Relatório TP01 - AEDSII

Aluno: Cauã Bandeira Nobre
Data: 14/11/2025

1. Introdução

Este relatório descreve a implementação de um simulador em Python que gerencia a persistência de registros de alunos em um arquivo .DAT, e compara três estratégias distintas de alocação de registros dentro de blocos de tamanho fixo:

1. **Registros de Tamanho Fixo:** Todos os registros ocupam o mesmo espaço (o tamanho do maior registro possível), preenchido com caracteres de *padding* (#).
2. **Registros de Tamanho Variável (Contíguos):** Cada registro ocupa apenas o espaço necessário. Se um registro não cabe em um bloco, ele é movido *inteiro* para o próximo bloco, gerando fragmentação interna.
3. **Registros de Tamanho Variável (Com Espalhamento):** O registro pode ser dividido (espalhado) entre blocos, preenchendo 100% do bloco atual antes de continuar no próximo.

2. Decisões de Projeto e Implementação

- main.py: Interface de usuário (CLI) e orquestrador da simulação.
- student.py: Definição da estrutura de dados (@dataclass) do Aluno.
- utils.py: Função generate_students() para criação de dados fictícios.
- simulator.py: Contém a classe StorageSimulator e as lógicas de serialização (pack_fixed, pack_variable), que são o núcleo do trabalho.

Serialização

- **Tamanho Fixo:** A função pack_fixed utiliza struct.pack para os tipos numéricos e preenche os campos de string com ljust(MAX_SIZE, b'#') para garantir um tamanho constante de **163 bytes** por registro.
- **Tamanho Variável:** A função pack_variable utiliza um delimitador nulo (b'\0') após cada campo de string, permitindo que o tamanho total do registro reflita o tamanho real dos dados.

Gerenciamento de Blocos

A classe StorageSimulator gerencia um bytearray (o current_block) que é "descarregado" (_flush_block) para o arquivo alunos.dat sempre que o bloco atinge seu limite ou uma estratégia o força.

3. Metodologia de Teste e Análise de Resultados

Para avaliar o impacto de cada estratégia, foram executados três cenários de teste distintos.

3.1. Teste 1: Comparação de Baseline

Este teste compara a eficiência geral das três estratégias sob condições idênticas.

- **Parâmetros da Simulação:**
  - Número de Registros: **1.000**
  - Tamanho do Bloco: **1024 bytes**

Resultados (Preencha com seus dados)

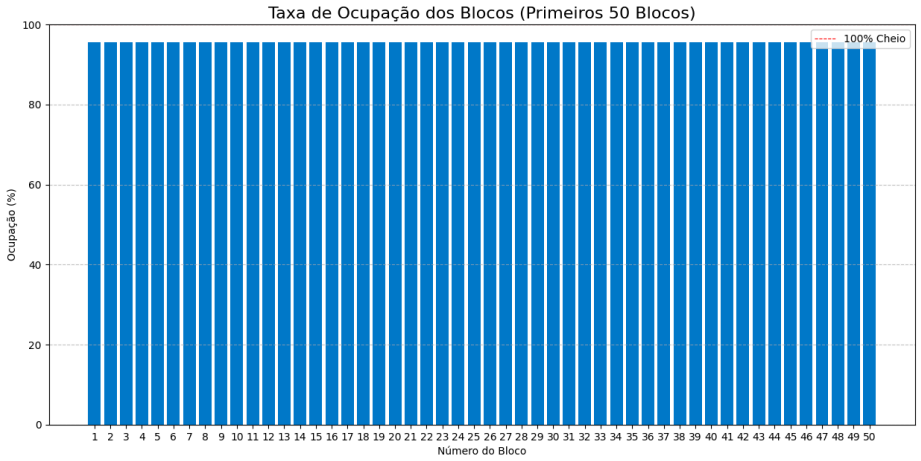
Estratégia	Blocos Utilizados	Ocupação Média (%)	Eficiência (Dados Úteis) (%)	Blocos Parciais
Tamanho Fixo	167	95.32%	52.96%	167
Var. Contíguo	93	94.96%	94.96%	93
Var. Espalhado	89	99.09%	99.09%	1

