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

IIC2233 Programación Avanzada (2025-1)

Tarea 1

Entrega

- **Fecha y hora oficial (sin atraso):** miércoles 26 de marzo de 2025, 20:00.
- **Fecha y hora máxima (2 días de atraso):** viernes 28 de marzo de 2024, 20:00.
- **Lugar:** Repositorio personal de GitHub — Carpeta: `Tareas/T1/`
  El código debe estar en la rama (*branch*) por defecto del repositorio: `main`.
- **Pauta de corrección**: **en este enlace**.
- **Formulario entrega atrasada: en este enlace**
- **Bases generales de tareas (descuentos)**: **en este enlace**.
- **Ejecución de tarea**: La tarea será ejecutada **únicamente** desde la terminal del computador. Además, durante el proceso de corrección, se cambiará el nombre de la carpeta "`T1/`" por otro nombre y se ubicará la terminal dentro de dicha carpeta antes de ejecutar la tarea. **Los *paths* relativos utilizados en la tarea deben ser coherentes con esta instrucción.**

Objetivos

- Desarrollar algoritmos para la resolución de problemas complejos.
- Aplicar competencias asimiladas en "Introducción a la Programación" para el desarrollo de una solución a un problema.
- Procesar *input* del usuario de forma robusta, manejando potenciales errores de formato.
- Interactuar con diversos archivos de Python y ejecutarlos mediante terminal.
- Trabajar con archivos de texto para leer, escribir y procesar datos.
- Escribir código utilizando paquetes externos (*i.e.* código no escrito por el estudiante), como por ejemplo, módulos que pertenecen a la biblioteca estándar de Python y módulos no estándares.
- Familiarizarse con el proceso de entrega de tareas y uso de buenas prácticas de programación.
- Aplicar conceptos de programación orientada a objetos (POO) para resolver un problema.

Índice

1. *DCCortaRamas*

Durante el verano, te diste cuenta que pasar todo el día viendo *memes* y series no sería una buena inversión de tu tiempo. Es por esto que decidiste embarcarte en un nuevo *hobby*, que involucrara ~~tocar un poco de pasto~~ establecer una mejor relación con la naturaleza: el arte del bonsái, que corresponde a cultivar árboles y arbustos de forma estéticamente agradable, manteniendolos de un tamaño controlado.

Según el video tutorial que viste sobre esta práctica, títulado "*Cómo ser un maestro del bonsái en 10 minutos*", aprendiste que una de las mejores prácticas es manipular tu árbol bonsái de forma que este termine siendo **simétrico**, es decir, en base a un eje vertical, las dos mitades del bonsái son reflexiones entre sí en término de ramas y flores presentes. Sin embargo, debido al alto costo de los materiales del *hobby*, has decidido primero crear un programa que permita simular las acciones que debes realizar en tu bonsái, para prácticar antes de comenzar a cortar bonsáis reales. Has decidido llamar a tu creación: *DCCortaRamas*.

2. Bonsáis

Para poder prácticar tus habilidades en el arte del bonsái, deberás poder simular un bonsái y las acciones aplicables a este mediante un programa en Python. Para esto, representaremos la estructura del bonsái como una lista de listas:

- El bonsái será representado como una **lista**, que contiene dentro de sí un grupo de **nodos**. Un camino entre dos nodos corresponde a una **rama**.
- Cada nodo corresponde a una sección del bonsái que potencialmente puede dividir la rama actual en a lo más dos sub-ramas, además de potencialmente contener una **flor**. Cada nodo es representado por una **lista**, que contiene la siguiente información:
  - El primer elemento corresponde a un `string` que representa el identificador único del nodo.
  - El segundo elemento es un `bool` que identifica si el nodo contiene actualmente una flor o no.
  - El tercer elemento es un `bool` que identifica si el nodo puede ser editado: es posible agregar una flor, quitar una flor, o cortar la rama que termina en dicho nodo por completo.
  - El cuarto elemento es una lista de dos elementos, cuyo primer y segundo elemento corresponde al identificador del hijo izquierdo y derecho del nodo, respectivamente. En caso de no tener hijo izquierdo o derecho, el elemento en dicha posición tendrá un valor de `0`.

Es importante notar que en la definición de esta estructura, siempre se deben cumplir las siguientes condiciones:

- No pueden existir dos nodos con el mismo identificador.
- El nodo raíz (nodo que es marcado como inicio del bonsái) siempre será el con un identificador con valor igual a "`1`".
- La lista que representa la estructura del bonsái solo puede contener nodos que estén siendo usados en la distribución actual del bonsái. Es decir, nodos que estén conectados a la raíz por un camino.
- Los nodos dentro de la lista que representa el bonsái pueden venir en cualquier orden, no necesariamente se definirán en orden decreciente desde la raíz.
- Un nodo puede poseer a lo más, dos hijos. Un nodo hijo siempre tendrá asignada la posición de Izquierdo o Derecho.
- Dos nodos distintos no pueden tener al mismo nodo como hijo.

- La existencia de una flor en un nodo puede ser modificada solo si el nodo **permite edición**. De la misma forma, un nodo puede ser removido solo si **ese nodo** y **todos** sus hijos permiten edición.

