Trabalho Prático 01: Manipulação e Organização de Arquivos de Dados

Relatório Técnico

Autor: Gustavo Henrique F. Pimentel **Matrícula:** 22.1.8039

Orientador: Rafael Alexandre **Disciplina:** Algoritmos e Estruturas de Dados II
Instituição: Universidade Federal de Ouro Preto (UFOP)

Local e Data: Ouro Preto, 12 de novembro de 2025

Sumário

---

1. Introdução

A forma como os dados são fisicamente armazenados em disco é um pilar fundamental em sistemas de gestão de bases de dados e aplicações que manipulam grandes volumes de informação. A organização dos registros influencia diretamente o desempenho, a eficiência de armazenamento e a complexidade de acesso aos dados .

O presente trabalho teve como objetivo desenvolver um programa em Python para simular, implementar e avaliar o impacto de três estratégias distintas de organização de registros: **Tamanho Fixo**, **Tamanho Variável Contíguo** e **Tamanho Variável Espalhado**. A simulação considerou a restrição de armazenamento em blocos de tamanho fixo, conforme especificado no roteiro da disciplina de Algoritmos e Estruturas de Dados II .

2. Metodologia e Decisões de Projeto

Para a realização da simulação, foram adotadas as seguintes decisões de projeto e implementadas as seguintes ferramentas:

- **Linguagem de Programação:** Python 3.x.
- **Geração de Dados:** Utilização da biblioteca `Faker` para gerar um conjunto de N registros fictícios de alunos (onde N é definido pelo usuário), seguindo os campos especificados no enunciado (Matrícula, Nome, CPF, Curso, Filiação, Ano de Ingresso e Coeficiente Acadêmico - CA).
- **Caractere de Preenchimento (*Padding*):** Foi definido o caractere `b'#'` binário como o byte de preenchimento para espaços não utilizados em blocos ou campos.

2.1. Estratégia 1: Registros de Tamanho Fixo

Nesta abordagem, cada registro ocupa o mesmo número de bytes em disco, independentemente do seu conteúdo real.

- **Decisão de Projeto (Serialização):** O tamanho fixo do registro foi calculado em **167 bytes**, baseado no maior tamanho possível de cada campo. Para garantir este tamanho, os campos numéricos (Matrícula, Ano, CA) foram serializados em bytes usando a biblioteca `struct` do Python. Os campos de texto (*String*) foram preenchidos com `b'#'` para atingirem o seu tamanho máximo estipulado (ex: Nome com 50 bytes).
- **Alocação:** Foi seguida a regra de que cada registro deve ser armazenado integralmente dentro de um único bloco. Se um registro de 167 bytes não coubesse no espaço restante do bloco atual, ele era movido integralmente para o bloco seguinte.

2.2. Estratégia 2: Registros de Tamanho Variável

Nesta abordagem, o tamanho do registro depende do conteúdo real dos seus campos, eliminando o desperdício interno (*padding*).

- **Decisão de Projeto (Serialização):** Para identificar o fim de um campo e o início do próximo, foi adotada uma estratégia de delimitadores. O caractere `|` (*pipe*) foi usado para separar os campos, e (quebra de linha) foi usado o caractere para marcar o fim de um registro. `\n`

Esta estratégia foi implementada em duas variações:

2.2.1. Contíguos (Sem Espalhamento)

- **Alocação:** Se um registro variável (ex: 120 bytes) não coubesse no espaço restante de um bloco (ex: 100 bytes), o registro era movido integralmente para o próximo bloco. O espaço restante de 100 bytes no bloco anterior era preenchido com `b'#'`, gerando **desperdício externo**.

2.2.2. Espalhados (Com Fragmentação)

- **Alocação:** Se um registro não coubesse, ele era fragmentado. O programa preenchia o espaço restante do bloco atual com a primeira parte do registro (enchendo-o a 100%) e escrevia o restante do registro no início do bloco seguinte. Esta lógica elimina o desperdício externo, exceto no último bloco.

