Tarea 4
Implementación de TADs
Curso 2022

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.
Se incluyen test de tiempo. Estos son test en los que el tiempo de ejecución tiene un timeout exigente.
Se debe recordar que:

- **Como cada tarea, es individual y eliminatoria.**
- **La evaluación se hace con casos de prueba que se publican al terminar el plazo de entrega.**
- **Quien no apruebe en la entrega podrá realizar una re-entrega en un plazo de hasta 24 horas después de publicado el resultado.**
- **La aprobación en la entrega otorga 2 puntos; la aprobación en la re-entrega no otorga puntos.**
- **La tarea debe compilar y funcionar correctamente con la regla make testing en las máquinas de la Facultad.**
- **El archivo a entregar se debe generar mediante la regla make entrega y no se le debe cambiar el nombre en el proceso de entrega.**

2. Materiales

Los archivos para la 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 **utils.h**, **info.h**, **palabras.h**, **cadena.h**, **colCadenas.h**, **iterador.h**, **abb.h**, **pila.h**, **cola.h** y **aplicaciones.h**. Estos archivos no se pueden modificar.

En el directorio `src` se incluyen ya implementados **utils.cpp**, **info.cpp** y **palabras.cpp**. Además, se incluyen los archivos **plantilla_cadena.cpp**, **plantilla_colCadenas.cpp**, **plantilla_iterador.cpp**, **plantilla_abb.cpp**, **plantilla_pila.cpp**, **plantilla_cola.cpp** y **plantilla_aplicaciones.cpp**. Estos siete archivos contienen las funciones que se deben implementar pero con lo mínimo para que pueda compilar. Se pueden usar para lo cual se les debe cambiar el nombre (esto es remover `plantilla`), poner la cédula (7 dígitos) y completar el cuerpo de las funciones y el struct.

3. Desarrollo

Ver la Sección **Desarrollo** de Estructura y funcionamiento del Laboratorio.

En esta tarea se deben implementar los archivos **cadena.cpp**, **colCadenas.cpp**, **iterador.cpp**, **abb.cpp**, **pila.cpp**, **cola.cpp**, y **aplicaciones.cpp**.

La generación del ejecutable se hace compilando los archivos implementados **cadena.cpp**, **colCdenas.cpp**, **iterador.cpp**, **abb.cpp**, **pila.cpp**, **cola.cpp**, **aplicaciones.cpp**, **info.cpp**, **utils.cpp**, **palabras.cpp** y **principal.cpp** y enlazando los archivos objeto obtenidos.
El archivo **Makefile** (ver el material disponible sobre Makefile) provee para la compilación la regla **principal**, que es la predeterminada. Por lo tanto el ejecutable se obtiene mediante

```
$ make
```

Se sugiere probar cada uno de los test provistos y al final confirmar mediante la regla **testing**:

```
$ make testing
```

Se debe verificar la compilación y ejecución en las máquinas de Facultad ya que es con esas máquinas con las que se hace la evaluación.

4. Entrega

Ver la Sección **Entregas** de Reglamento del Laboratorio.

4.1. Plazos de entrega

El plazo para la entrega es el **lunes 30 de mayo a las 15:00**.

4.2. Archivo a entregar y procedimiento

Se debe entregar el archivo **Entrega4.tar.gz**, que contiene los módulos a implementar **cadena.cpp**, **colCadenas.cpp**, **iterador.cpp**, **abb.cpp**, **pila.cpp**, **cola.cpp** y **aplicaciones.cpp**.
Este archivo se obtiene al ejecutar la regla `entrega` del archivo *Makefile*:

```
$ make entrega
```

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

El archivo a entregar **DEBE** ser generado mediante este procedimiento. Si se lo genera mediante alguna otra herramienta (por ejemplo, usando un entorno gráfico) **la tarea no será corregida**, independientemente de la calidad del contenido. Tampoco será corregida si el nombre del archivo se modifica en el proceso de entrega. Si alguna de estas dos irregulardaides ocurre en la entrega la calificación será *Insuficiente* lo que obligará a hacer la reentrega. Si ocurre en la reentrega la calificación será `No Aprobado` lo que implicará la pérdida de la tarea, y por lo tanto del curso.
El archivo que queda en el receptor debe poder descomprimirse con el comando:

```
$ tar zxvf Entrega3.tar.gz
```

