Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Estructura de Datos
Sección A

PROYECTO
Manual de TECNICO

Ludwing Alexander López Ortiz
201907608

INTRODUCCIÓN

El presente proyecto tiene como objetivo la implementación de nuevas funcionalidades en un sistema de gestión de aeropuertos previamente desarrollado. Este sistema, que ya incluía la gestión de vuelos, pasajeros, equipajes y otras características esenciales, se verá ampliado con la incorporación de estructuras de datos avanzadas. Entre las estructuras a considerar se encuentran el árbol binario de búsqueda equilibrado, el árbol B, la matriz dispersa, la tabla hash y los grafos. Estas mejoras permitirán un manejo más eficiente y robusto de la información relacionada con el aeropuerto, optimizando así el rendimiento del sistema y su capacidad para manejar datos complejos y voluminosos.

En el contexto de la simulación de la llegada de aviones, se ha identificado la necesidad de analizar una nueva entidad: los pilotos del aeropuerto. La entidad piloto será fundamental y se integrará en diversas estructuras de datos, interactuando de manera significativa con los aviones y las rutas que estos recorren. Los pilotos no solo serán administrados de manera individual, sino que también se considerará su participación en la asignación de vuelos y en la optimización de rutas aéreas.

El árbol binario de búsqueda equilibrado se utilizará para mantener un registro eficiente de los pilotos, permitiendo una rápida inserción, eliminación y búsqueda de información. El árbol B, por su parte, se aplicará en la gestión de datos de mayor volumen y con requerimientos de acceso más complejos, como podrían ser los registros históricos de vuelos y pilotos. La matriz dispersa será empleada para representar relaciones esparsas, como la asociación entre pilotos y vuelos o destinos. La tabla hash facilitará la búsqueda y recuperación rápida de datos específicos, optimizando el acceso a información clave. Finalmente, los grafos permitirán modelar y analizar las rutas que los aviones pueden recorrer, proporcionando una visión detallada y dinámica de las posibles trayectorias aéreas.

La integración de estas estructuras de datos avanzadas no solo mejorará la eficiencia y funcionalidad del sistema de gestión de aeropuertos, sino que también proporcionará una plataforma robusta y flexible para futuras expansiones y mejoras. La correcta administración y análisis de la entidad piloto, junto con la optimización de rutas, contribuirán significativamente a la operación eficiente del aeropuerto, garantizando una gestión más precisa y efectiva de los recursos y mejorando la experiencia tanto de los empleados como de los pasajeros. Este proyecto representa un paso importante hacia la modernización y sofisticación de los sistemas de gestión aeroportuaria, alineándose con las demandas crecientes de un sector en constante evolución.

OBJETIVOS

General

Aplicar los conocimientos del curso de Estructuras de Datos en la creación de soluciones de software.

Específicos

- Hacer un uso correcto de la memoria dinámica y apuntadores en el lenguaje de programación C++.
- Aplicar los conocimientos adquiridos sobre estructuras de datos no lineales.
- Utilizar la herramienta graphviz para generar reportes.

Especificaciones del Sistema Recomendado

Para asegurar el rendimiento óptimo y la estabilidad del programa, se recomienda utilizar un sistema con las siguientes especificaciones mínimas:

Hardware Recomendado

- **Procesador (CPU)**:
  - Tipo: Intel Core i5 o superior / AMD Ryzen 5 o superior
  - Velocidad: 3.0 GHz o superior
- **Memoria (RAM)**:
  - Capacidad: 8 GB mínimo (16 GB recomendado para tareas intensivas)
- **Almacenamiento**:
  - Tipo: Unidad de Estado Sólido (SSD) para mejor rendimiento
  - Capacidad: 256 GB mínimo (recomendado 512 GB o superior)
- **Tarjeta Gráfica (GPU)**:
  - Tipo: NVIDIA GeForce GTX 1050 o equivalente (recomendado NVIDIA GTX 1650 o superior)
  - Memoria: 2 GB mínimo (4 GB recomendado)
- **Monitor**:
  - Resolución: 1920x1080 (Full HD) o superior
  - Tamaño: 21 pulgadas o superior
- **Puertos y Conectividad**:
  - Puertos USB 3.0 o superior
  - Conexión a Internet: Ethernet o Wi-Fi 802.11ac

