1.2/ArvoreB.h

```c
/*---------------- File: ArvoreB.h  --------------------+
|TAD: Arvore B                                          |
|                                                       |
| Baseado no material do Prof. Rafael Sachetto          |
| Implementado por Guilherme C. Pena em 02/12/2023      |
+-------------------------------------------------------+ */

#ifndef ARVOREB_H
#define ARVOREB_H

#include <stdio.h>
#include <stdlib.h>

#define M 2
#define MM (M*2) //2M
#define FALSE 0
#define TRUE 1

typedef struct Registro{
    int chave;
    /*outros componentes*/
}Registro;

typedef struct Pagina* Apontador;

typedef struct Pagina{
    int n; //Qtd de registros na pagina
    Registro r[MM];
    Apontador p[MM + 1];
}Pagina;

typedef struct Pagina* ArvoreB;

ArvoreB* criaArvoreB(){
    ArvoreB* raiz;
    raiz = (ArvoreB*) malloc (sizeof(ArvoreB));
    if(raiz != NULL)
        *raiz = NULL;
    return raiz;
}

Pagina* criapagina(){
    Pagina* pag;
    pag = (Pagina*) malloc (sizeof(Pagina));
    pag->n = 0;
    int i;
    for(i=0; i<(MM + 1); i++)
        pag->p[i] = NULL;
    return pag;
}

void destroiPagina(Pagina *pag){
    if(pag == NULL) return;
    int i;
    for(i=0; i<(MM + 1); i++)
        destroiPagina(pag->p[i]);
    free(pag);
```

```c
}

void destroiArvoreB(ArvoreB *raiz){
    if(raiz != NULL){
        destroiPagina(*raiz);
        free(raiz);
    }
}

int estaVazia(ArvoreB* raiz){
    if(raiz == NULL) return 0;
    return (*raiz == NULL);
}

void pre_ordem(Pagina* raiz, int nivel){
    if(raiz != NULL){
        int i;
        printf("Nivel %d: ", nivel);
        for(i=0; i<raiz->n; i++)
            printf("%d ", raiz->r[i].chave);
        printf("\n");

        //int contP = 0;
        //for(i=0; i<MM+1; i++) if(raiz->p[i] != NULL) contP++;
        //printf("%d ponteiros\n", contP);

        for(i=0; i<raiz->n+1; i++)
            pre_ordem(raiz->p[i], nivel+1);
    }
}

void imprimeArvoreB(ArvoreB* raiz){
    if(raiz == NULL) return;
    if(estaVazia(raiz)){
        printf("Arvore B Vazia!\n");
        return;
    }
    pre_ordem(*raiz, 0);
    printf("\n");
}

//Procedimento de PESQUISAR

void pesquisaRec(Registro *x, Apontador ap){
    if (ap == NULL) {
        printf("Registro nao esta presente na arvore\n");
        return;
    }
    int i = 1;
    while (i < ap->n && x->chave > ap->r[i - 1].chave) i++;
    if (x->chave == ap->r[i - 1].chave) {
        printf("Registro (chave %d) encontrado!\n", x->chave);
        *x = ap->r[i - 1]; // Atribui reg
        return;
    }
    if (x->chave < ap->r[i - 1].chave)
        pesquisaRec(x, ap->p[i - 1]);
    else
        pesquisaRec(x, ap->p[i]);
}
```

```
118
119 void pesquisaArvoreB(ArvoreB *raiz, Registro *reg){
120     if(raiz == NULL) return;
121     if(estaVazia(raiz)){
122         printf("Arvore B Vazia!\n");
123         return;
124     }
125     pesquisaRec(reg, *raiz);
126 }
127
128
129 //Procedimento de INSERIR
130
131 void insereNaPagina(Apontador ap, Registro reg, Apontador apDir) {
132     int k, NaoAchouPosicao;
133     k = ap->n;
134     NaoAchouPosicao = (k > 0);
135
136     while (NaoAchouPosicao) {
137         if (reg.chave >= ap->r[k - 1].chave) {
138             NaoAchouPosicao = FALSE;
139             break;
140         }
141
142         ap->r[k] = ap->r[k - 1]; // Atribui reg
143         ap->p[k + 1] = ap->p[k];
144         k--;
145
146         if (k < 1) NaoAchouPosicao = FALSE;
147     }
148
149     ap->r[k] = reg; // Atribui reg
150     ap->p[k + 1] = apDir;
151     ap->n++;
