

Tarea 3

Manejo dinámico de estructuras lineales y arborescentes

Curso 2021

Índice

1. Introducción y objetivos	2
2. Materiales	2
3. ¿Qué se pide?	2
4. Descripción de los módulos y algunas funciones	2
4.1. Módulo Iterador	2
4.2. Módulo <i>Binario</i>	3
4.2.1. <i>insertarEnBinario</i> (<i>TInfo i, TBinario b</i>)	3
4.2.2. <i>removerDeBinario</i> (<i>nat elem, TBinario b</i>)	3
4.2.3. <i>alturaBinario</i> (<i>TBinario b</i>)	4
4.2.4. <i>buscarSubarbol</i> (<i>nat elem, TBinariot b</i>)	4
4.2.5. <i>imprimirBinario</i> (<i>TBinario b</i>)	5
4.2.6. <i>menores</i> (<i>double cota, TBinario b</i>)	5
4.2.7. <i>sumaUltimosPositivos</i> (<i>nat i, binario_t b</i>)	6
4.2.8. <i>esAVL</i> (<i>TBinariot b</i>)	6
4.3. Módulo <i>UsoTads</i>	7
4.3.1. <i>esCamino</i> (<i>TCadena c, TBinariot b</i>)	8
5. Entrega	8
5.1. Plazos de entrega	8
5.2. Identificación de los archivos de las entregas	8
5.3. Individualidad	9
6. Funciones auxiliares	10

1. Introducción y objetivos

Esta tarea tiene los siguientes objetivos específicos:

- Trabajar sobre el manejo dinámico de memoria, incorporando estructuras arborescentes.
- Trabajar con **árboles binarios de búsqueda**.
- Realizar funciones que cumplan restricciones en el orden del tiempo de ejecución.
- Implementar funciones recursivas.

En la presente tarea continuaremos trabajando sobre el manejo dinámico de memoria, incorporando estructuras arborescentes. Trabajaremos con **árboles binarios de búsqueda**.

Los elementos de los árboles son de tipo `TInfo`, el cual ya fue utilizado en la tarea anterior. Además se agrega una **lista con posición implícita**.

Se agregaron restricciones en el orden del tiempo de ejecución para algunas operaciones de los diferentes módulos. Esto puede requerir modificaciones en las implementaciones realizadas en la tarea anterior y así garantizar que la solución a entregar cumpla con dichas restricciones.

El correcto uso de la memoria y el cumplimiento de las restricciones de tiempo serán parte de la evaluación.

2. Materiales

Los archivos para la tarea se extraen de *MaterialesTarea3.tar.gz*. Para conocer la estructura de los directorios ver la Sección **Materiales** de [Estructura y funcionamiento del Laboratorio](#).

En esta tarea los archivos en el directorio `include` son los de la tarea anterior `utils.h`, `info.h`, `cadena.h` y `usoTads.h`, y se agregan `iterador.h` y `binario.h`.

En el directorio `src` se incluyen ya implementados `utils.cpp` e `info.cpp`.

3. ¿Qué se pide?

Ver la Sección **Desarrollo** de [Estructura y funcionamiento del Laboratorio](#).

En esta tarea solo se deben implementar los archivos `cadena.cpp`, `usoTads.cpp`, `iterador.cpp` y `binario.cpp`.

Se debe tener en cuenta que en `cadena.h` se mantienen las mismas funciones que antes, pero en `usoTads.h` se agregaron nuevas funciones.

Se sugiere implementar y probar los tipos y las funciones siguiendo el orden de los ejemplos que se encuentran en el directorio `test`. En la primera línea de cada caso se indica que entidades se deben haber implementado. Después de implementar se compila y se prueba el caso.

4. Descripción de los módulos y algunas funciones

En esta sección se explica, mediante ejemplos, el comportamiento de algunas funciones representativas de los módulos a implementar. Se utilizará una notación gráfica para representar árboles, donde el árbol vacío se representa con el símbolo \bullet . Los elementos de tipo `TInfo` se representan con el formato definido en la función `infoATexto`.

