Problema 1: Sistema de navegación de páginas web

Objetivo:

Simular cómo un navegador permite navegar hacia atrás, adelante y cargar nuevas páginas, usando estructuras de datos adecuadas.

Diseño

Usaremos dos pilas:

- back_stack: almacena el historial hacia atrás.
- forward_stack: almacena el historial hacia adelante.

Métodos:

- load_page(url): limpia forward_stack, agrega la actual a back_stack, y carga una nueva.
- go_back(): mueve la página actual a forward_stack y recupera la anterior de back_stack.
- go_forward(): mueve la actual a back_stack y recupera una de forward_stack.


Inicio
¿Existe una página actual?
Sí
No
Agregar la página actual a la pila 'back_stack'
Limpiar la pila 'forward_stack'
Establecer la nueva URL como la página actual
Fin del proceso de carga

Problema 2: Sistema por orden de llegada

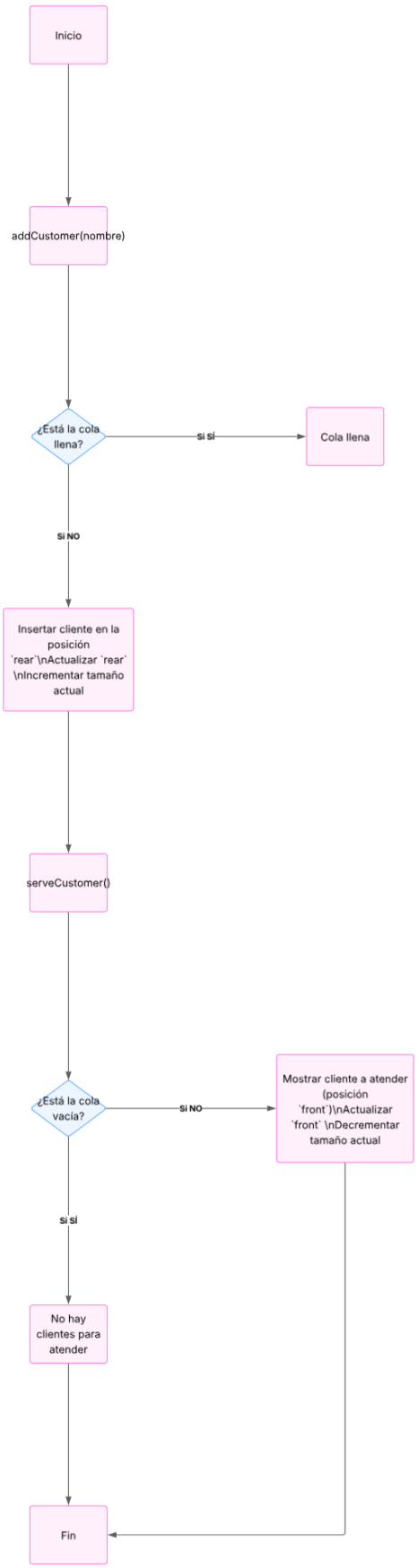
Objetivo:

Diseñar un sistema para una entidad pública (ej. mesa de ayuda) donde los clientes sean atendidos **en el orden que llegan**, usando una cola FIFO.

Diseño del sistema

Estructura ideal para este sistmea:

- Usaremos una **cola circular** para manejar el flujo de clientes.
- Estructura principal: Queue
    - Arreglo de clientes
    - Índice de inicio (front)
    - Índice de final (rear)
    - Tamaño máximo y actual


Inicio
addCustomer(nombre)
¿Está la cola llena?
Si Sí
Cola llena
Si NO
Insertar cliente en la posición `rear`\nActualizar `rear` \nIncrementar tamaño actual
serveCustomer()
¿Está la cola vacía?
Si NO
Mostrar cliente a atender (posición `front`)\nActualizar `front` \nDecrementar tamaño actual
Si Sí
No hay clientes para atender
Fin

Problema 3: Autocompletar con Trie

Objetivo:

Crear una función que, dado un prefijo, devuelva todas las palabras que lo comiencen. Ideal para buscadores, formularios, etc.

