

Tarea 4

Implementación de TADs

Curso 2021

Índice

1. Introducción y objetivos	2
2. Materiales	2
3. ¿Qué se pide?	2
4. Descripción de los módulos y algunas funciones	3
4.1. Módulos de tareas anteriores	3
4.1.1. <i>recorridaPorNiveles</i>	4
4.2. Módulos nuevos	4
4.2.1. Módulo <i>pila</i>	4
4.2.2. Módulo <i>colaBinarios</i>	4
5. Entrega	4
5.1. Plazos de entrega	5
5.2. Identificación de los archivos de las entregas	5
5.3. Individualidad	5

1. Introducción y objetivos

El objetivo de esta tarea es la implementación de **Tipos Abstractos de Datos (TADs)** y de nuevos algoritmos cumpliendo requerimientos de tiempo de ejecución. Mantenemos los módulos de las tareas anteriores. Los detalles sobre los módulos de tareas anteriores se presentan en la Sección 4.1.

También trabajaremos sobre módulos nuevos, que implementan pilas, y colas de árboles binarios. 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.

Recordar que:

- El correcto uso de la memoria será parte de la evaluación.
- En esta tarea se puede trabajar únicamente en forma individual.

El resto del presente documento se organiza de la siguiente forma. En la Sección 2 se presenta una descripción de los materiales disponibles para realizar la presente tarea, y en la Sección 3 se detalla el trabajo a realizar. Luego, en la Sección 4 se explica mediante ejemplos el comportamiento esperado de algunas de las funciones a implementar.

2. Materiales

Los materiales para realizar esta tarea se extraen de *MaterialesTarea4.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 a los que se agregan `pila.h` y `colaBinarios.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 se deben implementar los mismos archivos que en la tarea anterior y además los nuevos. Se debe tener en cuenta que en `binario.h` y `usoTads.h` se agregaron nuevas funciones.

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.

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 de prueba se indica que entidades se deben haber implementado. Después de implementar se compila y se prueba el caso.

Los siguientes comandos compilan, ejecutan el test (sin `valgrind`) y verifican que la salida generada por su programa es igual a la salida esperada:

```
$ make
$ ./principal < test/N.in > test/N.sal
$ diff test/N.out test/N.sal
```

Si la salida generada por su programa es igual a la salida esperada no se imprime nada.

Pero, para tener en cuenta el correcto uso de la memoria y ser notificado de posibles errores de memoria, se debe ejecutar además el siguiente comando:

```
$ valgrind --leak-check=full ./principal <test/NN.in> test/NN.sal
```

Se puede hacer una verificación rápida mediante reglas t-CAS0, donde CAS0 se debe sustituir por el nombre de uno de los archivos in del directorio test:

```
$ make t-01
---- Bien ----
```

Si el resultado no es correcto se debería proceder al método descrito antes.

Al terminar, para confirmar, se compila y prueban todos los casos mediante la regla `testing` de Makefile:

```
$ make testing
```

Al ejecutar el comando `make testing` los casos se prueban usando `valgrind` y la utilidad `timeout`:

```
$ timeout 4 valgrind -q --leak-check=full ./principal < test/01.in > test/01.sal
```

`timeout` es una utilidad que establece un plazo para la ejecución de un proceso (4 en este ejemplo). Si la ejecución demorara más de ese plazo `timeout` terminaría ese proceso.

Los test que controlan eficiencia de tiempo se corren sin el uso de `valgrind`. En estos test mantener habilitado los `assert` puede enlentecer la ejecución por lo que se sugiere incluir la opción de compilación `-DNDEBUG`. En Makefile esto se consigue comentando la línea 50

```
# CCFLAGS = -Wall -Werror -I$(HDIR) -g
```

y descomentando la línea 53

```
CCFLAGS = -Wall -Werror -I$(HDIR) -g -DNDEBUG
```

Se debe recordar que la regla `testing` se dejará disponible sobre la fecha de entrega de la tarea.

Ver el material disponible sobre *Makefile*: [Automatización de procedimientos](#).

Ver también el [Método de trabajo sugerido](#).

4. Descripción de los módulos y algunas funciones

4.1. Módulos de tareas anteriores


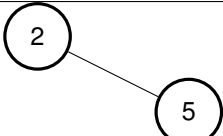
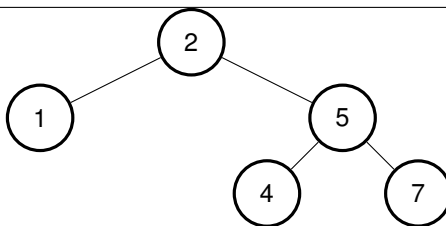
Se mantienen los mismos módulos de la Tarea 3: *utils*, *info*, *cadena*, *usoTads*, *iterador*, y *binario*.

En el módulo *binario* se agregó la operación *binarioBalanceado*. Dado un arreglo ordenado de naturales devuelve un *TBinario* balanceado. El criterio de balanceo está definido por la cantidad de nodos en cada subárbol. Se incluye una prueba de tiempo para esta operación.

En el módulo *usoTads* se agregaron las operaciones *cantidadEnIterador*, *enAlguno*, *soloEnA* y *recorridaPorNiveles*.

4.1.1. recorridaPorNiveles

Devuelve un TIterador con los componentes naturales de los elementos del árbol por niveles, desde el más profundo hasta la raíz. La separación entre niveles se indica mediante el valor `UINT_MAX` (definido en la biblioteca `limits.h`).

b	Salida esperada
	2
	5 2
	4 7 1 5 2

Cuadro 1: Se muestra en la salida los elementos que se obtendrían al recorrer el TIterador resultado. Se usa `|` como separador de niveles en lugar de `UINT_MAX`.

Se incluye una prueba de tiempo para esta operación.

4.2. Módulos nuevos

4.2.1. Módulo *pila*

En este módulo se debe implementar pilas acotadas (de tipo `TPila`) de elementos de tipo `nat`.

4.2.2. Módulo *colaBinarios*

En este módulo se debe implementar colas (de tipo `TColaBinarios`) de elementos de tipo árboles de tipo `TBinario`. Es posible que un árbol sea subárbol de otro, o sea, los árboles pueden compartir memoria.

5. Entrega

Se mantienen las consideraciones reglamentarias y de procedimiento de la tarea anterior.

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

-- El directorio y archivo a entregar es:
???/tarea4/Entrega4.tar.gz

Con esto se empaquetan los módulos implementados y se los comprime.
Además se genera el archivo `sha.txt` que contiene una clave que identifica al archivo `Entrega4.tar.gz`.
Ese archivo también se puede generar mediante la utilidad `shasum`:

```
$ shasum -b Entrega4.tar.gz > sha.txt
```

5.1. Plazos de entrega

El plazo para la entrega es el **martes 25 de mayo 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](#).