

UFSJ - Ciências da Computação

Laboratório de Programação 2

Roteiro 8

Nome: Geraldo Arthur Detomi

1.1) TAD: Arvore Binária de Pesquisa (ABP)

roteiro_8/ABP.h

```
1  #ifndef ABP_H
2  #define ABP_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  typedef struct NO {
8      int info;
9      struct NO *esq;
10     struct NO *dir;
11 } NO;
12
13 typedef struct NO *ABP;
14
15 NO *alocarNO();
16
17 void liberarNO(NO *q);
18
19 ABP *criaABP();
20
21 void destroiRec(NO *no);
22
23 void destroiABP(ABP *raiz);
24
25 int estaVazia(ABP *raiz);
26
27 int insereRec(NO **raiz, int elem);
28
29 int insereIte(NO **raiz, int elem);
30
31 int insereElem(ABP *raiz, int elem);
32
33 int pesquisaRec(NO **raiz, int elem);
34
35 int pesquisaIte(NO **raiz, int elem);
36
37 int pesquisa(ABP *raiz, int elem);
38
39 int removeRec(NO **raiz, int elem);
40
41 NO *removeAtual(NO *atual);
42
43 int removeIte(NO **raiz, int elem);
44
45 int removeElem(ABP *raiz, int elem);
46 void em_ordem(NO *raiz, int nivel);
47
48 void pre_ordem(NO *raiz, int nivel);
49
50 void pos_ordem(NO *raiz, int nivel);
51
52 void imprime(ABP *raiz);
53
54 int get_quantidade_nos(ABP *raiz);
55
56 #endif
```

