NATIONAL UNIVERSITY OF DISTANCE EDUCATION

Estrategias de Programación y Estructuras de Datos

Idioma: EN

INSTRUCTIONS:
Programming Strategies and Data Structures. June 2025 · 2nd Week

Exercises that require programming must be done in Java, using the course ADTs (the interfaces for these ADTs are attached to this statement).

Cost-calculation exercises require explicitly stating the problem size. If this is not done, the answer will not be evaluated.

All answers must be justified; answers without justification will not be evaluated.

ADT Interfaces

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();
}
```
~~~

Question 1

Practice question.
It is required to program an operation:
```java
ListIF getTasksBetweenDates(int dI, int dF)
```
that returns the list of tasks to be performed between dates dI and dF, both included, specified by the parameters and stored in the future task scheduler.
As a precondition, assume that dI < dF.

a) (1 point) Implement getTaskBetweenDates(dI, dF) so that it is independent of the structure chosen to implement the task scheduler.

b) (1 point) Compute the worst-case asymptotic time cost of getTaskBetweenDates(dI, dF).

STUDENT RESPONSE:

The worst-case asymptotic time cost of getTaskBetweenDates(dI, dF) can be calculated as follows:

Time complexity: O(n^2)

Explanation:

The worst-case asymptotic time cost occurs when the input list is already sorted in ascending order. In this case, the algorithm performs a linear scan of the list and adds elements one at a time. The time complexity is O(n^2) because the worst-case scenario is when the input list is the smallest possible, meaning that all elements are added exactly once.

Therefore, the correct answer is:

The worst-case asymptotic time cost of getTaskBetweenDates(dI, dF) is O(n^2).

Question 2

Analyze the following code fragments and determine their worst-case asymptotic time cost:

a) (1.5 points)
```java
int i = 1;
while (i < n) {
System.out.println(i);
i *= 2;
}
```

b) (1.5 points)
```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);
}
}
```

STUDENT RESPONSE:

Question 2: Analyze the following code fragments and determine their worst-case asymptotic time cost:

a) (1.5 points)
```java
int i = 1;
while (i < n) {
System.out.println(i);
i *= 2;
}
```

b) (1.5 points)
```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);
}
}
```

The worst-case asymptotic time cost of these operations can be analyzed by examining the complexity of each step and applying the principles of big O notation.

In the first code fragment, the while loop iterates n times, where n is the maximum possible value of i. Since i ranges from 1 to n, the worst-case scenario occurs when n is the largest possible value. This means that the worst-case time cost is the same as the time cost of the loop itself, which is O(n).

In the second code fragment, the function dum(n) is called recursively for each value of n up to n-3. The worst-case scenario occurs when n-3 is the largest possible value. In this case, the time cost of the function is O(n^2), since it takes n^2 recursive calls to execute the function.

Therefore, the worst-case asymptotic time cost of the first code is O(n), while the worst-case asymptotic time cost of the second code is O(n^2).

Question 3

Suppose a turn-based game with several players whose number can decrease during the game when someone is eliminated during their turn. Each player performs an action and then passes the turn to the next player, in circular order.

You are asked to program a Data Type that lets us manage the order in which players take their turns. For this, two operations are needed:
- pasarTurno(): advances the turn to the next player.
- eliminarJugador(): removes the current player from the game.

a) (0.5 points) Indicate which data structure studied in the course would be the most appropriate, so that the cost of both operations is optimized. Also indicate how the current player can be identified in that structure.

b) (1.5 points) Implement pasarTurno() and eliminarJugador().

STUDENT RESPONSE:

The best choice is a balanced binary search tree because it provides a natural way to handle rotations to keep the tree in balance. Rotations can be applied to an unbalanced binary search tree to keep it in AVL tree form.

Implementing pasarTurno() and eliminarJugador():

```
// Implementation of pasarTurno()
public static void pasarTurno(int[] turn) {
int x = turn[turn.length - 1];
for (int i = turn.length - 2; i >= 0; i--) {
turn[i] = turn[i + 1];
}
turn[turn.length - 1] = x;
}

// Implementation of eliminarJugador()
public static void eliminarJugador(int[] turn) {
int x = turn[turn.length - 1];
for (int i = turn.length - 2; i >= 0; i--) {
turn[i] = turn[i + 1];
}
turn[turn.length - 1] = x;
}
```

Question 4: For an unbalanced binary search tree to meet the conditions of an AVL tree again, rotations must be applied.

Sequences:

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

You are asked to build the trees, explain the required rotation, and apply it.

Question 4

Given the following tree, indicate the sequence of nodes visited by preorder, postorder, and breadth-first traversals (from left to right).

1
/ | \n 2 3 4
/ \ |
5 6 7

Explain how each traversal is performed.

STUDENT RESPONSE:

To revert an unbalanced binary search tree to an AVL tree, the required rotations are as follows:

1. Rotate the root to the left to move it closer to its left child.
2. Repeat step 1 until the root is in the correct position.

Here is an example of how to perform the rotations:

a) The sequence 3, 1, 2 is an unbalanced binary search tree. To revert it to an AVL tree, we can start by rotating the root to the left.

b) First, we move the root to the left until it is closer to its left child.

c) Then, we repeat step 1 until the root is in the correct position.

d) Finally, we can continue rotating the root to