Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2022-1

2 de Junio de 2022
Actividad Sumativa

Actividad Sumativa 4

Template

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AS4/`
- **Hora del *push*:** 16:40

Importante: Antes de comenzar, comprueba que Git este funcionando correctamente en tu repositorio privado. Para esto, **sube los archivos base de la actividad de inmediato** (*add*, *commit*, *push*). Se espera que en esta actividad (así como en las demás actividades y tareas) utilices Git a lo largo de **todo tu desarrollo** como una herramienta, no sólo como un método de entrega. Es por esto que recomendamos enfáticamente que vayas subiendo tus cambios constantemente (***push***), ya que **problemas de último minuto** relacionados con la entrega y Git **no serán considerados**.

Introducción

¡Hoy es el cumpleaños del profesor Nicolás Elliott! (**en realidad no**) y para celebrarlo como corresponde el profe Cristián Ruz ha planeado una pizzaton totalmente sorpresa después de la actividad. Para esto tendrá que ir al local PequeñoCésar y hacer la fila para comprar las pizzas con piña favoritas de Nicolás, pero a Ruz lo distraen los mensajes de telegram ~~preguntándole por que se fue antes de la 17:00~~ y se le termina colando gente en la fila. Una vez compradas las pizzas tendrá que recorrer el campus para llegar al lugar donde ha citado a los profesores y ayudantes para la pizzaton, sin embargo en ciertos lugares del campus hay unos ayudantes muy chismosos[1] y que podrían arruinar la sorpresa. Deberás ayudar al profe Ruz a sortear todos estos inconvenientes y sacar adelante el Cumpleaños Sorpresa del profe Elliott.

Flujo del programa

El programa se simula mediante la ejecución del archivo `main.py`, el cual crea la fila de la pizzería como lista ligada y el campus como un grafo. Luego, pide por consola qué opción quieres simular. Las opciones consisten en las diferentes etapas que tienes que implementar: desde los eventos de la fila de la pizzería hasta los diferentes recorridos a través del campus. Una vez que hayas implementado la opción en cuestión, al seleccionar la opción podrás ver su resultado por consola.

[1] Sapos en la jerga chilena

Archivos

Para esta actividad cuentas con los siguientes archivos:

```
AS4
├── lectura_archivos.py No debes modificarlo: Lee los archivos de la actividad
├── main.py Ejecutar : Instancia las clases para la simulación Se debe ejecutar para
│   probar el código
├── parametros.py No debes modificarlo: Contiene los parámetros
├── Parte1
│   ├── eventos_fila.csv No debes modificarlo: Contiene la información de la fila
│   ├── fila_pizzas.py Debes modificarlo: Contiene las clases Cliente y FilaPizza que
│   │   deberás modificar en la Parte 1
│   ├── lectura_archivos_parte1.py No debes modificarlo: Lee los archivos de la actividad
│   └── pizzeria.py No debes modificarlo: Contiene las clases necesarias para simular los
│       eventos en la fila de la pizzería
└── Parte2
    ├── ayudantes.csv No debes modificarlo: Contiene la información de los ayudantes
    ├── lugares.csv No debes modificarlo: Contiene la información de los nodos
    ├── distancia_caminos.csv No debes modificarlo: Contiene la relación entre los
    │   diferentes nodos
    ├── ayudantes.py No debes modificarlo: Contiene la clase Ayudante que modela a los
    │   ayudantes
    ├── campus.py No debes modificarlo: Contiene las clases necesarias para simular el grafo
    │   del campus
    ├── lectura_archivos_parte2.py No debes modificarlo: Lee los archivos de la actividad
    └── recorrido_campus.py Debes modificarlo: Contiene las diferentes funciones de
        recorrido que deberás implementar en la Parte 2
```

IMPORTANTE: recuerda que para probar tu código debes correr el archivo `main.py` que se te entrega, no lo pruebes desde los módulos específicos. Recuerda también detener el programa y volverlo a correr cada vez que hagas modificaciones al código.

