Universidade Federal do Amazonas
Instituto de Computação
Bancos de Dados I – 2025/02
Professor: Altigran Soares da Silva <alti@icomp.ufam.edu.br>

Trabalho Prático 25/09/2025
Data da Entrega: 24/10/2025

1. Apresentação

Este trabalho consiste na implementação de programas para armazenamento e consulta em dados armazenados em memória secundária utilizando as estruturas de arquivo de dados e índice estudadas nas aulas. Os programas devem fornecer suporte para a inserção, assim como diferentes formas de busca, seguindo as técnicas apresentadas nas aulas de organização e indexação de arquivos.

O trabalho deve ser implementado na **linguagem C++** utilizando as bibliotecas padrão de chamadas de sistema disponíveis no Linux.

Os dados para testes e avaliação dos programas implementados estão disponíveis neste arquivo em formato CSV que servirá apenas como **entrada de dados**.

2. Descrição do trabalho

O arquivo de dados deverá armazenar registros de dados sobre artigos científicos publicados em conferências. A estrutura deste arquivo será a seguinte:

Campo	Tipo	Descrição
ID	inteiro	Código identificador do artigo
Título	alfa 300	Título de artigo
Ano	inteiro	Ano de publicação do artigo
Autores	alfa 150	Lista dos autores do artigo
Citações	inteiro	Número de vezes que o artigo foi citado
Atualização	data e hora	Data e hora da última atualização dos dados
Snippet	alfa entre 100 e 1024	Resumo textual do artigo

Os seguintes programas devem ser implementados:

a) ***upload <file>*:** Programa que fará a carga inicial dos dados de entrada que irá criar um banco de dados composto pelos seguintes arquivos:

- **Arquivo de dados** organizado por hashing
- **Arquivo de índice primário** usando uma B+Tree armazenada em memória secundária
- **Arquivo de índice secundário** usando uma outra B+Tree em memória secundária

b) *findrec <ID>:* Programa que busca diretamente no **arquivo de dados** por um registro com o ID informado e, se existir, retorna os campos do registro, a quantidade de blocos lidos para encontrá-lo e a quantidade total de blocos do arquivo de dados;

c) *seek1 <ID>*: Programa que devolve o registro com ID igual ao informado, se existir, pesquisando através do **arquivo de índice primário,** mostrando todos os campos, a quantidade de blocos lidos para encontrá-lo no arquivo de índice e a quantidade total de blocos do arquivo de índice primário;

d) *seek2 <Titulo>*: Programa que mostra os dados do registro que possua o Título igual ao informado, se existir, pesquisando através do **arquivo de índice secundário**, informando a quantidade de blocos lidos para encontrá-lo no arquivo de índice e a quantidade total de blocos do arquivo de índice secundário

3. Registro das Equipes

Para registrar a equipe, os membros deverão postar uma mensagem com os **nomes** e **e-mails** dos membros da equipe no canal **#trabalho-pratico-2** até as **23:59 do dia 06/10/2025.**

Atenção: (1) os e-mails de todos os integrantes devem ser do domínio `icomp.ufam.edu.br;` (2) As equipes devem ter 2 ou 3 integrantes; (2) **Não serão aceitas mudanças nos integrantes da equipe depois que mensagem for postada portanto mensagens editadas serão desconsideradas.**

4. O que entregar

a) Os arquivos-fonte dos programas, com comentários adequados para permitir a correção

b) Os programas prontos para serem testados. Os nomes dos programas devem seguir as instruções indicadas no item 2 deste trabalho.

c) A documentação do projeto dos programas deve ser disponibilizada em um único arquivo PDF nomeado **TP2_documentação.pdf**, registrando as todas as decisões de projeto tomadas, incluindo:

- **a.** A estrutura de cada arquivo de dados e índices;
- **b.** Quais fontes formam cada programa;
- **c.** As funções que cada fonte contém;
- **d.** Quem desenvolveu cada fonte/função;
- **e.** Qual o papel de cada função;

d) Entrega com Docker
Inclua na raiz do repositório:

- `Dockerfile` (construção reprodutível da aplicação em C++).
- `docker-compose.yml` (opcional, apenas se ajudar a orquestrar entradas/volumes).
- `Makefile` com, no mínimo:
  - `make build` ⟶ compila binários localmente;
  - `make docker-build` ⟶ `docker build` da imagem;
  - `make docker-run-<prog>` para cada programa (`upload`, `findrec`, `seek1`, `seek2`).
- Estrutura de diretórios sugerida:

```
/app
├── src/                (código C++)
├── include/            (headers)
├── bin/                (binários gerados)
├── data/               (vazia no repositório; usada em runtime)
├── tests/              (opcional)
├── Dockerfile
├── docker-compose.yml  (opcional)
├── Makefile
├── TP2_documentação.pdf
└── README.md
```

Especificação mínima do Dockerfile (exemplo orientativo):

```
# exemplo base: pode usar debian:bookworm-slim ou ubuntu:24.04
FROM debian:bookworm-slim

RUN apt-get update && apt-get install -y --no-install-recommends \
      g++ make cmake && \
    rm -rf /var/lib/apt/lists/*

WORKDIR /app
COPY . /app

# compile usando Makefile padrão (ajuste conforme seu build)
RUN make build
```

```
# diretório para dados persistentes (montado como volume)
VOLUME ["/data"]

# variáveis de ambiente para facilitar testes
ENV CSV_PATH=/data/input.csv \
    DATA_DIR=/data/db \
    LOG_LEVEL=info

# por padrão, mostra ajuda dos binários
CMD ["bash", "-lc", "echo 'Use: docker run ... upload|findrec|seek1|seek2'; ls -l
bin/"]
```

Requisitos de execução no contêiner:

- Todos os quatro programas devem ser executáveis dentro do contêiner, por exemplo:
  - `docker run --rm -v $(pwd)/data:/data tp2 ./bin/upload /data/input.csv`
  - `docker run --rm -v $(pwd)/data:/data tp2 ./bin/findrec 123`
  - `docker run --rm -v $(pwd)/data:/data tp2 ./bin/seek1 123`
  - `docker run --rm -v $(pwd)/data:/data tp2 "./bin/seek2 Um Título Exato"`
- O diretório `/data` deve ser usado para entrada (CSV) e persistência dos arquivos de dados e índices (ex.: `/data/db`), permitindo reexecução entre contêineres.
- O CSV de entrada não deve ser copiado para dentro da imagem; será montado via `-v`.
- Os binários devem ficar em `/app/bin` e serem chamados explicitamente (ou configurar *entrypoints* específicos, ver abaixo).

Entrypoints opcionais (aceito, mas não obrigatório):

- Criar *scripts* leves em `/app/bin` como `entry-upload`, `entry-findrec`, etc., e parametrizar `ENTRYPOINT`/`CMD` no `Dockerfile` para reduzir verbosidade dos comandos.
- Alternativamente, usar `docker-compose.yml` com serviços:
- `services:`
- `  upload:`
- `    image: tp2`
- `    volumes: ["./data:/data"]`
- `    command: ["./bin/upload", "/data/input.csv"]`

Logs e mensuração:

- Cada programa deve imprimir claramente:
  - Caminhos dos arquivos usados (dados e índices);
  - Quantidade de blocos lidos por operação;
  - Tempo de execução (ms) por operação (medido no próprio programa, não por `time` do shell);
  - Mensagens de erro bem formatadas.
- Permitir configurar nível de log por `ENV LOG_LEVEL` (`error`, `warn`, `info`, `debug`).

Reprodutibilidade e portabilidade:

- A imagem deve construir e executar em qualquer host com Docker recente.
- Não dependa de caminhos absolutos fora de /app e /data.
- Fixe versões de pacotes quando possível e evite ferramentas desnecessárias.

README.md (obrigatório):

- Instruções de build local e via Docker.
- Comandos de execução para os quatro programas.
- Layout dos arquivos gerados em /data/db.
- Exemplo de entrada/saída (pequeno).

5. Como entregar

Através do Github Classroom neste assignment

O repositório deve conter:

- Código C++;
- `TP2_documentação.pdf`;
- `Dockerfile`, `Makefile` e (opcional) `docker-compose.yml`;
- `README.md` com todos os comandos necessários para **construir** e **executar** o projeto em Docker.

