathbb{Z})(0 \leq i \leq cp.data.length() \rightarrow_L (2i+1 < cp.data.length() \wedge_L cp.data[i]_1 > cp.data[2i+1]_1) \wedge (2i+2 < cp.data.length() \wedge_L cp.data[i]_1 > cp.data[2i+2]_1)) \wedge |altura(cp.data, 2) - altura(cp.data, 1)| \leq 1$}

aux altura ($a : array < T >, i : int$)=$ifThenElse(i < a.length(), 1 + max(altura(2i+1), altura(2i+2)), 0)$

6.2

modulo $colaprioridad\langle T\rangle$ implementa $ColaPrioridad\langle T\rangle$ {

```
    var data : array⟨⟨T, ℝ⟩⟩
```

```
    proc encolar (cp : colaprioridad⟨T⟩, e : T, p : ℝ):  {
        i := 0
        copia := new array⟨⟨T, ℝ⟩⟩
        while (i < cp.data.length()) do
            copia[i] := cp.data[i]
            i++
        endwhile
        copia[i] := ⟨e, p⟩
```

```
        while
(((i%2 = 0 & copia[i]_1 > copia[(i − 2)/2]_1) ∥ (i%2! = 0 & copia[i]_1 > copia[(i − 1)/2]_1)) & i! = 0) do
            if (i%2 = 0) then
                copia[i-2]/2 := ⟨e, p⟩
                copia[i] := padre
                padre := copia[(i-2)/2]
                i := (i-2)/2
            else
                copia[i-1]/2 := ⟨e, p⟩
                copia[i] := padre
                padre := copia[(i-1)/2]
                i := (i-1)/2
        cp.data := copia
    }

    proc desencolar (cp:colaprioridad⟨T⟩): T {
        max := cp.data[0]
        copia := new array⟨T⟩(cp.data.length() − 1)
        copia[0] := cp.data[cp.data.length()-1]
        i := 1
        while (i < copia.length()) do
            copia[i] := cp.data[i]
            i++
        endwhile
        i := 0
        while (2i + 1 < copia.length() ∥ 2i + 1 < copia.length()) do
            if (copia[2i + 1]_1 > copia[2i + 2]_1) then
                raiz := copia[i]
                copia[i] := copia[2i+1]
                copia[2i+1] := raiz
                i := 2i+1
            else
                raiz := copia[i]
                copia[i] := copia[2i+2]
                copia[2i+2] := raiz
                i := 2i+2
            endif
        endwhile
        cp.data := copia
        return max_0
    }

    proc cambiarPrioridad (cp : colaprioridad⟨T⟩, e : T, p : ℝ):  {
        while (cp.data[i]_0! = e) do
            i++
        endwhile
        cp.data[i] := ⟨e, p⟩
```

(en esta parte del proc debo subir o bajar el elemento hasta que su padre sea mas grande y sus hijos mas chicos) }

}

7

En todos los lugares donde considero la prioridad, en vez de un ifThenElse deberia tener una estructura if(prior1)...elseIf(prior2)...else... donde prior1 y prior2 serian las prioridades. En el caso de los while deberia considerar que si al comparar los valores obtengo una igualdad entra en juego la segunda prioridad

8

```
proc ordenar (a:array⟨T⟩): array⟨T⟩ {
    h := new colaPrioridadLog⟨T⟩
        h.colaPrioridadDesdeSecuencia(a)
    res := new array⟨T⟩(a.length())
    i := 0
    while (i < a.length()) do
        prox := desencolarMax(h)
        array[i] := prox
    endwhile
    return res
}
```

9 HACER

10 HACER

11

Nodo es $struct\langle valor : T, alfabeto : vector\langle Nodo\rangle\rangle$
palabra es $seq\langle T\rangle$

```
modulo trieDePalabras⟨T⟩ implementa TrieDePalabras {
    var raiz : Nodo

    proc primerPalabra (t:trieDePalabras⟨T⟩): palabra {
        act := t.raiz
        i := 0
        while (i ≤ 25 & actual.alfabeto! = null) do
            if (actual.alfabeto[i]!=null) then
                res := res+actual.valor
                actual := actual.alfabeto[i]
                i := 0
            else
                i++
```

```
        endwhile
        return res
    }

    proc ultimaPalabra (t:trieLetras⟨T⟩): palabra {
        act := t.raiz
        i := 25
        while (i ≥ 0 & actual.alfabeto! = null) do
            if (actual.alfabeto[i]!=null) then
                res := res+actual.valor
                actual := actual.alfabeto[i]
                i := 25
            else
                i- -
        endwhile
        return res
    }

    proc buscarIntervalo (t:trieLetras⟨T⟩; p1, p2 : palabras): seq⟨palabra⟩ {
        p := null
        while (p < p2) do
            p := primerPalabra(t)
            borrar(p,t)
            if (p1 ≤ p ≤ p2) then
                res := res+p
            endif
        endwhile
        return res
    }

}
```

12

Nodo es $struct\langle val : int, izq : Nodo, der : Nodo\rangle$

```
    proc filRep (mb : array⟨array⟨ℤ⟩⟩): array⟨ℤ⟩ {
        fil := 0
        act := new Nodo⟨ℤ⟩
        raiz := new Nodo⟨ℤ⟩
        res := new array⟨ℤ⟩(mb.length())
        i := 0
        while (i¡mb.length()) do
            res[i] := -1
        endwhile
        while (f < mb.length()) do
            col := 0
            act := raiz
```

```
        while (c < mb[fil][col].length()) do
            if (mb[fil][col] = 0) then
                act := act.izq
                if (c=mb[f].length-1 & act.val!=null) then
                    res[act.val] := f
                endif
                act.val := f
            endif
            if (mb[fil][col] = 1) then
                act := act.der
                if (c=mb[f].length-1 & act.val!=null) then
                    res[act.val] := f
                endif
                act.val := f
            endif
            c++
        endwhile
        f++
    endwhile
    return res
}
```

13

Nota: voy a suponer una funcion booleana `esMayus()` que devuelve si una letra es mayuscula
Nodo es $struct\langle pals : vector\langle String\rangle, abecedario : vector\langle Nodo\rangle\rangle$
Supongo que al inicializar el Nodo pals se inicializa vacio y abecedario con 27 Nodos que representan a cada letra del abecedario

```
proc pal2Patron (pal:String): array⟨⟩ {
    i := 0
    patron := new array⟨Char⟩
    while (i < pal.length()) do
        if (esMayus(pal[i])) then
            patron := patron+pal[i]
        endif
        i++
    endwhile
    return patron
}

proc array2trie (a:array⟨String⟩): Nodo {
    i := 0
    listaPatrones := newarray⟨(String, String)⟩(a.length())
    while (i < a.length()) do
        patron := pal2Patron(a[i])
```

```
            listaPatrones := listaPatrones+(patron,a[i])
            i++
        endwhile
        res := new Nodo
        i := 0
        while (i < listaPatrones.length()) do
            j := 0
            act := res
            while (j < listaPatrones[i]_0.length()) do
                act := act.abecedario[j]
                j++
            endwhile
            act.pals.agregarAtras(listaPatrones[i]_1)
            i++
        endwhile
        return res
    }

    proc palabrasConPatron (a:array⟨String⟩, p:String): array⟨String⟩ {
        trie := array2trie(a)
        i := 0
        act := trie
        while (i < p.length()) do
            letra := p[i]
            act := act.abecedario[letra]
            i++
        endwhile
Falta terminar aca      }
```