Parte 1: Implementación de la fila

Cuando Ruz llega al PequeñoCésar, se encuentra con un local caótico: la gente no sabe como organizarse para pedir sus pizzas. Ante esta situación deciden formar una fila y como tu estas muy familiarizado con ellas por calentar tu comida en el CAI, decides intervenir para ayudar al profesor a llegar a la caja.

En esta parte deberás implementar la fila del local como **lista ligada**. Para esto deberás modificar las clases `Cliente` y `FilaPizza` en el archivo `fila_pizzas.py` que se encuentra dentro de la carpeta `/Parte1`:

- `class Cliente`: Representa al cliente de la pizzería, que posteriormente se formará en la fila. Contiene los siguientes métodos:
  - `def __init__(self, nombre: str)`: Inicializador de la clase que recibe un string que corresponde al nombre del cliente, deberás definir un atributo `nombre` que lo guarde y el atributo `siguiente` que inicialmente toma valor `None` y representara al cliente que viene después en la fila Debes modificarlo
  - `def __str__(self) -> str`: Este método retorna el nombre del cliente No debes modificarlo

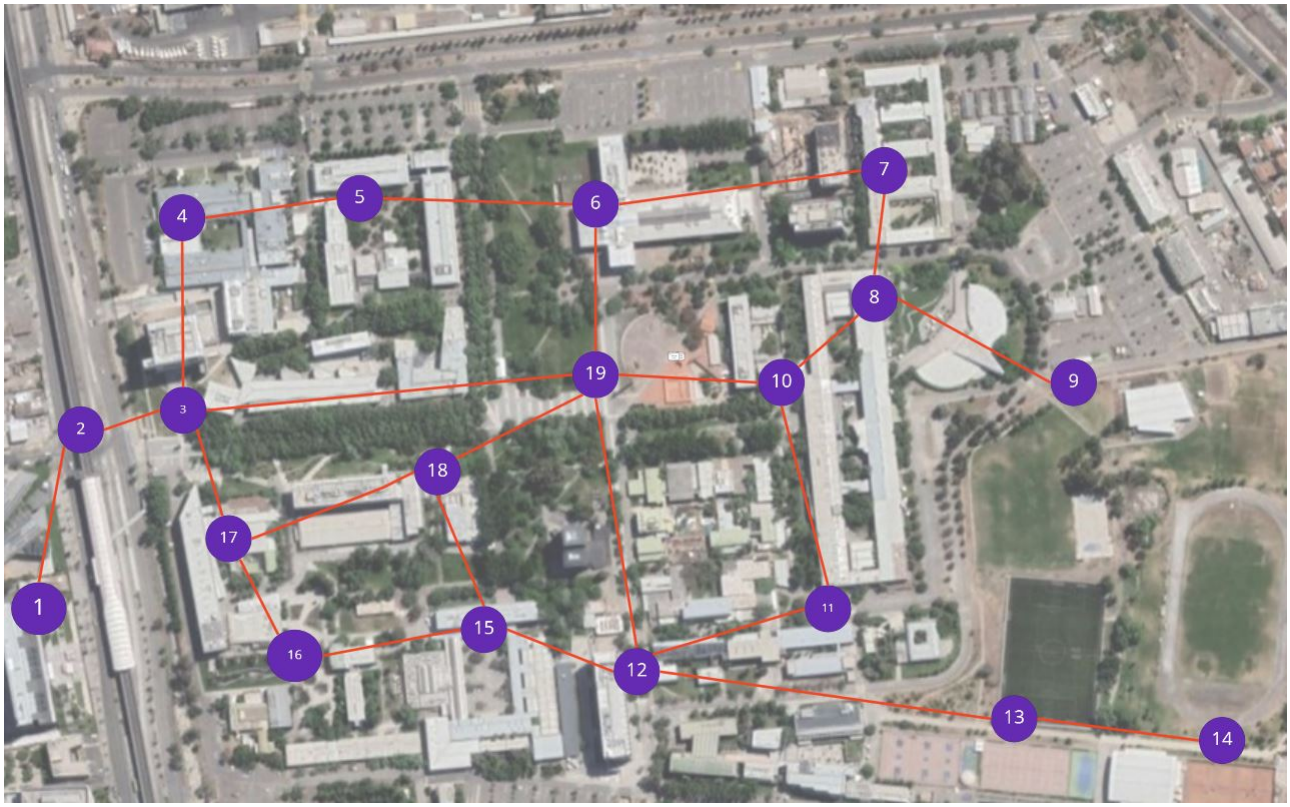
- `class FilaPizza`: Esta clase representa la Fila del PequeñoCésar. En esta se ordenan los clientes, los cuales pueden ponerse al final de la fila, colarse o ser atendidos. Tiene los siguientes métodos:
  - `def __init__(self)`: Inicializador de la clase que no recibe argumentos. Deberás definir el atributo `primero`, que inicialmente tiene valor `None` y representa al primer cliente de la fila, y el atributo `ultimo` que también toma el valor inicial `None` y representa al ultimo cliente de la fila. Finalmente deberás definir el atributo `largo` que inicialmente toma valor 0 y define el largo de la fila **Debes modificarlo**
  - `def llega_cliente(self, cliente: Cliente)`: Este método recibe una instancia de la clase Cliente y la agrega al final de la fila, deberás implementarlo tomando en cuenta los casos en que la fila está vacía y en los cuales tiene gente desde antes. Por último, deberás aumentar el largo de la fila en 1. Corresponde a un append de la lista ligada. **Debes modificarlo**
  - `def alguien_se_cuela(self, cliente: Cliente, posicion: int)`: Este método recibe una instancia de la clase cliente y una posición en la que este se cuela, deberás agregar a este cliente en la posición especificada. Tienes que tener en cuenta los siguientes casos:
    - El cliente se intenta colar en una fila vacía, este se colocará en la primera (que a la vez la es última) posición
    - El cliente se cuela en la posición 0, es decir al inicio de la fila
    - El cliente se cuela en una posición mayor o igual al largo de la fila, en este se colocará de último
    - El cliente se cuela en una posición mayor a cero y menor que el largo de la fila

