Relatório doTrabalho Prático 01: Manipulação e Organização de Arquivos de Dados

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

Aluno: Luiz Henrique Meira Andrade Leite

1. Contextualização

Em sistemas de gerenciamento de banco de dados e aplicações que manipulam grandes volumes de informação, a forma como os dados são armazenados fisicamente em disco influencia diretamente o desempenho e a eficiência do sistema.

Neste trabalho, você deverá desenvolver um programa que **simule a persistência de registros em um arquivo de dados (**.DAT**)**, considerando as restrições de armazena- mento em blocos e diferentes estratégias de organização de registros.

O objetivo é compreender as implicações práticas das decisões de projeto relacionadas ao **tamanho dos registros**, **limitação de blocos** e **métodos de alocação de dados em disco**.

2. Objetivos

- Implementar a leitura e gravação de registros em arquivos binários (.DAT);
- Simular o armazenamento de registros em **blocos de tamanho fixo** (em bytes), especificado pelo usuário;
- Avaliar o impacto das diferentes estratégias de organização:
    1. Registros de tamanho fixo;
    2. Registros de tamanho variável, com duas variações:
        (a) Contíguos (sem espalhamento);
        (b) Espalhados (fragmentados entre blocos).
- Calcular e exibir estatísticas de uso dos blocos e eficiência de armazenamento.

3. Especificação do Problema

O programa deverá simular o armazenamento de um conjunto de registros de alunos, com os

seguintes campos:

Campo	Tipo / Tamanho sugerido	Descrição
Matrícula	Inteiro (9 dígitos)	Identificador único do aluno
Nome	String (até 50 caracteres)	Nome completo do aluno
CPF	String (11 caracteres)	CPF do aluno
Curso	String (até 30 caracteres)	Curso em que o aluno está matriculado
Filiação (mãe)	String (até 30 caracteres)	Nome completo da mãe
Filiação (pai)	String (até 30 caracteres)	Nome completo do pai
Ano de Ingresso	Inteiro (4 dígitos)	Ano de entrada na instituição
CA	Float (2 casas decimais)	Coeficiente Acadêmico

O conjunto de registros pode ser gerado automaticamente (por exemplo, utilizando uma biblioteca de geração de dados fictícios).

O usuário deverá informar:

1. O tamanho máximo do bloco (em bytes);
2. O modo de armazenamento:
   (a) Registros de tamanho fixo;
   (b) Registros de tamanho variável;
3. Caso o modo 2 seja selecionado, escolher:
   (a) Registros contíguos;
   (b) Registros espalhados.

4. Regras de Armazenamento

a) Tamanho Fixo

- Todos os registros ocupam o mesmo número de bytes, definido a partir do maior tamanho possível de um registro;
- Caso um campo não utilize todos os bytes, o espaço restante deve ser preenchido com um caractere de preenchimento (por exemplo, # ou espaço);
- Cada registro deve ser armazenado integralmente dentro de um único bloco.

b) Tamanho Variável

- O tamanho de cada registro depende do conteúdo real (por exemplo, nomes de tamanhos distintos);
- Quando o registro não couber completamente em um bloco, há duas possibilidades:
  - **Sem espalhamento:** o registro é movido integralmente para o próximo bloco;
  - **Com espalhamento:** parte do registro é gravada no bloco atual e o restante continua no bloco seguinte, com referência indicando continuidade.

5. Saídas Esperadas

Após o armazenamento, o programa deve exibir:

1. Número total de blocos utilizados;
2. Percentual médio de ocupação de cada bloco;
3. Número de blocos parcialmente utilizados;
4. Eficiência de armazenamento (% de bytes ocupados por dados úteis);
5. Mapa de ocupação dos blocos (exemplo textual).

Exemplo ilustrativo:

Bloco 1: 512 bytes (100% cheio)
Bloco 2: 486 bytes (95% cheio)
Bloco 3: 210 bytes (41% cheio) Total de
blocos: 3
Eficiência total: 78,6%

6. Roteiro de Implementação (Passo a Passo)

1. **Geração dos dados:**
   - Criar um conjunto de registros de alunos com valores fictícios;
   - Permitir que o usuário defina o número total de registros a serem gerados.
2. **Definição dos parâmetros de armazenamento:**
   - Solicitar ao usuário o tamanho máximo do bloco (em bytes);
   - Permitir escolher entre registros de tamanho fixo ou variável;

- Em caso de registros variáveis, perguntar se haverá espalhamento.

3. **Simulação de escrita:**
   - Calcular o tamanho de cada registro;
   - Organizar os registros dentro dos blocos conforme a estratégia escolhida;
   - Criar o arquivo `alunos.dat` e gravar o conteúdo de cada bloco sequencialmente.
4. **Cálculo de estatísticas:**
   - Calcular a quantidade total de blocos;
   - Determinar a taxa de ocupação de cada bloco;
   - Calcular a eficiência total do armazenamento.
5. **Exibição dos resultados:**
   - Exibir o resumo estatístico;
   - Imprimir o mapa textual de ocupação dos blocos;
   - Opcionalmente, gerar um gráfico de barras com a taxa de ocupação de cada bloco.