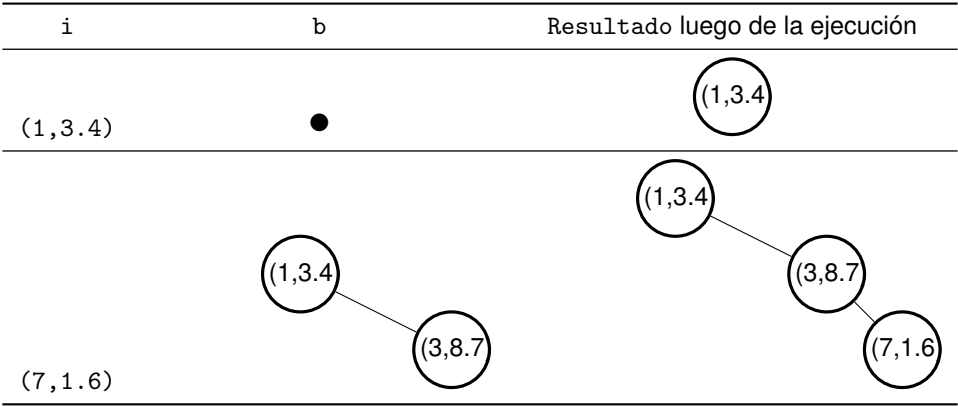
4.1. Módulo Iterador

En este módulo se debe implementar iteradores (de tipo `TIterador`) sobre elementos de tipo `nat`. Un iterador es una colección de elementos que se puede recorrer.

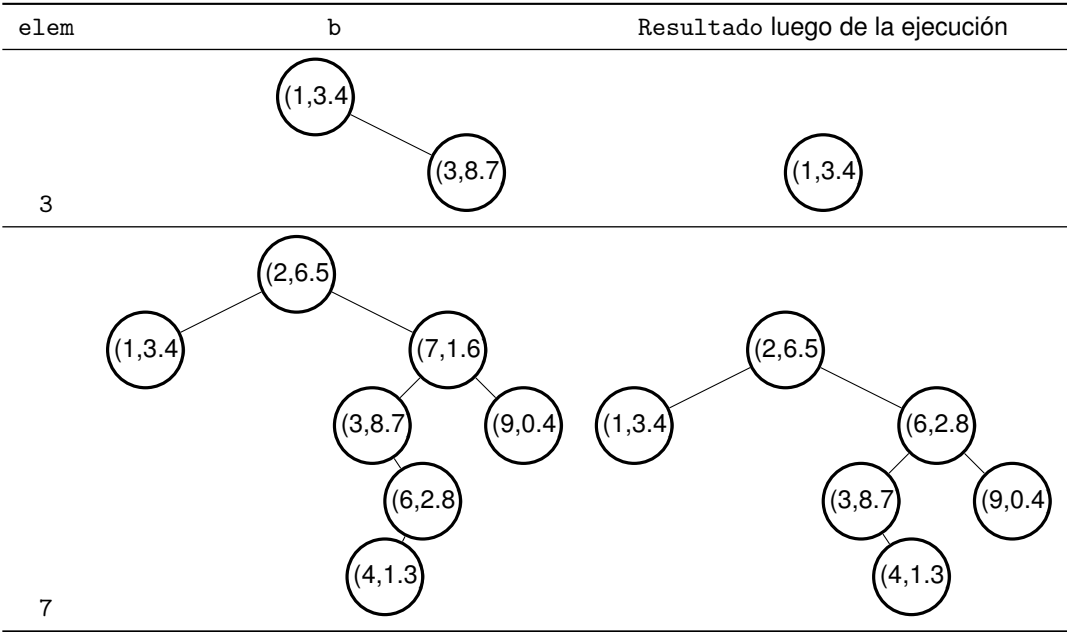
4.2. Módulo *Binario*

En este módulo se implementará un árbol binario de búsqueda (de tipo *TBinario*) de elementos de tipo *TInfo*. La propiedad de orden de los árboles es definida por el componente natural de sus elementos. A continuación presentaremos mediante ejemplos algunas funciones representativas del módulo **binario**.


