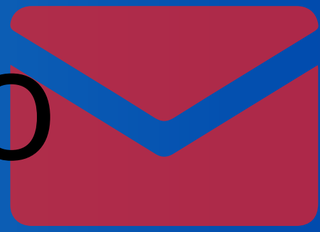
Cliente de Correo Electrónico



Implementación en Python con POO, estructuras de datos y algoritmos

Docente: Bianco, Ángel Leonardo.

Integrantes Grupo 16: Soto Lucía, Lepin Ian , Gómez Sofia.

Comisión 3

CLIENTE DE CORREO ELECTRÓNICO

Implementación completa en Python aplicando Programación Orientada a Objetos, estructuras de datos avanzadas, algoritmos de búsqueda y gestión de mensajes.

¿Qué aprenderás?

.Cómo funciona un cliente de correo por dentro

.Cómo se organiza la información

.Cómo se envían y reciben mensajes

.Qué estructuras (árbol, grafo, heap) usa un sistema real

¿Qué es este proyecto?

Este proyecto simula el funcionamiento REAL de un cliente de correo, como Gmail o Outlook, pero simplificado para entender sus bases internas.

Incluye:

Usuarios: cada uno con sus carpetas y mensajes.

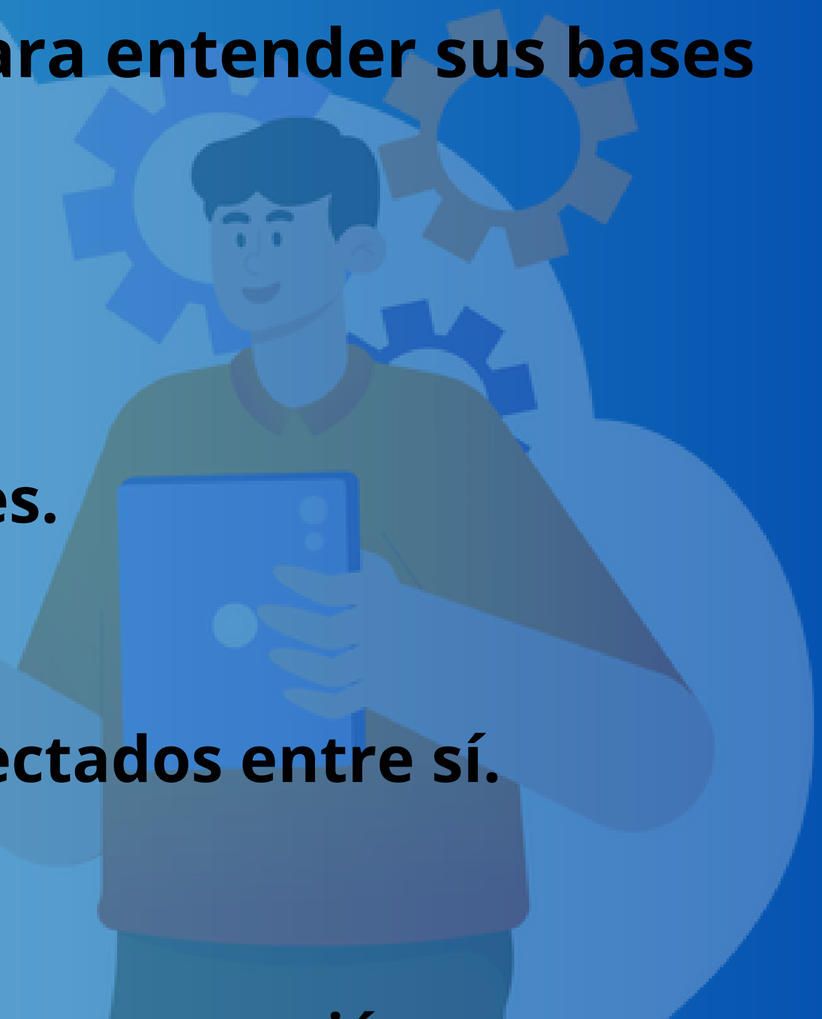
Servidores: que procesan y entregan el correo.

