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

```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();

void liberarNO(NO *q);

ABP *criaABP();

void destroiRec(NO *no);

void destroiABP(ABP *raiz);

int estaVazia(ABP *raiz);

int insereRec(NO **raiz, int elem);

int insereIte(NO **raiz, int elem);

int insereElem(ABP *raiz, int elem);

int pesquisaRec(NO **raiz, int elem);

int pesquisaIte(NO **raiz, int elem);

int pesquisa(ABP *raiz, int elem);

int removeRec(NO **raiz, int elem);

NO *removeAtual(NO *atual);

int removeIte(NO **raiz, int elem);

int removeElem(ABP *raiz, int elem);
void em_ordem(NO *raiz, int nivel);

void pre_ordem(NO *raiz, int nivel);

void pos_ordem(NO *raiz, int nivel);

void imprime(ABP *raiz);

int get_quantidade_nos(ABP *raiz);

#endif
```

roteiro_8/ABP.c

```c

#include "ABP.h"

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

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

```

```c
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;
  } 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;
}

```

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

int pesquisa(ABP *raiz, int elem) {
  if (raiz == NULL)
    return 0;
  if (estaVazia(raiz))
    return 0;
  return pesquisaRec(raiz, elem);
  // return pesquisaIte(raiz, elem);
}

int removeRec(NO **raiz, int elem) {
  if (*raiz == NULL)
    return 0;
  if ((*raiz)->info == elem) {
    NO *aux;
    if ((*raiz)->esq == NULL && (*raiz)->dir == NULL) {
      // Caso 1 - NO sem filhos
      printf("Caso 1: Liberando %d..\n", (*raiz)->info);
      liberarNO(*raiz);
      *raiz = NULL;
    } else if ((*raiz)->esq == NULL) {
      // Caso 2.1 - Possui apenas uma subarvore direita
      printf("Caso 2.1: Liberando %d..\n", (*raiz)->info);
      aux = *raiz;
      *raiz = (*raiz)->dir;
      liberarNO(aux);
    } else if ((*raiz)->dir == NULL) {
      // Caso 2.2 - Possui apenas uma subarvore esquerda
      printf("Caso 2.2: Liberando %d..\n", (*raiz)->info);
      aux = *raiz;
      *raiz = (*raiz)->esq;
      liberarNO(aux);
    } else {
      // Caso 3 - Possui as duas subarvores (esq e dir)
      // Duas estrategias:
      // 3.1 - Substituir pelo NO com o MAIOR valor da subarvore esquerda
      // 3.2 - Substituir pelo NO com o MENOR valor da subarvore direita
      printf("Caso 3: Liberando %d..\n", (*raiz)->info);
      // Estrategia 3.1:
      NO *Filho = (*raiz)->esq;
      while (Filho->dir != NULL) // Localiza o MAIOR valor da subarvore esquerda
        Filho = Filho->dir;
      (*raiz)->info = Filho->info;
      Filho->info = elem;
      return removeRec(&(*raiz)->esq, elem);
    }
    return 1;
  } else if (elem < (*raiz)->info)
    return removeRec(&(*raiz)->esq, elem);
  else
    return removeRec(&(*raiz)->dir, elem);
```

```c
}

NO *removeAtual(NO *atual) {
  NO *no1, *no2;
  // Ambos casos no if(atual->esq == NULL)
  // Caso 1 - NO sem filhos
  // Caso 2.1 - Possui apenas uma subarvore direita
  if (atual->esq == NULL) {
    no2 = atual->dir;
    liberarNO(atual);
    return no2;
  }
  // Caso 3 - Possui as duas subarvores (esq e dir)
  // Estrategia:

  no1 = atual;
  no2 = atual->esq;
  while (no2->dir != NULL) {
    no1 = no2;
    no2 = no2->dir;
  }
  if (no1 != atual) {
    no1->dir = no2->esq;
    no2->esq = atual->esq;
  }
  no2->dir = atual->dir;
  liberarNO(atual);
  return no2;
}

int removeIte(NO **raiz, int elem) {
  if (*raiz == NULL)
    return 0;
  NO *atual = *raiz, *ant = NULL;
  while (atual != NULL) {
    if (elem == atual->info) {
      if (atual == *raiz)
        *raiz = removeAtual(atual);
      else {
        if (ant->dir == atual)
          ant->dir = removeAtual(atual);
        else
          ant->esq = removeAtual(atual);
      }
      return 1;
    }
    ant = atual;
    if (elem < atual->info)
      atual = atual->esq;
    else
      atual = atual->dir;
  }
  return 0;
}

