Organização de Arquivos

Setor, trilha, superfície e prato: endereçamento do disco de fábrica

Página de disco (cluster): unidade de transferência de dados entre Ram(primária) e disco (secundária). Sequência contínua de setores no disco. Ela é física e consecutiva. Conversa-se aí com o disco. Ela é uma ideia. Os setores é a parte física mesmo que vem de fábrica já dividido.

Descritor do arquivo: guarda as descrições do arquivo

fopen é caro => TEM QUE DAR fclose

### **GPT: Como fopen() funciona**

* Especificação do Modo de Abertura: Ao chamar fopen(), você especifica o caminho do arquivo e o modo de abertura (por exemplo, leitura, escrita, adição).
* Verificação de Permissões: O sistema verifica se o programa tem permissão para acessar o arquivo no modo desejado.
* Alocação de Recursos: Se a permissão for concedida, o sistema operacional aloca recursos para gerenciar o arquivo. Isso inclui alocar um buffer para o fluxo de dados e um objeto FILE para representar o fluxo.
* Retorno do Ponteiro FILE: fopen() retorna um ponteiro para o objeto FILE se o arquivo for aberto com sucesso; caso contrário, retorna NULL.

Mesmo que eu escrevo um char, ele já tem 4kb porque ele tem uma página de disco pra isso.

fread(), mesmo que tenha um byte, ele vai ler a página de disco inteira e trazer pra RAM com um acesso a disco. O ponteiro que vai lendo ele vai andando a medida que você pede. O que pode mudar onde esse ponteiro amarelo está é o fseek()

fread() sem acesso a disco, se a página já estiver na RAM.

Bufferização resume o número de acesso a discos (reúne tudo e escreve, no caso do fwrite())

SSD usa memória flash

Fingimento: tira da tabela (endereçamento), mas ele ainda tá lá. Por isso que tem programas que conseguem recuperar. Só perde quando você sobrescreve (fwrite)

(👉)Ex: 4 acessos a discos: 4 rotação, 4 seek(pesado), 4 transferência (isso no HD). Se eu colocar páginas de disco sequenciais NO DISCO, o tempo de seek diminui. Reescreve e demora 8 acessos a disco (4 read 4 write).

Byte offset: endereço dentro do arquivo que vc quer

Todo campo de tamanho fixo pode ter espaços nulos se ele não for totalmente ocupado

Campo: nome, endereço, data de nascim, etc

Pesquisa = Busca facilitada (sinônimos): vantagem do banco de tamanho fixo -> pular ou fórmula matemática

Ver vantagens e desvantagens de campo de tamanho fixo e variável

No variável tem como fazer sem \0, colocando quando bytes tem o dado antes dele.

Registro é um conjunto de campos

Escrito por Cristina: \*Se todos os campos são de tamanho fixo, então o registro são de tamanho fixo\*

Se eu tenho um registro de tamanho fixo, eu posso ter um registro de tamanho variável (é bom inclusive). O mesmo com campo variável e fixo (ao msm tempo)

Preferência: tamanho fixo, índices (pular) e delimitador (‘\0’)

RRN = Número relativo do registro (n-1)

Este número sequencial, o RRN, funciona como um índice que permite o acesso direto ao registro correspondente

O número antes do registro nunca conta o 4 bytes do número

Outro método: contar o número de campos; a cada x campos começa um novo registro

Campos de tamanho fixo não podem ser identificados por delimitador ou indicador de tamanho

Remoção lógica: marcar como removido( \*; campo adicional ✓)

-> Busca por RRN:

abrir o arquivo fopen()

testar se não está no final do arquivo ou se o arquivo existe

a = fseek(RRN \* TamanhoRegistro) -> RRN = 1

b = fread(TamnhoRegistro)

se o primeiro byte for diferente de ‘\*’ então faça:

imprime na tela o registro separado por campo a campo

senão faça:

imprime “registro não encontrado”

fclose()

PREZE PELA MODULARIZAÇÃO!!!

