Realizei a impressão do código em um único arquivo para facilitar minha análise.

```c
#include <stdio.h>
#include <stdlib.h>
#define LADO_ESQ 0
#define LADO_DIR 1
#define false 0
#define true 1
int count = 1;


typedef struct no{
    char valor;
    struct no *filhoEsquerda;
    struct no *filhoDireita;
} No;


typedef struct {
    No *raiz;
} ArvoreBinaria;

ArvoreBinaria *criar() {
    ArvoreBinaria *arv = (ArvoreBinaria *)malloc(sizeof(ArvoreBinaria));
    if (arv != NULL) {
        arv->raiz = NULL;
    }
    return arv;
}

//criar raiz (root)
No *criarRaiz(ArvoreBinaria *arvore, char valor) {
    arvore->raiz = (No *) malloc(sizeof (No));
    if (arvore->raiz != NULL) {
        arvore->raiz->filhoEsquerda = NULL;
        arvore->raiz->filhoDireita = NULL;
        arvore->raiz->valor = valor;
    }
    return arvore->raiz;
}

No *add(int lado, No *v, char w){
    No *v_f = (No*) malloc(sizeof(No));

    if (v_f != NULL) {
        v_f->filhoDireita = NULL;
        v_f->filhoEsquerda = NULL;
        v_f->valor = w;

        if (lado == LADO_ESQ){
            v->filhoEsquerda = v_f;
        } else {
            v->filhoDireita = v_f;
        }
    }
    count++;
    return v_f;
}

int is_empty(ArvoreBinaria *arvore){
    if (arvore->raiz == NULL)
        return true;
    else
```

```
62          return false;
63  }
64
65  void pre_ordem_aux(No *raiz){
66      if ( raiz != NULL){
67          printf("%c - ", raiz->valor);
68          pre_ordem_aux(raiz->filhoEsquerda);
69          pre_ordem_aux(raiz->filhoDireita);
70      }
71  }
72
73  void pre_ordem (ArvoreBinaria *arvore){
74      pre_ordem_aux(arvore->raiz);
75  }
76
77  void em_ordem_aux(No *raiz){
78      if (raiz != NULL){
79          em_ordem_aux(raiz->filhoEsquerda);
80          printf("%c - ", raiz->valor);
81          em_ordem_aux(raiz->filhoDireita);
82      }
83  }
84
85  void em_ordem(ArvoreBinaria *arvore){
86      em_ordem_aux(arvore->raiz);
87  }
88
89  void pos_ordem_aux(No *raiz){
90      if(raiz != NULL){
91          pos_ordem_aux(raiz->filhoEsquerda);
92          pos_ordem_aux(raiz->filhoDireita);
93          printf("%c - ", raiz->valor);
94      }
95  }
96
97  void pos_ordem(ArvoreBinaria *arvore){
98      pos_ordem_aux(arvore->raiz);
99  }
100
101 int prof_d(No *raiz, int valor, int comparador){
102     int saida = 0;
103
104     do {
105
106         if (NULL == raiz)
107             break;
108
109         if (raiz->valor == valor) {
110             saida = comparador;
111             break;
112         }
113
114         saida = prof_d(raiz->filhoEsquerda, valor, comparador+1);
115
116         if (saida)
117             break;
118
119         saida = prof_d(raiz->filhoDireita, valor, comparador+1);
120
121         if(saida)
122             break;
```

```
123
124     }while(0);
125
126     return saida;
127 }
128
129 int find_tree(ArvoreBinaria *arvore, char valor){ - O(n)
130     int saida = 0;
131     if (arvore->raiz == NULL)  O(1)
132         return -1;
133     else                       O(n)
134         return (prof_d(arvore->raiz, valor, 0));
135
136 }
137
138 void clear(ArvoreBinaria *arvore){  O(1) - Constante
139     free(arvore);
140 }
141
142 int main (){
143     ArvoreBinaria *av = criar();
144
145     printf("%d", is_empty(av));
146
147     printf("\n");
148
149     criarRaiz(av, 'A');
150
151     add(LADO_ESQ, av->raiz, 'B');
152
153     add(LADO_ESQ, av->raiz->filhoEsquerda, 'D');
154
155     add(LADO_DIR, av->raiz->filhoEsquerda->filhoEsquerda, 'G');
156
157     add(LADO_DIR, av->raiz, 'C');
158
159     add(LADO_ESQ, av->raiz->filhoDireita, 'E');
160
161     add(LADO_DIR, av->raiz->filhoDireita, 'F');
162
163     add(LADO_ESQ, av->raiz->filhoDireita->filhoEsquerda, 'H');
164
165     add(LADO_DIR, av->raiz->filhoDireita->filhoEsquerda, 'I');
166
167     printf("\nBuscando F - Nivel %i\n", find_tree(av, 'F'));
168
169     pre_ordem(av);
170
171     printf("\n");
172
173     em_ordem(av);
174
175     printf("\n");
176
177     pos_ordem(av);
178
179     printf("\n");
180
181     printf("%i", count);
182
183     printf("\n");
```