Análise

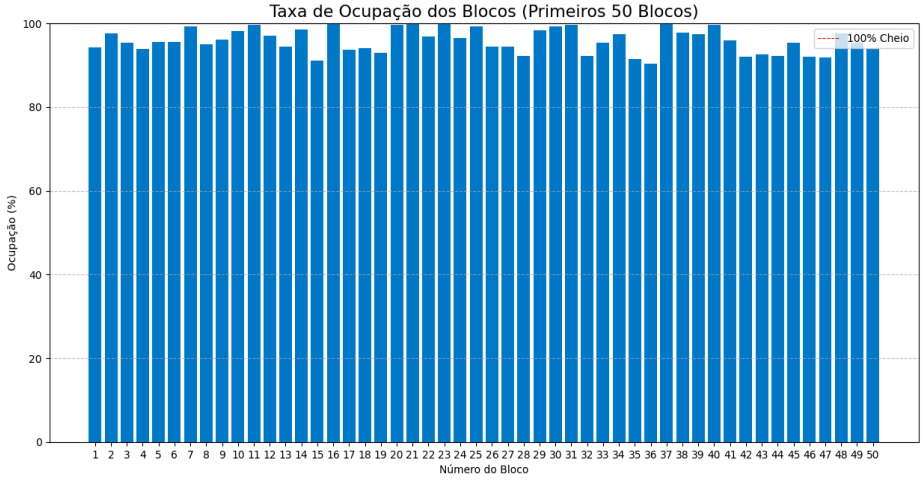
- A estratégia de **Tamanho Fixo** foi a que apresentou o pior desempenho, exigindo **167 blocos** para armazenar os dados. A descoberta mais importante está na discrepância entre a "Ocupação Média" (95.32%) e a "Eficiência" (**52.96%**). Isso demonstra que, embora os blocos estivessem 95% cheios, quase metade desse espaço (a diferença entre 95.32% e 52.96%) era composta por *padding* (caracteres '#') inútil, e não por dados reais dos alunos.
- O modo **Variável Contíguo** (sem espalhamento) representou uma melhoria drástica, reduzindo o uso de disco para **93 blocos**. O dado mais revelador é que **todos os 93 blocos** foram contados como "parciais". Isso comprova a existência de **fragmentação interna**: em cada bloco, sobrou um espaço no final que não pôde ser aproveitado pelo registro seguinte (que era maior que esse espaço), forçando-o a se mover para o bloco seguinte. Isso estabilizou a eficiência em **94.96%**.
- A estratégia **Variável com Espalhamento** foi a mais eficiente, utilizando apenas **89 blocos**. Os números de "Ocupação" e "Eficiência" (**99.09%**) são quase perfeitos. O dado principal é que houve apenas **1 Bloco Parcial**. Isso significa que esta estratégia eliminou completamente a fragmentação interna; todos os blocos (exceto o último do arquivo) foram preenchidos em 100% da sua capacidade, provando ser a solução com o melhor aproveitamento de espaço.

Gráficos de Ocupação

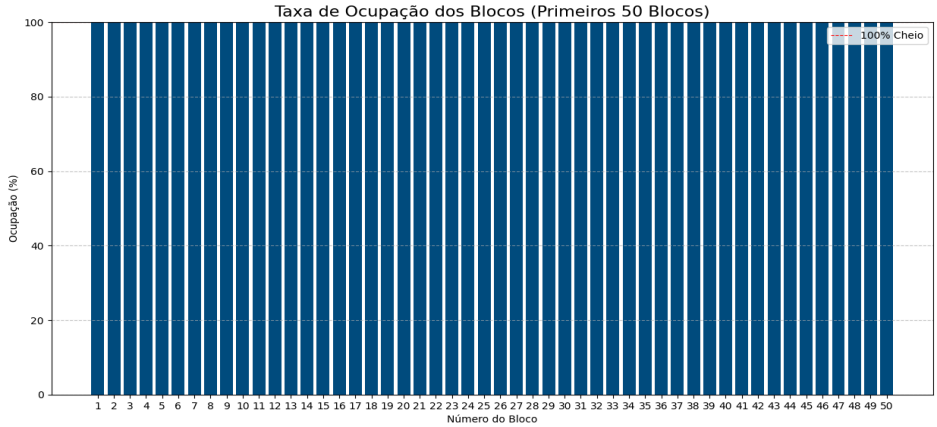
Tamanho Fixo



Var. Contíguo



Var. Espalhado



3.2. Teste 2: Impacto da Fragmentação Interna

Este teste analisa como o tamanho do bloco afeta o desperdício (fragmentação interna) na estratégia Variável Contígua.

