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:

Pregunta 1

a) Programación de la operación getTasksBetweenDates(dI, dF)

```java
public ListIF<TaskIF> getTasksBetweenDates(int dI, int dF) {
ListIF<TaskIF> result = new ListImplementation<TaskIF>(); // Suponiendo una implementación de ListIF
IteratorIF<TaskIF> it = planner.iterator(); // Suponiendo que planner es el planificador de tareas y tiene un método iterator()

while (it.hasNext()) {
TaskIF task = it.next();
int date = task.getDate(); // Suponiendo que TaskIF tiene un método getDate()

if (date >= dI && date <= dF) {
result.insert(result.size(), task); // Insertamos al final de la lista resultado
}
}

return result;
}
```

b) Cálculo del coste asintótico temporal

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

1. La inicialización de la lista resultado y el iterador es de coste constante, O(1).
2. El bucle while itera sobre todas las tareas en el planificador, lo que tiene un coste de O(n).
- Dentro del bucle, se realizan operaciones de coste constante: obtener la tarea actual (O(1)), obtener su fecha (O(1)), comparar la fecha con dI y dF (O(1)), e

insertar en la lista resultado (O(1) si se inserta al final y la lista es adecuada).

Por lo tanto, el coste total es O(n) porque se itera una vez sobre todas las tareas.

La respuesta final es:

a) La implementación solicitada está en el código proporcionado.

b) O(n) siendo n el número de tareas en el planificador.

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 es O(log n).

b) El coste asintótico temporal 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 para gestionar el orden de los jugadores es una cola circular o una lista circular doblemente enlazada, ya que permite un acceso y eliminación eficientes en cualquier posición. Sin embargo, dado que en la asignatura se han estudiado estructuras como listas, pilas, colas y árboles, y considerando las operaciones requeridas, una cola (QueueIF) sería una estructura adecuada si se implementa de manera circular. El jugador actual se puede identificar manteniendo una referencia al elemento actual en la cola.

b)

```java
public class Jugadores {
private QueueIF<Jugador> cola;
private Jugador actual;

public Jugadores() {
cola = new Queue(); // Suponiendo que Queue es una implementación de QueueIF
}

public void pasarTurno() {
actual = cola.getFirst();
cola.enqueue(cola.getFirst());
cola.dequeue();
}

public void eliminarJugador() {
cola.dequeue();
if (!cola.isEmpty()) {
actual = cola.getFirst();
} else {
actual = 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) La secuencia de inserción es 3, 1, 2.

1. Se inserta 3:
- Raiz: 3
- Izquierda: vacio
- Derecha: vacio

2. Se inserta 1:
- Raiz: 3
- Izquierda: 1
- Derecha: vacio

3. Se inserta 2:
- Raiz: 3
- Izquierda: 1
- Derecha: vacio
- Hijo derecho de 1: 2

Arbol resultante:
3
/
1
\
2

El factor de equilibrio del nodo 3 es -1 (altura izquierda = 2, altura derecha = 0) y el del nodo 1 es +1 (altura izquierda = 0, altura derecha = 1), lo que indica que el árbol está desequilibrado en el nodo 3. Se necesita una rotación derecha.

Aplicando la rotación derecha:
2
/ \
1 3

b) La secuencia de inserción es 3, 2, 1.

1. Se inserta 3:
- Raiz: 3

2. Se inserta 2:
- Raiz: 3
- Izquierda: 2
- Derecha: vacio

3. Se inserta 1:
- Raiz: 3
- Izquierda: 2
- Derecha: vacio
- Hijo izquierdo de 2: 1

Arbol resultante:
3
/
2
/
1

El factor de equilibrio del nodo 3 es -2 (altura izquierda = 3, altura derecha = 0) y el del nodo 2 es -1 (altura izquierda = 2, altura derecha = 0), lo que indica que el árbol está desequilibrado en el nodo 3. Se necesita una rotación derecha.

Aplicando la rotación derecha simple no es suficiente, se necesita una rotación doble (izquierda-derecha).

Primero, rotación izquierda en el nodo 3 (sobre su hijo izquierdo 2):
3
/
1
\
2

Luego, rotación derecha en el nodo 3:
2
/ \
1 3

Sin embargo, en este caso, directamente se puede aplicar la rotación derecha y queda:
2
/ \
1 3

c) La secuencia de inserción es 7, 15, 10.

1. Se inserta 7:
- Raiz: 7

2. Se inserta 15:
- Raiz: 7

- Izquierda: vacio
- Derecha: 15

3. Se inserta 10:
- Raiz: 7
- Izquierda: vacio
- Derecha: 15
- Hijo izquierdo de 15: 10

Arbol resultante:
7
\
15
/
10

El factor de equilibrio del nodo 7 es -1 (altura izquierda = 0, altura derecha = 2) y el del nodo 15 es +1 (altura izquierda = 1, altura derecha = 0), lo que indica que el árbol está desequilibrado en el nodo 7. Se necesita una rotación izquierda.

Aplicando la rotación izquierda:
7
\
10
\
15

No, directamente se aplica la rotación izquierda en el nodo 7 y queda:
10
/ \
7 15

d) La secuencia de inserción es 11, 22, 35.

1. Se inserta 11:
- Raiz: 11

2. Se inserta 22:
- Raiz: 11
- Izquierda: vacio
- Derecha: 22

3. Se inserta 35:
- Raiz: 11
- Izquierda: vacio
- Derecha: 22
- Hijo derecho de 22: 35

Arbol resultante:
11

\
22
\
35

El factor de equilibrio del nodo 11 es -3 (altura izquierda = 0, altura derecha = 3) y el del nodo 22 es -1 (altura izquierda = 0, altura derecha = 2), lo que indica que el árbol está desequilibrado en el nodo 11. Se necesita una rotación izquierda.

Aplicando la rotación izquierda simple no es suficiente, se necesita una rotación simple izquierda.

Primero, el árbol queda:
11
\
22
\
35

Aplicando la rotación:
22
/ \
11 35