**Héctor Gutierrez**

**Camila Rodriguez**

**Julian Espinoza**

**Cesar Silva**

**Act 1.3 – Actividad Integral de Conceptos Básicos y Algoritmos Fundamentales**

**Código:**

//SituacionP.h

#pragma once

#include <vector>

#include <string>

class Registro {

public:

std::string fecha;

std::string hora;

char punto\_entrada;

std::string ubi;

// Constructor predeterminado sin argumentos

Registro() : fecha(""), hora(""), punto\_entrada('\0'), ubi("") {}

Registro(const std::string& f, const std::string& h, char p, const std::string& u);

//staic es para proporcionar una definición de la función que no está vinculada a un objeto en particular

// y se puede llamar sin crear un objeto.

static void ordenaMerge(std::vector<Registro>& registros, int inicio, int fin);

static void merge(std::vector<Registro>& registros, int inicio, int mitad, int fin);

static int busquedaBinaria(const std::vector<Registro>& registros, const std::string& serieABuscar);

};

// SituacionP.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "SituacionP.h"

#include <vector>

#include <iostream>

#include <string>

using namespace std;

// Esta línea de código es parte de la implementación del constructor de la clase Registro y

// se utiliza para configurar los valores iniciales de los miembros de datos cuando se crea un nuevo objeto Registro.

Registro::Registro(const std::string& f, const std::string& h, char p, const std::string& u)

: fecha(f), hora(h), punto\_entrada(p), ubi(u) {}

void Registro::ordenaMerge(std::vector<Registro>& registros, int inicio, int fin) {

if (inicio < fin) {

int mitad = (inicio + fin) / 2;

ordenaMerge(registros, inicio, mitad); // "Conquer"

ordenaMerge(registros, mitad + 1, fin); // "Conquer"

merge(registros, inicio, mitad, fin); // "Combine"

}

}

void Registro::merge(std::vector<Registro>& registros, int inicio, int mitad, int fin) {

int n1 = mitad - inicio + 1;

int n2 = fin - mitad;

vector<Registro> L(n1);

vector<Registro> R(n2);

// Aqui se copian los datos a los vectores temporales L y R

for (int i = 0; i < n1; i++) {

L[i] = registros[inicio + i];

}

for (int i = 0; i < n2; i++) {

R[i] = registros[mitad + 1 + i];

}

// Aqui ya se ordenan los datos

int i = 0, j = 0, k = inicio;

while (i < n1 && j < n2) {

// Primero, compara por UBI

//L[i] puede utilizar . para acceder a los miembros de datos de la estructura Registro

// porque L[i] es un objeto Registro

if (L[i].ubi < R[j].ubi) { // Si la UBI de L es menor que la UBI de R

registros[k] = L[i];

i++;

}

else if (L[i].ubi > R[j].ubi) { // Si la UBI de L es mayor que la UBI de R

registros[k] = R[j];

j++;

}

else { // Si las UBI son iguales, compara por fecha en formato día/mes/año

std::string fechaL = L[i].fecha;

std::string fechaR = R[j].fecha;

int diaL, mesL, anioL, diaR, mesR, anioR;

// sscanf() es una función de la biblioteca estándar de C que se utiliza para leer datos de una cadena de caracteres.

// fechaL.c\_str() convierte el objeto string fechaL a una cadena de caracteres de C.

sscanf(fechaL.c\_str(), "%d/%d/%d", &diaL, &mesL, &anioL);

sscanf(fechaR.c\_str(), "%d/%d/%d", &diaR, &mesR, &anioR);

if (anioL < anioR || (anioL == anioR && mesL < mesR) || (anioL == anioR && mesL == mesR && diaL < diaR)) {

registros[k] = L[i];

i++;

}

else {

registros[k] = R[j];

j++;

}

}

k++;

}

while (i < n1) { // Aqui se copian los elementos restantes de L

registros[k] = L[i];

i++;

k++;

}

while (j < n2) { // Aqui se copian los elementos restantes de R

registros[k] = R[j];

j++;

k++;

}

}

int Registro::busquedaBinaria(const std::vector<Registro>& registros, const std::string& serieABuscar) {

int bajo = 0;

int alto = registros.size() - 1;

vector<int> resultados;

while (bajo <= alto) {

int central = (bajo + alto) / 2;

const std::string& ubiCentral = registros[central].ubi; // Aqui se toma una UBI del registro que se encuentra en la posición central

if (ubiCentral < serieABuscar) {

bajo = central + 1;

}

else {

alto = central - 1;

}

}

// Encontrar todas las entradas que coinciden

int indice = bajo;

while (indice < registros.size()) {

const std::string& ubi = registros[indice].ubi;

