

**UNIVERSIDAD PRIVADA DEL NORTE**

**FACULTAD DE INGENIERIA DE SISTEMAS**

**Integrantes:**

* Harold Antony Aiquipa Arce
* Jimmy Jack Mechato More
* Alonso Manuel Reyes Lopez

**Docente:**

* Jorge Chicana Aspajo

**Curso:**

* Análisis de Algoritmos y Estrategias

de Programación

**LIMA – PERÚ**

**2024**

**CONTENIDO**

[I.INTRODUCCIÓN 3](#_Toc171292855)

[1.1. Definición de objetivos 3](#_Toc171292856)

[1.2. Descripción del Problema 4](#_Toc171292857)

[II.ANALISIS DE LA SOLUCIÓN 5](#_Toc171292858)

[2.1. Elección de la solución 5](#_Toc171292859)

[2.2. Herramientas de Ingeniería 5](#_Toc171292860)

[III.DESARROLLO DE LA SOLUCIÓN 7](#_Toc171292861)

[3.1. Formulación del Pseudocódigo 7](#_Toc171292862)

[3.2. Implementación del Algoritmo 7](#_Toc171292863)

[IV.RESULTADOS 26](#_Toc171292864)

[4.1. Análisis empírico 49](#_Toc171292865)

[4.2. Evaluación 51](#_Toc171292866)

[V.CONCLUSIONES 52](#_Toc171292867)

[VI.REFERENCIAS O BIBLIOGRAFÍA 54](#_Toc171292868)

[VII. ANEXOS 54](#_Toc171292869)

# I.INTRODUCCIÓN

## 1.1. Definición de objetivos

Objetivo general

Desarrollar un sistema de cajero automático multifunción que permita realizar operaciones de retiro, depósitos, transferencias, pago de servicios, consulta de saldos y movimientos, gestionando múltiples cajeros ubicados en distintos lugares, con la capacidad de añadir nuevos clientes, actualizar dispensadores de billetes, y asegurar la validez y seguridad de todas las transacciones realizadas.

Objetivos específicos

1. Implementar un sistema de menús y submenús intuitivos que permitan al usuario navegar fácilmente entre las distintas funcionalidades del cajero automático.
2. Desarrollar algoritmos eficientes para la gestión de transacciones, incluyendo retiros, depósitos, transferencias y pagos de servicios, garantizando la actualización correcta de saldos de cuentas y dispensadores de billetes.
3. Validar cada operación realizada en el sistema, asegurando la correcta ejecución de las transacciones y proporcionando mensajes claros al usuario sobre el éxito o fracaso de las mismas.
4. Incorporar algoritmos de ordenamiento y búsqueda para optimizar la gestión de clientes y sus respectivas transacciones.
5. Implementar técnicas de programación como fuerza bruta, algoritmos voraces o programación dinámica para optimizar el rendimiento del sistema.
6. Diseñar el sistema para que sea tolerante a fallos, garantizando la integridad y disponibilidad de los datos incluso en caso de errores o fallos del sistema.
7. Proporcionar un desglose detallado de billetes en las operaciones de retiro y depósito, actualizando correctamente el saldo del cliente y el dispensador de billetes.
8. Registrar todas las operaciones realizadas con fecha y hora, permitiendo la consulta de movimientos con un sistema de colores opcional para diferenciar entre operaciones de ingreso y salida.
9. Facilitar la adición de nuevos clientes y la actualización de sus datos, incluyendo la creación de cuentas y la asignación de contraseñas seguras.
10. Asegurar que el sistema sea intuitivo y fácil de usar, con una interfaz clara y mensajes comprensibles para el usuario.

## 1.2. Descripción del Problema

En la actualidad, los cajeros automáticos son una herramienta fundamental en el sistema bancario, permitiendo a los clientes realizar diversas operaciones financieras de manera rápida y conveniente. Sin embargo, muchos cajeros automáticos carecen de funcionalidades avanzadas y presentan problemas de gestión eficiente, seguridad y usabilidad. La problemática se centra en la falta de sistemas integrados que permitan una gestión eficaz de múltiples cajeros, la actualización correcta de saldos y dispensadores de billetes, y la validación y seguridad de las transacciones.

El campo de acción de este proyecto se enfoca en la mejora de la infraestructura y funcionalidad de los cajeros automáticos, abordando las siguientes problemáticas específicas:

1. **Gestión Inadecuada de Cajeros**: Los sistemas actuales no permiten una gestión centralizada y eficiente de múltiples cajeros ubicados en distintos lugares, lo que dificulta la administración y el mantenimiento de los mismos.
2. **Actualización de Dispensadores**: La falta de una actualización correcta y en tiempo real de los dispensadores de billetes puede llevar a problemas de disponibilidad de efectivo y errores en las transacciones de retiro y depósito.
3. **Validación de Operaciones**: Los sistemas existentes no siempre validan adecuadamente las transacciones, lo que puede resultar en errores, transacciones fallidas y una mala experiencia para el usuario.
4. **Seguridad y Tolerancia a Fallos**: Es crucial garantizar que todas las operaciones sean seguras y que el sistema sea capaz de manejar fallos sin comprometer la integridad de los datos ni la satisfacción del cliente.
5. **Usabilidad y Experiencia del Usuario**: Muchos cajeros automáticos presentan interfaces complicadas y mensajes poco claros, lo que dificulta la realización de operaciones por parte del usuario.

# II.ANALISIS DE LA SOLUCIÓN

## 2.1. Elección de la solución

|  |  |
| --- | --- |
| *Algoritmo: “Ordenamiento de Burbuja*” | |
| *Ventajas* | *Desventajas* |
| |  | | --- | | - El algoritmo de burbuja es fácil de entender e implementar. |  |  | | --- | |  | | -Mantiene el orden relativo de los elementos iguales. | |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  | | --- | | -En listas muy pequeñas o casi ordenadas, puede ser eficiente en términos de tiempo debido a su simplicidad. |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | |  | | --- | | -No requiere estructuras de datos adicionales. |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | |  | | | | | |  | | --- | | -Realiza muchas comparaciones. |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | |  | | --- | | -Hay muchos otros algoritmos de ordenamiento como el Quicksort o mergesort, que son mucho más eficientes para listas grandes. |  |  |  |  | | --- | --- | --- | | |  | | --- | | -El uso intensivo de intercambios puede tener un impacto negativo en la memoria. |  |  | | --- | | -No es adecuado para conjuntos de datos grandes. | | | | |

## 2.2. Herramientas de Ingeniería

#### IDE: Spyder

Spyder es un entorno de desarrollo integrado (IDE) para Python, especialmente diseñado para la ciencia de datos y la computación científica. Algunas de sus características clave incluyen:

* **Interfaz gráfica amigable**: Spyder proporciona una interfaz de usuario fácil de usar que incluye un editor de código, una consola IPython y paneles para explorar variables, archivos y gráficos.
* **Editor de código avanzado**: Incluye un editor de código con funciones como resaltado de sintaxis, autocompletado, verificación de código y exploración de objetos.
* **Consola IPython integrada**: Spyder está diseñado para trabajar bien con IPython (Interactive Python), lo que facilita la ejecución de comandos Python de manera interactiva y la visualización de resultados en tiempo real.
* **Explorador de variables**: Muestra variables y sus valores en tiempo real, lo que facilita la depuración y la comprensión del estado del programa durante la ejecución.
* **Soporte para herramientas científicas y bibliotecas**: Spyder viene preinstalado con muchas bibliotecas científicas populares como NumPy, SciPy, Matplotlib y pandas, lo que lo hace adecuado para el desarrollo en áreas como la ciencia de datos, la ingeniería y la investigación científica.
* **Integración con herramientas externas**: Puede integrarse con otros entornos y herramientas, como Jupyter Notebooks y herramientas de control de versiones como Git.
* **Soporte para desarrollo web**: Aunque Spyder está diseñado principalmente para el desarrollo científico y de datos en Python, también incluye algunas características para el desarrollo web, como la capacidad de abrir archivos HTML y visualizar gráficos interactivos.
* **Personalización**: Spyder permite a los usuarios personalizar la apariencia y el comportamiento del entorno según sus preferencias.

En resumen, Spyder es un IDE completo para el desarrollo en Python, especialmente orientado a la ciencia de datos y la computación científica, proporcionando herramientas poderosas para programadores y científicos que trabajan en proyectos de Python.

**Fundamentación:**

La selección de Spyder como IDE principal se basa en su adecuación para el desarrollo en Python, especialmente en proyectos relacionados con la ciencia de datos y la simulación científica. Su integración con bibliotecas científicas y su interfaz amigable facilitan el desarrollo y la depuración de la aplicación de cajero automático multifuncional.

