UNIVERSIDADE FEDERAL DE OURO PRETO
DEPARTAMENTO DE COMPUTAÇÃO E SISTEMAS

Disciplina: Turma: Algoritmos e Estruturas de Dados II
Professor: Rafael Alexandre

Aluno: João Victor Teixeira Pereira
Matrícula: 22.1.8052

Trabalho Prático 01: Manipulação e Organização de Arquivos de Dados

João Monlevade
2025

1. Descrição da Aplicação

Este projeto implementa um simulador de armazenamento de dados em disco, conforme especificado no Trabalho Prático 01 da disciplina de Algoritmos e Estruturas de Dados II, do professor Rafael Alexandre. O programa gera um conjunto de registros fictícios de **Alunos** e simula como eles seriam armazenados em um arquivo binário (**alunos.dat**).

O objetivo principal é avaliar e comparar três estratégias distintas de organização de registros, considerando a limitação de armazenamento em blocos de tamanho fixo:

1. Registros de Tamanho Fixo
2. Registros de Tamanho Variável (Contíguos / Sem Espalhamento)
3. Registros de Tamanho Variável (Espalhados / Com Espalhamento)

Após a simulação, o programa cria os blocos de registros, calcula e exibe estatísticas detalhadas sobre a eficiência do armazenamento, como a taxa de ocupação dos blocos e a eficiência total .

2. Decisões de Projeto

A arquitetura do projeto foi guiada pelas seguintes decisões:

- **Linguagem:** Python foi escolhido como a linguagem de desenvolvimento. Esta decisão foi motivada pela sua vasta biblioteca padrão (como **struct** para manipulação binária e **os** para arquivos), excelente ecossistema (biblioteca **Faker** para geração de dados) e facilidade para rápida prototipação e futura implementação de interfaces gráficas.
- **Modularidade:** O código foi estruturado em módulos distintos para garantir clareza e organização. A separação inclui:
    - **Aluno.py:** Definição do modelo de dados.
    - **Dados.py:** Geração dos dados fictícios.
    - **Parametros.py:** Coleta de inputs do usuário.
    - **Serializacao.py:** Lógica central de serialização e alocação em blocos.
    - **Estatisticas.py:** Cálculo e exibição dos resultados.
    - **main.py:** Ponto de entrada que orquestra a execução.
- **Serialização:** Para a manipulação binária, foi utilizada a biblioteca **struct** do Python para "empacotar" tipos numéricos (**int, float**) e o método **.encode('utf-8')** para strings.
- **Visualização:** Além do arquivo principal **alunos.dat**, o programa oferece a opção de criar uma pasta **blocos_dat/** contendo arquivos individuais para cada bloco (**bloco_1.dat, bloco_2.dat,** etc.), permitindo uma inspeção visual direta do resultado da alocação.

3. Detalhes Importantes de Implementação

A implementação de cada estratégia seguiu regras específicas:

- **Tamanho Fixo:**
    - Todos os registros são convertidos para um tamanho fixo (171 bytes no nosso caso), preenchendo campos de string com o caractere **#** (padding) até seu tamanho máximo.
    - A alocação é simples: **N** registros são colocados em cada bloco, onde **N** é o **tamanho_bloco // tamanho_registro.** O espaço restante no bloco é preenchido com **padding**.
- **Tamanho Variável (Geral):**
    - Para identificar o início e o fim de cada registro, foi adotada uma estratégia de prefixo de tamanho.
    - Cada registro (ou fragmento de registro) é precedido por um **unsigned short** de 2 bytes que armazena o tamanho do "payload" (dados) a seguir.
- **Variável - Contíguo (Sem Espalhamento):**
    - O programa calcula o tamanho total do registro (**prefixo + payload**).
    - Se o registro não couber no espaço restante do bloco atual, ele é movido integralmente para o próximo bloco. O espaço que sobrou no bloco anterior é considerado fragmentação interna e preenchido com **#**.
- **Variável - Espalhado (Com Espalhamento):**
    - Esta é a estratégia mais eficiente em uso de espaço. Se um registro não cabe, ele é fragmentado.
    - O bit mais significativo (MSB) do prefixo de tamanho de 2 bytes é usado como uma flag de continuação (**0x8000**).
    - Se **MSB=0**, os 15 bits restantes indicam o tamanho do último (ou único) fragmento do registro.
    - Se **MSB=1**, os 15 bits restantes indicam o tamanho de um fragmento que continua no próximo bloco.