-> Remoção por RRN:

abrir arquivo para leitura e escrita

testar se arquivo existe ou se está EOF(End Of File)

fseek (RRN \* tamReg)

b <- fread(array de tamanho 1)

se b != \*: // Se existe   
 se primeiro byte for != \* então: // Esse cara é necessário porque fwrite é custoso (economiza 1 acesso a disco em alguns casos)

mudar o primeiro byte p/ \*

// O registro não removeu exatamente

// Não se pode shiftar disco (por isso o asterisco)

-> Inserção (sempre no final, mesmo que algum tenha \*) => Porque tem que procurar os RRN

abrir arquivo escrita/leitura

testar se existe

fseek (final do arquivo)

fwrite(Cristina | | | xxxx…)

fclose()

// Pode colocar esses numa lista de registro removidas pra fazer reuso (programar lista em disco)

-> Como faço para apagar de verdade (físico) os asteriscos? Eu crio um novo arquivo, copio tudo que não tinha asterisco no velho e apago o antigo

Algoritmo disso acima:

fopen() o arquivo existente para leitura

abrir arquivo novo para escrita -> Página de disco sequencial do disco

enquanto arquivo existente não acabou (while (! EOF)) então:

ler o registro do arquivo existente

se o primeiro byte for != \*:

escreve registro no novo arquivo

fclose(arquivo 1)

fclose(arquivo 2)

remover o arquivo existente

renomeio o arquivo novo para o mesmo nome do velho (rename)

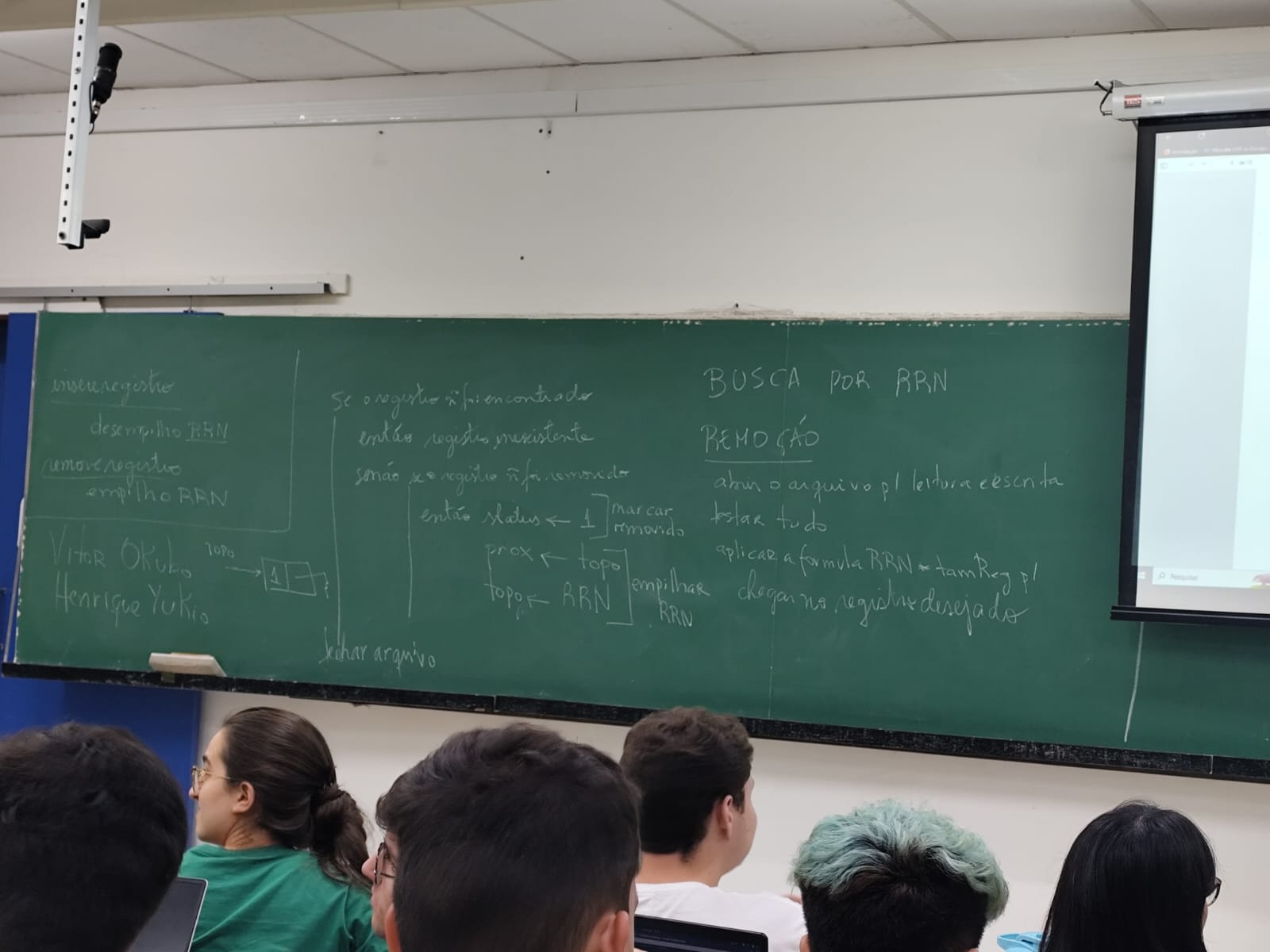
(👉)Páginas sequenciais no disco => o arquivo ficou menor e bem mais fácil de ser lido

