Programación II

Trabajo Práctico Obligatorio - 2024

Algunas aclaraciones:

- El Trabajo Práctico Obligatorio (TPO) será desarrollado en forma grupal, de no más de 6 integrantes.
- Tiene disponible el archivo P2lib.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 ser aprobada la **defensa individual** del mismo. En esta se solicitará la explicación de alguno/s de los ejercicios planteados en el mismo.
- El uso de colecciones de Java no está permitido. 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.

Primera parte:

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.
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, salvo en el caso de haber utilizado recursividad.

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, leer detenidamente los comentarios de cada método.

2. Se define un nuevo TDA denominado ConjuntoMamushkaTDA basado en ConjuntoTDA, con la particularidad de que se permite más de una acepción de cada elemento agregado. Tal cual como en ConjuntoTDA, no existe orden alguno. Su especificación se muestra en el anexo, leer detenidamente los comentarios de cada método.

3. 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, leer detenidamente los comentarios de cada método.

4. Se define un nuevo TDA denominado DiccionarioSimpleModTDA basado en DiccionarioSimpleTDA, con la particularidad de registrar la cantidad de veces que el valor se ve modificado (entrada: clave-valor-factorMod). Su especificación se muestra en el anexo, leer detenidamente los comentarios de cada método.

5. 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.

6. Se define un método que reciba una PilaTDA y devuelva un float (número real) con el porcentaje de cantidad de elementos pares de la pila.

7. Se define un método que reciba una PilaTDA y devuelva un ConjuntoTDA con los elementos repetidos de la pila.

8. 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.

9. 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.

10. 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.

11. Se define un método que reciba un DiccionarioMultipleTDA y devuelva una ColaTDA con todos los valores del diccionario, sin ninguna repetición.

12. Se define un método que calcule la suma de los elementos con un valor impar de un ABB.

13. Se define un método que calcule la cantidad de hojas con un valor par de un ABB.

14. 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.

15. 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:

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.

16. 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

17. 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 30

18. Partiendo de un AVL vacío, se solicita realizar las siguientes operaciones:
    a. Agregar el 8
    b. Agregar el 5
    c. Agregar el 6
    d. Agregar el 3
    e. Agregar el 1
    f. Eliminar el 3

19. 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

20. 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;
    }

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

    public Respuesta agregar(int valor); //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 sacar(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 elegir(); //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 boolean pertenece(int valor); //Devuelve un booleano indicando si
                //un valor pertenece al conjunto.

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


public interface ConjuntoMamushkaTDA {
        void inicializar( ); //inicializa el TDA

        void guardar(int dato); //agrega el elemento dato al TDA

        void sacar(int dato); //elimina del TDA una acepción del elemento dato

        int elegir( ); //muestra un elemento agregado al TDA (arbitrario)

        int perteneceCant(int dato); //devuelve la cantidad de veces que se
                //encuentra el elemento dato en el TDA

        boolean estaVacio( ); //devuelve un boolean que informa si el TDA está
                //vacío
}


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();
}

public interface DiccionarioSimpleModTDA {
        public void inicializarDiccionario(); //inicializa el TDA

        public void agregar(int clave, int valor); //agrega el par clave-valor al TDA,
                //conjuntamente con la cantidad de veces que dicho valor se vio
                //modificado. Si la clave se agrega por primera vez, el factor de
                //modificaciones será 0. Si la clave se agrega por segunda vez,
                //modificándose el valor, el factor de
                //modificaciones será 1. Y así sucesivamente.

        public void eliminar(int clave); //elimina la clave del TDA, con su valor y
                //factor de modificaciones

        public int recuperar(int clave); //devuelve el valor asociado a la clave, que
                //se supone existente

        public int recuperarMod(int clave); //devuelve la cantidad de
                //modificaciones que sufrió el valor relacionado a dicha clave, que se
                //supone existente

        public ConjuntoTDA claves(); //devuelve el conjunto de claves
}
```