Atividade 10 - Árvores Binárias e Busca Binária

13 de novembro de 2025

Problema A

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct no {
  int chave;
  struct no* esquerda;
  struct no* direita;
  struct no* pai;
};
typedef struct no No;

struct arvore_binaria {
  No *raiz;
};
typedef struct arvore_binaria ArvoreBinaria;

ArvoreBinaria* CriarArvore();
No* CriarNo(int chave);
void Inserir(ArvoreBinaria *arvore, No *novo_no);
No* BuscarNo(ArvoreBinaria *arvore, int chave);
void Remover(ArvoreBinaria *arvore, int chave);
No* EncontrarAntecessor(No *no);
No* EncontrarMaximo(No *no);
void PercorrerPreOrdem(No *no_atual, int *primeira_impressao);
void PercorrerEmOrdem(No *no_atual, int *primeira_impressao);
void PercorrerPosOrdem(No *no_atual, int *primeira_impressao);
void DestruirArvore(ArvoreBinaria *arvore);
void DestruirNos(No *raiz);

int main(){
    char a, b;

    ArvoreBinaria *arvore = CriarArvore();

    while(scanf("%c%c", &a, &b) != EOF){
        if(a == 'A') return 0;

        if(b == ' '){
            int chave;
            scanf("%d\n", &chave);
            if(a == 'I'){
```

```
43                  Inserir(arvore, CriarNo(chave));
44              } else if (a == 'R'){
45                  Remover(arvore, chave);
46              } else if (a == 'P'){
47                  No *no = BuscarNo(arvore, chave);
48
49                  if(no == NULL) printf("%d nao existe\n", chave);
50                  else printf("%d existe\n", chave);
51              }
52              continue;
53          }
54
55          char lixo[50];
56          scanf("%s\n", lixo);
57
58          int primeira_impressao = 1;
59          if(a == 'I'){
60              PercorrerEmOrdem(arvore->raiz, &primeira_impressao);
61          } else if (b == 'O'){
62              PercorrerPosOrdem(arvore->raiz, &primeira_impressao);
63          } else if (b == 'R'){
64              PercorrerPreOrdem(arvore->raiz, &primeira_impressao);
65          }
66          printf("\n");
67      }
68      return 0;
69  }
70
71  void Inserir(ArvoreBinaria *arvore, No *novo_no) {
72    No *pai_atual = NULL;
73    No *no_atual = arvore->raiz;
74
75    while (no_atual != NULL) {
76      pai_atual = no_atual;
77      if (novo_no->chave < no_atual->chave)
78        no_atual = no_atual->esquerda;
79      else
80        no_atual = no_atual->direita;
81    }
82
83    novo_no->pai = pai_atual;
84
85    if (pai_atual == NULL) {
86      arvore->raiz = novo_no;
87    } else {
88      if (novo_no->chave < pai_atual->chave)
89        pai_atual->esquerda = novo_no;
90      else
91        pai_atual->direita = novo_no;
92    }
93  }
94
95  No* BuscarNo(ArvoreBinaria *arvore, int chave) {
96    No *no_atual = arvore->raiz;
97
98    while (no_atual != NULL) {
99      if (chave == no_atual->chave)
100       return no_atual;
```

```
101     if (chave < no_atual->chave)
102       no_atual = no_atual->esquerda;
103     else
104       no_atual = no_atual->direita;
105   }
106   return NULL;
107 }
108 
109 No* EncontrarMaximo(No *no){
110     while(no->direita != NULL){
111         no = no->direita;
112     }
113     return no;
114 }
115 
116 No* EncontrarAntecessor(No *no){
117     if (no->esquerda != NULL){
118         return EncontrarMaximo(no->esquerda);
119     }
120 
121     return no->pai;
122 }
123 
124 void Remover(ArvoreBinaria *arvore, int chave) {
125     if(arvore == NULL) return;
126 
127     No *no = arvore->raiz;
128 
129     if(no == NULL) return;
130 
131     while(no != NULL && no->chave != chave){
132         if(chave < no->chave){
133             no = no->esquerda;
134         } else {
135             no = no->direita;
136         }
137     }
138 
139     if(no == NULL) return;
140 
141     No *pai = no->pai;
142 
143     // Caso o no seja um no folha
144     if(no->esquerda == NULL && no->direita == NULL){
145         if(pai == NULL){
146             free(no);
147             arvore->raiz = NULL;
148         } else {
149             if(pai->esquerda == no){
150                 free(no);
151                 pai->esquerda = NULL;
152             } else {
153                 free(no);
154                 pai->direita = NULL;
155             }
156         }
157         return;
158     }
```

