**Héctor Gutierrez**

**Camila Rodriguez**

**Julian Espinoza**

**Cesar Silva**

**Act 2.3 - Actividad Integral estructura de datos lineales (Evidencia Competencia)**

**Código:**

//SituacionProblema.h

#pragma once

#include <string>

#include <map>

#include <iostream>

class ShipRecord {

public:

std::string fecha;

char punto\_entrada;

std::string ubi;

ShipRecord\* next;

ShipRecord(const std::string& \_fecha, char \_punto\_entrada, const std::string& \_ubi)

: fecha(\_fecha), punto\_entrada(\_punto\_entrada), ubi(\_ubi), next(nullptr) {}

};

class ShipRecordList {

public:

ShipRecord\* head;

ShipRecord\* tail;

ShipRecordList() : head(nullptr), tail(nullptr) {}

void append(const std::string& fecha, char punto\_entrada, const std::string& ubi) {

ShipRecord\* newNode = new ShipRecord(fecha, punto\_entrada, ubi);

if (head == nullptr) {

head = tail = newNode;

}

else {

tail->next = newNode;

tail = newNode;

}

}

};

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

struct Nodo {

std::string fecha;

std::string hora;

int valor1;

int valor2;

Nodo\* siguiente;

Nodo(const std::string& fecha, int valor1, int valor2)

: fecha(fecha), valor1(valor1), valor2(valor2), siguiente(nullptr) {}

};

class ListaEnlazada {

private:

Nodo\* cabeza;

public:

ListaEnlazada() : cabeza(nullptr) {}

void agregar(const std::string& fecha, int valor1, int valor2) {

Nodo\* nuevoNodo = new Nodo(fecha, valor1, valor2);

nuevoNodo->siguiente = cabeza;

cabeza = nuevoNodo;

}

void imprimirCantidadEntradasPorMes(); // Declaración de la función

~ListaEnlazada() {

while (cabeza) {

Nodo\* temp = cabeza;

cabeza = cabeza->siguiente;

delete temp;

}

}

};

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

bool compareRecords(const ShipRecord& a, const ShipRecord& b);

void processRecords(ShipRecord\* head, const std::string& serieABuscar, std::map<std::string, std::pair<int, int>>& mesPorMes);

void deleteList(ShipRecord\*& head);

ShipRecord\* mergeSort(ShipRecord\* head);

ShipRecordList sortList(ShipRecordList& list);

bool binarySearch(ShipRecord\* head, const std::string& targetSerie);

//SituacionProblema.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "SituacionProblema.h"

#include <iostream>

#include <fstream>

#include <string>

#include <sstream>

//compareEecords se utiliza para comparar dos registros de buques y ordenarlos por fecha y ubicacion en orden ascendente

bool compareRecords(const ShipRecord& a, const ShipRecord& b) {

int yearA, monthA, dayA;

int yearB, monthB, dayB;

sscanf(a.fecha.c\_str(), "%d/%d/%d", &dayA, &monthA, &yearA);

sscanf(b.fecha.c\_str(), "%d/%d/%d", &dayB, &monthB, &yearB);

if (a.ubi == b.ubi) {

if (yearA != yearB) {

return yearA < yearB;

}

else if (monthA != monthB) {

return monthA < monthB;

}

else {

return dayA < dayB;

}

}

else {

return a.ubi < b.ubi;

}

}

//map es para realizar un seguimiento de la cantidad de entradas de buques por mes en funcion de una fecha dada

// La clave del mapa es la fecha en formato "MM / AAAA" y el valor es un par de enteros que representan la cantidad de entradas de buques en el mediterraneo y el mar rojo respectivamente

//La funcion processRecors es para procesar los registros de buques y actualizar el mapa con la cantidad de entradas de buques por mes

// tambien sirve para verificar si la serie a buscar existe en la lista de registros de buques

void processRecords(ShipRecord\* head, const std::string& serieABuscar, std::map<std::string, std::pair<int, int>>& mesPorMes) {

ShipRecord\* current = head;

while (current != nullptr) {

if (current->ubi.substr(0, 3) == serieABuscar) {

std::string fechaCompleta = current->fecha;

if (mesPorMes.find(fechaCompleta) == mesPorMes.end()) {

mesPorMes[fechaCompleta] = std::make\_pair(0, 0);

}

if (current->punto\_entrada == 'M') {

mesPorMes[fechaCompleta].first++;

}

else if (current->punto\_entrada == 'R') {

mesPorMes[fechaCompleta].second++;

}

}

current = current->next;

}

}

void deleteList(ShipRecord\*& head) {

while (head != nullptr) {

ShipRecord\* temp = head;

head = head->next;

delete temp;

