Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2233 Programación Avanzada (2019-1)

Tarea XX

Entrega

- **Tarea**
  - **Fecha y hora:** domingo 30 de junio de 2019, 20:00
  - **Lugar:** GitHub — Carpeta: `Tareas/TXX/`

Objetivos

- Implementar una estructura de datos con un comportamiento específico.
- Recorrer una estructura de datos.
- Identificar caminos en una estructura de datos.
- Leer, cargar y guardar archivos JSON.
- Modificar una estructura de datos.
- Aplicar el conocimiento de excepciones, para levantar excepciones personalizadas.

Índice

1. Un DCCuento de Navidad

En el DCC existe una gran preocupación por la disonancia entre la típica decoración de Navidad y la estación del año, por lo que se decidió cambiar la fecha de celebración de la Navidad al **25 de junio**. Para la decoración de la festividad, los distintos cuerpos de ayudantes se comprometieron con determinados elementos y Programación Avanzada, el mejor cuerpo de ayudantes de todo el departamento, se comprometió con el árbol de Navidad. A falta de pinos, se decidió traer y decorar un Algarrobo.

Figura 1: Primer intento (fallido) de decoración del Algarrobo

Pero decorar el árbol no es tan fácil. Para aquello, hay que distribuir muchos adornos numerados de igual peso y al mismo tiempo mantener la forma balanceada del árbol. Los adornos se entregan desordenados y se necesita de tu ayuda como **experto programador** para determinar cómo distribuir los adornos de tal forma que queden ordenados y para que el Algarrobo se mantenga balanceado.

2. `AlgarroboTree`

Para esta evaluación, deberás implementar una estructura de datos y utilizarla para resolver una serie de funciones. En particular, implementarás el árbol **binario** llamado `AlgarroboTree`.

2.1. Reglas de un `AlgarroboTree`

Las reglas para que un árbol sea llamado `AlgarroboTree` son:

- Todo nodo de un `AlgarroboTree` tiene referencia a otros dos nodos: el hijo derecho y el hijo izquierdo.
- Cada nodo encapsula un valor de tipo `int`.
- Para cada nodo, su hijo derecho es `None` u otro nodo del árbol.
- Para cada nodo, su hijo izquierdo es `None` u otro nodo del árbol.
- El valor que guarda el nodo padre **es menor** a los valores que tienen sus hijos. Entre los hijos, no hay ninguna condición para sus valores. El hijo derecho puede ser menor al izquierdo o viceversa.

- Para que un nivel cualquiera del árbol contenga nodos, todos los niveles anteriores deben estar llenos de nodos. En forma de ejemplo, si el nivel 2 tiene un nodo, entonces el nivel 0 y nivel 1 deben tener todos los nodos posibles. A continuación se muestra como son los niveles en un `AlgarroboTree`.

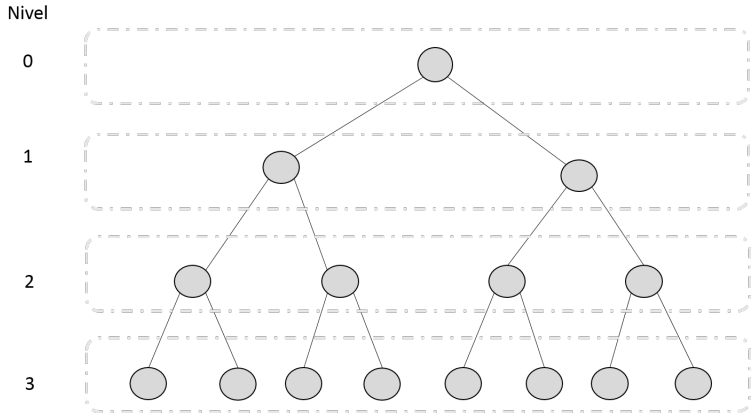

Figura 2: Niveles en un `AlgarroboTree`

- Todos los niveles se llenan de izquierda a derecha. En otras palabras, se puede decir que a excepción del primer nodo de cada nivel (de izquierda a derecha), todos los demás deben tener un nodo vecino izquierdo.