int removeElem(ABP *raiz, int elem) {
  if (pesquisa(raiz, elem) == 0) {
    printf("Elemento inexistente!\n");
    return 0;
  }
  // return removeRec(raiz, elem);
  return removeIte(raiz, elem);
}

void em_ordem(NO *raiz, int nivel) {
  if (raiz != NULL) {
    em_ordem(raiz->esq, nivel + 1);
    printf("[%d, %d] ", raiz->info, nivel);
    em_ordem(raiz->dir, nivel + 1);
  }
}

void pre_ordem(NO *raiz, int nivel) {
  if (raiz != NULL) {
    printf("[%d, %d] ", raiz->info, nivel);
    pre_ordem(raiz->esq, nivel + 1);
    pre_ordem(raiz->dir, nivel + 1);
  }
}

void pos_ordem(NO *raiz, int nivel) {
```

```c
  if (raiz != NULL) {
    pos_ordem(raiz->esq, nivel + 1);
    pos_ordem(raiz->dir, nivel + 1);
    printf("[%d, %d] ", raiz->info, nivel);
  }
}

void imprime(ABP *raiz) {
  if (raiz == NULL)
    return;
  if (estaVazia(raiz)) {
    printf("Arvore Vazia!\n");
    return;
  }
  printf("\nEm Ordem: ");
  em_ordem(*raiz, 0);
  printf("\nPre Ordem: ");
  pre_ordem(*raiz, 0);
  printf("\nPos Ordem: ");
  pos_ordem(*raiz, 0);
  printf("\n");
}

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

  int qtd = 1;

  qtd += get_quantidade_nos(&(*raiz)->esq);

  qtd += get_quantidade_nos(&(*raiz)->dir);

  return qtd;
}

```

roteiro_8/1-1.c

```c
#include "ABP.h"

#define MAX_OPTIONS 10

enum options {
  CRIAR = 0,
  INSERIR,
  BUSCAR,
  REMOVER,
  IMPRIMIR_ORDEM,
  IMPRIMIR_PRE_ORDEM,
  IMPRIMIR_POS_ORDEM,
  QUANTIDADE_NOS,
  DESTRUIR,
  SAIR,
};

int get_option() {
  int opt = -1;
  do {
    printf("\tOperacoes\n");
    printf("[%d] Criar árvore ABP, ", CRIAR);
    printf("[%d] Inserir um elemento, ", INSERIR);
    printf("[%d] Buscar um elemento, ", BUSCAR);
    printf("[%d] Remover um elemento, ", REMOVER);
    printf("[%d] Imprimir a ABP em ordem, ", IMPRIMIR_ORDEM);
    printf("[%d] Imprimir a ABP em pré-ordem, ", IMPRIMIR_PRE_ORDEM);
    printf("[%d] Imprimir a ABP em pós-ordem, ", IMPRIMIR_POS_ORDEM);
    printf("[%d] Mostrar a quantidade de nós na ABP, ", QUANTIDADE_NOS);
    printf("[%d] Destruir a ABP, ", DESTRUIR);
    printf("[%d] Sair do programa \n", SAIR);

    printf("\nInsira a opção desejada: ");
    scanf("%d", &opt);
    printf("\n");

    if (opt < 0 || opt >= MAX_OPTIONS) {
      printf("Opção escolhida inválida!\n");
    }

  } while (opt < 0 || opt >= MAX_OPTIONS);

```

```c
  return opt;
}

int get_valor(char *msg) {
  int value;

  printf("%s", msg);
  scanf("%d", &value);

  return value;
}

int main() {
  int opt, valor;

