Ayudantia Examen IIC2233

By: Mahund

Estructura de Datos (EDD)

Listas Ligadas

- Forma de almacenamiento de informacion en que cada dato (nodo) esta relacionado con el siguiente de forma lineal.
- Se implementa clase nodo con atributo *nodo siguiente*

Arboles Binarios

- Arbol **ordenado** en que cada padre tiene 2 y solo 2 hijos.
- No hay una forma única para definir esta estructura.
- Forma clasica:
  - nodo posee su valor, sus hijos y su ID.
  - arbol recibe valores y se encarga de crear nodo con este valor y ubicarlo en el lugar correcto.

Grafos

- Nodos conectados multidireccionalmente con la opcion de poseer un *sentido*.
- Implementacion es facil con atributos de conecciones (set), id y valor.

Busqueda en EDD

Recursiva vs Iterativo

- No existe un metodo mejor a otro, sino que dependera del objetivo que se quiera cumplir.

DFS vs BFS

- **DFS:** busca llegar al lugar más bajo de la rama antes de pasar a la siguiente (utiliza stacks).
- **BFS:** busca recorrer todo un *nivel jerarquico* del arbol antes de pasar a la siguiente (utiliza colas).

  > El codigo es muy similar, solo cambia el edd utilizado para almacenar los nodos a visitar.

Paths

- Manera como se representa una ubicacion en un computador.
- Varia segun sistema operativo.

Paths: absolutos vs relativos

- **Absoluto:** Parte desde la carpeta raiz del computador. (`/Carpeta_A/Carpeta_B`)
- **Relativo:** Parte desde la carpeta en que el archivo ejecutado se encuentra. (`../Carpeta_A`)

Comandos:

- `.` : carpeta actual.
- `..` : carpeta padre.
- `/` : separador de carpetas en windows.

OS en Pytohn

- Libreria **OS** permite trabajar con paths.
- `os.getcwd` : obtiene path absoluto
- **Falta rellenar aca**

Strings

- Estructura inmutable que reptresenta texto en python, por lo que cada accion crea un string nuevo.
- Permite utilizar *formats*: `f"Este texto es para la variable {nombre_variable}"`

Bytes

- **Bit:** unidad basica de informacion, solo es 0 o 1.
- **Byte:** conjunto de 8 bits (256 combinaciones).
- En *Python* son inmutables.

Bytearray

- Arreglo de bytes, diseñada para poder editar y trabajar sobre estos.
- Funcionamiento: `var = bytearray(bytes_normales)` .
- Tiene metodos similares a las listas: `extend, append, etc` .

Excepciones y Testing

Excepcion

- Evento que ocurre durante la ejecucion del programa e interrumple el flujo de este.
- Existen distintos tipos, que nos permiten personalizar el comportamiento del programa frente a esto.
- Para **manejar** las excepciones se utilizan: `try, except, else, finally` .

- **try:** bloque de codigo donde se piensa se levantara un error.
- **except:** bloque de codigo que se ejecute si se levanta cierto tipo de error.
- **else:** bloque de codigo si no se levanta ningun error.
- **finally:** bloque de codigo que se ejecuta siempre.

  > Si hay un return en el finally no se detiene la funcion (revisar documentacion.)

Lanzamiento de Excepciones

- Podemos lanzar nuestras propias excepciones para manejar errores propios.
- Mediante `raise` podemos levantar excepciones ya existentes cuando se desee.
- Podemos generar `excepciones personalizadas` heredando de excepciones existentes o de exception.

Testing

- Se utiliza libreria `unittest` para crear test unitarios, asi prueban una parte muy puntual del codigo.
- Se hereda de `TestCase` , crando metodos que *inicien* su nombre con **test**.
- Se revisaba utilizando `assets` , que haga coincidir lo esperado de lo recibido.
- Con metodos `setUp` y `tearDown` que permiten preparar todo lo necesario para los test y finalizar correctamente estos.

Threading

- Un thread es una secuencia pequeña de tareas que puede ser ejecutada por una misma seccion de codigo.
- Permiten simular programas en *paralelo*.
- Utilizado en movimiento de elementos, acciones o interfaz.

Uso de Threads

- Se utiliza de 2 formas:
  i. Herencia de Thread.
  ii. Funcion como target.
- Metodo start inicia la funcion del metodo run (que debe ser sobreescrita).

  > Run son acciones a ejecutar, start se encarga de otorgar caracteristicas de thread.

- `Timer` de threading sirve para ejecutar el codigo cada cierto tiempo.
- La **gran** desventaja de los threads es la **sincronizacion**.

Metodos utiles

- `thread.is_alive()` : retorna estado de vida de un thread.
- *daemon threads*: no impiden que programa principal termine.

- `join` : al llamar join de otro thread, el thread inicial detendra su ejecucion hasta que el otro thread termine.
- Señales:
  i. wait: espera que señal se activada.
  ii. set: activa la señal.
  iii. clear: desactiva la señal.

  > Cualquiera las puede utilizar, joins dependen del tread que lo posea.

- *lock*: permiten que threads no accedan al mismo tiempo al mismo recurso.

Funcional

- `map(funcion, iterable)` :
  - Recibe funcion e iterable, ejecutando la funcion en cada elemento del iterable y retorna generador de resultados.
- `reduce(funcion, iterable, primer_valor_inicial)` :
  - En la primera iteracion recibe los 2 primeros elementos del iterable, aplica una funcion a estos y almacena el resultado, para aplicar la funcion al resultado anterior con el siguiente elemento del iterable.
  - Se puede definir un primer valor inicial como tercer parametro.
- `filter(funcion_condicional, iterable)` :
  - Recibe funcion e iterable, aplica funcion a cada elemento del iterable y retorna generador de resultados en que la aplicacion de la funcion en él retorne True.

Modelacion

- Representacion grafica de como esta organizado un programa dentro del paradigma de programacion orientada a objetos.

Diagrama de Clase

- **Clase:** posee atributos (con properties) y metodos.
  - *Caracteristicas:*
    a. Public: +.
    b. Protected: #.
    c. Private: -.
    d. Package: ~.
  - Tipo de dato: establecer el tipo que sera el atributo.
  - Establecer multiplicidad *n*, *n…1*, etc.
  - Property: << `get/set` >> nombre: tipo.

    > +metodo_publico(param_1: str): None [recibe string y retorna None]

- **Relaciones:**
  i. Herencia: clase A hereda de clase B (flecha apunta de quien hereda [B]).

ii. Agregacion: existencia no depende de quien los contiene (flecha apunta a quien fue agregado [personas de una empresa]).
iii. Composicion: existencia depende de quien los contiene (flecha apunta a quien fue compuesto [ladrillos de una muralla]).

Multiherencia y problema de Diamante

- Uso de super() para evitar conflicto en herencias multiples.