Documentação do Trabalho Prático 2 de Bancos de Dados 1

Professor: Altigran Soares da Silva (alti@icomp.ufam.edu.br)
Monitor: Tarsis Azevedo (tarsis.azevedo@icomp.ufam.edu.br)

Integrantes da Equipe:

- Gabriel Silva Pacheco de Carvalho (gabriel.pacheco@icomp.ufam.edu.br)
- Max de Souza Barbosa (msb@icomp.ufam.edu.br)
- Noah Diunkz Oliveira Maia (diunkz@icomp.ufam.edu.br)

Essa documentação busca registrar todas as decisões de projeto tomadas durante a produção do Trabalho Prático 2 da disciplina Banco de Dados 1.

Este trabalho prático possui como objetivo a implementação de programas para armazenamento e consulta em dados armazenados em memória secundária, através da utilização de estruturas de arquivo de dados e índice vistos em sala de aula. A linguagem utilizada para a implementação foi C++.

O arquivo de dados armazena registros de dados sobre artigos científicos publicados em conferências. A estrutura dele é descrita a seguir:

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 programas implementados neste trabalho prático estão descritos a seguir.

- **Programa upload <file>**

É o programa que fará a carga inicial dos dados de entrada. Cria um banco de dados composto pelos seguintes arquivos: arquivo de dados, arquivo de índice primário, arquivo de índice secundário.

Esse programa é compilado através de quatro arquivos fonte, descritos nas seções de 1 a 4.

1. **upload.cpp** (desenvolvido por Gabriel)

É o código responsável por criar o banco de dados composto pelos seguintes arquivos: arquivo de dados (organizado por hashing), arquivo de índice primário (utilizado uma B+Tree armazenada em memória secundária), arquivo de índice secundário (utilizado uma B+Tree armazenada em memória secundária).

Além das funções implementadas nele, também faz o uso de outras que foram implementadas por outros códigos fonte, que serão descritos após o fim da descrição do atual.

Dependências:

- **hash.hpp**
- **IndicePrimario.hpp**
- **IndiceSecundario.hpp**
- **stdio.h**
- **stdlib.h**
- **string**
- **fstream**
- **iostream**
- **cstring**

Funções e suas descrições:

- **removeArquivos**: seu objetivo é remover os arquivos especificados pelos seus respectivos nomes, anteriormente gerados no diretório.
- **copiaString**: seu objetivo é copiar uma string b para uma string a, de tamanho a ser especificado.
- **parserLinhaArquivoDeEntrada**: seu objetivo é analisar uma linha de um arquivo de entrada (parser) e armazenar as informações em uma estrutura de artigo (Article).
- **main**: execução do programa.

2. **hash.cpp** (desenvolvido por Gabriel)

Esse código possui um arquivo de cabeçalho (**hash.hpp**), descrito a seguir.

Definições:

- **Tamanho do tipo alfa para o título** (a): 300
- **Tamanho do tipo alfa para os autores** (b): 150
- **Tamanho do tipo alfa para a atualização** (c): 20
- **Tamanho do tipo alfa para o snippet** (d): 1024

- **Número de buckets**: 270973
- **Número de registros**: 7
- **Tamanho do corpo do bloco**: número de registros * (12 + a + b + c + d)
- **Nome do arquivo hashing**: arquivo

Dependências:

- **fstream**

Estruturas:

- **Article**: id (inteiro não sinalizado), title (string), year (inteiro não sinalizado), author (string), citations (inteiro não sinalizado), update (string), snippet (string).
- **Block**: nRegisters (inteiro não sinalizado), body (string).

Cabeçalhos de funções:

- void **inicializaArquivoDeSaida** (fstream *f)
- bool **insereArquivoHash** (fstream *f, Article article)
- Article **buscaRegistroPorId** (fstream *f,int id)
- Article **buscaBucketPorTitulo** (fstream *f, int posicao, char title[T_TITLE])
- void **imprimirRegistroArt** (Article article)

Focando agora no código propriamente dito, ele é responsável por implementar as funções para realizar a construção do **arquivo de dados**, que é organizado por **hashing**.