Filtros: para clasificar automáticamente mensajes.

Carpetas y subcarpetas: estructura tipo árbol.

Colas de prioridad: para manejar urgentes.

Grafo: para representar la red de servidores conectados entre sí.



Por qué es importante:

→ Permite ver cómo se combinan varios conceptos de programación para crear un sistema complejo.

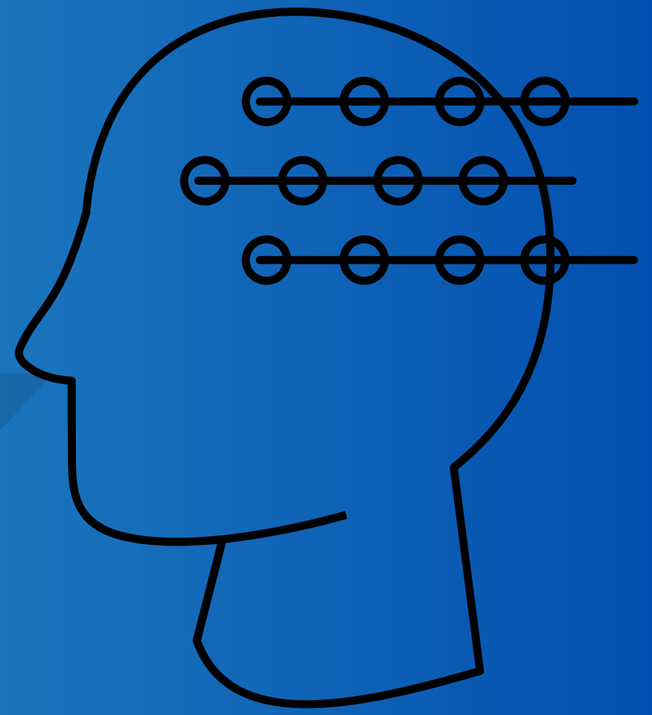
¿Por qué es importante?

Este proyecto es clave porque combina varios temas difíciles de forma práctica:

- **POO (Programación Orientada a Objetos)**
  Creación de clases, atributos, métodos y relación entre módulos.
- **Árboles**
  Permiten crear carpetas dentro de carpetas sin límite.
- **Grafos**
  Los servidores están conectados como nodos en una red.
- **Algoritmos BFS y DFS**
  Para buscar rutas entre servidores.
- **Heap (Cola de prioridad)**
  Para ordenar mensajes según urgencia.

Arquitectura general del sistema

El sistema está dividido en módulos, cada uno con una función específica:



Interfaces: definen reglas obligatorias para las clases.

Mensaje: la unidad de información del correo.

Carpetas: árbol jerárquico para almacenar mensajes.

Usuario: dueño de carpetas y de mensajes.

Servidor de correo: gestiona el envío/recepción.

Filtros: clasifican automáticamente.

Conexiones entre servidores: grafo que permite rutas de envío.

¿Qué es una INTERFAZ en Python?

Una interfaz funciona como un contrato.
No dice cómo se hace algo, sino qué debe existir.

En Python se usa:

```
class ICarpeta(ABC):
    @abstractmethod
    def agregar_mensaje(self, mensaje):
        pass
```

Esto significa:

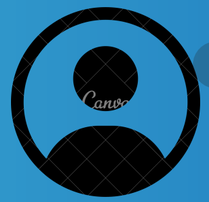
“Toda Carpeta debe tener un método agregar_mensaje”.
Si no lo implementás → ERROR.

Ventaja:
Garantiza que todas las clases funcionen igual y tengan los mismos métodos.