roteiro_8/ABP.c

```
1
2  #include "ABP.h"
3
4  NO *alocarNO() { return (NO *)malloc(sizeof(NO)); }
5
6  void liberarNO(NO *q) { free(q); }
7
```

```
8 | ABP *criaABP() {
9 |     ABP *raiz = (ABP *)malloc(sizeof(ABP));
10 |     if (raiz != NULL)
11 |         *raiz = NULL;
12 |     return raiz;
13 | }
14 |
15 | void destroiRec(NO *no) {
16 |     if (no == NULL)
17 |         return;
18 |     destroiRec(no->esq);
19 |     destroiRec(no->dir);
20 |     liberarNO(no);
21 |     no = NULL;
22 | }
23 |
24 | void destroiABP(ABP *raiz) {
25 |     if (raiz != NULL) {
26 |         destroiRec(*raiz);
27 |         free(raiz);
28 |     }
29 | }
30 |
31 | int estaVazia(ABP *raiz) {
32 |     if (raiz == NULL)
33 |         return 0;
34 |     return (*raiz == NULL);
35 | }
36 |
37 | int insereRec(NO **raiz, int elem) {
38 |     if (*raiz == NULL) {
39 |         NO *novo = alocarNO();
40 |         if (novo == NULL)
41 |             return 0;
42 |         novo->info = elem;
43 |         novo->esq = NULL;
44 |         novo->dir = NULL;
45 |         *raiz = novo;
46 |     } else {
47 |         if ((*raiz)->info == elem) {
48 |             printf("Elemento Existente!\n");
49 |             return 0;
50 |         }
51 |         if (elem < (*raiz)->info)
52 |             return insereRec(&(*raiz)->esq, elem);
53 |         else if (elem > (*raiz)->info)
54 |             return insereRec(&(*raiz)->dir, elem);
55 |     }
56 |     return 1;
57 | }
58 |
59 | int insereIte(NO **raiz, int elem) {
60 |     NO *aux = *raiz, *ant = NULL;
61 |     while (aux != NULL) {
62 |         ant = aux;
63 |         if (aux->info == elem) {
64 |             printf("Elemento Existente!\n");
65 |             return 0;
66 |         }
67 |         if (elem < aux->info)
68 |             aux = aux->esq;
69 |         else
70 |             aux = aux->dir;
71 |     }
72 |     NO *novo = alocarNO();
73 |     if (novo == NULL)
74 |         return 0;
75 |     novo->info = elem;
76 |     novo->esq = NULL;
77 |     novo->dir = NULL;
78 |     if (ant == NULL) {
79 |         *raiz = novo;
80 |     } else {
81 |         if (elem < ant->info)
82 |             ant->esq = novo;
83 |         else
84 |             ant->dir = novo;
85 |     }
86 |     return 1;
87 | }
88 |
```

```
89 int insereElem(ABP *raiz, int elem) {
90     if (raiz == NULL)
91         return 0;
92     return insereRec(raiz, elem);
93     // return insereIte(raiz, elem);
94 }
95
96 int pesquisaRec(NO **raiz, int elem) {
97     if (*raiz == NULL)
98         return 0;
99     if ((*raiz)->info == elem)
100         return 1;
101     if (elem < (*raiz)->info)
102         return pesquisaRec(&(*raiz)->esq, elem);
103     else
104         return pesquisaRec(&(*raiz)->dir, elem);
105 }
106
107 int pesquisaIte(NO **raiz, int elem) {
108     NO *aux = *raiz;
109     while (aux != NULL) {
110         if (aux->info == elem)
111             return 1;
112         if (elem < aux->info)
113             aux = aux->esq;
114         else
115             aux = aux->dir;
116     }
117     return 0;
118 }
119
120 int pesquisa(ABP *raiz, int elem) {
121     if (raiz == NULL)
122         return 0;
123     if (estaVazia(raiz))
124         return 0;
125     return pesquisaRec(raiz, elem);
126     // return pesquisaIte(raiz, elem);
127 }
128
129 int removeRec(NO **raiz, int elem) {
130     if (*raiz == NULL)
131         return 0;
132     if ((*raiz)->info == elem) {
133         NO *aux;
134         if ((*raiz)->esq == NULL && (*raiz)->dir == NULL) {
135             // Caso 1 - NO sem filhos
136             printf("Caso 1: Liberando %d..\n", (*raiz)->info);
137             liberarNO(*raiz);
138             *raiz = NULL;
139         } else if ((*raiz)->esq == NULL) {
140             // Caso 2.1 - Possui apenas uma subarvore direita
141             printf("Caso 2.1: Liberando %d..\n", (*raiz)->info);
142             aux = *raiz;
143             *raiz = (*raiz)->dir;
144             liberarNO(aux);
145         } else if ((*raiz)->dir == NULL) {
146             // Caso 2.2 - Possui apenas uma subarvore esquerda
147             printf("Caso 2.2: Liberando %d..\n", (*raiz)->info);
148             aux = *raiz;
149             *raiz = (*raiz)->esq;
150             liberarNO(aux);
151         } else {
152             // Caso 3 - Possui as duas subarvoretas (esq e dir)
153             // Duas estrategias:
154             // 3.1 - Substituir pelo NO com o MAIOR valor da subarvore esquerda
155             // 3.2 - Substituir pelo NO com o MENOR valor da subarvore direita
156             printf("Caso 3: Liberando %d..\n", (*raiz)->info);
157             // Estrategia 3.1:
158             NO *Filho = (*raiz)->esq;
159             while (Filho->dir != NULL) // Localiza o MAIOR valor da subarvore esquerda
160                 Filho = Filho->dir;
161             (*raiz)->info = Filho->info;
162             Filho->info = elem;
163             return removeRec(&(*raiz)->esq, elem);
164         }
165         return 1;
166     } else if (elem < (*raiz)->info)
167         return removeRec(&(*raiz)->esq, elem);
168     else
169         return removeRec(&(*raiz)->dir, elem);
170 }
```

```
170 }
171
172 NO *removeAtual(NO *atual) {
173     NO *no1, *no2;
174     // Ambos casos no if(atual->esq == NULL)
175     // Caso 1 - NO sem filhos
176     // Caso 2.1 - Possui apenas uma subarvore direita
177     if (atual->esq == NULL) {
178         no2 = atual->dir;
179         liberarNO(atual);
180         return no2;
181     }
182     // Caso 3 - Possui as duas subarvoretas (esq e dir)
183     // Estrategia:
184
185     no1 = atual;
186     no2 = atual->esq;
187     while (no2->dir != NULL) {
188         no1 = no2;
189         no2 = no2->dir;
190     }
191     if (no1 != atual) {
192         no1->dir = no2->esq;
193         no2->esq = atual->esq;
194     }
195     no2->dir = atual->dir;
196     liberarNO(atual);
197     return no2;
198 }
199
200 int removeIte(NO **raiz, int elem) {
201     if (*raiz == NULL)
202         return 0;
203     NO *atual = *raiz, *ant = NULL;
204     while (atual != NULL) {
205         if (elem == atual->info) {
206             if (atual == *raiz)
207                 *raiz = removeAtual(atual);
208             else {
209                 if (ant->dir == atual)
210                     ant->dir = removeAtual(atual);
211                 else
212                     ant->esq = removeAtual(atual);
213             }
214             return 1;
215         }
216         ant = atual;
217         if (elem < atual->info)
218             atual = atual->esq;
219         else
220             atual = atual->dir;
221     }
222     return 0;
223 }
224
225 int removeElem(ABP *raiz, int elem) {
226     if (pesquisa(raiz, elem) == 0) {
227         printf("Elemento inexistente!\n");
228         return 0;
229     }
230     // return removeRec(raiz, elem);
231     return removeIte(raiz, elem);
232 }
233
234 void em_ordem(NO *raiz, int nivel) {
235     if (raiz != NULL) {
236         em_ordem(raiz->esq, nivel + 1);
237         printf("[%d, %d] ", raiz->info, nivel);
238         em_ordem(raiz->dir, nivel + 1);
239     }
240 }
241
242 void pre_ordem(NO *raiz, int nivel) {
243     if (raiz != NULL) {
244         printf("[%d, %d] ", raiz->info, nivel);
245         pre_ordem(raiz->esq, nivel + 1);
246         pre_ordem(raiz->dir, nivel + 1);
247     }
248 }
249
250 void pos_ordem(NO *raiz, int nivel) {
```

```

251     if (raiz != NULL) {
252         pos_ordem(raiz->esq, nivel + 1);
253         pos_ordem(raiz->dir, nivel + 1);
254         printf("[%d, %d] ", raiz->info, nivel);
255     }
256 }
257
258 void imprime(ABP *raiz) {
259     if (raiz == NULL)
260         return;
261     if (estaVazia(raiz)) {
262         printf("Arvore Vazia!\n");
263         return;
264     }
265     printf("\nEm Ordem: ");
266     em_ordem(*raiz, 0);
267     printf("\nPre Ordem: ");
268     pre_ordem(*raiz, 0);
269     printf("\nPos Ordem: ");
270     pos_ordem(*raiz, 0);
271     printf("\n");
272 }
273
274 int get_quantidade_nos(ABP *raiz) {
275     if (*raiz == NULL) {
276         return 0;
277     }
278
279     int qtd = 1;
280
281     qtd += get_quantidade_nos(&(*raiz)->esq);
282
283     qtd += get_quantidade_nos(&(*raiz)->dir);
284
285     return qtd;
286 }
287