Sugestão (não obrigatório): configurar um *workflow* simples do **GitHub Actions** que faça `docker build` para validar a construção da imagem a cada *push*.

6. Avaliação

Execução: Execução correta: 30%; Saída legível: 15%; Estilo de programação: Código bem estruturado: 15%; Documentação: Código legível: 15%, Descrição das estruturas de dados e principais decisões: 25%

Critérios adicionais relacionados a Docker (incluídos nos itens acima):

- A imagem **constrói sem erros** a partir do `Dockerfile`.
- Os quatro programas **executam** corretamente **dentro do contêiner**, usando `/data` para entrada e persistência.
- O `README.md` contém **comandos testáveis** de `docker build` e `docker run`.
- Reprodutibilidade: não depender de recursos externos não documentados.

7. Comentários Gerais

Comece a fazer este trabalho logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar. Seja ético, desenvolva seu trabalho, não copie de outras equipes e nem da Internet.

O professor poderá pedir defesa do trabalho apresentado, sendo que somente um dos alunos será chamado para defender o trabalho pela equipe. Assim, é importante que todos os alunos participem do desenvolvimento.

Pode ser uma boa oportunidade para que os alunos exercitem o desenvolvimento usando programação por pares, que além de ajudar na qualidade do código desenvolvido, contribui para o aprendizado de todos os membros da equipe. Alguns ambientes de desenvolvimento integrado tem bom suporte para programação por pares remota.

Bônus - Dicas para o Desenvolvimento do Trabalho

1. Entendimento Profundo das Requisições: Certifique-se de que todos os membros da equipe compreendam completamente os requisitos do trabalho, incluindo as estruturas dos dados, os tipos de busca solicitados, e como os índices primário e secundário devem ser implementados e utilizados.
2. Planejamento e Divisão de Tarefas: Divida o trabalho de forma eficiente entre os membros da equipe, assegurando que cada um tenha uma parte clara do projeto para desenvolver. Isso pode incluir a implementação de diferentes partes do código, como a criação do arquivo de dados, a implementação das estruturas de índice ou a elaboração da documentação.
3. Familiarize-se com C++ e as Bibliotecas Necessárias: Como o projeto deve ser implementado em C++, é crucial que todos os membros da equipe tenham uma boa compreensão da linguagem e das bibliotecas padrão de chamadas de sistema no Linux. Dediquem tempo para revisar ou aprender sobre manipulação de arquivos, estruturas de dados (como árvores B+), e técnicas de hashing em C++.
4. Utilização de Controle de Versão: Use ferramentas de controle de versão como Git e GitHub para gerenciar o código-fonte do projeto. Isso facilita a colaboração entre os membros da equipe, permite o acompanhamento de mudanças e ajuda na integração das diferentes partes do projeto.
5. Testes Contínuos: Implemente testes desde o início do desenvolvimento. Isso ajuda a identificar e corrigir erros precocemente, garantindo que cada parte do sistema funcione corretamente antes de integrá-la ao projeto como um todo.

6. Documentação Detalhada: Mantenha a documentação atualizada ao longo do desenvolvimento do projeto. A documentação deve incluir a estrutura de cada arquivo de dados e índices, as funções implementadas, a divisão de trabalho dentro da equipe, e as decisões de projeto tomadas.
7. Desenvolvimento Colaborativo e Revisões de Código: Pratique programação por pares sempre que possível, pois isso ajuda a melhorar a qualidade do código e o aprendizado mútuo. Faça revisões de código regulares com os membros da equipe para garantir que o código seja limpo, bem estruturado e fácil de entender.
8. Preparação para a Defesa do Trabalho: Prepare-se para a possibilidade de defesa do trabalho, assegurando que todos os membros da equipe entendam todas as partes do projeto e possam discuti-las adequadamente.
9. Ética e Originalidade: Mantenha a integridade acadêmica. Desenvolva seu próprio trabalho e evite copiar de outras equipes ou da internet. A colaboração é importante, mas o plágio é inaceitável e pode resultar em penalidades severas.