152 }
153
154
155
156 void insereRec(Registro reg, Apontador ap, int *Cresceu, Registro *regRetorno,
    Apontador *apRetorno) {
157     int i = 1, j;
158     Apontador apTemp;
159
160     if (ap == NULL) {
161         printf("Inserio %d..\n", reg.chave);
162         *Cresceu = TRUE;
163         *regRetorno = reg; // Atribui reg
164         *apRetorno = NULL;
165         return;
166     }
167
168     while (i < ap->n && reg.chave > ap->r[i - 1].chave) i++;
169
170     if (reg.chave == ap->r[i - 1].chave) {
171         printf("Erro: Registro ja esta presente\n");
172         *Cresceu = FALSE;
173         return;
174     }
175
176     if (reg.chave < ap->r[i - 1].chave) i--;
```

```
177
178     insereRec(reg, ap->p[i], Cresceu, regRetorno, apRetorno);
179
180     if (!*Cresceu) return;
181
182     if (ap->n < MM) { /* Página tem espaço */
183         insereNaPagina(ap, *regRetorno, *apRetorno);
184         *Cresceu = FALSE;
185         return;
186     }
187
188     /* Overflow: Página tem que ser dividida */
189     //Original Comentado
190     //apTemp = (Pagina*) malloc (sizeof(Pagina));
191     //apTemp->n = 0;
192     //apTemp->p[0] = NULL;
193
194     apTemp = criapagina();
195
196     if (i < (M + 1) ) {
197         insereNaPagina(apTemp, ap->r[MM - 1], ap->p[MM]);
198         ap->n--;
199         ap->p[MM] = NULL;
200         insereNaPagina(ap, *regRetorno, *apRetorno);
201     } else {
202         insereNaPagina(apTemp, *regRetorno, *apRetorno);
203     }
204
205     for (j = M + 2; j <= MM; j++) {
206         insereNaPagina(apTemp, ap->r[j - 1], ap->p[j]);
207         ap->p[j] = NULL;
208     }
209
210     ap->n = M;
211     apTemp->p[0] = ap->p[M + 1];
212     ap->p[M + 1] = NULL;
213     *regRetorno = ap->r[M]; // Atribui reg
214     *apRetorno = apTemp;
215 }
216
217 void insereArvoreB(ArvoreB *raiz, Registro reg) {
218     if(raiz == NULL) return;
219
220     int Cresceu;
221     Registro regRetorno;
222     Pagina *apRetorno, *apTemp;
223
224     insereRec(reg, *raiz, &Cresceu, &regRetorno, &apRetorno);
225
226     if (Cresceu) { /* Árvore cresce na altura pela raiz */
227         //apTemp = (Pagina *) malloc (sizeof(Pagina));
228         apTemp = criapagina();
229         apTemp->n = 1;
230         apTemp->r[0] = regRetorno;
231         apTemp->p[1] = apRetorno;
232         apTemp->p[0] = *raiz;
233         *raiz = apTemp;
234     }
235 }
236
```

```
237
238 //Procedimento de RETIRAR
239 void Reconstitui(Apontador apPag, Apontador apPai, int PosPai, int *Diminuiu) {
240     Pagina *Aux;
241     int DispAux, j;
242
243     if (PosPai < apPai->n) { /* Aux = Pagina a direita de apPag */
244         Aux = apPai->p[PosPai + 1];
245         DispAux = (Aux->n - M + 1) / 2;
246
247         apPag->r[apPag->n] = apPai->r[PosPai];
248         apPag->p[apPag->n + 1] = Aux->p[0];
249         apPag->n++;
250
251         if (DispAux > 0) { /* Existe folga: transfere de Aux para apPag */
252             for (j = 1; j < DispAux; j++)
253                 insereNaPagina(apPag, Aux->r[j - 1], Aux->p[j]);
254
255             apPai->r[PosPai] = Aux->r[DispAux - 1];
256             Aux->n -= DispAux;
257
258             for (j = 0; j < Aux->n; j++)
259                 Aux->r[j] = Aux->r[j + DispAux];
260
261             for (j = 0; j <= Aux->n; j++)
262                 Aux->p[j] = Aux->p[j + DispAux];
263
264             *Diminuiu = FALSE;
265         } else { /* Fusão: intercala Aux em apPag e libera Aux */
266             for (j = 1; j <= M; j++)