  ABP *arvore = NULL;

  do {
    opt = get_option();

    switch (opt) {
    case CRIAR:
      printf("Executando comando...\n");

      if (arvore == NULL) {
        arvore = criaABP();
      } else {
        destroiABP(arvore);
        arvore = NULL;
        arvore = criaABP();
      }

      printf("Árvore ABP criada com sucesso!\n");
      break;
    case INSERIR:
      printf("Executando comando...\n");

      if (arvore == NULL) {
        printf("Árvore ABP não incializada impossível realizar operação..\n");
        break;
      }

      valor = get_valor("Insira um valor: ");

      if (insereElem(arvore, valor)) {
        printf("Elemento inserido na árvore com sucesso!\n");
      } else {
        printf("Falha ao inserir elemento!\n");
      }

      break;
    case BUSCAR:
      printf("Executando comando...\n");

      if (arvore == NULL) {
        printf("Árvore ABP não incializada impossível realizar operação..\n");
        break;
      }

      valor = get_valor("Insira o valor para buscar: ");

      if (pesquisa(arvore, valor)) {
        printf("Elemento está presente na árvore ABP!\n");
      } else {
        printf("Elemento não está presente na árvore ABP!\n");
      }

      break;
    case REMOVER:
      printf("Executando comando...\n");

      if (arvore == NULL) {
        printf("Árvore ABP não incializada impossível realizar operação..\n");
        break;
      }

      valor = get_valor("Elemento a se remover: ");

      if (removeElem(arvore, valor)) {
        printf("Elemento removido com sucesso!\n");
      }
```

```
124
125       break;
126     case IMPRIMIR_ORDEM:
127       printf("Executando comando...\n");
128
129       if (arvore == NULL) {
130         printf("Árvore ABP não incializada 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 incializada 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 incializada 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 incializada 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 incializada impossível realizar operação..\n");
183         break;
184       }
185
186       destroiABP(arvore);
187       arvore = NULL;
188
189       printf("Árvore ABP destruida 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-or

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-or

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-or

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-or

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-or

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-or

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-or

Insira a opção desejada: 8

Executando comando...
Árvore ABP destruida 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-or

Insira a opção desejada: 9

Finalizando programa! Até mais!
```

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

roteiro_8/ABP_aluno.h

```h
#ifndef ABP_ALUNO_H
#define ABP_ALUNO_H

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

typedef struct Aluno {
  char nome[50];
  int matricula;
  double nota;
} Aluno;

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

typedef struct NO *ABP;

NO *alocarNO();

void liberarNO(NO *q);

ABP *criaABP();

void destroiRec(NO *no);

void destroiABP(ABP *raiz);