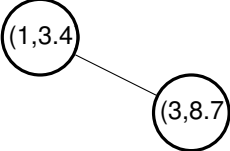
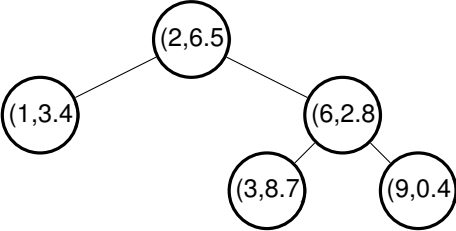
4.2.1. *insertarEnBinario (TInfo i, TBinario b)*



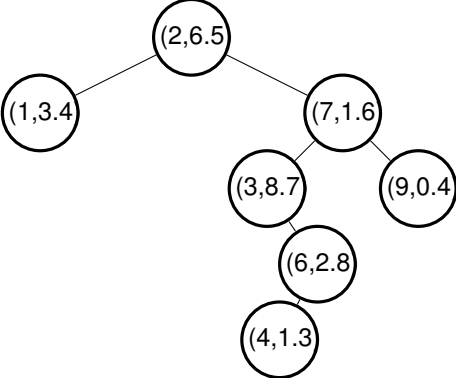

4.2.2. *removerDeBinario (nat elem, TBinario b)*



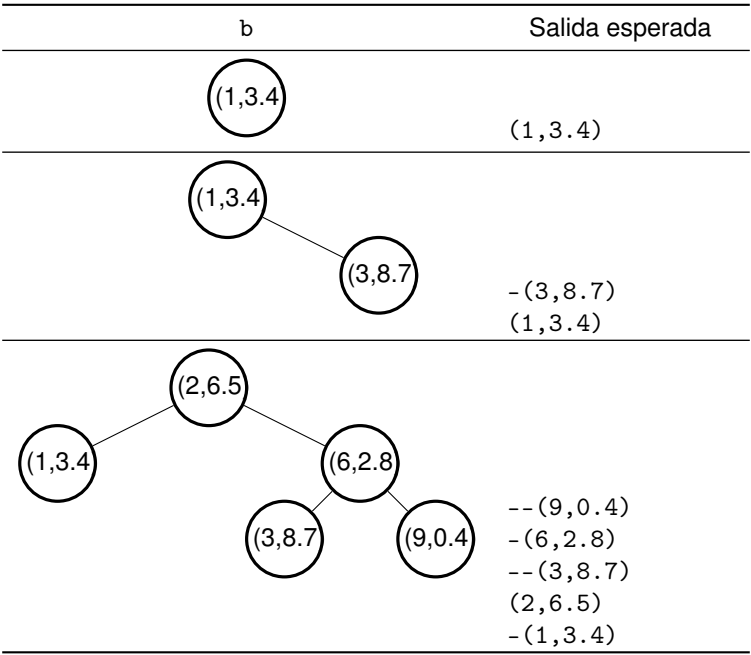
4.2.3. alturaBinario (TBinario b)

b	Altura de b
	1
	2
	3

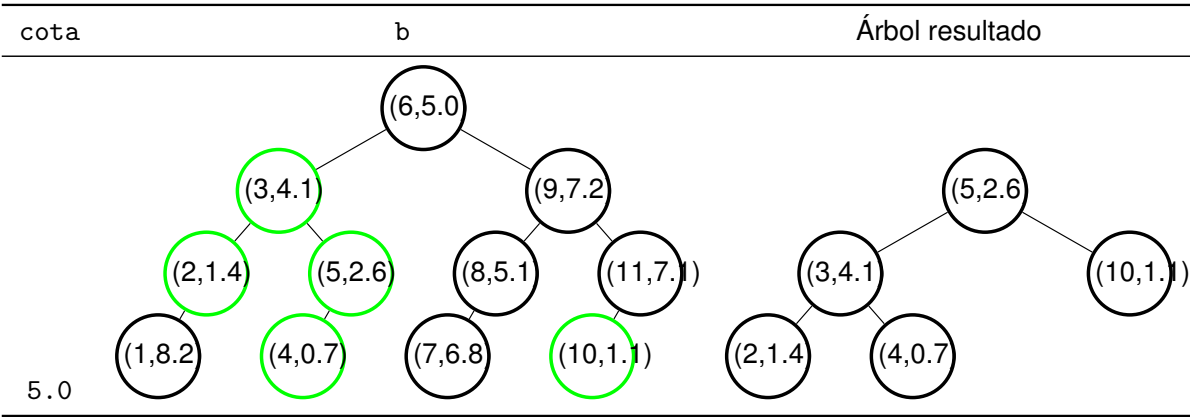
4.2.4. buscarSubarbol (nat elem, TBinariot b)