// Aqui aunque se inserte 1 elemento lo va comparar igual con el tamaño de la serie a buscar

if (ubi.substr(0, serieABuscar.size()) == serieABuscar) {

resultados.push\_back(indice);

indice++;

}

else {

break;

}

}

// Imprimir las entradas encontradas

if (!resultados.empty()) {

cout << "Entradas encontradas para la serie " << serieABuscar << ":" << endl;

for (int resultado : resultados) {

cout << registros[resultado].ubi << " " << registros[resultado].fecha << " " << registros[resultado].hora << " " << registros[resultado].punto\_entrada << endl;

}

}

else {

cout << "No se encontraron entradas para la serie " << serieABuscar << "." << endl;

}

return -1;

}

//main.cpp

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include "SituacionP.h"

using namespace std;

int main() {

// Leer el nombre del archivo

string nombreArchivo;

cout << "Ingrese el nombre del archivo de entrada: ";

cin >> nombreArchivo;

ifstream archivo(nombreArchivo);

if (!archivo) {

cerr << "No se pudo abrir el archivo." << endl;

return 1;

}

vector<Registro> registros;

string fecha, hora, ubi;

char puntoEntrada;

// Aqui se lee el archivo y se guardan los registros en el vector

// cada uno con su fecha, hora, punto de entrada y UBI

while (archivo >> fecha >> hora >> puntoEntrada >> ubi) {

Registro registro(fecha, hora, puntoEntrada, ubi);

registros.push\_back(registro);

}

archivo.close();

//-----------------------------------------------------------

// Ordenar los registros utilizando Merge Sort

Registro::ordenaMerge(registros, 0, registros.size() - 1);

// Mostrar los registros ordenados por UBI + Fecha

cout << "Registros ordenados por UBI + Fecha:" << endl;

for (const Registro& reg : registros) {

cout << reg.ubi << " " << reg.fecha << endl;

}

// Solicitar al usuario la serie a buscar (los primeros tres caracteres del UBI)

cout << endl;

string serieABuscar;

cout << "Ingrese los primeros tres caracteres de la serie a buscar: ";

cin >> serieABuscar;

Registro registro;

registro.busquedaBinaria(registros, serieABuscar);

return 0;

}

**Casos de prueba:**

|  |  |  |
| --- | --- | --- |
|  |  | **Casos de Prueba** |
| Input |  | Output |
| Cantidad | busquedadBinaria | ordenaMerge |
| bitacora.txt  4TL | 4TLB0 25/08/202305:18 R  4TLB9 15/02/2023 09:04 R  4TLC2 12/12/2023 07:54 M  4TLD8 07/04/2023 16:46 R  4TLE0 26/07/2023 20:20 M  4TLE1 03/11/2023 19:48 M  4TLE6 21/07/2023 18:18 M  4TLE9 12/07/2023 13:26 R  4TLF1 11/07/2023 21:03 M  4TLF4 20/03/2023 02:14 R  ………. | 0AEA1 17/09/2023  0AEA2 02/10/2023  0AEA3 06/06/2023  0AEB2 28/03/2023  0AEB7 02/05/2023  0AEC0 15/06/2023  0AEC2 23/05/2023  0AEE1 13/07/2023  0AEE2 11/04/2023  …………. |
| bitacora.txt  ds | 4TLB0 25/08/2023 05:18 R  4TLB9 15/02/2023 09:04 R  4TLC2 12/12/2023 07:54 M  4TLD8 07/04/2023 16:46 R  4TLE0 26/07/2023 20:20 M  4TLE1 03/11/2023 19:48 M  4TLE6 21/07/2023 18:18 M  4TLE9 12/07/2023 13:26 R  ………. | No se encontraron entradas para la serie ds. |
| bitacora.ta  ds | No se pudo abrir el archivo. | No se pudo abrir el archivo. |
| bitacora.txt  1 | 4TLB0 25/08/2023 05:18 R  4TLB9 15/02/2023 09:04 R  4TLC2 12/12/2023 07:54 M  4TLD8 07/04/2023 16:46 R  4TLE0 26/07/2023 20:20 M  4TLE1 03/11/2023 19:48 M  4TLE6 21/07/2023 18:18 M  4TLE9 12/07/2023 13:26 R  4TLF1 11/07/2023 21:03 M  4TLF4 20/03/2023 02:14 R  ………. | 1AEA2 03/05/2023 01:56 R  1AEA4 26/11/2023 15:56 M  1AEA5 03/05/2023 15:48 R  1AEA7 12/03/2023 07:57 M  1AEB1 02/06/2023 16:46 M  1AEB6 04/07/2023 19:04 M  1AEB9 18/05/2023 02:02 R  1AEC2 20/04/2023 02:47 M  1AEC3 04/06/2023 21:14 M  1AEC9 14/01/2023 22:07 R  ………… |