Documentação do Trabalho Prático 1

- Timóteo Fonseca *21553687*
- Oscar Othon *21551563*

Arquivos de dados e índices

Eis os formatos dos arquivos gerados pelo programa. Todos se baseiam na unidade de `bloco`, que neste programa foi definido como consistindo em um bloco de 4096 bytes representando um setor da unidade de armazenamento de dados.

Arquivo de hashing

Começa sempre com um bloco contendo o cabeçalho do arquivo. Neste cabeçalho contém o tamanho do arquivo em blocos, contando o cabeçalho.

O arquivo usa a técnica do hashing perfeito, em que existe uma posição pré-definida para cada id de registro. No caso, cada posição é um bloco reservado para este registro. É possível que um bloco não esteja ocupado - por exemplo, havendo um registro de id 1 e outro de 3, mas nenhum de id 2, a posição para o bloco de id 2 continuará existindo entre eles mesmo não estando ocupada. Os registros têm um campo que indicam se estão válidos ou não. Se um registro for lido de uma posição não ocupada, ele virá com o campo inválido.

Arquivos de índice

Começam sempre com um bloco contendo o cabeçalho do arquivo. Neste cabeçalho contém o endereço da raiz da árvore e a quantidade de blocos no arquivo, contando com o cabeçalho.

Cada bloco subsequente representa um nó da árvore.

Summary

Members	Descriptions
`define`BLOCK_SIZE	Definição do tamanho do Bloco.
`define`FILEPATH	Caminho do local onde os arquivos do banco de dados serão criados.
`define`ID_TREE_FILENAME	Nome do arquivo que guardará os índices do arquivo de índice primário.
`define`ID_TREE_FILEPATH	Caminho completo pro arquivo que guardará os índices do arquivo de índice primário.
`define`TITLE_TREE_FILENAME	Nome do arquivo que guardará os índices do arquivo de índice secundário.
`define`TITLE_TREE_FILEPATH	Caminho completo pro arquivo que guardará os índices do arquivo de índice secundário.

Members	Descriptions
define HASHFILE_FILENAME	Nome do arquivo de hashing que guardará os registros.
define HASHFILE_FILEPATH	Caminho completo pro arquivo de hashing.
define BTREE_ORDER	Macro para definição da ordem máxima do nó de um BTree, baseado no tipo de dados que ele guardará.
define ID_ORDER	Macro com a ordem máxima que um nó de id's pode ter sem ultrapassar o tamanho do bloco.
define TITLE_ORDER	Macro com a ordem máxima que um nó de títulos pode ter sem ultrapassar o tamanho do bloco.
define TITLE_CHAR_MAX	Tamanho máximo da cadeia de caracteres do título.
define AUTHORS_CHAR_MAX	Tamanho máximo da cadeia de caracteres dos nomes de autores.
define TIMESTAMP_CHAR_MAX	Tamanho máximo da cadeia de caracteres do timestamp.
define SNIPPET_CHAR_MAX	Tamanho máximo da cadeia de caracteres do resumo.
public static void readField(char * field,std::FILE * file,int fieldSize)	Lê a coluna de um registro a partir do arquivo CSV.
public static bool readEntry(Entry& e,std::FILE * file)	Lê por inteiro, campo a campo, um registro do arquivo CSV.
public static void printEntry(const Entry& e)	Imprime um registro campo por campo.
public void upload(const char * filePath)	Recebe um arquivo CSV e cria uma base de dados baseado nele.
public static void foundEntryMessage(const Entry& e,std::size_t blocksRead,std::size_t blockCount)	Função que mostra que um registro foi encontrado, quantos blocos foram lidos para encontrá-lo e quantos blocos o arquivo possui no momento.
public static bool findEntryAndPrint (std::FILE * hashfile,long offset,std::size_t blocksReadSoFar,std::size_t blockCount)	Função que acha um registro no arquivo e printa se a busca foi completa ou não, ou se ocorreu algum erro de leitura.
public void findrec(long id)	Busca no arquivo de hashing um registro a partir de seu ID.
public void seek1(long id)	Busca um registro a partir de seu ID usando o arquivo de índices primário.
public void seek2(const char * title)	Busca um registro a partir de seu título usando o arquivo de índices secundário.
public void upload(const char * filePath)	Recebe um arquivo CSV e cria uma base de dados baseado nele.
public void findrec(long id)	Busca no arquivo de hashing um registro a partir de seu ID.
public void seek1(long id)	Busca um registro a partir de seu ID usando o arquivo de índices primário.
public void seek2(const char * title)	Busca um registro a partir de seu título usando o arquivo de índices secundário.
public int main(int argc,char ** argv)	Entrada main do programa.