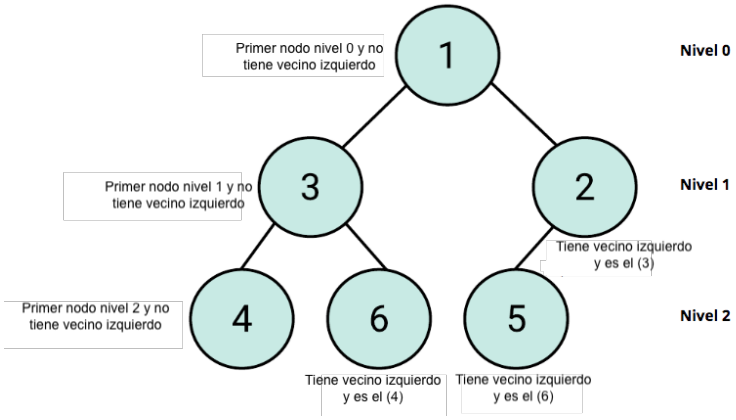

Figura 3: Explicación de primer nodo por nivel y nodos vecinos en un `AlgarroboTree`

A continuación se muestra una lista de árboles que **cumplen** con ser `AlgarroboTree`:

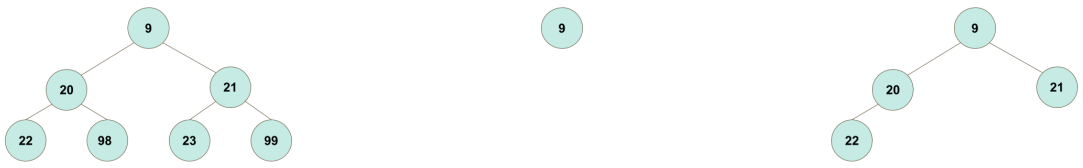

Figura 4: Ejemplos de tres `AlgarroboTree`

A continuación se muestra una lista de árboles que **no cumplen** con ser `AlgarroboTree`:

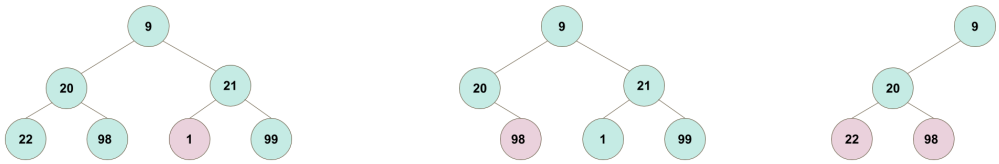

Figura 5: Ejemplos de tres árboles que no son `AlgarroboTree`

El primer árbol tiene un nodo (1) cuyo valor no es mayor al de su padre. El segundo árbol tiene un nodo (98) que no es el primero del nivel y no tiene vecino izquierdo. El último árbol tiene dos nodos (22 y 98) en el nivel 2, pero el nivel 1 no está completo.

2.2. Inserción

El proceso para insertar un dato en un `AlgarroboTree` consiste en siempre llenar un nivel de izquierda a derecha antes de insertar en el siguiente nivel[1], y reordenar después si es necesario. Para entender más este proceso, se muestra un ejemplo de inserción, pero en el caso simple donde no es necesario reordenar:

Figura 6: Inserción de un nodo con valor 11.

En la figura 6, el árbol de la izquierda es el estado inicial y se inserta el valor 11, obteniendo la imagen de la derecha. Como el nivel 2 no está lleno, se completa con el primer nodo de izquierda a derecha que sea `None`.

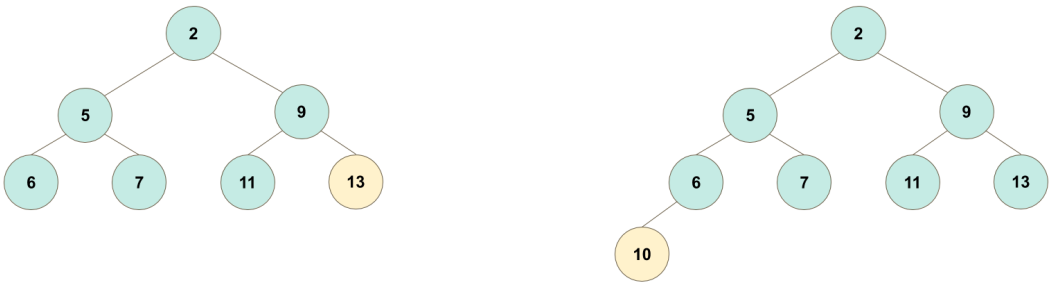

Figura 7: Inserción de nodos con valores 13 y 10.

