UNIVERSIDAD DE CALDAS

Facultad de Ingenierías

Análisis y Estrategias de Solución:

Sistema de Gestión de Bibliotecas (SGB)

Elaborado por:

Juan Camilo Cruz Parra

Jaime Andrés Cardona Díaz

Daner Alejandro Salazar Colorado

Presentado a:

Mario Alejandro Bravo Ortiz

Asignatura:

Técnicas de Programación

Manizales, Caldas

2025

Índice general

1. Análisis y Estrategias de Solución

En este documento explicamos las decisiones técnicas y las estrategias que usamos para construir el Sistema de Gestión de Bibliotecas (SGB). La idea es justificar por qué elegimos ciertas estructuras y algoritmos para cumplir con lo que se pedía.

1.1 Carga de Datos

El sistema necesita leer información de diferentes fuentes. En lugar de obligar a usar un solo tipo de archivo, creamos una función que detecta automáticamente si el archivo es CSV o JSON.

Solución implementada: El sistema mira la extensión del archivo y decide cómo leerlo. Esto es muy útil porque nos permite trabajar con archivos de Excel (CSV) o con datos que vienen de aplicaciones web (JSON) sin tener que transformar nada manualmente.

```
# Detectamos el formato autom ticamente
if ruta.endswith('.csv'):
    reader = csv.DictReader(f)  # Para archivos tipo Excel
elif ruta.endswith('.json'):
    data = json.load(f)         # Para archivos tipo Web
```

Listing 1.1: Detección automática de formato de archivo

1.2 Búsquedas Rápidas: Doble Lista

Buscar un libro en una lista desordenada es lento si se tienen muchos datos. Para solucionarlo, usamos una estrategia de dos listas.

Solución implementada:

- **Inventario general:** guarda los libros en el orden en que llegaron.
- **Inventario ordenado:** guarda los mismos libros pero ordenados por ISBN.

Gracias a la lista ordenada, podemos usar la **búsqueda binaria**. En lugar de revisar libro por libro, el sistema divide la lista y encuentra el libro mucho más rápido.

```
# B squeda Binaria: Mucho m s r pida que revisar uno por uno
while low <= high:
    mid = (low + high) // 2
    if inventario[mid].isbn == isbn:
        return mid  #  Encontrado !
```

```
    elif inventario[mid].isbn < isbn:
        low = mid + 1
    else:
        high = mid - 1
```

Listing 1.2: Búsqueda binaria sobre inventario ordenado

1.3 Historial de Préstamos (Pilas)

Para el historial de lo que ha leído un usuario, se analizó cómo se usa esa información. Generalmente, uno quiere ver lo último que pidió prestado.

Solución implementada: Usamos una **pila** (*stack*). Esta estructura funciona como una pila de platos: el último que se pone es el primero que se saca. Así, siempre tenemos a la mano el préstamo más reciente sin tener que buscar en todo el historial.

```
# Con la pila, el  ltimo  dato siempre est  accesible
def apilar(self, item):
    self.items.append(item)

def desapilar(self):
    return self.items.pop()
```

Listing 1.3: Operaciones básicas sobre la pila de historial

1.4 Reservas y Lista de Espera

Cuando un libro se agota, se necesita una forma de organizar a las personas que lo quieren.

Solución implementada: Se creó una lista de espera. Lo interesante es que el sistema revisa esta lista automáticamente: apenas alguien devuelve el libro, el sistema se lo asigna de una vez a la siguiente persona que estaba esperando. De esta forma no se pierde tiempo y el libro circula rápido.

1.5 Algoritmos de Ordenamiento

No se usa el mismo método para ordenar todo; depende de qué se esté ordenando.

- **Merge Sort (para reportes de valor):** se usa para ordenar todos los libros por precio. Es un algoritmo muy confiable que no se pone lento ni siquiera si la lista está muy desordenada.
- **Insertion Sort (para historiales):** se usa para ordenar los préstamos por fecha. Como los historiales se van llenando poco a poco, suelen estar casi ordenados. En este caso, este

algoritmo es más rápido y sencillo que otros más complejos.

1.6 Optimizando la Estantería

Se tenía el reto de elegir los mejores libros para una estantería que no aguanta mucho peso.

Solución implementada (backtracking): El sistema prueba diferentes combinaciones de libros para ver cuál suma más valor sin pasarse del peso límite. Para que no se tarde mucho probando combinaciones poco útiles, se añadió una poda: si un libro pesa muy poco (menos de 0.5 kg), se descarta de una vez. Esto hace que el cálculo sea mucho más rápido.

```
# Si el libro es muy ligero, lo saltamos para ahorrar tiempo
if libro.peso < 0.5:
    continue

if peso_actual + libro.peso <= max_peso:
    # Probamos si esta combinaci n funciona
    ...
```

Listing 1.4: Poda de combinaciones poco relevantes

1.7 Recursividad

Se usan funciones recursivas (que se llaman a sí mismas) para algunos cálculos.

- **Recursión de pila:** se usa para encontrar el libro más ligero. Es una forma limpia de recorrer la lista paso a paso.
- **Recursión de cola:** se usa para calcular el peso promedio. Es una técnica eficiente que ayuda a que el programa no consuma tanta memoria mientras hace las sumas, ya que la llamada recursiva es la última operación.

1.8 Conclusión

Cada decisión técnica tiene una razón. Se eligieron estas estructuras y algoritmos para que el sistema sea rápido, fácil de usar y capaz de manejar diferentes situaciones sin problemas. La combinación de doble lista para búsquedas rápidas, pilas para historiales, listas de espera automatizadas, algoritmos de ordenamiento adaptados al contexto y backtracking optimizado con poda permite que el Sistema de Gestión de Bibliotecas (SGB) cumpla los requerimientos funcionales con un diseño sólido y mantenible.