elem	b	Árbol resultado
6		

4.2.5. *imprimirBinario (TBinario b)*



4.2.6. *menores (double cota, TBinario b)*



Cuadro 1: Ejemplo de llamada a la función menores

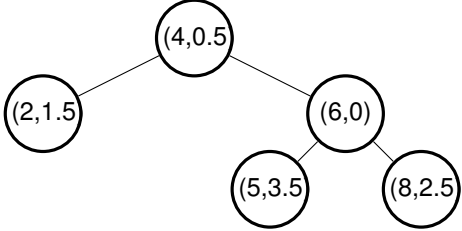
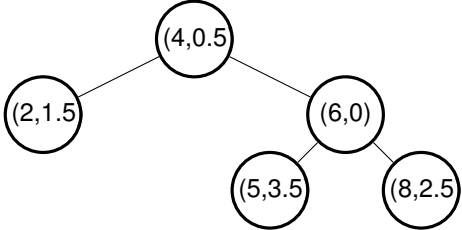
En el Cuadro 1 se muestra un ejemplo de llamada a la función `menores`. En dicho ejemplo se destacan en verde los nodos en donde se cumple la condición, cuyas copias serán los nodos del árbol resultado. El resultado de la operación en el árbol que tenga como raíz uno de los nodos que cumplen con la condición, es un árbol que en cuya raíz está una copia de ese nodo. Los dos subárboles son el resultado de la operación aplicada en los hijos de ese nodo.

En los nodos cuyos elementos son (1,8.2) y (7,6.8) no se cumple la condición y no tienen descendientes. En (8,5.1) no se cumple y tampoco se cumple en sus descendientes. Por lo tanto el resultado del filtrado en cualquiera de esos nodos es el árbol vacío.

En (11,7.1) no se cumple la condición y no hay nodos en su subárbol derecho. El resultado en este nodo es, entonces, el resultado en su hijo izquierdo, que es el árbol cuyo único nodo es una copia de (10,1.1), por ser el único nodo que cumple con la condición en el subárbol izquierdo. De manera análoga, ni en (9,7.2) ni en su subárbol izquierdo hay nodos que cumplan la condición, por lo que el resultado en (9,7.2) es el resultado en su hijo derecho.

En la raíz del árbol, (6,5.0), no se cumple la condición pero en sus dos subárboles hay nodos que la cumplen. En el resultado la raíz debe ser la copia de algún otro nodo y el árbol devuelto debe mantener la propiedad de orden. Se elige como raíz una copia del nodo mayor (según el orden definido) de los que cumplen la condición en el subárbol izquierdo, que es (5,2.6). Este nodo debe removerse del resultado de la llamada en (3,4.1) para pasar a ser la raíz del árbol resultado.

4.2.7. *sumaUltimosPositivos* (*nat i*, *binario_t b*)

i	b	Resultado esperado
1	●	0
0	(4,0.5)	0
1	(4,0.5)	0.5
2	(4,0.5)	0.5
2		6.0
4		8

4.2.8. *esAVL* (*TBinariot b*)

Los árboles AVL son un tipo particular de ABB que están siempre equilibrados en el sentido de que para todos los nodos, la altura de la rama izquierda no difiere en más de una unidad de la altura de la rama derecha. Debido a esta propiedad, el orden de búsqueda en el árbol es siempre $O(\log(n))$.

