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 TaskFilter {

public static ListIF<TaskIF> getTasksBetweenDates(int dI, int dF) {
// This implementation is independent of the underlying data structure.
// It assumes the TaskIF interface has a method to filter tasks by date.
List<TaskIF> tasks = new ArrayList<>(); // Placeholder for the actual task list
// In a real implementation, this would retrieve tasks from the task scheduler.
// For this example, we'll just return an empty list.

// Simulate filtering based on dates (replace with actual filtering logic)
for (int i = dI; i <= dF; i++) {
tasks.add(new TaskIF() { // Dummy TaskIF implementation
@Override
public int getDate() {
return i;
}
});
}

return tasks;
}
}
```

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

El coste asintótico temporal en el peor de los casos es O(dF - dI + 1), que se puede simplificar a O(n), donde n es la diferencia entre las fechas dF y dI. Esto se debe a que el bucle `for` itera desde dI hasta dF, realizando una operación

(agregar una tarea) en cada iteración. La creación de la lista `tasks` tiene un coste O(n), pero este es dominado por el coste del bucle.

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(2^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:

a) Una lista enlazada circular sería la estructura de datos más adecuada. La identificación del jugador actual se puede realizar mediante un puntero al nodo actual de la lista, que apunta al siguiente nodo en la dirección circular.

b)

```java
public class PlayerOrder {
private Node head;
private int playerCount;

private class Node {
public PlayerOrder.Node(int player) {
this.player = player;
}
public int player;
public Node next;
}

public PlayerOrder(int count) {
this.head = new Node(0);
this.head.next = head;
this.playerCount = count;
}

public void pasarTurno() {
if (playerCount <= 1) {
return;
}
Node current = head;
Node previous = null;
do {
previous = current;
current = current.next;
} while (current != head);
```

```
head = current;
}

public void eliminarJugador(int playerToRemove) {
Node current = head;
Node previous = null;
do {
previous = current;
current = current.next;
} while (current != head);

if (previous.player == playerToRemove) {
previous.next = current.next;
} else {
// Player not found, do nothing
}
playerCount--;
}
}
```

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) 3, 1, 2

Árbol:
```
3
/ \
1 2
```
Rotación: Ninguna. El árbol es un BST equilibrado.

b) 3, 2, 1

Árbol:
```
3
/
2
/
1
```
Rotación: Rotación simple a la derecha en el nodo 3.

c) 7, 15, 10

Árbol:
```
7
/ \
15 10
```
Rotación: Ninguna. El árbol es un BST equilibrado.

d) 11, 22, 35

Árbol:
```
11
/ \
22 35
```

```
Rotación: Ninguna. El árbol es un BST equilibrado.
```