Members	Descriptions
class BTree	Classe de árvore B.
struct BTree::BNode	Nó da árvore B.
struct Entry	Registro básico do arquivo.
struct BTree::FileHeader	Dados do cabeçalho do arquivo.
struct HashfileHeader	Struct do cabeçalho do arquivo de hashing.
struct IdPointer	Struct auxiliar para guardar os índices primários.
struct BTree::OverflowResult	Registro auxiliar com o resultado de um overflow de inserção.
struct BTree::Statistics	Estrutura usada pela árvore para guardar métricas de sua execução.
struct TitlePointer	Struct auxiliar para guardar os índices secundários.
union Block	Union para leitura e escrita, em bloco, de tipos arbitrários.

Members

define BLOCK_SIZE

Definição do tamanho do Bloco.

O tamanho foi definido como sendo de tamanho 4096 por ser o tamanho do setor físico indicado através do comando `fdisk -l` na máquina do Timóteo, um Linux Mint 18.1 Cinnamon 64-bit.

define FILEPATH

Caminho do local onde os arquivos do banco de dados serão criados.

Este é um define auxiliar que é usado apenas na definição de outros defines.

Por padrão, os arquivos serã criados na mesma pasta que o executável.

define ID_TREE_FILENAME

Nome do arquivo que guardará os índices do arquivo de índice primário.

define ID_TREE_FILEPATH

Caminho completo pro arquivo que guardará os índices do arquivo de índice primário.

define TITLE_TREE_FILENAME

Nome do arquivo que guardará os índices do arquivo de índice secundário.

define TITLE_TREE_FILEPATH

Caminho completo pro arquivo que guardará os índices do arquivo de índice secundário.

defineHASHFILE_FILENAME

Nome do arquivo de hashing que guardará os registros.

defineHASHFILE_FILEPATH

Caminho completo pro arquivo de hashing.

defineBTREE_ORDER

Macro para definição da ordem máxima do nó de um BTree, baseado no tipo de dados que ele guardará.

O cálculo é feito a partir do tamanho de BLOCK_SIZE e determina a ordem máxima que o nó pode ter sem ultrapassar o tamanho do bloco, considerando que ele estará guardando dados de um tipo cujo tamanho em bytes é T_SIZE.

A fórmula do macro foi obtido a partir desta fórmula inicial:

```
BLOCK_SIZE = sizeof(long) + sizeof(bool) + sizeof(int) + (2 * M + 1) * sizeof(T) + (2 * M + 2) * sizeof(long)
```

Ele é baseado nos campos padrões da estrutura BNode, interna à classe BTree.

defineID_ORDER

Macro com a ordem máxima que um nó de id's pode ter sem ultrapassar o tamanho do bloco.

defineTITLE_ORDER

Macro com a ordem máxima que um nó de títulos pode ter sem ultrapassar o tamanho do bloco.

defineTITLE_CHAR_MAX

Tamanho máximo da cadeia de caracteres do título.

defineAUTHORS_CHAR_MAX

Tamanho máximo da cadeia de caracteres dos nomes de autores.

defineTIMESTAMP_CHAR_MAX

Tamanho máximo da cadeia de caracteres do timestamp.

defineSNIPPET_CHAR_MAX

Tamanho máximo da cadeia de caracteres do resumo.

public static voidreadField(char * field,std::FILE * file,int fieldSize)

Lê a coluna de um registro a partir do arquivo CSV.