La Figura 1 muestra un ejemplo de la estructura de un bonsái y cómo sería su representación visual:

```
[
  ["2",True,True,["4","0"]],
  ["1",True,True,["2","3"]],
  ["4",False,True,["0","0"]],
  ["5",True,False,["0","0"]],
  ["3",False,True,["0","5"]]
]
```

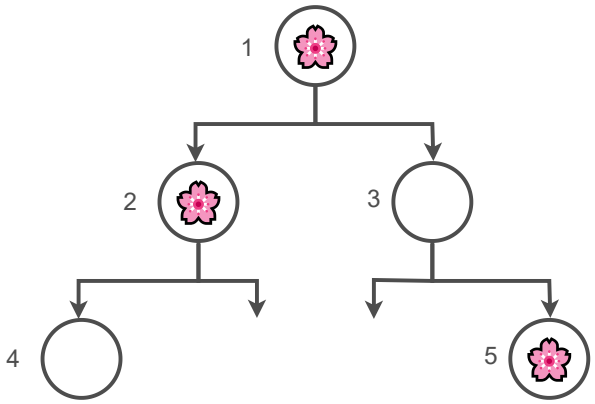
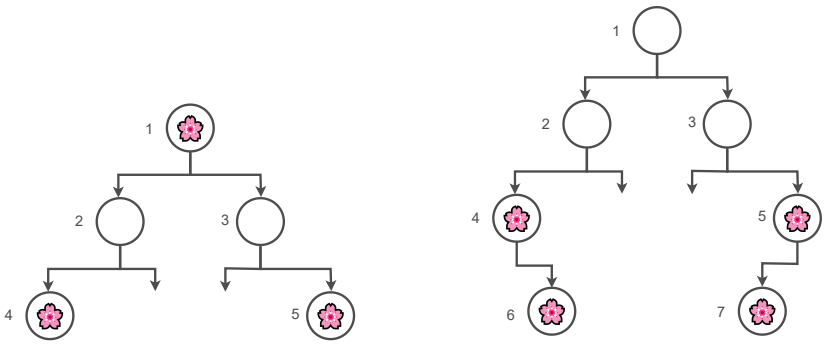


Figura 1: Representación de un bonsái no simétrico en forma de estructura y visual

Simetría de un bonsái

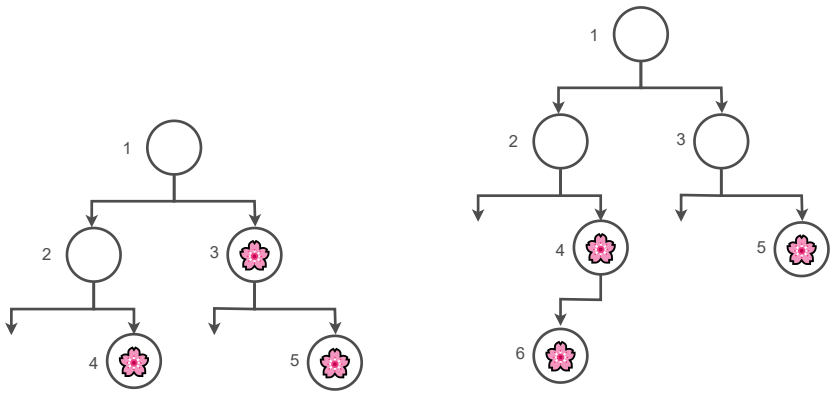
Un bonsái es considerado simétrico si, bajo un eje vertical que pasa por el medio de este, se cumple que **los nodos al lado derecho del eje y al lado izquierdo de este son reflejas entre sí**, considerando también la **existencia de flores** dentro de cada nodo.

A continuación se muestran algunos ejemplos de bonsáis simétricos y no simétricos:



(a) (b)

Figura 2: Ejemplos de bonsáis simétricos



(a) (b)

Figura 3: Ejemplos de bonsáis no simétricos

Edición de un bonsái

Un bonsái puede ser editando de tres formas: agregando una flor a un nodo, quitando una flor a un nodo y quitando un nodo, lo que implica eliminar también a todos sus nodos hijos de la estructura del árbol.

Para tanto quitar como agregar una flor, esto será posible siempre y cuando el **nodo** en cuestión permita edición. En el caso de cortar un nodo, esto podrá hacerse siempre y cuando **tanto el nodo como todos sus hijos** permitan edición.

Siguiendo el ejemplo de la Figura 1: podríamos quitar o agregar flores al nodo 1, 2, 3 y 4, pero no al nodo 5, ya que este no permite edición. Adicionalmente, podríamos quitar el nodo 2 y 4, pero no los nodos 1, 3, y 5, ya que el nodo 5 no permite edición.

Finalmente, realizar ediciones a un bonsái tiene un costo asociado: tanto modificar el estado de una flor dentro de un bonsái como eliminar un nodo tendrán su propio costo respectivo, indicado al momento de inicializar un árbol. Cabe notar que el costo de eliminar un nodo corresponderá únicamente al costo asociado cortar un nodo individual y **no** es la suma de los costos de cortar el nodo y todos sus hijos en caso de tenerlos.

3. Flujo del programa

Tu objetivo en esta evaluación estará dividido en completar 2 partes:

Parte 1 Completar un archivo con las clases `Bonsai` y `DCCortaRamas`, las cuáles permitirán: abrir archivos, analizar bonsáis, aplicar cambios a estos y responder a consultas del programa.
Esta parte será corregida automáticamente mediante el uso de *tests*.

Parte 2 Confeccionar un menú por consola en dos partes, que permita interactuar mediante *DCCortaRamas* con bonsáis.
Esta parte será corregida manualmente por el cuerpo docente.

