Universidade Federal de São João del Rei
Departamento de Ciência da Computação
Curso de Ciência da Computação

Roteiro 8

Adélson de Oliveira Carmo Júnior
212050019

1 Árvore Binária de Pesquisa (ABP)

1.1 Reimplentação

Código

```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 tipo);

void aguardaLimpa();
void contador(NO* raiz, int nivel, int
    *cont);


#endif
```

codigos/questao11/questao11.h

```c
#include <stdio.h>
#include <stdlib.h>
#include "questao11.h"

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;
```

```
22      destroiRec(no->esq);
23      destroiRec(no->dir);
24      liberarNO(no);
25      no = NULL;
26  }
27
28  void destroiABP(ABP* raiz){
29      if(raiz != NULL){
30          destroiRec(*raiz);
31          free(raiz);
32      }
33  }
34
35  int estaVazia(ABP* raiz){
36      if(raiz == NULL) return 0;
37      return (*raiz == NULL);
38  }
39
40
41  int insereRec(NO** raiz, int elem){
42      if(*raiz == NULL){
43          NO* novo = alocarNO();
44          if(novo == NULL) return 0;
45          novo->info = elem;
46          novo->esq = NULL; novo->dir =
                NULL;
47          *raiz = novo;
48      }else{
49          if((*raiz)->info == elem){
50              printf("Elemento
                    Existente!\n");
51              return 0;
52          }
53          if(elem < (*raiz)->info)
54              return
                    insereRec(&(*raiz)->esq,
                    elem);
55          else if(elem > (*raiz)->info)
56              return
                    insereRec(&(*raiz)->dir,
                    elem);
57      }
58      return 1;
59  }
60
61  int insereIte(NO** raiz, int elem){
62      NO *aux = *raiz, *ant = NULL;
63      while (aux != NULL){
64          ant = aux;
65          if(aux->info == elem){
66              printf("Elemento
                    Existente!\n");
67              return 0;
68          }
69          if(elem < aux->info) aux =
                aux->esq;
70          else aux = aux->dir;
71      }
72      NO* novo = alocarNO();
73      if(novo == NULL) return 0;
74      novo->info = elem;
75      novo->esq = NULL; novo->dir = NULL;
76      if(ant == NULL){
77          *raiz = novo;
78      }else{
79          if(elem < ant->info) ant->esq =
                novo;
80          else ant->dir = novo;
81      }
82      return 1;
83  }
84
85  int insereElem(ABP* raiz, int elem){
86      if(raiz == NULL) return 0;
87      return insereRec(raiz, elem);
88      //return insereIte(raiz, elem);
89  }
90
91  int pesquisaRec(NO** raiz, int elem){
92      if(*raiz == NULL) return 0;
93      if((*raiz)->info == elem) return 1;
94      if(elem < (*raiz)->info)
95          return
                pesquisaRec(&(*raiz)->esq,
                elem);
96      else
97          return
                pesquisaRec(&(*raiz)->dir,
                elem);
98  }
99
100 int pesquisaIte(NO** raiz, int elem){
101     NO* aux = *raiz;
102     while(aux != NULL){
103         if(aux->info == elem) return 1;
104         if(elem < aux->info)
105             aux = aux->esq;
106         else
107             aux = aux->dir;
108     }
109     return 0;
110 }
111
112 int pesquisa(ABP* raiz, int elem){
113     if(raiz == NULL) return 0;
114     if(estaVazia(raiz)) return 0;
115     return pesquisaRec(raiz, elem);
116     //return pesquisaIte(raiz, elem);
117 }
118
119 int removeRec(NO** raiz, int elem){
120     if(*raiz == NULL) return 0;
121     if((*raiz)->info == elem){
122         NO* aux;
123         if((*raiz)->esq == NULL &&
                (*raiz)->dir == NULL){
124             //Caso 1 - NO sem filhos
125             printf("Caso 1: Liberando
```

```
                %d..\n", (*raiz)->info);
126             liberarNO(*raiz);
127             *raiz = NULL;
128         }else if((*raiz)->esq == NULL){
129             //Caso 2.1 - Possui apenas
                    uma subarvore direita
130             printf("Caso 2.1: Liberando
                    %d..\n", (*raiz)->info);
131             aux = *raiz;
132             *raiz = (*raiz)->dir;
133             liberarNO(aux);
134         }else if((*raiz)->dir == NULL){
135             //Caso 2.2 - Possui apenas
                    uma subarvore esquerda
136             printf("Caso 2.2: Liberando
                    %d..\n", (*raiz)->info);
137             aux = *raiz;
138             *raiz = (*raiz)->esq;
139             liberarNO(aux);
140         }else{
141             //Caso 3 - Possui as duas
                    subarvores (esq e dir)
142             //Duas estrategias:
143             //3.1 - Substituir pelo NO
