Nombres: Richard Steven

Apellidos: Arizandieta Yol

Carné: 202400111

Curso: (LAB) Estructura de Datos (Sección A)

Catedrático: Ing. Edgar René Ornelis Hoíl

Auxiliar: Daniel Monterroso

MANUAL TÉCNICO PARA PROYECTO

Sistema de Gestión de Aeropuerto

GUATEMALA 27 DE DICIEMBRE DE 2025

Introducción

El presente manual técnico documenta el desarrollo del Sistema de Gestión de Aeropuerto, una aplicación desarrollada en C++, cuyo objetivo es simular y administrar la información relacionada con aviones, pilotos, rutas y asignaciones de vuelos, aplicando estructuras de datos avanzadas vistas en el curso de Estructuras de Datos:

- Árbol B
- Árbol Binario de Búsqueda
- Tabla Hash
- Grafo dirigido
- Matriz dispersa

Además, se utilizan archivos JSON y TXT como fuentes de entrada y Graphviz para la generación de reportes visuales que representan el estado interno de cada estructura.

Requisitos del Sistema

Software Requerido:

- Sistema operativo: Windows 10 / 11, macOS o distribuciones Linux modernas.
- Compilador de C++: MinGW, GCC o Visual Studio
- Editor recomendado: Visual Studio Code o CLion
- Graphviz instalado y configurado en el PATH
- Consola de comandos: CMD o PowerShell

Hardware Requerido:

- Procesador: Intel Core i3 o superior.
- Memoria RAM: Mínimo 4 GB (8 GB recomendados).
- Espacio en disco: 300 MB libres.

Arquitectura del sistema

El sistema sigue una arquitectura modular, donde cada estructura de datos y funcionalidad se encuentra encapsulada en archivos .h independientes, facilitando el mantenimiento y comprensión del código.

Estructuras principales:

- Árbol B (orden 5)

    - Almacena los aviones con estado Disponible
    - Llave: Número de registro del avión

- Lista Circular Doblemente Enlazada
    - Aviones en estado Mantenimiento

- Árbol Binario de Búsqueda
    - Pilotos ordenados por horas de vuelo

- Tabla Hash
    - Pilotos indexados por ID
    - Tamaño M = 19

- Grafo Dirigido
    - Rutas aéreas entre ciudades

- Matriz Dispersa
    - Relación entre vuelo y ciudad destino asignada a un piloto


SistemaAeropuerto/
src/
main.cpp
modelos/
Avion.h
Piloto.h
estructuras/
ArbolBAviones.h
ArbolBinarioPilotos.h
TablaHashPilotos.h
GrafoRutas.h
MatrizDispersa.h
cargadores/
CargadorAviones.h
CargadorPilotos.h
CargadorRutas.h
utils/
Graphviz.h
Utilidades.h
archivos/
aviones.json
pilotos.json
rutas.txt
reportes/
*.dot
*.png

Descripción de módulos clave:

main.cpp: Archivo principal del sistema. Contiene el menú de opciones, controla el flujo del programa y coordina la interacción entre todas las estructuras. Desde aquí se ejecuta la carga de archivos, consultas, asignaciones y generación de reportes.

ArbolBAviones.h: Implementa un árbol B de orden 5 para almacenar los aviones disponibles. Permite inserción, búsqueda, recorrido y generación de reportes gráficos.

ListaCircularAviones.h: Administra los aviones en mantenimiento mediante una lista circular doblemente enlazada, permitiendo mover aviones entre estructuras según su estado.

ArbolBinarioPilotos.h: Almacena los pilotos ordenados por horas de vuelo. Permite recorridos:

- Preorden
- Inorden
- Postorden

TablaHashPilotos.h: Permite el acceso rápido a los pilotos mediante su ID. Utiliza la función:

h(llave) = llave mod 19

GrafoRutas.h: Implementa un grafo dirigido usando listas de adyacencia. Representa las rutas entre ciudades y permite calcular la ruta más corta.

MatrizDispersa.h: Relaciona vuelo y ciudad destino. Permite inserciones dinámicas y eliminación de nodos cuando un piloto se da de baja.