3.1. Parte 1 - Funcionalidades

En esta parte, deberás completar diversos métodos de las dos clases indicadas para trabajar con el sistema de *DCCortaRamas*. La corrección completa de esta parte será mediante *tests*. Es por esto que debes asegurarte que cada método se pueda ejecutar correctamente, que el archivo no presente errores de sintaxis, y no cambiar el nombre y ubicación del archivo.

Para cada clase y método indicado a continuación, **no puedes modificar su nombre, agregar nuevos argumentos o argumentos por defecto**. En caso de no respetar lo indicado, la evaluación presentará un fuerte descuento. Puedes crear nuevos atributos, nuevos métodos, archivos y/o funciones si estimas conveniente. También se permite crear funciones en otro módulo y que el método que pedimos completar únicamente llame a esa función externa. El requisito primordial es que debes mantener el formato de los métodos y atributos informados en este enunciado. Esto con el fin de que los *tests* se ejecuten correctamente.

Adicionalmente, para apoyar el desarrollo de esta parte, se proveerá de una batería de *tests* junto a los archivos base de la tarea, donde podrán revisar distintos casos con su respuesta esperada. Esta batería se denomina como "***tests* públicos**". Estos *tests* públicos corresponden a un segundo chequeo de la evaluación, es decir, no son representativos de todos los casos posibles que tiene una función/método. Al momento de corregir, el equipo docente usará una batería de *tests* distinta a estas pruebas, la cual llamamos ***tests* privados** y contiene *tests* que pueden evaluar *inputs* distintos y por ende, casos bordes que pueden no haber sido incluidos en los públicos. La batería de *tests* privada se publica posteriormente junto con las notas de la evaluación para que el estudiante corrobore su puntaje obtenido.

Por lo tanto, **es responsabilidad del estudiantado confeccionar una solución que cumplan con lo expuesto en el enunciado y que no esté creada solamente a partir de los *tests* públicos**. De ser necesario, el estudiantado deberá pensar en nuevos casos que sean distintos a los *tests* públicos. **No se aceptarán supuestos que funcionaron en los *tests* públicos, pero van en contra de lo expuesto en el enunciado.**

A continuación se describen clases `Bonsai` y `DCCortaRamas` que deberás completar. **Importante**: para todos los métodos de clase donde se deba abrir un archivo, puedes asumir que el formato será adecuado, pero que su contenido puede cambiar al momento de evaluar la entrega.

Modificar `class Bonsai:`

Clase que representa un árbol bonsái.

- No modificar `def __init__(self, identificador: str, costo_corte: int, costo_flor: int, estructura: list) -> None:`

  Inicializa una instancia de Bonsái y asigna los siguientes atributos:

`self.identificador`	Un `str` que representa el nombre de ese bonsái.
`self.costo_corte`	Un `int` que representa cuánto cuesta cortar un nodo del bonsái.
`self.costo_flor`	Un `int` que representa cuánto cuesta modificar una flor del bonsái.
`self.estructura`	Una lista de listas que contiene todos los nodos del bonsái actual, siguiendo el formato anteriormente explicado.

- Modificar `def cargar_bonsai_de_archivo(self, carpeta: str, archivo: str) -> None:`

  Esta función recibe el nombre de la subcarpeta y nombre del archivo dentro de esta que contiene la estructura del bonsái a cargar. Debes abrir el archivo indicado, usando el *encoding* `utf-8`, leer y procesar su contenido, y finalmente asignar la estructura obtenida en la variable `self.estructura`, siguiendo el formato explicado en Bonsáis.

  Puedes asumir que tanto la subcarpeta como nombre de archivo entregado existen dentro de la carpeta `data`, y que el nombre de archivo entregado incluye la extensión `.txt`.

- Modificar `def visualizar_bonsai(self, orientacion: str, emojis: bool, guardar_archivo: bool) -> None:`

  Este método recibe una orientación que puede ser "`Vertical`" u "`Horizontal`", un booleano que representa si se usarán emojis, y un booleano que representa si el resultado se visualizará o guardará en un archivo o no.

  Esta función debe generar una visualización del bonsái actualmente almacenado en `self.estructura`, usando **obligatoriamente** la función `visualizar_bonsai` del archivo `utilidades.pyc`, explicado en Archivos, usando como parámetros los definidos anteriormente. En caso de que el parámetro `guardar_archivo` sea falso, el bonsái solo debe ser impreso en consola, mientras que en caso de ser verdadero, se debe tomar el retorno de la función `visualizar_bonsai` y almacenarlo en un archivo cuyo nombre será `identificador.txt` (siendo `identificador` el nombre del bonsái), dentro de la carpeta `visualizaciones`. Deberás asegurarte de escribir el archivo usando el *encoding* `utf-8`.

  Puedes asumir que la carpeta `visualizaciones` siempre existirá.

Modificar `class DCCortaRamas:`

Clase que representa el sistema de edición de bonsáis, siguiendo las reglas definidas anteriormente.

- Modificar `def modificar_nodo(self, bonsai: Bonsai, identificador: str) -> str:`

  Este método recibe una instancia de bonsái y un identificador que corresponde al nodo que se desea editar.

  Deberás intentar modificar el nodo seleccionado, verificando si es posible de editar. En caso de tener una flor, debes eliminar la flor del nodo, y en caso de no tenerla, debes añadir una flor al nodo.

  Si la acción pudo ser realizada, deberás retornar el *str* `Realizado`. Si la acción no pudo ser realizada, deberás retornar el *str* `No permitido`. Si el nodo no existe en el bonsái, deberás retornar el *str* `No encontrado`.