**ABORDAGEM DINÂMICA**

(Várias remoções acontecendo ao mesmo tempo)

Utilizaremos pilha (são elementos que já foram removidos) -> ATENÇÃO: remoção empilha (delete)! Inserção desempilha (insert into)

O topo tá no registro de cabeçalho



Inserção

Se topo == -1

inserir no final do arquivo

Senão

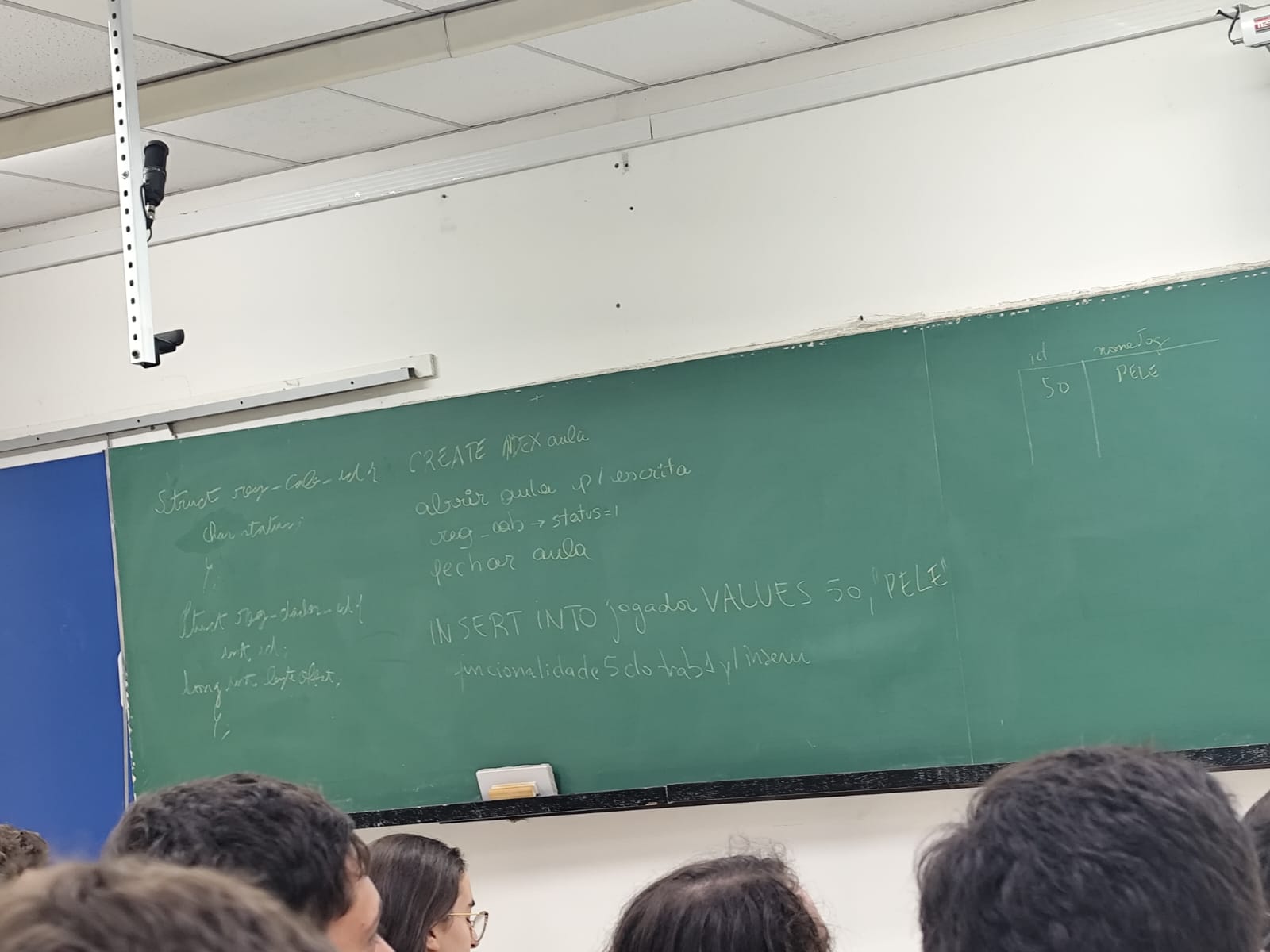
Aplicar a fórmula do RRN baseado no topo

