arvore1-1/ABP.h

```h
#ifndef ABP_H
#define ABP_H

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

typedef struct NO{
    int info;
    struct NO* esq;
    struct NO* dir;
}NO;

typedef struct NO* ABP;

NO* alocarNO(){
    return (NO*) malloc (sizeof(NO));
}

void liberarNO(NO* q){
    free(q);
}

ABP* criaABP(){
    ABP* raiz = (ABP*) malloc (sizeof(ABP));
    if(raiz != NULL)
        *raiz = NULL;
    return raiz;
}

void destroiRec(NO* no){
    if(no == NULL) return;
    destroiRec(no->esq);
    destroiRec(no->dir);
    liberarNO(no);
    no = NULL;
}

void destroiABP(ABP* raiz){
    if(raiz != NULL){
        destroiRec(*raiz);
        free(raiz);
    }
}

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


int insereRec(NO** raiz, int elem){
    if(*raiz == NULL){
        NO* novo = alocarNO();
        if(novo == NULL) return 0;
        novo->info = elem;
        novo->esq = NULL; novo->dir = NULL;
        *raiz = novo;
```

```c
    }else{
        if((*raiz)->info == elem){
            printf("Elemento Existente!\n");
            return 0;
        }
        if(elem < (*raiz)->info)
            return insereRec(&(*raiz)->esq, elem);
        else if(elem > (*raiz)->info)
            return insereRec(&(*raiz)->dir, elem);
    }
    return 1;
}

int insereIte(NO** raiz, int elem){
    NO *aux = *raiz, *ant = NULL;
    while (aux != NULL){
        ant = aux;
        if(aux->info == elem){
            printf("Elemento Existente!\n");
            return 0;
        }
        if(elem < aux->info) aux = aux->esq;
        else aux = aux->dir;
    }
    NO* novo = alocarNO();
    if(novo == NULL) return 0;
    novo->info = elem;
    novo->esq = NULL; novo->dir = NULL;
    if(ant == NULL){
        *raiz = novo;
    }else{
        if(elem < ant->info) ant->esq = novo;
        else ant->dir = novo;
    }
    return 1;
}

int insereElem(ABP* raiz, int elem){
    if(raiz == NULL) return 0;
    return insereRec(raiz, elem);
}

int pesquisaRec(NO** raiz, int elem){
    if(*raiz == NULL) return 0;
    if((*raiz)->info == elem) return 1;
    if(elem < (*raiz)->info)
        return pesquisaRec(&(*raiz)->esq, elem);
    else
        return pesquisaRec(&(*raiz)->dir, elem);
}

int pesquisaIte(NO** raiz, int elem){
    NO* aux = *raiz;
    while(aux != NULL){
        if(aux->info == elem) return 1;
        if(elem < aux->info)
            aux = aux->esq;
        else
            aux = aux->dir;
    }
```

```
118     return 0;
119 }
120 
121 int pesquisa(ABP* raiz, int elem){
122     if(raiz == NULL) return 0;
123     if(estaVazia(raiz)) return 0;
124     return pesquisaRec(raiz, elem);
125 }
126 
127 int removeRec(NO** raiz, int elem){
128     if(*raiz == NULL) return 0;
129     if((*raiz)->info == elem){
130         NO* aux;
131         if((*raiz)->esq == NULL && (*raiz)->dir == NULL){
132             //Caso 1 - NO sem filhos
133             printf("Caso 1: Liberando %d..\n", (*raiz)->info);
134             liberarNO(*raiz);
135             *raiz = NULL;
136         }else if((*raiz)->esq == NULL){
137             printf("Caso 2.1: Liberando %d..\n", (*raiz)->info);
138             aux = *raiz;
139             *raiz = (*raiz)->dir;
140             liberarNO(aux);
141         }else if((*raiz)->dir == NULL){
142             printf("Caso 2.2: Liberando %d..\n", (*raiz)->info);
143             aux = *raiz;
