Entrega 1: Segundo Reto de Informática II

1st Daniela Escobar Velandia
Dpto. de Ingeniería Electrónica y Telecomunicaciones
Universidad de Antioquia
Medellín, Colombia
daniela.escobarv@udea.edu.co

2nd Yonathan López Mejía
Dpto. de Ingeniería Electrónica y Telecomunicaciones
Universidad de Antioquia
Medellin, Colombia
harley.lopez@udea.edu.co

***Abstract*—Este documento presenta el proceso de análisis y diseño para la gestión de un sistema de reservas de alojamientos. La propuesta contempla la implementación de cinco clases principales: Anfitrion, Huesped, Alojamiento, Reserva y Fecha, junto con sus atributos y relaciones. Para el acceso y manipulación eficiente de los datos, se emplean listas simplemente enlazadas y tablas hash desarrolladas de forma propia. Además, se establecen criterios específicos para garantizar la validez de los datos y la integridad de las relaciones entre las entidades.**

***Keywords*—Gestión de reservas, estructuras de datos personalizadas, listas enlazadas, tablas hash.**

I. Introducción

El crecimiento del turismo local ha impulsado la creación de nuevas alternativas de alojamiento que promueven una experiencia más cercana con la cultura regional. En este contexto surge UdeAStay, una propuesta de plataforma digital orientada a la gestión de estadías hogareñas en el departamento de Antioquia. El presente documento aborda la primera fase de desarrollo del sistema: el análisis del problema y el diseño de una solución estructurada, basada en principios de programación orientada a objetos.

En esta propuesta se modelan cinco clases principales —Anfitrion, Huesped, Alojamiento, Reserva y Fecha— con atributos y comportamientos definidos que permiten representar las interacciones reales entre usuarios. Uno de los principales desafíos del desarrollo es la restricción de no utilizar herencia ni bibliotecas estándar, lo que obliga a un diseño cuidadoso, enfocado en la eficiencia del manejo de memoria y la representación precisa de relaciones entre objetos.

Para ello, se emplean estructuras de datos propias como listas simplemente enlazadas y tablas hash, y se definen reglas específicas sobre la validez de los datos, su almacenamiento en archivos de texto y la integridad de las relaciones entre entidades. El diseño incluye el uso de un diagrama UML simplificado y decisiones técnicas orientadas a garantizar un sistema funcional, organizado y sostenible, que sirva como base para su posterior implementación.

II. Propuesta de solución

A. Condiciones y restricciones

Para la realización del reto se asume lo siguiente:

- Se tienen 5 clases: Afitrion, Huesped, Alojamiento, Reserva y Fecha.
- No se permite el uso de herencia.
- Se agregó un campo de contraseña para que el usuario inicie sesión.
- Los datos en los archivos de texto provienen de una fuente que garantiza su integridad y que responden al formato que se definirá más adelante.
- Las fechas siguen el siguiente formato de 10 caracteres: dd/mm/aaaa.
- Las estructuras de datos a utilizar son: mapas hash y listas simples enlazadas de implementación propia.
- Los códigos de los alojamientos y las reservas son numéricos e incrementales. Así: el alojamiento con código 1 se agregó al sistema antes que el alojamiento de código 2. Igualmente en el caso de las reservaciones.
- La identificación del huésped es un número de cédula colombiano u otro código exclusivamente numérico.
- Una reservación cancelada no se agrega al histórico.
- Se agregan al histórico únicamente aquellas reservaciones que a la fecha de corte se hayan completado.
- Los alojamientos están asociados siempre a un usuario que existe en el archivo de usuarios.
- Las reservas están asociadas a un usuario y un alojamiento que existen en el archivo de usuarios y alojamientos.

B. Formato de los archivos de texto

- **anfitriones.txt**: Se almacena en cada línea los datos del anfitrión en el siguiente formato `[cedula] [password] [antiguedad] [calificacion]`. Cada campo se separa por espacio y no deben existir dos anfitriones con el mismo código.
- **huespedes.txt**: Se almacena en cada línea los datos del huésped en el siguiente formato `[cedula] [password] [antiguedad] [calificacion]`. Cada campo se separa por espacio y no deben existir dos huéspedes con el mismo código.
- **reservaciones.txt**: Se almacena en cada línea los datos de la reservación en el siguiente formato: `[fecha_inicio];[duracion_noches];[codigo_reservacion];[codigo_alojamiento];[codigo_huesped];[pago];[fecha_pago];[precio];[ano`

taciones]. No habrá dos reservaciones con el mismo código.

