UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

Estrategias de Programación y Estructuras de Datos

Idioma: ES

INSTRUCCIONES:
Estrategias de Programación y Estructuras de Datos. Junio 2025 · 2ª Semana

Los ejercicios que requieran programación se deberán realizar en lenguaje Java, utilizando los TADs de la asignatura (adjuntas a este enunciado están las interfaces de dichos TADs).

Los ejercicios de cálculo de coste requieren que se explicite cuál es el tamaño del problema. Si no se hace, la respuesta no se evaluará.

Todas las respuestas deberán justificarse; no se evaluarán respuestas sin justificar.

Interfaces de los TADs

CollectionIF
```java
public interface CollectionIF {
public int size();
public boolean isEmpty();
public boolean contains(E e);
public void clear();
}
```

SequenceIF
```java
public interface SequenceIF extends CollectionIF {
public IteratorIF iterator();
}
```

ListIF
```java
public interface ListIF extends SequenceIF {
public E get(int pos);
public void set(int pos, E e);
public void insert(int pos, E elem);
public void remove(int pos);
}
```

StackIF
```java
public interface StackIF extends SequenceIF {
public E getTop();
public void push(E elem);
public void pop();
}
```

QueueIF
```java
public interface QueueIF extends SequenceIF {
public E getFirst();
```

```
public void enqueue(E elem);
public void dequeue();
}
```

TreeIF
```java
public interface TreeIF extends CollectionIF {
public E getRoot();
public boolean isLeaf();
public int getNumChildren();
public int getFanOut();
public int getHeight();
public IteratorIF iterator(Object mode);
}
```

GTreeIF
```java
public interface GTreeIF extends TreeIF {
enum IteratorModes { PREORDER, POSTORDER, BREADTH }
public void setRoot(E e);
public ListIF> getChildren();
public GTreeIF getChild(int pos);
public void addChild(int pos, GTreeIF e);
public void removeChild(int pos);
}
```

BTreeIF
```java
public interface BTreeIF extends TreeIF {
enum IteratorModes { PREORDER, POSTORDER, BREADTH, INORDER, RLBREADTH }
public BTreeIF getLeftChild();
public BTreeIF getRightChild();
public void setRoot(E e);
public void setLeftChild(BTreeIF child);
public void setRightChild(BTreeIF child);
public void removeLeftChild();
public void removeRightChild();
}
```

BSTreeIF
```java
public interface BSTreeIF> extends TreeIF {
enum IteratorModes { DIRECTORDER, REVERSEORDER }
enum Order { ASCENDING, DESCENDING }

public BSTree getLeftChild();
public BSTree getRightChild();
public void add(E e);
public void remove(E e);
public Order getOrder();
}
```

Pregunta 1

Pregunta sobre la práctica.
Se desea programar una operación:
```java
ListIF getTasksBetweenDates(int dI, int dF)
```
que devuelva la lista de tareas que deben realizarse entre las fechas dI y dF, ambas incluidas, indicadas por los parámetros y que se encuentran almacenadas en el planificador de tareas futuras.
Como precondición asumiremos que dI < dF.

a) (1 punto) Programe la operación getTaskBetweenDates(dI, dF) de forma que sea independiente de la estructura escogida para implementar el planificador de tareas.

b) (1 punto) Calcule el coste asintótico temporal en el caso peor de la operación getTaskBetweenDates(dI, dF).

RESPUESTA DEL ESTUDIANTE:

```java
import java.util.ArrayList;
import java.util.List;

public class TaskPlanner {

private ListIF<TaskIF> tasks;

public TaskPlanner(ListIF<TaskIF> tasks) {
this.tasks = tasks;
}

public ListIF<TaskIF> getTasksBetweenDates(int dI, int dF) {
ListIF<TaskIF> result = new ArrayList<>();
for (TaskIF task : tasks) {
if (task.getDate() >= dI && task.getDate() <= dF) {
result.add(task);
}
}
return result;
}
}

// Assuming TaskIF has a getDate() method
interface TaskIF {
int getDate();
}

// Coste asintótico temporal en el caso peor: O(n)
// Donde n es el número total de tareas almacenadas en el planificador.
// En el peor caso, se debe iterar sobre todas las tareas para verificar si cada una se encuentra dentro del rango de fechas especificado.
```

Pregunta 2

Analice los siguientes fragmentos de código y determine su coste asintótico temporal en el caso peor:

a) (1,5 puntos)

```java
int i = 1;
while (i < n) {
System.out.println(i);
i *= 2;
}
```

b) (1,5 puntos)

```java
public static int dum(int n) {
if (n == 0) {
return 0;
} else if (n <= 3) {
return 1;
} else {
return dum(n-1) + dum(n-2) + dum(n-3);
}
}
```

RESPUESTA DEL ESTUDIANTE:

a) O(log n)

b) O(3^n)

Pregunta 3

Supongamos un juego por turnos con varios jugadores cuyo número puede disminuir durante el juego cuando alguno queda eliminado durante su turno. Cada jugador realiza una acción y luego pasa el turno al siguiente, en orden circular.

