# Edinson Pedroza – Recuperación

# Link del repositorio:

https://github.com/EdinsonPedroza/RECUPERACION-SISTEMAS-OPERATIVOS.git

# PROBLEMA 1 DOCUMENTADO:

```
#include <iostream>


class DiskDrive {
   public:
      // Atributos
      int diskSize; // Tamano del disco en GB
      int sectorsPerTrack; // Numero de sectores por pista
      int diskPlates; // Numero de platos del disco
      const int bytesPerSector = 512; // Numero de bytes por sector

      // Constructor
      DiskDrive(int size, int sectors) {
         diskSize = size;
         sectorsPerTrack = sectors;
      }

      // Metodo para calcular el numero total de sectores
      int getTotalSectors() {
         return (diskSize * 1000000000) / bytesPerSector;
      }

      // Metodo para leer el numero de pistas o platos y calcular el otro valor
```

```
void leerDiscoPlato() {
    char option;
    std::cout << "Quieres ingresar el numero de pistas (T) o el numero de platos (P)? ";
    std::cin >> option;
    if (option == 'T' || option == 't') {
        int tracks;
        std::cout << "Ingrese el numero de pistas: ";
        std::cin >> tracks;
        diskPlates = diskSize / (sectorsPerTrack * tracks);
        std::cout << "El numero de platos es: " << diskPlates << "\n";
    }
    else if (option == 'P' || option == 'p') {
        int plates;
        std::cout << "Ingrese el numero de platos: ";
        std::cin >> plates;
        diskPlates = plates;
        int tracks = diskSize / (sectorsPerTrack * plates);
        std::cout << "El numero de pistas es: " << tracks << "\n";
    }
    else {
        std::cout << "Opcion invalida.\n";
    }
}

// Metodo para imprimir los atributos del disco
void imprimirInformacion() {
    std::cout << "Tamano del disco: " << diskSize << " GB\n";
    std::cout << "Numero total de sectores: " << getTotalSectors() << "\n";
    std::cout << "Numero de sectores por pista: " << sectorsPerTrack << "\n";
```

```
            std::cout << "Numero de platos del disco: " << diskPlates << "\n";
        }
};

int main() {
    // Leer el tamano del disco y el numero de sectores por pista desde la consola
    int size, sectors;
    std::cout << "Ingrese el tamano del disco en GB: ";
    std::cin >> size;
    std::cout << "Ingrese el numero de sectores por pista: ";
    std::cin >> sectors;

    // Crear un objeto de la clase DiskDrive con los datos ingresados
    DiskDrive disk(size, sectors);

    // Leer el numero de pistas o platos y calcular el otro valor
    disk.leerDiscoPlato();

    // Imprimir la informacion del disco
    disk.imprimirInformacion();

    return 0;
}
```

# PROBLEMA 2 DOCUMENTADO:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
```

```
using namespace std;

// Funcion para simular el algoritmo FCFS
void fcfs(int pistas, int pos_inicial, vector<int> solicitudes) {
    cout << "Algoritmo FCFS:\n";
    int movimientos = 0; // contador de movimientos del cabezal
    int pos_actual = pos_inicial; // posicion actual del cabezal
    // Se recorre la secuencia de solicitudes en orden de llegada
    for (int pista : solicitudes) {
        // Se calcula la distancia entre la posicion actual y la pista solicitada
        int distancia = abs(pista - pos_actual);
        // Actualizar el contador de movimientos y la posicion actual
        movimientos += distancia;
        pos_actual = pista;
        // Se muestra el movimiento realizado
        cout << "Movido de la pista " << pos_actual - distancia << " a la pista " << pos_actual << "\n";
    }
    // Se muestra el total de movimientos realizados
    cout << "Total de movimientos: " << movimientos << "\n";
}

// Funcion para simular el algoritmo SCAN
void scan(int pistas, int pos_inicial, vector<int> solicitudes) {
    cout << "Algoritmo SCAN:\n";
    int movimientos = 0; // contador de movimientos del cabezal
    int pos_actual = pos_inicial; // posicion actual del cabezal
    bool direccion = true; // direccion del movimiento del cabezal (true = derecha, false = izquierda)
    // Ordenar la secuencia de solicitudes de menor a mayor
```