Diseño del sistema

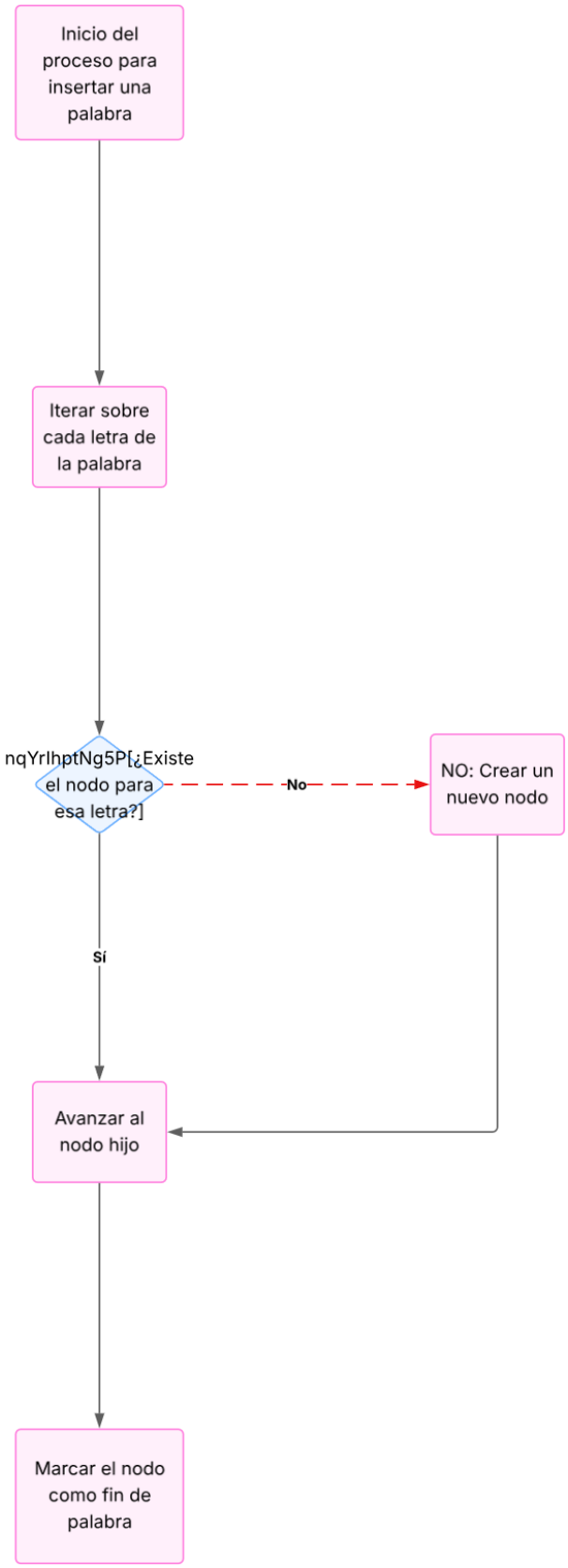
Usaremos una estructura de datos **Trie** (árbol de prefijos), que permite búsquedas eficientes por prefijo.

Estructura principal

- Cada nodo del trie representa una letra.
- Cada nodo tiene:
    - Un diccionario de hijos (children)
    - Una marca si es final de palabra (is_end_of_word)

Métodos requeridos

- insert(word) → Inserta una palabra al trie.
- autocomplete(prefix) → Devuelve todas las palabras que comienzan con ese prefijo.


Inicio del proceso para insertar una palabra
Iterar sobre cada letra de la palabra
nqYrlhptNg5P[¿Existe el nodo para esa letra?]
No
NO: Crear un nuevo nodo
Sí
Avanzar al nodo hijo
Marcar el nodo como fin de palabra

Problema 4: Recomendador tipo “los que compraron X también compraron Y”

Objetivo:

Construir un sistema que sugiera productos a un usuario basándose en las compras de otros usuarios similares.

Diseño del sistema

Vamos a usar:

- Un diccionario (dict) que almacena, por cada usuario, la lista de productos comprados.
- Otro diccionario (dict) que relaciona productos entre sí según co-ocurrencia: "producto X" → {"producto Y": número de veces que X y Y fueron comprados juntos}

Este enfoque es similar al usado por Amazon y otros motores de recomendación colaborativos simples.

Funciones necesarias

- add_purchase(usuario, producto)
  Agrega una compra y actualiza relaciones de co-compra.
- get_recommendations(usuario)
  Sugiere productos que el usuario aún no ha comprado, pero que otros compran junto a los suyos.


Inicio de recomendaciones
Obtener productos comprados por el usuario
Revisar las relaciones de co-compra
Acumular productos relacionados que el usuario no tiene
Ordenar recomendaciones por frecuencia de co-compra
Devolver la lista de sugerencias
Fin del proceso de recomendación