En esta ocasión se les solicita evaluar si dado un ABB, el mismo cumple con la propiedad AVL, es decir, verificar si el valor absoluto de la diferencia de las alturas de los subárboles izquierdo y derecho de **cada** nodo es menor o igual a 1.

b	Resultado esperado
●	true
<pre> graph TD A["(5,1.1)"] --- B["(3,1.1)"] A --- C["(6,3.2)"] </pre>	true
<pre> graph TD A["(5,1.1)"] --- B["(3,1.1)"] A --- C["(6,3.2)"] C --- D["(8,1.1)"] </pre>	true
<pre> graph TD A["(5,1.1)"] --- B["(3,1.1)"] A --- C["(6,3.2)"] B --- D["(1,1.1)"] C --- E["(8,1.1)"] E --- F["(7,1.1)"] E --- G["(9,1.1)"] </pre>	false

Se debe notar que:

- El árbol vacío se considera balanceado.
- En el cuarto ejemplo de la tabla, a pesar de que la raíz está balanceada, el resultado de la llamada a la función es `false` porque el nodo (6,3.2) no está balanceado ya que su subárbol izquierdo tiene altura 0 mientras que su subárbol derecho tiene altura 2.

4.3. Módulo *UsoTads*

Este módulo corresponde a funciones que usan las operaciones declaradas en los módulos *info*, *cadena* y *binario*.

4.3.1. *esCamino* (*TCadena c*, *TBinariot b*)

c	b	Resultado esperado
[]	●	true
[(2,5.0)]		true
[(2,5.0), (5,3.1)]		false
[(2,5.0), (5,3.1)]		false
[(2,5.0), (5,3.1), (4,1.8)]		true

5. Entrega

Ver la Sección **Entregas** de [Reglamento del Laboratorio](#).

Se debe entregar el siguiente archivo, que contiene los módulos implementados `cadena.cpp`, `usoTads.cpp`, `iterador.cpp` y `binario.cpp`:

■ Entrega3.tar.gz

Este archivo se obtiene al ejecutar la regla `entrega` del archivo *Makefile*:

```
$ make entrega
tar zcvf Entrega3.tar.gz -C src cadena.cpp usoTads.cpp binario.cpp iterador.cpp
cadena.cpp
usoTads.cpp
binario.cpp
iterador.cpp
```

Con esto se empaquetan los módulos implementados y se los comprime.

5.1. Plazos de entrega

El plazo para la entrega es el **martes 27 de abril a las 18 horas**.

5.2. Identificación de los archivos de las entregas

Cada uno de los archivos a entregar debe contener, en la primera línea del archivo, un comentario con el número de cédula del estudiante, **sin el guión y sin dígito de verificación**. Ejemplo:

```
/* 1234567 */
```


5.3. Individualidad

Ver la Sección **Individualidad** de [Reglamento del Laboratorio](#).

6. Funciones auxiliares

Para la implementación de algunas de las funciones de esta tarea podrían necesitarse funciones auxiliares. En C no se puede definir una función anidada dentro de otra como se puede hacer en Pascal. Por lo tanto la función auxiliar debe definirse fuera de las funciones en las que se va a usar. Además, la declaración debe estar antes de que la función sea usada.

Esto puede llegar a generar un conflicto en el momento del enlazado si en otros archivos también hubiera funciones auxiliares con la misma firma ya que en C las funciones de manera predeterminada tienen ámbito global. Para evitar este problema se debe anteponer la palabra reservada `static` a la declaración de la función. Con esto queda como local al archivo. Su ámbito va desde la declaración hasta el fin del archivo.

```
// Declaración de 'auxiliar' usada en 'f' y 'g'.
// Debe preceder a la implementación de las funciones que la usan.
static int auxiliar(int p);
// La implementación se puede hacer junto con la declaración
// o después.

// f y g usan 'auxiliar'
void f() {
    ...
    int a = auxiliar(5);
    ...
}

void g() {
    ...
    int a = auxiliar(8);
    ...
}

// implementación de auxiliar
static int auxiliar(int p) {
    ...

    ...
    return p + ...;
}
```