- **Parâmetros da Simulação:**
  - Número de Registros: **1.000**
  - Estratégia: **Tamanho Variável (Contíguo)**

Resultados (Preencha com seus dados)

Tamanho do Bloco	Blocos Utilizados	Ocupação Média (%)
256 bytes	462	76.36%
1024 bytes	92	95.31%
8192 bytes	12	92.02%

Análise

- **Blocos pequenos (256 bytes):** Apresentaram a pior performance, com apenas **76.36%** de ocupação média. Isso ocorre porque o espaço médio desperdiçado no final de cada bloco (a fragmentação interna) representa um percentual muito alto do bloco. Por exemplo, desperdiçar 60 bytes em um bloco de 256 bytes é uma perda de ~23%.
- **Blocos muito grandes (8192 bytes):** Também mostraram uma queda na eficiência, caindo para **92.02%**. Embora o espaço desperdiçado *percentualmente* por bloco seja pequeno, o número total de blocos é muito menor (12 blocos). Isso faz com que o último bloco, que fica parcialmente preenchido, tenha um peso muito maior na média geral, "puxando" a ocupação média para baixo.
- **Blocos médios (1024 bytes):** Para este conjunto de 1.000 registros, o tamanho de 1024 bytes provou ser o "ponto de equilíbrio", atingindo a maior ocupação média de **95.31%**. Ele é grande o suficiente para que a fragmentação interna de cada bloco seja percentualmente pequena, mas não tão grande a ponto de o último bloco distorcer a média.
-

3.3. Teste 3: Cenário de "Edge Case" (Registro > Bloco)

Este teste valida a robustez das estratégias quando um único registro é maior que o bloco.

- **Parâmetros da Simulação:**
  - Número de Registros: **5**
  - Tamanho do Bloco: **100 bytes** (Nota: O registro de tamanho fixo tem 163 bytes)

Resultados (Preencha com seus dados)

Estratégia	Resultado da Execução
Tamanho Fixo	Erro: Registro 769817863 (163b) é maior que o bloco (100b) e não pode ser escrito. Erro: Registro 507598414 (163b) é maior que o bloco (100b) e não pode ser escrito. Erro: Registro 327489552 (163b) é maior que o bloco (100b) e não pode ser escrito. Erro: Registro 337348872 (163b) é maior que o bloco (100b) e não pode ser escrito. Erro: Registro 625392772 (163b) é maior que o bloco (100b) e não pode ser escrito.
Var. Contíguo	Erro: Registro 586984813 (111b) é maior que o bloco (100b) e não pode ser escrito. Erro: Registro 116989217 (107b) é maior que o bloco (100b) e não pode ser escrito.
Var. Espalhado	Bloco 1: 100 bytes usados / 100 bytes (100.00% cheio) Bloco 2: 100 bytes usados / 100 bytes (100.00% cheio) Bloco 3: 100 bytes usados / 100 bytes (100.00% cheio) Bloco 4: 100 bytes usados / 100 bytes (100.00% cheio) Bloco 5: 83 bytes usados / 100 bytes (83.00% cheio) Gerando gráfico em 'ocupacao_blocos.png'...

Análise

- Isso prova que as estratégias **'Tamanho Fixo'** e **'Variável Contíguo'** falham quando um único registro é maior que o tamanho do bloco.
    - No modo **Fixo**, o registro (163b) era inerentemente maior que o bloco (100b).
    - No modo **Contíguo**, os registros gerados aleatoriamente (ex: 111b, 107b) também eram maiores que o bloco.
    - Em ambos os casos, como essas estratégias não permitem a divisão de

registros, o programa corretamente identificou o erro e **se recusou a escrever os dados**, resultando em perda de informação.

- A estratégia **'Variável com Espalhamento'** foi a **única** capaz de lidar com essa situação. Conforme o *log* (Bloco 1: 100%, Bloco 2: 100%, etc.), ela pegou os registros grandes e os *dividiu* (espalhou) por múltiplos blocos, preenchendo-os completamente um a um até o último.
- Esta simulação demonstra que o espalhamento é a única estratégia robusta que garante a persistência dos dados independentemente do tamanho do bloco, eliminando o risco de falha por registros muito grandes.
-

4. Conclusão

Os testes demonstraram que não existe uma estratégia "perfeita", mas sim uma troca (*trade-off*) entre simplicidade de implementação e eficiência de espaço:

1. **Tamanho Fixo:** É o mais simples de gerenciar (cálculo de posição de registro é O(1)), mas é o mais ineficiente em armazenamento se os dados tiverem tamanhos variáveis.
2. **Variável Contíguo:** Oferece um bom equilíbrio, mas sofre com fragmentação interna, desperdiçando espaço no final de quase todos os blocos.
3. **Variável com Espalhamento:** É o mais eficiente em espaço, minimizando o desperdício. No entanto, é o mais complexo de implementar, pois na prática exigiria um sistema de ponteiros ou metadados para "remontar" o registro durante a leitura.

A escolha da estratégia correta dependerá dos requisitos da aplicação: se a prioridade é a eficiência de espaço (ex: arquivamento de logs), o espalhamento é ideal. Se a prioridade é a velocidade de acesso e modificação (ex: banco de dados transacional), o tamanho fixo pode ser preferível.