```

roteiro_8/1-1.c

```

1  #include "ABP.h"
2
3  #define MAX_OPTIONS 10
4
5  enum options {
6      CRIAR = 0,
7      INSERIR,
8      BUSCAR,
9      REMOVER,
10     IMPRIMIR_ORDEM,
11     IMPRIMIR_PRE_ORDEM,
12     IMPRIMIR_POS_ORDEM,
13     QUANTIDADE_NOS,
14     DESTRUIR,
15     SAIR,
16 };
17
18 int get_option() {
19     int opt = -1;
20     do {
21         printf("\tOperacoes\n");
22         printf("[%d] Criar árvore ABP, ", CRIAR);
23         printf("[%d] Inserir um elemento, ", INSERIR);
24         printf("[%d] Buscar um elemento, ", BUSCAR);
25         printf("[%d] Remover um elemento, ", REMOVER);
26         printf("[%d] Imprimir a ABP em ordem, ", IMPRIMIR_ORDEM);
27         printf("[%d] Imprimir a ABP em pré-ordem, ", IMPRIMIR_PRE_ORDEM);
28         printf("[%d] Imprimir a ABP em pós-ordem, ", IMPRIMIR_POS_ORDEM);
29         printf("[%d] Mostrar a quantidade de nós na ABP, ", QUANTIDADE_NOS);
30         printf("[%d] Destruir a ABP, ", DESTRUIR);
31         printf("[%d] Sair do programa \n", SAIR);
32
33         printf("\nInsira a opção desejada: ");
34         scanf("%d", &opt);
35         printf("\n");
36
37         if (opt < 0 || opt >= MAX_OPTIONS) {
38             printf("Opção escolhida inválida!\n");
39         }
40
41     } while (opt < 0 || opt >= MAX_OPTIONS);
42

```

```
43     return opt;
44 }
45
46 int get_valor(char *msg) {
47     int value;
48
49     printf("%s", msg);
50     scanf("%d", &value);
51
52     return value;
53 }
54
55 int main() {
56     int opt, valor;
57
58     ABP *arvore = NULL;
59
60     do {
61         opt = get_option();
62
63         switch (opt) {
64             case CRIAR:
65                 printf("Executando comando...\n");
66
67                 if (arvore == NULL) {
68                     arvore = criaABP();
69                 } else {
70                     destroiABP(arvore);
71                     arvore = NULL;
72                     arvore = criaABP();
73                 }
74
75                 printf("Árvore ABP criada com sucesso!\n");
76                 break;
77             case INSERIR:
78                 printf("Executando comando...\n");
79
80                 if (arvore == NULL) {
81                     printf("Árvore ABP não inicializada impossível realizar operação..\n");
82                     break;
83                 }
84
85                 valor = get_valor("Insira um valor: ");
86
87                 if (insereElem(arvore, valor)) {
88                     printf("Elemento inserido na árvore com sucesso!\n");
89                 } else {
90                     printf("Falha ao inserir elemento!\n");
91                 }
92
93                 break;
94             case BUSCAR:
95                 printf("Executando comando...\n");
96
97                 if (arvore == NULL) {
98                     printf("Árvore ABP não inicializada impossível realizar operação..\n");
99                     break;
100                 }
101
102                 valor = get_valor("Insira o valor para buscar: ");
103
104                 if (pesquisa(arvore, valor)) {
105                     printf("Elemento está presente na árvore ABP!\n");
106                 } else {
107                     printf("Elemento não está presente na árvore ABP!\n");
108                 }
109
110                 break;
111             case REMOVER:
112                 printf("Executando comando...\n");
113
114                 if (arvore == NULL) {
115                     printf("Árvore ABP não inicializada impossível realizar operação..\n");
116                     break;
117                 }
118
119                 valor = get_valor("Elemento a se remover: ");
120
121                 if (removeElem(arvore, valor)) {
122                     printf("Elemento removido com sucesso!\n");
123                 }
```

```
124
125     break;
126 case IMPRIMIR_ORDEM:
127     printf("Executando comando...\n");
128
129     if (arvore == NULL) {
130         printf("Árvore ABP não inicializada impossível realizar operação..\n");
131         break;
132     }
133
134     em_ordem(*arvore, 0);
135
136     printf("Árvore imprimida com sucesso!\n");
137
138     break;
139 case IMPRIMIR_PRE_ORDEM:
140     printf("Executando comando...\n");
141
142     if (arvore == NULL) {
143         printf("Árvore ABP não inicializada impossível realizar operação..\n");
144         break;
145     }
146
147     pre_ordem(*arvore, 0);
148
149     printf("Árvore imprimida com sucesso!\n");
150
151     break;
152 case IMPRIMIR_POS_ORDEM:
153     printf("Executando comando...\n");
154
155     if (arvore == NULL) {
156         printf("Árvore ABP não inicializada impossível realizar operação..\n");
157         break;
158     }
159
160     pos_ordem(*arvore, 0);
161
162     printf("Árvore imprimida com sucesso!\n");
163
164     break;
165 case QUANTIDADE_NOS:
166     printf("Executando comando...\n");
167
168     if (arvore == NULL) {
169         printf("Árvore ABP não inicializada impossível realizar operação..\n");
170         break;
171     }
172
173     int qtd_nos = get_quantidade_nos(arvore);
174
175     printf("A quantidade de nós é %d\n", qtd_nos);
176
177     break;
178 case DESTRUIR:
179     printf("Executando comando...\n");
180
181     if (arvore == NULL) {
182         printf("Árvore ABP não inicializada impossível realizar operação..\n");
183         break;
184     }
185
186     destroiABP(arvore);
187     arvore = NULL;
188
189     printf("Árvore ABP destruída com sucesso!");
190
191     break;
192 case SAIR:
193     if (arvore != NULL) {
194         destroiABP(arvore);
195         arvore = NULL;
196     }
197
198     printf("Finalizando programa! Até mais!\n");
199     break;
200 }
201 } while (opt != SAIR);
202
203 return 0;
204 }
```