  Ejemplo: Si se tiene un bonsai cuya estructura es `[["1", True, False, ["2", "3"]], ["2", True, True, ["0", "0"]], ["3", True, False, ["0", "0"]]]`, y se solicita cambiar el nodo

`"10"`, el bonsai deberá retornar `"No encontrado"`. Si se solicita cambiar el nodo `"1"`, el bonsái deberá retornar `"No permitido"`. Si se solicita cambiar el nodo `"2"`, el bonsai resultante tendrá la estructura `[["1", True, False, ["2", "3"]], ["2", False, True, ["0", "0"]], ["3", True, False, ["0", "0"]]]` y la función retornará `"Realizado"`.

- Modificar `def quitar_nodo(self, bonsai: Bonsai, identificador: str) -> str:`

  Este método recibe una instancia de bonsái y un identificador que corresponde al nodo que se desea elimnar.

  Deberás intentar eliminar el nodo seleccionado, siguiendo las reglas de edición definidas anteriormente. En caso de poder hacerlo, debes eliminar tanto al nodo como a todos sus hijos de la estructura del bonsái.

  Si la acción pudo ser realizada, deberás retornar el *str* `Realizado`. Si la acción no pudo ser realizada, deberás retornar el *str* `No permitido`. Si el nodo no existe en el bonsái, deberás retornar el *str* `No encontrado`.

  Ejemplo: Si se tiene un bonsai cuya estructura es `[["1", True, False, ["2", "3"]], ["2", True, True, ["4", "0"]], ["3", True, False, ["0", "0"]], ["4", True, True, ["0", "0"]]]`, y se solicita quitar el nodo `"10"`, el bonsai deberá retornar `"No encontrado"`. Si se solicita quitar el nodo `"1"`, el bonsái deberá retornar `"No permitido"`. Si se solicita quitar el nodo `"2"`, el bonsái resultante tendrá la estructura `[["1", True, False, ["0", "3"]], ["3", True, False, ["0", "0"]]]` y la función retornará `"Realizado"`.

- Modificar `def es_simetrico(self, bonsai: Bonsai) -> bool:`

  Este método recibe una instancia de bonsái.

  Debes verificar si el bonsái es simétrico según las reglas indicadas anteriormente. En caso de serlo, deberás retornar `True`, y en caso contrario, retornar `False`.

  Es importante que esta función **no haga ningún cambio** sobre la instancia de bonsái entregada.

  Ejemplo: Una instancia de bonsái con estructura `[["1", True, True, ["2", "3"]], ["2", False, True, ["0", "0"]], ["3", False, False, ["0", "0"]]]` retornará `True`, mientras que uno con estructura `[["1", True, True, ["2", "3"]], ["2", True, True, ["0", "0"]], ["3", False, False, ["0", "0"]]]` retornará `False` ya que a pesar de ser simétrico en nodos, en la izquierda hay nodos con flores que en la derecha no tienen.

- Modificar `def emparejar_bonsai(self, bonsai: Bonsai) -> list:`
  Este método recibe una instancia de bonsái.

  Debes intentar emparejar el bonsái ingresado, mediante una sucesión de ediciones de flores y/o corte de ramas. En caso de encontrar una secuencia de instrucciones **válidas** que logren hacer que el bonsái sea simétrico, el método deberás retornar una lista con el siguiente formato:

  `[True, [[Accion1, Identificador1], [Accion2, Identificador2], ..., [AccionN, IdentificadorN]]]`

  Donde el primer elemento `True` indica que se encontró una solución, y el segundo elemento corresponde a una lista que contiene una sucesión de instrucciones que siguen el formato `[Acción, Identificador]`, donde `Accion` puede ser `"Modificar Flor"` o `"Quitar Nodo"` e `Identificador` corresponde al identificador del nodo al que se le aplicará la acción.

La secuencia de instrucciones debe ser **válida**, es decir, debe cumplir con estas dos condiciones:

- Ninguna instrucción individual debe ser sobre un identificador no existente, ni se debe incluir una instrucción sobre un nodo que no permite ediciones.
- Cada nodo puede estar en a lo más **una instrucción** en la secuencia. Por ejemplo, no sería válido que dentro de la secuencia este la instrucción `["Modificar Flor", identificador]` y más adelante nuevamente `["Modificar Flor", identificador]` ó `["Quitar Nodo", identificador]`, ya que implicaría que en la secuencia se está editando el nodo `identificador` más de una vez.

Si existe más de una serie de instrucciones que haga al bonsái simétrico, basta con que retornes solo una.

En caso de no encontrar una secuencia de instrucciones válidas, debes retornar `[False, []]`, lo que indica que no hubo solución y por ende, no existen instrucciones para emparejar el bonsái.

Es importante que esta función **no haga ningún cambio** sobre la instancia de bonsái entregada.

Pista: Para completar este método puede que te sea útil investigar sobre el concepto de *backtracking* y cómo podrías aplicarlo para programar esta función.

Ejemplo: Una instancia de bonsái con estructura `[["1", True, True, ["2", "3"]], ["2", True, True, ["4", "0"]], ["3", False, False, ["0", "0"]], ["4", True, True, ["0", "0"]]]` podría retornar `[True, [["Quitar Nodo", "4"], ["Modificar Flor", "2"]]]`. Mientras que una estructura `[["1", True, True, ["2", "3"]], ["2", True, False, ["0", "0"]], ["3", False, False, ["0", "0"]]]` debería retornar `[False, []]`.