3. Resultados e Análise Comparativa

Para avaliar o desempenho de cada estratégia, foi executada uma simulação com os seguintes parâmetros:

- **Número de Registros:** 100
- **Tamanho Máximo do Bloco:** 512 bytes

Os resultados obtidos e a análise comparativa estão resumidos na Tabela 1.

Tabela 1: Análise Comparativa das Estratégias de Armazenamento

Estratégia de Armazenamento	Blocos Usados	Ocupação Média	Eficiência Total	Tipo de Desperdício Principal
1. Tamanho Fixo	34	95,93%	95,93%	Interno (*Padding* nos campos)
2. Variável (Contíguo)	23	87,38%	87,38%	Externo (Espaço não utilizado no final dos blocos)
3. Variável (Espalhado)	20	99,04%	99,04%	Mínimo (Apenas no último bloco)

Análise dos Resultados:

1. **Tamanho Fixo:** Esta estratégia foi a que mais consumiu espaço, utilizando 34 blocos. A alta ocupação média (95,93%) é um reflexo da coincidência de que 3 registros de 167 bytes (501 bytes) se aproximam do tamanho do bloco (512 bytes). O principal problema é o **desperdício interno**, causado pelo preenchimento (`b'#'`) em campos que não utilizam seu tamanho máximo.
2. **Variável (Contíguo):** Ao eliminar o desperdício interno, esta estratégia reduziu drasticamente o uso de espaço para 23 blocos. No entanto, a sua eficiência total foi a mais baixa (87,38%), pois introduz o **desperdício externo**. Muitos blocos ficaram parcialmente cheios quando o registro seguinte não cabia e tinha de ser movido.
3. **Variável (Espalhado):** Esta foi a estratégia mais eficiente em termos de utilização de espaço. Ao permitir a fragmentação de registros, ela elimina quase todo o
   desperdício, exceto no último bloco. Utilizou apenas 20 blocos e alcançou uma eficiência quase perfeita de **99,04%**, com 19 dos 20 blocos ficando 100% cheios.

4. Conclusão

Os objetivos do trabalho foram alcançados com sucesso, demonstrando na prática as implicações de cada estratégia de organização de arquivos de dados.

Conclui-se que existe um *trade-off* (compromisso) claro entre a simplicidade de acesso e a eficiência de armazenamento:

- A estratégia de **Tamanho Fixo**, embora seja a mais simples de implementar para acesso aleatório (pois a posição de qualquer registro N pode ser calculada diretamente), é a que mais desperdiça espaço em disco.
- A estratégia de **Tamanho Variável com Espalhamento** (fragmentação) é, inquestionavelmente, a mais eficiente em termos de utilização de espaço, maximizando o uso dos blocos. No entanto, ela introduz uma maior
  complexidade na lógica de leitura, uma vez que um único registro pode estar dividido por múltiplos blocos.

A escolha ideal entre as estratégias dependerá, portanto, das prioridades do sistema: **velocidade de acesso** (favorecendo o Tamanho Fixo) ou **eficiência de armazenamento** (favorecendo o Tamanho Variável Espalhado).## 5. Repositório e Arquivos de Entrega

O código-fonte completo (`trabalho_aeds.py`) e os arquivos de dados gerados durante a simulação estão disponíveis no repositório GitHub do projeto.

Repositório GitHub: https://github.com/gustavohfpimentel/Trabalho-de-Aeds-2

Os seguintes arquivos de dados foram gerados:

- `alunos_fixo.dat` : Arquivo de dados gerado com a estratégia de Registros de Tamanho Fixo.
- `alunos_var_contiguo.dat` : Arquivo de dados gerado com a estratégia de Registros de Tamanho Variável Contíguo.
- `alunos_var_espalhado.dat` : Arquivo de dados gerado com a estratégia de Registros de Tamanho Variável Espalhado.*