MEGATRON 3000

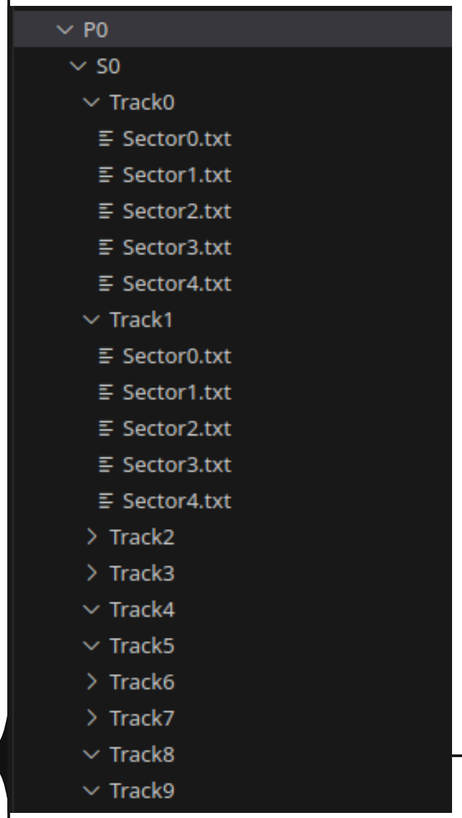
Autor: Arthur Patrick Meza Pareja

ÍNDICE

de contenido

ESTRUCTURA

```
Disco(int nPlatos, int nSuperficies, int nPistas, int nSectores, int capSector, const string& nombre)
    : numPlatos(nPlatos), numSuperficiesPorPlato(nSuperficies), numPistasPorSuperficie(nPistas),
      numSectoresPorPista(nSectores), capacidadSectorBytes(capSector), nombreDisco(nombre),
      lastPlatoWritten(0), lastSuperficieWritten(0), lastPistaWritten(0), lastSectorWritten(0) {
    rutaBaseDisco = "./" + nombreDisco + "_disk";
    MKDIR(rutaBaseDisco.c_str()); // Crear directorio base del disco
```




```
CONFIG#2#5#10#5#512#disco
```

ESTRUCTURA

```
bool cargarConfiguracionYDicc() {
    string tempRutaDiccionario = "Disco/Plato0/Superficie0/Pista0/Sector1.txt"; // Ruta fija para el diccionario
    ifstream tempDiccionarioFile(tempRutaDiccionario);
    if (!tempDiccionarioFile.is_open()) {
        cerr << "Error: No se pudo abrir el archivo del diccionario de datos: " << tempRutaDiccionario << endl;
        return false;
    }
```

Nos ayudara para cargar el diccionario de datos, a la RAM
cuando se cierra el programa .

CARGAR DATASET

COMO SE CARGA

Lo hace de forma de un cilindro
, dato por dato

```
// Bucle principal para recorrer los platos
for (int p = 0; p < numPlatos; ++p) {
    int current_plato = (startPlato + p) % numPlatos;
    for (int t = 0; t < numPistasPorSuperficie; ++t) {
        int current_pista = (startPista + t) % numPistasPorSuperficie;
        for (int s = 0; s < numSuperficiesPorPlato; ++s) {
            // Asegurarse de que la iteración comience desde la superficie correcta
            int current_superficie = (startSuperficie + s) % numSuperficiesPorPlato;
            int actual_start_sector = (t == 0 && s == 0) ? startSector : 0;


            Pista* pistaObj = platos[current_plato]->getSuperficie(current_superficie)->getPista(current_pista);
            if (pistaObj == nullptr) continue;

            for (int sec = 0; sec < numSectoresPorPista; ++sec) {
                int current_sector = (actual_start_sector + sec) % numSectoresPorPista;
```

ESQUEMA Y DICCIONARIO

Para el esquema se teien la funcion cargarCSV()