- Modificar `def emparejar_bonsai_ahorro(self, bonsai: Bonsai) -> list:`

Este método recibe una instancia de bonsái.

Debes intentar emparejar el bonsái ingresado, siguiendo las **mismas reglas** que el método anterior.

Sin embargo, a diferencia del método anterior, este método debe **llevar registro del costo total** de la sucesión de instrucciones realizada sobre el bonsái, basándose en los costos de modificación de flores y corte de nodos asociados al bonsái. En caso de encontrar una o más soluciones para emparejar el bonsái, debes retornar de forma **obligatoria** la serie de instrucciones que minimice el costo total de ejecución. En caso de que haya más de una solución con el costo mínimo, puedes retornar cualquiera.

En caso de encontrar una solución que minimice el costo total, debes retornar una lista con el formato:

```
[True, costo, [[Accion1, Identificador1], [Accion2, Identificador2], ...,
                                [AccionN, IdentificadorN]]]
```

Donde el primer elemento `True` indica que se encontró una solución, el segundo elemento indica cuánto cuesta realizar la serie de acciones y el tercer elemento es la lista de acciones a realizar sobre el bonsái para dejarlo simétrico.

En caso de no encontrar una secuencia de instrucciones válidas, debes retornar `[False, []]`, lo que significa que no hubo solución y por ende, no existen instrucciones para emparejar el bonsái.

Es importante que esta función **no haga ningún cambio** sobre la instancia de bonsái entregada.

Pista: Para completar este método puede que te sea útil investigar sobre el concepto de *backtracking* y cómo podrías aplicarlo para programar esta función.

Ejemplo: Una instancia de bonsái con estructura `[["1", True, True, ["2", "3"]], ["2", True, False, ["4", "0"]], ["3", True, False, ["0", "5"]], ["4", True, True, ["0", "0"]], ["5", False, True, ["0", "0"]]]`, un costo de modificación de flor de 2 y un costo de corte de nodo de 5 podría retornar `[True, 2,[["Modificar Flor", "4"]]]`, en vez de retornar `[True, 10,[["Quitar Nodo", "4"], ["Quitar Nodo", "5"]]]`, ya que modificar una flor es más barato que cortar 2 nodos.

- Modificar `def comprobar_solucion(self, bonsai: Bonsai, instrucciones: list) -> list:`

Este método recibe una instancia de bonsái y una lista de instrucciones que siguen el formato:

`[[Accion1, Identificador1], [Accion2, Identificador2], ..., [AccionN, IdentificadorN]]`

Deberás verificar que la serie de instrucciones es una solución válida. Para esto:

- Todas las instrucciones de la lista son válidas, es decir, posibles de realizar a la estructura del bonsái. No se aplican instrucciones a un nodo que no existe, o que existe pero no permite edición.
- Cada nodo está en a lo más una instrucción en la secuencia.
- Tras aplicar la serie de instrucciones al bonsái, el resultado termina con un bonsái simétrico.

En caso de ser una serie válida, deberás retornar una lista con el formato `[True, costo]`, donde `True` indica que la solución es válida y `costo` corresponde al costo de realizar la serie sobre el bonsái. En caso de no ser válida, debes retornar `[False, 0]`

Es importante que esta función **no haga ningún cambio** sobre la instancia de bonsái entregada.

Ejemplo: Para un bonsái con estructura `[["1", True, True, ["2", "3"]], ["2", True, False, ["0", "0"]], ["3", False, True, ["0", "0"]]]` y un costo de modificación de hoja de 2, la serie de instrucciones `[["Modificar Flor", "3"]]` debería retornar `[True, 2]`. Mientras que la serie de instrucciones `[["Modificar Flor", "2"]]` debería retornar `[False, 0]`.

3.2. Parte 2 - Menú

En esta parte deberás implementar un menú que permita interactuar con los bonsáis y las funcionalidades asociadas de *DCCortaRamas*. Para esto, se ejecutará tu programa mediante un archivo `main.py`. Las diversas acciones a ejecutar en los menús deberán aceptar entradas de usuario mediante terminal.

La interacción del menú se divide en dos partes: Menú de Inicio y Menú de Acciones. En esta parte de la tarea, se evaluará principalmente que (1) ambos menús sean a prueba de errores, (2) ambos incluyan todas las opciones solicitadas, (3) que los parámetros recibidos por el usuario se manejen de forma correcta, y (4) que cada opción del menú llame, mediante código, a la función o método correspondiente.

En esta parte no se evaluará si la opción seleccionada realmente hace lo esperado, eso será evaluado en la Parte 1 mediante *tests*. En esta parte se espera un correcto manejo de *inputs* y llamar, en el código, a las funciones o métodos que correspondan según la opción elegida.

3.2.1. Menú de Inicio

Este es el menú encargado de iniciar la interacción con el usuario y realizar la carga de *DCCortaRamas*. A continuación se entrega un ejemplo de cómo se podría ver el menú de inicio tras haber ejecutado por primera vez `python3 main.py` :

```
¡Bienvenido a DCCortaramas!

*** Menú de Inicio ***

[1] Cargar bonsái
[2] Salir del programa

Indique su opción (1, 2):
```

Figura 4: Ejemplo de Menú de Inicio

A continuación, se detalla la función que debe cumplir cada una de las acciones que podrá realizar el usuario en el Menu de Inicio:

1. **Cargar bonsái**: Una vez seleccionada, se debe pedir al usuario que ingrese un nombre de carpeta y nombre de archivo. En caso de existir la carpeta, y el archivo dentro de esta, la función debe cargar los contenidos del archivo en un bonsái y continuar al Menú de Acciones.
2. **Salir del programa**: Termina la ejecución del programa.

