<!--Universidad Católica de Córdoba-->

Proyecto Final Algoritmos y Estructuras de Datos {

<Por="Chamaza Florencia, Rucci Lucia y Silvestrini Mia"/>

}

Contenidos

01 Introducción

02 Librerías

03 HashMapList.h y VectorVentas

04 Archivo CSV

05 Implementacion del menú

Introducción {

Desarrollamos un Sistema de Análisis de Ventas para una empresa de envíos en Sudamérica. El sistema procesa un archivo CSV y permite consultar, modificar y analizar las ventas de forma eficiente.Optamos por diseñar un sistema que sea eficiente tanto en tiempos de acceso como en el manejo de grandes volúmenes de datos. Por eso combinamos estructuras de acceso rápido (HashMapList) con estructuras de recorrido completo (vector), y utilizamos algoritmos de ordenamiento adecuados

Recursividad

Listas

Pilas

Colas

Árboles

Hash

Ordenamiento

Búsqueda

}

Librerías estándar de C++ {

`#include <iostream>` → entrada/salida de datos
`#include <fstream>` → manejo de archivos (lectura CSV)
`#include <sstream>` → manipulación de strings y parsing
`#include <string>` → manejo de cadenas
`#include <ctime>` → medición de tiempo
`#include <limits>` → validación de entrada de datos
`#include <vector>` → en el uso de vectorVentas

}

Librerías desarrolladas {

Estructuras de datos:

- Nodo.h → nodo genérico para listas
- Lista.h → lista enlazada simple
- HashEntry.h → entrada de hash
- HashMap.h → HashMap simple
- HashMapList.h → HashMap con colisiones por lista
- ArbolBinarioAVL.h → Almacenar datos ordenados

Algoritmos de ordenamiento:

- quickSort.h → algoritmo QuickSort

Archivos funcionales del proyecto:

- Ventas.h → definición de la estructura y manejo de datos de ventas
- MenuEstadisticas.h → funciones para cálculo y visualización de estadísticas
- MenuConsultas.h → funciones para consultas dinámicas
- MenuModificaciones.h → funciones para agregar, eliminar y modificar ventas

}

Librerías descartadas {

- Lista doble → No se usó ya que la lista simple fue suficiente para las operaciones requeridas (inserción, recorrido). No era necesario el doble enlace.
- Pila → En el caso de análisis de ventas no requerimos un comportamiento LIFO para los procesos principales.
- Cola → No se utilizó ya que el flujo de procesamiento no necesitaba un manejo secuencial FIFO. Las ventas no requerían ser procesadas en orden de llegada.
- ShellSort, InsertSort, BubbleSort → Se prefirió QuickSort, más eficiente (O(n log n)) para grandes volúmenes de datos como los que teníamos en ventas.

}

HashMapList.h y VectorVentas {

- HashMapList y vectorVentas son estructuras independientes.
- Cuando cargo las ventas del CSV, guardo cada venta en ambas estructuras: en HashMapList para tener acceso rápido por clave, y en vectorVentas para poder recorrer todas las ventas fácilmente.
- El HashMapList no recorre el vector, ni el vector recorre el HashMap — cada uno cumple un rol distinto.
- Uso vectorVentas para almacenar todas las ventas y poder recorrerlas fácilmente.
- Uso HashMapList para tener acceso rápido por clave (por ejemplo al modificar o eliminar una venta), y en muchas funciones uso HashMapList auxiliares para agrupar datos mientras recorro el vectorVentas.
- Son estructuras complementarias.

}

Lectura y procesamiento del archivo CSV {

- Archivo: ventas_sudamerica.csv
- Lectura línea por línea con ifstream y stringstream
- Al procesar cada línea:
  - Se carga la venta en vectorVentas
  - Se actualizan automáticamente las estadísticas en las estructuras de datos
- Se mide tiempo de ejecución y cantidad de IF utilizados

}

Implementación del menú {

Menú implementado en C++ con submenús:

- MenuEstadisticas.h
- MenuConsultas.h
- MenuModificaciones.h

Menú principal guía al usuario:

- Carga de datos
- Estadísticas de ventas
- Modificación de ventas
- Consultas dinámicas
- Salir

Textos explicativos en cada opción

Reprocesamiento automático tras cada modificación

}

CONCLUSION

El sistema implementado es eficiente y flexible, y permite analizar grandes volúmenes de ventas de manera rápida.
Las decisiones de diseño (HashMapList + vector + AVL + QuickSort) fueron pensadas para lograr un equilibrio entre rendimiento, claridad de código y facilidad de mantenimiento

<!--Universidad Católica de Córdoba-->

Gracias {

<Por="Chamaza Florencia, Rucci Lucia y Silvestrini Mia"/>

}