Saída do terminal:

```

[arthurdetomi] at arthurdetomi-System-Product-Name in ~/Documents/UFSJ-Graduacao/UFSJ-2025_1/Lab_Prog_2/roteiro_8 on mainxxx 25-05-25 - 16:58:33
./1-1.out
Operacoes
[0] Criar árvore ABP, [1] Inserir um elemento, [2] Buscar um elemento, [3] Remover um elemento, [4] Imprimir a ABP em ordem, [5] Imprimir a ABP em pré-ordem
Insira a opção desejada: 0
Executando comando...
Árvore ABP criada com sucesso!
Operacoes
[0] Criar árvore ABP, [1] Inserir um elemento, [2] Buscar um elemento, [3] Remover um elemento, [4] Imprimir a ABP em ordem, [5] Imprimir a ABP em pré-ordem
Insira a opção desejada: 1
Executando comando...
Insira um valor: 1
Elemento inserido na árvore com sucesso!
Operacoes
[0] Criar árvore ABP, [1] Inserir um elemento, [2] Buscar um elemento, [3] Remover um elemento, [4] Imprimir a ABP em ordem, [5] Imprimir a ABP em pré-ordem
Insira a opção desejada: 1
Executando comando...
Insira um valor: 3
Elemento inserido na árvore com sucesso!
Operacoes
[0] Criar árvore ABP, [1] Inserir um elemento, [2] Buscar um elemento, [3] Remover um elemento, [4] Imprimir a ABP em ordem, [5] Imprimir a ABP em pré-ordem
Insira a opção desejada: 1
Executando comando...
Insira um valor: 2
Elemento inserido na árvore com sucesso!
Operacoes
[0] Criar árvore ABP, [1] Inserir um elemento, [2] Buscar um elemento, [3] Remover um elemento, [4] Imprimir a ABP em ordem, [5] Imprimir a ABP em pré-ordem
Insira a opção desejada: 4
Executando comando...
[1, 0] [2, 2] [3, 1] Árvore imprimida com sucesso!
Operacoes
[0] Criar árvore ABP, [1] Inserir um elemento, [2] Buscar um elemento, [3] Remover um elemento, [4] Imprimir a ABP em ordem, [5] Imprimir a ABP em pré-ordem
Insira a opção desejada: 7
Executando comando...
A quantidade de nós é 3
Operacoes
[0] Criar árvore ABP, [1] Inserir um elemento, [2] Buscar um elemento, [3] Remover um elemento, [4] Imprimir a ABP em ordem, [5] Imprimir a ABP em pré-ordem
Insira a opção desejada: 8
Executando comando...
Árvore ABP destruída com sucesso!
Operacoes
[0] Criar árvore ABP, [1] Inserir um elemento, [2] Buscar um elemento, [3] Remover um elemento, [4] Imprimir a ABP em ordem, [5] Imprimir a ABP em pré-ordem
Insira a opção desejada: 9
Finalizando programa! Até mais!