Software Recomendado

- **Sistema Operativo (OS)**:
  - Windows 10 Pro (64-bit) o superior / macOS 10.15 (Catalina) o superior / Ubuntu 20.04 LTS o superior
- **Entorno de Desarrollo**:
  - IDE: Microsoft Visual Studio 2019 o superior / CLion / Code::Blocks
  - Compilador: GCC 9.3 o superior / Clang 10.0 o superior / MSVC (Microsoft Visual C++)
- **Librerías y Dependencias**:
  - Graphviz para la visualización de grafos y estructuras de datos
  - CMake para la gestión de la construcción del software (opcional)
- **Software Adicional**:
  - Navegador Web: Google Chrome o Mozilla Firefox para la documentación y recursos en línea
  - Herramientas de Version Control: Git para la gestión del código fuente

Recomendaciones Adicionales

- **Respaldo de Datos**: Se recomienda utilizar un servicio de almacenamiento en la nube o un disco duro externo para realizar copias de seguridad periódicas de los proyectos y datos importantes.
- **Mantenimiento del Sistema**: Mantener el sistema operativo y todo el software relacionado actualizado para evitar problemas de compatibilidad y aprovechar las mejoras de seguridad.

Librerías Utilizadas

1. **iostream**: Esta librería se utiliza para realizar operaciones de entrada y salida, como leer desde el teclado y escribir en la consola.
2. **json.hpp**: Esta librería pertenece a la biblioteca JSON para C++ (nlohmann::json) y se utiliza para manejar archivos JSON, proporcionando funciones para parsear, manipular y generar JSON.
3. **fstream**: Se usa para trabajar con archivos de entrada y salida. En este código se utiliza para leer archivos.
4. **sstream**: Proporciona clases para operar con cadenas de caracteres como si fueran flujos de datos. Se utiliza para dividir líneas de texto en componentes separados.
5. **string**: Proporciona la clase string, que representa cadenas de caracteres.
6. **regex**: Proporciona soporte para expresiones regulares, utilizadas aquí para buscar y extraer patrones en cadenas de texto.

Clases Importadas

1. **CircularDoble**: Una clase que probablemente implementa una lista doblemente enlazada circular.
2. **ArbolB**: Una clase que implementa un árbol B.
3. **ABB**: Una clase que implementa un árbol binario de búsqueda.
4. **Hash**: Una clase que implementa una tabla hash.
5. **ListaD**: Una clase que probablemente implementa una lista de adyacencia.
6. **Matriz**: Una clase que implementa una matriz dispersa.

Métodos Utilizados y Explicación del Código

Estructuras

Se crean instancias de las clases importadas:

cpp
Copiar código

```
CircularDoble Circular;
BTree ArbolB(3);
ArbolDeBusquedaBinaria Bb;
TablaHash tabla;
ListaEnlazadaD adyascencia;
MatrizDispersa matrix;
```

Funciones para Leer Archivos JSON

1. **leerJSON_Aviones()**:
    - Solicita al usuario la ruta del archivo JSON de aviones.
    - Lee y parsea el archivo JSON.
    - Itera sobre los elementos del JSON y extrae los datos de cada avión.
    - Agrega vuelos y destinos a la matriz dispersa.
    - Si el avión está disponible, se agrega al árbol B, de lo contrario, se agrega a la lista circular doble.
    - Muestra los datos cargados en la consola.
2. **leerJSON_Pilotos()**:
    - Similar a `leerJSON_Aviones()`, pero para pilotos.
    - Agrega pilotos a la matriz dispersa, al árbol binario de búsqueda y a la tabla hash.