7. Solução: Geração de Dados

O arquivo gerarAlunos.ts cria registros fictícios.

- O usuário define a quantidade total.

8. Entrada dos parâmetros

```
npm start -- NUM_ALUNOS TAMANHO_DO_BLOCO MODO SPREAD
```

npm start -- <NUM_ALUNOS> <TAMANHO_DO_BLOCO> <MODO SPREAD>

Exemplo Fixo:
npm start -- 50 512 fixo
Exemplo Contíguo:
npm start -- 100 512 contig no
Exemplo Espalhado:
npm start -- 100 512 contig yes
Exemplo Fragmentado:
npm start -- 100 512 contig yes

9. Saídas reais:

As saídas reais do programa dependem do tipo de tamanho escolhido, e do número de alunos e tamanho do bloco escolhidos.

Registros Variáveis Contíguos (spread=true)
BlocoSize: 600 bytes
Total de blocos: 5
Blocos parcialmente usados: 5
Ocupação média (% por bloco): 96.10
Eficiência total (% bytes úteis): 96.10

Bloco 1: 600 bytes (591 bytes usados) - 98.5%
Bloco 2: 600 bytes (594 bytes usados) - 99%
Bloco 3: 600 bytes (591 bytes usados) - 98.5%
Bloco 4: 600 bytes (594 bytes usados) - 99%
Bloco 5: 600 bytes (513 bytes usados) - 85.5%

Relatório - Registros Variáveis Contíguos (spread=true)
Gerado em: 2025-11-15T17:45:28.769Z
Total Blocos: 5
Blocos parcialmente usados: 5
Ocupação média por bloco: 96.10%
Eficiência total: 96.10%

Detalhes dos blocos:

Bloco 1: 600 bytes (591 bytes usados) - 98.5%
Bloco 2: 600 bytes (594 bytes usados) - 99%
Bloco 3: 600 bytes (591 bytes usados) - 98.5%
Bloco 4: 600 bytes (594 bytes usados) - 99%
Bloco 5: 600 bytes (513 bytes usados) - 85.5%

10. Soluções apresentadas:

O projeto foi produzido em Typescript, utilizando a lib “fs/promises”, que é necessária para a criação e manipulação de arquivos .DAT no ambiente NODE. As principais funções, como a criação dos blocos de acordo com o tipo de armazenamento foram dívidas nas pastas storage, e foram utilizados buffers sequenciais para simular blocos reais.

Os tipos de armazenamento, quantidade de alunos e tamanho dos blocos são selecionados a partir de comandos no terminal, de forma direta, e há diferenciação no caso de arquivos contíguos, por meio de “contig yes” e “contig no”. No caso de “no”, não há divisão de um último registro de um bloco, caso ele não caiba inteiramente nela, fazendo com que este registro seja armazenado no próximo bloco.**RELATÓRIO DO TRABALHO PRÁTICO 01: MANIPULAÇÃO E ORGANIZAÇÃO DE ARQUIVOS DE DADOS**

1. INTRODUÇÃO (Contextualização)

Em sistemas de gerenciamento de banco de dados e aplicações que manipulam grandes volumes de informação, a forma como os dados são armazenados fisicamente em disco influencia diretamente o desempenho e a eficiência do sistema.

Neste trabalho, você deverá desenvolver um programa que simule a persistência de registros em um arquivo de dados (.DAT), considerando as restrições de armazenamento em blocos e

diferentes estratégias de organização de registros. O objetivo é compreender as implicações práticas das decisões de projeto relacionadas ao tamanho dos registros, limitação de blocos e métodos de alocação de dados em disco.

2. OBJETIVOS

- Implementar a leitura e gravação de registros em arquivos binários (.DAT).
- Simular o armazenamento de registros em blocos de tamanho fixo (em bytes), especificado pelo usuário.
- Avaliar o impacto das diferentes estratégias de organização:
    - Registros de tamanho fixo;
    - Registros de tamanho variável, com duas variações: contíguos (sem espalhamento) e espalhados (fragmentados entre blocos).
- Calcular e exibir estatísticas de uso dos blocos e eficiência de armazenamento.

3. ESPECIFICAÇÃO DO PROBLEMA

O programa deverá simular o armazenamento de um conjunto de registros de alunos com os seguintes campos:

Campo	Tipo / Tamanho sugerido	Descrição
Matrícula	Inteiro (9 dígitos)	Identificador único do aluno
Nome	String (até 50 caracteres)	Nome completo do aluno
CPF	String (11 caracteres)	CPF do aluno
Curso	String (até 30 caracteres)	Curso em que o aluno está matriculado
Filiação (mãe)	String (até 30 caracteres)	Nome completo da mãe
Filiação (pai)	String (até 30 caracteres)	Nome completo do pai
Ano de Ingresso	Inteiro (4 dígitos)	Ano de entrada na instituição
CA	Float (2 casas decimais)	Coeficiente Acadêmico

O conjunto de registros pode ser gerado automaticamente (por exemplo, utilizando uma biblioteca de geração de dados fictícios).

