

# Tarea 4

## Implementación de Tipos Abstractos de Datos

### Curso 2018

#### Índice

|   |          |
|---|----------|
| <b>1. Introducción</b>  | <b>2</b> |
| <b>2. Materiales</b>  | <b>2</b> |
| <b>3. ¿Qué se pide?</b>                                       | <b>3</b> |
| 3.1. Verificación de la implementación . . . . .              | 3        |
| <b>4. Descripción de los módulos y algunas funciones</b>      | <b>3</b> |
| 4.1. Módulos de tareas anteriores . . . . .                   | 3        |
| 4.1.1. Módulo <i>info</i> . . . . .                           | 3        |
| 4.1.2. Módulo <i>cadena</i> . . . . .                         | 3        |
| 4.1.3. Módulo <i>binario</i> . . . . .                        | 3        |
| 4.2. Módulos nuevos . . . . .                                 | 4        |
| 4.2.1. Módulo <i>pila</i> . . . . .                           | 4        |
| 4.2.2. Módulo <i>cola_binarios</i> . . . . .                  | 4        |
| 4.2.3. Módulo <i>iterador</i> . . . . .                       | 4        |
| 4.2.4. Módulo <i>conjunto</i> . . . . .                       | 4        |
| 4.2.5. Módulo <i>uso_tads</i> . . . . .                       | 4        |
| <b>5. Entrega</b>   | <b>5</b> |
| 5.1. Plazos de entrega . . . . .                              | 5        |
| 5.2. Identificación de los archivos de las entregas . . . . . | 6        |

## 1. Introducción

El objetivo de esta tarea es la implementación de **Tipos Abstractos de Datos (TADs)**, cumpliendo requerimientos de tiempo de ejecución. Trabajaremos sobre los módulos `cadena` y `binario` de las tareas anteriores. En estos módulos, se agregaron restricciones en el tiempo de ejecución en el peor caso para algunas operaciones. Esto puede requerir modificaciones en las implementaciones realizadas en tareas anteriores, y así garantizar que la solución a entregar cumpla con dichas restricciones. Los detalles sobre los módulos de tareas anteriores se presentan en la Sección 4.1.

También trabajaremos sobre módulos nuevos, que implementan pilas, colas de árboles binarios, iteradores y conjuntos. Los detalles de los módulos nuevos se presentan en la Sección 4.2.

Como corresponde al concepto de TAD no se puede usar la representación de un tipo fuera de su propio módulo. Por ejemplo, en `uso_tads.cpp` no se puede usar `rep_cadena` sino las operaciones declaradas en `cadena.h`.

El correcto uso de la memoria será parte de la evaluación.

El cumplimiento de las restricciones de tiempo será parte de la evaluación.

La tarea otorga un punto a los grupos que la aprueben en la primera instancia de evaluación. La adjudicación de los puntos de las tareas de laboratorio queda condicionada a la respuesta correcta de una pregunta sobre el laboratorio en el segundo parcial. La adjudicación se aplica de manera individual a cada estudiante del grupo.

## 2. Materiales

Los materiales para realizar esta tarea se encuentran en el archivo *MaterialesTarea4.tar.gz* que se obtiene en la carpeta *Materiales* de la sección *Laboratorio* del sitio EVA del curso <sup>1</sup>.

A continuación se detallan los archivos que se entregan. La estructura de directorios es igual a la de la tarea anterior.

- **info.h**: módulo de definición del tipo `info_t`
- **cadena.h**: módulo de definición de `cadena`.
- **binario.h**: módulo de definición de árbol binario de búsqueda
- **pila.h**: módulo de definición de pilas de elementos de tipo `string`
- **cola\_binarios.h**: módulo de definición de colas de árboles binarios de búsqueda
- **iterador.h**: módulo de definición de iteradores sobre elementos de tipo `string`
- **conjunto.h**: módulo de definición de conjuntos de elementos de tipo `info_t`
- **uso\_tads.h**: módulo de definición de funciones sobre `tads`
- Casos de prueba en el directorio **test**
- **Makefile**: archivo para usar con el comando `make`, que provee las reglas `principal`, `testing`, `clean` y entrega como en las tareas anteriores.
- **principal.cpp**: módulo principal.

**Ninguno de estos archivos deben ser modificados.**

<sup>1</sup><https://eva.fing.edu.uy/course/view.php?id=132&section=6>

### 3. ¿Qué se pide?

Para cada archivo de encabezamiento **.h**, descrito en la Sección 2, se debe implementar un archivo **.cpp** que implemente **todas** las definiciones de tipo y funciones declaradas en el archivo de encabezamiento correspondiente. Los archivos **cpp** serán los entregables y deben quedar en el directorio **cpp**.