```
159
160     // Caso o no tenha um filho a esquerda
161     if(no->direita == NULL){
162         if(pai == NULL){
163             arvore->raiz = no->esquerda;
164             no->esquerda->pai = NULL;
165             free(no);
166         } else {
167             if(pai->esquerda == no){
168                 pai->esquerda = no->esquerda;
169                 free(no);
170                 pai->esquerda->pai = pai;
171             } else {
172                 pai->direita = no->esquerda;
173                 free(no);
174                 pai->direita->pai = pai;
175             }
176         }
177         return;
178     }
179
180     // Caso o no tenha um filho a direita
181     if(no->esquerda == NULL){
182         if(pai == NULL){
183             arvore->raiz = no->direita;
184             no->direita->pai = NULL;
185             free(no);
186         } else {
187             if(pai->esquerda == no){
188                 pai->esquerda = no->direita;
189                 free(no);
190                 pai->esquerda->pai = pai;
191             } else {
192                 pai->direita = no->direita;
193                 free(no);
194                 pai->direita->pai = pai;
195             }
196         }
197         return;
198     }
199
200     // Caso o no tenha dois filhos
201     No* antecessor = EncontrarAntecessor(no);
202
203     no->chave = antecessor->chave;
204
205     // Caso em que o antecessor nao tem filhos
206     if(antecessor->esquerda == NULL){
207         if(antecessor->pai->esquerda == antecessor){
208             antecessor->pai->esquerda = NULL;
209         } else {
210             antecessor->pai->direita = NULL;
211         }
212         free(antecessor);
213     }
214     // Caso em que o antecessor tem filho a esquerda
215     else {
216         if(antecessor->pai->esquerda == antecessor){
```

```
217             antecessor->pai->esquerda = antecessor->esquerda;
218             antecessor->esquerda->pai = antecessor->pai;
219         } else {
220             antecessor->pai->direita = antecessor->esquerda;
221             antecessor->esquerda->pai = antecessor->pai;
222         }
223         free(antecessor);
224     }
225     return;
226 }
227
228 ArvoreBinaria* CriarArvore() {
229     ArvoreBinaria *arvore = (ArvoreBinaria*)malloc(sizeof(ArvoreBinaria)
        );
230     arvore->raiz = NULL;
231     return arvore;
232 }
233
234 No* CriarNo(int chave) {
235     No *novo_no = (No*)malloc(sizeof(No));
236     novo_no->chave = chave;
237     novo_no->pai = NULL;
238     novo_no->esquerda = NULL;
239     novo_no->direita = NULL;
240     return novo_no;
241 }
242
243 void PercorrerPreOrdem(No *no_atual, int *primeira_impressao) {
244     if (no_atual == NULL)
245         return;
246
247     if (*primeira_impressao == 1) {
248         printf("%d", no_atual->chave);
249         *primeira_impressao = 0;
250     } else {
251         printf(" %d", no_atual->chave);
252     }
253
254     if (no_atual->esquerda != NULL)
255         PercorrerPreOrdem(no_atual->esquerda, primeira_impressao);
256
257     if (no_atual->direita != NULL)
258         PercorrerPreOrdem(no_atual->direita, primeira_impressao);
259
260     return;
261 }
262
263 void PercorrerEmOrdem(No *no_atual, int *primeira_impressao) {
264     if (no_atual == NULL)
265         return;
266
267     if (no_atual->esquerda != NULL)
268         PercorrerEmOrdem(no_atual->esquerda, primeira_impressao);
269
270     if (*primeira_impressao == 1) {
271         printf("%d", no_atual->chave);
272         *primeira_impressao = 0;
273     } else {
```