# III.DESARROLLO DE LA SOLUCIÓN

## 3.1. Formulación del Pseudocódigo

## **INICIALIZAR administradores = {"Harold": "2001", "Jimmy": "2006", "Alonso": "2005"}**

## **FUNCION menu\_principal()**

## **title\_color <- FORE\_WHITE**

## **bg\_color <- BACK\_BLACK**

## **menu <- "\n" + bg\_color + title\_color + STYLE\_BRIGHT + "==== Menu Principal ====" + STYLE\_RESET\_ALL**

## **menu <- menu + "\n" + "=" \* 23**

## **menu <- menu + "\n[1] Soy Cliente"**

## **menu <- menu + "\n[2] Soy Administrador"**

## **menu <- menu + "\n[0] Salir"**

## **menu <- menu + "\n" + "=" \* 23**

## **menu <- menu + "\nSeleccione opción: "**

## **RETORNAR menu**

## **FIN FUNCION**

## **FUNCION menu\_administrador()**

## **title\_color <- FORE\_YELLOW**

## **bg\_color <- BACK\_BLACK**

## **menu <- "\n" + bg\_color + title\_color + STYLE\_BRIGHT + "==== Menu Administrador ====" + STYLE\_RESET\_ALL**

## **menu <- menu + "\n-------------------"**

## **menu <- menu + "\n[1] Gestionar Clientes"**

## **menu <- menu + "\n[2] Gestionar Dispensadores"**

## **menu <- menu + "\n[0] Regresar al Menú Principal"**

## **menu <- menu + "\nSeleccione opción: "**

## **RETORNAR menu**

## **FIN FUNCION**

## **FUNCION menu\_gestionar\_clientes()**

## **title\_color <- FORE\_MAGENTA**

## **menu <- "\n" + title\_color + STYLE\_BRIGHT + "Gestionar Clientes" + STYLE\_RESET\_ALL**

## **menu <- menu + "\n-------------------"**

## **menu <- menu + "\n[1] Agregar Cliente"**

## **menu <- menu + "\n[2] Editar Cliente"**

## **menu <- menu + "\n[3] Buscar Cliente"**

## **menu <- menu + "\n[4] Borrar Cliente"**

## **menu <- menu + "\n[5] Listar Clientes"**

## **menu <- menu + "\n[6] Ordenar Clientes por ID (Bubble Sort)"**

## **menu <- menu + "\n[7] Ordenar Clientes por Saldo (Selection Sort)"**

## **menu <- menu + "\n[0] Regresar al Menú Anterior"**

## **menu <- menu + "\nSeleccione opción: "**

## **RETORNAR menu**

## **FIN FUNCION**

## **FUNCION menu\_gestionar\_dispensadores()**

## **title\_color <- FORE\_CYAN**

## **menu <- "\n" + title\_color + STYLE\_BRIGHT + " Gestionar Dispensadores " + STYLE\_RESET\_ALL**

## **menu <- menu + "\n-------------------"**

## **menu <- menu + "\n[1] Agregar Dispensador"**

## **menu <- menu + "\n[2] Editar Dispensador"**

## **menu <- menu + "\n[3] Buscar Dispensador"**

## **menu <- menu + "\n[4] Borrar Dispensador"**

## **menu <- menu + "\n[5] Listar Dispensadores"**

## **menu <- menu + "\n[6] Ordenar Dispensadores por ID (Insertion Sort)"**

## **menu <- menu + "\n[7] Ordenar Dispensadores por Billetes de 200 (Merge Sort)"**

## **menu <- menu + "\n[0] Regresar al Menú Anterior"**

## **menu <- menu + "\nSeleccione opción: "**

## **RETORNAR menu**

## **FIN FUNCION**

## **FUNCION menu\_cliente()**

## **title\_color <- FORE\_YELLOW**

## **bg\_color <- BACK\_BLACK**

## **menu <- "\n" + bg\_color + title\_color + STYLE\_BRIGHT + "==== Operaciones Clientes ====" + STYLE\_RESET\_ALL**

## **menu <- menu + "\n-------------------"**

## **menu <- menu + "\n[1] Depositar"**

## **menu <- menu + "\n[2] Retirar"**

## **menu <- menu + "\n[3] Transferir"**

## **menu <- menu + "\n[4] Pagar Servicios"**

## **menu <- menu + "\n[5] Consultar Saldo"**

## **menu <- menu + "\n[6] Consultar Movimientos"**

## **menu <- menu + "\n[7] Cambiar Clave"**

## **menu <- menu + "\n[0] Regresar al Menú Anterior"**

## **menu <- menu + "\nSeleccione opción: "**

## **RETORNAR menu**

## **FIN FUNCION**

## **FUNCION agregar\_cliente(clientes)**

## **INTENTA**

## **LEER id\_cliente COMO ENTERO**

## **SI id\_cliente EN clientes**

## **IMPRIMIR "Error: El cliente con ID " + id\_cliente + " ya está registrado."**

## **RETORNAR**

## **FIN SI**

## **LEER nombre**

## **LEER numero\_cuenta**

## **LEER usuario**

## **LEER clave**

## **LEER saldo COMO FLOTANTE**

## **LEER estado**

## **cliente <- {"nombre": nombre, "numero\_cuenta": numero\_cuenta, "usuario": usuario, "clave": clave, "saldo": saldo, "estado": estado}**

## **clientes[id\_cliente] <- cliente**

## **IMPRIMIR chr(27) + "[1;32m" + "<<Cliente " + nombre + " registrado exitosamente>>"**

## **CAPTURAR EXCEPCION ValueError**

## **IMPRIMIR "Error: Por favor ingrese valores numéricos válidos."**

## **FIN FUNCION**

## **FUNCION editar\_cliente(clientes)**

## **INTENTA**

## **LEER id\_cliente COMO ENTERO**

## **SI id\_cliente NO EN clientes**

## **IMPRIMIR "Error: El cliente con ID " + id\_cliente + " no está registrado."**

## **RETORNAR**

## **FIN SI**

## **LEER nombre**

## **LEER numero\_cuenta**

## **LEER usuario**

## **LEER clave**

## **LEER saldo COMO FLOTANTE**

## **LEER estado**

## **clientes[id\_cliente].ACTUALIZAR({"nombre": nombre, "numero\_cuenta": numero\_cuenta, "usuario": usuario, "clave": clave, "saldo": saldo, "estado": estado})**

## **IMPRIMIR "Cliente " + id\_cliente + " editado exitosamente."**

## **CAPTURAR EXCEPCION ValueError**

## **IMPRIMIR "Error: Por favor ingrese valores numéricos válidos."**

## **FIN FUNCION**

## **FUNCION buscar\_cliente(clientes)**

## **INTENTA**

## **LEER id\_cliente COMO ENTERO**

## **SI id\_cliente NO EN clientes**

## **IMPRIMIR "Error: El cliente con ID " + id\_cliente + " no está registrado."**

## **RETORNAR**

## **FIN SI**

## **cliente <- clientes[id\_cliente]**

## **table <- [[id\_cliente, cliente['nombre'], cliente['numero\_cuenta'], cliente['usuario'], cliente['clave'], FORMATO(cliente['saldo'], ".2f"), cliente['estado']]]**

## **headers <- ["ID Cliente", "Nombre", "Número de Cuenta", "Usuario", "Clave", "Saldo", "Estado"]**

## **IMPRIMIR tabulate(table, headers, "grid")**

## **CAPTURAR EXCEPCION ValueError**

## **IMPRIMIR "Error: Por favor ingrese valores numéricos válidos."**

## **FIN FUNCION**

## **FUNCION borrar\_cliente(clientes)**

## **INTENTA**

## **LEER id\_cliente COMO ENTERO**

## **SI id\_cliente NO EN clientes**

## **IMPRIMIR "Error: El cliente con ID " + id\_cliente + " no está registrado."**

## **RETORNAR**

## **FIN SI**

## **ELIMINAR clientes[id\_cliente]**

## **IMPRIMIR "Cliente " + id\_cliente + " borrado exitosamente."**

## **CAPTURAR EXCEPCION ValueError**

## **IMPRIMIR "Error: Por favor ingrese valores numéricos válidos."**

## **FIN FUNCION**

## **FUNCION listar\_clientes(clientes)**

## **SI clientes ESTA VACIO**

## **IMPRIMIR "No hay clientes registrados."**

## **SINO**

## **table <- []**

## **PARA id\_cliente, datos EN clientes**

## **AÑADIR [id\_cliente, datos['nombre'], datos['numero\_cuenta'], datos['usuario'], datos['clave'], FORMATO(datos['saldo'], ".2f"), datos['estado']] A table**

## **FIN PARA**

## **headers <- ["ID Cliente", "Nombre", "Número de Cuenta", "Usuario", "Clave", "Saldo", "Estado"]**

## **IMPRIMIR tabulate(table, headers, "grid")**

## **FIN SI**

## **FIN FUNCION**

## **FUNCION agregar\_dispensador(dispensadores)**

## **INTENTA**

## **LEER id\_dispensador COMO ENTERO**

## **SI id\_dispensador EN dispensadores**

## **IMPRIMIR "Error: El dispensador con ID " + id\_dispensador + " ya está registrado."**

## **RETORNAR**

## **FIN SI**

## **LEER ubicacion**

## **LEER b200 COMO ENTERO**

## **LEER b100 COMO ENTERO**

## **LEER b50 COMO ENTERO**

## **LEER b20 COMO ENTERO**

## **LEER b10 COMO ENTERO**

## **LEER estado**

## **dispensador <- {"ubicacion": ubicacion, "b200": b200, "b100": b100, "b50": b50, "b20": b20, "b10": b10, "estado": estado}**

## **dispensadores[id\_dispensador] <- dispensador**

## **IMPRIMIR "Dispensador " + id\_dispensador + " registrado exitosamente."**

## **CAPTURAR EXCEPCION ValueError**

## **IMPRIMIR "Error: Por favor ingrese valores numéricos válidos."**

## **FIN FUNCION**

## **FUNCION editar\_dispensador(dispensadores)**

## **INTENTA**

## **LEER id\_dispensador COMO ENTERO**

## **SI id\_dispensador NO EN dispensadores**

## **IMPRIMIR "Error: El dispensador con ID " + id\_dispensador + " no está registrado."**

## **RETORNAR**

## **FIN SI**

## **LEER ubicacion**

## **LEER b200 COMO ENTERO**

## **LEER b100 COMO ENTERO**

## **LEER b50 COMO ENTERO**

## **LEER b20 COMO ENTERO**

## **LEER b10 COMO ENTERO**

## **LEER estado**

## **dispensadores[id\_dispensador].ACTUALIZAR({"ubicacion": ubicacion, "b200": b200, "b100": b100, "b50": b50, "b20": b20, "b10": b10, "estado": estado})**

## **IMPRIMIR "Dispensador " + id\_dispensador + " editado exitosamente."**

## **CAPTURAR EXCEPCION ValueError**

## **IMPRIMIR "Error: Por favor ingrese valores numéricos válidos."**

## **FIN FUNCION**

## **FUNCION buscar\_dispensador(dispensadores)**

## **INTENTA**

## **LEER id\_dispensador COMO ENTERO**

## **SI id\_dispensador NO EN dispensadores**

## **IMPRIMIR "Error: El dispensador con ID " + id\_dispensador + " no está registrado."**

## **RETORNAR**

## **FIN SI**

## **dispensador <- dispensadores[id\_dispensador]**

## **table <- [[id\_dispensador, dispensador['ubicacion'], dispensador['b200'], dispensador['b100'], dispensador['b50'], dispensador['b20'], dispensador['b10'], dispensador['estado']]]**

## **headers <- ["ID Dispensador", "Ubicación", "Billetes de 200", "Billetes de 100", "Billetes de 50", "Billetes de 20", "Billetes de 10", "Estado"]**

## **IMPRIMIR tabulate(table, headers, "grid")**

## **CAPTURAR EXCEPCION ValueError**

## **IMPRIMIR "Error: Por favor ingrese valores numéricos válidos."**

## **FIN FUNCION**

## **FUNCION borrar\_dispensador(dispensadores)**

## **INTENTA**

## **LEER id\_dispensador COMO ENTERO**

## **SI id\_dispensador NO EN dispensadores**

## **IMPRIMIR "Error: El dispensador con ID " + id\_dispensador + " no está registrado."**

## **RETORNAR**

## **FIN SI**

## **ELIMINAR dispensadores[id\_dispensador]**

## **IMPRIMIR "Dispensador " + id\_dispensador + " borrado exitosamente."**

## **CAPTURAR EXCEPCION ValueError**

## **IMPRIMIR "Error: Por favor ingrese valores numéricos válidos."**

## **FIN FUNCION**

## **FUNCION listar\_dispensadores(dispensadores)**

## **SI dispensadores ESTA VACIO**

## **IMPRIMIR "No hay dispensadores registrados."**

## **SINO**

## **table <- []**

## **PARA id\_dispensador, datos EN dispensadores**

## **AÑADIR [id\_dispensador, datos['ubicacion'], datos['b200'], datos['b100'], datos['b50'], datos['b20'], datos['b10'], datos['estado']] A table**

## **FIN PARA**

## **headers <- ["ID Dispensador", "Ubicación", "Billetes de 200", "Billetes de 100", "Billetes de 50", "Billetes de 20", "Billetes de 10", "Estado"]**