En la figura 7, se inserta el valor 13 (imagen de la izquierda). Como el nivel 2 no está lleno, se completa con el primer nodo de izquierda a derecha que sea `None`. Finalmente, se inserta el valor 10 (imagen de la

[1]Este proceso se ve similar a como se recorre con BFS.

derecha). Como el nivel 2 ya está lleno, se inicia con el nivel 3.

En los ejemplos anteriores, se cumple inmediatamente la regla de orden de valores en los nodos: 11 es mayor que 9, 13 es mayor que 9 y 10 es mayor que 6. Pero esto no siempre es así necesariamente, por ejemplo, si se inserta el valor 1, este correspondería ser insertado como el hijo derecho del nodo con valor 6. Pero, 1 es menor que 6, por lo que tras insertarlo es necesario reordenar valores o realizar *swapping* (se explica a continuación). Puedes asumir que no enfrentarás el caso cuando el valor a insertar es igual al valor de algún otro nodo, es decir, siempre trabajaremos con valores únicos.

2.3. *Swapping*

Para poder arreglar un `AlgarroboTree` luego de insertar un dato, se utiliza la técnica de *swapping*, lo cual implica intercambiar el valor que tiene un hijo con su padre en caso de que el valor del hijo sea menor al padre. Este proceso se inicia desde el nodo que fue insertado y se va realizando tantas veces sea necesario hasta que el valor del nodo ya no puede subir más porque el padre tiene un valor menor o se llegó hasta el nodo raíz. A continuación se muestra un ejemplo donde se inserta un dato y se hace *swapping* para mantener las propiedades de un `AlgarroboTree`.

Figura 8: Inserción con valor 8.

En la figura 8, se inserta el valor 8. Se debe hacer *swap* de los valores 8 y 9 para que se cumpla que el padre es menor a sus hijos. Después de hacer *swap* (imagen de la derecha), la propiedad de orden se cumple para todos los nodos, y se finaliza.

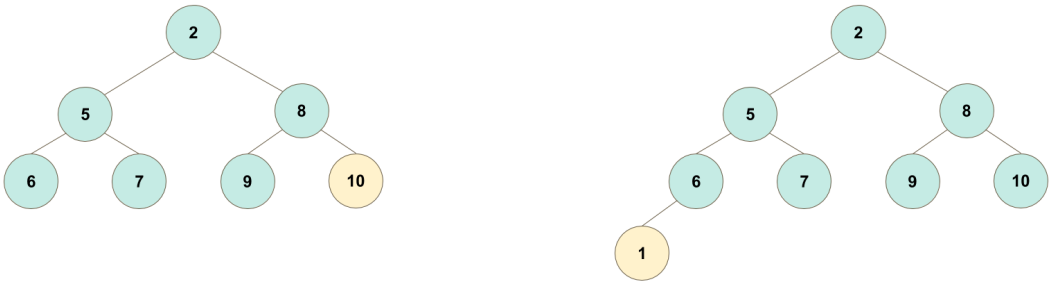

Figura 9: Inserción de nodos con valores 10 y 1.

En la figura 9, se inserta el valor 10 (imagen de la izquierda). No se necesita hacer *swap* porque el padre del 10 es menor a 10. Luego se inserta el valor 1 (imagen de la derecha). Se debe hacer *swap* entre 1 y 6 para que se cumpla que el padre es menor a sus hijos.

En la figura 10, se hace *swap* entre 1 y 6 (imagen de la izquierda). Se debe hacer *swap* nuevamente, pero esta vez entre 1 y 5 (imagen de la derecha).