```

1.2) TAD: Árvore Binária de Pesquisa (ABP) com info igual a Aluno e chave da árvore como nome do aluno

roteiro_8/ABP_aluno.h

```

1  #ifndef ABP_ALUNO_H
2  #define ABP_ALUNO_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  typedef struct Aluno {
8      char nome[50];
9      int matricula;
10     double nota;
11 } Aluno;
12
13 typedef struct NO {
14     Aluno info;
15     struct NO *esq;
16     struct NO *dir;
17 } NO;
18
19 typedef struct NO *ABP;
20
21 NO *alocarNO();
22
23 void liberarNO(NO *q);
24
25 ABP *criaABP();
26
27 void destroiRec(NO *no);
28
29 void destroiABP(ABP *raiz);
30
31 int estaVazia(ABP *raiz);

```



```

32
33 int insereRec(NO **raiz, Aluno aluno);
34
35 int insereElem(ABP *raiz, Aluno aluno);
36
37 int pesquisaRec(NO **raiz, Aluno aluno);
38
39 int pesquisa(ABP *raiz, Aluno aluno);
40
41 int removeRec(NO **raiz, Aluno aluno);
42
43 int removeElem(ABP *raiz, Aluno aluno);
44
45 void em_ordem(NO *raiz, int nivel);
46
47 void pre_ordem(NO *raiz, int nivel);
48
49 void pos_ordem(NO *raiz, int nivel);
50
51 void imprime(ABP *raiz);
52
53 int get_quantidade_nos(ABP *raiz);
54
55 Aluno *get_aluno_maior_nota(ABP *raiz);
56
57 Aluno *get_aluno_pior_nota(ABP *raiz);
58
59 #endif

```

roteiro_8/ABP_aluno.c

```

1
2 #include "ABP_aluno.h"
3
4 #include <string.h>
5
6 NO *alocarNO() { return (NO *)malloc(sizeof(NO)); }
7
8 void liberarNO(NO *q) { free(q); }
9
10 ABP *criaABP() {
11     ABP *raiz = (ABP *)malloc(sizeof(ABP));
12     if (raiz != NULL)
13         *raiz = NULL;
14     return raiz;
15 }
16
17 void destroiRec(NO *no) {
18     if (no == NULL)
19         return;
20     destroiRec(no->esq);
21     destroiRec(no->dir);
22     liberarNO(no);
23     no = NULL;
24 }
25
26 void destroiABP(ABP *raiz) {
27     if (raiz != NULL) {
28         destroiRec(*raiz);
29         free(raiz);
30     }
31 }
32
33 int estaVazia(ABP *raiz) {
34     if (raiz == NULL)
35         return 0;
36     return (*raiz == NULL);
37 }
38
39 int insereRec(NO **raiz, Aluno aluno) {
40     if (*raiz == NULL) {
41         NO *novo = alocarNO();
42         if (novo == NULL)
43             return 0;
44         novo->info = aluno;
45         novo->esq = NULL;
46         novo->dir = NULL;
47         *raiz = novo;
48     } else {
49
50         int result_comp = strcmp(aluno.nome, (*raiz)->info.nome);
51