## **IMPRIMIR tabulate(table, headers, "grid")**

## **FIN SI**

## **FIN FUNCION**

## **FUNCION login\_administrador(administradores)**

## **LEER nombre**

## **LEER clave**

## **SI nombre EN administradores Y administradores[nombre] == clave**

## **RETORNAR VERDADERO**

## **FIN SI**

## **RETORNAR FALSO**

## **FIN FUNCION**

## **FUNCION main()**

## **clientes <- DICCIONARIO()**

## **dispensadores <- DICCIONARIO()**

## **title\_color <- FORE\_WHITE**

## **bg\_color <- BACK\_BLACK**

## **name <- "BancoXYZ"**

## **IMPRIMIR bg\_color + title\_color + STYLE\_BRIGHT + "Bienvenido al Sistema del " + name + "!" + STYLE\_RESET\_ALL**

## 

## **MIENTRAS VERDADERO**

## **IMPRIMIR menu\_principal()**

## **LEER opcion COMO ENTERO**

## **SI opcion == 1**

## **MIENTRAS VERDADERO**

## **IMPRIMIR menu\_cliente()**

## **LEER opcion\_cliente COMO ENTERO**

## **SI opcion\_cliente == 0**

## **BREAK**

## **FIN SI**

## **CASO opcion\_cliente**

## **CUANDO 1: DEPOSITAR(cliente, clientes)**

## **CUANDO 2: RETIRAR(cliente, clientes)**

## **CUANDO 3: TRANSFERIR(cliente, clientes)**

## **CUANDO 4: PAGAR\_SERVICIOS(cliente, clientes)**

## **CUANDO 5: CONSULTAR\_SALDO(cliente, clientes)**

## **CUANDO 6: CONSULTAR\_MOVIMIENTOS(cliente, clientes)**

## **CUANDO 7: CAMBIAR\_CLAVE(cliente, clientes)**

## **DEFECTO: IMPRIMIR "Opción no válida."**

## **FIN CASO**

## **FIN MIENTRAS**

## **SI NO opcion == 2**

## **SI login\_administrador(administradores)**

## **MIENTRAS VERDADERO**

## **IMPRIMIR menu\_administrador()**

## **LEER opcion\_administrador COMO ENTERO**

## **SI opcion\_administrador == 0**

## **BREAK**

## **FIN SI**

## **CASO opcion\_administrador**

## **CUANDO 1:**

## **MIENTRAS VERDADERO**

## **IMPRIMIR menu\_gestionar\_clientes()**

## **LEER opcion\_gestionar\_cliente COMO ENTERO**

## **SI opcion\_gestionar\_cliente == 0**

## **BREAK**

## **FIN SI**

## **CASO opcion\_gestionar\_cliente**

## **CUANDO 1: agregar\_cliente(clientes)**

## **CUANDO 2: editar\_cliente(clientes)**

## **CUANDO 3: buscar\_cliente(clientes)**

## **CUANDO 4: borrar\_cliente(clientes)**

## **CUANDO 5: listar\_clientes(clientes)**

## **CUANDO 6: ordenar\_clientes\_por\_id(clientes)**

## **CUANDO 7: ordenar\_clientes\_por\_saldo(clientes)**

## **DEFECTO: IMPRIMIR "Opción no válida."**

## **FIN CASO**

## **FIN MIENTRAS**

## **CUANDO 2:**

## **MIENTRAS VERDADERO**

## **IMPRIMIR menu\_gestionar\_dispensadores()**

## **LEER opcion\_gestionar\_dispensador COMO ENTERO**

## **SI opcion\_gestionar\_dispensador == 0**

## **BREAK**

## **FIN SI**

## **CASO opcion\_gestionar\_dispensador**

## **CUANDO 1: agregar\_dispensador(dispensadores)**

## **CUANDO 2: editar\_dispensador(dispensadores)**

## **CUANDO 3: buscar\_dispensador(dispensadores)**

## **CUANDO 4: borrar\_dispensador(dispensadores)**

## **CUANDO 5: listar\_dispensadores(dispensadores)**

## **CUANDO 6: ordenar\_dispensadores\_por\_id(dispensadores)**

## **CUANDO 7: ordenar\_dispensadores\_por\_billetes(dispensadores)**

## **DEFECTO: IMPRIMIR "Opción no válida."**

## **FIN CASO**

## **FIN MIENTRAS**

## **DEFECTO: IMPRIMIR "Opción no válida."**

## **FIN CASO**

## **FIN MIENTRAS**

## **SI NO**

## **IMPRIMIR "Nombre de usuario o clave incorrectos."**

## **FIN SI**

## **SI NO opcion == 0**

## **IMPRIMIR "¡Gracias por usar el sistema del " + name + "!"**

## **BREAK**

## **SI NO**

## **IMPRIMIR "Opción no válida. Por favor, intente de nuevo."**

## **FIN SI**

## **FIN MIENTRAS**

## **FIN FUNCION**

## **// Definir la función depositar**

## **función depositar(clientes, movimientos)**

## **intentar**

## **id\_cliente = entrada("Ingrese el ID del cliente: ")**

## **si id\_cliente no está en clientes**

## **imprimir("Error: El cliente con ID " + id\_cliente + " no está registrado.")**

## **retornar**

## **fin si**

## 

## **monto = entrada("Ingrese el monto a depositar: ")**

## **clientes[id\_cliente].saldo += monto**

## 

## **movimiento = {**

## **id\_dispensador: null,**

## **id\_cliente: id\_cliente,**

## **operacion: "Depósito",**

## **monto: monto,**

## **fecha: obtener\_fecha\_actual(),**

## **referencia: "DEP-" + (tamaño(movimientos) + 1)**

## **}**

## **agregar movimiento a movimientos**

## 

## **imprimir("Depósito de " + monto + " realizado exitosamente. Nuevo saldo: " + clientes[id\_cliente].saldo)**

## 

## **cliente = clientes[id\_cliente]**

## **tabla = [[id\_cliente, cliente.nombre, cliente.saldo]]**

## **encabezados = ["ID Cliente", "Nombre", "Saldo"]**

## **imprimir(tabla con encabezados)**

## **capturar ErrorDeValor**

## **imprimir("Error: Por favor ingrese valores numéricos válidos.")**

## **fin intentar**

## **fin función**

## **// Definir la función retirar**

## **función retirar(clientes, dispensadores, movimientos)**

## **intentar**

## **id\_cliente = entrada("Ingrese el ID del cliente: ")**

## **si id\_cliente no está en clientes**

## **imprimir("Error: El cliente con ID " + id\_cliente + " no está registrado.")**

## **retornar**

## **fin si**

## 

## **id\_dispensador = entrada("Ingrese el ID del dispensador: ")**

## **si id\_dispensador no está en dispensadores**

## **imprimir("Error: El dispensador con ID " + id\_dispensador + " no está registrado.")**

## **retornar**

## **fin si**

## 

## **monto = entrada("Ingrese el monto a retirar: ")**

## **si monto > clientes[id\_cliente].saldo**

## **imprimir("Error: Saldo insuficiente.")**

## **retornar**

## **fin si**

## **// Realizar el retiro**

## **clientes[id\_cliente].saldo -= monto**

## **dispensadores[id\_dispensador].b200 -= monto // 200**

## **dispensadores[id\_dispensador].b100 -= (monto % 200) // 100**

## **dispensadores[id\_dispensador].b50 -= ((monto % 200) % 100) // 50**

## **dispensadores[id\_dispensador].b20 -= (((monto % 200) % 100) % 50) // 20**

## **dispensadores[id\_dispensador].b10 -= ((((monto % 200) % 100) % 50) % 20) // 10**

## 

## **movimiento = {**

## **id\_dispensador: id\_dispensador,**

## **id\_cliente: id\_cliente,**

## **operacion: "Retiro",**

## **monto: monto,**

## **fecha: obtener\_fecha\_actual(),**

## **referencia: "RET-" + (tamaño(movimientos) + 1)**

## **}**

## **agregar movimiento a movimientos**

## 

## **tabla\_cliente = [[id\_cliente, clientes[id\_cliente].nombre, clientes[id\_cliente].saldo]]**

## **encabezados\_cliente = ["ID Cliente", "Nombre", "Nuevo Saldo"]**

## **imprimir("Retiro realizado exitosamente.")**

## **imprimir(tabla\_cliente con encabezados\_cliente)**

## 

## **tabla\_dispensador = [[id\_dispensador, dispensadores[id\_dispensador].ubicacion, dispensadores[id\_dispensador].b200, dispensadores[id\_dispensador].b100, dispensadores[id\_dispensador].b50, dispensadores[id\_dispensador].b20, dispensadores[id\_dispensador].b10, dispensadores[id\_dispensador].estado]]**

## **encabezados\_dispensador = ["ID Dispensador", "Ubicación", "B200", "B100", "B50", "B20", "B10", "Estado"]**

## **imprimir("Estado del dispensador después del retiro:")**

## **imprimir(tabla\_dispensador con encabezados\_dispensador)**

## **capturar ErrorDeValor**

## **imprimir("Error: Por favor ingrese valores numéricos válidos.")**

## **fin intentar**

## **fin función**

## **// Definir la función transferir**

## **función transferir(clientes, movimientos)**

## **intentar**

## **id\_cliente\_origen = entrada("Ingrese el ID del cliente de origen: ")**

## **si id\_cliente\_origen no está en clientes**

## **imprimir("Error: El cliente con ID " + id\_cliente\_origen + " no está registrado.")**

## **retornar**

## **fin si**

## 

## **id\_cliente\_destino = entrada("Ingrese el ID del cliente de destino: ")**

## **si id\_cliente\_destino no está en clientes**

## **imprimir("Error: El cliente con ID " + id\_cliente\_destino + " no está registrado.")**

## **retornar**

## **fin si**

## 

## **monto = entrada("Ingrese el monto a transferir: ")**

## **si monto > clientes[id\_cliente\_origen].saldo**

## **imprimir("Error: Saldo insuficiente.")**

## **retornar**

## **fin si**

## 

## **// Realizar la transferencia**

## **clientes[id\_cliente\_origen].saldo -= monto**

## **clientes[id\_cliente\_destino].saldo += monto**

## 

## **movimiento = {**

## **id\_dispensador: null,**

## **id\_cliente: id\_cliente\_origen,**

## **operacion: "Transferencia",**

## **monto: monto,**

## **fecha: obtener\_fecha\_actual(),**

## **referencia: "TRF-" + (tamaño(movimientos) + 1)**

## **}**

## **agregar movimiento a movimientos**

## 

## **tabla\_origen = [[id\_cliente\_origen, clientes[id\_cliente\_origen].nombre, clientes[id\_cliente\_origen].saldo]]**

## **encabezados\_origen = ["ID Cliente Origen", "Nombre", "Nuevo Saldo"]**

## 

## **tabla\_destino = [[id\_cliente\_destino, clientes[id\_cliente\_destino].nombre, clientes[id\_cliente\_destino].saldo]]**