Essa função é chamada para a leitura do arquivo. Nela são tratadas todas as possíveis exceções que podem ocorrer na leitura, essas exceções sao a presença no local incorreto de diversos símbolos descritos a seguir.

Os símbolos são: ; (ponto e vírgula), ``(contra-barra n), (contra-barra r), "(aspas), `NULL`.

No caso das aspas é necessário ser verificado se elas realmente se encontram em um local inválido, ou se elas são o final de uma coluna. Para isso fazemos uma etapa de verificação a mais nas aspas, nessa etapa verificamos se as aspas são seguidas de `\r`, `;`, ``, EOF. Se ela for seguida por algum desses significa que essas aspas estão demarcando o final de uma coluna.

Parameters

- `field` Ponteiro para o vetor de caracteres onde a coluna lida será guardada.
- `file` O arquivo sendo lido.
- `fieldSize` O tamanho máximo, em caracteres, que a string do campo pode ter.

Oscar Othon

`public static boolreadEntry(Entry& e,std::FILE * file)`

Lê por inteiro, campo a campo, um registro do arquivo CSV.

Recebendo o arquivo pelo parâmetro std::FILE *file, essa função percorre por completa uma linha de instância de registro no arquivo, dividindo esse artigo em seus campos de título, ano, autor, citações, atualização e snippet. Cada um desses campos é separado ou por meio da utilização da função readField(), ou utilizando-se fscanf().

Parameters

- `e` O registro do registro a ter seus campos lidos.
- `file` O arquivo sendo lido.

Returns

Se o registro foi lido com sucesso ou não, dará falso ao chegar no final do arquivo pois não haverão mais registros para ler.

Oscar Othon

`public static voidprintEntry(constEntry& e)`

Imprime um registro campo por campo.

Parameters

- `e` O registro a ser impresso.

Oscar Othon

public voidupload(const char * filePath)

Recebe um arquivo CSV e cria uma base de dados baseado nele.

Faz a carga inicial da massa de testes para seu banco de dados e cria um arquivo de dados organizado por hashing, um arquivo de índice primário usando B-Tree e outro arquivo de índice de índice secundário usando B-Tree.

A função upload faz várias verificações antes de começar a leitura do arquivo. Os detalhes são exibidos em tela. Ela precisa verificar se é possível criar cada um dos arquivos necessários no local do executável, e apenas caso tudo esteja correto ela procede à leitura dos registros.

No fim de uma execução bem sucedida o programa terá três arquivos gerados, todos com extensão `.bin`. Um deles será o arquivo de hashing, o outro o arquivo de índices primários, e por último o de índices secundários.

É esperado que essa função seja chamada antes de todas as demais.

Apenas arquivos em formato CSV serão aceitos.

Parameters

- `filePath` Caminho pro arquivo CSV a ser lido.

Oscar Othon

public static voidfoundEntryMessage(constEntry& e,std::size_t blocksRead,std::size_t blockCount)

Função que mostra que um registro foi encontrado, quantos blocos foram lidos para encontrá-lo e quantos blocos o arquivo possui no momento.

Parameters

- `e` Um registro do arquivo.
- `blocksRead` O número de blocos lidos até chegar no registro.
- `blockCount` A contagem do número total de blocos no arquivo.

Oscar Othon

public static boolfindEntryAndPrint(std::FILE * hashfile,long offset,std::size_t blocksReadSoFar,std::size_t blockCount)

Função que acha um registro no arquivo e printa se a busca foi completa ou não, ou se ocorreu algum erro de leitura.

Essa função recebe o arquivo já aberto, se dirige até o offset e, então, lê o registro.

Parameters

- `*hashfile` Ponteiro do arquivo pro arquivo de hashing.
- `offset` O offset do registro no arquivo.
- `blocksReadSoFar` Quantos blocos foram lidos até o momento.
- `blockCount` A contagem do número total de blocos no arquivo.

Oscar Othon

`public voidfindrec(long id)`

Busca no arquivo de hashing um registro a partir de seu ID.