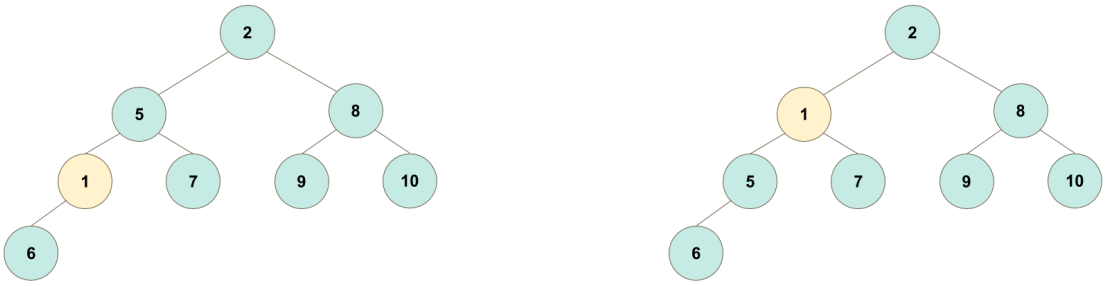

Figura 10: *Swapping* tras inserción de nodo con valor 1.

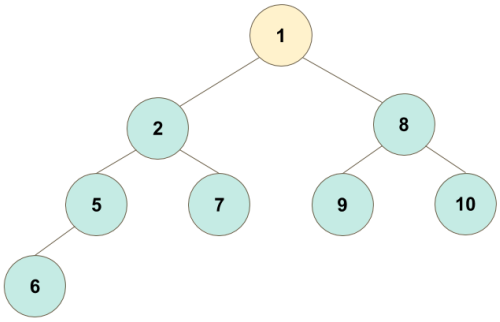

Figura 11: Último *swap* tras insersión de nodo con valor 1.

Finalmente en la figura 11, se realiza el *swap* de nuevo entre 1 y 2, para que se cumpla que el padre es menor a sus hijos. El nodo con valor 1 quedó como raíz así que ya no es necesario realizar más *swaps*.

3. Consultas

Para esta tarea, deberás completar tres funciones, de las cuales, al menos las últimas dos deben funcionar con tu implementación de la estructura de datos.

3.1. `search(tree_path: str, value: int)` → `[int]`

El primer argumento representa el *path* a un archivo JSON el cual contiene información de un árbol y el segundo argumento representa un valor `int` a buscar dentro del árbol. Esta función debe primero verificar si el árbol entregado posee las propiedades de un `AlgarroboTree` y luego buscar la ruta desde el nodo raíz hasta el nodo que contenga el valor entregado en `value`. Finalmente, la función retorna una lista con la ruta encontrada. Esta lista contiene los valores que guarda cada nodo de la ruta encontrado, por lo tanto es una lista de `int`, no de nodos.

El formato del archivo JSON indicado en el argumento `tree_path` estructura cada nodo como un objeto JSON que anida otros nodos dentro de el. El formato es:

```
{
    "value": valor_en_nodo,
    "left": {
```

```
        "value": valor_en_hijo_izquierdo,
        "left": {...},
        "right": {...}
    },
    "right": {
        "value": valor_en_hijo_derecho,
        "left": {...},
        "right": {...}
    }
}
```

Los campos `"left"` y `"right"` especifican información de los nodos hijos izquierdo y derecho respectivamente, también como objetos JSON. En caso de que un nodo tenga como hijo a `None`, este se escribe como `"None"` en este archivo. Puedes asumir que **todos los valores en nodos serán únicos** y el archivo a abrir vendrá siempre con el formato descrito anteriormente. Por ejemplo, dado el siguiente árbol:

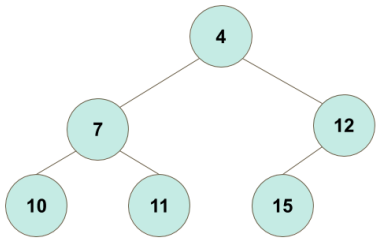

Figura 12: Árbol con los elemento `[10, 4, 12, 7, 11, 15]`

El diccionario que lo representa es:

```
{
    "value": 4,
    "left": {
        "value": 7,
        "left": {
            "value": 10,
            "left": "None",
            "right": "None"
        },
        "right": {
            "value": 11,
            "left": "None",
            "right": "None"
        }
    },
    "right": {
        "value": 12,
        "left": {
            "value": 15,
            "left": "None",
            "right": "None"
```

```
        },
        "right": "None"
    }
}
```