```
274         printf(" %d", no_atual->chave);
275     }
276 
277     if (no_atual->direita != NULL)
278         PercorrerEmOrdem(no_atual->direita, primeira_impressao);
279 
280     return;
281 }
282 
283 void PercorrerPosOrdem(No *no_atual, int *primeira_impressao) {
284     if (no_atual == NULL)
285         return;
286 
287     if (no_atual->esquerda != NULL)
288         PercorrerPosOrdem(no_atual->esquerda, primeira_impressao);
289 
290     if (no_atual->direita != NULL)
291         PercorrerPosOrdem(no_atual->direita, primeira_impressao);
292 
293     if (*primeira_impressao == 1) {
294         printf("%d", no_atual->chave);
295         *primeira_impressao = 0;
296     } else {
297         printf(" %d", no_atual->chave);
298     }
299 
300     return;
301 }
302 
303 void DestruirNos(No *raiz) {
304     if (raiz == NULL)
305         return;
306 
307     if (raiz->esquerda != NULL)
308         DestruirNos(raiz->esquerda);
309 
310     if (raiz->direita != NULL)
311         DestruirNos(raiz->direita);
312 
313     free(raiz);
314     return;
315 }
316 
317 void DestruirArvore(ArvoreBinaria *arvore) {
318     DestruirNos(arvore->raiz);
319     free(arvore);
320 }
```

Problema B

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Estrutura de um no da arvore binaria
struct no {
  int chave;              // Valor armazenado no no
  struct no* esquerda;    // Ponteiro para o filho a esquerda
  struct no* direita;     // Ponteiro para o filho a direita
  struct no* pai;         // Ponteiro para o no pai
};
typedef struct no No;

// Estrutura da arvore binaria de busca
struct arvore_binaria {
  No *raiz;               // Ponteiro para o no raiz da arvore
};
typedef struct arvore_binaria ArvoreBinaria;

ArvoreBinaria* CriarArvore();
No* CriarNo(int chave);
void Inserir(ArvoreBinaria *arvore, No *novo_no);
No* BuscarNo(ArvoreBinaria *arvore, int chave);
void PercorrerPreOrdem(No *no_atual, int *primeira_impressao);
void PercorrerEmOrdem(No *no_atual, int *primeira_impressao);
void PercorrerPosOrdem(No *no_atual, int *primeira_impressao);
void DestruirNos(No *raiz);
void DestruirArvore(ArvoreBinaria *arvore);

void CopiaNosArvore(ArvoreBinaria *destino, No* no_atual);

int main(){
    int n;
    scanf("%d", &n);

    ArvoreBinaria *arvore1 = CriarArvore();

    for(int i = 0; i < n; ++i){
        int chave;
        scanf("%d", &chave);

        Inserir(arvore1, CriarNo(chave));
    }

    int m;
    scanf("%d", &m);

    ArvoreBinaria *arvore2 = CriarArvore();

    for(int j = 0; j < m; ++j){
        int chave;
        scanf("%d", &chave);

        Inserir(arvore2, CriarNo(chave));
    }

