Projecto para: LICENCIATURA EN INGENIERÍA EN INFORMÁTICA Y SISTEMAS

Projecto 2

Sebastian Echeverria Flores 1138122

Estudiante de Universidad Rafael Landivar

Noviembre 2024

Projecto para LICENCIATURA EN INGENIERÍA EN INFORMÁTICA Y SISTEMAS

Projecto 2

Ingeniero: JOSEPH ABRAHAM SOTO GUTIÉRREZ

Projecto para LICENCIATURA EN INGENIERÍA EN INFORMÁTICA Y SISTEMAS

Sebastian Echeverria Flores 1138122

Noviembre 2024

Department of INGENIERÍA EN INFORMÁTICA Y SISTEMAS
Estudiante de Universidad Rafael Landivar

Table of Contents

Abstract

Projecto 2

Sebastian Echeverria

Departamento de Ingenieria.

Estudiante en la universidad

Rafael Landivar

Chapter 1

Introducción

El proyecto implica la implementación de un programa en C++ para administrar grandes cantidades de datos en memoria. Se describen tres funcionalidades principales: carga de datos desde archivos de texto, búsqueda por llave utilizando búsqueda binaria, y búsqueda por valor utilizando búsqueda secuencial.

Chapter 2

Objetivos

1. **Gestión Eficiente de Datos:** Desarrollar un programa en C++ que permita administrar grandes volúmenes de datos en memoria de manera eficiente y organizada.

2. **Carga y Almacenamiento de Datos:** Implementar una funcionalidad para cargar registros únicos desde archivos de texto, asegurando la integridad de los datos y su almacenamiento en una estructura adecuada.

3. **Ordenamiento de Datos:** Garantizar el ordenamiento ascendente de los registros utilizando una lista doblemente enlazada, basándose en una llave única generada mediante una función hash.

4. **Funcionalidades de Búsqueda:**

    - Proporcionar al usuario opciones de búsqueda por llave y valor.
    - Realizar búsquedas eficientes por llave utilizando el algoritmo de búsqueda binaria, ofreciendo resultados precisos y rápidos.
    - Implementar la búsqueda por valor, permitiendo localizar sets de datos que coincidan exactamente con el término buscado, con una optimización opcional mediante búsqueda binaria dentro de los sets ordenados.

5. **Interacción con el Usuario:** Proporcionar una interfaz de usuario amigable que permita al usuario cargar más datos desde archivos de texto y realizar operaciones de búsqueda de manera intuitiva.

6. **Optimización de Rendimiento:** Identificar áreas donde se puedan aplicar estrategias de optimización para mejorar la eficiencia en la búsqueda y manipulación de datos, como la implementación de algoritmos de búsqueda más rápidos o el uso de estructuras de datos adecuadas.

7. **Entregables Claros y Documentación:** Presentar una solución integral que incluya el código fuente del programa en C++, un programa ejecutable y un documento corto con un análisis claro de la complejidad temporal de los métodos de búsqueda implementados, asegurando una entrega comprensible y completa del proyecto.

Chapter 3

Metodologia

1. **Función Hash: Complejidad Temporal:** La complejidad temporal de una función hash depende de su implementación específica. En general, una buena función hash debe tener una complejidad cercana a O(1) para realizar la transformación de la llave primaria a una llave única de longitud constante.

2. **Carga de Datos: Complejidad Temporal:** Siendo la carga de datos desde un archivo de texto una operación que se ejecuta una vez al inicio del programa, su complejidad depende principalmente del tamaño de los datos en el archivo. Si se tienen "n" sets de datos, la complejidad sería O(n).

3. **Búsqueda por Llave (Búsqueda Binaria): Complejidad Temporal:** La búsqueda binaria tiene una complejidad temporal de O(log n), donde "n" representa la cantidad de sets de datos almacenados. Dado que la lista está ordenada, la búsqueda binaria puede encontrar la llave deseada de manera eficiente.

4. **Búsqueda por Valor (Búsqueda Secuencial con posible Optimización Binaria): Complejidad Temporal:** La búsqueda secuencial por valor en el peor de los casos tiene una complejidad de O(n*m), donde "n" es la cantidad de sets de datos y "m" es la cantidad de datos en cada set. Si se optimiza la búsqueda dentro del set de datos utilizando búsqueda binaria (suponiendo que los datos estén ordenados), se puede reducir la complejidad a O(n*log m).

5. **Complejidad Total del Programa:** La complejidad total del programa depende de la combinación de las operaciones realizadas. La carga de datos inicial es O(n). Las búsquedas por llave tienen una complejidad de O(log n), mientras que las búsquedas por valor pueden variar entre O(nm) *y* *O(*$n$log m) si se aplica la optimización de búsqueda binaria en los sets.

Chapter 4

Referencias

1. **Libros de Estructuras de Datos y Algoritmos en C++:**
   - "Estructuras de datos y algoritmos en C++" de Adam Drozdek.
   - "Algorithms in C++" de Robert Sedgewick.
2. **Documentación de C++:** Sitios web como cppreference.com y cplusplus.com proporcionan documentación detallada de la biblioteca estándar de C++ y su sintaxis.
3. **Referencias sobre Funciones Hash:**
   - "Introduction to Algorithms" de Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest y Clifford Stein.
   - Blogs y artículos especializados en algoritmos de hash y funciones hash en C++.
4. **Recursos sobre Complejidad Temporal:**
   - Cursos en línea sobre análisis de algoritmos que cubran la notación Big O y la complejidad temporal.
   - Libros de teoría de la computación que aborden el análisis de complejidad temporal.
5. **Foros y Comunidades en línea:** Plataformas como Stack Overflow, Reddit (por ejemplo, r/cpp) y otros foros de programación pueden ofrecer discusiones útiles y ejemplos de código para resolver problemas específicos en C++.

Chapter 5

Referencias

1. **Libros de Estructuras de Datos y Algoritmos en C++:**
   - "Estructuras de datos y algoritmos en C++" de Adam Drozdek.
   - "Algorithms in C++" de Robert Sedgewick.
2. **Documentación de C++:** Sitios web como cppreference.com y cplusplus.com proporcionan documentación detallada de la biblioteca estándar de C++ y su sintaxis.
3. **Referencias sobre Funciones Hash:**
   - "Introduction to Algorithms" de Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest y Clifford Stein.
   - Blogs y artículos especializados en algoritmos de hash y funciones hash en C++.
4. **Recursos sobre Complejidad Temporal:**
   - Cursos en línea sobre análisis de algoritmos que cubran la notación Big O y la complejidad temporal.
   - Libros de teoría de la computación que aborden el análisis de complejidad temporal.
5. **Foros y Comunidades en línea:** Plataformas como Stack Overflow, Reddit (por ejemplo, r/cpp) y otros foros de programación pueden ofrecer discusiones útiles y ejemplos de código para resolver problemas específicos en C++.