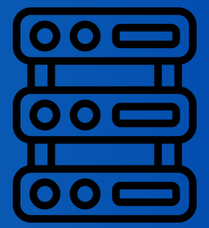
Interfaces del sistema

ICarpeta: 

.agregar mensajes
.buscar mensajes
.listar contenido
.mover mensajes
.eliminar mensajes

IUsuario: 

.enviar mensajes
.recibir mensajes
.listar carpetas

IServidorCorreo: 

.registrar usuarios
.enviar correo
.entregar al destinatario

Importante:

Las interfaces ayudan a que el código sea ordenado y fácil de mantener.

Clase Mensaje

Representa un correo electrónico.

Cada mensaje tiene:

.asunto
.cuerpo
.remitente
.prioridad (1 urgente → 5 baja)

Código:

```
class Mensaje:
    def __init__(self, asunto, cuerpo, remitente, prioridad=5):
        self.asunto = asunto
        self.cuerpo = cuerpo
        self.remitente = remitente
        self.prioridad = prioridad
```

Ejemplo:

Mensaje urgente informando un cambio de reunión → prioridad 1.

Newsletter semanal → prioridad 5.

Clase Carpeta (Árbol recursivo)

Una carpeta puede tener:

- ✓ **muchos mensajes**
- ✓ **muchas subcarpetas**
- ✓ **y esas subcarpetas también pueden tener más…**

Esto forma un árbol, igual que las carpetas del explorador de archivos.

Las carpetas se guardan en una lista:

```
self._subcarpetas = []
```

Ejemplo real:

.Bandeja de entrada

.Trabajo

.Proyecto X

.Personal

.Viajes

Recursividad en las carpetas

La recursividad se usa cuando algo puede contener “más de sí mismo”.

Ejemplo:

Para buscar un mensaje, la carpeta revisa dentro de ella… y luego dentro de cada subcarpeta… y así sucesivamente.

Esto permite encontrar un mensaje aunque esté 5 niveles adentro

Agregar mensajes

Código:

```
def agregar_mensaje(self, mensaje):
    self._mensajes.append(mensaje)
```

agrega el mensaje a la lista interna.

Esto ocurre cuando:

.El usuario recibe un mensaje

.Un filtro decide que corresponde a otra carpeta

Es la base del almacenamiento del sistema.

Buscar mensajes – Eliminar y mover mensajes

La búsqueda funciona así:

.Revisa los mensajes de la carpeta actual.

.Si no lo encuentra:
→ llama al mismo método pero en cada subcarpeta (recursión).

.Si lo encuentra → lo devuelve.

Es un algoritmo de búsqueda en árbol profundo.

Eliminar:

.Primero busca el mensaje en la carpeta actual.

.Si no está → busca en subcarpetas recursivamente.

.Si lo encuentra → lo saca de la lista.

Mover:

.Elimina el mensaje de la carpeta original.

.Usa agregar_mensaje() para insertarlo en la nueva.

📌 Se parece a cortar y pegar.

Clase Usuario

Cada usuario tiene:

- .su nombre
- .un conjunto de carpetas
- .una cola de prioridad para urgentes
- .métodos para enviar/recibir

Al crear un usuario, automáticamente se crean sus carpetas base:

- .Bandeja de entrada
- .Enviados
- .Spam

Recibir mensajes

Si el mensaje es urgente (prioridad 1):

heapq.heappush(self._urgentes, (mensaje.prioridad, mensaje))

Esto lo mete en la cola de prioridad.

Luego SIEMPRE se guarda en:

Bandeja de entrada

📌 Tener la cola de urgentes permite que el usuario pueda verlos primero.

Enviar mensajes: El usuario NO envía el mensaje directamente.

Llama al servidor:

servidor.enviar_mensaje(...)

Esto es correcto porque:

✓ un cliente de correo no contacta al otro usuario directamente

✓ siempre pasa por un servidor

Cola de Prioridades

La prioridad determina el orden:

.prioridad 1 → primero
.prioridad 2 → segundo
...
.prioridad 5 → último

heapq es eficiente para extraer el elemento más prioritario.