    Finalmente deberás aumentar el atributo `largo` en uno. Corresponde a un insert en la lista ligada. **Debes modificarlo**
  - `def cliente_atendido(self)` → Cliente: Este método actualizará la fila haciendo que el primer cliente sea atendido, actualizando los atributos ”primero”, últimoz ”largo”. Para esto puedes considerar que solo se atenderán clientes si ya hay gente en la fila (no se atenderá cuando la fila este vacía), pero si debes tomar en consideración el caso en el que se atiende al último cliente de la fila. Finalmente, este método deberá retornar al cliente que fue atendido. **Debes modificarlo**
  - `def __str__(self) -> str`: En este método deberás retornar un string que contenga todos los nombres (en orden de primero a último) de los clientes que están realizando la fila. **Debes modificarlo**

Parte 2: Recorrido

Ahora con las ricas pizzas ya compradas, el profe Cristián tiene que llegar al lugar donde cito a todos al cumpleaños sorpresa del profe Nico. Sin embargo, como hay unos ayudantes muy chismosos que solo quieren arruinar la sorpresa, tendrá que evitarlos a toda costa. Para esto se te ocurre modelar el campus como un **grafo**, pero como es muy pillo el profesor Ruz, se te adelantó y ya modeló el grafo del campus y te explica cómo lo hizo:

"La siguiente imagen ilustra cómo se ve el grafo que representa el campus San Joaquín:



Este grafo se crea a partir de la clase `CreadorCampus` que instancia una clase `Campus`, el cual será el grafo. Los nodos de este son modelados mediante la clase `Lugar`, el cual tiene el atributo `self.vecinos` con el que vas a tener que trabajar (el resto de atributos no son de importancia para esta parte). Debido a los molestos ayudantes chismosos, primero vas a tener que comprobar que el `Lugar` esté libre de ellos antes de considerarlo dentro del recorrido. Para esto, ya cuentas con la función `comprobar_chismoso(lugar: Lugar)` que te permite identificar cuando hay un chismoso en un `Lugar`."