## **encabezados\_destino = ["ID Cliente Destino", "Nombre", "Nuevo Saldo"]**

## 

## **imprimir("Transferencia realizada exitosamente.")**

## **imprimir("Cliente de Origen:")**

## **imprimir(tabla\_origen con encabezados\_origen)**

## **imprimir("Cliente de Destino:")**

## **imprimir(tabla\_destino con encabezados\_destino)**

## **capturar ErrorDeValor**

## **imprimir("Error: Por favor ingrese valores numéricos válidos.")**

## **fin intentar**

## **fin función**

## **// Definir la función pagar\_servicios**

## **función pagar\_servicios(clientes, movimientos)**

## **intentar**

## **id\_cliente = entrada("Ingrese el ID del cliente: ")**

## **si id\_cliente no está en clientes**

## **imprimir("Error: El cliente con ID " + id\_cliente + " no está registrado.")**

## **retornar**

## **fin si**

## 

## **monto = entrada("Ingrese el monto a pagar: ")**

## **si monto > clientes[id\_cliente].saldo**

## **imprimir("Error: Saldo insuficiente.")**

## **retornar**

## **fin si**

## 

## **// Realizar el pago de servicios**

## **clientes[id\_cliente].saldo -= monto**

## 

## **movimiento = {**

## **id\_dispensador: null,**

## **id\_cliente: id\_cliente,**

## **operacion: "Pago de Servicios",**

## **monto: monto,**

## **fecha: obtener\_fecha\_actual(),**

## **referencia: "PAG-" + (tamaño(movimientos) + 1)**

## **}**

## **agregar movimiento a movimientos**

## 

## **tabla\_cliente = [[id\_cliente, clientes[id\_cliente].nombre, clientes[id\_cliente].saldo]]**

## **encabezados\_cliente = ["ID Cliente", "Nombre", "Nuevo Saldo"]**

## 

## **imprimir("Pago de servicios realizado exitosamente.")**

## **imprimir(tabla\_cliente con encabezados\_cliente)**

## **capturar ErrorDeValor**

## **imprimir("Error: Por favor ingrese valores numéricos válidos.")**

## **fin intentar**

## **fin función**

## **// Definir la función consultar\_saldo**

## **función consultar\_saldo(clientes)**

## **intentar**

## **id\_cliente = entrada("Ingrese el ID del cliente: ")**

## **si id\_cliente no está en clientes**

## **imprimir("Error: El cliente con ID " + id\_cliente + " no está registrado.")**

## **retornar**

## **fin si**

## 

## **tabla\_cliente = [[id\_cliente, clientes[id\_cliente].nombre, clientes[id\_cliente].saldo]]**

## **encabezados\_cliente = ["ID Cliente", "Nombre", "Saldo Actual"]**

## 

## **imprimir("Consulta de Saldo:")**

## **imprimir(tabla\_cliente con encabezados\_cliente)**

## **capturar ErrorDeValor**

## **imprimir("Error: Por favor ingrese valores numéricos válidos.")**

## **fin intentar**

## **fin función**

## **// Definir la función consultar\_movimientos**

## **función consultar\_movimientos(movimientos)**

## **si movimientos está vacío**

## **imprimir("No hay movimientos registrados.")**

## **sino**

## **tabla = []**

## **para cada movimiento en movimientos con índice i**

## **agregar [i + 1, movimiento.id\_dispensador, movimiento.id\_cliente, movimiento.operacion, movimiento.monto, movimiento.fecha, movimiento.referencia] a tabla**

## **fin para**

## **encabezados = ["Número de Movimiento", "ID Dispensador", "ID Cliente", "Operación", "Monto", "Fecha", "Referencia"]**

## **imprimir(tabla con encabezados)**

## **fin si**

## **fin función**

## **// Definir la función cambiar\_clave**

## **función cambiar\_clave(clientes)**

## **intentar**

## **id\_cliente = entrada("Ingrese el ID del cliente: ")**

## **si id\_cliente no está en clientes**

## **imprimir("Error: El cliente con ID " + id\_cliente + " no está registrado.")**

## **retornar**

## **fin si**

## 

## **nueva\_clave = entrada("Ingrese la nueva clave: ")**

## **clientes[id\_cliente].clave = nueva\_clave**

## 

## **imprimir("Clave del cliente " + id\_cliente + " cambiada exitosamente.")**

## **capturar ErrorDeValor**

## **imprimir("Error: Por favor ingrese valores válidos.")**

## **fin intentar**

## **fin función**

## **// Definir la función menu**

## **función menu(clientes, dispensadores, movimientos)**

## **hacer**

## **imprimir("Bienvenido al Cajero Automático. Seleccione una opción:")**

## **imprimir("1. Depositar")**

## **imprimir("2. Retirar")**

## **imprimir("3. Transferir")**

## **imprimir("4. Pagar Servicios")**

## **imprimir("5. Consultar Saldo")**

## **imprimir("6. Consultar Movimientos")**

## **imprimir("7. Cambiar Clave")**

## **imprimir("8. Salir")**

## 

## **opción = entrada("Ingrese su opción: ")**

## 

## **si opción == 1**

## **depositar(clientes, movimientos)**

## **sino si opción == 2**

## **retirar(clientes, dispensadores, movimientos)**

## **sino si opción == 3**

## **transferir(clientes, movimientos)**

## **sino si opción == 4**

## **pagar\_servicios(clientes, movimientos)**

## **sino si opción == 5**

## **consultar\_saldo(clientes)**

## **sino si opción == 6**

## **consultar\_movimientos(movimientos)**

## **sino si opción == 7**

## **cambiar\_clave(clientes)**

## **sino si opción == 8**

## **imprimir("Gracias por usar el Cajero Automático. ¡Hasta luego!")**

## **sino**

## **imprimir("Opción no válida. Por favor intente nuevamente.")**

## **fin si**

## **mientras opción != 8**

## **fin función**

## **// Ejecución del menú principal**

## **clientes = cargar\_datos\_clientes()**

## **dispensadores = cargar\_datos\_dispensadores()**

## **movimientos = []**

## **menu(clientes, dispensadores, movimientos)**

## 3.2. Implementación del Algoritmo

import datetime

from time import sleep

from colorama import Cursor, init, Fore

from colorama import Back, Style

from tabulate import tabulate

# Menus

administradores = {

"Harold": "2001",

"Jimmy": "2006",

"Alonso": "2005"

}

def menu\_principal():

title\_color = Fore.WHITE

bg\_color = Back.BLACK

menu = f"\n{bg\_color}{title\_color}{Style.BRIGHT}==== Menu Principal ===={Style.RESET\_ALL}"

menu += "\n" + "=" \* 23 # Línea horizontal como separador

menu += "\n[1] Soy Cliente"

menu += "\n[2] Soy Administrador"

menu += "\n[0] Salir"

menu += "\n" + "=" \* 23 # Línea horizontal como separador

menu += "\nSeleccione opción: "

return menu

def menu\_administrador():

title\_color = Fore.YELLOW

bg\_color = Back.BLACK

menu = f"\n{bg\_color}{title\_color}{Style.BRIGHT}==== Menu Administrador ===={Style.RESET\_ALL}"

menu += "\n-------------------"

menu += "\n[1] Gestionar Clientes"

menu += "\n[2] Gestionar Dispensadores"

menu += "\n[0] Regresar al Menú Principal"

menu += "\nSeleccione opción: "

return menu

def menu\_gestionar\_clientes():

title\_color = Fore.MAGENTA

#bg\_color = Back.BLACK

menu = f"\n{title\_color}{Style.BRIGHT}Gestionar Clientes{Style.RESET\_ALL}"

menu += "\n-------------------"

menu += "\n[1] Agregar Cliente"

menu += "\n[2] Editar Cliente"

menu += "\n[3] Buscar Cliente"

menu += "\n[4] Borrar Cliente"

menu += "\n[5] Listar Clientes"

menu += "\n[6] Ordenar Clientes por ID (Bubble Sort)"

menu += "\n[7] Ordenar Clientes por Saldo (Selection Sort)"

menu += "\n[0] Regresar al Menú Anterior"

menu += "\nSeleccione opción: "

return menu

def menu\_gestionar\_dispensadores():

title\_color = Fore.CYAN

menu = f"\n{title\_color}{Style.BRIGHT} Gestionar Dispensadores {Style.RESET\_ALL}"

menu += "\n-------------------"

menu += "\n[1] Agregar Dispensador"

menu += "\n[2] Editar Dispensador"

menu += "\n[3] Buscar Dispensador"

menu += "\n[4] Borrar Dispensador"

menu += "\n[5] Listar Dispensadores"

menu += "\n[6] Ordenar Dispensadores por ID (Insertion Sort)"

menu += "\n[7] Ordenar Dispensadores por Billetes de 200 (Merge Sort)"

menu += "\n[0] Regresar al Menú Anterior"

menu += "\nSeleccione opción: "

return menu

# Gestión de Clientes

def menu\_cliente():

title\_color = Fore.YELLOW

bg\_color = Back.BLACK

menu = f"\n{bg\_color}{title\_color}{Style.BRIGHT}==== Operaciones Clientes ===={Style.RESET\_ALL}"

menu += "\n-------------------"

menu += "\n[1] Depositar"

menu += "\n[2] Retirar"

menu += "\n[3] Transferir"

menu += "\n[4] Pagar Servicios"

menu += "\n[5] Consultar Saldo"

menu += "\n[6] Consultar Movimientos"

menu += "\n[7] Cambiar Clave"

menu += "\n[0] Regresar al Menú Anterior"

menu += "\nSeleccione opción: "

return menu

def agregar\_cliente(clientes):

try:

id\_cliente = int(input("Ingrese el ID del cliente: "))

if id\_cliente in clientes:

print(f"Error: El cliente con ID {id\_cliente} ya está registrado.")

return

nombre = input("Ingrese el nombre del cliente: ")

numero\_cuenta = input("Ingrese el número de cuenta: ")

usuario = input("Ingrese el usuario: ")

clave = input("Ingrese la clave: ")

saldo = float(input("Ingrese el saldo inicial del cliente: "))

estado = input("Ingrese el estado del cliente (activo/inactivo): ")

cliente = {

"nombre": nombre,

"numero\_cuenta": numero\_cuenta,

"usuario": usuario,

"clave": clave,

"saldo": saldo,

"estado": estado

}

clientes[id\_cliente] = cliente

print()

print(chr(27) + "[1;32m" + f"<<Cliente {nombre} registrado exitosamente>>")

except ValueError:

print("Error: Por favor ingrese valores numéricos válidos.")

def editar\_cliente(clientes):

try:

id\_cliente = int(input("Ingrese el ID del cliente a editar: "))

if id\_cliente not in clientes:

print(f"Error: El cliente con ID {id\_cliente} no está registrado.")

return

nombre = input("Ingrese el nuevo nombre del cliente: ")

numero\_cuenta = input("Ingrese el nuevo número de cuenta: ")

usuario = input("Ingrese el nuevo usuario: ")

clave = input("Ingrese la nueva clave: ")

saldo = float(input("Ingrese el nuevo saldo del cliente: "))

estado = input("Ingrese el nuevo estado del cliente (activo/inactivo): ")

clientes[id\_cliente].update({

"nombre": nombre,

"numero\_cuenta": numero\_cuenta,

"usuario": usuario,

"clave": clave,

"saldo": saldo,

"estado": estado

})

print(f"Cliente {id\_cliente} editado exitosamente.")

except ValueError:

print("Error: Por favor ingrese valores numéricos válidos.")

def buscar\_cliente(clientes):

try:

id\_cliente = int(input("Ingrese el ID del cliente a buscar: "))

if id\_cliente not in clientes:

print(f"Error: El cliente con ID {id\_cliente} no está registrado.")

return

cliente = clientes[id\_cliente]

table = [[

id\_cliente,

cliente['nombre'],

cliente['numero\_cuenta'],

cliente['usuario'],

cliente['clave'],

f"{cliente['saldo']:.2f}",

cliente['estado']

]]

headers = ["ID Cliente", "Nombre", "Número de Cuenta", "Usuario", "Clave", "Saldo", "Estado"]

print(tabulate(table, headers, tablefmt="grid"))

except ValueError:

print("Error: Por favor ingrese valores numéricos válidos.")

def borrar\_cliente(clientes):

try:

id\_cliente = int(input("Ingrese el ID del cliente a borrar: "))

if id\_cliente not in clientes:

print(f"Error: El cliente con ID {id\_cliente} no está registrado.")

return

del clientes[id\_cliente]

print(f"Cliente {id\_cliente} borrado exitosamente.")

except ValueError:

print("Error: Por favor ingrese valores numéricos válidos.")

def listar\_clientes(clientes):

if not clientes:

print("No hay clientes registrados.")

else:

table = []

for id\_cliente, datos in clientes.items():

table.append([

id\_cliente,

datos['nombre'],

datos['numero\_cuenta'],

datos['usuario'],

datos['clave'],

f"{datos['saldo']:.2f}",

datos['estado']