3.2.2. Menú de Acciones

Luego de haber seleccionado la opción 1 del Menú de Inicio e ingresado la información pedida, se abrirá un segundo menú encargado de realizar las acciones sobre los predios cargados. A continuación se entrega un ejemplo de cómo se podría ver el Menú de Acciones:

```
¡Bienvenido a DCCortaramas!
Es hora de convertir tu bonsái en un hermoso bonsái.

*** Menú de Acciones ***

[1] Visualizar bonsái
[2] Modificar Hoja
[3] Cortar Rama
[4] Verificar Simetría
[5] Podar bonsái
[6] Salir del programa

Indique su opción (1, 2, 3, 4, 5, 6):
```

Figura 5: Ejemplo de Menú de Acciones

A continuación, se detalla la función que debe cumplir cada una de las acciones que podrá realizar el usuario:

1. **Visualizar bonsái**: Esta acción deberá visualizar el bonsái actualmente almacenado en la instancia de Bonsai cargada e imprimirlo en la terminal, haciendo uso de la función `visualizar_bonsai`.
2. **Modificar Flor**: Esta acción intentará modificar el estado de una de las flores del Bonsái.

   Deberás solicitar al usuario que ingrese el *str* correspondiente al nodo cuya flor se desea modificar. Luego, deberás verificar si es posible editar dicho nodo y en caso de poderse, modificar el estado de la flor de dicho nodo (agregar una si el nodo no posee flor, y quitarla si la posee). Deberás imprimir en consola si fue posible la modificación, y en caso de serlo, cuál fue el resultado final de esta (adición o remoción de la flor).
3. **Cortar Nodo**: Esta acción intentará eliminar uno de los nodos del Bonsái.

   Deberás solicitar al usuario que ingrese el *str* correspondiente al nodo que se desea eliminar. Luego, deberás verificar si es posible editar dicho nodo y en caso de poderse, eliminarlo junto a todos sus hijos. Deberás imprimir en consola si fue posible la eliminación, y en caso de serlo, cuáles nodos fueron eliminados.
4. **Verificar Simetría**: Esta acción deberá verificar si el bonsái actualmente almacenado es simétrico o no, e informarle el resultado al usuario.
5. **Podar bonsái**: Esta acción deberá intentar podar el bonsái actualmente almacenado para que sea simétrico. Deberás solicitar al usuario que decida si la poda del bonsái considerará ahorros o no y usar el método respectivo según la respuesta.

   En caso de lograrlo, deberá retornar al usuario una visualización del bonsái podado haciendo uso de la función `visualizar_bonsai`, e indicar en consola la serie de acciones realizadas al bonsái para llegar a la simetría. Si fue una poda considerando ahorros, deberá también incluir el costo total de la poda.

   En caso de no lograrlo, deberá informar al usuario de esto.
6. **Salir del programa**: Termina la ejecución del programa.

4. Archivos

4.1. Código inicial

Para esta tarea, se te hará entrega de diversos archivos que deberás completar con funcionalidades. Puedes crear más archivos si lo estimas conveniente.

- Modificar `dccortaramas.py`: Aquí encontrarás la definición básica de las clases `Bonsai` y `DCCortaramas` que debes completar.
- Crear `main.py`: Este será el archivo que será ejecutado para levantar el menú por consola.
- No modificar `utilidades.pyc`: Este archivo contiene la función `visualizar_bonsai` que deberás ocupar en esta tarea. Más información de este archivo en la sección de `utilidades.pyc`.
- No modificar `data/`: Esta carpeta contendrá subcarpetas, que contienen archivos que reperesentan la estructura de un bonsái. El contenido de estos archivos podrán cambiar al momento de evaluar, manteniendo el formato.
- No modificar `visualizaciones/`: Carpeta originalmente vacía, en donde deberás generar y almacenar los archivos con visualizaciones de los bonsáis pedidos en las secciones anteriores.
- No modificar `tests_publicos/`: Esta carpeta contendrá una serie de archivos `.py` que corresponden a diferentes *tests* para apoyar el desarrollo de la Parte 1. Puedes utilizar los archivos entregados para ir viendo si lo desarrollado hasta el momento cumple con lo esperado en esta evaluación. **Importante**: los *tests* entregados en esta carpeta no serán los mismos que se utilizarán para la corrección de la evaluación.
- Modificar `README.md`: Contendrá inicialmente las actualizaciones a la tarea aplicadas hasta el momento. Debe modificarse para personalizar tu propio `README`.
- No modificar `README_inicial.md`: Es la base desde la cual deberás construir tu propio `README`.

4.2. `data/**/**.txt`

Para poder poner en práctica tus habilidades de bonsái, se te facilitará un conjunto de archivos que contendrán información sobre distintos bonsáis a cargar con los que podrás interactuar. Debes recordar que el nombre y contenido de estos archivos podrá cambiar al momento de evaluar, pero siempre manteniendo el formato.