En esta consulta, pueden ocurrir ciertos errores como: el *path* del archivo es inválido, el árbol no cumple con las propiedades de un `AlgarroboTree` o el número buscado no existe en el árbol. Por lo tanto, te debes encargar de detectar estos casos con las siguientes excepciones personalizadas que fueron entregadas en el archivo `exceptions.py` y levantarlas[2] cuando se identifican:

- `NotFoundFile`: se levanta cuando el *path* entregado no corresponde a un archivo. Esta excepción contiene un atributo `path` que indica cual es el *path* inválido.
- `InvalidTree`: se levanta cuando el árbol entregados para cargar no cumplen con las propiedades de un `AlgarroboTree`. Esta excepción contiene un atributo `tree` que corresponde a un `int` con el valor del raíz del árbol que no cumple con ser `AlgarroboTree`.
- `NotFoundValue`: se levanta cuando el valor a buscar no existe en el `AlgarroboTree`. Esta excepción no contiene ningún atributo.

Los pasos para ejecutar esta consulta son, en el siguiente orden:

1. Verificar que el *path* corresponda a un archivo existente, sino levantar la excepción `NotFoundFile`
2. Verificar que el árbol cumpla con la propiedades de un `AlgarroboTree`, sino levantar la excepción `InvalidTree`
3. Verificar que el número a buscar esté el árbol, sino levantar la excepción `NotFoundValue`.
4. Retornar la lista desde el nodo raíz hasta el nodo que contenga el valor buscado.

3.2. `build_and_save(data: [int], output_path: str)` → `None`

El primer argumento representa una lista de valores a insertar en un `AlgarroboTree` y el segundo argumento representa el *path* para guardar un JSON con la estructura del árbol resultante. Para esto, deberás crear un árbol vacío (sin nodos) e ir agregando uno por uno los datos siguiendo el orden de la lista, siempre balanceando luego de agregar cada dato para asegurar que el árbol resultante sea un `AlgarroboTree`. Finalmente deberás generar un archivo JSON con la representación del árbol.

Puede asumir que `data` será una lista de solo números enteros y únicos, y que el `output_path` será válido. El formato del archivo a escribir es igual al formato de los archivos a leer en la consulta 1.

3.3. `route(data: [int], start_value: int, end_value: int)` → `[int]`

De forma similar a la `build_and_save`, el primer argumento representa una lista de números enteros a insertar en un `AlgarroboTree`. Luego de construir el árbol, esta función retorna una lista con la ruta desde el nodo que posea el valor `start_value` hasta el nodo que posea el valor `end_value`. Puede asumir que ambos valores estarán en el árbol y que `data` tendrá solo valores únicos. A continuación se muestra, como ejemplo, el árbol generado a partir de la lista `[4, 7, 15, 10, 11, 12]`[3] y 3 rutas distintas que se pueden solicitar.

[2] Recuerde la *keyword* `raise`. Para más detalle revisar el material de excepciones
[3] Hubo un *swapping* entre el 15 y el 12 después de insertar el último dato

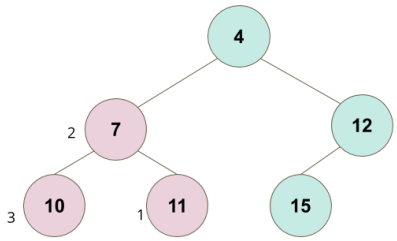

Figura 13: Ruta desde `11` a `10` = `[11, 7, 10]`

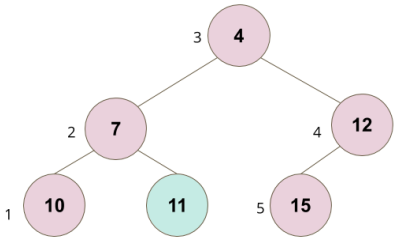

Figura 14: Ruta desde `10` a `15` = `[10, 7, 4, 12, 15]`

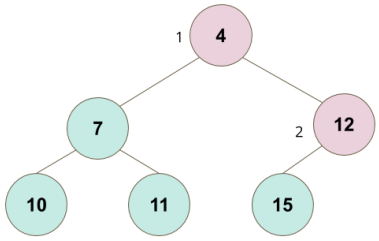

Figura 15: Ruta desde `4` a `12` = `[4, 12]`

4. Archivos entregados

Para esta tarea se entregarán una diversa cantidad de archivos y carpetas que serán usados en la corrección. A continuación se detalla cada elemento entregado:

- `main.py`: archivo a completar con las tres consultas descritas en la sección 3.
- `tree.py`: archivo a completar con un boceto básico de las clases, atributos y métodos mínimos[4] que deberás ocupar en esta tarea. En particular, la clase es
  - `AlgarroboTree`: clase que representa al árbol en sí. Posee los métodos:
    - `insert(value: int)` que se encarga de insertar el valor donde corresponde en el árbol.
    - `balance()` que se encarga de realizar los *swapping* necesarios para mantener las reglas de un `AlgarroboTree`.

    Además posee los siguientes atributos:
    - `self.root` que representa el nodo raíz del árbol.
  - `Node`: que representa un nodo del árbol. Contiene los siguientes atributos:
    - `self.left_child` que puede ser `None` o `Node` y representa al hijo izquierdo del nodo.
    - `self.right_child` que puede ser `None` o `Node` y representa al hijo derecho del nodo.
    - `self.value` que es un `int` y corresponde al valor que guarda el nodo.
- `exceptions.py`: archivo con las clases para las tres excepciones personalizadas. No debes editarlo.
- `test.py`: utiliza los tres archivos anteriores para probar las funcionalidades y calcular las décimas según la cantidad de test exitosos. No debes editarlo, y puedes utilizarlo para probar si tu programa está funcionando correctamente.
- `test`: carpeta con diversos archivos JSON para realizar la corrección.
- `student_solution`: carpeta que se usará para guardar algunos archivos.

Específicamente, para esta tarea tu labor será completar los archivos `main.py` y `tree.py`. El resto de los archivos **no debe ser editado**, ya que podría causarte errores con el corrector automatizado.

Importante: las carpetas `test` y `student_solution` solo son creadas con fines de mantener un orden en su repositorio, pero al momento de corregir puede o no existir dichas carpetas.

[4] Puedes agregar otros atributos y métodos que estimes convenientes, pero si debes usar los mencionados.

5. Metodología de corrección

Para esta tarea, **deberás utilizar la estructura entregada** (clases `AlgarroboTree` y `Node`) como mínimo en la segunda y tercera función. El archivo `test.py` va a validar mediante el uso de los atributos `self.root`, `self.left_child`, `self.right_child` y `self.value` y los métodos `insert` y `balance` que se esté usando correctamente la estructura entregada. En caso que se caiga el programa o se detecte que no fue ocupada la estructura, **no se evaluarán ni la segunda y tercera función de dicha tarea**.

Por otro lado, debido al corto tiempo que se dispone para corregir estas tareas, la corrección será automatizada. Por lo tanto, es deber del alumno asegurar que las consultas retornen y/o guarden los resultados en el formato solicitado. Para cada consulta se realizarán una cierta cantidad de *tests* y la cantidad de décimas otorgadas será calculada como:

```
round(cantidad_test_exitosos / cantidad_test_totales, 1)
```

La función `round(dato, decimales)` corresponde a redondear el resultado a 1 decimal. Por lo tanto:

- `round(0.56, 1) = 0.6`
- `round(0.54, 1) = 0.5`
- `round(1, 1) = 1`
- `round(0.99, 1) = 1`

El archivo a completar debe ser `main.py` y `tree.py`. Además en el archivo `test.py` incluye funciones para comprobar lo implementado en `main.py`, este les servirá como ejemplo del proceso de corrección de esta tarea.

6. Restricciones y alcances

- Esta tarea es **estrictamente individual**, y está regida por el **Código de honor de Ingeniería**.
- Tu programa debe ser desarrollado en Python 3.6.
- No existe condición previa para que estas décimas sean válidas, es decir, se aplican directamente al promedio sin necesidad de estar aprobando tareas.
- No puede utilizar ninguna librería aparte de `json`.
- En esta tarea no se aplicarán descuentos.
- No se aceptará entregas atrasadas.
- Como el proceso está automatizado, esta tarea no cuenta con instancia de recorrección. Por lo tanto, verifique entregar los resultados con el formato solicitado.

Las tareas que no cumplan con las restricciones del enunciado obtendrán la calificación mínima (1,0).