Graphviz.h: Genera los archivos .dot y .png para visualizar gráficamente cada estructura del sistema.

Descripción de Archivos Clave:

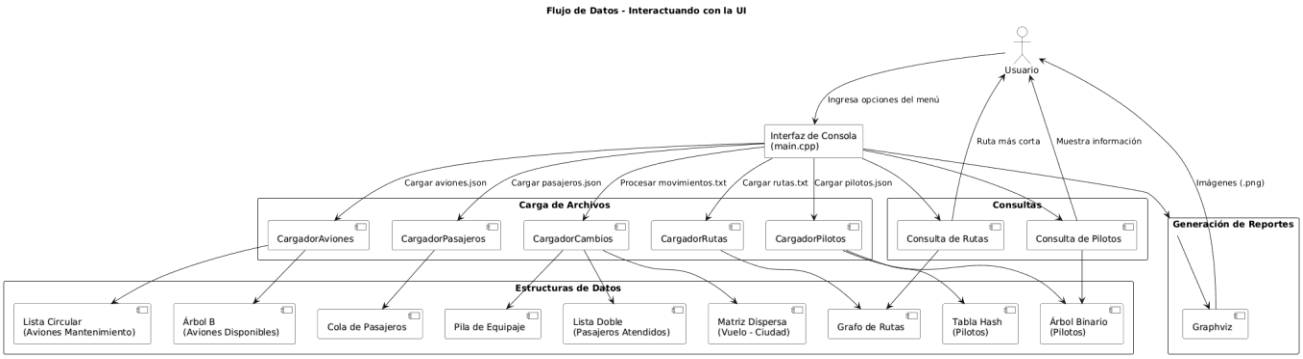
Archivo	Ubicación	Función principal	Descripción
main.cpp	/src/	Controlador principal	Contiene el menú principal del sistema y controla el flujo de ejecución. Desde este archivo se realizan las cargas de archivos, consultas, asignaciones de vuelos, ejecución de algoritmos y generación de reportes con Graphviz.

Avion.h	/src/modelos/	Modelo de datos	Define la estructura del objeto Avión con atributos como número de registro, modelo, fabricante, año de fabricación, capacidad, peso máximo, aerolínea y estado.
Piloto.h	/src/modelos/	Modelo de datos	Define la estructura del objeto Piloto con información como ID, nombre, horas de vuelo y estado dentro del sistema.
ArbolBAviones.h	/src/estructuras/	Árbol B	Implementa un árbol B de orden 5 para almacenar los aviones disponibles, utilizando como llave el número de registro. Permite inserción, búsqueda, recorrido y generación de reportes gráficos.
ListaCircularAviones.h	/src/estructuras/	Lista circular	Administra los aviones en mantenimiento mediante una lista circular doblemente enlazada. Permite mover aviones entre estructuras según su estado.
ArbolBinarioPilotos.h	/src/estructuras/	Árbol binario de búsqueda	Almacena los pilotos ordenados por horas de vuelo. Permite recorridos en preorden, inorden y postorden, mostrados en consola y en reportes gráficos.
TablaHashPilotos.h	/src/estructuras/	Tabla hash	Permite el acceso rápido a los pilotos mediante su ID. Utiliza una tabla hash de tamaño 19 con la función de dispersión llave mod 19.
GrafoRutas.h	/src/estructuras/	Grafo dirigido	Implementa un grafo dirigido utilizando listas de adyacencia para representar las rutas entre ciudades y sus distancias. Permite calcular la ruta más corta entre dos destinos.
MatrizDispersa.h	/src/estructuras/	Matriz dispersa	Relaciona los vuelos con las ciudades destino asignadas a pilotos. Permite inserciones dinámicas y eliminación

