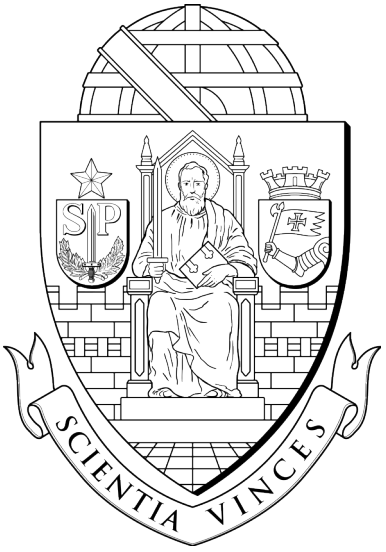

Universidade de São Paulo

Algorítmos e estrutura de dados 2

B-tree

Professor:
Leonardo Tortoro Pereira

Monitor:
Henrique Gomes Zanin

Sumário

Lista de Códigos

1 Resumo

O seguinte trabalho propõem a implementação de uma árvore B para indexação de registros de tamanho fixo. A implementação da árvore B deve armazenar nos nós os *relative record numbers* definidos com o tipo *long*, além de variáveis auxiliares de controle para manutenção das páginas de disco. Todos os arquivos gerados devem estar no formato binário, evitando desperdício de espaço com tipos primitivos.

2 Introdução

Originalmente concebida por R. Bayer e E. McCreight em uma competição organizada pela Boeing[1], a árvore B tem como objetivo prover uma estrutura de dados que otimize a inserção e consulta de registros por meio da paginação dos nós da árvore. A paginação do nó permite reduzir o numero de *Seeks* no disco rígido para a recuperar a chave de um registro. Neste trabalho você deve implementar em linguagem C as operações de busca e inserção em árvore B em sua forma mais encontrada na literatura, com as chaves dos registros nos nós intermediários.

3 Implementação

3.1 Registros

Os campos do registro estão definidos na Tabela 1.Não há muito segredo, reutilize as funções codificadas no exercício de índices primários.

Tabela 1: Campos do registro

Campo	**Tipo de dado**
nUSP	int
Nome	char[x]
Sobrenome	char[x]
Curso	char[x]
Nota	float

3.2 B-tree

A cada inserção de um novo registro a chave primária deverá ser consultada na B-tree para evitar a inserção de chaves duplicadas. Novas chaves são adicionadas

em nós folhas, caso o nó supere o limite máximo de chaves as operação de divisão e promoção são chamadas para garantir o balanceamento da árvore. O aquivo da árvore poderá ser organizado da seguinte forma:

Header
[RRN da Raiz][Espaço livre]

Nós
[Chaves primárias][RRNs dos registros][Filhos][Numero de chaves][Flag isLeaf][Espaço livre]

A escolha de deixar uma página disponível para o header evita o acréscimo de espaços livres nas demais páginas que contém os nós. Caso contrário, de cada nova página deveriam ser retirados 8 bytes de espaço útil para manter todos os nós com o mesmo tamanho. Uma página para o header também viabiliza implementações de variáveis de controle caso haja necessidade.
Cada nó, quando carregado na memória, pode ser armazenado em uma TAD.
A operação de split consiste na divisão do nó quando este ultrapassa o numero máximo de registros definidos previamente em uma constante. Nesse caso é criado um novo nó contendo a metade superior do nó dividido e a menor das maiores chaves estabelecidas no novo nó é selecionada para a promoção. Na operação de promoção as chaves promovidas dos nós inferiores são adicionadas nos nós internos e caso superem o numero máximo são divididos e uma nova chave é promovida. O máximo de promoções e divisões que uma árvore pode experimentar é igual a sua altura. Todas as inserções nos nós devem ser feitas de forma ordenada, dispensando algoritmos de ordenação.
A quantidade de registros em cada nó pode variar de acordo com o tamanho da pagina do sistema operacional. Para defini-la você deve levar em consideração que há 2 campos que não variam com a quantidade de registros, são eles: o numero de chaves e a *flag* que indica se o nó é uma folha ou um nó intermediário, totalizando 3 bytes fixos. Os demais campos variam de acordo com a equação (1), o espaço livre é inversamente proporcional ao numero de chaves e é dado pela equação (2):

$$bytes = (MAXKEYS * 4) + (MAXKEYS * 8) + ((MAXKEYS + 1) * 8) + 3 \quad (1)$$

$$Livre = TamPag - bytes \quad (2)$$

Onde TamPag é o tamanho da página definida na formatação do disco, nós usaremos **4096** bytes nesse trabalho

Exemplo de header file para a Btree:

Código 1: btree.h

```
#ifndef __BTREE__
#define __BTREE__

#include <utils.h>

#define PAGESIZE 4096
#define TREE_HEADER PAGESIZE
#define MAXKEYS 204
#define AUX_FIELDS_SIZE_ON_PAGE (2+1) /*number of keys and "is leaf" bool*/
#define FREE_SPACE_ON_PAGE (PAGESIZE - ((MAXKEYS*4)+(MAXKEYS*8)+((MAXKEYS+1)*8)+3))

typedef struct record{
    int key;
    long recordRRN;
}record;

typedef struct page{
    record *records;
    long *childs;
    short numberOfKeys;
    BOOL isLeaf;
} btPage;

typedef struct promotedkey{
    int key;
    long recordRRN;
    long childs[2];
} promotedKey;

btPage *getOrCreateRoot(FILE *);
btPage *getRoot(FILE *);
Errors bTreeInsert(record *, btPage *, FILE *);
long bTreeSelect(btPage *, int, FILE *);
#endif
```

4 Operacionalização do banco de dados

4.0.1 Insert

Para utilizar o comando "insert", os dados seguem um padrão para serem inseridos. Para strings, basta colocar a string dentro de aspas, por exemplo "gabriel", para inteiros basta o numero como 4, para numeros racionais deve-se colocar os decimais após um "."da seguinte maneira: 9.25. Concatenando tudo na ordem descrita

na Tabela 1, segue um breve exemplo.

insert 11289521,"Gabriel","Marin","bsi",8.25

Caso a chave já esteja cadastrada, o comando deve retornar **"O Registro ja existe!"**

4.0.2 Search

Na a instrução "search"é necessário passar apenas a chave que deseja buscar concatenada com o próprio comando. Segue o exemplo:

search 11289521

Caso a chave não exista, o comando deve retornar **"Registro nao encontrado!**"

4.0.3 Update

O comando update deve receber um registro com chave já cadastrada. a estrutura é idêntica ao insert. Caso a chave não exista, o comando deve retornar **"Registro nao encontrado!"**

update 11289521,"Onofre","Martins","bsi",8.25

4.0.4 Exit

Simples comando para sair do programa, segue exemplo:

exit

5 Saída esperada

O seu programa deve exibir os registros da seguinte forma:

Código 2: Saida esperada

```
1 --------------------------------
2 nUSP: 7
3 Nome: Anthony
4 Sobrenome: Fernando
5 Curso: Sistemas de Informacao
6 Nota: 9.10
7 --------------------------------
8 --------------------------------
9 nUSP: 2
10 Nome: Vincent
11 Sobrenome: James
12 Curso: Sistemas de Informacao
13 Nota: 9.10
14 --------------------------------
```

Referências

[1] Michael J Folk and Bill Zoellick. *File structures*, volume 2. Addison-Wesley Reading, 1992.