**Árvore B/B+** : pensando apenas na memória principal não há sentido implementar este tipo de árvore, porém quando implementada para acesso ao disco, faz-se mais sentido pelo custo que há quando faz-se leitura / escrita em disco.

Dado (composto por):

1 - Chave

2 - Informações

Árvore:

1 - Ponteiro raiz (pont. p/ nó)

2 - Ordem

Nó:

1 - Vetor de dados

2 - Vetor de ponteiros

3 - Contador de nº de chaves

<https://stuff.mit.edu/afs/sipb/user/gamadrid/nscript/btree.c>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dado.h

---------------------------------------------------------------------------------------------------------------------------------

#ifndef \_DADOS\_H\_

#define \_DADOS\_H\_

struct dado{

int chave;

void \*info;

};

#endif

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

arvoreb.h

---------------------------------------------------------------------------------------------------------------------------------

// definição da árvore em si

// #ifndef....

#include no.h

struct arvoreB {

struct no \*raiz; // ponteiro para a raiz

int ordem; // ordem desta arvore

};

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

no.h

--------------------------------------------------------------------------------

// definição dos nos a arvore B

//#//#ifndef...

#include "dado.h"

#include "no.h"

#define ORDEM 5

#define MAXIMO\_CHAVES (ORDEM - 1)

#define MINIMO\_CHAVES ((ORDEM - 1)/2)

// Prototipos

int noCheio(struct no \*);

int violacaoNo(struct no \*);

int localizaChave(struct no \*, int);

struct no {

struct dado dado[MAXIMO\_CHAVES]; // dados armazenados no nó

struct no \*pont[ORDEM]; // ponteiros para filhos

int cont; // numero de dados validos

};

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

no.c

--------------------------------------------------------------------------------

#include "no.h"

/\*

NO\_CHEIO

Descrição: Verifica se um no esta cheio ou não

Entrada: ponteiro para um no

Saida: Verdadeiro se estiver cheio; falso caso contrario

\*/

int noCheio(struct no \*no){

return no -> cont == MAXIMO\_CHAVES

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

arvoreb.c

--------------------------------------------------------------------------------

#include "arvoreb.h"

/\*

INSIRA

Descrição: Insere, sem repetição, um novo dado na árvore, organizando segundo sua chave

Entrada: ponteiro para o nó raiz e o dado

Saída: o nó raiz modificado (por referência) quando for a primeira inserção

Retorno: Verdadeiro, se houve a inserção. Falso, caso a chave já exista

\*/

int insira(struct no \*\*raiz, struct novoDado){

if (\*raiz == 0) { // primeira inserção

\*raiz = malloc(sizeof(struct no));

\*raiz->cont = 1;

\*raiz->dado[0] = novoDado;

\*raiz->pont[0] = \*raiz->pont[1] = 0;

}

else { // todas as outras inserções

status = busqueEInsira(\*raiz, novoDado, &houvePromocao,

&dadoPromovido, &novoNo);

if(houvePromocao){

novaRaiz = malloc(sizeof(struct no));

novaRaiz->cont = 1;

novaRaiz->pont[0] = \*raiz;

novaRaiz->pont[1] = novoNo;

\*raiz = novaRaiz;

}

}

int busqueEInsira(struct no\* noAtual, struct dado novoDado, int \*houvePromocao, struct dado\* dadoPromovido, struct no\* novoNo){

pos = localizaNo(noAtual, novoDado); // verifica existencia no nó

if (pos != -1){

return 0; // Falso: a chave já existe

}

else {

if(ehFolha(noAtual)){

// inserção na folha

}

else{

// continua a inserção recursivamente

}

}

}

}