Programación II

Trabajo Práctico Obligatorio

2023 MANO

Algunas aclaraciones:

- El Trabajo Práctico Obligatorio (TPO) será desarrollado en forma grupal, de no más de 6 integrantes.
- Se deberá entregar un único proyecto exportado a zip que cuente con estrategia, código en lenguaje Java y estimación de costos, para cada uno de los ejercicios de implementación/uso. El proyecto debe contener los tres paquetes vistos durante la cursada, con toda interfaz, implementación y método externo utilizados. Para el caso de ejercicios de AVL y Árbol B se solicitará entregar un documento pdf con los diagramas explicativos, paso a paso.
- La fecha de entrega límite será el día de la defensa individual a las 23.59hs.
- Tiene disponible el archivo implementaciones.jar con las interfaces e implementaciones de los ocho TDA vistos en clase. Este archivo puede ser importado y utilizado para realizar las pruebas que sean necesarias.
- Para aprobar el presente TPO debe haberse entregado el proyecto y documento, y aprobada la defensa individual.
- El uso de sentencias "break" y "continue" no está permitido, así como tampoco de colecciones de Java. Las únicas estructuras que pueden utilizarse son arreglos, listas enlazadas definidas en clase, objetos definidos en el mismo ejercicio y cualquiera de los ocho TDA vistos.
- Se recomienda leer con detalle el enunciado antes de comenzar, y definir la estrategia antes de codificar.
- Se recomienda realizar comentarios dentro del código para facilitar su lectura.

Primera parte

Pila, Cola, Cola con Prioridad, Conjunto, Diccionario Simple, Diccionario Doble, ABB y Grafo.
Se solicita para cada ejercicio:

a) Describir la estrategia utilizada para el/los método/s. De recibir estructuras por parámetro, las mismas no deben verse modificadas.
   Se debe utilizar Javadoc, con las siguientes etiquetas:
   - tarea (breve descripción del método)
   - parámetros (parámetros de entrada, con su tipo y qué representa cada uno)
   - devuelve (valor de retorno, con su tipo y qué representa)
   - precondiciones
   - postcondiciones
   - costo

b) Escribir el código en lenguaje Java de el/los método/s. De recibir estructuras por parámetro, las mismas no deben verse modificadas.

c) Estimar la complejidad (se debe realizar una breve explicación que luego será agregada al Javadoc del inciso a).

Segunda parte

AVL, Árbol B.
Partiendo de un AVL/Árbol B, según sea el caso, se solicita realizar las operaciones explicitadas. Debe fundamentarse cada paso (incluyendo el nombre de la operación que interviene, si existiese) y realizarse el diagrama correspondiente paso a paso.

Primera parte – implementaciones:

1. Se define un nuevo TDA denominado ConjuntoEspecialTDA basado en ConjuntoTDA, con la particularidad de permitir determinar si las operaciones se realizan correctamente, o no. Algunos de sus métodos devuelven el objeto Respuesta, que contiene dos elementos: un booleano que determina la correctitud de ejecución y un entero que informa lo solicitado por el método en sí, si el método lo requiere y su ejecución fue satisfactoria. Su especificación se muestra en el anexo.
2. Se define un nuevo TDA denominado MultiPilaTDA basado en PilaTDA, con la particularidad de recibir una PilaTDA por parámetro al apilar (la misma debe apilarse a continuación de la multipila), y otra al desapilar (la misma debe chequear que los valores tope de la multipila coincidan para desapilar, sino no debe hacer nada). Tanto en el método apilar como en el método desapilar, ambas pilas vienen inicializadas y contienen cualquier cantidad de elementos (incluso cero). El método tope devuelve una PilaTDA con los primeros elementos de la multipila, se recibe por parámetro un número mayor o igual que cero, que representa la cantidad de ellos (de recibir un número superior a la cantidad de elementos de la multipila, debe devolver todos). Se solicita realizar la presente implementación con el TDA ya visto PilaTDA, o en su defecto con estructuras dinámicas (no puede realizarse la implementación con estructuras estáticas). Su especificación se muestra en el anexo.
3. Se busca implementar un DiccionarioSimpleTDA usando únicamente una ColaPrioridadTDA. Aclaración: se mantiene la interfaz de DiccionarioSimpleTDA; en la implementación en vez de utilizar un arreglo de enteros (estructura estática) o una lista enlazada (estructura dinámica), sólo puede usarse una ColaPrioridadTDA.