}

}

ShipRecord\* merge(ShipRecord\* left, ShipRecord\* right) {

ShipRecord\* result = nullptr;

if (left == nullptr) {

return right;

}

if (right == nullptr) {

return left;

}

if (compareRecords(\*left, \*right)) {

result = left;

result->next = merge(left->next, right);

}

else {

result = right;

result->next = merge(left, right->next);

}

return result;

}

ShipRecord\* getMiddle(ShipRecord\* head) {

if (head == nullptr) {

return head;

}

ShipRecord\* slow = head;

ShipRecord\* fast = head;

while (fast->next != nullptr && fast->next->next != nullptr) {

slow = slow->next;

fast = fast->next->next;

}

return slow;

}

ShipRecord\* mergeSort(ShipRecord\* head) {

if (head == nullptr || head->next == nullptr) {

return head;

}

ShipRecord\* middle = getMiddle(head);

ShipRecord\* left = head;

ShipRecord\* right = middle->next;

middle->next = nullptr;

left = mergeSort(left);

right = mergeSort(right);

return merge(left, right);

}

//La funcion sortList tiene una complejidad computacional de O (n log n) ya que utiliza el algoritmo de ordenacion de mezcla para ordenar la lista.

ShipRecordList sortList(ShipRecordList& list) {

ShipRecordList result;

result.head = mergeSort(list.head);

result.tail = list.head;

while (result.tail != nullptr && result.tail->next != nullptr) {

result.tail = result.tail->next;

}

return result;

}

//La funcion binarySearch tiene una complejidad computacional de O (log n) ya que la lista esta ordenada y se divide a la mitad en cada iteracion de la funcion.

bool binarySearch(ShipRecord\* head, const std::string& targetSerie) {

ShipRecord\* left = head;

ShipRecord\* right = nullptr;

while (left != right) {

ShipRecord\* middle = left;

int count = 0;

// Avanzar 'middle' a la mitad de la lista

while (middle != right) {

middle = middle->next;

count++;

}

// Avanzar 'middle' a la mitad de la lista nuevamente

middle = left;

for (int i = 0; i < count / 2; i++) {

middle = middle->next;

}

// Verificar si 'middle' es el elemento deseado

if (middle->ubi.substr(0, 3) == targetSerie) {

return true;

}

// Actualizar 'left' y 'right' según la comparación

if (middle->ubi.substr(0, 3) < targetSerie) {

left = middle->next;

}

else {

right = middle;

}

}

// Verificar el último elemento (cuando la lista tiene un solo elemento)

if (left && left->ubi.substr(0, 3) == targetSerie) {

return true;

}

return false;

}

// La funcion imprimirCantidadEntradasPorMes tiene una complejidad computacional de O (n) ya que recorre la lista una vez para extraer el mes y el año de cada fecha y luego recorre el mapa una vez para imprimir la cantidad de entradas de buques por mes.

void ListaEnlazada::imprimirCantidadEntradasPorMes() {

// Implementación de la función imprimirCantidadEntradasPorMes

std::map<std::string, std::pair<int, int>> cantidadPorMes;

Nodo\* actual = cabeza;

while (actual) {

// Extraer mes y año de la fecha

std::string fecha = actual->fecha;

std::string mes = fecha.substr(3, 2);

std::string anio = fecha.substr(6, 4);

// Incrementar la cantidad de entrada de buques de esa serie en ese mes

cantidadPorMes[mes + " " + anio].first += actual->valor1;

cantidadPorMes[mes + " " + anio].second += actual->valor2;

actual = actual->siguiente;

}

// Imprimir la cantidad de entrada de buques por mes

std::cout << "Cantidad de Entradas de Buques por Mes:" << std::endl;

std::cout << "M A CM CR" << std::endl;

for (const auto& entry : cantidadPorMes) {

std::istringstream ss(entry.first);

std::string month, year;

ss >> month >> year;

if (month == "01") {

std::cout << "Jan ";

}

else if (month == "02") {

std::cout << "Feb ";

}

else if (month == "03") {

std::cout << "Mar ";

}

else if (month == "04") {

std::cout << "Apr ";

}

else if (month == "05") {

std::cout << "May ";

}

else if (month == "06") {

std::cout << "Jun ";

}

else if (month == "07") {

std::cout << "Jul ";

}

else if (month == "08") {

std::cout << "Aug ";

}

else if (month == "09") {

std::cout << "Sep ";

}

else if (month == "10") {

std::cout << "Oct ";

}

else if (month == "11") {

std::cout << "Nov ";

}

else {

std::cout << "Dec ";

