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
public ListIF<TaskIF> getTasksBetweenDates(int dI, int dF) {
ListIF<TaskIF> result = new List<>();
IteratorIF<TaskIF> it = scheduler.iterator();
while (it.hasNext()) {
TaskIF task = it.getNext();
int taskDate = task.getDate();
if (taskDate >= dI && taskDate <= dF) {
result.insert(result.size(), task);
}
}
return result;
}
```

b) El tamaño del problema es el número de tareas almacenadas en el planificador, denotado como n.

El coste asintótico temporal en el caso peor de la operación getTasksBetweenDates(dI, dF) es O(n).

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) El coste asintótico temporal en el caso peor es O(log n).

b) El coste asintótico temporal en el caso peor es 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:

a) La estructura de datos más adecuada es una cola circular implementada con una lista enlazada. El jugador actual se identifica mediante un puntero o índice que apunta al primer elemento de la cola.

b)

```java
public class TurnManager {
private ListIF<Integer> players;
private int currentPlayerIndex;

public TurnManager(ListIF<Integer> initialPlayers) {
this.players = initialPlayers;
this.currentPlayerIndex = 0;
}

public void pasarTurno() {
if (!players.isEmpty()) {
currentPlayerIndex = (currentPlayerIndex + 1) % players.size();
}
}

public void eliminarJugador() {
if (!players.isEmpty()) {
players.remove(currentPlayerIndex);
if (currentPlayerIndex == players.size()) {
currentPlayerIndex = 0;
}
}
}

public int getCurrentPlayer() {
return players.get(currentPlayerIndex);
}
}
```

```
```

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) Secuencia: 3, 1, 2

- Árbol inicial:

```
3
/
1
```

- Inserción de 2:

```
3
/
1
\
2
```

- Desequilibrio: Nodo 3 con factor de equilibrio -2 (izquierda más pesado).
- Rotación necesaria: Rotación doble derecha-izquierda (primero rotación derecha en 1, luego rotación izquierda en 3).
- Árbol equilibrado:

```
2
/ \
1 3
```

---

b) Secuencia: 3, 2, 1

- Árbol inicial:

```
3
/
2
```

- Inserción de 1:

```
3
```

/
2
/
1
```
- Desequilibrio: Nodo 3 con factor de equilibrio -2 (izquierda más pesado).
- Rotación necesaria: Rotación simple derecha en 3.
- Árbol equilibrado:
```
2
/ \
1 3
```

---

**c) Secuencia: 7, 15, 10**

- Árbol inicial:
```
7
\
15
```
- Inserción de 10:
```
7
\
15
/
10
```
- Desequilibrio: Nodo 7 con factor de equilibrio +2 (derecha más pesado).
- Rotación necesaria: Rotación doble izquierda-derecha (primero rotación izquierda en 15, luego rotación derecha en 7).
- Árbol equilibrado:
```
10
/ \
7 15
```

---

**d) Secuencia: 11, 22, 35**

- Árbol inicial:
```
11
\

22
```
- Inserción de 35:
```
11
\
22
\
35
```
- Desequilibrio: Nodo 11 con factor de equilibrio +2 (derecha más pesado).
- Rotación necesaria: Rotación simple izquierda en 11.
- Árbol equilibrado:
```
22
/ \
11 35
```
```