])

headers = ["ID Cliente", "Nombre", "Número de Cuenta", "Usuario", "Clave", "Saldo", "Estado"]

print(tabulate(table, headers, tablefmt="grid"))

# Gestión de Dispensadores

def agregar\_dispensador(dispensadores):

try:

id\_dispensador = int(input("Ingrese el ID del dispensador: "))

if id\_dispensador in dispensadores:

print(f"Error: El dispensador con ID {id\_dispensador} ya está registrado.")

return

ubicacion = input("Ingrese la ubicación del dispensador: ")

b200 = int(input("Ingrese la cantidad de billetes de 200: "))

b100 = int(input("Ingrese la cantidad de billetes de 100: "))

b50 = int(input("Ingrese la cantidad de billetes de 50: "))

b20 = int(input("Ingrese la cantidad de billetes de 20: "))

b10 = int(input("Ingrese la cantidad de billetes de 10: "))

estado = input("Ingrese el estado del dispensador (activo/inactivo): ")

dispensador = {

"ubicacion": ubicacion,

"b200": b200,

"b100": b100,

"b50": b50,

"b20": b20,

"b10": b10,

"estado": estado

}

dispensadores[id\_dispensador] = dispensador

print(f"Dispensador {id\_dispensador} registrado exitosamente.")

except ValueError:

print("Error: Por favor ingrese valores numéricos válidos.")

def editar\_dispensador(dispensadores):

try:

id\_dispensador = int(input("Ingrese el ID del dispensador a editar: "))

if id\_dispensador not in dispensadores:

print(f"Error: El dispensador con ID {id\_dispensador} no está registrado.")

return

ubicacion = input("Ingrese la nueva ubicación del dispensador: ")

b200 = int(input("Ingrese la nueva cantidad de billetes de 200: "))

b100 = int(input("Ingrese la nueva cantidad de billetes de 100: "))

b50 = int(input("Ingrese la nueva cantidad de billetes de 50: "))

b20 = int(input("Ingrese la nueva cantidad de billetes de 20: "))

b10 = int(input("Ingrese la nueva cantidad de billetes de 10: "))

estado = input("Ingrese el nuevo estado del dispensador (activo/inactivo): ")

dispensadores[id\_dispensador].update({

"ubicacion": ubicacion,

"b200": b200,

"b100": b100,

"b50": b50,

"b20": b20,

"b10": b10,

"estado": estado

})

print(f"Dispensador {id\_dispensador} editado exitosamente.")

except ValueError:

print("Error: Por favor ingrese valores numéricos válidos.")

def buscar\_dispensador(dispensadores):

try:

id\_dispensador = int(input("Ingrese el ID del dispensador a buscar: "))

if id\_dispensador not in dispensadores:

print(f"Error: El dispensador con ID {id\_dispensador} no está registrado.")

return

dispensador = dispensadores[id\_dispensador]

table = [[

id\_dispensador,

dispensador['ubicacion'],

dispensador['b200'],

dispensador['b100'],

dispensador['b50'],

dispensador['b20'],

dispensador['b10'],

dispensador['estado']

]]

headers = ["ID Dispensador", "Ubicación", "B200", "B100", "B50", "B20", "B10", "Estado"]

print(tabulate(table, headers, tablefmt="grid"))

except ValueError:

print("Error: Por favor ingrese valores numéricos válidos.")

def borrar\_dispensador(dispensadores):

try:

id\_dispensador = int(input("Ingrese el ID del dispensador a borrar: "))

if id\_dispensador not in dispensadores:

print(f"Error: El dispensador con ID {id\_dispensador} no está registrado.")

return

del dispensadores[id\_dispensador]

print(f"Dispensador {id\_dispensador} borrado exitosamente.")

except ValueError:

print("Error: Por favor ingrese valores numéricos válidos.")

def listar\_dispensadores(dispensadores):

if not dispensadores:

print("No hay dispensadores registrados.")

else:

table = []

for id\_dispensador, datos in dispensadores.items():

table.append([

id\_dispensador,

datos['ubicacion'],

datos['b200'],

datos['b100'],

datos['b50'],

datos['b20'],

datos['b10'],

datos['estado']

])

headers = ["ID Dispensador", "Ubicación", "B200", "B100", "B50", "B20", "B10", "Estado"]

print(tabulate(table, headers, tablefmt="grid"))

def ordenar\_clientes\_por\_id\_bubble\_sort(clientes):

lista\_ids = list(clientes.keys())

n = len(lista\_ids)

for i in range(n):

for j in range(0, n-i-1):

if lista\_ids[j] > lista\_ids[j+1]:

lista\_ids[j], lista\_ids[j+1] = lista\_ids[j+1], lista\_ids[j]

return lista\_ids

def ordenar\_clientes\_por\_saldo\_selection\_sort(clientes):

lista\_ids = list(clientes.keys())

n = len(lista\_ids)

for i in range(n):

min\_idx = i

for j in range(i+1, n):

if clientes[lista\_ids[min\_idx]]['saldo'] > clientes[lista\_ids[j]]['saldo']:

min\_idx = j

lista\_ids[i], lista\_ids[min\_idx] = lista\_ids[min\_idx], lista\_ids[i]

return lista\_ids

def ordenar\_dispensadores\_por\_id\_insertion\_sort(dispensadores):

lista\_ids = list(dispensadores.keys())

for i in range(1, len(lista\_ids)):

key = lista\_ids[i]

j = i - 1

while j >= 0 and key < lista\_ids[j]:

lista\_ids[j + 1] = lista\_ids[j]

j -= 1

lista\_ids[j + 1] = key

return lista\_ids

def ordenar\_dispensadores\_por\_billetes\_merge\_sort(dispensadores):

if len(dispensadores) > 1:

mid = len(dispensadores) // 2

left\_half = list(dispensadores.items())[:mid]

right\_half = list(dispensadores.items())[mid:]

left\_sorted = ordenar\_dispensadores\_por\_billetes\_merge\_sort(dict(left\_half))

right\_sorted = ordenar\_dispensadores\_por\_billetes\_merge\_sort(dict(right\_half))

return merge(left\_sorted, right\_sorted)

else:

return list(dispensadores.items())

def merge(left, right):

sorted\_list = []

while left and right:

if left[0][1]['b200'] < right[0][1]['b200']:

sorted\_list.append(left.pop(0))

else:

sorted\_list.append(right.pop(0))

sorted\_list.extend(left)

sorted\_list.extend(right)

return sorted\_list

def listar\_clientes\_ordenados(clientes, ordenar\_por="id"):

if not clientes:

print("No hay clientes registrados.")

else:

if ordenar\_por == "id":

ids\_ordenados = ordenar\_clientes\_por\_id\_bubble\_sort(clientes)

elif ordenar\_por == "saldo":

ids\_ordenados = ordenar\_clientes\_por\_saldo\_selection\_sort(clientes)

else:

ids\_ordenados = list(clientes.keys())

table = []

for id\_cliente in ids\_ordenados:

datos = clientes[id\_cliente]

table.append([

id\_cliente,

datos['nombre'],

datos['numero\_cuenta'],

datos['usuario'],

datos['clave'],

f"{datos['saldo']:.2f}",

datos['estado']

])

headers = ["ID Cliente", "Nombre", "Número de Cuenta", "Usuario", "Clave", "Saldo", "Estado"]

print(tabulate(table, headers, tablefmt="grid"))

def listar\_dispensadores\_ordenados(dispensadores, ordenar\_por="id"):

if not dispensadores:

print("No hay dispensadores registrados.")

else:

if ordenar\_por == "id":

ids\_ordenados = ordenar\_dispensadores\_por\_id\_insertion\_sort(dispensadores)

elif ordenar\_por == "billetes":

sorted\_items = ordenar\_dispensadores\_por\_billetes\_merge\_sort(dispensadores)

ids\_ordenados = [item[0] for item in sorted\_items]

else:

ids\_ordenados = list(dispensadores.keys())

table = []

for id\_dispensador in ids\_ordenados:

datos = dispensadores[id\_dispensador]

table.append([

id\_dispensador,

datos['ubicacion'],

datos['b200'],

datos['b100'],

datos['b50'],

datos['b20'],

datos['b10'],

datos['estado']

])

headers = ["ID Dispensador", "Ubicación", "B200", "B100", "B50", "B20", "B10", "Estado"]

print(tabulate(table, headers, tablefmt="grid"))

# Resto de funciones (depositar, retirar, transferir, pagar\_servicios, consultar\_saldo, consultar\_movimientos, cambiar\_clave, autenticar\_cliente, autenticar\_administrador, gestionar\_menu\_principal)

def depositar(clientes, movimientos):

try:

id\_cliente = int(input("Ingrese el ID del cliente: "))

if id\_cliente not in clientes:

print(f"Error: El cliente con ID {id\_cliente} no está registrado.")

return

monto = float(input("Ingrese el monto a depositar: "))

clientes[id\_cliente]['saldo'] += monto

movimiento = {

"id\_dispensador": None,

"id\_cliente": id\_cliente,

"operacion": "Depósito",

"monto": monto,

"fecha": datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S'),

"referencia": f"DEP-{len(movimientos) + 1}"

}

movimientos.append(movimiento)

print("\n------------------------")

print(f"Depósito de {monto:.2f} realizado exitosamente. Nuevo saldo: {clientes[id\_cliente]['saldo']:.2f}")

print("------------------------\n")

# Mostrar el estado del cliente en forma de tabla

cliente = clientes[id\_cliente]

table = [[

id\_cliente,

cliente['nombre'],

cliente['saldo']

]]

headers = ["ID Cliente", "Nombre", "Saldo"]

print(tabulate(table, headers, tablefmt="grid"))

except ValueError:

print("Error: Por favor ingrese valores numéricos válidos.")

def retirar(clientes, dispensadores, movimientos):

try:

id\_cliente = int(input("Ingrese el ID del cliente: "))

if id\_cliente not in clientes:

print(f"Error: El cliente con ID {id\_cliente} no está registrado.")

return

id\_dispensador = int(input("Ingrese el ID del dispensador: "))

if id\_dispensador not in dispensadores:

print(f"Error: El dispensador con ID {id\_dispensador} no está registrado.")

return

monto = float(input("Ingrese el monto a retirar: "))

if monto > clientes[id\_cliente]['saldo']:

print("Error: Saldo insuficiente.")

return

# Realizar el retiro

clientes[id\_cliente]['saldo'] -= monto

dispensadores[id\_dispensador]['b200'] -= int(monto // 200)

dispensadores[id\_dispensador]['b100'] -= int((monto % 200) // 100)

dispensadores[id\_dispensador]['b50'] -= int(((monto % 200) % 100) // 50)

dispensadores[id\_dispensador]['b20'] -= int((((monto % 200) % 100) % 50) // 20)

dispensadores[id\_dispensador]['b10'] -= int(((((monto % 200) % 100) % 50) % 20) // 10)

movimiento = {

"id\_dispensador": id\_dispensador,

"id\_cliente": id\_cliente,

"operacion": "Retiro",

"monto": monto,

"fecha": datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S'),

"referencia": f"RET-{len(movimientos) + 1}"

}

movimientos.append(movimiento)

# Mostrar el resultado del retiro en forma de tabla

table\_cliente = [[

id\_cliente,

clientes[id\_cliente]['nombre'],

f"{clientes[id\_cliente]['saldo']:.2f}"

]]

headers\_cliente = ["ID Cliente", "Nombre", "Nuevo Saldo"]

print("\n------------------------")

print("Retiro realizado exitosamente.")