Servidor de Correo

Es el corazón del sistema.

Se encarga de:

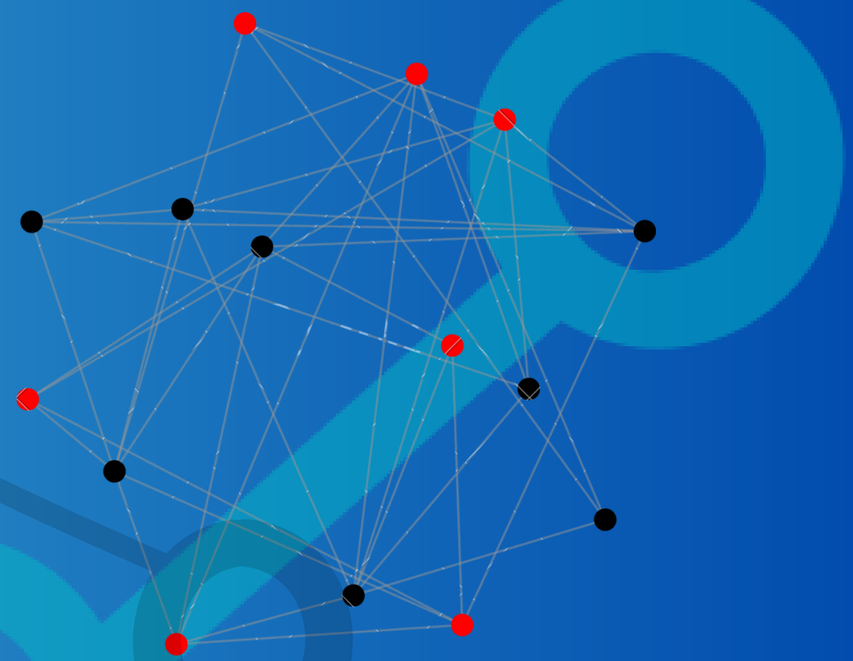
- ✓ **Registrar usuarios**
- ✓ **Enviar correos**
- ✓ **Clasificar con filtros**
- ✓ **Entregar mensajes**
- ✓ **Conexiones con otros servidores**

📌 **Es el equivalente a Gmail, Outlook, Yahoo Mail, etc.**

Conectar servidores (grafo)

Cada servidor puede conectarse a otros:

```
def conectar(self, servidor):
        self.conexiones.append(servidor)
        servidor.conexiones.append(self)
```



Esto crea un grafo NO dirigido.

Sirve para simular redes reales, donde un mail puede pasar por varios servidores antes de llegar.

Filtros automáticos

Funcionan analizando el asunto del mensaje.

Ejemplos típicos:

_Si contiene “promoción”, “gratis”, “oferta” → va a Spam.

_Si contiene “trabajo”, “proyecto” → va a Trabajo.

Permite que la bandeja de entrada esté más ordenada sin intervención manual.

Envío completo del mensaje

El flujo:

1)Usuario escribe y manda.

2)Servidor crea el objeto Mensaje.

3)Aplica filtros (Spam, Trabajo, etc).

4)Decide si pasa por otros servidores (grafo).

5)Llega al servidor destino.

6)El destinatario lo recibe.

7)Se coloca en la carpeta correcta.

Es una simulación simplificada del proceso real de envío de emails.

BFS: Ruta entre servidores

BFS (Breadth First Search) busca el camino más corto:

_Ideal para encontrar la ruta más eficiente.

_Avanza por niveles: primero vecinos, luego vecinos de vecinos.