Dentro de la carpeta "`data/`" encontrarás distintas subcarpetas, y dentro de ellas, distintos archivos en formato `.txt`, donde cada archivo contiene el formato de un bonsái en particular. El nombre del archivo (excluyendo la extensión) corresponderá al nombre del bonsái, y el contenido de cada archivo estará dado por diversas líneas de texto, en donde cada línea contiene información de un nodo del bonsái, separado por comas (","). Todas las líneas y sus elementos tendrán siempre el mismo formato:

- El primer elemento será un *string* que representa el identificador del nodo.
- El segundo elemento será un *string* ("`T`" o "`F`"). En caso de ser "`T`", significa que ese nodo contiene una flor y en caso de ser "`F`", ese nodo no contiene una flor.
- El tercer elemento será un *string* ("`T`" o "`F`"). En caso de ser "`T`", significa que ese nodo puede ser modificado (ya sea para agregar o quitarle una flor, o cortar dicho nodo del bonsái) y en caso de ser "`F`", ese nodo no puede ser modificado de ninguna forma.
- El cuarto elemento es un string con el formato "`Izq;Der`". `Izq` corresponderá al identificador del hijo izquierdo del nodo y `Der` al identificador del hijo derecho. Si uno de estos valores es `0`, eso significa

que el nodo no tiene un hijo en esa posición.

A continuación se presentan dos ejemplos de un archivo "`bonsai.txt`" y su representación visual como bonsái:

```
1,T,F,2;3
2,F,F,0;0
3,T,F,0;0
```

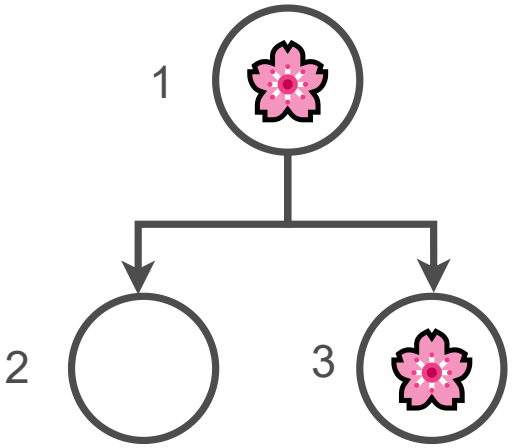


Figura 6: Representación de un bonsái en forma de archivo y visual

```
1,T,T,2;3
2,T,T,4;0
3,F,T,0;5
4,F,T,0;0
5,T,F,0;0
```

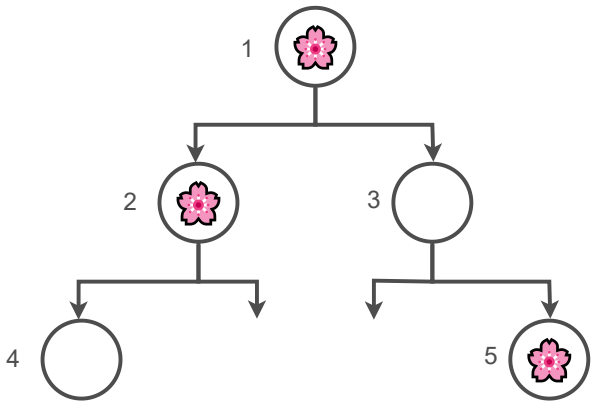


Figura 7: Representación de un bonsái en forma de archivo y visual

4.3. `utilidades.pyc`

Para facilitar tu trabajo y evaluar el uso de módulos, se te hará entrega del módulo `utilidades.pyc` que contiene una función que deberás utilizar en el desarrollo de la tarea. Este archivo se encuentra compilado, por lo que no se puede visualizar su contenido, y funciona únicamente con la versión de Python del curso: 3.11.X con X mayor a 7. Utilizar una versión distinta a 3.11 provocará un error de `magic number`.

Este archivo hace uso de forma interna de `PrettyPrintTree`, una librería externa de Python utilizada para visualizar bonsáis en la terminal. Por lo tanto, para hacer uso del archivo `utilidades.py`, deberás primero ejecutar el comando:

```
pip install PrettyPrintTree
```

O su equivalente en tu computadora. Luego, solo será necesario realizar `import utilidades` desde otro archivo `.py` para utilizar sus funciones.

La función que contiene `utilidades.pyc` es:

- `visualizar_bonsai(`
  `estructura: list, orientacion: str, emojis: bool, guardar: bool`
  `) -> None or str:`
  Esta función recibe una lista que sigue la estructura del atributo `estructura` de la clase `Bonsái` definida previamente, un *string* que puede ser "`Horizontal`" o "`Vertical,`" y representa la orientación en que se dibujará el bonsái, un *booleano* que indicará si la visualización incluirá emojis o solo texto, y finalmente un *booleano* que indica si el bonsái se imprimirá en consola directamente o se guardará como *string*.

  Luego, la función se encargará de generar la visualización del bonsái definido por la estructura y el resto de los argumentos entregados. Si el argumento `guardar` es `True`, la función retornará un *string* que representa el bonsái dibujado, y si es `False`, la función imprimirá el bonsái directamente en consola y no retornará nada.

Será tu deber importar **correctamente** este archivo y hacer uso de su función para el desarrollo de la tarea. Recuerda que **no** debes modificar su contenido.

5. `.gitignore`

Para esta tarea **deberás utilizar un** `.gitignore` para ignorar los archivos indicados, este deberá estar dentro de tu carpeta `Tareas/T1/`.

Los elementos que no debes subir y **debes ignorar mediante** `.gitignore` para esta tarea son:

- El enunciado.
- El archivo `utilidades.pyc`
- La carpeta `data/` y todo lo contenido en ella.
- La carpeta `tests_publicos/` y todo lo contenido en ella.
- La carpeta `visualizaciones/` y todo lo contenido en ella.
- El archivo `README_inicial.md`.