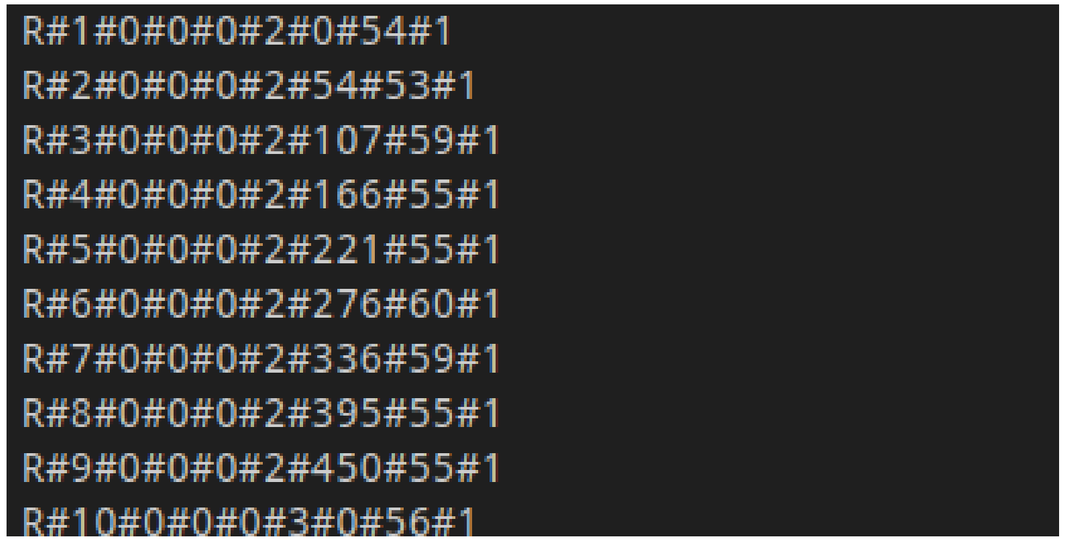
```
string linea;
// La primera línea es el esquema
getline(ssCSV, linea);
if (linea.empty()) {
    cerr << "El archivo CSV no tiene esquema." << endl;
    return;
}

// Almacenar el esquema en Sector0.txt
string rutaSector0 = rutaBaseDisco + "/P0/S0/Track0/Sector0.txt";
Sector sector0(rutaSector0, capacidadSectorBytes);
string esquemaConPrefijo = "R1#" + linea + "\n";
sector0.escribir(esquemaConPrefijo, true); // Sobreescribir esquema
tablaEsquema = linea; // Actualizar esquema en RAM

cout << "Esquema cargado: " << tablaEsquema << endl;
```

```
struct RecordMetadata {
    long idRegistro;
    int platoIdx;
    int superficieIdx;
    int pistaIdx;
    int sectorGlobalEnPista;
    long offset;
    int tamRegistro;
    bool ocupado;
};
```

DICCIONARIO

Disco::diccionarioDeDatosEnRAM (Miembro de la clase Disco)
Propósito: Un contenedor en memoria (RAM) que almacena todas las instancias de RecordMetadata. Es el "índice principal" del sistema mientras está en ejecución.

Disco::persistirDiccionario()
Propósito: La función encargada de guardar el estado actual del diccionarioDeDatosEnRAM desde la memoria volátil hacia el disco persistente.



CILINDRO

Ayuda que el cabezal no se mueva muco

```
tuple<int, int, int, int, long> encontrarEspacioCilindrico(int tamanoRequerido) {
    // Intentar continuar desde la última posición escrita para locality
    int startPlato = lastPlatoWritten;
    int startPista = lastPistaWritten;
    int startSuperficie = lastSuperficieWritten;
    int startSector = lastSectorWritten;

    // Bucle principal para recorrer los platos
    for (int p = 0; p < numPlatos; ++p) {
        int current_plato = (startPlato + p) % numPlatos;
        for (int t = 0; t < numPistasPorSuperficie; ++t) {
            int current_pista = (startPista + t) % numPistasPorSuperficie;
            for (int s = 0; s < numSuperficiesPorPlato; ++s) {
                // Asegurarse de que la iteración comience desde la superficie correcta
                int current_superficie = (startSuperficie + s) % numSuperficiesPorPlato;
                int actual_start_sector = (t == 0 && s == 0) ? startSector : 0;
```