En algunas operaciones se piden requerimientos de tiempo de ejecución. Ese tiempo es el del peor caso, a menos que se especifique otra cosa de manera explícita.

#### 3.1. Verificación de la implementación

Para testear la implementación debe generar el ejecutable y cumplir con todas las pruebas utilizando los archivos **.in** y **.out**. Tenga en cuenta que el archivo **Makefile** proveerá las reglas **principal**, **testing** y **clean** como en las tareas anteriores.

## 4. Descripción de los módulos y algunas funciones

### 4.1. Módulos de tareas anteriores

#### 4.1.1. Módulo *info*

Este módulo se implementó en la tarea 1, y representa pares de enteros y frases.

#### 4.1.2. Módulo *cadena*

Este módulo se implementó en la tarea 2, y representa cadenas de elementos de tipo `info_t`. Para esta tarea, se agregaron los tiempos de ejecución de las operaciones.

#### 4.1.3. Módulo *binario*

Este módulo se implementó en la tarea 3, y representa un árbol binario de búsqueda (de tipo `binario_t`) de elementos de tipo `info_t`. Para esta tarea se agregó la operación `crear_balanceado`.

También se agregaron los tiempos de ejecución de las operaciones.

A continuación se brindan detalles de la operación `crear_balanceado`.

**crear\_balanceado** Aquí se fundamenta (sin demostrar) la cota de tiempo  $O(n \cdot \log n)$  requerida en la ejecución de `crear_balanceado` en `binario.h`. Esta fundamentación muestra que es posible alcanzar la cota requerida.

Es importante recordar que la entrada está ordenada y no tiene elementos repetidos, y que el árbol es binario de búsqueda.

Teniendo en cuenta que la cantidad de niveles de un árbol balanceado de  $n$  nodos es  $O(\log n)$  (ver expresión (1)), se puede llegar a la cota solicitada si en la implementación se logra que el tiempo de procesamiento dedicado a cada nivel sea  $O(n)$ . Esto no significa que la implementación debe hacerse “por niveles”, sino que, para cada nivel, la suma de los tiempos dedicados a sus nodos sea  $O(n)$ . Y, a su vez, esto se puede lograr si el tiempo dedicado al procesamiento de cada nodo es lineal respecto al tamaño de su entrada.

Para simplificar, supongamos que la cantidad de elementos de la entrada,  $n$ , genera un árbol perfecto de  $h$  niveles (un árbol es perfecto si todas las hojas están en el último nivel). En el nivel  $i$ -ésimo, la cantidad de nodos es  $2^{i-1}$  (se considera que la raíz está en el nivel 1). Esto implica que la cantidad de nodos es

$$n = 1 + 2 + 2^2 + \dots + 2^{h-1} = \sum_{i=1}^h 2^{i-1} = 2^h - 1, \quad (1)$$

de donde  $h = \log(n + 1)$  y por lo tanto  $h = O(\log n)$ .

El tiempo de ejecución de un algoritmo que implementa de manera eficiente lo solicitado es:

$$\begin{aligned}
& O(n) + 2 \cdot O(n/2) + 4 \cdot O(n/4) + \dots + 2^{h-1} \cdot O(n/2^{h-1}) \\
&= O(n(1 + 2^1 \cdot 1/2^1 + 2^2 \cdot 1/2^2 + \dots + 2^{h-1} \cdot 1/2^{h-1})) \\
&= O(n \cdot h) \\
&= O(n \cdot \log n).
\end{aligned}$$

Esto se justifica gráficamente en la figura 1, en la que en cada nodo se muestra el tamaño de la entrada que debe procesar, que es igual a la cantidad de nodos del subárbol del que es raíz.