O arquivo de hashing usa o hashing perfeito, então a função apenas calcula o offset do arquivo baseado em seu ID e faz a leitura do bloco em que ele se encontra.

Caso não haja nenhum registro válido na posição encontrada através do ID, a função informará.

Parameters

- `id` O ID do registro a ser buscado.

Oscar Othon

`public voidseek1(long id)`

Busca um registro a partir de seu ID usando o arquivo de índices primário.

O seek1 usa o arquivo de índices primário, organizado em índices de uma árvore-B, para encontrar o registro buscado.

Caso ele não seja encontrado, a função informa.

A função costuma sempre precisar ler mais blocos do que a função findrec, porque o arquivo de hashing perfeito tem acesso em O(1).

Parameters

- `id` O ID do registro a ser buscado.

Oscar Othon

`public voidseek2(const char * title)`

Busca um registro a partir de seu título usando o arquivo de índices secundário.

Já que o título não é uma chave, e sim um campo qualquer, não existe ordenação entre eles no arquivo de hashing. Então é necessário utilizarmos uma árvore-B para poder fazermos dessa busca algo mais plausível em questão de tempo. Sem a árvore teríamos que fazer uma busca linear pelo arquivo de hashing.

Caso o registro não possa ser encontrado, a função informará nos resultados.

Parameters

- `title` Título pelo qual o registro será buscado.

Oscar Othon

`public void upload(const char * filePath)`

Recebe um arquivo CSV e cria uma base de dados baseado nele.

Faz a carga inicial da massa de testes para seu banco de dados e cria um arquivo de dados organizado por hashing, um arquivo de índice primário usando B-Tree e outro arquivo de índice de índice secundário usando B-Tree.

A função upload faz várias verificações antes de começar a leitura do arquivo. Os detalhes são exibidos em tela. Ela precisa verificar se é possível criar cada um dos arquivos necessários no local do executável, e apenas caso tudo esteja correto ela procede à leitura dos registros.

No fim de uma execução bem sucedida o programa terá três arquivos gerados, todos com extensão `.bin`. Um deles será o arquivo de hashing, o outro o arquivo de índices primários, e por último o de índices secundários.

É esperado que essa função seja chamada antes de todas as demais.

Apenas arquivos em formato CSV serão aceitos.

Parameters

- `filePath` Caminho pro arquivo CSV a ser lido.

Oscar Othon

`public void findrec(long id)`

Busca no arquivo de hashing um registro a partir de seu ID.

O arquivo de hashing usa o hashing perfeito, então a função apenas calcula o offset do arquivo baseado em seu ID e faz a leitura do bloco em que ele se encontra.

Caso não haja nenhum registro válido na posição encontrada através do ID, a função informará.

Parameters

- `id` O ID do registro a ser buscado.

Oscar Othon

`public void seek1(long id)`

Busca um registro a partir de seu ID usando o arquivo de índices primário.

O seek1 usa o arquivo de índices primário, organizado em índices de uma árvore-B, para encontrar o registro buscado.

Caso ele não seja encontrado, a função informa.

A função costuma sempre precisar ler mais blocos do que a função findrec, porque o arquivo de hashing perfeito tem acesso em O(1).

Parameters

- `id` O ID do registro a ser buscado.

Oscar Othon

`public void seek2(const char * title)`

Busca um registro a partir de seu título usando o arquivo de índices secundário.

Já que o título não é uma chave, e sim um campo qualquer, não existe ordenação entre eles no arquivo de hashing. Então é necessário utilizarmos uma árvore-B para poder fazermos dessa busca algo mais plausível em questão de tempo. Sem a árvore teríamos que fazer uma busca linear pelo arquivo de hashing.

Caso o registro não possa ser encontrado, a função informará nos resultados.

Parameters

- `title` Título pelo qual o registro será buscado.

Oscar Othon

`public int main(int argc,char ** argv)`

Entrada main do programa.

O procedimento main representa um programa que simplesmente recebe argumentos e chama as funções apropriadas para lidar com eles.