Se desea programar un Tipo de Datos que nos permita gestionar el orden en el que los jugadores van a tener su turno. Para ello, se necesitan dos operaciones:
- pasarTurno(): avanza el turno al siguiente jugador.
- eliminarJugador(): elimina al jugador actual del juego.

a) (0,5 puntos) Indique qué estructura de datos de las estudiadas en la asignatura sería la más adecuada, de forma que se optimice el coste de las dos operaciones anteriores. Indique, además, cómo se puede identificar al jugador actual en dicha estructura.

b) (1,5 puntos) Programe las operaciones pasarTurno() y eliminarJugador().

RESPUESTA DEL ESTUDIANTE:

La estructura de datos más adecuada es una lista doblemente enlazada circular. El jugador actual se identifica manteniendo un puntero al nodo que representa al jugador actual en la lista.

```java
public class CircularDoublyLinkedList<E> {

private Node<E> current;
private Node<E> head;
private int size;

private static class Node<E> {
E data;
Node<E> next;
Node<E> prev;

Node(E data) {
this.data = data;
}
}

public CircularDoublyLinkedList() {
head = null;
current = null;
size = 0;
}

public void add(E data) {
Node<E> newNode = new Node<>(data);
if (head == null) {
head = newNode;
newNode.next = newNode;
newNode.prev = newNode;
current = newNode;
```

```
} else {
newNode.next = head;
newNode.prev = head.prev;
head.prev.next = newNode;
head.prev = newNode;
}
size++;
}

public void pasarTurno() {
if (current != null) {
current = current.next;
}
}

public void eliminarJugador() {
if (current != null) {
if (size == 1) {
head = null;
current = null;
} else {
Node<E> nextNode = current.next;
current.prev.next = nextNode;
nextNode.prev = current.prev;
current = nextNode;
}
size--;
}
}

public int size() {
return size;
}

public E getCurrentPlayer() {
return (current != null) ? current.data : null;
}
}
```

Pregunta 4

Para que un árbol binario de búsqueda no equilibrado vuelva a cumplir las condiciones para ser un árbol AVL, es necesario aplicar rotaciones.

Secuencias:
a) 3, 1, 2
b) 3, 2, 1
c) 7, 15, 10
d) 11, 22, 35

Se pide construir los árboles, explicar la rotación necesaria y aplicarla.

RESPUESTA DEL ESTUDIANTE:

a) Árbol inicial:
```
3
/ \
1 2
```
Desbalance: El nodo 3 tiene un factor de equilibrio de -1 (altura izquierda 1, altura derecha 0).
Rotación: Rotación simple a la izquierda sobre el nodo 3.
Árbol resultante:
```
2
/ \
1 3
```

b) Árbol inicial:
```
3
/ \
2 1
```
Desbalance: El nodo 3 tiene un factor de equilibrio de 1 (altura izquierda 0, altura derecha 1).
Rotación: Rotación simple a la derecha sobre el nodo 3.
Árbol resultante:
```
2
/ \
3 1
```

c) Árbol inicial:
```
7
\
15
```

/
10
```
Desbalance: El nodo 15 tiene un factor de equilibrio de -1 (altura izquierda 1, altura derecha 0). El nodo 7 tiene un factor de equilibrio de 1 (altura izquierda 0, altura derecha 2).
Rotación: Rotación doble a la izquierda sobre el nodo 7 (primero rotación simple a la derecha sobre el nodo 15, luego rotación simple a la izquierda sobre el nodo 7).
Árbol resultante:
```
10
/ \
7 15
```

d) Árbol inicial:
```
11
\
22
\
35
```
Desbalance: El nodo 22 tiene un factor de equilibrio de -1 (altura izquierda 1, altura derecha 0). El nodo 11 tiene un factor de equilibrio de 1 (altura izquierda 0, altura derecha 2).
Rotación: Rotación doble a la derecha sobre el nodo 11 (primero rotación simple a la izquierda sobre el nodo 22, luego rotación simple a la derecha sobre el nodo 11).
Árbol resultante:
```
22
/ \
11 35
```
```