Función para Leer Archivo de Texto con Rutas

leerArchivo():

- Solicita al usuario la ruta del archivo de texto.
- Lee el archivo línea por línea.
- Parse las líneas en el formato "origen/destino/distancia".
- Agrega las rutas a la lista de adyacencia.

Función para Leer Archivo de Texto con Movimientos

leerMovimientos():

- Similar a `leerArchivo()`, pero maneja diferentes formatos de líneas.
- Procesa líneas en el formato "MantenimientoAviones" y "DarDeBaja".
- Realiza operaciones de mantenimiento, como mover aviones entre estructuras.

Funciones de Utilidad

1. **Mensaje()**:
    - Muestra un menú de opciones al usuario.
2. **Consulta_horas()**:
    - Muestra submenú para consultar horas de vuelo en diferentes órdenes (preorden, inorden, postorden) en el árbol binario de búsqueda.
3. **RutaMasCorta()**:
    - Solicita al usuario origen y destino.
    - Llama al método para encontrar la ruta más corta en la lista de adyacencia.

Función Principal

main():

- Muestra el menú principal en un bucle.
- Ejecuta las opciones seleccionadas por el usuario.
- Incluye manejo de errores para la carga de archivos JSON.

Recomendaciones

1. **Manejo de Errores**: Es recomendable agregar más manejo de errores para robustecer el programa, por ejemplo, verificando si los archivos existen antes de intentar abrirlos.
2. **Modularización**: Se podría mejorar la modularización del código moviendo cada funcionalidad a una clase o módulo separado.
3. **Comentarios**: Agregar más comentarios para explicar secciones críticas del código ayudaría en la mantenibilidad.
4. **Validación de Entrada**: Se debe validar la entrada del usuario para evitar errores o comportamientos inesperados.
5. **Optimización**: Revisar la eficiencia de las operaciones, especialmente al manejar estructuras de datos grandes.

CAPTURAS DE CÓDIGO

LECTOR DE JSON

```
json leerJSON_Aviones() {

    std::string documento; //variable con nombre del documento
    std::string ruta = "C:/Users/ludwi/OneDrive/Escritorio/-201907608_EDD_Proyecto/"; //ruta de archivo predefinido

    std::cout << "INGRESE LA RUTA DEL ARCHIVO: " << std::endl; //ruta: C:/Users/ludwi/OneDrive/Escritorio/-201907608_EDD_Proyecto/aviones.json
    std::cin >> documento; //INGRESAR EL NUMERO SELECCIONADO DE LA OPCION

    ruta = ruta +documento; //agregar el nombre del archivo a la ruta

    std::cout << "ruta: " <<ruta<< std::endl; //imprimir ruta del archivo

    std::ifstream inputFile(ruta);
    json jsonData;

    if (inputFile.is_open()) {
        inputFile >> jsonData;
        inputFile.close();
    } else {
        std::cerr << "No se pudo abrir el archivo: " << std::endl;
        throw std::runtime_error("Archivo no encontrado");
    }

     // Mostrar el contenido del JSON
    //std::cout << jsonData.dump(4) << std::endl;
    std::cout<< "DATOS QUE SER CARGARON: "<< std::endl;
    std::cout<< "-------------------------------------------------------------"<< std::endl;

    // Acceder a los elementos del JSON
    for (const auto& aviones : jsonData) {

        //OBTENER LOS ELEMENTOS DEL JSON AVIONES
        std::string vuelo = aviones["vuelo"];
        std::string numero_Registro = aviones["numero_de_registro"];
        std::string modelo = aviones["modelo"];
        int capacidad = aviones["capacidad"];
        std::string aerolinea = aviones["aerolinea"];
        std::string estado = aviones["estado"];
        std::string destino = aviones["ciudad_destino"];


        matrix.agregarVueloDestino(vuelo, destino); //crear vuelos y destinos (filas y columnas) en la matriz dispersa


        if (estado == "Disponible")
        {
            Avion avion = {vuelo, numero_Registro, modelo, capacidad, aerolinea, estado, destino};
            ArbolB.insert(avion);
        }else
        {
            //agregar a la lista circular doble
            Circular.agregar(vuelo,numero_Registro,modelo,capacidad,aerolinea,estado,destino);
        }


    //MOSTRAR LOS DATOS DEL JSCON CARGADOS
        std::cout<< vuelo<< std::endl;
        std::cout<< numero_Registro<< std::endl;
        std::cout<< modelo<< std::endl;
        std::cout<< capacidad<< std::endl;
        std::cout<< aerolinea<< std::endl;
        std::cout<< destino<< std::endl;
        std::cout<< estado<< std::endl;
        std::cout<< "-------------------------------------------------------------"<< std::endl;


    }

     return jsonData; // Retornar los datos JSON leídos

}