```

```
52     if (result_comp == 0) {
53         printf("Aluno Existente!\n");
54         return 0;
55     }
56     if (result_comp < 0)
57         return insereRec(&(*raiz)->esq, aluno);
58     else if (result_comp > 0)
59         return insereRec(&(*raiz)->dir, aluno);
60 }
61 return 1;
62 }
63
64 int insereElem(ABP *raiz, Aluno aluno) {
65     if (raiz == NULL)
66         return 0;
67     return insereRec(raiz, aluno);
68 }
69
70 int pesquisaRec(NO **raiz, Aluno aluno) {
71     if (*raiz == NULL)
72         return 0;
73
74     int result_comp = strcmp((*raiz)->info.nome, aluno.nome);
75
76     if (result_comp == 0)
77         return 1;
78     if (result_comp < 0)
79         return pesquisaRec(&(*raiz)->esq, aluno);
80     else
81         return pesquisaRec(&(*raiz)->dir, aluno);
82 }
83
84 int pesquisa(ABP *raiz, Aluno aluno) {
85     if (raiz == NULL)
86         return 0;
87     if (estaVazia(raiz))
88         return 0;
89     return pesquisaRec(raiz, aluno);
90 }
91
92 int removeRec(NO **raiz, Aluno aluno) {
93     if (*raiz == NULL)
94         return 0;
95
96     int result_comp = strcmp((*raiz)->info.nome, aluno.nome);
97
98     if (result_comp == 0) {
99         NO *aux;
100         if ((*raiz)->esq == NULL && (*raiz)->dir == NULL) {
101             // Caso 1 - NO sem filhos
102             printf("Caso 1: Liberando %s..\n", (*raiz)->info.nome);
103             liberarNO(*raiz);
104             *raiz = NULL;
105         } else if ((*raiz)->esq == NULL) {
106             // Caso 2.1 - Possui apenas uma subarvore direita
107             printf("Caso 2.1: Liberando %s..\n", (*raiz)->info.nome);
108             aux = *raiz;
109             *raiz = (*raiz)->dir;
110             liberarNO(aux);
111         } else if ((*raiz)->dir == NULL) {
112             // Caso 2.2 - Possui apenas uma subarvore esquerda
113             printf("Caso 2.2: Liberando %s..\n", (*raiz)->info.nome);
114             aux = *raiz;
115             *raiz = (*raiz)->esq;
116             liberarNO(aux);
117         } else {
118             // Caso 3 - Possui as duas subarvoretas (esq e dir)
119             // Duas estrategias:
120             // 3.1 - Substituir pelo NO com o MAIOR valor da subarvore esquerda
121             // 3.2 - Substituir pelo NO com o MENOR valor da subarvore direita
122             printf("Caso 3: Liberando %s..\n", (*raiz)->info.nome);
123             // Estrategia 3.1:
124             NO *Filho = (*raiz)->esq;
125             while (Filho->dir != NULL) // Localiza o MAIOR valor da subarvore esquerda
126                 Filho = Filho->dir;
127             (*raiz)->info = Filho->info;
128             Filho->info = aluno;
129             return removeRec(&(*raiz)->esq, aluno);
130         }
131         return 1;
132     } else if (result_comp < 0)
```

```
133     return removeRec(&(*raiz)->esq, aluno);
134 else
135     return removeRec(&(*raiz)->dir, aluno);
136 }
137
138 int removeElem(ABP *raiz, Aluno aluno) {
139     if (pesquisa(raiz, aluno) == 0) {
140         printf("Elemento inexistente!\n");
141         return 0;
142     }
143     return removeRec(raiz, aluno);
144 }
145
146 void em_ordem(NO *raiz, int nivel) {
147     if (raiz != NULL) {
148         em_ordem(raiz->esq, nivel + 1);
149         printf("[%s, %d] ", raiz->info.nome, nivel);
150         em_ordem(raiz->dir, nivel + 1);
151     }
152 }
153
154 void pre_ordem(NO *raiz, int nivel) {
155     if (raiz != NULL) {
156         printf("[%s, %d] ", raiz->info.nome, nivel);
157         pre_ordem(raiz->esq, nivel + 1);
158         pre_ordem(raiz->dir, nivel + 1);
159     }
160 }
161
162 void pos_ordem(NO *raiz, int nivel) {
163     if (raiz != NULL) {
164         pos_ordem(raiz->esq, nivel + 1);
165         pos_ordem(raiz->dir, nivel + 1);
166         printf("[%s, %d] ", raiz->info.nome, nivel);
167     }
168 }
169
170 void imprime(ABP *raiz) {
171     if (raiz == NULL)
172         return;
173     if (estaVazia(raiz)) {
174         printf("Arvore Vazia!\n");
175         return;
176     }
177     printf("\nEm Ordem: ");
178     em_ordem(*raiz, 0);
179     printf("\nPre Ordem: ");
180     pre_ordem(*raiz, 0);
181     printf("\nPos Ordem: ");
182     pos_ordem(*raiz, 0);
183     printf("\n");
184 }
185
186 int get_quantidade_nos(ABP *raiz) {
187     if (*raiz == NULL) {
188         return 0;
189     }
190
191     int qtd = 1;
192
193     qtd += get_quantidade_nos(&(*raiz)->esq);
194
195     qtd += get_quantidade_nos(&(*raiz)->dir);
196
197     return qtd;
198 }
199
200 Aluno *get_aluno_maior_nota(ABP *raiz) {
201     if (*raiz == NULL)
202         return NULL;
203
204     Aluno *melhor_esq = get_aluno_maior_nota(&(*raiz)->esq);
205     Aluno *melhor_dir = get_aluno_maior_nota(&(*raiz)->dir);
206
207     Aluno *melhor = &(*raiz)->info;
208
209     if (melhor_esq != NULL && (*melhor_esq).nota > (*melhor).nota) {
210         melhor = melhor_esq;
211     }
212
213     if (melhor_dir != NULL && (*melhor_dir).nota > (*melhor).nota) {
```

```
214     melhor = melhor_dir;
215 }
216
217 return melhor;
218 }
219
220 Aluno *get_aluno_pior_nota(ABP *raiz) {
221     if (*raiz == NULL)
222         return NULL;
223
224     Aluno *pior_esq = get_aluno_pior_nota(&(*raiz)->esq);
225     Aluno *pior_dir = get_aluno_pior_nota(&(*raiz)->dir);
226
227     Aluno *pior = &(*raiz)->info;
228
229     if (pior_esq != NULL && (*pior_esq).nota < (*pior).nota) {
230         pior = pior_esq;
231     }
232
233     if (pior_dir != NULL && (*pior_dir).nota < (*pior).nota) {
234         pior = pior_dir;
235     }
236
237     return pior;
238 }
239
```