			de nodos cuando un piloto se da de baja.
CargadorAviones.h	/src/cargadores/	Carga de datos	Lee el archivo aviones.json, interpreta los datos, crea objetos de tipo Avión y los inserta en las estructuras correspondientes según su estado.
CargadorPilotos.h	/src/cargadores/	Carga de datos	Lee el archivo pilotos.json, interpreta los registros y crea los objetos Piloto que se insertan en el árbol binario y la tabla hash.
CargadorRutas.h	/src/cargadores/	Carga de datos	Procesa el archivo de rutas en formato TXT, creando el grafo dirigido con las ciudades y distancias correspondientes.
Utilidades.h	/src/utils/	Funciones auxiliares	Contiene funciones de limpieza y normalización de cadenas para facilitar la lectura correcta de los archivos de entrada.
Graphviz.h	/src/utils/	Generación de reportes	Se encarga de generar los archivos .dot y .png mediante Graphviz para visualizar gráficamente las estructuras de datos del sistema.
***.dot**	/reportes/	Archivo Graphviz	Archivos intermedios generados para la creación de los reportes gráficos del sistema.
***.png**	/reportes/	Reporte gráfico	Imágenes generadas por Graphviz que representan visualmente el estado de las estructuras de datos.

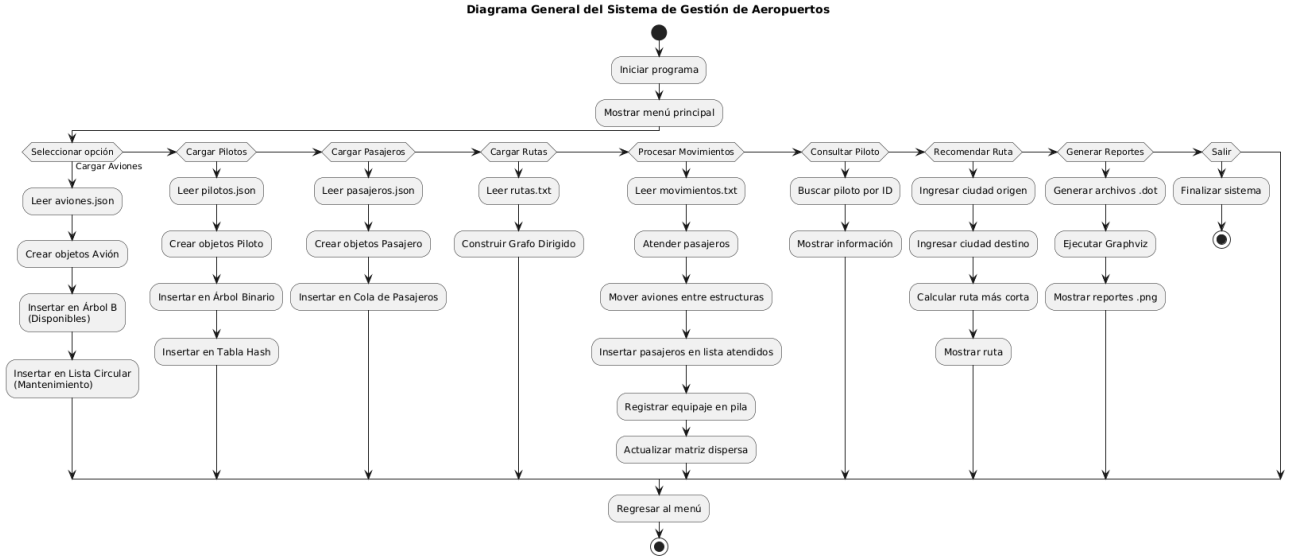
Flujo de Datos

- **Interactuando con la UI**



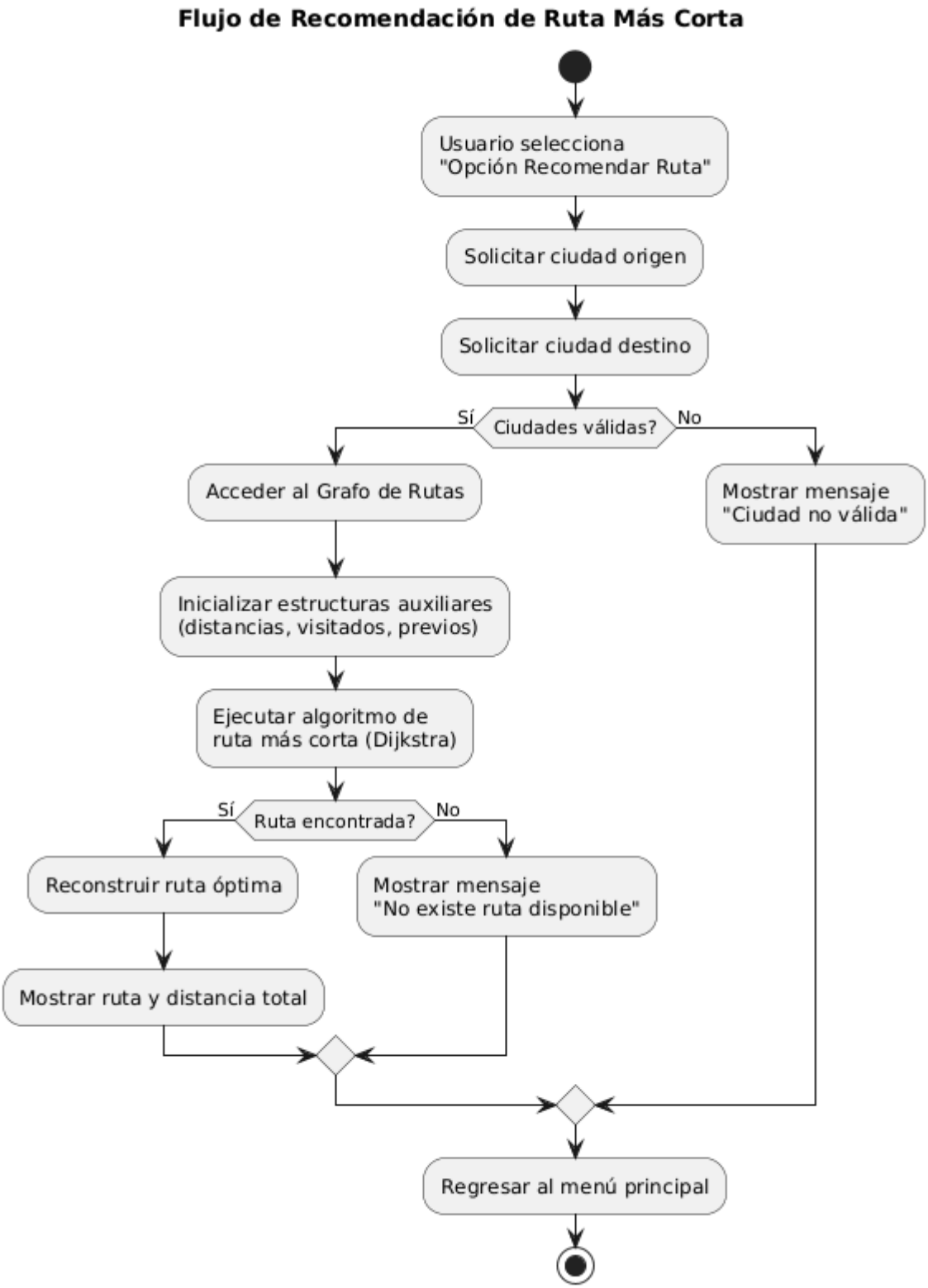
Ver imagen en grande

- **Diagrama de Flujo simple**



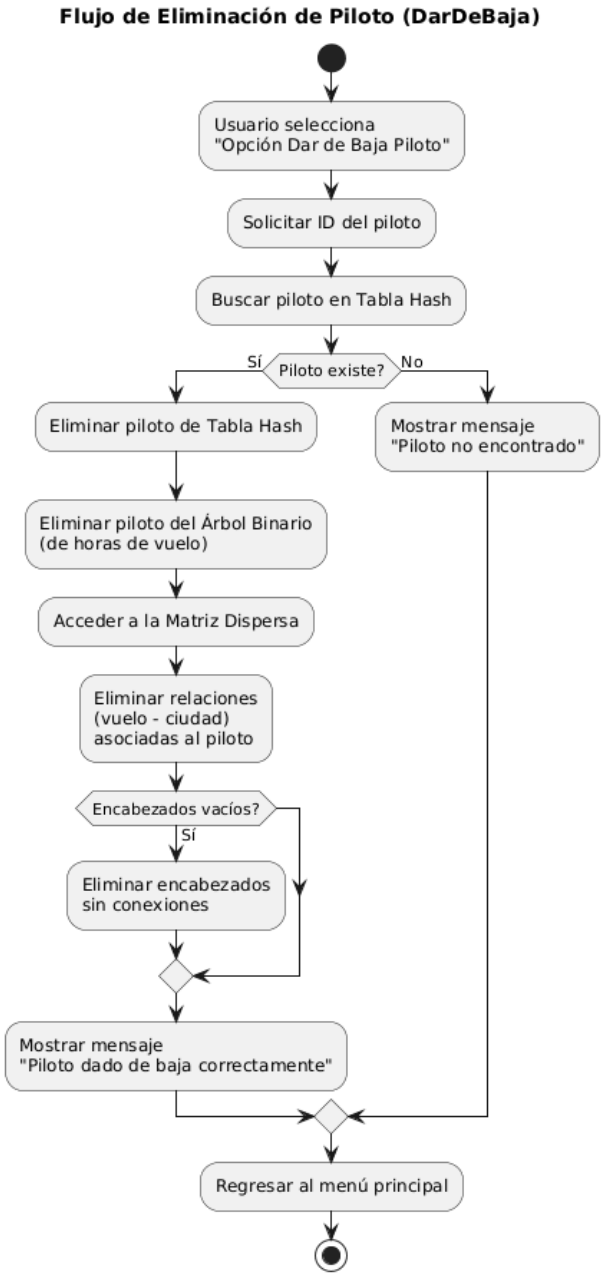
Ver imagen en grande

- **Flujo de Recomendación de Ruta más Corta de Dijkstra**



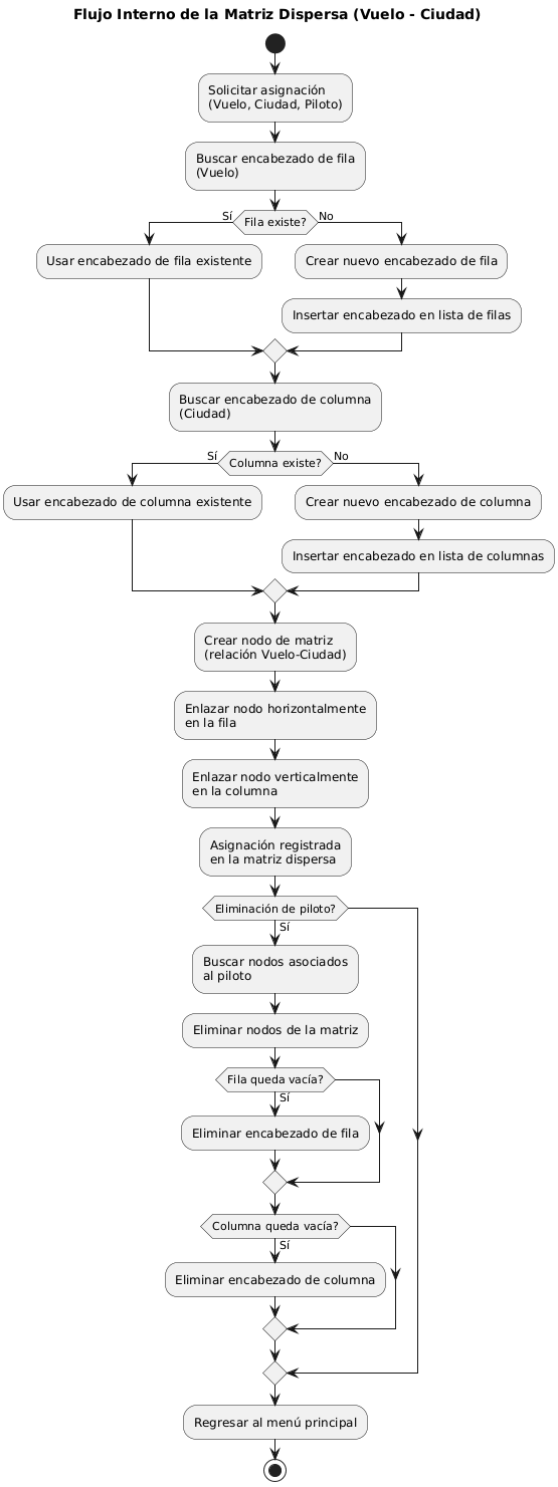
Ver imagen en grande

- **Flujo de Eliminación de Piloto**



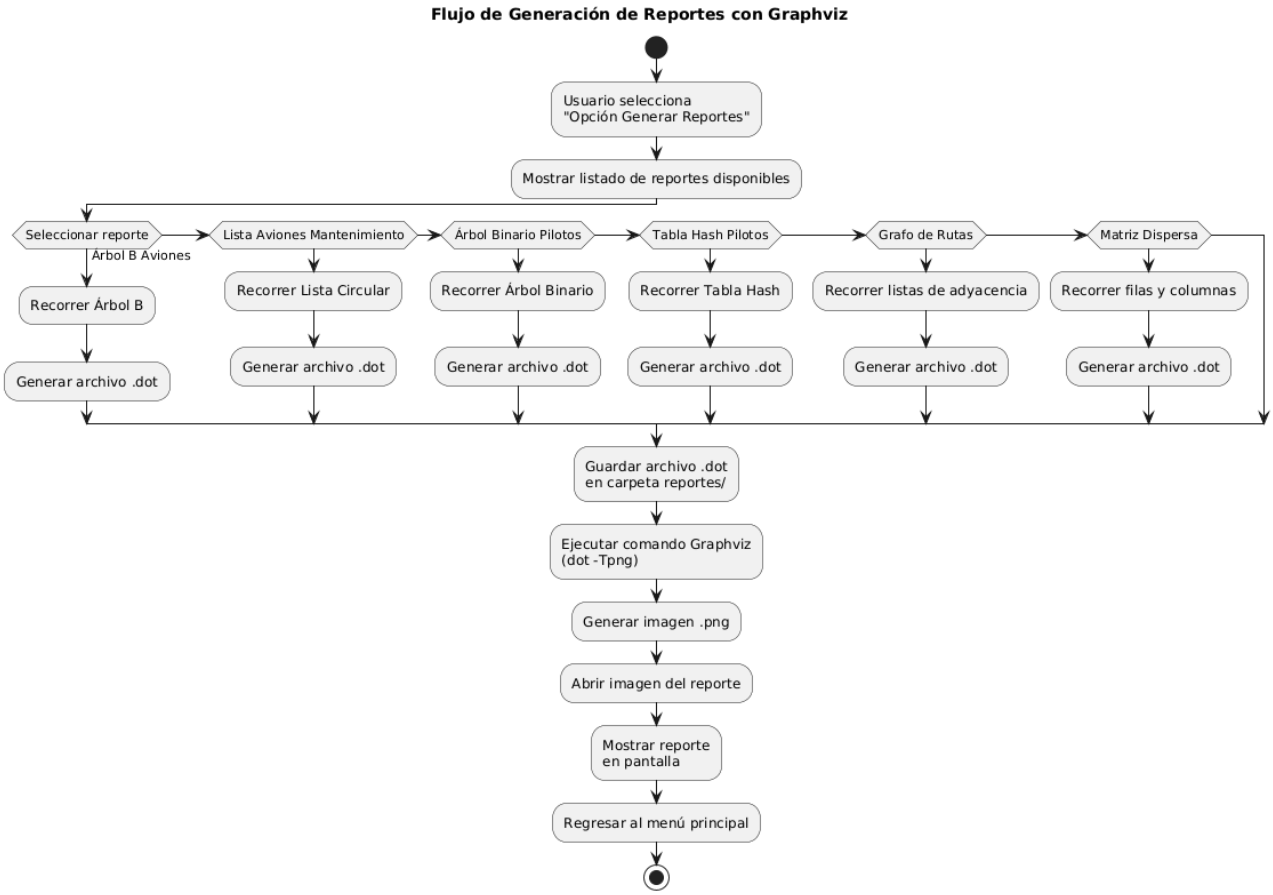
Ver imagen en grande

- **Flujo Interno de la Matriz Dispersa**



Ver imagen en grande

- **Flujo de Generación de Reportes con Graphviz**



Ver imagen en grande

Decisiones Técnicas y Limitaciones

Decisiones Técnicas:

1. Uso del lenguaje C++

Se eligió el lenguaje de programación C++ debido a su eficiencia en el manejo de memoria dinámica y su capacidad para implementar estructuras de datos complejas como árboles, grafos, tablas hash y matrices dispersas. Además, permite un control detallado sobre el uso de punteros, lo cual es fundamental para los objetivos del curso.