```

LECTOR DE ARCHIVOS DE TEXTO

```cpp
void leerArchivo() {

    std::string documento; //variable con nombre del documento
    std::string ruta = "C:/Users/ludwi/OneDrive/Escritorio/-201907608_EDD_Proyecto/"; //ruta de archivo predefinido

    std::cout << "INGRESE LA RUTA DEL ARCHIVO: " << std::endl; //ruta: C:/Users/ludwi/OneDrive/Escritorio/-201907608_EDD_Proyecto/aviones.json
    std::cin >> documento; //INGRESAR EL NUMERO SELECCIONADO DE LA OPCION

    ruta = ruta +documento; //agregar el nombre del archivo a la ruta

    std::cout << "ruta: " <<ruta<< std::endl; //imprimir ruta del archivo

    std::ifstream archivo(ruta);
    if (!archivo.is_open()) {
        std::cerr << "No se pudo abrir el archivo " << documento << std::endl;
        return;
    }

    std::string linea;
    while (std::getline(archivo, linea)) {
        if (linea.back() == ';') {
            linea.pop_back();  // Elimina el último carácter ';'
        }

        std::istringstream stream(linea);
        std::string origen, destino;
        int distancia;
        if (std::getline(stream, origen, '/') && std::getline(stream, destino, '/') && (stream >> distancia)) {

            Ruta r = {origen, destino, distancia}; //se crea el objeto de estructura ruta

            adyascencia.agregarRuta(r); //se agrega a la lista de adyascencia

        } else {
            std::cerr << "Formato incorrecto en la línea: " << linea << std::endl;
        }
    }

    archivo.close();
}
```

INGRESAR RUTA

```cpp
void RutaMasCorta(){

    //variables para pedir ruta
    std::string origen;
    std::string destino;

    std::cout << "|| INGRESE UN ORIGEN: ";
    std::cin >> origen;
    std::cout << std:: endl; //salto de linea

    std::cout << "|| INGRESE UN DESTINO: ";
    std::cin >> destino;

    adyascencia.encontrarRutaMasCorta(origen, destino); //se llama a la funcion destino para realizar la ruta


}
```

ÓRDENES DE HORAS

```
oid Consulta_horas()
{
    bool salida = true; //variable para entrar y salir del ciclo del menu de ordenes
    while (salida) //ciclo para mostrar los ordenes
    {
        std::cout << "|| SELECCIONES UN ORDEN: ------------||" << std::endl;
        std::cout << "|| 1. PREORDEN.                     ||" << std::endl;
        std::cout << "|| 2. INORDEN.                      ||" << std::endl;
        std::cout << "|| 3. POSTORDEN.                    ||" << std::endl;
        std::cout << "|| 4. REGRESAR.                     ||" << std::endl;

        int seleccion; //variable seleccion de menu
        std::cin >> seleccion; //INGRESAR EL NUMERO SELECCIONADO DE LA OPCION

        switch (seleccion)
        {
        case 1:
            Bb.mostrarPreorden();
            break;
        case 2:
            Bb.mostrarInorden();
            break;
        case 3:
            Bb.mostrarPostorden();
            break;
        case 4:
            salida = false; //salir del sistema y volver al menu principal
            break;

        default:

             std::cout << "|| NO INGRESO UNA OPCION VALIDA, VUELVA A INGRESAR UNA OPCION QUE SI SEA VALIDA PORFAVOR.||" << std::endl;
            break;
        }

    }
```