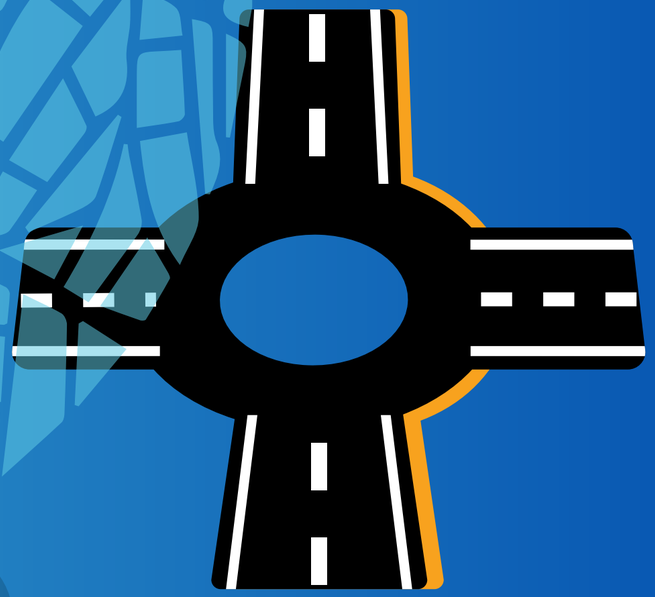
_Perfecto si queremos minimizar el recorrido.

Ejemplo:

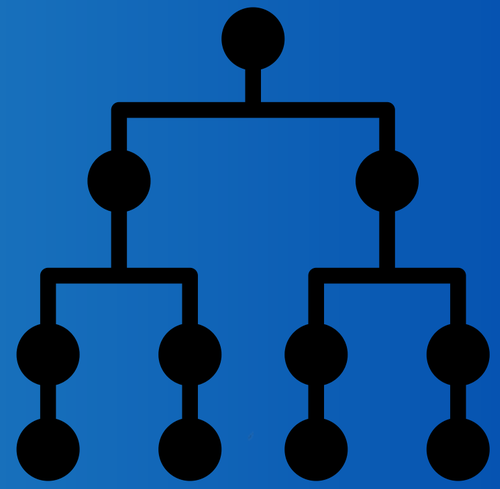
Si A → B → C → D

y A también conecta con Z → X → D

BFS va por A → B → C → D (más corto).



DFS: Búsqueda profunda



DFS explora “a fondo” antes de volver atrás.

Sirve para:

_explorar toda la red

_detectar conexiones profundas

_depurar errores

_saber si hay caminos alternativos

⚙️ ANÁLISIS DE EFICIENCIA (Big-O)

📁 1. Carpetas (Árbol recursivo)

Buscar mensaje → O(n)

Recorre la carpeta y todas sus subcarpetas.

Eliminar / mover mensaje → O(n)

Debe encontrarlo antes de moverlo

Agregar mensaje → O(1)

Insertar al final de la lista es constante.

✉️ 3. Filtros automáticos

Revisión de filtros → O(F · L)

F = número de filtros

L = longitud del asunto

Revisa palabra por palabra si coincide con el asunto.

⚡ 2. Cola de prioridad (heap)

Insertar mensaje urgente → O(log k)

Obtener mensaje urgente → O(log k)

(k = cantidad de urgentes)

🌐 4. Grafo de servidores

BFS (ruta más corta) → O(V + E)

DFS (búsqueda profunda) → O(V + E)

V = servidores, E = conexiones

Conclusiones

Este proyecto:

✓ El sistema implementado simula el funcionamiento real de un cliente de correo electrónico, utilizando conceptos avanzados de programación.

✓ Combina estructuras fundamentales: POO, árboles, grafos, recursividad y colas de prioridad, permitiendo un flujo completo de envío y recepción de mensajes.

✓ La arquitectura modular facilita la escalabilidad, el mantenimiento y la incorporación de nuevas funciones como filtros, más carpetas o servidores adicionales.

✓ El uso de algoritmos de búsqueda (BFS y DFS) permite representar rutas y conexiones de forma similar a redes reales de servidores de correo.

✓ La clasificación automática, el manejo de prioridades y el almacenamiento jerárquico muestran un sistema robusto, organizado y funcional.