Recuerda **no ignorar archivos vitales de tu tarea como los que tú debes modificar, o tu tarea no podrá ser revisada**. Es importante que hagan un correcto uso del archivo `.gitignore`, es decir, los archivos **deben** no subirse al repositorio debido al uso correcto del archivo `.gitignore` y no debido a otros medios.

6. Importante: Corrección de la tarea

Para esta tarea, el carácter funcional del programa será el pilar de la corrección, es decir, **sólo se corrigen tareas que se puedan ejecutar**. Por lo tanto, se recomienda hacer periódicamente pruebas de ejecución de su tarea y *push* en sus repositorios, corroborando que, para las primeras 4 funciones (`cargar_bonsai_de_archivo`, `visualizar_bonsai`, `modificar_nodo`, `quitar_nodo`) **cada prueba individual se ejecute en un tiempo acotado de 10 segundos**, y para las últimas 4 funciones (`es_simetrico`, `emparejar_bonsai`, `emparejar_bonsai_ahorro`, `comprobar_solucion`) **cada prueba**

individual se ejecute en un tiempo acotado de 10 segundos, en caso contrario se asumirá un resultado incorrecto.

En el **siguiente enlace** se encuentra la distribución de puntajes. En color **amarillo** se encuentra cada ítem que será evaluado a nivel funcional y de código, es decir, aparte de que funcione, se revisará que el código esté bien confeccionado y que la funcionalidad esté correctamente integrada en el programa. Todo aquel que no esté pintado de amarillo significa que será evaluado si y sólo si se puede probar con la ejecución de su tarea o con los *tests*.

Importante: Todo ítem corregido por el cuerpo docente será evaluado únicamente de forma ternaria: cumple totalmente el ítem, cumple parcialmente o no cumple con lo mínimo esperado. Del mismo modo, todo ítem corregido automáticamente será evaluado de forma ternaria: puntaje completo si pasa todos los *tests* de dicho ítem, medio punto para quienes pasan más del 70 % de los *test* de dicho ítem, y 0 puntos para quienes no superan el 70 % de los *tests* en dicho ítem.Finalmente, todos los descuentos serán asignados manualmente por el cuerpo docente respetando lo expuesto en **el documento de bases generales**.

La corrección se realizará en función del último *commit* realizado antes de la fecha oficial de entrega. Si se desea continuar con la evaluación en el periodo de entrega atrasado, es decir, realizar un nuevo *commit* después de la fecha de entrega, **es imperante responder el formulario de entrega atrasada** sin importar si se utilizará o no cupones. Responder este formulario es el mecanismo que el curso dispone para identificar las entregas atrasadas. El enlace al formulario está en la primera hoja de este enunciado.

Para terminar, si durante la realización de tu tarea se te presenta algún problema o situación que pueda afectar tu rendimiento, no dudes en contactar al ayudante de Bienestar de tu sección. El correo está en el **siguiente enlace**.

6.1. Ejecución de *tests*

Para la corrección automática se entregarán varios archivos `.py` los cuales contienen diferentes *tests* que ayudan a validar el desarrollo de la Parte 1 - Funcionalidades. Para ejecutar estos *tests*, primero debes posicionar tu terminal/consola en la carpeta de la tarea `Tareas/T1/`. Luego, desde esta misma, debes escribir el siguiente comando para ejecutar todos los *tests*:

- `python3 -m unittest discover tests_publicos -v -b`

En cambio, si deseas ejecutar un subconjunto de *tests*, puedes hacerlo escribiendo lo siguiente:

- `python3 -m unittest -v -b tests_publicos.<test_N>`
  Reemplazando `<test_N>` por el test que desees probar.

Por ejemplo, si quisieras probar si realizaste correctamente el método `cargar_bonsai_de_archivo` de `Bonsai`, deberás escribir lo siguiente:

- `python3 -m unittest -v -b tests_publicos.test_00_cargar_bonsai_de_archivo`

Importante: recuerda que si `python3` no funciona, probar con el comando específico de tu computador. Este puede ser `py`, `python`, `py3` o `python3.11`.

7. 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 3.11.X con X mayor o igual a 7.
- Tu programa debe estar compuesto por uno o más archivos de extensión `.py` que estén correctamente ordenados por carpeta. **No se revisará archivos en otra extensión como `.ipynb`**.

- Si no se encuentra especificado en el enunciado, supón que el uso de cualquier librería Python **está prohibido**. Pregunta en la *issue* especial del **foro** si es que es posible utilizar alguna librería en particular.
- Debes adjuntar un archivo `README.md` **conciso y claro**, donde describas los alcances de tu programa, cómo correrlo, las librerías usadas, los supuestos hechos, y las referencias a código externo. El no incluir este archivo o bien incluirlo pero que se encuentre vacío conllevará un **descuento** en tu nota.
- Esta tarea se debe desarrollar **exclusivamente** con los contenidos liberados al momento de publicar el enunciado. No se permitirá utilizar contenidos que se vean posterior a la publicación de esta evaluación.
- Se encuentra estrictamente prohibido citar código que haya sido publicado **después de la liberación del enunciado**. En otras palabras, solo se permite citar contenido que ya exista previo a la publicación del enunciado. Además, se encuentra estrictamente prohibido el uso de herramientas generadoras de código para el apoyo de la evaluación.
- Cualquier aspecto no especificado queda a tu criterio, siempre que no pase por sobre otro que sí sea especificado por enunciado.

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