4.3. 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 */
```

4.4. Individualidad

Ver la Sección **Individualidad** de Reglamento del Laboratorio.

5. Descripción de los módulos y algunas funciones

En esta sección se describen los módulos que componen la tarea. Estos son *utils*, *info*, *palabras*, *cadena*, *colCadenas*, *iterador*, *abb*, *pila*, *cola*, *aplicaciones* y *principal*.

5.1. Módulo *utils*

Igual al de la tarea anterior.

5.2. Módulo *info*

Igual al de la tarea anterior.

5.3. Módulo *palabras*

Igual al de la tarea anterior.

5.4. Módulo *cadena*

Es el de la tarea anterior, al que se mueven operaciones que en la tarea 2 estaban en *aplicaciones*.

5.5. Módulo *colCadenas*

Es el de la tarea 2, pero la cantidad de cadenas de una colección es determinada al ser creada en lugar de por una constante.

5.6. Módulo *Iterador*

Igual al de la tarea anterior.

5.7. Módulo *abb*

Igual al de la tarea anterior.

5.8. Módulo *pila*

Pila de elementos de tipo TInfo.

5.9. Módulo *cola*

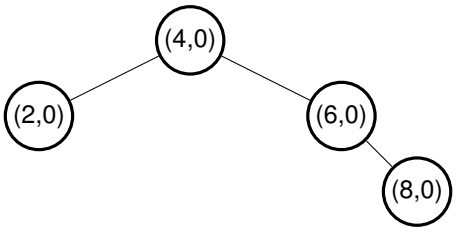
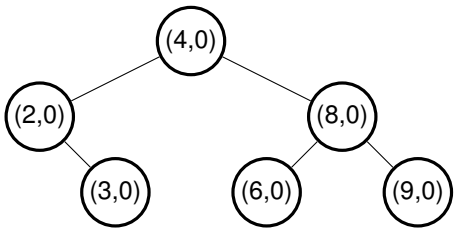
Cola de elementos de tipo TInfo.

5.10. Módulo *aplicaciones*

Es el de la tarea anterior al se vuelven a incluir operaciones que estaban en la tarea 2 y otras nuevas.
A continuación presentaremos mediante ejemplos algunas funciones representativas del módulo **abb**.

El árbol vacío se representa con ●.

5.10.1. *crearBalanceado(TInfo *arreglo, nat n)*

arreglo	Resultado
[]	●
[(8,0)]	
[(2,0),(4,0),(6,0),(8,0)]	
[(2,0),(3,0),(4,0),(6,0),(8,0),(9,0)]	

5.10.2. *ordenadaPorModulo(nat p, TCadena cad)*

La `TCola` resultado se representa con el frente como el elemento de más a la izquierda.

p	cadena	Resultado
8	[]	[]
8	[(14,2.0),(18,5.0),(10,3.0)]	[(18,5.0),(10,3.0),(14,2.0)]
6	[(14,2.0),(18,5.0),(10,3.0)]	[(18,5.0),(14,2.0),(10,3.0)]

Para entender los ejemplos es conveniente calcular los restos.

En el segundo ejemplo si los componentes naturales se sustiuyen por el resto de dividir entre 8 la cadena sería `[(6,2.0), (2,5.0), (2,3.0)]`.

En el tercer ejemplo si los componentes naturales se sustiuyen por el resto de dividir entre 6 la cadena sería `[(2,2.0), (0,5.0), (4,3.0)]`.

5.10.3. *menoresQueElResto(TCadena cad, nat cantidad*

La `TPila` resultado se representa con la cima como el elemento de más a la izquierda.

cadena	Resultado
[]	[]
[3,0]	[3,0]
[(10,0),(18,0),(10,1)]	[(10,1)]
[(9,0),(9,1),(10,0)]	[(10,0),(9,1)]

A. Tests de Tiempo

En esta tarea se incluyen tests de tiempo de ejecución.
En ellos el tamaño de la entrada es mayor a los que se usan para evaluar la funcionalidad. El propósito incial es evaluar que las funciones cumplan los órdenes de tiempo establecidos en la especificación. Es posible que además de cumplir con ese requerimiento sea necesario ajustar detalles que contribuyan a mejorar la eficiencia.

La ejecución del programa debe pasar los test en las máquinas *pcunix* de la facultad.

Los test de tiempo se llaman `tiempo-` seguido de lo que están evaluando. Puede ser una función o todo un módulo.

Los comandos para estos test tienen tres parámetros,

1. tamaño de la entrada
2. cantidad de iteraciones
3. timeout

Se puede ver en `principal.cpp` como se procesa cada comando.

El tercer parámetro, `timeout`, es diferente al que se usa en `Makefile`, en donde se establece una restricción al tiempo total de ejecución del caso. Este parámetro, en cambio, es en relación solo a la parte del tiempo de ejecución que se está evaluando. Por ejemplo en `tiempo_palabras`, el tiempo de construcción del árbol queda fuera de ese `timeout`.
El tiempo a evaluar es el que va desde la línea

```
        clock_t tm = clock();
```

(que está después de haber construido el árbol)
hasta la línea

```
      tm = clock() - tm;
```

porque solo se está evaluando el tiempo de ejecución de `buscarFinPrefijo`.

En esta tarea el `timeout` en los test es 2 (segundos). Este debería ser un tiempo holgado para cualquier implementación que cumpla los órdenes requeridos (se espera que el tiempo de ejecución sea de solo algunas décimas de segundo).

Tal vez sea convenienete en las primeras pruebas correr los tests con tamaños de entrada y cantidad de iteraciones (los dos primeros parámetros) más chicos.

Para saber que tan lejos se está de cumplir con el `timeout` se puede ver el tiempo de ejecución. Para eso se debe descomentar de manera provisoria en `principal.cpp` la línea

```
        // printf("%f \n", tiempo);
```

(hay una para cada comando)
Si se corre

```
$ ./principal < test/tiempo_TTTT.in
```

se verá el tiempo de ejecución del fragamento de código evaluado.

Por ejemplo, si se espera que el tiempo de ejecución tenga orden de crecimiento $O(n)$, entonces si se realizan sucesivas ejecuciones del comando anterior duplicando en cada una el tamaño de la entrada en el

archivo `.in` debería verse que el tiempo de ejecución (aproximadamente) también se duplica. Esto puede no cumplirse para tamaños de entrada muy chicos.