Sintaxe do uso:

```
$ <nome-do-executavel> upload <caminho-do-arquivo : string>
$ <nome-do-executavel> findrec <id : inteiro>
$ <nome-do-executavel> seek1 <id : inteiro>
$ <nome-do-executavel> seek2 <titulo : string>
```

Detalhe importante sobre o comando `upload` e seu argumento `caminho-do-arquivo`: o caminho do arquivo a subir precisa estar no formato CSV com as colunas pré-definidas de cada registro (seguindo o formato da estrutura Entry, salvo o campo `valid` dela). Qualquer outro formato resultará em comportamento indefinido do programa.

Detalhe importante sobre o comando `seek2` e seu argumento `titulo`: `titulo` não é uma string entre aspas (`"`). Se um título possuir espaços, espaços devem ser fornecidos com a contrabarra.

Exemplo:

```
$ bd seek2 Título\ com\ espaços
```

Parameters

- `argc` Quantidade de argumentos.
- `argv` Valores em string dos argumentos.

Timóteo Fonseca

class `BTree`

Classe de árvore B.

Classe de árvore B multi-uso usada para guardar índices e títulos no trabalho.

O parâmetro de template TKey determina o tipo de dado que será guardado nos nós da árvore. TKey deve ser um tipo POD (Plain Old Data type) para poder ser sequenciado no arquivo quando o nó for escrito.

O parâmetro de template M determina a ordem da árvore. Em condições normais, um nó pode ter no máximo 2M dados e 2M+1 apontadores de nó. Em estado de overflow, ele pode temporariamente ter 2M+1 dados e 2M+2 apontadores de nó.

Antes de usar uma BTree deve sempre se chamar o método load (em caso de apenas leitura) ou create (em caso de escrita de um novo arquivo).

No fim da inserção, o método finishInsertions() deve ser chamado para a quantidade de blocos do arquivo ser atualizada no cabeçalho.

Exemplo de uso:

```
BTree<int, 2> arvore;
arvore.create("nomedoarquivo.bin");
arvore.insert(1);
arvore.finishInsertions();

arvore.load("nomedoarquivo.bin");
auto valor = arvore.seek(1);
if (valor) f(*valor); // Fazer algo com o valor, se encontrado.
```

Summary

Members	Descriptions
`publicBTree()`	Construtor padrão.
`public~BTree()`	Destrutor padrão.
`public boolcreate(const char * filepath)`	Método de inicialização da árvore, para escrita.
`public boolload(const char * filepath)`	Método de inicialização da árvore, para leitura.
`public voidinsert(const TKey & key)`	Insere um dado dentro da árvore.
`public std::unique_ptr< TKey >seek(const TKey & key)`	Busca um dado que seja equivalente ao dado fornecido.
`publicStatisticsgetStatistics(bool includeFileBlockCount) const`	Retorna as estatísticas da árvore até então.
`public voidresetStatistics()`	Atribui valor 0 a todos os campos das estatísticas atuais.
`public voidfinishInsertions()`	Atualiza o cabeçalho com o total de blocos no arquivo.

Members

`publicBTree()`

Construtor padrão.

Inicializa com valores 0 as estatísticas e nulifica o ponteiro de arquivo da BTree.

Timóteo Fonseca

`public~BTree()`

Destrutor padrão.

Fecha o arquivo se ele estiver aberto.

`public boolcreate(const char * filepath)`

Método de inicialização da árvore, para escrita.

Deve sempre ser chamado antes de começar a inserção na árvore, mas também permite a leitura.

Se já houver um arquivo em filepath, ele será sobreescrito.

O arquivo novo será criado com um cabeçalho e um nó raiz vazio.

Parameters

- `filepath` Caminho do arquivo onde os dados da árvore serão escritos.

Returns

Se foi possível criar o arquivo em filepath.

Timóteo Fonseca

`public boolload(const char * filepath)`

Método de inicialização da árvore, para leitura.

Deve sempre ser chamado antes de começar a leitura da árvore. Inserções não serão possíveis.

O arquivo deve existir previamente, de preferência criado por outra execução da BTree após uma chamada de create().