- **alojamientos.txt**: Se almacena en cada línea los datos de las reservaciones en el siguiente formato: [nombre];[codigo_alojamiento];[anfitrion_responsable];[departamento];[tipo];[direccion];[precio];[ameneidades]. No habrá dos alojamientos con el mismo código. Se asume que todos los alojamientos leídos desde archivos están asociados a un anfitrión que existe.

ESTRUCTURAS DE DATOS

Las estructuras de datos a utilizar deben garantizar tiempos de acceso rápidos, flexibilidad al aumento de tamaño de datos y almacenamiento en memoria no contigua. Para esto se eligieron y se implementaron dos tipos: mapas no ordenados y listas simples. Los primeros se utilizarán para almacenar todos los punteros a las instancias de las clases creadas, mientras que los segundos, debido a su naturaleza "lenta" para la consulta, serán empleados para mantener el rastreo de los alojamientos asociados a los huéspedes de las reservas vinculadas a un usuario, o de las reservas asociadas a un alojamiento. Estos últimos se consideran que nunca serán grandes si se comparan con el número de elementos totales de cada clase, por lo cual, la penalización en las operaciones no deberá ser significativa.

Unordered_Map

Un unordered_map es una estructura de datos que almacena pares *clave-valor* y permite acceder a los valores de manera eficiente a partir de su clave, sin importar el orden de inserción.

Características principales:

- Basado en una *tabla hash* que distribuye las claves en diferentes *buckets* o cubetas, usando una función hash.
- La función hash transforma la clave en un índice numérico que indica en qué bucket se debe almacenar el par.
- Para manejar *colisiones* (cuando dos claves diferentes tienen el mismo índice), se utiliza la técnica de *encadenamiento*, donde cada bucket contiene una lista enlazada de elementos.
- Las operaciones típicas (inserción, búsqueda y eliminación) tienen un tiempo promedio cercano a $O(1)$.

Funcionamiento interno

Para insertar un nuevo par (clave, valor):

- Se calcula el índice aplicando la función hash a la clave.
- Se revisa el bucket correspondiente en la tabla.
- Si la clave ya existe, se actualiza su valor.
- Si no, se añade un nuevo nodo al inicio de la lista enlazada en ese bucket.

Para buscar un valor dado su clave:

- Se calcula el índice con la función hash.
- Se recorre la lista enlazada del bucket hasta encontrar la clave o llegar al final.

Para eliminar un par:

- Se busca el nodo con la clave.
- Se elimina el nodo ajustando los punteros de la lista enlazada.
- Se devuelve el valor eliminado para que el usuario pueda gestionarlo (liberarlo, reutilizarlo, etc.).

Consideraciones importantes

- La eficiencia depende en gran medida de la calidad de la función hash y del factor de carga (relación entre el número de elementos y el tamaño de la tabla).
- La función hash utilizada es *djb2*, un algoritmo simple y efectivo para tipos básicos.
- No se realiza *rehashing* automático; por ello, se reserva un tamaño inicial suficiente para evitar alta carga.
- La memoria usada incluye tanto la tabla de buckets como los nodos en las listas enlazadas.
- Es responsabilidad del usuario liberar la memoria de los valores cuando se eliminen.

Esta estructura permite un acceso rápido y flexible a los datos mediante claves personalizadas, siendo muy útil para implementar mapas y diccionarios en C++.

Lista simple enlazada

En este proyecto se implementa una **lista simplemente enlazada** como estructura de datos personalizada, sin utilizar la STL (Standard Template Library). Esta lista permite almacenar punteros a objetos de distintas clases, tales como Reserva o Alojamiento, y es utilizada para representar colecciones dinámicas dentro del sistema.