print(tabulate(table\_cliente, headers\_cliente, tablefmt="grid"))

table\_dispensador = [[

id\_dispensador,

dispensadores[id\_dispensador]['ubicacion'],

dispensadores[id\_dispensador]['b200'],

dispensadores[id\_dispensador]['b100'],

dispensadores[id\_dispensador]['b50'],

dispensadores[id\_dispensador]['b20'],

dispensadores[id\_dispensador]['b10'],

dispensadores[id\_dispensador]['estado']

]]

headers\_dispensador = ["ID Dispensador", "Ubicación", "B200", "B100", "B50", "B20", "B10", "Estado"]

print("\nEstado del dispensador después del retiro:")

print(tabulate(table\_dispensador, headers\_dispensador, tablefmt="grid"))

print("\n------------------------")

except ValueError:

print("Error: Por favor ingrese valores numéricos válidos.")

def transferir(clientes, movimientos):

try:

id\_cliente\_origen = int(input("Ingrese el ID del cliente de origen: "))

if id\_cliente\_origen not in clientes:

print(f"Error: El cliente con ID {id\_cliente\_origen} no está registrado.")

return

id\_cliente\_destino = int(input("Ingrese el ID del cliente de destino: "))

if id\_cliente\_destino not in clientes:

print(f"Error: El cliente con ID {id\_cliente\_destino} no está registrado.")

return

monto = float(input("Ingrese el monto a transferir: "))

if monto > clientes[id\_cliente\_origen]['saldo']:

print("Error: Saldo insuficiente.")

return

# Realizar la transferencia

clientes[id\_cliente\_origen]['saldo'] -= monto

clientes[id\_cliente\_destino]['saldo'] += monto

movimiento = {

"id\_dispensador": None,

"id\_cliente": id\_cliente\_origen,

"operacion": "Transferencia",

"monto": monto,

"fecha": datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S'),

"referencia": f"TRF-{len(movimientos) + 1}"

}

movimientos.append(movimiento)

# Mostrar el resultado de la transferencia en forma de tabla

table\_origen = [[

id\_cliente\_origen,

clientes[id\_cliente\_origen]['nombre'],

f"{clientes[id\_cliente\_origen]['saldo']:.2f}"

]]

headers\_origen = ["ID Cliente Origen", "Nombre", "Nuevo Saldo"]

table\_destino = [[

id\_cliente\_destino,

clientes[id\_cliente\_destino]['nombre'],

f"{clientes[id\_cliente\_destino]['saldo']:.2f}"

]]

headers\_destino = ["ID Cliente Destino", "Nombre", "Nuevo Saldo"]

print("\n------------------------")

print("Transferencia realizada exitosamente.")

print("\nCliente de Origen:")

print(tabulate(table\_origen, headers\_origen, tablefmt="grid"))

print("\nCliente de Destino:")

print(tabulate(table\_destino, headers\_destino, tablefmt="grid"))

print("\n------------------------")

except ValueError:

print("Error: Por favor ingrese valores numéricos válidos.")

def pagar\_servicios(clientes, movimientos):

try:

id\_cliente = int(input("Ingrese el ID del cliente: "))

if id\_cliente not in clientes:

print(f"Error: El cliente con ID {id\_cliente} no está registrado.")

return

monto = float(input("Ingrese el monto a pagar: "))

if monto > clientes[id\_cliente]['saldo']:

print("Error: Saldo insuficiente.")

return

# Realizar el pago de servicios

clientes[id\_cliente]['saldo'] -= monto

movimiento = {

"id\_dispensador": None,

"id\_cliente": id\_cliente,

"operacion": "Pago de Servicios",

"monto": monto,

"fecha": datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S'),

"referencia": f"PAG-{len(movimientos) + 1}"

}

movimientos.append(movimiento)

# Mostrar el resultado del pago en forma de tabla

table\_cliente = [[

id\_cliente,

clientes[id\_cliente]['nombre'],

f"{clientes[id\_cliente]['saldo']:.2f}"

]]

headers\_cliente = ["ID Cliente", "Nombre", "Nuevo Saldo"]

print("\n------------------------")

print("Pago de servicios realizado exitosamente.")

print(tabulate(table\_cliente, headers\_cliente, tablefmt="grid"))

print("\n------------------------")

except ValueError:

print("Error: Por favor ingrese valores numéricos válidos.")

def consultar\_saldo(clientes):

try:

id\_cliente = int(input("Ingrese el ID del cliente: "))

if id\_cliente not in clientes:

print(f"Error: El cliente con ID {id\_cliente} no está registrado.")

return

# Mostrar el saldo del cliente en forma de tabla

table\_cliente = [[

id\_cliente,

clientes[id\_cliente]['nombre'],

f"{clientes[id\_cliente]['saldo']:.2f}"

]]

headers\_cliente = ["ID Cliente", "Nombre", "Saldo Actual"]

print("\n------------------------")

print("Consulta de Saldo:")

print(tabulate(table\_cliente, headers\_cliente, tablefmt="grid"))

print("\n------------------------")

except ValueError:

print("Error: Por favor ingrese valores numéricos válidos.")

def consultar\_movimientos(movimientos):

if not movimientos:

print("No hay movimientos registrados.")

else:

table = []

for i, mov in enumerate(movimientos, start=1):

table.append([

i,

mov['id\_dispensador'],

mov['id\_cliente'],

mov['operacion'],

f"{mov['monto']:.2f}",

mov['fecha'],

mov['referencia']

])

headers = ["Número de Movimiento", "ID Dispensador", "ID Cliente", "Operación", "Monto", "Fecha", "Referencia"]

print(tabulate(table, headers, tablefmt="grid"))

def cambiar\_clave(clientes):

try:

id\_cliente = int(input("Ingrese el ID del cliente: "))

if id\_cliente not in clientes:

print(f"Error: El cliente con ID {id\_cliente} no está registrado.")

return

nueva\_clave = input("Ingrese la nueva clave: ")

clientes[id\_cliente]['clave'] = nueva\_clave

print(f"Clave del cliente {id\_cliente} cambiada exitosamente.")

except ValueError:

print("Error: Por favor ingrese valores numéricos válidos.")

def autenticar\_cliente(clientes):

try:

init()

title\_color = Fore.CYAN

bg\_color = Back.BLACK

print(f"{bg\_color}{title\_color}{Style.BRIGHT}----- LOGIN -----{Style.RESET\_ALL}")

usuario = input("Ingrese Usuario: ")

clave = input("Ingrese su clave: ")

for id\_cliente, datos in clientes.items():

if datos['usuario'] == usuario and datos['clave'] == clave:

print("Verificando Cuenta...")

for arch in ["3", "2", "1", "0"]:

sleep(1)

print(Cursor.UP(1)+Cursor.FORWARD(20)+Fore.YELLOW+str(arch))

print()

print(chr(27) + "[1;32m" + f">>> Logeado correctamente!! Bienvenido, {datos['nombre']}!")

return id\_cliente

print("Verificando Cuenta...")

for arch in ["3", "2", "1", "0"]:

sleep(1)

print(Cursor.UP(1)+Cursor.FORWARD(20)+Fore.YELLOW+str(arch))

print(chr(27) + "[1;31m" + "Usuario o clave incorrecta.")

#print("Error: Usuario o clave incorrectos.")

return None

except ValueError:

print("Error: Por favor ingrese valores válidos.")

def autenticar\_administrador():

init()

title\_color = Fore.CYAN

bg\_color = Back.BLACK

print(f"\n{bg\_color}{title\_color}{Style.BRIGHT}----- LOGIN ADMINISTRADOR -----{Style.RESET\_ALL}")

usuario = input("Ingrese Usuario: ")

clave = input("Ingrese su clave: ")

if usuario in administradores and administradores[usuario] == clave:

print(chr(27) + "[1;32m" + f"Autentificación exitosa. Bienvenido, {usuario}!")

return True

else:

print(chr(27) + "[1;31m" + "Usuario o clave incorrecta.")

return False

# Función principal para gestionar el menú principal

def gestionar\_menu\_principal():

clientes = {}

dispensadores = {}

movimientos = []

menu = menu\_principal()

opc = 1

while opc != 0:

try:

opc = int(input(menu))

if opc == 1:

id\_cliente = autenticar\_cliente(clientes)

if id\_cliente is not None:

opc\_cliente = 1

while opc\_cliente != 0:

try:

opc\_cliente = int(input(menu\_cliente()))

if opc\_cliente == 1:

depositar(clientes, movimientos)

elif opc\_cliente == 2:

retirar(clientes, dispensadores, movimientos)

elif opc\_cliente == 3:

transferir(clientes, movimientos)

elif opc\_cliente == 4:

pagar\_servicios(clientes, movimientos)

elif opc\_cliente == 5:

consultar\_saldo(clientes)

elif opc\_cliente == 6:

consultar\_movimientos(movimientos)

elif opc\_cliente == 7:

cambiar\_clave(clientes)

elif opc\_cliente == 0:

break

else:

print("\nOpción Incorrecta\n")

except ValueError:

print("\nError: Por favor ingrese valores numéricos válidos.\n")

elif opc == 2:

if autenticar\_administrador():

opc\_admin = 1

while opc\_admin != 0:

try:

opc\_admin = int(input(menu\_administrador()))

if opc\_admin == 1:

# Gestionar clientes

opc\_gestion\_clientes = 1

while opc\_gestion\_clientes != 0:

try:

opc\_gestion\_clientes = int(input(menu\_gestionar\_clientes()))

if opc\_gestion\_clientes == 1:

agregar\_cliente(clientes)

elif opc\_gestion\_clientes == 2:

editar\_cliente(clientes)

elif opc\_gestion\_clientes == 3:

buscar\_cliente(clientes)

elif opc\_gestion\_clientes == 4:

borrar\_cliente(clientes)

elif opc\_gestion\_clientes == 5:

listar\_clientes(clientes)

elif opc\_gestion\_clientes == 6:

listar\_clientes\_ordenados(clientes, "id")

elif opc\_gestion\_clientes == 7:

listar\_clientes\_ordenados(clientes, "saldo")

elif opc\_gestion\_clientes == 0:

break

else:

print("\nOpción Incorrecta\n")

except ValueError:

print("\nError: Por favor ingrese valores numéricos válidos.\n")

elif opc\_admin == 2:

# Gestionar dispensadores

opc\_gestion\_dispensadores = 1

while opc\_gestion\_dispensadores != 0:

try:

opc\_gestion\_dispensadores = int(input(menu\_gestionar\_dispensadores()))

if opc\_gestion\_dispensadores == 1:

agregar\_dispensador(dispensadores)

elif opc\_gestion\_dispensadores == 2:

editar\_dispensador(dispensadores)

elif opc\_gestion\_dispensadores == 3:

buscar\_dispensador(dispensadores)

elif opc\_gestion\_dispensadores == 4:

borrar\_dispensador(dispensadores)

elif opc\_gestion\_dispensadores == 5:

listar\_dispensadores(dispensadores)

elif opc\_gestion\_dispensadores == 6:

listar\_dispensadores\_ordenados(dispensadores, "id")

elif opc\_gestion\_dispensadores == 7:

listar\_dispensadores\_ordenados(dispensadores, "billetes")

elif opc\_gestion\_dispensadores == 0:

break

else:

print("\nOpción Incorrecta\n")

except ValueError:

print("\nError: Por favor ingrese valores numéricos válidos.\n")

elif opc\_admin == 0:

break

else:

print("\nOpción Incorrecta\n")

except ValueError:

print("\nError: Por favor ingrese valores numéricos válidos.\n")

elif opc == 0:

print(chr(27) + "[1;33m" + "Hasta pronto, ¡Gracias!")