                    com o MAIOR valor da
                    subarvore esquerda
144             //3.2 - Substituir pelo NO
                    com o MENOR valor da
                    subarvore direita
145             printf("Caso 3: Liberando
                    %d..\n", (*raiz)->info);
146             //Estrategia 3.1:
147             NO* Filho = (*raiz)->esq;
148             while(Filho->dir !=
                    NULL)//Localiza o MAIOR
                    valor da subarvore
                    esquerda
149                 Filho = Filho->dir;
150             (*raiz)->info = Filho->info;
151             Filho->info = elem;
152             return
                    removeRec(&(*raiz)->esq,
                    elem);
153         }
154         return 1;
155     }else if(elem < (*raiz)->info)
156         return removeRec(&(*raiz)->esq,
                elem);
157     else
158         return removeRec(&(*raiz)->dir,
                elem);
159 }
160
161 NO* removeAtual(NO* atual){
162     NO* no1, *no2;
163     //Ambos casos no if(atual->esq ==
            NULL)
164     //Caso 1 - NO sem filhos
165     //Caso 2.1 - Possui apenas uma
            subarvore direita
166     if(atual->esq == NULL){
167         no2 = atual->dir;
168         liberarNO(atual);
169         return no2;
170     }
171     //Caso 3 - Possui as duas
            subarvores (esq e dir)
172     //Estrategia:
173
174     no1 = atual;
175     no2 = atual->esq;
176     while(no2->dir != NULL){
177         no1 = no2;
178         no2 = no2->dir;
179     }
180     if(no1 != atual){
181         no1->dir = no2->esq;
182         no2->esq = atual->esq;
183     }
184     no2->dir = atual->dir;
185     liberarNO(atual);
186     return no2;
187 }
188
189 int removeIte(NO** raiz, int elem){
190     if(*raiz == NULL) return 0;
191     NO* atual = *raiz, *ant = NULL;
192     while(atual != NULL){
193         if(elem == atual->info){
194             if(atual == *raiz)
195                 *raiz =
                        removeAtual(atual);
196             else{
197                 if(ant->dir == atual)
198                     ant->dir =
                            removeAtual(atual);
199                 else
200                     ant->esq =
                            removeAtual(atual);
201             }
202             return 1;
203         }
204         ant = atual;
205         if(elem < atual->info)
206             atual = atual->esq;
207         else
208             atual = atual->dir;
209     }
210     return 0;
211 }
212
213 int removeElem(ABP* raiz, int elem){
214     if(pesquisa(raiz, elem) == 0){
215         printf("Elemento
                inexistente!\n");
216         return 0;
217     }
218     //return removeRec(raiz, elem);
```

```
219     return removeIte(raiz, elem);
220 }
221
222 void em_ordem(NO* raiz, int nivel){
223     if(raiz != NULL){
224         em_ordem(raiz->esq, nivel+1);
225         printf("[%d, %d] ",
              raiz->info,nivel);
226         em_ordem(raiz->dir, nivel+1);
227     }
228 }
229
230 void pre_ordem(NO* raiz, int nivel){
231     if(raiz != NULL){
232         printf("[%d, %d] ",
              raiz->info,nivel);
233         pre_ordem(raiz->esq, nivel+1);
234         pre_ordem(raiz->dir, nivel+1);
235     }
236 }
237
238 void pos_ordem(NO* raiz, int nivel){
239     if(raiz != NULL){
240         pos_ordem(raiz->esq, nivel+1);
241         pos_ordem(raiz->dir, nivel+1);
242         printf("[%d, %d] ",
              raiz->info,nivel);
243     }
244 }
245
246 void imprime(ABP* raiz, int tipo){
247     if(raiz == NULL) return;
248     if(estaVazia(raiz)){
249         printf("Arvore Vazia!\n");
250         return;
251     }
252     switch (tipo){
253     case 0:
254         printf("\nEm Ordem: ");
              em_ordem(*raiz, 0);
255         break;
256
257     case 1:
258         printf("\nPre Ordem: ");
              pre_ordem(*raiz, 0);
259         break;
260
261     case 2:
262         printf("\nPos Ordem: ");
              pos_ordem(*raiz, 0);
263         break;
264     default:
265         break;
266     }
267     printf("\n");
268 }
269
270 /* Funcao para esperar resposta do
      usuario e depois limpar*/
271 void aguardaLimpa(){
272     getchar();
273     printf("\n\nAperte qualquer tecla
          para continuar\n");
274     getchar();
275     system("clear");
276 }
277
278 void contador(NO* raiz, int nivel, int
      *cont){
279     if(raiz != NULL){
280         contador(raiz->esq, nivel+1,
              cont);
281         (*cont)++;
282         contador(raiz->dir, nivel+1,
              cont);
283     }
284 }
```