roteiro_8/1-2.c

```
1  #include "ABP_aluno.h"
2
3  #define MAX_OPTIONS 12
4
5  enum options {
6      CRIAR = 0,
7      INSERIR,
8      BUSCAR,
9      REMOVER,
10     MELHOR_ALUNO,
11     PIOR_ALUNO,
12     IMPRIMIR_ORDEM,
13     IMPRIMIR_PRE_ORDEM,
14     IMPRIMIR_POS_ORDEM,
15     QUANTIDADE_NOS,
16     DESTRUIR,
17     SAIR,
18 };
19
20 int get_option() {
21     int opt = -1;
22     do {
23         printf("\tOperacoes\n");
24         printf("[%d] Criar árvore ABP, ", CRIAR);
25         printf("[%d] Inserir um aluno, ", INSERIR);
26         printf("[%d] Buscar um aluno, ", BUSCAR);
27         printf("[%d] Remover um aluno, ", REMOVER);
28         printf("[%d] Mostrar melhor aluno, ", MELHOR_ALUNO);
29         printf("[%d] Mostrar pior aluno, ", PIOR_ALUNO);
30         printf("[%d] Imprimir a ABP em ordem, ", IMPRIMIR_ORDEM);
31         printf("[%d] Imprimir a ABP em pré-ordem, ", IMPRIMIR_PRE_ORDEM);
32         printf("[%d] Imprimir a ABP em pós-ordem, ", IMPRIMIR_POS_ORDEM);
33         printf("[%d] Mostrar a quantidade de nós na ABP, ", QUANTIDADE_NOS);
34         printf("[%d] Destruir a ABP, ", DESTRUIR);
35         printf("[%d] Sair do programa \n", SAIR);
36
37         printf("\nInsira a opção desejada: ");
38         scanf("%d", &opt);
39         printf("\n");
40
41         if (opt < 0 || opt >= MAX_OPTIONS) {
42             printf("Opção escolhida inválida!\n");
43         }
44     } while (opt < 0 || opt >= MAX_OPTIONS);
45
46     return opt;
47 }
48
49
50 int get_valor(char *msg) {
51     int value;
52
53     printf("%s", msg);
```

```
54     scanf("%d", &value);
55
56     return value;
57 }
58
59 int main() {
60     int opt;
61
62     ABP *arvore = NULL;
63
64     do {
65         opt = get_option();
66
67         switch (opt) {
68             case CRIAR:
69                 printf("Executando comando...\n");
70
71                 if (arvore == NULL) {
72                     arvore = criaABP();
73                 } else {
74                     destroiABP(arvore);
75                     arvore = NULL;
76                     arvore = criaABP();
77                 }
78
79                 printf("Árvore ABP criada com sucesso!\n");
80                 break;
81             case INSERIR:
82                 printf("Executando comando...\n");
83
84                 if (arvore == NULL) {
85                     printf("Árvore ABP não inicializada impossível realizar operação..\n");
86                     break;
87                 }
88
89                 Aluno aluno;
90
91                 printf("Nome aluno:");
92                 scanf("%49s", aluno.nome);
93                 aluno.matricula = get_valor("Matricula: ");
94                 printf("Nota:");
95                 scanf("%lf", &aluno.nota);
96
97                 if (insereElem(arvore, aluno)) {
98                     printf("Aluno inserido na árvore com sucesso!\n");
99                 } else {
100                     printf("Falha ao inserir aluno!\n");
101                 }
102
103                 break;
104             case BUSCAR:
105                 printf("Executando comando...\n");
106
107                 if (arvore == NULL) {
108                     printf("Árvore ABP não inicializada impossível realizar operação..\n");
109                     break;
110                 }
111
112                 Aluno aluno_search;
113
114                 printf("Insira o nome do aluno para busca:");
115                 scanf("%49s", aluno_search.nome);
116
117                 if (pesquisa(arvore, aluno_search)) {
118                     printf("aluno está presente na árvore ABP!\n");
119                 } else {
120                     printf("aluno não está presente na árvore ABP!\n");
121                 }
122
123                 break;
124             case REMOVER:
125                 printf("Executando comando...\n");
126
127                 if (arvore == NULL) {
128                     printf("Árvore ABP não inicializada impossível realizar operação..\n");
129                     break;
130                 }
131
132                 Aluno aluno_removal;
133
134                 printf("Insira o nome do aluno para remover:");
```

```
135     scanf("%49s", aluno_removal.nome);
136
137     if (removeElem(arvore, aluno_removal)) {
138         printf("aluno removido com sucesso!\n");
139     }
140
141     break;
142 case MELHOR_ALUNO:
143     printf("Executando comando...\n");
144
145     if (arvore == NULL) {
146         printf("Árvore ABP não inicializada impossível realizar operação..\n");
147         break;
148     }
149
150     Aluno *aluno_temp = get_aluno_maior_nota(arvore);
151
152     if (aluno_temp != NULL) {
153         printf("0 aluno com melhor nota é:\n");
154         printf("Nome : %s Matricula: %d, Nota :%.2f\n", aluno_temp->nome,
155             aluno_temp->matricula, aluno_temp->nota);
156     } else {
157         printf("Falha ao buscar melhor aluno\n");
158     }
159
160     break;
161 case PIOR_ALUNO:
162     printf("Executando comando...\n");
163
164     if (arvore == NULL) {
165         printf("Árvore ABP não inicializada impossível realizar operação..\n");
166         break;
167     }
168
169     Aluno *aluno_temp_2 = get_aluno_pior_nota(arvore);
170
171     if (aluno_temp_2 != NULL) {
172         printf("0 aluno com pior nota é:\n");
173         printf("Nome : %s Matricula: %d, Nota :%.2f\n", aluno_temp_2->nome,
174             aluno_temp_2->matricula, aluno_temp_2->nota);
175     } else {
176         printf("Falha ao buscar pior aluno\n");
177     }
178
179     break;
180 case IMPRIMIR_ORDEM:
181     printf("Executando comando...\n");
182
183     if (arvore == NULL) {
184         printf("Árvore ABP não inicializada impossível realizar operação..\n");
185         break;
186     }
187
188     em_ordem(*arvore, 0);
189
190     printf("Árvore imprimida com sucesso!\n");
191
192     break;
193 case IMPRIMIR_PRE_ORDEM:
194     printf("Executando comando...\n");
195
196     if (arvore == NULL) {
197         printf("Árvore ABP não inicializada impossível realizar operação..\n");
198         break;
199     }
200
201     pre_ordem(*arvore, 0);
202
203     printf("Árvore imprimida com sucesso!\n");
204
205     break;
206 case IMPRIMIR_POS_ORDEM:
207     printf("Executando comando...\n");
208
209     if (arvore == NULL) {
210         printf("Árvore ABP não inicializada impossível realizar operação..\n");
211         break;
212     }
213
214     pos_ordem(*arvore, 0);
215
```

```
216     printf("Árvore imprimida com sucesso!\n");
217
218     break;
219 case QUANTIDADE_NOS:
220     printf("Executando comando...\n");
221
222     if (arvore == NULL) {
223         printf("Árvore ABP não inicializada impossível realizar operação..\n");
224         break;
225     }
226
227     int qtd_nos = get_quantidade_nos(arvore);
228
229     printf("A quantidade de nós é %d\n", qtd_nos);
230
231     break;
232 case DESTRUIR:
233     printf("Executando comando...\n");
234
235     if (arvore == NULL) {
236         printf("Árvore ABP não inicializada impossível realizar operação..\n");
237         break;
238     }
239
240     destroiABP(arvore);
241     arvore = NULL;
242
243     printf("Árvore ABP destruída com sucesso!");
244
245     break;
246 case SAIR:
247     if (arvore != NULL) {
248         destroiABP(arvore);
249         arvore = NULL;
250     }
251
252     printf("Finalizando programa! Até mais!\n");
253     break;
254 }
255 } while (opt != SAIR);
256
257 return 0;
258 }
```