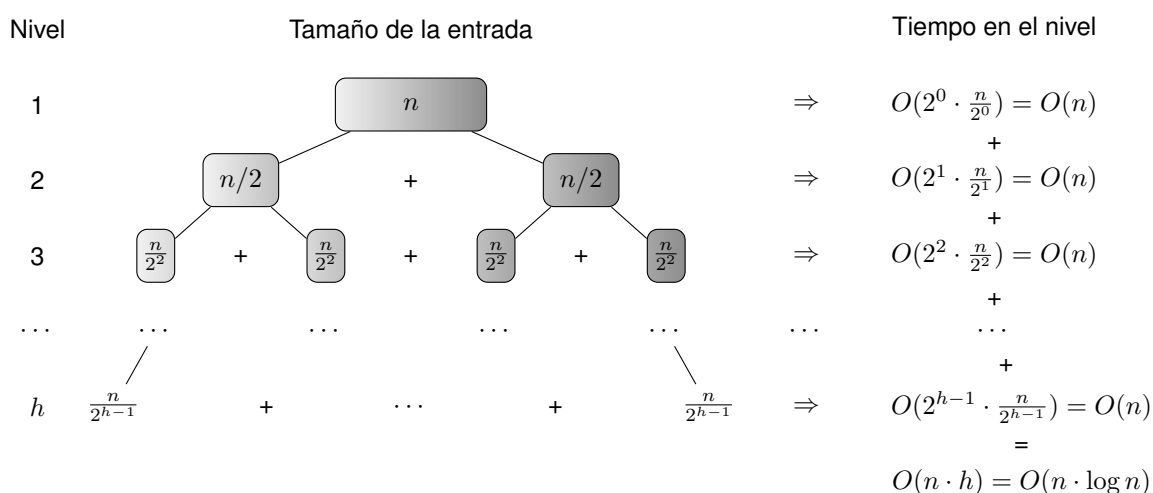


Figura 1: **Crear un árbol balanceado.** En cada nodo se muestra el tamaño de la entrada que se debe tratar. Si el procesamiento del nodo se hace en tiempo lineal respecto a ese tamaño, el procesamiento de todo el nivel se hace en tiempo  $O(n)$ , y el de todo el algoritmo en tiempo  $O(n \cdot \log n)$ .

## 4.2. Módulos nuevos

### 4.2.1. Módulo *pila*

En este módulo se implementará una pila (de tipo `pila_t`) de elementos de tipo string.

### 4.2.2. Módulo *cola\_binarios*

En este módulo se implementará una cola (de tipo `cola_binarios_t`) de elementos de tipo árboles binarios de búsqueda (de tipo `binario_t`). Es posible que un árbol sea subárbol de otro, o sea, los árboles pueden compartir memoria.

### 4.2.3. Módulo *iterador*

En este módulo se implementará un iterador (de tipo `iterador_t`) sobre elementos de tipo string. Un iterador es una colección de elementos que se puede recorrer.


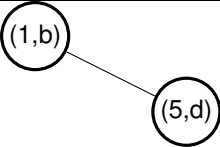
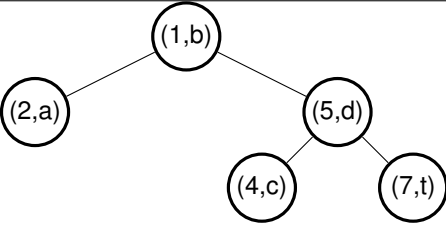
### 4.2.4. Módulo *conjunto*

En este módulo se implementará un conjunto (de tipo `conjunto_t`) de elementos de tipo `info_t`. Una restricción adicional es que no puede haber dos elementos con el mismo dato de texto.

### 4.2.5. Módulo *uso\_tads*

Este módulo reemplaza el módulo `uso_cadena` y se agregan las operaciones `esta_ordenada_por_frase`, `imprimir_por_niveles`, `inverso_de_iter` y `rango_en_conjunto`.

**imprimir\_por\_niveles** Se muestran algunos ejemplos de lo que se espera de la ejecución de la función `imprimir_por_niveles`.

|   |                 |
|---|-----------------|
|  | Salida esperada |
|  | b               |
|  | d<br>b          |
|   | c t<br>a d<br>b |

## 5. Entrega

Se mantienen las consideraciones reglamentarias y de procedimiento de las tareas anteriores.

Se debe entregar el siguiente archivo, que contiene los módulos implementados *binario.cpp*, *cadena.cpp*, *cola\_binarios.cpp*, *conjunto.cpp*, *info.cpp*, *iterador.cpp*, *pila.cpp* *uso\_tads.cpp*:

### ■ Entrega4.tar.gz

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

```
$ make entrega
tar zcvf Entrega4.tar.gz -C src info.cpp cadena.cpp binario.cpp pila.cpp cola_binarios.cpp conjunto
info.cpp
cadena.cpp
binario.cpp
pila.cpp
cola_binarios.cpp
conjunto.cpp
iterador.cpp
uso_tads.cpp
```

**NO SE PUEDEN ENTREGAR MÓDULOS ADICIONALES A LOS SOLICITADOS**

### 5.1. Plazos de entrega

El plazo para la entrega es el **miércoles 30 de mayo a las 14 horas**.

**NO SE ACEPTARÁN ENTREGAS DE TRABAJOS FUERA DE FECHA Y HORA. LA NO ENTREGA O LA ENTREGA FUERA DE LOS PLAZOS INDICADOS IMPLICA LA PÉRDIDA DEL CURSO.**

## 5.2. Identificación de los archivos de las entregas

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

```
/* 1234567 - 2345678 */
```