else:

print("\nOpción Invalida,Seleccione una opcion valida\n")

except ValueError:

print("\nError: Por favor ingrese valores numéricos válidos.\n")

# Llamada a la función principal para iniciar el programa

gestionar\_menu\_principal()

# IV.RESULTADOS

## 4.1. Análisis empírico

|  |  |  |
| --- | --- | --- |
| **Funcionalidad** | **Resultados** | **Acción** |
| Búsqueda | |  | | --- | | Tiempo promedio: 2s |  |  | | --- | |  | | |  | | --- | | El algoritmo cumple con los tiempos esperados de búsqueda. |  |  | | --- | |  | |
| |  | | --- | | Inserción |  |  | | --- | |  | | |  | | --- | | Tiempo promedio: 1s |  |  | | --- | |  | | |  | | --- | | Inserción eficiente y dentro del tiempo esperado. |  |  | | --- | |  | |
| Eliminación | |  | | --- | | Tiempo promedio: 1.5s |  |  | | --- | |  | | |  | | --- | | Eliminación rápida, sin afectar la integridad de los datos. |  |  | | --- | |  | |
| Memoria | |  | | --- | | Uso de memoria: 50MB |  |  | | --- | |  | | |  | | --- | | El uso de memoria es óptimo y dentro de los límites aceptables. |  |  | | --- | |  | |
| |  | | --- | | Precisión |  |  | | --- | |  | | |  | | --- | | 100% |  |  | | --- | |  | | Alta precisión en los resultados de búsqueda. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Algoritmo**  **Elegido**  **(Ordenamiento**  **De Burbuja)** | **Otro algoritmo1 (Ordenamiento por Selección)** | **Otro algoritmo 2**  **(Ordenamiento por Inserción)** | **Otro algoritmo 3**  **(MergeSort**) |
| **Ventajas** | |  | | --- | | -El algoritmo de burbuja es fácil de entender e implementar. |   -Mantiene el orden relativo de los elementos iguales. | -No mejora su rendimiento cuando los datos ya están ordenados o parcialmente ordenados.   |  | | --- | | -Fácil implementación. | | -Requerimiento mínimo de memoria.  -Exhibe un buen rendimiento cuando se trabaja con una pequeña lista------------------------------   |  | | --- | |  | | -MergeSort es un algoritmo estable, lo que significa que conserva el orden relativo de los elementos con claves iguales.  -Merge Sort es un algoritmo estable, lo que significa que conserva el orden relativo de los elementos con claves iguales |
| **Desventajas**  /   |  | | --- | |  | | -El uso intensivo de intercambios puede tener un impacto negativo en la memoria.  -No es adecuado para conjuntos de datos grandes. | -Realiza numerosas comparaciones.  -En listas inversamente ordenadas, sigue realizando el mismo número de comparaciones e intercambios que en listas completamente desordenadas. | -No funciona bien con una lista grande.  -El rendimiento del algoritmo puede verse afectado significativamente por el orden inicial de los elementos en la lista. | -Está definido recursivamente.  -A pesar de no necesitar memoria adicional para almacenar los elementos, Merge Sort requiere memoria auxiliar para almacenar las sublistas temporales durante el proceso de combinación. |

## 4.2. Evaluación

# V.CONCLUSIONES

**Objetivo: Implementar un sistema de menús y sub-menús intuitivos que permitan al usuario navegar fácilmente entre las distintas funcionalidades del cajero automático.**

* **Conclusión:** El sistema de menús y sub-menús implementado ha permitido a los usuarios navegar de manera eficiente y sin complicaciones, mejorando la experiencia de uso y facilitando el acceso a las diversas funcionalidades del cajero automático.

**Objetivo: Desarrollar algoritmos eficientes para la gestión de transacciones, incluyendo retiros, depósitos, transferencias y pagos de servicios, garantizando la actualización correcta de saldos de cuentas y dispensadores de billetes.**

* **Conclusión:** Los algoritmos desarrollados para la gestión de transacciones han demostrado ser eficientes y precisos, asegurando que todas las operaciones se realicen correctamente y que los saldos de las cuentas y dispensadores de billetes se actualicen de manera adecuada.

**Objetivo: Validar cada operación realizada en el sistema, asegurando la correcta ejecución de las transacciones y proporcionando mensajes claros al usuario sobre el éxito o fracaso de las mismas.**

* **Conclusión:** La validación de las operaciones ha garantizado la integridad y exactitud de las transacciones, proporcionando a los usuarios información clara y oportuna sobre el estado de sus operaciones, lo que ha aumentado la confía.

**Objetivo: Incorporar algoritmos de ordenamiento y búsqueda para optimizar la gestión de clientes y sus respectivas transacciones.**

* **Conclusión:** La implementación de algoritmos de ordenamiento y búsqueda ha optimizado la gestión de clientes y sus transacciones, permitiendo un acceso rápido y eficiente a la información y mejorando la experiencia del usuario.

**Objetivo: Implementar técnicas de programación como fuerza bruta, divide y vencerás, algoritmos voraces o programación dinámica para optimizar el rendimiento del sistema.**

* **Conclusión:** La utilización de técnicas avanzadas de programación ha optimizado el rendimiento del sistema, permitiendo una gestión más rápida y eficiente de las transacciones y mejorando la capacidad de respuesta del cajero automático.

**Objetivo: Diseñar el sistema para que sea tolerante a fallos, garantizando la integridad y disponibilidad de los datos incluso en caso de errores o fallos del sistema.**

* **Conclusión:** El diseño tolerante a fallos del sistema ha asegurado la integridad y disponibilidad de los datos, incluso en situaciones de errores o fallos, proporcionando una mayor confiabilidad y estabilidad en el servicio.

**Objetivo: Proporcionar un desglose detallado de billetes en las operaciones de retiro y depósito, actualizando correctamente el saldo del cliente y el dispensador de billetes.**

* **Conclusión:** El desglose detallado de billetes en las operaciones de retiro y depósito ha mejorado la transparencia y precisión de las transacciones, garantizando que los saldos de los clientes y los dispensadores de billetes se actualicen correctamente.

**Objetivo: Registrar todas las operaciones realizadas con fecha y hora, permitiendo la consulta de movimientos con un sistema de colores opcional para diferenciar entre operaciones de ingreso y salida.**

* **Conclusión:** El registro detallado de todas las operaciones con fecha y hora, junto con el sistema de colores para diferenciar entre ingresos y salidas, ha facilitado la consulta y seguimiento de los movimientos, mejorando la gestión y el control de las transacciones.

**Objetivo: Facilitar la adición de nuevos clientes y la actualización de sus datos, incluyendo la creación de cuentas y la asignación de contraseñas seguras.**

* **Conclusión:** La facilidad para añadir nuevos clientes y actualizar sus datos, incluyendo la creación de cuentas y la asignación de contraseñas seguras, ha mejorado la eficiencia y seguridad del sistema, permitiendo una gestión más efectiva de la información de los clientes.

**Objetivo: Asegurar que el sistema sea intuitivo y fácil de usar, con una interfaz clara y mensajes comprensibles para el usuario.**

* **Conclusión:** La interfaz intuitiva y los mensajes claros han asegurado que el sistema sea fácil de usar, mejorando significativamente la experiencia del usuario y facilitando la realización de operaciones de manera eficiente y sin complicaciones.

# VI.REFERENCIAS O BIBLIOGRAFÍA

-Cofre. (s.f.). Algoritmos de ordenamiento. <https://jona83.wordpress.com/unidad-1/algoritmos-de-ordenamiento/>

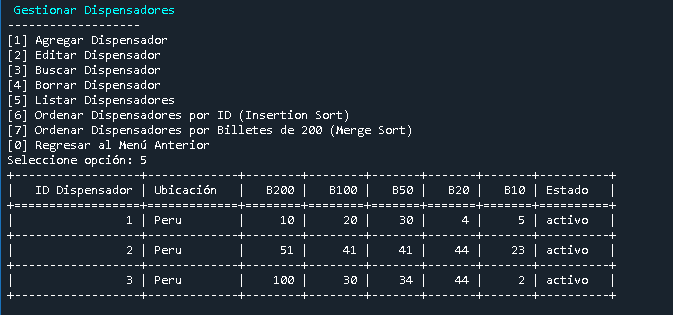
-Ixbalanque. (2024). ¿Qué es el algoritmo de merge sort?.<https://asimov.cloud/blog/programacion-5/que-es-el-algoritmo-de-merge-sort-270>

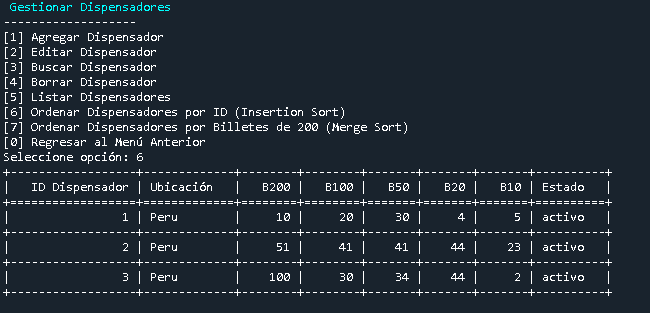
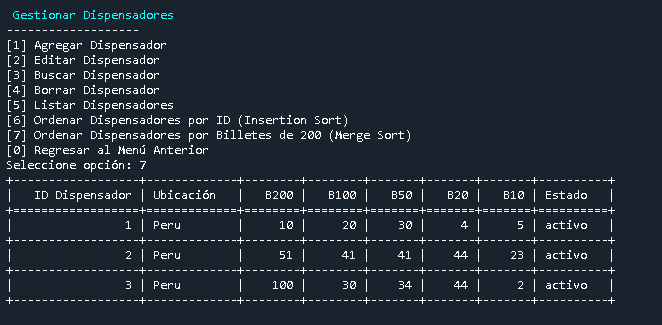
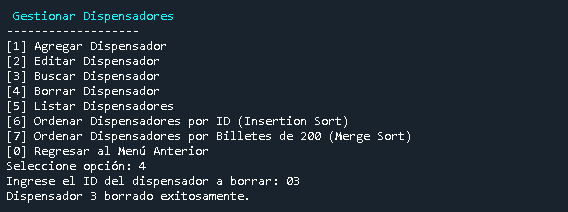
-Spyder. (s/f). Spyder IDE - The Scientific Python Development Environment.https://www.spyder-ide.org/

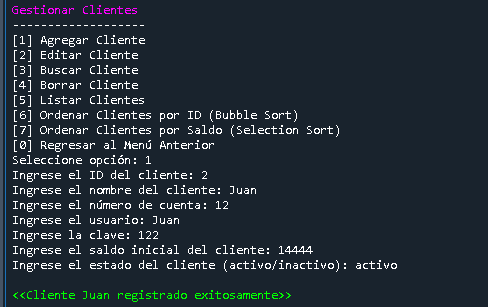
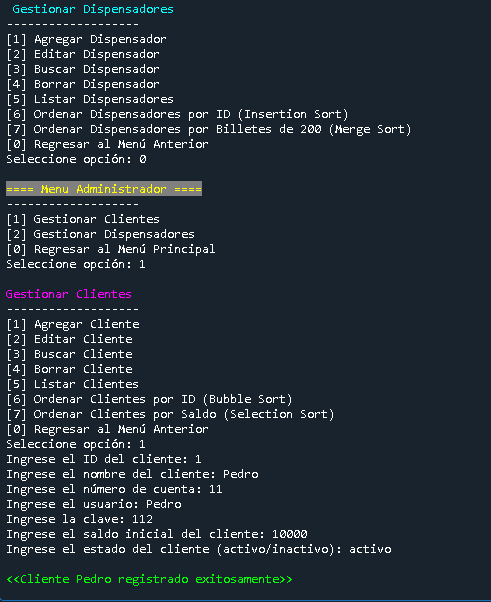
-Vásquez. (s.f.). Unidad I: Algoritmos de búsqueda y ordenamiento. <https://cristianvasquez40.wordpress.com/unidad-i-algoritmos-de-busqueda-y-ordenamiento/>