144             *raiz = (*raiz)->esq;
145             liberarNO(aux);
146         }else{
147             printf("Caso 3: Liberando %d..\n", (*raiz)->info);
148             //Estrategia 3.1:
149             NO* Filho = (*raiz)->esq;
150             while(Filho->dir != NULL)
151                 Filho = Filho->dir;
152             (*raiz)->info = Filho->info;
153             Filho->info = elem;
154             return removeRec(&(*raiz)->esq, elem);
155         }
156         return 1;
157     }else if(elem < (*raiz)->info)
158         return removeRec(&(*raiz)->esq, elem);
159     else
160         return removeRec(&(*raiz)->dir, elem);
161 }
162 
163 NO* removeAtual(NO* atual){
164     NO* no1, *no2;
165     if(atual->esq == NULL){
166         no2 = atual->dir;
167         liberarNO(atual);
168         return no2;
169     }
170     no1 = atual;
171     no2 = atual->esq;
172     while(no2->dir != NULL){
173         no1 = no2;
174         no2 = no2->dir;
175     }
176     if(no1 != atual){
177         no1->dir = no2->esq;
```

```
178         no2->esq = atual->esq;
179     }
180     no2->dir = atual->dir;
181     liberarNO(atual);
182     return no2;
183 }
184 
185 int removeIte(NO** raiz, int elem){
186     if(*raiz == NULL) return 0;
187     NO* atual = *raiz, *ant = NULL;
188     while(atual != NULL){
189         if(elem == atual->info){
190             if(atual == *raiz)
191                 *raiz = removeAtual(atual);
192             else{
193                 if(ant->dir == atual)
194                     ant->dir = removeAtual(atual);
195                 else
196                     ant->esq = removeAtual(atual);
197             }
198             return 1;
199         }
200         ant = atual;
201         if(elem < atual->info)
202             atual = atual->esq;
203         else
204             atual = atual->dir;
205     }
206     return 0;
207 }
208 
209 int removeElem(ABP* raiz, int elem){
210     if(pesquisa(raiz, elem) == 0){
211         printf("Elemento inexistente!\n");
212         return 0;
213     }
214     return removeIte(raiz, elem);
215 }
216 
217 void em_ordem(NO* raiz, int nivel){
218     if(raiz != NULL){
219         em_ordem(raiz->esq, nivel+1);
220         printf("[%d, %d] \n", raiz->info,nivel);
221         em_ordem(raiz->dir, nivel+1);
222     }
223 }
224 
225 void pre_ordem(NO* raiz, int nivel){
226     if(raiz != NULL){
227         printf("[%d, %d] \n", raiz->info,nivel);
228         pre_ordem(raiz->esq, nivel+1);
229         pre_ordem(raiz->dir, nivel+1);
230     }
231 }
232 
233 void pos_ordem(NO* raiz, int nivel){
234     if(raiz != NULL){
235         pos_ordem(raiz->esq, nivel+1);
236         pos_ordem(raiz->dir, nivel+1);
237         printf("[%d, %d] \n", raiz->info,nivel);
```

```
238     }
239 }
240 
241 void imprime(ABP* raiz){
242     if(raiz == NULL) return;
243     if(estaVazia(raiz)){
244         printf("Arvore Vazia!\n");
245         return;
246     }
247     printf("\nEm Ordem: "); em_ordem(*raiz, 0);
248     printf("\nPre Ordem: "); pre_ordem(*raiz, 0);
249     printf("\nPos Ordem: "); pos_ordem(*raiz, 0);
250     printf("\n");
251 }
252 
253 void numero_de_nos(NO** raiz, int* contador){
254     if(*raiz == NULL){
255         return;
256     }
257     numero_de_nos(&(*raiz)->esq,  contador);
258     numero_de_nos(&(*raiz)->dir, contador);
259     (*contador)++;
260 }
261 
262 
263 #endif
```