Cátedra: Estructuras de Datos

Carrera: Licenciatura en Sistemas

Año: 2022 – 2do Cuatrimestre

Facultad de Ciencias de la Administración
Universidad Nacional de Entre Ríos

Trabajo Práctico Nro 4:

Árboles Binarios y Heaps

Condiciones de Entrega

- El trabajo práctico deberá ser:
  - Realizado en forma individual o en grupos de no más de tres alumnos.
  - Publicado en un repositorio en GitHub creado por el equipo de trabajo cuya URL deberá especificarse en la Actividad del Campus Virtual o cargado en la sección del Campus Virtual correspondiente, junto con los archivos de código fuente que requieran las consignas. Cada uno de ellos en sus respectivos formatos pero comprimidos en un único archivo con formato zip, rar, tar.gz u otro.
    - En caso de realizar el Trabajo Práctico en grupo, deberá indicarse el apellido y nombre de los integrantes del mismo.
  - Ser entregado el **Lunes 31 de Octubre de 2022.**
- El práctico será evaluado con nota numérica y conceptual (Excelente, Muy Bueno, Bueno, Regular y Desaprobado), teniendo en cuenta la exactitud y claridad de las respuestas, los aportes adicionales fuera de los objetivos de los enunciados y la presentación estética.
- Las soluciones del grupo deben ser de autoría propia. Aquellas que se detecten como idénticas entre diferentes grupos serán clasificadas como **MAL** para todos los involucrados en esta situación que será comunicada en la devolución.
- Los ejercicios que exijan codificación se valorarán de acuerdo a su exactitud, prolijidad (identación y otras buenas prácticas).

Consideraciones

El equipo de cátedra considera que la corrección de los Trabajos Prácticos presentados por los alumnos es más ágil si no se programan interfaces de consola de comandos que exigen al usuario interactuar con la aplicación. Por tanto, se solicita no programar clases cliente que hagan uso de la función **`input()`** o similares.

Ejercicios de Programación

1. Programe la clase **`LinkedBinaryTreeExt`** como clase derivada de **`LinkedBinaryTree`** y **`LinkedBinaryTreeExtAbstract`** que se detalla a continuación:

```
from abc import ABC, abstractmethod
from typing import Any, List, Union
from data_structures import BinaryTreeNode

class LinkedBinaryTreeExtAbstract(ABC):
    """ Conjunto de métodos adicionales a LinkedBinaryTree"""

    @abstractmethod
    def hermanos(self, nodo1 : BinaryTreeNode, nodo2: BinaryTreeNode) -> bool:
        """ Indica si node1 y node2 son hermanos.

        Args:
            nodo1 (BinaryTreeNode): nodo que debe pertenecer al árbol.
            nodo2 (BinaryTreeNode): nodo que debe pertenecer al árbol.

        Returns:
            bool: True si los nodos tienen el mismo padre, False en caso contrario.
        """
        pass

    @abstractmethod
    def hojas(self) -> List[Any]:
        """ Devuelve los elementos de los nodos que no tienen ningún hijo.

        Returns:
            List[Any]: lista formada por los elementos de los nodos hoja.
        """
        pass

    @abstractmethod
    def internos(self) -> List[Any]:
        """ Devuelve los elementos de los nodos que tienen padre y algún hijo.

        Returns:
            List[Any]: lista formada por los elementos de los nodos internos.
        """
        pass

    @abstractmethod
    def profundidad(self, nodo : BinaryTreeNode) -> int:
        """ Devuelve la longitud del camino entre la raíz y un nodo.

        Args:
            nodo (BinaryTreeNode): nodo del que se quiere conocer la profundidad.

        Returns:
            int: devuelve el número de arcos entre la raíz y nodo. 0 si nodo es la raíz.
        """
        pass

    @abstractmethod
    def altura(self, nodo : BinaryTreeNode) -> int:
        """ Retorna la longitud del camino entre nodo y la hoja más lejana.

        Args:
            nodo (BinaryTreeNode): nodo del que se quiere conocer la altura.

        Returns:
            int: Devuelve 0 en caso que nodo sea hoja, caso contrario, la cantidad de arcos
                 entre nodo y la hoja más lejana.
```

```
        """
        pass
```