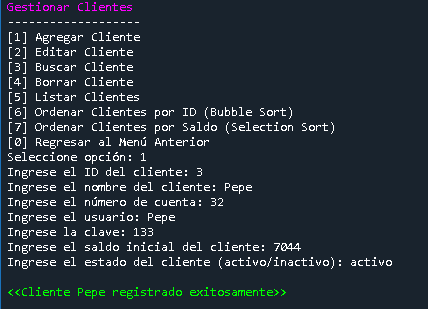
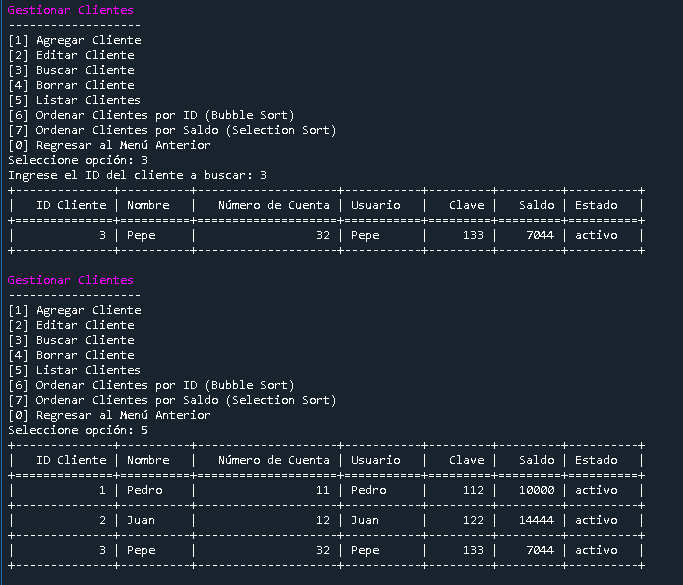
-Wikipedia. (s.f.). Ordenamiento por selección. En Wikipedia, La enciclopedia libre. <https://es.wikipedia.org/wiki/Ordenamiento_por_selecci%C3%B3n>

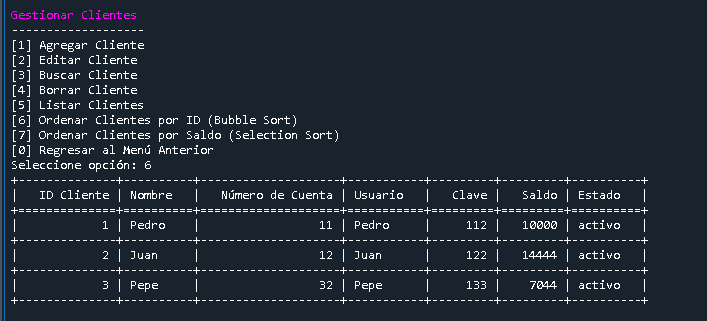
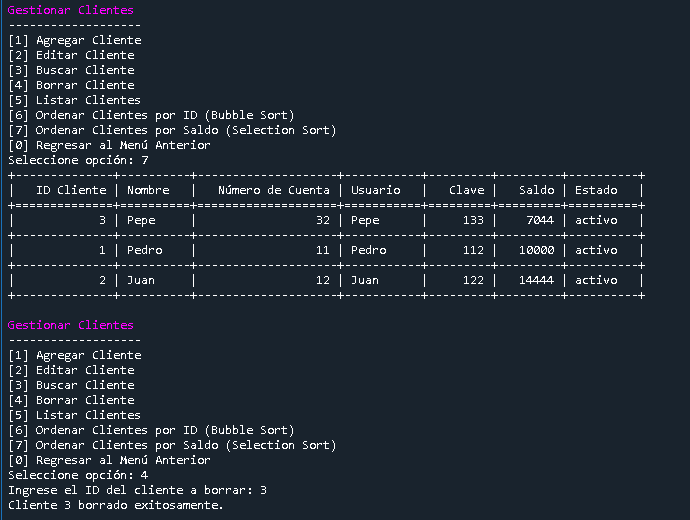
# VII. ANEXOS

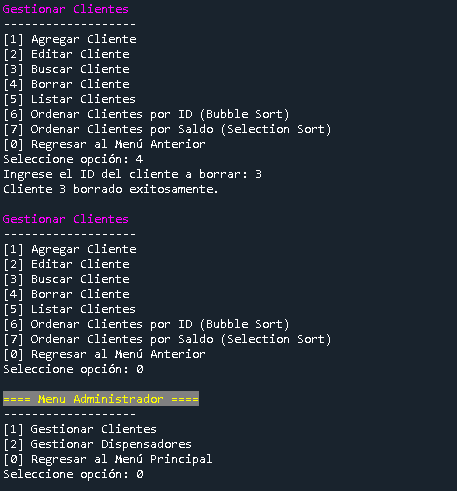
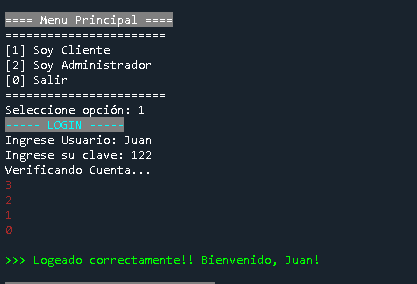
****

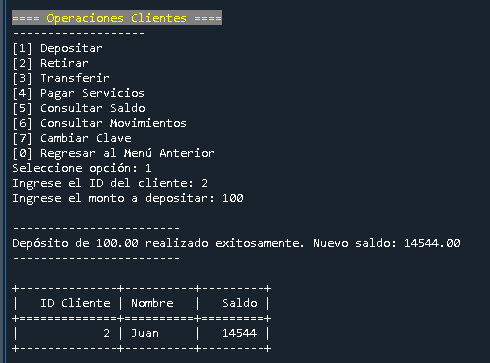
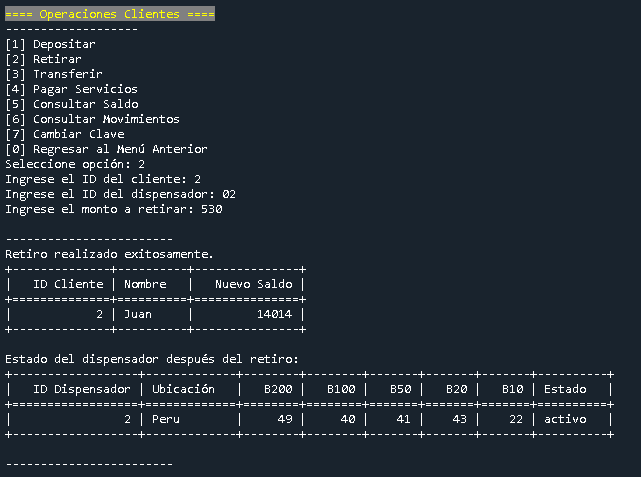
****

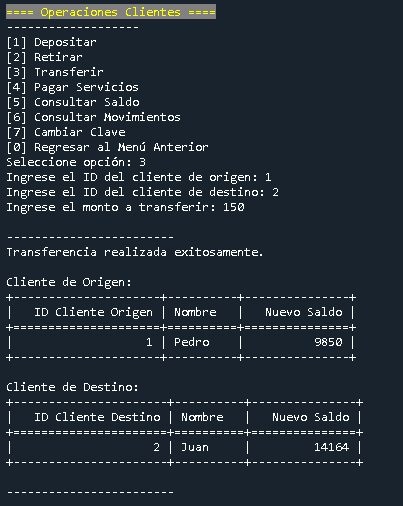
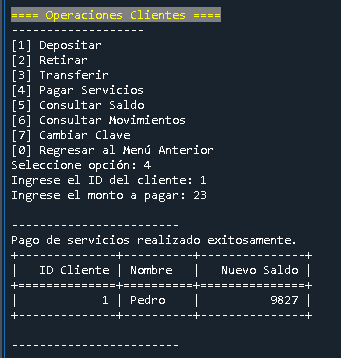
****

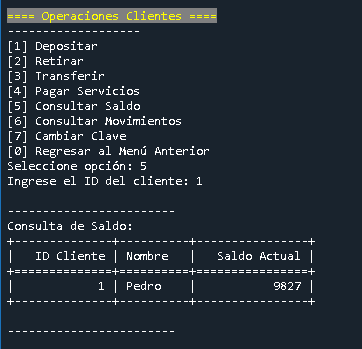
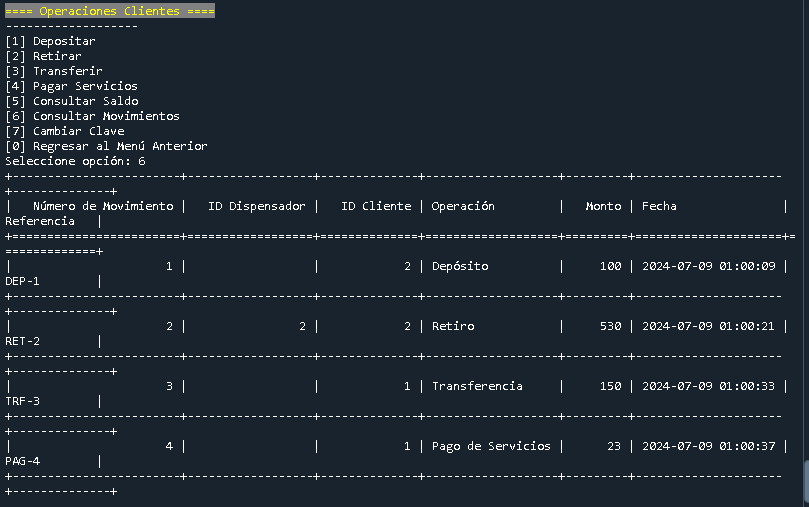
****

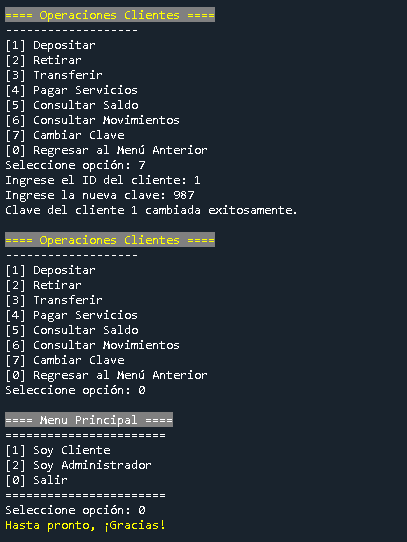
****

****

****

****

****

****

**Enlaces:**

Link del Software: <https://www.mediafire.com/file/9x1s7nw7o3iql2g/ORIGEN_vFINAL.zip/file>

Link del Video: <https://1drv.ms/v/s!ArL4F5wNQHW7gT1iHCZ6I9VUcxM_?e=c3az3V>

(Cargar el video en 1012p HD para mejor visualización en “Settings”)