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
```

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

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

int pesquisaRec(NO **raiz, Aluno aluno) {
  if (*raiz == NULL)
    return 0;

  int result_comp = strcmp((*raiz)->info.nome, aluno.nome);

  if (result_comp == 0)
    return 1;
  if (result_comp < 0)
    return pesquisaRec(&(*raiz)->esq, aluno);
  else
    return pesquisaRec(&(*raiz)->dir, aluno);
}

int pesquisa(ABP *raiz, Aluno aluno) {
  if (raiz == NULL)
    return 0;
  if (estaVazia(raiz))
    return 0;
  return pesquisaRec(raiz, aluno);
}

int removeRec(NO **raiz, Aluno aluno) {
  if (*raiz == NULL)
    return 0;

  int result_comp = strcmp((*raiz)->info.nome, aluno.nome);

  if (result_comp == 0) {
    NO *aux;
    if ((*raiz)->esq == NULL && (*raiz)->dir == NULL) {
      // Caso 1 - NO sem filhos
      printf("Caso 1: Liberando %s..\n", (*raiz)->info.nome);
      liberarNO(*raiz);
      *raiz = NULL;
    } else if ((*raiz)->esq == NULL) {
      // Caso 2.1 - Possui apenas uma subarvore direita
      printf("Caso 2.1: Liberando %s..\n", (*raiz)->info.nome);
      aux = *raiz;
      *raiz = (*raiz)->dir;
      liberarNO(aux);
    } else if ((*raiz)->dir == NULL) {
      // Caso 2.2 - Possui apenas uma subarvore esquerda
      printf("Caso 2.2: Liberando %s..\n", (*raiz)->info.nome);
      aux = *raiz;
      *raiz = (*raiz)->esq;
      liberarNO(aux);
    } else {
      // Caso 3 - Possui as duas subarvores (esq e dir)
      // Duas estrategias:
      // 3.1 - Substituir pelo NO com o MAIOR valor da subarvore esquerda
      // 3.2 - Substituir pelo NO com o MENOR valor da subarvore direita
      printf("Caso 3: Liberando %s..\n", (*raiz)->info.nome);
      // Estrategia 3.1:
      NO *Filho = (*raiz)->esq;
      while (Filho->dir != NULL) // Localiza o MAIOR valor da subarvore esquerda
        Filho = Filho->dir;
      (*raiz)->info = Filho->info;
      Filho->info = aluno;
      return removeRec(&(*raiz)->esq, aluno);
    }
    return 1;
  } else if (result_comp < 0)
```

```
    return removeRec(&(*raiz)->esq, aluno);
  else
    return removeRec(&(*raiz)->dir, aluno);
}

int removeElem(ABP *raiz, Aluno aluno) {
  if (pesquisa(raiz, aluno) == 0) {
    printf("Elemento inexistente!\n");
    return 0;
  }
  return removeRec(raiz, aluno);
}

void em_ordem(NO *raiz, int nivel) {
  if (raiz != NULL) {
    em_ordem(raiz->esq, nivel + 1);
    printf("[%s, %d] ", raiz->info.nome, nivel);
    em_ordem(raiz->dir, nivel + 1);
  }
}

void pre_ordem(NO *raiz, int nivel) {
  if (raiz != NULL) {
    printf("[%s, %d] ", raiz->info.nome, nivel);
    pre_ordem(raiz->esq, nivel + 1);
    pre_ordem(raiz->dir, nivel + 1);
  }
}

void pos_ordem(NO *raiz, int nivel) {
  if (raiz != NULL) {
    pos_ordem(raiz->esq, nivel + 1);
    pos_ordem(raiz->dir, nivel + 1);
    printf("[%s, %d] ", raiz->info.nome, nivel);
  }
}

void imprime(ABP *raiz) {
  if (raiz == NULL)
    return;
  if (estaVazia(raiz)) {
    printf("Arvore Vazia!\n");
    return;
  }
  printf("\nEm Ordem: ");
  em_ordem(*raiz, 0);
  printf("\nPre Ordem: ");
  pre_ordem(*raiz, 0);
  printf("\nPos Ordem: ");
  pos_ordem(*raiz, 0);
  printf("\n");
}

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

  int qtd = 1;

  qtd += get_quantidade_nos(&(*raiz)->esq);

  qtd += get_quantidade_nos(&(*raiz)->dir);

  return qtd;
}

Aluno *get_aluno_maior_nota(ABP *raiz) {
  if (*raiz == NULL)
    return NULL;

  Aluno *melhor_esq = get_aluno_maior_nota(&(*raiz)->esq);
  Aluno *melhor_dir = get_aluno_maior_nota(&(*raiz)->dir);

  Aluno *melhor = &(*raiz)->info;

  if (melhor_esq != NULL && (*melhor_esq).nota > (*melhor).nota) {
    melhor = melhor_esq;
  }

  if (melhor_dir != NULL && (*melhor_dir).nota > (*melhor).nota) {
```

```
    melhor = melhor_dir;
  }

  return melhor;
}

Aluno *get_aluno_pior_nota(ABP *raiz) {
  if (*raiz == NULL)
    return NULL;

  Aluno *pior_esq = get_aluno_pior_nota(&(*raiz)->esq);
  Aluno *pior_dir = get_aluno_pior_nota(&(*raiz)->dir);

  Aluno *pior = &(*raiz)->info;

  if (pior_esq != NULL && (*pior_esq).nota < (*pior).nota) {
    pior = pior_esq;
  }

  if (pior_dir != NULL && (*pior_dir).nota < (*pior).nota) {
    pior = pior_dir;
  }

  return pior;
}

