Documentação do Trabalho 2 de Banco de Dados

Equipe:
Agnaldo Brito
Pedro dos Santos
Victor Thury

1) Estrutura dos arquivos de dados e índices:
    - Hash: cada bucket tem 2 blocos e cada bloco tem 4096 bytes
        - O arquivo de dados organizado por hash possui um número de buckets igual ao macro LARGE_PRIME definido no definitions.h. Esse número é um primo grande para evitar colisões.

    - B+ tree para índice primário:
        - Cada nó possui 4096 bytes para caber em um bloco.
        - Cada nó possui dois vetores. Um para guardar os filhos de um nó (que podem ser ponteiros para o arquivo de dados ou para o arquivo de índice) e um para guardar as chaves.
        - Cada nó também possui um *booleano* que informa se ele é um nó folha ou não.

    - B+ tree para índice secundário:
        - A estrutura interna é semelhante a do índice primário (devido ao uso de templates), porém os vetores e a ordem são menores.

2) Quais fontes formam cada programa:
    - Upload:
        - Btree/btree.h
        - B-tree/secondary.h
        - hashFile.h
        - definitions.h

    - findrec:
        - definitions.h
        - hashFile.h

    - seek1
        - btree.h
        - definitions.h

    - seek2
        - btree.h
        - definitions.h

3) Funções de cada fonte:
    - A função dos src/ é ter os arquivos de hashing e realizar o upload e o findrec e além disso, possui a pasta da b+ tree.
    - A função do src/B-tree/ é fazer o upload da B+ tree (pode ser para arquivo de índice primário e secundário) além de também realizar o seek1 e seek2.

4) Contribuição de cada integrante:

Pedro foi responsável por: elaborar o parser e a parte que engloba as funções relacionadas ao hashing, tais como upload no arquivo de dados e findrec

Agnaldo e Victor trabalharam na elaboração da b+ tree, fazendo tanto para arquivo de índice primário e secundário. Agnaldo teve foco em fazer para arquivo de índice primário e no seek1, enquanto Victor focou em índice secundário e no seek2.

5) Papel de cada função:

A maioria das funções da B+ tree possuem nome autodescritivo, como “inserir_no”, então vamos focar nas funções que acreditamos de um pouco mais de descrição

Na B+ tree:

**buscar_chave_maior**: Função auxiliar para encontrar a posição da primeira chave maior ou igual a uma chave de referência usada para achar a posição na qual uma nova chave deve ser inserida.

**deslocar_chaves**: Função auxiliar: move todas as chaves a partir de uma certa posição, juntamente com seus ponteiros direitos, criando espaço para inserir novas chaves.

**inserir**: Possui duas versões, uma é chamada pelo usuário, outra é sobrecarregada com argumentos da inserção em árvore: privada, não pode ser utilizada pelo usuário. A versão sobrecarrega caminha a árvore recursivamente sempre verificando se chegou em um nó folha, ao achar um o valor é inserido e logo depois é verificado se é necessário fazer uma divisão do nó. Como a função é recursiva todos os nós visitados passam pela verificação se é necessário fazer a divisão implementada pela função **divide_no**.

**divide_no:** Função que cria divide o nó em 2 e atualiza a referência do nó pai. Para isto é criado um novo nó que possuem os dados do nó original da metade até o final fazendo assim a divisão, depois o nó original é atualizado para ter apenas os dados do início até o meio, e no final atualizamos as referências para o nó pai, incluindo os ponteiros e uma chave de referência. Também nesta função é verificado se estamos dividindo o nó raiz e se sim criamos um nó que assumirá a referência de raiz.Função que cria divide o nó em 2 e atualiza a referência do nó pai. Para isto é criado um novo nó que possuem os dados do nó original da metade até o final fazendo assim a divisão, depois o nó original é atualizado para ter apenas os dados do início até o meio, e no final atualizamos as referências para o nó pai, incluindo os ponteiros e uma chave de referência. Também nesta função é verificado se estamos dividindo o nó raiz e se sim criamos um nó que assumirá a referência de raiz.

**criar_no**: Criar o nó da arvore b+. Os ponteiros são inicializados com uma representação de vazio, no caso -1, e é passado como parâmetro se esse nó deve ser considerado um nó folha ou não.

No hash:

Em hashFile -

**calculateHash**: realiza o cálculo do hash para definir onde irá realizar a inserção.

**commitInsertion**: garante que realizou a escrita no disco

**insertItem**: realiza a inserção do item conforme a função de hash

**getLineFromBlock**: Recebe uma chave (no caso o id) e encontra o bucket em que ela está com a função de hash. Após isso, escaneia os blocos de entrada e retorna a linha inteira se ela exisitr.

Em Block -

**insertItem**: realiza inserção do item no bloco.

**getItem**: recupera o item no bloco

**Como rodar:**

1. Primeiro é necessário unzipar o arquivo de entrada, então siga os seguintes comandos:
    a. cd tp2
    b. cd data
    c. unzip entrada.zip
    d. cd ..

2. Agora será feita a montagem do arquivo de hash e das árvores com o comando abaixo. **ATENÇÃO:** Este comando pode demorar **bastante** para rodar. E tambem é possivel mudar o arquivo de entrada mudando o valor da variável *data* lembrando que o arquivo CSV de entre precisa estar no diretório *data.*
    a. make all data=entrada.csv
3. Para rodar o *findrec* basta rodar o seguinte comando:
    a. make find id=<id_a_ser_buscado>
4. Para rodar o *seek1* basta rodar o seguinte comando:
    a. make seek1 <id_a_ser_buscado>
5. Para rodar o *seek2* é necessário rodar os seguintes comandos:
    a. make createSeek2
    b. ./seek2 <titulo_a_ser_buscado>

**ATENÇÃO**: Caso queira rodar um arquivo de entrada diferente, no caso um novo hash e novas árvores, será necessário rodar o comando “make clean” e limpar o diretório *data* e

deixar apenas os arquivos **.csv** pois neste diretório são armazenadas o arquivo de dado do hash e os arquivos de índice das árvores!