Parameters

- `filepath` Caminho pro arquivo onde se encontram os dados da árvore.

Returns

Se foi possível abrir o arquivo em filepath.

Timóteo Fonseca

`public voidinsert(const TKey & key)`

Insere um dado dentro da árvore.

Parameters

- `key` O dado a ser inserido.

Timóteo Fonseca

`public std::unique_ptr< TKey >seek(const TKey & key)`

Busca um dado que seja equivalente ao dado fornecido.

Não há garantias que o dado realmente será buscado. Caso não seja encontrado, a função retorna um ponteiro nulo.

Parameters

- `key` O dado que se está buscando.

Returns

Ponteiro com o dado ou nulo.

Timóteo Fonseca

public Statistics getStatistics(bool includeFileBlockCount) const

Retorna as estatísticas da árvore até então.

A inclusão da quantidade de blocos no arquivo, nas estatísticas, é opcional pois a árvore vai ter que fazer uma leitura do arquivo para poder obter este dado, o que é uma operação custosa.

Parameters

- `includeFileBlockCount` Se deve incluir (true) ou não (false) a quantidade de blocos do arquivo nas estatísticas.

Returns

Estatísticas.

Timóteo Fonseca

public void resetStatistics()

Atribui valor 0 a todos os campos das estatísticas atuais.

Timóteo Fonseca

public void finishInsertions()

Atualiza o cabeçalho com o total de blocos no arquivo.

Muito importante de se usar no final da utilização de uma árvore após ela ser inicializada com create().

Timóteo Fonseca

struct BTree::BNode

Nó da árvore B.

Summary

Members	Descriptions
`public long offset`	Endereço no disco.
`public bool isLeaf`	Booleano indicando se o nó é folha (true) ou nó interno (false).
`public std::size_t keysCount`	Quantidade de dados dentro do nó, no momento.
`public TKey keys`	Os dados do nó.
`public long children`	Os apontadores de nó.
`public void initialize(bool isLeaf,int keysCount)`	Inicializa o nó.

Members	Descriptions
`public boolisFull() const`	Verifica se o nó está em sua capacidade máxima (acima de 2M dados).
`public std::unique_ptr< TKey >seek(const TKey & key,BTree& tree) const`	Busca um dado que seja equivalente ao dado fornecido.

Members

`public longoffset`

Endereço no disco.

O ideal é que o nó, antes de ser escrito no registro pela primeira vez, esteja com o offset de valor -1. Após a primeira escrita, o offset passa a realmente possuir seu endereço no disco.

`public boolisLeaf`

Booleano indicando se o nó é folha (true) ou nó interno (false).

`public std::size_tkeysCount`

Quantidade de dados dentro do nó, no momento.

`public TKeykeys`

Os dados do nó.

Na prática, um nó só costuma utilizar as 2M primeiras posições deste vetor. O espaço adicional de 1 dado é usado temporariamente para lidar com overflows.

`public longchildren`

Os apontadores de nó.

Na prática, um nó só costuma utilizar as 2M+1 primeiras posições deste vetor. O espaço adicional de 1 dado é usado temporariamente para lidar com overflows.

`public voidinitialize(bool isLeaf,int keysCount)`

Inicializa o nó.

O valor do campo offset do nó não é fornecido: ele é sempre inicializado como -1, para permitir que se saiba se ele já foi escrito no arquivo ou não.

Se um nó não for lido através de um readFromDisk da BTree, é obrigatório que esse método seja chamado, pois ele funciona como um construtor do BNode.

A inicialização foi implementada como um método ao invés de um construtor pois é necessário que o BNode seja uma classe de tipo POD (Plain Old Data type) para poder ser lido e escrito no arquivo, além de para também poder ser usado como argumento de template para o Block.

Parameters

- `isLeaf` Se o nó vai ser inicializado como folha (true) ou não (false).
- `keysCount` Quantidade inicial de dados no nó.

Timóteo Fonseca

`public boolisFull() const`

Verifica se o nó está em sua capacidade máxima (acima de 2M dados).