Variáveis globais:

- int **collision**

Funções e suas descrições:

- **atualizaCabecalhoIndPrimario**: seu objetivo é inicializar um arquivo de saída, preenchendo-o com blocos vazios.
- **hashing**: seu objetivo é implementar um algoritmo de hashing simples (modular).
- **consultaBucketPorId**: seu objetivo é consultar e retornar o conteúdo de um bucket específico no arquivo de dados, com base no ID fornecido.
- **imprimirRegistroArt**: seu objetivo é imprimir os valores dos campos de um registro de um artigo (Article).

- **insereArquivoHash**: seu objetivo é inserir um registro de um artigo (Article) em um arquivo de dados, utilizando o método de hashing.
- **buscaRegistroPorId**: seu objetivo é buscar um registro no arquivo de dados, com base no ID fornecido.
- **buscaBucketPorPosicao**: seu objetivo é buscar um bucket (que representa um bloco) no arquivo de dados, com base na posição fornecida.
- **buscaBucketPorTitulo**: seu objetivo é buscar um registro no arquivo de dados, com base no título fornecido, realizando busca sequencial dentro de um bucket específico.

3. **IndicePrimario.cpp** (desenvolvido por Noah)

Esse código possui um arquivo de cabeçalho (**IndicePrimario.hpp**), descrito a seguir.

Definições:

- **Nome do arquivo**: "primaryIndexFile"
- **Ordem da árvore**: 1000
- **Quantidade de apontadores para cada nó**: 2 * ordem da árvore
- **Quantidade de chaves para cada nó**: quantidade de apontadores para cada nó - 1

Dependências:

- **stdio.h**
- **stdlib.h**
- **string.h**
- **string**
- **iostream**
- **fstream**
- **hash.hpp**

Estruturas:

- **NohPrimario**: tamanho (inteiro), posicao (inteiro), chave (vetor de inteiros), apontador (vetor de inteiros).
- **AuxNode**: chave (inteiro), ponteiroEsquerda (inteiro), ponteiroDireita (inteiro).
- **Header**: posicaoRaiz (inteiro), qtdNoh (inteiro).

Cabeçalhos de funções:

- void **InsereArqIndicePrim** (fstream *hashFile, fstream *primIdxFile)
- void **seek1** (const char *caminhoArquivoDados, const char *caminhoArquivoIndice, int chave)

Focando agora no código propriamente dito, ele é responsável por implementar as funções para realizar a construção do **arquivo de índice primário**, através do uso de uma **B+Tree** armazenada em memória secundária.

Dependências:

- **IndicePrimario.hpp**

Variáveis globais:

- Header ***header**
- fstream ***indexFile**, ***dataFile**

Funções e suas descrições:

- **atualizaCabecalhoIndPrimario**: seu objetivo é atualizar o cabeçalho de um arquivo de índice primário.
- **inicializaCabecalhoPrimario**: seu objetivo é inicializar o cabeçalho de um arquivo de índice primário.
- **criarNovoNohPrimario**: seu objetivo é criar e inicializar um novo nó primário.
- **insereNohArqIndice**: seu objetivo é inserir um nó primário em um arquivo de índice.
- **atualizaNohArquivo**: seu objetivo é atualizar um nó primário em um arquivo de índice.
- **InsereChavPagDisponivel**: seu objetivo é inserir uma chave e seu respectivo apontador em um nó primário que possui espaço disponível para essa inserção.
- **InsereChavNohDisponivel**: seu objetivo é inserir uma chave e seu respectivo apontador em um nó primário que possui espaço disponível para essa inserção. Apesar de ter o mesmo objetivo da anterior, há detalhes diferentes em cada implementação.
- **InsereChavPagCheia**: seu objetivo é lidar com a inserção de uma chave e seu respectivo apontador em um nó primário quando a página está cheia.