```
    sort(solicitudes.begin(), solicitudes.end());
    // Se encuentra el indice de la primera solicitud mayor o igual que la posicion inicial
    int indice = lower_bound(solicitudes.begin(), solicitudes.end(), pos_inicial) - solicitudes.begin();
    // Se recorre la secuencia de solicitudes desde el indice hasta el final, y luego desde el indice
hasta el inicio
    while (indice >= 0 && indice < solicitudes.size()) {
        // Se obtiene la pista solicitada en el indice actual
        int pista = solicitudes[indice];
        // Se calcula la distancia entre la posicion actual y la pista solicitada
        int distancia = abs(pista - pos_actual);
        // Se actualiza el contador de movimientos y la posicion actual
        movimientos += distancia;
        pos_actual = pista;
        // Se muestra el movimiento realizado
        cout << "Movido de la pista " << pos_actual - distancia << " a la pista " << pos_actual << "\n";
        // Si se llega al final o al inicio del disco, cambiar la direccion del movimiento
        if (indice == solicitudes.size() - 1 || indice == 0) {
            direccion = !direccion;
        }
        // Avanzar o retroceder el indice segun la direccion del movimiento
        if (direccion) {
            indice++;
        } else {
            indice--;
        }
    }
    // Se muestra el total de movimientos realizados
    cout << "Total de movimientos: " << movimientos << "\n";
}
```

```
// Algoritmo C-SCAN
void cscan(int pistas, int pos_inicial, vector<int> solicitudes) {
    cout << "Simulando algoritmo C-SCAN:\n";
    int movimientos = 0; // contador de movimientos del cabezal
    int pos_actual = pos_inicial; // posicion actual del cabezal
    bool direccion = true; // direccion del movimiento del cabezal (true = derecha, false = izquierda)
    // Se ordena la secuencia de solicitudes de menor a mayor
    sort(solicitudes.begin(), solicitudes.end());
    // Se encuentra el indice de la primera solicitud mayor o igual que la posicion inicial
    int indice = lower_bound(solicitudes.begin(), solicitudes.end(), pos_inicial) - solicitudes.begin();
    // Se recorre la secuencia de solicitudes desde el indice hasta el final, y luego desde el inicio hasta el indice
    while (indice >= 0 && indice < solicitudes.size()) {
        // Se obtiene la pista solicitada en el indice actual
        int pista = solicitudes[indice];
        // Se calcula la distancia entre la posicion actual y la pista solicitada
        int distancia = abs(pista - pos_actual);
        // Se actualiza el contador de movimientos y la posicion actual
        movimientos += distancia;
        pos_actual = pista;
        // Se muestra el movimiento realizado
        cout << "Movido de la pista " << pos_actual - distancia << " a la pista " << pos_actual << "\n";
        // Si se llega al final del disco, mover el cabezal al inicio del disco y cambiar la direccion del movimiento
        if (indice == solicitudes.size() - 1) {
            movimientos += pistas - 1;
            pos_actual = 0;
            cout << "Movido de la pista " << pistas - 1 << " a la pista " << pos_actual << "\n";
            direccion = !direccion;
```

```
        }
        // Si se llega al inicio del disco, mover el cabezal al final del disco y cambiar la direccion del movimiento
        if (indice == 0) {
            movimientos += pistas - 1;
            pos_actual = pistas - 1;
            cout << "Movido de la pista " << 0 << " a la pista " << pos_actual << "\n";
            direccion = !direccion;
        }
        // Avanzar o retroceder el indice segun la direccion del movimiento
        if (direccion) {
            indice++;
        } else {
            indice--;
        }
    }
    // Se muestra el total de movimientos realizados
    cout << "Total de movimientos: " << movimientos << "\n";
}



int main() {
    // Se pide al usuario los datos necesarios para simular los algoritmos
    cout << "Ingrese el numero de pistas del disco:\n";
    int pistas;
    cin >> pistas;
    cout << "Ingrese la posicion inicial del cabezal:\n";
    int pos_inicial;
    cin >> pos_inicial;
```

```
cout << "Ingrese el numero de solicitudes de acceso a pistas:\n";
int n;
cin >> n;
cout << "Ingrese la secuencia de solicitudes de acceso a pistas:\n";
vector<int> solicitudes(n);
for (int i = 0; i < n; i++) {
   cin >> solicitudes[i];
}
// Se pide al usuario que algoritmo desea utilizar
cout << "Ingrese el numero del algoritmo que desea utilizar:\n";
cout << "1. FCFS\n";
cout << "2. SCAN\n";
cout << "3. C-SCAN\n";
int opcion;
cin >> opcion;

switch (opcion) {
   case 1:
      fcfs(pistas, pos_inicial, solicitudes);
      break;
   case 2:
      scan(pistas, pos_inicial, solicitudes);
      break;
   case 3:
      cscan(pistas, pos_inicial, solicitudes);
      break;
   default:
      cout << "Opcion invalida.\n";
      break;
```

```
    }


    return 0;
}
```