Timóteo Fonseca

`public std::unique_ptr< TKey >seek(const TKey & key,BTree& tree) const`

Busca um dado que seja equivalente ao dado fornecido.

Realiza a busca propriamente dita pelo dado, usando a instância de tree passado por parâmetro para poder realizar leituras no arquivo.

Retorna um ponteiro nulo se não encontrar o dado.

Returns

Ponteiro com o dado ou nulo.

Timóteo Fonseca

struct `Entry`

Registro básico do arquivo.

Summary

Members	Descriptions
`public boolvalid`	Se o registro constitui um registro válido no arquivo.
`public intid`	Identificador do registro.
`public chartitle`	Título do livro.
`public intyear`	Ano de publicação do livro.
`public charauthors`	Autores do livro.
`public intcitations`	Quantidade de citações do livro.
`public charupdateTimestamp`	Timestamp de atualização do livro.
`public charsnippet`	Resumo do livro.

Members

`public boolvalid`

Se o registro constitui um registro válido no arquivo.

public intid

Identificador do registro.

public chartitle

Título do livro.

public intyear

Ano de publicação do livro.

public charauthors

Autores do livro.

public intcitations

Quantidade de citações do livro.

public charupdateTimestamp

Timestamp de atualização do livro.

public charsnippet

Resumo do livro.

struct BTree::FileHeader

Dados do cabeçalho do arquivo.

Summary

Members	Descriptions
public longrootAddress	
public unsigned intblockCount	

Members

public longrootAddress

public unsigned intblockCount

struct HashfileHeader

Struct do cabeçalho do arquivo de hashing.

Summary

Members	Descriptions
`public int blockCount`	

Members

`public int blockCount`

struct `IdPointer`

Struct auxiliar para guardar os índices primários.

Summary

Members	Descriptions
`public int id`	Id do registro.
`public long offset`	Offset do registro no arquivo de hashing.
`public inline bool operator<(const IdPointer& that) const`	Comparador de < para o struct auxiliar, assim poderá ser usado na BTree.

Members

`public int id`

Id do registro.

`public long offset`

Offset do registro no arquivo de hashing.

`public inline bool operator<(const IdPointer& that) const`

Comparador de < para o struct auxiliar, assim poderá ser usado na BTree.

struct `BTree::OverflowResult`

Registro auxiliar com o resultado de um overflow de inserção.

Summary

Members	Descriptions
`public TKey middle`	O dado que, após o split de nós, é o valor do meio e deverá ser inserido no nó pai.
`public long rightNode`	Offset de nó que vai precisar estar no apontador à direita de onde `middle` for inserido, no nó pai.

Members

public TKeymiddle

O dado que, após o split de nós, é o valor do meio e deverá ser inserido no nó pai.

public longrightNode

Offset de nó que vai precisar estar no apontador à direita de onde `middle` for inserido, no nó pai.

struct BTree::Statistics

Estrutura usada pela árvore para guardar métricas de sua execução.

Timóteo Fonseca

Summary

Members	Descriptions
`public unsigned int`blocksRead	Quantidade de blocos lidos.
`public unsigned int`blocksCreated	Quantidade de blocos criados.
`public unsigned int`blocksInDisk	Quantidade de blocos que já foram inseridos em disco.

Members

public unsigned intblocksRead

Quantidade de blocos lidos.

public unsigned intblocksCreated

Quantidade de blocos criados.

public unsigned intblocksInDisk

Quantidade de blocos que já foram inseridos em disco.

struct TitlePointer

Struct auxiliar para guardar os índices secundários.

Summary

Members	Descriptions
`public char`title	String com o título do registro.
`public long`offset	Offset do registro no arquivo de hashing.

Members	Descriptions
public inline booloperator<(const TitlePointer& that) const	Comparador de < para o struct auxiliar, assim poderá ser usado na BTree.

Members

public chartitle

String com o título do registro.

public longoffset

Offset do registro no arquivo de hashing.

public inline booloperator<(constTitlePointer& that) const

Comparador de < para o struct auxiliar, assim poderá ser usado na BTree.