O usuário deverá informar:

1. O tamanho máximo do bloco (em bytes);
2. O modo de armazenamento: Registros de tamanho fixo ou Registros de tamanho variável;
3. Caso o modo variável seja selecionado, escolher: Registros contíguos ou Registros espalhados.

4. REGRAS DE ARMAZENAMENTO

4.1 Tamanho Fixo

- Todos os registros ocupam o mesmo número de bytes, definido a partir do maior tamanho possível de um registro.
- Caso um campo não utilize todos os bytes, o espaço restante deve ser preenchido com um caractere de preenchimento (por exemplo, # ou espaço).
- Cada registro deve ser armazenado integralmente dentro de um único bloco.

4.2 Tamanho Variável

- O tamanho de cada registro depende do conteúdo real (por exemplo, nomes de tamanhos distintos).
- Quando o registro não couber completamente em um bloco, há duas possibilidades:
    - **Sem espalhamento (Contíguo):** o registro é movido integralmente para o próximo bloco.
    - **Com espalhamento (Fragmentado):** parte do registro é gravada no bloco atual e o restante continua no bloco seguinte, com referência indicando continuidade.

5. SAÍDAS ESPERADAS

Após o armazenamento, o programa deve exibir:

- Número total de blocos utilizados;
- Percentual médio de ocupação de cada bloco;
- Número de blocos parcialmente utilizados;
- Eficiência de armazenamento (% de bytes ocupados por dados úteis);
- Mapa de ocupação dos blocos (exemplo textual).

6. ROTEIRO DE IMPLEMENTAÇÃO (Passo a Passo)

6.1 Geração dos Dados

- Criar um conjunto de registros de alunos com valores fictícios.
- Permitir que o usuário defina o número total de registros a serem gerados.

6.2 Definição dos Parâmetros de Armazenamento

- Solicitar ao usuário o tamanho máximo do bloco (em bytes).
- Permitir escolher entre registros de tamanho fixo ou variável.
- Em caso de registros variáveis, perguntar se haverá espalhamento.

6.3 Simulação de Escrita

- Calcular o tamanho de cada registro.
- Organizar os registros dentro dos blocos conforme a estratégia escolhida.
- Criar o arquivo `alunos.dat` e gravar o conteúdo de cada bloco sequencialmente.

6.4 Cálculo de Estatísticas

- Calcular a quantidade total de blocos.
- Determinar a taxa de ocupação de cada bloco.
- Calcular a eficiência total do armazenamento.

6.5 Exibição dos Resultados

- Exibir o resumo estatístico.
- Imprimir o mapa textual de ocupação dos blocos.
- Opcionalmente, gerar um gráfico de barras com a taxa de ocupação de cada bloco.

7. DETALHES DE IMPLEMENTAÇÃO

7.1 Geração de Dados

O arquivo `gerarAlunos.ts` cria registros fictícios. O usuário define a quantidade total.

7.2 Entrada dos Parâmetros

A execução é feita via terminal, com o comando: `npm start -- <NUM_ALUNOS> <TAMANHO_DO_BLOCO> <MODO SPREAD>`.

- **Exemplo Fixo:** `npm start -- 50 512 fixo`
- **Exemplo Contíguo (sem espalhamento):** `npm start -- 100 512 contig no`
- **Exemplo Espalhado/Fragmentado (com espalhamento):** `npm start -- 100 512 contig yes`

7.3 Saídas Reais (Exemplo)

Exemplo de saídas para Registros Variáveis Contíguos (`spread=true`):

- `BlocoSize: 600 bytes`
- `Total de blocos: 5`
- `Blocos parcialmente usados: 5`
- `Ocupação média (% por bloco): 96.10`
- `Eficiência total (% bytes úteis): 96.10`

Detalhes dos blocos:

- Bloco 1: 600 bytes (591 bytes usados) - 98.5%
- Bloco 2: 600 bytes (594 bytes usados) - 99%
- Bloco 3: 600 bytes (591 bytes usados) - 98.5%
- Bloco 4: 600 bytes (594 bytes usados) - 99%
- Bloco 5: 600 bytes (513 bytes usados) - 85.5%

8. CONCLUSÃO (Soluções Apresentadas)

O projeto foi produzido em Typescript, utilizando a biblioteca “fs/promises”, que é necessária para a criação e manipulação de arquivos .DAT no ambiente NODE. As principais funções, como a criação dos blocos de acordo com o tipo de armazenamento, foram divididas nas pastas `storage`, e foram utilizados *buffers* sequenciais para simular blocos reais.

Os tipos de armazenamento, quantidade de alunos e tamanho dos blocos são selecionados a partir de comandos no terminal, de forma direta. Há diferenciação no caso de arquivos contíguos por meio de “contig yes” e “contig no”. No caso de “no”, não há divisão de um último registro de um bloco, caso ele não caiba inteiramente nele, fazendo com que este registro seja armazenado no próximo bloco.

PS: Formatei o relatório para um padrão ABNT por meio de IA por razões de teste, caso não seja

penalizado, utilizarei esta prática novamente para deixar próximos trabalhos nesta matéria mais entendíveis.