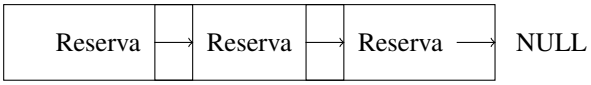
Una lista simplemente enlazada consiste en una secuencia de nodos, donde cada nodo contiene un puntero a un objeto y un puntero al siguiente nodo. Esta estructura permite una gestión dinámica de memoria eficiente, especialmente útil cuando la cantidad de elementos varía durante la ejecución del programa.

Cada objeto de tipo Usuario contiene un puntero a una lista enlazada. Si el usuario es anfitrion tendrá punteros a alojamientos y si es huesped tendrá punteros a reservaciones.

Cada objeto de tipo Alojamiento contiene un puntero a una lista enlazada de las reservas activas en el alojamiento.

La implementación de esta estructura permite realizar operaciones como inserción, eliminación y recorrido de manera iterativa, utilizando únicamente punteros crudos, lo que asegura un control explícito sobre la memoria.

A continuación, se ilustra gráficamente el concepto de una lista simplemente enlazada:



La principal desventaja es que en el peor caso buscar en una lista es O(n) con n el número de elementos, al igual que en una búsqueda. Las inserciones son O(1) si se agregan nodos

siempre al inicio. Esto no deberá ser un problema teniendo en cuenta que un usuario huésped no debería tener demasiados alojamientos o reservaciones asociadas a la vez. Mientras que un alojamiento no debería estar a la vez presente en muchas reservaciones.

Clases

El programa hará uso de 5 clases, a saber: Anfitrion, Huesped, Alojamiento, Reserva y Fecha. El diagrama de clases está incluido en la figura 1.

Clase Anfitrion

Representa a los administradores y responsables de los alojamientos. Se diferencian de los huéspedes únicamente en que pueden hacer históricos. 1 anfitrión es responsable de 1 o muchos alojamientos.

Clase Huesped

Son las personas que están registradas en la aplicación para reservar alojamientos. Pueden o no tener reservas activas. 1 huesped tiene 0 o muchas reservas.

Las clases **Anfitrion** y **Huesped** se distinguen por lo que pueden hacer (sus métodos) y por el tipo de lista que almacenan. Ambas podrían ser hijas de una clase Usuario pero para el reto no se puede usar herencia. Se especifica además que las contraseñas son alfanuméricas sin espacios.

Clase Alojamiento

Son los lugares que reservan los huéspedes y se asocian a éstos por medio de las reservaciones y son, además, administrados por los anfitriones. 1 alojamiento es administrado por un anfitrión.

No se usan constructores de copia pues se busca manipular todo el programa únicamente con punteros y de esta manera hacer el uso estrictamente necesario de la memoria.

Clase: Fecha

Esta clase permite definir fechas a partir de entradas numéricas o de una cadena de texto de la forma dd/mm/aaaa. Cada fecha está asociada a una reservación y, aunque a una fecha pueden estar asociadas varias reservaciones, se considera que con la primera relación es suficiente para lo que requiere el reto.

Los métodos y atributos de estas clases están descritos en el siguiente diagrama realizado sobre PlantUML. Los métodos públicos de las clases son aquellos con puntos verdes y los atributos y métodos privados con cuadros rojos.

Fig. 1. Diagrama UML.

III. Conclusiones

El uso de archivos de texto puede parecer simple pero es una decisión práctica que permite mantener un control claro sobre la información y facilita futuras ampliaciones del sistema.

La implementación propia de listas simplemente enlazadas y tablas hash demuestra que es posible desarrollar mecanismos eficientes para la gestión dinámica de datos sin depender de bibliotecas estándar. El uso del algoritmo djb2 como función hash y del encadenamiento para resolución de colisiones favorece el acceso en tiempo constante promedio.

El uso de estructuras dinámicas como listas enlazadas permite que el sistema pueda escalar con facilidad, adaptándose al crecimiento del número de alojamientos o reservas, sin comprometer la eficiencia ni el uso de memoria.

Si bien se reconoce que las listas enlazadas no son óptimas para búsquedas frecuentes, su uso en contextos donde el número de elementos es limitado (como las reservas por usuario o los alojamientos por anfitrión) permite mantener un equilibrio entre simplicidad estructural y eficiencia operativa. Esta decisión refleja una buena lectura de la relación entre el dominio del problema y las herramientas disponibles.