- **InsereChavNohCheio**: seu objetivo é lidar com a inserção de uma chave e seu respectivo apontador em um nó primário quando ele já está cheio.
- **consultaNohPrimArquivo**: seu objetivo é consultar e retornar um nó primário do arquivo de índice, com base na posição informada.
- **insereChaveArvore**: seu objetivo é inserir uma chave e um apontador na árvore, tratando os diversos casos de divisão de páginas que podem surgir no processo.
- **insereNaArvore**: seu objetivo é inserir uma chave e um apontador na árvore. Ela utiliza a função anterior em sua implementação.
- **populaArqIndicePrim**: seu objetivo é popular o arquivo de índice primário com as chaves iniciais de cada bloco do arquivo de dados.
- **InsereArqIndicePrim**: seu objetivo é realizar a inserção dos registros no arquivo de índice primário.
- **consultaCabecalhoArqIndicePrim**: seu objetivo é consultar e armazenar o cabeçalho do arquivo de índice primário.
- **abreArqIndice**: seu objetivo é abrir o arquivo de índice primário com base no caminho fornecido.
- **abreArqDados**: seu objetivo é abrir o arquivo de dados com base no caminho fornecido.
- **closeFiles**: seu objetivo é fechar e liberar os recursos dos arquivos abertos e do cabeçalho.
- **pesquisaChavArqIndPrim**: seu objetivo é realizar uma pesquisa de uma chave em um arquivo de índice primário.
- **lerBlocoNoArqDados**: seu objetivo é ler um bloco de dados em uma determinada posição de um arquivo de dados.
- **copDadosSourceParaTarget**: seu objetivo é copiar os dados de um artigo fonte para um artigo alvo.
- **consultDadosRegPorID**: seu objetivo é consultar e retornar um registro de dados específico, com base no seu ID.
- **seek1**: seu objetivo é buscar um registro com base no ID (chave primária) no arquivo de dados.

4. **IndiceSecundario.cpp** (desenvolvido por Noah)

Esse código possui um arquivo de cabeçalho (**IndiceSecundario.hpp**), descrito a seguir.

Definições:

- **Nome do arquivo**: "secondaryIndexFile"
- **Ordem da árvore**: 50
- **Quantidade de apontadores para cada nó**: 2 * ordem da árvore + 1
- **Quantidade de chaves para cada nó**: 2 * ordem da árvore

Dependências:

- **fstream**
- **hash.hpp**

Estruturas:

- **Node**: tamanho (inteiro), posicao (inteiro), chave (string), apontador (vetor de inteiros).
- **NodeAux**: chave (string), ponteiroEsquerda (inteiro), ponteiroDireita (inteiro).
- **Head**: posicaoRaiz (inteiro), qtdNoh (inteiro).

Cabeçalhos de funções:

- void **InsereArqIndiceSec** (fstream *hashFile, fstream *secIdxFile);
- int **pesquisaChavArqIndSec** (fstream *caminhoArquivoDados, fstream *caminhoArquivoIndice, char chave[300])

Focando agora no código propriamente dito, ele é responsável por implementar as funções para realizar a construção do **arquivo de índice secundário**, através do uso de uma **B+Tree** armazenada em memória secundária.

Dependências:

- **stdio.h**
- **stdlib.h**
- **string.h**
- **string**
- **iostream**
- **fstream**
- **IndiceSecundario.hpp**
- **hash.hpp**

Variáveis globais:

- Head ***head**

- fstream ***hashF**, ***secF**

Funções e suas descrições:

- **atualizaCabecalhoIndSecundario**: seu objetivo é atualizar o cabeçalho de um arquivo de índice secundário.
- **inicializaCabecalhoSecundario**: seu objetivo é inicializar o cabeçalho de um arquivo de índice secundário.
- **criarNovoNohSecundario**: seu objetivo é criar e inicializar um novo nó secundário.
- **insereNohArqIndice**: seu objetivo é inserir um nó secundário em um arquivo de índice.
- **atualizaNohArquivo**: seu objetivo é atualizar um nó secundário em um arquivo de índice.
- **InsereChavPagDisponivel**: seu objetivo é inserir uma chave e seu respectivo apontador em um nó secundário que possui espaço disponível para essa inserção.
- **InsereChavNohDisponivel**: seu objetivo é inserir uma chave e seu respectivo apontador em um nó secundário que possui espaço disponível para essa inserção. Apesar de ter o mesmo objetivo da anterior, há detalhes diferentes em cada implementação.
- **InsereChavPagCheia**: seu objetivo é lidar com a inserção de uma chave e seu respectivo apontador em um nó secundário quando a página está cheia.
- **InsereChavNohCheio**: seu objetivo é lidar com a inserção de uma chave e seu respectivo apontador em um nó secundário quando ele já está cheio.
- **consultaNohSecArquivo**: seu objetivo é consultar e retornar um nó secundário do arquivo de índice, com base na posição informada.
- **insereChaveArvore**: seu objetivo é inserir uma chave e um apontador na árvore, tratando os diversos casos de divisão de páginas que podem surgir no processo.
- **populaArqIndiceSec**: seu objetivo é popular o arquivo de índice secundário com as chaves iniciais de cada bloco do arquivo de dados.
- **InsereArqIndiceSec**: seu objetivo é realizar a inserção dos registros no arquivo de índice secundário.

    - **consultaCabecalhoArqIndiceSec**: seu objetivo é consultar e armazenar o cabeçalho do arquivo de índice secundário.
    - **pesquisaChavArqIndSec**: seu objetivo é realizar uma pesquisa de uma chave em um arquivo de índice secundário.

- **Programa findrec <ID>**

É o programa que busca diretamente no arquivo de dados por um registro com o ID informado. Caso ele exista, são retornados os campos do registro, a quantidade de blocos lidos para encontrá-lo e a quantidade total de blocos do arquivo de dados.

Esse programa é compilado através de dois arquivos fonte, o hash.cpp (já descrito anteriormente) e o **findrec.cpp** (desenvolvido por Gabriel), que, além de criar a interface para a execução do programa, utiliza a função **buscaRegistroPorId** (implementada no arquivo hash.cpp) para realizar a busca.

O **findrec.cpp** possui duas dependências, a biblioteca **iostream** e o cabeçalho hash.hpp. Possui apenas a função main, responsável por realizar a execução do programa.

- **Programa seek1 <ID>**

É o programa que devolve o registro com ID igual ao informado, caso ele exista, fazendo a busca através do arquivo de índice primário. Se encontrar o registro desejado, são retornados os campos do registro, a quantidade de blocos lidos para encontrá-lo e a quantidade total de blocos do arquivo de índice primário.

Esse programa é compilado através de três arquivos fonte, o hash.cpp e o IndicePrimario.cpp, já descritos anteriormente, e o **seek1.cpp** (desenvolvido por Noah), que utiliza a função **seek1** (implementada no arquivo IndicePrimario.cpp) para realizar a busca.

O **seek1.cpp** possui uma dependência, o cabeçalho IndicePrimario.hpp, e a função main, responsável por realizar a execução do programa.

- **Programa seek2 <Título>**

É o programa que devolve o registro com título igual ao informado, caso ele exista, fazendo a busca através do arquivo de índice secundário. Se encontrar o registro desejado, são retornados os campos do registro, a quantidade de blocos lidos para encontrá-lo e a quantidade total de blocos do arquivo de índice secundário.

Esse programa é compilado através de três arquivos fonte, o hash.cpp e o IndiceSecundario.cpp, já descritos anteriormente, e o **seek2.cpp** (desenvolvido por Max), que utiliza a função **pesquisaChavArqIndSec** (implementada no arquivo IndiceSecundario.cpp) para realizar a busca.

O seek2.cpp possui algumas dependências: iostream, stdlib.h, string, IndiceSecundario.hpp, hash.hpp. Além disso, assim como o anterior, possui apenas a função main, responsável por realizar a execução do programa.