2. Implementación manual de estructuras de datos

Todas las estructuras de datos utilizadas en el sistema fueron implementadas manualmente, sin hacer uso de contenedores avanzados de la STL. Entre ellas se incluyen:

- Árbol B de orden 5
- Árbol binario de búsqueda
- Tabla hash con listas para colisiones
- Grafo dirigido con listas de adyacencia
- Matriz dispersa con encabezados dinámicos

Esta decisión garantiza una comprensión profunda del funcionamiento interno de cada estructura y cumple con los lineamientos académicos del proyecto.

3. Arquitectura modular

El sistema fue diseñado siguiendo una arquitectura modular, separando los componentes en carpetas como modelos, estructuras, cargadores y utilidades. Esto permite:

- Mayor claridad del código
- Facilidad de mantenimiento
- Posibilidad de ampliaciones futuras

4. Uso de Graphviz para generación de reportes

Se utilizó Graphviz para la generación de reportes gráficos debido a su capacidad para representar visualmente estructuras complejas como árboles, grafos y matrices dispersas. Los reportes se generan automáticamente desde la aplicación, cumpliendo con las restricciones del proyecto.5. Uso de Graphviz para generar reportes

5. Lectura de archivos sin librerías externas

Los archivos JSON y TXT se procesan sin el uso de librerías externas, utilizando lectura de archivos y limpieza manual de cadenas. Esta decisión se tomó para mantener el proyecto acorde al nivel académico solicitado y evitar dependencias innecesarias.

6. Uso de consola como interfaz

La interfaz del sistema se basa exclusivamente en consola, lo cual simplifica la interacción, reduce la complejidad del proyecto y cumple con los requerimientos del curso, priorizando la lógica de las estructuras sobre la presentación gráfica.

7. Procesamiento en memoria

Toda la información del sistema se maneja en memoria durante la ejecución del programa. Esto simplifica el diseño, mejora el rendimiento y evita la necesidad de mecanismos adicionales de persistencia no solicitados en el enunciado.

Limitaciones del Sistema

1. No existe persistencia de datos

Los cambios realizados durante la ejecución del programa no se guardan en archivos. Al finalizar la aplicación, toda la información vuelve a su estado inicial definido en los archivos de entrada.

2. Validación limitada de archivos de entrada

El sistema asume que los archivos JSON y TXT están correctamente formateados. Errores de sintaxis o datos inconsistentes pueden provocar comportamientos inesperados, ya que el manejo de errores se mantiene básico.

3. Interfaz únicamente por consola

El sistema no cuenta con una interfaz gráfica, lo cual limita la experiencia de usuario frente a aplicaciones reales. Sin embargo, esta decisión fue tomada para priorizar el aprendizaje de estructuras de datos.

4. Falta de control de concurrencia

La aplicación no está diseñada para manejar múltiples usuarios simultáneamente. Todas las operaciones se realizan de forma secuencial, lo cual es adecuado para un entorno académico.

5. Capacidad de búsqueda limitada

Las búsquedas se realizan únicamente por ciertos atributos (como ID de piloto o número de registro del avión), sin mecanismos avanzados de filtrado o búsqueda combinada.

6. Dependencia de memoria disponible

El sistema no impone límites explícitos al número de elementos almacenados. El rendimiento y estabilidad dependen de la memoria disponible del equipo donde se ejecuta la aplicación.

7. Manejo básico de errores

No se contemplan de forma exhaustiva casos como:

- Archivos inexistentes
- Permisos insuficientes
- Datos incompletos
- Operaciones sobre estructuras vacías

Esto se realizó para mantener la complejidad del proyecto dentro de los objetivos del curso.