En esta parte deberás implementar un recorrido **BFS** o **DFS** (es decir **SOLO UNO** de los dos, el que prefieras). Para esto modifica **una** las funciones `bfs_iterativo` y `dfs_iterativo` en el archivo `recorrido_campus.py` que se encuentra dentro de la carpeta `/Parte2`:

- `def comprobar_chismoso(lugar: Lugar) -> bool`: Dado un determinado nodo `Lugar` retorna `True` o `False` según hayan o no chismosos en el. **No debes modificarlo**
- `def bfs_iterativo(inicio: Lugar, final: Lugar) -> bool`: Este método recibe un nodo de inicio y uno final, los que representan desde donde se parte y hasta donde se quiere llegar en el grafo. Deberás implementar el algoritmo de búsqueda **BFS** en esta función, utilizando la función `comprobar_chismoso(lugar: Lugar)`. Cuando haya un chismoso en el nodo, Ruz no podrá pasar por él, por ende no podrá llegar a los vecinos de los nodos con chismosos. En cambio, si no hay

ningún chismoso, Ruz puede transitar por él sin problema. Debes retornar `True` si Ruz puede llegar al `Final` y `False` si no puede. **Debes modificarlo**

- `def dfs_iterativo(inicio: Lugar, final: Lugar) -> bool`: Este método recibe un nodo de inicio y uno final, los que representan desde donde se parte y hasta donde se quiere llegar en el grafo respectivamente. Deberás implementar el algoritmo de búsqueda **DFS** en esta función, utilizando la función `comprobar_chismoso(lugar: Lugar)`. Cuando haya un chismoso en el nodo, Ruz no podrá pasar por él, por ende no podrá llegar a los vecinos de los nodos con chismosos. En cambio, si no hay ningún chismoso, Ruz puede transitar por él sin problema. Debes retornar `True` si Ruz puede llegar al `Final` y `False` si no puede. **Debes modificarlo**

Una vez que logres implementar lo anterior, Ruz te pide obtener el largo del camino que va a realizar. Para esto deberás modificar **SOLO UNA** de las siguientes funciones de acuerdo al recorrido que quieras hacer. *Hint*: puedes reutilizar el código anterior y modifícalo para lograr la funcionalidad que se pide:

- `def bfs_iterativo_largo(inicio: Lugar, final: Lugar) -> int`: Este método recibe un nodo de inicio y uno final, los que representan desde donde se parte y hasta donde se quiere llegar en el grafo. Deberás implementar el algoritmo de búsqueda **BFS** en esta y al igual que en la función anterior debes chequear que el lugar no tenga chismosos antes de que Ruz pase por él. Deberás retornar como `int` el **largo** del camino si Ruz puede llegar al `Final` y si no puede retornar **-1**. **Debes modificarlo**
- `def dfs_iterativo_largo(inicio: Lugar, final: Lugar) -> int`: Este método recibe un nodo de inicio y uno final, los que representan desde donde se parte y hasta donde se quiere llegar en el grafo respectivamente. Deberás implementar el algoritmo de búsqueda **DFS** en esta y al igual que en la función anterior debes chequear que el lugar no tenga chismosos antes de que Ruz pase por el. Deberás retornar como `int` el **largo** del camino si Ruz puede llegar al `Final` y si no puede retornar **-1**. **Debes modificarlo**

Bonus 1

Como Ruz está muy ansioso por el cumpleaños sorpresa, quiere asegurarse de que exploró todas las posibilidades de recorrido. Para esto te pide que implementes el otro método de búsqueda **que no implementaste antes**.