LIBRERÍAS Y ESTRUCTURAS IMPLEMENTADAS

```
#include <iostream>
#include "json.hpp"
#include <fstream>
#include <sstream>
#include <string>
#include <regex>

#include "CircularDoble.h" //incluir clase lista circular doble
#include "ArbolB.h" //incluimos el arbol B
#include "ABB.h" //arbol binario de busqueda
#include "Hash.h" //tabla hash
#include "ListaD.h" //Lista de adyascencia
#include "Matriz.h" //matriz dispersa

using json = nlohmann::json;


//estructuras
CircularDoble Circular;
BTree ArbolB (3);
ArbolDeBusquedaBinaria Bb;
TablaHash tabla;
ListaEnlazadaD adyascencia;
MatrizDispersa matrix;
```

MAIN

```cpp
int main(int argc, char const *argv[])
{

    bool exit = true; // VARIABLE DE SALIDA DEL CICLO

    while (exit)
    {
        int seleccion; //variable seleccion de menu

        Mensaje(); //llamamos la presentacion del mensaje

        std::cin >> seleccion; //INGRESAR EL NUMERO SELECCIONADO DE LA OPCION

        switch (seleccion)
        {
        case 1:
            std::cout << "|| OPCION 1 CARGA DE AVIONES. ||" << std::endl;

             try {
                    json pasajerosData = leerJSON_Aviones();
            } catch (const std::runtime_error& e) {
                    std::cerr << "Error: " << e.what() << std::endl;

            }

            break;
        case 2:
            std::cout << "|| OPCION 2. CARGA DE PILOTOS. ||" << std::endl;

            try {
                    json pasajerosData = leerJSON_Pilotos();
            } catch (const std::runtime_error& e) {
                    std::cerr << "Error: " << e.what() << std::endl;

            }
            break;
        case 3:
            std::cout << "|| OPCION 3. CARGA DE RUTAS. ||" << std::endl;
            leerArchivo(); //se lee el archivo de rutas
            break;
        case 4:
            std::cout << "|| OPCION 4. CARGA DE MOVIMIENTOS. ||" << std::endl;
            leerMovimientos();
            break;
        case 5:
            std::cout << "|| OPCION 5. CONSULTA DE HORA DE VUELO. ||" << std::endl;
            Consulta_horas(); //funcion para llamar consulta de ordenes de horas
            break;

        case 6:
            std::cout << "|| OPCION 6. RECOMENDAR RUTA. ||" << std::endl;
            RutaMasCorta(); //funcion para llamar la ruta
            break;

        case 7:

            std::cout << "|| OPCION 7. VISUALIZAR REPORTES. ||" << std::endl;
            matrix.mostrarMatriz();
            Circular.graficarLista();
            ArbolB.graphviz("ArbolB.dot");
            tabla.graficar("Hash.dot");
            adyascencia.graficarGrafo("grafo.dot");
            matrix.graficarMatriz();
            matrix.graficarMatriz2();
            Bb.graficarArbol("Arbolbb.dot");


            return 0;
            break;

        case 8:

            exit = false; //salida del ciclo en caso de cerrar programa

            break;

        default:

            std::cout << " NO INGRESO UNA OPCION VALIDA! " << std::endl;

            break;
        }
    }



    return 0;
}

```