Introducción a la Programación

Prof. Agustín Gravano

Primer semestre de 2022

Clase teórica 22: Clases y objetos (parte III)

Frutería Online

Para nuestra frutería, queremos modelar:

- Fruta $\longrightarrow$ `class` **`Fruta`** ✓
- Catálogo de frutas, a cargar de un CSV $\longrightarrow$ `class` **`Catalogo`** ✓
- Carrito de compras, con operaciones para agregar/sacar kg de fruta del carrito y para consultar el importe total $\longrightarrow$ `class` **`Carrito`** ✓

Hoy veremos:

1. Complejidad algorítmica de las operaciones provistas por estas clases.
2. Dos posibles interfaces de usuario que usan estas clases.

```python
class Carrito:
    def __init__(self):
        ''' Inicializa un carrito vacío. Requiere: nada. '''
        self.kg_fruta:Dict[Fruta, int] = dict()

    def __repr__(self) -> str:
        ''' Devuelve un string con los datos del carrito. Requiere: nada. '''
        return str(self.kg_fruta)

    def agregar(self, f:Fruta, p:int):
        ''' Agrega p kilos de la fruta f al carrito. Requiere: p>0. '''
        if f in self.kg_fruta:
            self.kg_fruta[f] = self.kg_fruta[f] + p
        else:
            self.kg_fruta[f] = p

    def sacar(self, f:Fruta, p:int):
        ''' Saca p kilos de la fruta f al carrito.
            Requiere: p>0, hay al menos p kilos de f. '''
        self.kg_fruta[f] = self.kg_fruta[f] - p
        if self.kg_fruta[f] == 0:
            self.kg_fruta.pop(f)

    def calcular_precio_total(self) -> float:
        ''' Calcula el precio total de la fruta en el carrito. Req: nada.'''
        vr:float = 0.0
        for fruta in self.kg_fruta:
            peso:int = self.kg_fruta[fruta]
            vr = vr + peso * fruta.precio
        return vr
```

Complejidad algorítmica

¿Cuál es la complejidad de las siguientes operaciones? (N es la cantidad de claves en el diccionario `self.kg_fruta`)

```
def agregar(self, f:Fruta, p:int):
    if f in self.kg_fruta:                                  O(N)
        self.kg_fruta[f] = self.kg_fruta[f] + p             O(N)
    else:
        self.kg_fruta[f] = p                                O(N)
```

agregar es $\max(O(N), O(N), O(N)) = O(N)$

```
def sacar(self, f:Fruta, p:int):
    ''' Saca p kilos de la fruta f del carrito.
        Requiere: p>0, hay al menos p kilos de f. '''
    self.kg_fruta[f] = self.kg_fruta[f] - p             O(N)
    if self.kg_fruta[f] == 0:                           O(N)
        self.kg_fruta.pop(f)                            O(N)
```

sacar es $O(N) + \max(O(N), O(N)) = O(N)$

```
def calcular_precio_total(self) -> float:
    vr:float = 0.0                                      O(1)
    for fruta in self.kg_fruta:                         O(N) iteraciones
        peso:int = self.kg_fruta[fruta]                     O(N)
        vr = vr + peso * fruta.precio                       O(1)
    return vr                                           O(1)
```

calcular_precio_total es $O(1) + O(N) * (O(N) + O(1)) + O(1) = O(N^2)$ 🤨

Complejidad algorítmica

¿Cómo podríamos lograr que `calcular_precio_total` tenga $O(N)$?

En la definición que vimos, en cada iteración del ciclo consultamos el valor de la clave actual en el diccionario. Como esa operación de consulta tiene complejidad lineal en el peor caso, el algoritmo entero es cuadrático.

Los diccionarios tienen un iterador especial, `d.items()`, que en cada iteración nos da el par (clave, valor) al mismo tiempo. Ejemplo:

```
from typing import Dict
d:Dict[str,str] = dict()
d['gato'] = 'felino'
d['ratón'] = 'roedor'
d['tigre'] = 'felino'
for (k, v) in d.items():
      # En cada iteración obtenemos un par clave-valor del diccionario.
      print("El", k, "es un", v)
```

Esto imprime:

```
El gato es un felino
El ratón es un roedor
El tigre es un felino
```

(en algún orden)

Complejidad algorítmica

Entonces, cambiamos este código:

```
def calcular_precio_total(self) -> float:
    vr:float = 0.0                                  O(1)
    for fruta in self.kg_fruta:                     O(N) iteraciones
        peso:int = self.kg_fruta[fruta]                 O(N)
        vr = vr + peso * fruta.precio                   O(1)
    return vr                                       O(1)
```

por este otro:

```
def calcular_precio_total(self) -> float:
    vr:float = 0.0                                  O(1)
    for fruta, peso in self.kg_fruta.items():       O(N) iteraciones
        vr = vr + peso * fruta.precio                   O(1)
    return vr                                       O(1)
```

Ahora, `calcular_precio_total` tiene complejidad:

$$O(1) + O(N) * O(1) + O(1) = O(N)$$

Complejidad algorítmica

¿Cómo podríamos lograr que `calcular_precio_total` tenga $O(1)$?