REGISTRO-AGREGAR

```
void insertarRegistro(const string& datosRegistro) {
    // Calcular el tamaño del registro (datos + delimitador de nueva línea)
    int tamanoRequerido = datosRegistro.length() + 1; // +1 para el '\n'

    // Encontrar espacio en el disco utilizando la lógica cilíndrica
    auto [platoIdx, superficieIdx, pistaIdx, sectorGlobalEnPista, offset] = encontrarEspacioCilindrico(tamanoRequerido);

    if (platoIdx == -1) {
        cout << "No hay espacio suficiente en el disco para el registro: " << datosRegistro << endl;
        return;
    }

    // Obtener el sector donde se escribirá el registro
    Sector* sectorAEscribir = platos[platoIdx]
                              ->getSuperficie(superficieIdx)
                              ->getPista(pistaIdx)
                              ->getSector(sectorGlobalEnPista);
```

```
void eliminarRegistro(long id) {
    bool encontrado = false;
    for (auto& rm : diccionarioDeDatosEnRAM) {
        if (rm.idRegistro == id) {
            if (rm.ocupado) {
                rm.ocupado = false; // Marcamos como no ocupado
                encontrado = true;
                cout << "Registro ID " << id << " marcado como eliminado (lógicamente)." << endl;
            } else {
                cout << "Registro ID " << id << " ya está eliminado." << endl;
            }
            break;
        }
    }
    if (!encontrado) {
        cout << "Registro ID " << id << " no encontrado." << endl;
    }
    persistirDiccionario(); // Persistir el cambio
}
```

REGISTRO-ELIMINAR

```cpp
void eliminarRegistro(long id) {
    bool encontrado = false;
    for (auto& rm : diccionarioDeDatosEnRAM) {
        if (rm.idRegistro == id) {
            if (rm.ocupado) {
                rm.ocupado = false; // Marcamos como no ocupado
                encontrado = true;
                cout << "Registro ID " << id << " marcado como eliminado (lógicamente)." << endl;
            } else {
                cout << "Registro ID " << id << " ya está eliminado." << endl;
            }
            break;
        }
    }
    if (!encontrado) {
        cout << "Registro ID " << id << " no encontrado." << endl;
    }
    persistirDiccionario(); // Persistir el cambio
}
```

MAPA DE BITS

```
VOID MOSTRARMAPADEBITS() {
COUT << "\N--- MAPA DE ASIGNACIÓN DE SECTORES ---
\N";
FOR (INT P = 0; P < NUMPLATOS; ++P) {
COUT << "PLATO " << P << ":\N";
FOR (INT S = 0; S < NUMSUPERFICIESPORPLATO; ++S) {
COUT << " SUPERFICIE " << S << ":\N";
FOR (INT T = 0; T < NUMPISTASPORSUPERFICIE; ++T) {
COUT << " PISTA " << T << ": ";
PISTA* PISTAOBJ = PLATOS[P]->GETSUPERFICIE(S)-
>GETPISTA(T);
IF (PISTAOBJ) {
FOR (INT SEC = 0; SEC < NUMSECTORESPORPISTA;
++SEC) {
IF (ISRESERVEDSECTOR(P, S, T, SEC)) {
COUT << "R"; // SECTOR RESERVADO
CONTINUE;
}
```

```
--- Mapa de Asignación de Sectores ---
Plato 0:
  Superficie 0:
    Pista 0: RROOO
    Pista 1: OOOOO
    Pista 2: OOOLL
    Pista 3: LLLLL
    Pista 4: LLLLL
    Pista 5: LLLLL
    Pista 6: LLLLL
    Pista 7: LLLLL
    Pista 8: LLLLL
    Pista 9: LLLLL
  Superficie 1:
    Pista 0: OOOOO
    Pista 1: OOOOO
    Pista 2: LLLLL
    Pista 3: LLLLL
    Pista 4: LLLLL
    Pista 5: LLLLL
    Pista 6: LLLLL
    Pista 7: LLLLL
    Pista 8: LLLLL
    Pista 9: LLLLL
  Superficie 2:
    Pista 0: OOOOO
    Pista 1: OOOOO
    Pista 2: LLLLL
    Pista 3: LLLLL
    Pista 4: LLLLL
    Pista 5: LLLLL
    Pista 6: LLLLL
    Pista 7: LLLLL
    Pista 8: LLLLL
    Pista 9: LLLLL
  Superficie 3:
```

REFERENCIAS

CPP REFERENCE

SILBERSCHATZ, A., KORTH, H. F., & SUDARSHAN, S. DATABASE SYSTEM CONCEPTS.

GARCIA-MOLINA, H. DATABASE SYSTEMS: THE COMPLETE BOOK. PEARSON EDUCATION INDIA.

OCTUBRE 2030

GRACIAS

www.unsitiogenial.es

MAITE ALLENDE