```

```c
    ArvoreBinaria *fusao = CriarArvore();

    CopiaNosArvore(fusao, arvore1->raiz);
    CopiaNosArvore(fusao, arvore2->raiz);

    int primeira_impressao = 1;
    PercorrerEmOrdem(fusao->raiz, &primeira_impressao);
    printf("\n");

    DestruirArvore(arvore1);
    DestruirArvore(arvore2);
    DestruirArvore(fusao);

    return 0;
}

void Inserir(ArvoreBinaria *arvore, No *novo_no) {
    if(BuscarNo(arvore, novo_no->chave) != NULL){
        return;
    }

    No *pai_atual = NULL;
    No *no_atual = arvore->raiz;

    // Percorre a arvore ate encontrar a posicao correta
    while (no_atual != NULL) {
        pai_atual = no_atual;
        if (novo_no->chave < no_atual->chave)
            no_atual = no_atual->esquerda;
        else
            no_atual = no_atual->direita;
    }

    // Define o pai do novo no
    novo_no->pai = pai_atual;

    // Se a arvore esta vazia, o novo no se torna a raiz
    if (pai_atual == NULL) {
        arvore->raiz = novo_no;
    } else {
        // Insere o novo no como filho esquerdo ou direito
        if (novo_no->chave < pai_atual->chave)
            pai_atual->esquerda = novo_no;
        else
            pai_atual->direita = novo_no;
    }
}

No* BuscarNo(ArvoreBinaria *arvore, int chave) {
    No *no_atual = arvore->raiz;

    // Percorre a arvore procurando pela chave
    while (no_atual != NULL) {
        if (chave == no_atual->chave)
            return no_atual;
        if (chave < no_atual->chave)
            no_atual = no_atual->esquerda;
        else
```

```
115             no_atual = no_atual->direita;
116     }
117     return NULL;  // Retorna NULL se nao encontrar
118 }
119
120 ArvoreBinaria* CriarArvore() {
121     ArvoreBinaria *arvore = (ArvoreBinaria*)malloc(sizeof(ArvoreBinaria)
        );
122     arvore->raiz = NULL;
123     return arvore;
124 }
125
126 No* CriarNo(int chave) {
127     No *novo_no = (No*)malloc(sizeof(No));
128     novo_no->chave = chave;
129     novo_no->pai = NULL;
130     novo_no->esquerda = NULL;
131     novo_no->direita = NULL;
132     return novo_no;
133 }
134
135 void CopiaNosArvore(ArvoreBinaria *destino, No* no_atual) {
136     if (no_atual == NULL)
137         return;
138
139     // Primeiro percorre a subarvore esquerda
140     if (no_atual->esquerda != NULL)
141         CopiaNosArvore(destino, no_atual->esquerda);
142
143     Inserir(destino, CriarNo(no_atual->chave));
144
145     // Por fim percorre a subarvore direita
146     if (no_atual->direita != NULL)
147         CopiaNosArvore(destino, no_atual->direita);
148
149     return;
150 }
151
152 void PercorrerPreOrdem(No *no_atual, int *primeira_impressao) {
153     if (no_atual == NULL)
154         return;
155
156     // Imprime a raiz primeiro
157     if (*primeira_impressao == 1) {
158         printf("%d", no_atual->chave);
159         *primeira_impressao = 0;
160     } else {
161         printf(" %d", no_atual->chave);
162     }
163
164     // Depois percorre a subarvore esquerda
165     if (no_atual->esquerda != NULL)
166         PercorrerPreOrdem(no_atual->esquerda, primeira_impressao);
167
168     // Por fim percorre a subarvore direita
169     if (no_atual->direita != NULL)
170         PercorrerPreOrdem(no_atual->direita, primeira_impressao);
171
```

```
172     return;
173 }
174 
175 void PercorrerEmOrdem(No *no_atual, int *primeira_impressao) {
176     if (no_atual == NULL)
177         return;
178 
179     // Primeiro percorre a subarvore esquerda
180     if (no_atual->esquerda != NULL)
181         PercorrerEmOrdem(no_atual->esquerda, primeira_impressao);
182 
183     // Depois imprime a raiz
184     if (*primeira_impressao == 1) {
185         printf("%d", no_atual->chave);
186         *primeira_impressao = 0;
187     } else {
188         printf(" %d", no_atual->chave);
189     }
190 
191     // Por fim percorre a subarvore direita
192     if (no_atual->direita != NULL)
193         PercorrerEmOrdem(no_atual->direita, primeira_impressao);
194 
195     return;
196 }
197 
198 void PercorrerPosOrdem(No *no_atual, int *primeira_impressao) {
199     if (no_atual == NULL)
200         return;
201 
202     // Primeiro percorre a subarvore esquerda
203     if (no_atual->esquerda != NULL)
204         PercorrerPosOrdem(no_atual->esquerda, primeira_impressao);
205 
206     // Depois percorre a subarvore direita
207     if (no_atual->direita != NULL)
208         PercorrerPosOrdem(no_atual->direita, primeira_impressao);
209 
210     // Por fim imprime a raiz
211     if (*primeira_impressao == 1) {
212         printf("%d", no_atual->chave);
213         *primeira_impressao = 0;
214     } else {
215         printf(" %d", no_atual->chave);
216     }
217 
218     return;
219 }
220 
221 void DestruirNos(No *raiz) {
222     if (raiz == NULL)
223         return;
224 
225     if (raiz->esquerda != NULL)
226         DestruirNos(raiz->esquerda);
227 
228     if (raiz->direita != NULL)
229         DestruirNos(raiz->direita);
```

```
230
231     free(raiz);
232     return;
233 }
234
235 void DestruirArvore(ArvoreBinaria *arvore) {
236     DestruirNos(arvore->raiz);
237     free(arvore);
238 }
```

Problema C

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

int buscaBinaria(int vetor[], int tamanho, int valor) {
    int esquerda = 0;
    int direita = tamanho;

    while (esquerda <= direita) {
        int meio = (esquerda + direita) / 2;

        if (vetor[meio] == valor) {
            return meio;
        }

        if (valor < vetor[meio]) {
            direita = meio - 1;
        }
        else {
            esquerda = meio + 1;
        }
    }

    return -1;
}

int main() {
    int n, q;

    scanf("%d", &n);

    int *vetor = (int*)malloc(n * sizeof(int));
    for (int i = 0; i < n; i++) {
        scanf("%d", &vetor[i]);
    }

    scanf("%d", &q);

    for (int i = 0; i < q; i++) {
        int valor;
        scanf("%d", &valor);

        int resultado = buscaBinaria(vetor, n, valor);

        if (resultado != -1) {
            printf("%d encontrado na posicao %d\n", valor, resultado);
        } else {
            printf("%d nao encontrado\n", valor);
        }
    }

    free(vetor);
    return 0;
}
```