Relatório - Trabalho sobre Árvore B

Nome: Thiago Pereira Camerato

Matrícula: 2212580

Introdução:

Este relatório descreve o trabalho realizado na implementação de uma árvore B em linguagem C. A árvore B é uma estrutura de dados que permite a inserção, exclusão e busca eficiente de elementos, mantendo a ordem das chaves. O objetivo deste trabalho foi criar um programa que realiza a inserção de 21 chaves em uma árvore B de ordem 5 e permite a impressão da árvore em ordem simétrica, bem como a busca e impressão de chaves em um intervalo específico.

Estrutura do Programa:

O programa é composto por diversas funções para criar, inserir e imprimir a árvore B. Aqui estão as funções principais implementadas:

```c
#define MAX 4
#define MIN 2

typedef struct no t_no;

struct no {
    int ndesc;
    int chave[MAX];
    t_no *ramo[MAX + 1];
};
```

1. `cria_no()`: Esta função aloca memória para um novo nó da árvore B e inicializa seus atributos.

```c
t_no *cria_no(void) {
    t_no *novo = (t_no *)malloc(sizeof(t_no));
    novo->ndesc = 0;
    for (int i = 0; i <= MAX; i++) {
        novo->ramo[i] = NULL;
    }
    return novo;
}
```

2. `imprime_intervalo(t_no *arv, int lim_inf, int lim_sup)`: Esta função percorre a árvore B em ordem simétrica e imprime todas as chaves no intervalo (lim_inf, lim_sup).

```c
void imprime_intervalo(t_no *arv, int lim_inf, int lim_sup) {
    if (arv) {
        int i;
        for (i = 0; i < arv->ndesc; i++) {
            if (arv->chave[i] >= lim_inf) {
                imprime_intervalo(arv->ramo[i], lim_inf, lim_sup);
            }

            if (arv->chave[i] >= lim_inf && arv->chave[i] <= lim_sup) {
                printf("%d ", arv->chave[i]);
            }

            if (arv->chave[i] <= lim_sup) {
                imprime_intervalo(arv->ramo[i + 1], lim_inf, lim_sup);
            }
        }
    }
}
```

3. `insere_chave(t_no *raiz, int chave)`: Esta função insere uma chave na árvore B de acordo com as regras da estrutura. Se o número de chaves em um nó atingir o limite, ocorre a divisão do nó para manter a propriedade da árvore B.

```c
t_no *insere_chave(t_no *raiz, int chave) {
    if (!raiz) {
        raiz = cria_no();
        raiz->chave[0] = chave;
        raiz->ndesc = 1;
        return raiz;
    } else {
        if (raiz->ndesc < MAX) {
            int i = raiz->ndesc - 1;
            while (i >= 0 && chave < raiz->chave[i]) {
                raiz->chave[i + 1] = raiz->chave[i];
                i--;
            }
            raiz->chave[i + 1] = chave;
            raiz->ndesc++;
        } else {
            int i = 0;
            while (i < raiz->ndesc && chave > raiz->chave[i]) {
                i++;
            }
            raiz->ramo[i] = insere_chave(raiz->ramo[i], chave);
        }

        if (raiz->ndesc > MAX) {
            int meio = raiz->ndesc / 2;
            t_no *novo = cria_no();
            for (int j = 0; j < MAX - meio; j++) {
                novo->chave[j] = raiz->chave[j + meio];
                novo->ramo[j] = raiz->ramo[j + meio];
            }
            novo->ramo[MAX - meio] = raiz->ramo[MAX];
            raiz->ndesc = meio;
            novo->ndesc = MAX - meio;
            return novo;
        }
    }
    return raiz;
}
```

4. `imprime_simetrica(t_no *raiz)`: Esta função imprime a árvore B em ordem simétrica

```c
void imprime_simetrica(t_no *raiz) {
    if (raiz) {
        for (int i = 0; i < raiz->ndesc; i++) {
            imprime_simetrica(raiz->ramo[i]);
            printf("%d ", raiz->chave[i]);
        }
        imprime_simetrica(raiz->ramo[raiz->ndesc]);
    }
}
```

Solução:

Passo a passo, o programa realiza as seguintes ações:

1. Cria um nó raiz da árvore B.

2. Insere 21 chaves na árvore B uma por uma.

3. Após cada inserção, imprime a árvore em ordem simétrica.

4. Em seguida, realiza a busca de chaves em intervalos específicos e imprime as chaves encontradas.

Aqui estão os resultados da impressão da árvore em ordem simétrica após cada inserção:

```
inserção de 010: 10
inserção de 040: 10 40
inserção de 050: 10 40 50
inserção de 060: 10 40 50 60
inserção de 070: 10 40 50 60 70
inserção de 075: 10 40 50 60 70 75
inserção de 080: 10 40 50 60 70 75 80
inserção de 090: 10 40 50 60 70 75 80 90
inserção de 100: 10 40 50 60 70 75 80 90 100
inserção de 110: 10 40 50 60 70 75 80 90 100 110
inserção de 115: 10 40 50 60 70 75 80 90 100 110 115
inserção de 120: 10 40 50 60 70 75 80 90 100 110 115 120
inserção de 130: 10 40 50 60 70 75 80 90 100 110 115 120 130
inserção de 135: 10 40 50 60 70 75 80 90 100 110 115 120 130 135
inserção de 140: 10 40 50 60 70 75 80 90 100 110 115 120 130 135 140
inserção de 170: 10 40 50 60 70 75 80 90 100 110 115 120 130 135 140 170
inserção de 200: 10 40 50 60 70 75 80 90 100 110 115 120 130 135 140 170 200
inserção de 220: 10 40 50 60 70 75 80 90 100 110 115 120 130 135 140 170 200 220
inserção de 230: 10 40 50 60 70 75 80 90 100 110 115 120 130 135 140 170 200 220 230
inserção de 240: 10 40 50 60 70 75 80 90 100 110 115 120 130 135 140 170 200 220 230 240
inserção de 250: 10 40 50 60 70 75 80 90 100 110 115 120 130 135 140 170 200 220 230 240 250
```

Aqui estão os resultados da busca e impressão de chaves em intervalos específicos:

```
--------------------------------------------------------------------------------------------------------
Chaves no intervalo (25, 75): 40 50 60 70
Chaves no intervalo (80, 110): 90 100
Chaves no intervalo (120, 140): 130 135
Chaves no intervalo (70, 60):
Chaves no intervalo (200, 230): 220
```

Observações e Conclusões:

O desenvolvimento deste programa permitiu a compreensão e implementação de uma árvore B de ordem 5, uma estrutura de dados eficiente para a busca e classificação de elementos. Durante a implementação, foram encontrados desafios na lógica de divisão de nós quando o número máximo de chaves era atingido. É importante garantir que a árvore mantenha a propriedade da árvore B após cada inserção.

No entanto, o programa funcionou conforme o esperado, permitindo a inserção, impressão e busca de chaves em intervalos específicos.