```

roteiro_8/1-2.c

```c
#include "ABP_aluno.h"

#define MAX_OPTIONS 12

enum options {
  CRIAR = 0,
  INSERIR,
  BUSCAR,
  REMOVER,
  MELHOR_ALUNO,
  PIOR_ALUNO,
  IMPRIMIR_ORDEM,
  IMPRIMIR_PRE_ORDEM,
  IMPRIMIR_POS_ORDEM,
  QUANTIDADE_NOS,
  DESTRUIR,
  SAIR,
};

int get_option() {
  int opt = -1;
  do {
    printf("\tOperacoes\n");
    printf("[%d] Criar árvore ABP, ", CRIAR);
    printf("[%d] Inserir um aluno, ", INSERIR);
    printf("[%d] Buscar um aluno, ", BUSCAR);
    printf("[%d] Remover um aluno, ", REMOVER);
    printf("[%d] Mostrar melhor aluno, ", MELHOR_ALUNO);
    printf("[%d] Mostrar pior aluno, ", PIOR_ALUNO);
    printf("[%d] Imprimir a ABP em ordem, ", IMPRIMIR_ORDEM);
    printf("[%d] Imprimir a ABP em pré-ordem, ", IMPRIMIR_PRE_ORDEM);
    printf("[%d] Imprimir a ABP em pós-ordem, ", IMPRIMIR_POS_ORDEM);
    printf("[%d] Mostrar a quantidade de nós na ABP, ", QUANTIDADE_NOS);
    printf("[%d] Destruir a ABP, ", DESTRUIR);
    printf("[%d] Sair do programa \n", SAIR);

    printf("\nInsira a opção desejada: ");
    scanf("%d", &opt);
    printf("\n");

    if (opt < 0 || opt >= MAX_OPTIONS) {
      printf("Opção escolhida inválida!\n");
    }

  } while (opt < 0 || opt >= MAX_OPTIONS);

  return opt;
}

int get_valor(char *msg) {
  int value;

  printf("%s", msg);
```

```
  scanf("%d", &value);

  return value;
}

int main() {
  int opt;

  ABP *arvore = NULL;

  do {
    opt = get_option();

    switch (opt) {
    case CRIAR:
      printf("Executando comando...\n");

      if (arvore == NULL) {
        arvore = criaABP();
      } else {
        destroiABP(arvore);
        arvore = NULL;
        arvore = criaABP();
      }

      printf("Árvore ABP criada com sucesso!\n");
      break;
    case INSERIR:
      printf("Executando comando...\n");

      if (arvore == NULL) {
        printf("Árvore ABP não incializada impossível realizar operação..\n");
        break;
      }

      Aluno aluno;

      printf("Nome aluno:");
      scanf("%49s", aluno.nome);
      aluno.matricula = get_valor("Matricula: ");
      printf("Nota:");
      scanf("%lf", &aluno.nota);

      if (insereElem(arvore, aluno)) {
        printf("Aluno inserido na árvore com sucesso!\n");
      } else {
        printf("Falha ao inserir aluno!\n");
      }

      break;
    case BUSCAR:
      printf("Executando comando...\n");

      if (arvore == NULL) {
        printf("Árvore ABP não incializada impossível realizar operação..\n");
        break;
      }

      Aluno aluno_search;

      printf("Insira o nome do aluno para busca:");
      scanf("%49s", aluno_search.nome);

      if (pesquisa(arvore, aluno_search)) {
        printf("aluno está presente na árvore ABP!\n");
      } else {
        printf("aluno não está presente na árvore ABP!\n");
      }

      break;
    case REMOVER:
      printf("Executando comando...\n");

      if (arvore == NULL) {
        printf("Árvore ABP não incializada impossível realizar operação..\n");
        break;
      }

      Aluno aluno_removal;

      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 incializada 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("O 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 incializada 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("O 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 incializada 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 incializada 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 incializada impossível realizar operação..\n");
211         break;
212       }
213
214       pos_ordem(*arvore, 0);
215
```

```c
      printf("Árvore imprimida com sucesso!\n");

      break;
    case QUANTIDADE_NOS:
      printf("Executando comando...\n");

      if (arvore == NULL) {
        printf("Árvore ABP não incializada impossível realizar operação..\n");
        break;
      }

      int qtd_nos = get_quantidade_nos(arvore);

      printf("A quantidade de nós é %d\n", qtd_nos);

      break;
    case DESTRUIR:
      printf("Executando comando...\n");

      if (arvore == NULL) {
        printf("Árvore ABP não incializada impossível realizar operação..\n");
        break;
      }

      destroiABP(arvore);
      arvore = NULL;

      printf("Árvore ABP destruida com sucesso!");

      break;
    case SAIR:
      if (arvore != NULL) {
        destroiABP(arvore);
        arvore = NULL;
      }

      printf("Finalizando programa! Até mais!\n");
      break;
    }
  } while (opt != SAIR);

  return 0;
}
```

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...
O 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...
O 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.