}

//std::cout << entry.first << " " << std::endl;

std::cout << year << " " << entry.second.first << " " << entry.second.second << std::endl;

}

}

// main.cpp

#include <iostream>

#include <fstream>

#include <string>

#include <sstream>

#include <map>

#include "SituacionProblema.h"

using namespace std;

int main() {

string archivoEntrada;

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

cin >> archivoEntrada;

ifstream archivo(archivoEntrada);

if (!archivo) {

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

return 1;

}

string linea2;

// Intentar leer una línea del archivo

if (!getline(archivo, linea2) || linea2.empty()) {

cerr << "El archivo esta vacio." << endl;

return 1;

}

ShipRecordList mediterraneoList;

ShipRecordList marRojoList;

string fecha, hora, linea;

while (getline(archivo, linea)) {

istringstream ss(linea);

ss >> fecha >> hora;

char puntoEntrada;

ss >> puntoEntrada;

string ubi;

ss >> ubi;

string recordFecha = fecha + " " + hora;

if (puntoEntrada == 'M') {

mediterraneoList.append(recordFecha, puntoEntrada, ubi);

}

else if (puntoEntrada == 'R') {

marRojoList.append(recordFecha, puntoEntrada, ubi);

}

}

mediterraneoList = sortList(mediterraneoList);

marRojoList = sortList(marRojoList);

// Imprimir las listas ordenadas

cout << "---------------------------------" << endl;

cout << "Lista ordenada del mediterraneo:" << endl;

cout << "---------------------------------" << endl;

ShipRecord\* current = mediterraneoList.head;

while (current != nullptr) {

cout << current->ubi << " " << current->fecha << " " << current->punto\_entrada << endl;

current = current->next;

}

cout << endl;

cout << "---------------------------------" << endl;

cout << "Lista ordenada del mar rojo:" << endl;

cout << "---------------------------------" << endl;

current = marRojoList.head;

while (current != nullptr) {

cout << current->ubi << " " << current->fecha << " " << current->punto\_entrada << endl;

current = current->next;

}

string serieABuscar;

cout << "Ingrese la serie a buscar (primeros tres caracteres del UBI): ";

cin >> serieABuscar;

map<string, pair<int, int>> mesPorMes;

// Verificar si la serie a buscar existe usando búsqueda binaria

bool foundInMediterraneo = binarySearch(mediterraneoList.head, serieABuscar);

bool foundInMarRojo = binarySearch(marRojoList.head, serieABuscar);

if (foundInMediterraneo || foundInMarRojo) {

// Si se encontró la serie, procesar los registros

processRecords(mediterraneoList.head, serieABuscar, mesPorMes);

processRecords(marRojoList.head, serieABuscar, mesPorMes);

ListaEnlazada listaMesPorMes;

// Agregar los elementos de mesPorMes a listaMesPorMes

for (auto it = mesPorMes.begin(); it != mesPorMes.end(); it++) {

// Obtener la fecha eliminando la hora (suponiendo que la fecha tiene un formato fijo de 10 caracteres)

string fecha = it->first.substr(0, 10);

listaMesPorMes.agregar(fecha, it->second.first, it->second.second);

}

listaMesPorMes.imprimirCantidadEntradasPorMes();

// Liberar la memoria de listaMesPorMes

listaMesPorMes.~ListaEnlazada();

}

else {

cout << "La serie especificada -> " << serieABuscar << " no se encontro en los registros." << endl;

}

// Liberar la memoria de las listas

deleteList(mediterraneoList.head);

deleteList(marRojoList.head);

return 0;

}

**Casos de prueba:**

**Casos de prueba**

Nombre de archivo de entrada: “bitacora2.txt”

**Caso prueba 1**

**\*archivo completo\***

|  |  |
| --- | --- |
| **Input** | **Ingrese el nombre del archivo de entrada:** bitacora2 |
| **Output** | No se pudo abrir el archivo. |