Saída do terminal:

```

arthurdetomi at arthurdetomi-System-Product-Name in ~/Documents/UFSJ-Graduacao/UFSJ-2025_1/Lab_Prog_2/roteiro_8 on mainxxx 25-05-25 - 16:54:55
└─ ./1-2.out
    Operacoes
[0] Criar árvore ABP, [1] Inserir um aluno, [2] Buscar um aluno, [3] Remover um aluno, [4] Mostrar melhor aluno, [5] Mostrar pior aluno, [6] Imprimir a A
[10] Destruir a ABP, [11] Sair do programa

Insira a opção desejada: 0

Executando comando...
Árvore ABP criada com sucesso!
    Operacoes
[0] Criar árvore ABP, [1] Inserir um aluno, [2] Buscar um aluno, [3] Remover um aluno, [4] Mostrar melhor aluno, [5] Mostrar pior aluno, [6] Imprimir a A
[10] Destruir a ABP, [11] Sair do programa

Insira a opção desejada: 1

Executando comando...
Nome aluno:Marcos
Matricula: 23847
Nota:1.2
Aluno inserido na árvore com sucesso!
    Operacoes
[0] Criar árvore ABP, [1] Inserir um aluno, [2] Buscar um aluno, [3] Remover um aluno, [4] Mostrar melhor aluno, [5] Mostrar pior aluno, [6] Imprimir a A
[10] Destruir a ABP, [11] Sair do programa

Insira a opção desejada: 1

Executando comando...
Nome aluno:Maria
Matricula: 1345
Nota:9.8
Aluno inserido na árvore com sucesso!
    Operacoes
[0] Criar árvore ABP, [1] Inserir um aluno, [2] Buscar um aluno, [3] Remover um aluno, [4] Mostrar melhor aluno, [5] Mostrar pior aluno, [6] Imprimir a A
[10] Destruir a ABP, [11] Sair do programa

Insira a opção desejada: 1

Executando comando...
Nome aluno:Luis
Matricula: 3875
Nota:7.6
Aluno inserido na árvore com sucesso!
    Operacoes
[0] Criar árvore ABP, [1] Inserir um aluno, [2] Buscar um aluno, [3] Remover um aluno, [4] Mostrar melhor aluno, [5] Mostrar pior aluno, [6] Imprimir a A
[10] Destruir a ABP, [11] Sair do programa

Insira a opção desejada: 4

Executando comando...
0 aluno com melhor nota é:
Nome : Maria Matricula: 1345, Nota :9.80
    Operacoes
[0] Criar árvore ABP, [1] Inserir um aluno, [2] Buscar um aluno, [3] Remover um aluno, [4] Mostrar melhor aluno, [5] Mostrar pior aluno, [6] Imprimir a A
[10] Destruir a ABP, [11] Sair do programa

Insira a opção desejada: 5

Executando comando...
0 aluno com pior nota é:

```

A complexidade da função que busca o aluno com a pior nota e da função que busca o com a melhor nota é $O(n)$, onde n representa a quantidade de nós da árvore. Como a chave da árvore não é a nota do aluno, é necessário visitar todos os nós para comparar as notas e encontrar a menor ou maior nota.