Primera parte – uso:

4. Se define un método que reciba una ColaTDA y devuelva una nueva ColaTDA con los elementos de la original, sin ninguna repetición. Se debe dejar el primer representante de cada uno de los repetidos, respetando el orden en que aparecen todos los elementos en la original.
5. Se define un método que reciba una PilaTDA y una ColaTDA y devuelva un ConjuntoTDA con los elementos comunes de la pila y de la cola.
6. Se define un método que reciba una PilaTDA y devuelva un DiccionarioSimpleTDA, en el cual se guardarán los elementos de la pila como claves, y la cantidad de apariciones de dicho elemento en la pila, como valores.
7. Se define un método que reciba un DiccionarioMultipleTDA y devuelva una ColaTDA con todos los valores del diccionario, sin ninguna repetición.
8. Se define un método que calcule la suma de los elementos con un valor impar de un ABB.
9. Se define un método que calcule la cantidad de hojas con un valor par de un ABB.
10. Se define un método que reciba un GrafoTDA y dos enteros que representen vértices, y devuelva un ConjuntoTDA con todos los vértices puente entre los vértices recibidos por parámetro. Se define que un vértice p es puente entre dos vértices o y d, si hay una arista que comienza en o y termina en p y otra que comienza en p y termina en d.
11. Se define un método que reciba un GrafoTDA y un entero que represente un vértice, y devuelva el grado del vértice recibido por parámetro. Se define el grado de un vértice v como el entero que es igual a la resta entre la cantidad de aristas que salen de v menos la cantidad de aristas que llegan a v.

Segunda parte – diagramas:

12. Partiendo de un AVL vacío, se solicita realizar las siguientes operaciones:
    a. Agregar el 20
    b. Agregar el 29
    c. Agregar el 21
    d. Agregar el 12
    e. Agregar el 7
    f. Eliminar el 20
13. Partiendo de un AVL vacío, se solicita realizar las siguientes operaciones:
    a. Agregar el 30
    b. Agregar el 36
    c. Agregar el 10
    d. Agregar el 15
    e. Agregar el 12
    f. Eliminar el 15
14. Partiendo de un Árbol B vacío de orden 4, se solicita realizar las siguientes operaciones:
    a. Agregar el 82
    b. Agregar el 12
    c. Agregar el 102
    d. Agregar el 36
    e. Agregar el 61
    f. Eliminar el 82
    g. Eliminar el 36
    h. Eliminar el 102
15. Partiendo de un Árbol B vacío de orden 5, se solicita realizar las siguientes operaciones:
    a. Agregar el 53
    b. Agregar el 62
    c. Agregar el 31
    d. Agregar el 105
    e. Agregar el 85
    f. Agregar el 55
    g. Eliminar el 105
    h. Eliminar el 62

Anexo - especificaciones:

```
public interface ConjuntoEspecialTDA {
    public class Respuesta {
        public boolean error;
        public int rta;
    }

    /**
     * Inicializa el conjunto.
     */
    public void inicializarConjunto();

    /**
     * Agrega un valor al conjunto.
     * La Respuesta devuelve el error en true si no se agrega un nuevo valor,
     * false si se agregó correctamente.
     */
    public Respuesta agregar(int valor);

    /**
     * Elimina un valor del conjunto.
     * La Respuesta devuelve el error en true si no se realiza una eliminación,
     * false si se eliminó el error.
     */
    public Respuesta sacar(int valor);

    /**
     * Devuelve un valor del conjunto,
     * si el conjunto no tenía valores, devuelve la Respuesta con error en true,
```

```
         * en caso contrario la Respuesta contiene el error en false y en rta el valor.
         */
        public Respuesta elegir();

        /**
         * Devuelve un booleano indicando si un valor pertenece al conjunto.
         */
        public boolean pertenece(int valor);

        /**
         * Devuelve un booleano indicando si el conjunto está vacío o no.
         */
        public boolean conjuntoVacio();

}


public interface MultiPilaTDA {
        /** Inserta la pila recibida en el tope de la multipila.
         * Si la multipila actualmente es: (tope) 3 - 5 - 7
         * Y la pila que se recibe es: (tope) 1 - 9
         * La multipila debe quedar: (tope) 1 - 9 - 3 - 5 - 7
         */
        public void apilar (PilaTDA valores);

        /** Desapila la pila recibida por parámetro de la multipila,
         * solo si el tope de la multipila coincide con la pila recibida.
         * Si la multipila actualmente es: (tope) 7 - 2 - 8 - 9
         * Y la pila que se recibe es: (tope) 7 - 2
         * La multipila debe quedar: (tope) 8 - 9
         * Si en cambio la pila que se recibe es: (tope) 7 - 2 - 3
         * No deben realizarse cambios en la multipila,
         * dado que no coincide con la pila recibida.
         */
        public void desapilar (PilaTDA valores);

        /** Devuelve una pila con los valores que estén en el tope de la multipila.
         * La cantidad de valores a devolver se define por parámetro y debe
         * preservarse el orden.
         * Si la cantidad es mayor al tamaño actual de la multipila,
         * se devuelven todos los valores de la multipila.
         * Si la multipila actualmente es: (tope) 4 - 2 - 9 - 7
         * Y se recibe por parámetro un 2, debe devolverse la pila: (tope) 4 - 2
         * Si se recibe por parámetro un 5, debe devolverse la pila: (tope) 4 - 2 - 9 - 7
         */
        public PilaTDA tope (int cantidad);

        /** Inicializa la pila */
        public void inicializarPila();

        /** Devuelve un booleano que indica si la pila está vacía */
        public boolean pilaVacia();
}
```