**Caso prueba 2**

**\*archivo completo\***

|  |  |
| --- | --- |
| **Input** | **Ingrese el nombre del archivo de entrada:** bitacora2.txt  **Ingrese la serie a buscar (primeros tres caracteres del UBI):** 4TL |
| **Output** | ---------------------------------  Lista ordenada del mediterraneo:  ---------------------------------  0AEA2 02/10/2023 02:51 M  0AEA3 06/06/2023 03:50 M  0AEC0 15/06/2023 04:21 M  0AEC2 23/05/2023 21:46 M  0AEE1 13/07/2023 10:29 M  0AEE5 18/04/2023 21:24 M  0AEE9 23/08/2023 04:18 M  0AEJ0 29/06/2023 07:42 M  0AEL3 14/05/2023 09:57 M  0AEL6 22/09/2023 12:31 M  0AEM0 19/08/2023 17:51 M  0AEM2 06/06/2023 02:49 M  0AEN2 26/11/2023 21:43 M  0AEQ9 09/09/2023 12:47 M  0AER1 02/04/2023 02:02 M  0AER8 11/09/2023 16:17 M  0AEV9 20/10/2023 17:35 M  0AEW5 20/02/2023 06:14 M  …….  ---------------------------------  Lista ordenada del mar rojo:  ---------------------------------  0AEA1 17/09/2023 22:23 R  0AEB2 28/03/2023 12:04 R  0AEB7 02/05/2023 02:18 R  0AEE2 11/04/2023 02:08 R  0AEE6 31/03/2023 06:55 R  0AEE9 30/08/2023 11:34 R  0AEG6 16/10/2023 02:12 R  0AEG8 17/01/2023 11:50 R  0AEH3 26/05/2023 14:22 R  0AEH7 06/09/2023 00:45 R  0AEI2 24/10/2023 12:20 R  0AEN8 30/11/2023 05:40 R  0AEO8 05/03/2023 00:37 R  0AEO9 21/10/2023 02:44 R  0AES9 07/04/2023 19:27 R  0AEU0 30/03/2023 16:51 R  0AEX5 06/01/2023 05:05 R  0AEX6 11/08/2023 14:01 R  0AEY7 09/02/2023 06:44 R  ……..  **Ingrese la serie a buscar (primeros tres caracteres del UBI):** 4TL  Cantidad de Entradas de Buques por Mes:  M         A       CM   CR  Jan    2023     1      0  Feb    2023    3       3  Mar    2023    0       1  Apr    2023     1       8  Jun    2023     0      2  Jul     2023     6      3  Aug   2023     2      1  Oct    2023     0      4  Nov   2023     1      2  Dec   2023     1      0 |

**Caso prueba 3**

**\*archivo completo\***

|  |  |
| --- | --- |
| **Input** | **Ingrese el nombre del archivo de entrada:** bitacora2.txt  **Ingrese la serie a buscar (primeros tres caracteres del UBI):** 000 |
| **Output** | Ingrese el nombre del archivo de entrada: bitacora2.txt  Ingrese la serie a buscar (primeros tres caracteres del UBI): 4TL  ---------------------------------  Lista ordenada del mediterraneo:  ---------------------------------  0AEA2 02/10/2023 02:51 M  0AEA3 06/06/2023 03:50 M  0AEC0 15/06/2023 04:21 M  0AEC2 23/05/2023 21:46 M  0AEE1 13/07/2023 10:29 M  0AEE5 18/04/2023 21:24 M  0AEE9 23/08/2023 04:18 M  0AEJ0 29/06/2023 07:42 M  0AEL3 14/05/2023 09:57 M  0AEL6 22/09/2023 12:31 M  0AEM0 19/08/2023 17:51 M  0AEM2 06/06/2023 02:49 M  0AEN2 26/11/2023 21:43 M  0AEQ9 09/09/2023 12:47 M  0AER1 02/04/2023 02:02 M  0AER8 11/09/2023 16:17 M  0AEV9 20/10/2023 17:35 M  0AEW5 20/02/2023 06:14 M  …….  ---------------------------------  Lista ordenada del mar rojo:  ---------------------------------  0AEA1 17/09/2023 22:23 R  0AEB2 28/03/2023 12:04 R  0AEB7 02/05/2023 02:18 R  0AEE2 11/04/2023 02:08 R  0AEE6 31/03/2023 06:55 R  0AEE9 30/08/2023 11:34 R  0AEG6 16/10/2023 02:12 R  0AEG8 17/01/2023 11:50 R  0AEH3 26/05/2023 14:22 R  0AEH7 06/09/2023 00:45 R  0AEI2 24/10/2023 12:20 R  0AEN8 30/11/2023 05:40 R  0AEO8 05/03/2023 00:37 R  0AEO9 21/10/2023 02:44 R  0AES9 07/04/2023 19:27 R  0AEU0 30/03/2023 16:51 R  0AEX5 06/01/2023 05:05 R  0AEX6 11/08/2023 14:01 R  0AEY7 09/02/2023 06:44 R  ……..  Ingrese la serie a buscar (primeros tres caracteres del UBI): 000  La serie especificada -> 000 no se encontro en los registros. |

**Caso prueba 4**

**\*archivo vacío\***

|  |  |
| --- | --- |
| **Input** | **Ingrese el nombre del archivo de entrada:** bitacora2.txt |
| **Output** | El archivo esta vacio. |