Estructuras de Datos Avanzadas
Tarea 3
Andrés Gómez de Silva Garza

Para resolver esta tarea debe:

- Basarse en mi solución al Proyecto 1 de Estructuras de Datos (calculadora), la cual se puede encontrar en ComunidadITAM y fue desarrollada en Java con NetBeans.
- Traducir las partes relevantes de dicho proyecto (ver abajo) a Python y hacerle modificaciones a dicha solución para procesar las expresiones aritméticas (ingresadas por el usuario en notación infija) en un solo paso mediante el uso de árboles de expresión, en lugar de mediante las dos fases (conversión de notación infija a postfija y evaluación de la expresión en notación postfija) de la solución original.
- Para esto habrá que definir la clase BinaryTree. La especificación de la funcionalidad de la interfaz BinaryTreeADT<T>, escrita en Java, se muestra más abajo. En lugar de incluir los cuatro métodos que regresan iteradores mostrados en la especificación, en la versión en Python cada uno de estos métodos puede sustituirse por un método de recorrido equivalente que debe regresar una lista en la que estén incluidos los datos de los nodos en el orden en el que se visitarían usando los cuatro tipos de recorridos correspondientes (pues las listas en Python son objetos iterables).
- También habrá que definir la clase ExpressionTree como subclase de la clase mencionada en el punto anterior. La funcionalidad mínima que debe tener la clase ExpressionTree son los métodos *evaluateTree* (que regresa el valor final que resulte de evaluar todos los nodos de un árbol de expresión) y *evaluateNode* (que regresa el valor numérico final que resulte de evaluar un nodo específico de un árbol de expresión…nótese que evaluar un nodo hoja implica simplemente obtener el valor numérico de ese nodo, mientras que evaluar un nodo interno implica evaluar—aplicando recursivamente esta misma definición—los hijos izquierdo y derecho del nodo y aplicar el operador almacenado en el nodo a los valores numéricos que resulten de dichas evaluaciones).
- Hacer pruebas con el código resultante para asegurarse de que siga funcionando correctamente la calculadora (ver abajo).
- Hacer pruebas con el código resultante para asegurarse de que los demás métodos implementados (por ejemplo, los métodos *iteratorInOrder*, etc.), aunque no se hayan usado en la funcionalidad de la calculadora, estén funcionando correctamente (por ejemplo, antes de evaluar el árbol de expresión final producido por su algoritmo, pueden imprimir dicho árbol e imprimir los resultados de hacer cuatro tipos de recorridos de él para evaluar el funcionamiento correcto de los cuatro tipos de recorridos).

La interfaz BinaryTreeADT<T> en Java es la siguiente:

```
public interface BinaryTreeADT<T> {
    public BinaryTreeNode<T> getRoot();
    public void setRoot(BinaryTreeNode<T> node);
    public boolean isEmpty();
    public int size();
    public boolean contains(T data);
    public BinaryTreeNode<T> find(T data);
    public Iterator<T> iteratorInOrder();
    public Iterator<T> iteratorPreOrder();
    public Iterator<T> iteratorPostOrder();
    public Iterator<T> iteratorLevelOrder();
    public String toString();
}
```

Nota: el ADT anterior supone que el árbol se va a representar con nodos enlazados (donde cada nodo es una instancia de BinaryTreeNode<T>). En caso de que se implementen con arreglos, los métodos *getRoot* y *find* deben regresar un *int* (la posición del arreglo en la que se encuentra el nodo raíz o el nodo que contiene el dato que se busca, respectivamente). También el método *setRoot* debe recibir un *int* (el índice en el que está almacenado el dato que se desea que se vuelva la raíz). Por otra parte, una de las representaciones posibles de árboles usando arreglos fija la posición de la raíz en el índice 0, por lo que regresar y recibir un *int* en *getRoot* y *setRoot*, respectivamente, sería redundante en este caso. Debido a estas diferencias posibles en la representación de los árboles y por ende el tipo de valores que tendrían que regresar o recibir los métodos, el ADT ya no sería tan genérico (ya no describiría los árboles binarios en general, sino los árboles binarios implementados con nodos enlazados (en el caso de la versión mostrada)).

La interfaz gráfica del proyecto de NetBeans se puede descartar de la versión en Python, la cual debe enfocarse exclusivamente en la parte que procesa (analiza, convierte y evalúa) expresiones aritméticas. En un documento aparte se explica cómo usar un árbol de expresión para reemplazar las dos fases de conversión y evaluación del proyecto de NetBeans original por una sola fase. Suponiendo que en Python han escrito una función *analiza_expresion* que regresa un valor booleano que indica si una cadena de caracteres representa una expresión aritmética válida, así como una función llamada *convertir_y_evaluar* que regresa un valor real que resulte de evaluar una expresión, incluya la siguiente función *procesa_expresion* en su programa de Python y realice al menos las seis pruebas mostradas abajo:

```
def procesa_expresion(exp):
    if analiza_expresion(exp):
        print("Resultado de evaluar la expresión: ",convertir_y_evaluar(exp))
    else:
        print("La expresión no es válida.")

#Pruebas:
#Deberían alternase fracaso-éxito-fracaso-éxito, etc.:
procesa_expresion("24-*5")
procesa_expresion("2+3")
procesa_expresion(")23*4(")
procesa_expresion("17*(2-4)-((67-18)/4)")
```

```
procesa_expresion("12..5-17")
procesa_expresion("7")
```

Para imprimir de manera amigable el contenido de un árbol binario se puede usar el siguiente algoritmo (mostrado en Java, pero tendrá que traducirlo a Python) dentro de la clase BinaryTree:

```
    // Adapted from: http://www2.hawaii.edu/~ztomasze/ta/ics211-1-
fa06/a4/BinaryTree.java:
    public String toString() {
        return toStringTree(getRoot(),0);
    }

    // Returns a multi-line string representation of the (sub-)tree rooted at
    // the given node:
    private String toStringTree(BinaryTreeNode<T> node,int indent) {
        int i;
        StringBuilder indentation=new StringBuilder(""),result=new
StringBuilder("");

        if(node!=null) {
            for(i=0;i<indent;i=i+1)
                indentation.append(" ");
            if(isLeaf(node)) {
                result.append("[");
                result.append(node.getData());
                result.append("]\n");
            }
            else {
                result.append("(");
                result.append(node.getData());
                result.append(")\n");
                if(node.getLeftChild()!=null) {
                    result.append(indentation);
                    result.append("Izq.: ");
                    result.append(toStringTree(node.getLeftChild(),indent+2));
                }
                if(node.getRightChild()!=null) {
                    result.append(indentation);
                    result.append("Der.: ");
                   result.append(toStringTree(node.getRightChild(),indent+2));
                }
            }
        }

        return result.toString();
    }
```

Su solución, como siempre, debe diseñarse respetando los principios del software de calidad: debe ser robusta, eficiente, modular, legible, amigable para el usuario, general, etc.