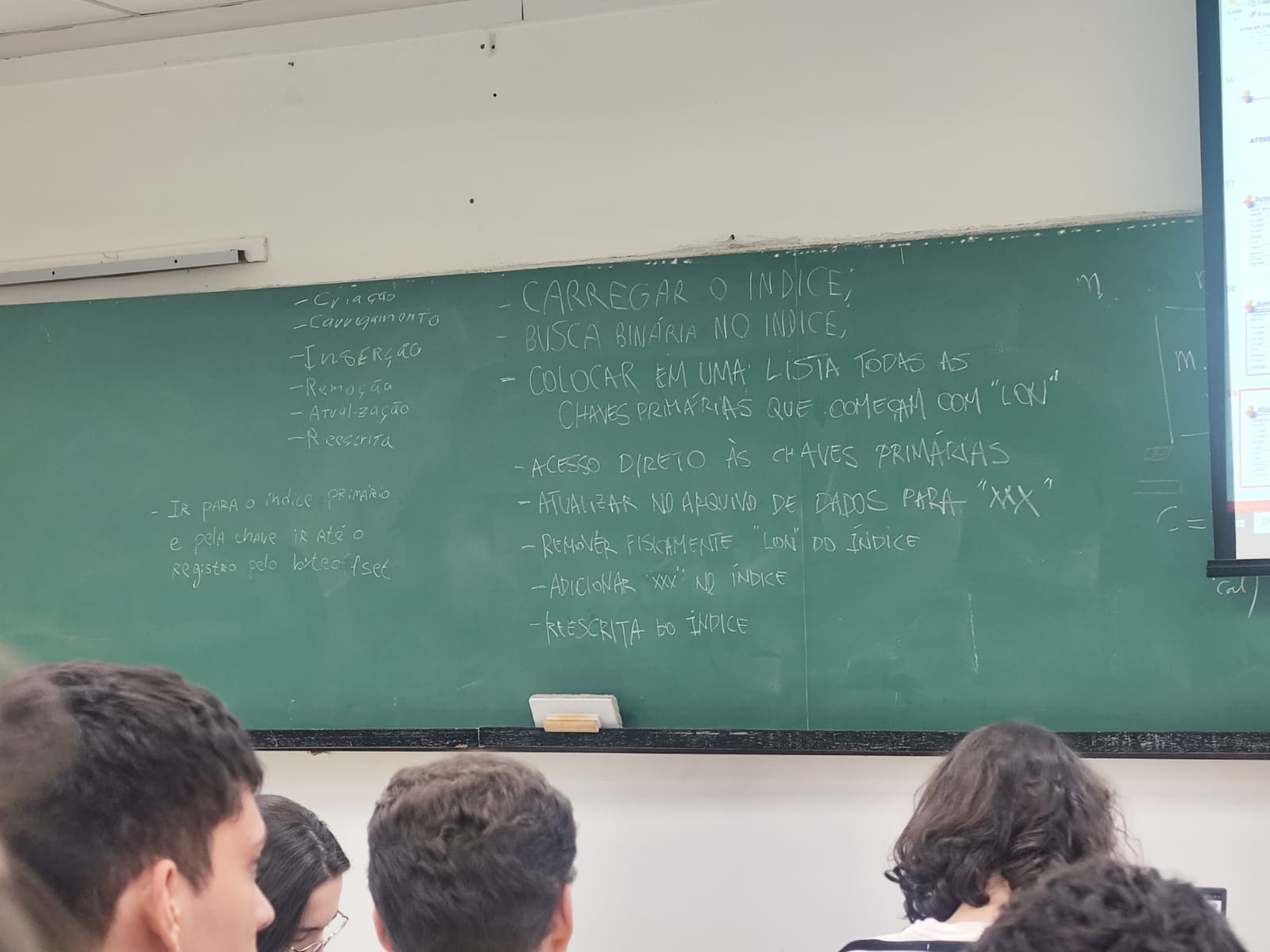
status = 0

topo = prox

escrever o novo registro

uma coisa boa do worst fit é que se vc precisar fazer um update vai ter espaço disponível.



Aulas índices:  
  


Árvores B

* Construída de baixo para cima
* Estrutura de indexação
* Volume grande de regs
* Poucos níveis => menos acessos
* Cada nó, uma página de disco
* Sempre balanceada
* Registro cabeçalho -> B tem o RRN do nó raiz
* Várias chaves num nó para reduzir acesso a disco

Árvore de ordem 5: ordem = número de descendentes -> chaves = ordem - 1

struct paginaArvoreB{

short contador;

char chaves[ordem-1]; // Ordem é passada ou definida

int descendentes[ordem];

int registros[ordem-1];

}