- `def bfs_iterativo(inicio: Lugar, final: Lugar) -> int`: Debes implementar la función que antes no implementaste. **Debes modificarlo**
- `def dfs_iterativo(inicio: Lugar, final: Lugar) -> int`: Debes implementar la función que antes no implementaste. **Debes modificarlo**

Bonus 2

Además Ruz te pide que le muestres cómo es el camino que tiene que seguir para llegar hasta el final. Para esto, modifica **UNO** de los algoritmos de recorrido anteriores (el que tu prefieras). Se te entrega el diccionario `padres` que representa la relación entre el nodo y los vecinos, donde la llave es el nodo vecino y el valor es el nodo que se está revisando. Puedes revisar como está hecha la función `creado_camino` para entender mejor la estructura del diccionario pedido. El archivo a modificar se encuentra en `recorrido_campus.py` dentro de la carpeta `Parte2`:

- `def creador_camino(diccionario_padres: dict, final: Lugar) -> list`: Esta función recibe un diccionario y un nodo final, tal que la llave, el valor del diccionario y el final sean de tipo `Lugar`.

Retorna de forma ordenada la relación entre los diferentes nodos que componen el diccionario, junto al final. El formato del diccionario de padres es el siguiente:

```
{
    hijo: padre,
    ...
}
```

Donde hijo y padre son instancias de la clase `Cliente` o `None` No debes modificarlo

- `def bfs_iterativo_camino(inicio: Lugar, final: Lugar) -> list`: Debes implementar el recorrido **BFS** al igual que en las secciones pasadas, teniendo las mismas consideraciones respecto a los nodos con chismosos. Además, tienes que ir guardando en el diccionario entregado `padres` la relación entre los nodos. Las llaves del diccionario es el vecino de un nodo como instancia de `Lugar` y el valor es el nodo como instancia de `Lugar`. Una vez que armes correctamente este diccionario, retorna el valor que entrega la función `creador_camino(diccionario_padres, final)` al darle como primer argumento el diccionario, y el segundo el final entregado en esta función. **Debes modificarlo**
- `def dfs_iterativo_camino(inicio: Lugar, final: Lugar) -> list`: Debes implementar el recorrido **DFS** al igual que en las secciones pasadas, teniendo las mismas consideraciones respecto a los nodos con chismosos. Además, tienes que ir guardando en el diccionario entregado `padres` la relación entre los nodos. Las llaves del diccionario es el vecino de un nodo como instancia de `Lugar` y el valor es el nodo como instancia de `Lugar`. Una vez que armes correctamente este diccionario, retorna el valor que entrega la función `creador_camino(diccionario_padres, final)` al darle como primer argumento el diccionario, y el segundo el final entregado en esta función. **Debes modificarlo**

Requerimientos

- (3.00 pts) Primera Parte
  - (0.25 pts) define correctamente el método `__init__` del `Cliente`
  - (0.5 pts) define correctamente el método `llega_cliente` de la `FilaPizza`
  - (1 pts) define correctamente el método `alguien_se_cuela` de la `FilaPizza`
  - (0.5 pts) define correctamente el método `cliente_atendido` de la `FilaPizza`
  - (1 pts) define correctamente el método `__str__` de la `FilaPizza`
- (3.00 pts) Segunda Parte
  - (1.5 pts) define correctamente una de las funciones `bfs_iterativo` o `dfs_iterativo`
  - (1.5 pts) define correctamente una de las funciones `bfs_iterativo_largo` o `bfs_iterativo_largo`
- (0.1 pts) Bonus 1
  - (0.1 pts) define correctamente la funcion que le falto entre `bfs_iterativo` y `dfs_iterativo`
- (0.4 pts) Bonus 2
  - (0.4 pts) define correctamente una de las funciones `bfs_iterativo_camino` o `dfs_iterativo_camino`