2. Programe en un archivo con nombre **`linked_binary_tree_ext_client.py`** un cliente para **`LinkedBinaryTreeExt`** donde se pongan a prueba **TODOS** los métodos definidos en esa clase con el siguiente árbol binario:

```
from data_structures import BinaryTreeNode
from linked_binary_tree_ext import LinkedBinaryTreeExt

nodo_a = BinaryTreeNode('A')
nodo_b = BinaryTreeNode('B')
nodo_c = BinaryTreeNode('C')
nodo_d = BinaryTreeNode('D')
nodo_f = BinaryTreeNode('F')
nodo_g = BinaryTreeNode('G')
nodo_h = BinaryTreeNode('H')
nodo_i = BinaryTreeNode('I')
nodo_k = BinaryTreeNode('K')
nodo_m = BinaryTreeNode('M')
nodo_n = BinaryTreeNode('N')

arbol = LinkedBinaryTreeExt()
arbol.add_left_child(None, nodo_a)

arbol.add_left_child(nodo_a, nodo_b)
arbol.add_right_child(nodo_a, nodo_f)

arbol.add_left_child(nodo_b, nodo_c)
arbol.add_right_child(nodo_b, nodo_d)

arbol.add_left_child(nodo_f, nodo_g)
arbol.add_right_child(nodo_f, nodo_k)

arbol.add_left_child(nodo_g, nodo_h)
arbol.add_right_child(nodo_g, nodo_i)

arbol.add_left_child(nodo_k, nodo_m)
arbol.add_right_child(nodo_k, nodo_n)

print(arbol)
```

3. Una **`Cola de Prioridad`** es una estructura de datos similar a una **`Cola`** donde los elementos tienen una prioridad asignada. Los elementos con menor prioridad serán procesados primero. Construya la clase **`UnsortedPriorityQueue`** que utilizando una **`lista Python no ordenada`** extienda la clase que se detalla a continuación:

```
from abc import ABC, abstractmethod
from typing import Any, Tuple

class UnsortedPriorityQueueAbstract(ABC):
    """Cola de prioridad mínima no ordenada utilzando representación posicional."""

    @abstractmethod
    def __len__(self) -> int:
        """ Devuelve la cantidad de elementos en la estructura.

        Returns:
            int: Cantidad de elementos en la estructura. 0 en caso que esté vacía.
        """
        pass

    @abstractmethod
    def is_empty(self) -> bool:
        """ Indica si la estructura está vacía o no.
```

```
        Returns:
            bool: True si está vacía. False en caso contrario.
        """
        pass

    @abstractmethod
    def add(self, k: Any, v: Any) -> None:
        """ Inserta un nuevo ítem al final de la estructura.

        Args:
            k (Any): Clave que determina la prioridad del ítem.
            v (Any): Valor del ítem.
        """
        pass

    @abstractmethod
    def min(self) -> Tuple[Any]:
        """ Devuelve una tupla conformada por la clave y valor del ítem con menor valor de
                clave.
         Raises:
            Exception: Arroja excepción si se invoca cuando la estructura está vacía.

        Returns:
            Tuple[Any]: Tupla de dos elementos: Clave y Valor del ítem.
        """
        pass

    @abstractmethod
    def remove_min(self) -> Tuple[Any]:
        """ Quita de la estructura el ítem con menor valor de clave.
         Raises:
            Exception: Arroja excepción si se invoca cuando la estructura está vacía.

        Returns:
            Tuple[Any]: Tupla de dos elementos: Clave y Valor del ítem.
        """
        pass
```

4. Programe en un archivo con nombre **`unsorted_priority_queue_client.py`** un cliente para **`UnsortedPriorityQueue`** donde se pongan a prueba **TODOS** los métodos definidos en esa clase.

5. Implemente a través de una clase con nombre **`PriorityQueueStack`** una estructura de datos Pila (Ed. LIFO) utilizando solamente una cola de prioridad y una variable de instancia adicional de tipo **`int`**.

6. Programe en un archivo con nombre **`pq_stack__client.py`** un cliente para **`PriorityQueueStack`** donde se pongan a prueba **TODOS** los métodos definidos en esa clase.

7. Implemente a través de una clase con nombre **`HeapQueue`** una estructura de datos Cola (Ed. FIFO) utilizando un Heap. Nota: No es necesario implementar **`__str__()`** ni **`__repr__()`**

8. Programe en un archivo con nombre **`heap_queue__client.py`** un cliente para **`HeapQueue`** donde se pongan a prueba **TODOS** los métodos definidos en esa clase.

Proyectos de Programación (opcional)

1. **Posición vs. Enlaces:** Implemente un Heap mínimo utilizando representación por enlaces.