267                 insereNaPagina(apPag, Aux->r[j - 1], Aux->p[j]);
268
269             free(Aux);
270
271             for (j = PosPai + 1; j < apPai->n; j++) {
272                 apPai->r[j - 1] = apPai->r[j];
273                 apPai->p[j] = apPai->p[j + 1];
274             }
275
276             apPai->n--;
277
278             if (apPai->n >= M) *Diminuiu = FALSE;
279
280         }
281     } else { /* Aux = Pagina a esquerda de apPag */
282         Aux = apPai->p[PosPai - 1];
283         DispAux = (Aux->n - M + 1) / 2;
284
285         for (j = apPag->n; j >= 1; j--)
286             apPag->r[j] = apPag->r[j - 1];
287
288         apPag->r[0] = apPai->r[PosPai - 1];
289
290         for (j = apPag->n; j >= 0; j--)
291             apPag->p[j + 1] = apPag->p[j];
292
293         apPag->n++;
294
295         if (DispAux > 0) { /* Existe folga: transfere de Aux para apPag */
296             for (j = 1; j < DispAux; j++)
```

```
297                 insereNaPagina(apPag, Aux->r[Aux->n - j], Aux->p[Aux->n - j + 1]);
298
299             apPag->p[0] = Aux->p[Aux->n - DispAux + 1];
300             apPai->r[PosPai - 1] = Aux->r[Aux->n - DispAux];
301             Aux->n -= DispAux;
302             *Diminuiu = 0;
303         } else { /* Fusão: intercala apPag em Aux e libera apPag */
304             for (j = 1; j <= M; j++)
305                 insereNaPagina(Aux, apPag->r[j - 1], apPag->p[j]);
306
307             free(apPag);
308             apPai->n--;
309
310             if (apPai->n >= M) *Diminuiu = FALSE;
311         }
312     }
313 }
314
315 void Antecessor(Apontador ap, int i, Apontador apPai, int *Diminuiu) {
316     if (apPai->p[apPai->n] != NULL) {
317         Antecessor(ap, i, apPai->p[apPai->n], Diminuiu);
318         if (*Diminuiu)
319             Reconstitui(apPai->p[apPai->n], apPai, (int)apPai->n, Diminuiu);
320         return;
321     }
322
323     ap->r[i - 1] = apPai->r[apPai->n - 1];
324     apPai->n--;
325     *Diminuiu = (apPai->n < M);
326 }
327
328 void removeRec(int Ch, Apontador *ap, int *Diminuiu) {
329     int j, i = 1;
330     Apontador Pag;
331
332     if (*ap == NULL) {
333         printf("Erro: registro nao esta na arvore\n");
334         *Diminuiu = FALSE;
335         return;
336     }
337
338     Pag = *ap;
339
340     while (i < Pag->n && Ch > Pag->r[i - 1].chave) i++;
341
342     if (Ch == Pag->r[i - 1].chave) {
343         if (Pag->p[i - 1] == NULL) { /* Pagina folha */
344             Pag->n--;
345             *Diminuiu = (Pag->n < M);
346
347             for (j = i; j <= Pag->n; j++) {
348                 Pag->r[j - 1] = Pag->r[j];
349                 Pag->p[j] = Pag->p[j + 1];
350             }
351
352             return;
353         }
354
355         /* Pagina nao e folha: trocar com antecessor */
356         Antecessor(*ap, i, Pag->p[i - 1], Diminuiu);
```

```
357
358         if (*Diminuiu) Reconstitui(Pag->p[i - 1], *ap, i - 1, Diminuiu);
359
360         return;
361     }
362
363     if (Ch > Pag->r[i - 1].chave) i++;
364
365     removeRec(Ch, &Pag->p[i - 1], Diminuiu);
366
367     if (*Diminuiu) Reconstitui(Pag->p[i - 1], *ap, i - 1, Diminuiu);
368 }
369
370 void removeArvoreB(ArvoreB *raiz, Registro reg) {
371     int Diminuiu;
372     Apontador Aux;
373
374     printf("Removendo %d..\n", reg.chave);
375
376     removeRec(reg.chave, raiz, &Diminuiu);
377
378     if (Diminuiu && (*raiz)->n == 0) { /* Arvore diminui na altura */
379         Aux = *raiz;
380         *raiz = Aux->p[0];
381         free(Aux);
382     }
383 }
384
385
386
387
388 #endif
```