Idea: Llevar la cuenta del precio total cuando agregamos o sacamos fruta.

```
    def __init__(self):
        # [...]
        self.precio_total:float = 0.0

    def agregar(self, f:Fruta, p:int):
        # [...]
        self.precio_total = self.precio_total + f.precio * p

    def sacar(self, f:Fruta, p:int):
        # [...]
        self.precio_total = self.precio_total - f.precio * p

    def calcular_precio_total(self) -> float:
        return self.precio_total
```

Estas operaciones son todas O(1). Entonces, no empeoran la complejidad de los métodos `__init__`, `agregar` y `sacar`.

¡La nueva implementación de `calcular_precio_total` es $O(1)$!

Atributos relacionados

Ahora los atributos `kg_fruta` y `precio_total` están relacionados:

$$\texttt{precio\_total} = \sum_{\texttt{f in kg\_fruta}} \texttt{kg\_fruta[f]} \times \texttt{f.precio}$$

¡Cuidado! Debemos mantener esa relación en todos los métodos. Ej:

```
def agregar(self, f:Fruta, p:int):
    ''' Agrega p kilos de la fruta f al carrito. Requiere: p>0 '''
    if f in self.kg_fruta:
        self.kg_fruta[f] = self.kg_fruta[f] + p
    else:
        self.kg_fruta[f] = p
    self.precio_total = self.precio_total + f.precio * p
```

```
def sacar(self, f:Fruta, p:int):
    ''' Saca p kilos de la fruta f al carrito.
        Requiere: p>0, hay al menos p kilos de f. '''
    self.kg_fruta[f] = self.kg_fruta[f] - p
    if self.kg_fruta[f] == 0:
        self.kg_fruta.pop(f)
    self.precio_total = self.precio_total - f.precio * p
```

Complejidad algorítmica

Esto explica por qué `len(xs)` es $O(1)$ en listas:

list

The Average Case assumes parameters generated uniformly at random.

Internally, a list is represented as an array; the largest costs come from growing bey
somewhere near the beginning (because everything after that must move). If you ne

Operation	Average Case	Amortized Worst Case
Copy	O(n)	O(n)
Append[1]	O(1)	O(1)
Pop last	O(1)	O(1)
Pop intermediate[2]	O(n)	O(n)
Insert	O(n)	O(n)
Get Item	O(1)	O(1)
...		
x in s	O(n)	
min(s), max(s)	O(n)	
Get Length	O(1)	O(1)

Las listas guardan su longitud en un atributo, y todos sus métodos (`append`, `pop`, etc.) lo mantienen actualizado siempre. Así, `len(xs)` siempre se resuelve en tiempo constante.

Archivos de la frutería online

- `fruta.py`, `carrito.py`. `catalogo.py`: Clases que encapsulan las entidades principales de nuestro proyecto: frutas, carrito de compras y catálogo de frutas.
- `fruta-test.py`, `carrito-test.py`, `catalogo-test.py`: Testing de unidad de las clases `Fruta`, `Carrito` y `Catalogo`.
- `frutas-para-testing1.csv`, `frutas-para-testing2.csv`: Archivos CSV preparados para testear la clase `Catalogo`.
- `main-txt.py`: Programa principal del proyecto. Implementa una interfaz interactiva en modo texto, para usar un carrito de compras.
- `main-gui.py`: Programa principal del proyecto. Implementa una interfaz interactiva en modo gráfico, para usar un carrito de compras.

  Este código se da solo a modo ilustrativo, y **no forma parte de los contenidos de la materia**. A quienes les interese el tema, recomendamos leer algún tutorial, como éste por ejemplo: `https://www.adictosaltrabajo.com/2020/06/30/interfaces-graficas-en-python-con-tkinter/`. Hay mucho material online para aprender a usar `tkinter` y otros módulos para armar GUIs.

Repaso de la clase de hoy

- Complejidad de métodos en clases.
- Programa principal e interfaces de usuario.

Bibliografía complementaria:

- APPP2, capítulos 12, 13 y 14 (excepto 14.9).
- HTCSP3, capítulos 15 y 16.

Con lo visto, ya pueden resolver la Guía de Ejercicios 9 completa.