codigos/questao11/questao11.c

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include "questao11.h"
 4
 5 int main(){
 6     int *cont =
         (int*)malloc(sizeof(int));
 7     int escolha, elem, busca;
 8     ABP *abt;
 9     do{
10         printf("O que deseja fazer?\n1
              - Criar ABP\n2 - Inserir um
              elemento\n"
11         "3 - Buscar um elemento\n4 -
              Remover um elemento\n5 -
              Imprimir a ABP em ordem"
12         "\n6 - Imprimir a ABP em
              pre-ordem\n7 - Imprimir a
              ABP em pos-ordem\n"
13         "8 - Mostrar a quantidade de
              nos na ABP\n9 - Destruir a
              ABP\n10 - Sair\n");
14         scanf("%d",&escolha);
15
16         switch (escolha){
17         case 10:
18             break;
19
20         case 1:
21             abt = criaABP();
22             printf("Arvore criada!");
23             aguardaLimpa();
```

```
24              break;
25
26          case 2:
27              printf("Digite o elemento
                    que deseja inserir: ");
28              scanf("%d",&elem);
29              insereElem(abt,elem);
30              aguardaLimpa();
31              break;
32
33          case 3:
34              printf("Digite o elemento
                    que deseja consultar: ");
35              scanf("%d",&elem);
36              busca = pesquisa(abt,elem);
37              if(busca == 1)
38                  printf("O numero esta
                        na arvore!");
39              else
40                  printf("O numero nao
                        esta na arvore!");
41              aguardaLimpa();
42              break;
43
44          case 4:
45              printf("Digite o elemento
                    que deseja remover: ");
46              scanf("%d",&elem);
47              removeElem(abt,elem);
48              aguardaLimpa();
49              break;
50
51          case 5:
52              imprime(abt,0);
53              aguardaLimpa();
54              break;
55
```

```
56          case 6:
57              imprime(abt,1);
58              aguardaLimpa();
59              break;
60
61          case 7:
62              imprime(abt,2);
63              aguardaLimpa();
64              break;
65
66          case 8:
67              *cont = 0;
68              contador(*abt,0,cont);
69              printf("O numero de nos eh:
                    %d!", *cont);
70              aguardaLimpa();
71              break;
72
73          case 9:
74              destroiABP(abt);
75              printf("Arvore destruida!");
76              aguardaLimpa();
77              break;
78
79          default:
80              printf("\nAlgo de errado
                    nao esta certo!\n\n");
81              break;
82          }
83      }while(escolha!=10);
84
85
86
87      return 0;
88  }
```

codigos/questao11/main.c

```
1 all: questao11.o
2   gcc questao11.o main.c -o main
3
4 questao11.o: questao11.h questao11.c
5   gcc -c questao11.c
6
7 clean:
8   rm -f questao11.o main
```

codigos/questao11/Makefile

Saída

```
O que deseja fazer?
1 - Criar ABP
2 - Inserir um elemento
3 - Buscar um elemento
4 - Remover um elemento
5 - Imprimir a ABP em ordem
6 - Imprimir a ABP em pre-ordem
7 - Imprimir a ABP em pos-ordem
8 - Mostrar a quantidade de nos na ABP
9 - Destruir a ABP
10 - Sair
1
Arvore criada!
```

Figura 1: Questão 2.1 - Cria árvore

```
O que deseja fazer?
1 - Criar ABP
2 - Inserir um elemento
3 - Buscar um elemento
4 - Remover um elemento
5 - Imprimir a ABP em ordem
6 - Imprimir a ABP em pre-ordem
7 - Imprimir a ABP em pos-ordem
8 - Mostrar a quantidade de nos na ABP
9 - Destruir a ABP
10 - Sair
2
Digite o elemento que deseja inserir: 7
```

Figura 2: Questão 2.1 - Insere Elemento

```
O que deseja fazer?
1 - Criar ABP
2 - Inserir um elemento
3 - Buscar um elemento
4 - Remover um elemento
5 - Imprimir a ABP em ordem
6 - Imprimir a ABP em pre-ordem
7 - Imprimir a ABP em pos-ordem
8 - Mostrar a quantidade de nos na ABP
9 - Destruir a ABP
10 - Sair
2
Digite o elemento que deseja inserir: 5
```

Figura 3: Questão 2.1 - Insere elemento

```
O que deseja fazer?
1 - Criar ABP
2 - Inserir um elemento
3 - Buscar um elemento
4 - Remover um elemento
5 - Imprimir a ABP em ordem
6 - Imprimir a ABP em pre-ordem
7 - Imprimir a ABP em pos-ordem
8 - Mostrar a quantidade de nos na ABP
9 - Destruir a ABP
10 - Sair
6

Pre Ordem: [7, 0] [5, 1]
```

Figura 4: Questão 2.1 - Imprime árvore em pré-ordem

```
O que deseja fazer?
1 - Criar ABP
2 - Inserir um elemento
3 - Buscar um elemento
4 - Remover um elemento
5 - Imprimir a ABP em ordem
6 - Imprimir a ABP em pre-ordem
7 - Imprimir a ABP em pos-ordem
8 - Mostrar a quantidade de nos na ABP
9 - Destruir a ABP
10 - Sair
7

Pos Ordem: [5, 1] [7, 0]
```

Figura 5: Questão 2.1 - Imprime árvore em pós-ordem

```
O que deseja fazer?
1 - Criar ABP
2 - Inserir um elemento
3 - Buscar um elemento
4 - Remover um elemento
5 - Imprimir a ABP em ordem
6 - Imprimir a ABP em pre-ordem
7 - Imprimir a ABP em pos-ordem
8 - Mostrar a quantidade de nos na ABP
9 - Destruir a ABP
10 - Sair
3
Digite o elemento que deseja consultar: 5
O numero esta na arvore!
```

Figura 6: Questão 2.1 - Busca elemento

```
O que deseja fazer?
1 - Criar ABP
2 - Inserir um elemento
3 - Buscar um elemento
4 - Remover um elemento
5 - Imprimir a ABP em ordem
6 - Imprimir a ABP em pre-ordem
7 - Imprimir a ABP em pos-ordem
8 - Mostrar a quantidade de nos na ABP
9 - Destruir a ABP
10 - Sair
4
Digite o elemento que deseja remover: 5
```

Figura 7: Questão 2.1 - Remove Elemento

```
O que deseja fazer?
1 - Criar ABP
2 - Inserir um elemento
3 - Buscar um elemento
4 - Remover um elemento
5 - Imprimir a ABP em ordem
6 - Imprimir a ABP em pre-ordem
7 - Imprimir a ABP em pos-ordem
8 - Mostrar a quantidade de nos na ABP
9 - Destruir a ABP
10 - Sair
5

Em Ordem: [7, 0]
```

Figura 8: Questão 2.1 - Imprime árvore na ordem

```
O que deseja fazer?
1 - Criar ABP
2 - Inserir um elemento
3 - Buscar um elemento
4 - Remover um elemento
5 - Imprimir a ABP em ordem
6 - Imprimir a ABP em pre-ordem
7 - Imprimir a ABP em pos-ordem
8 - Mostrar a quantidade de nos na ABP
9 - Destruir a ABP
10 - Sair
8
O numero de nos eh: 1!
```

Figura 9: Questão 2.1 - Quantidade de elementos na árvore

```
O que deseja fazer?
1 - Criar ABP
2 - Inserir um elemento
3 - Buscar um elemento
4 - Remover um elemento
5 - Imprimir a ABP em ordem
6 - Imprimir a ABP em pre-ordem
7 - Imprimir a ABP em pos-ordem
8 - Mostrar a quantidade de nos na ABP
9 - Destruir a ABP
10 - Sair
9
Arvore destruida!
```

Figura 10: Questão 2.1 - Destroi árvore

1.2 Modificar o TAD

Código

```c
#ifndef ABP_H_ALUNO_
#define ABP_H_ALUNO_

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

typedef struct aluno{
    char* nome;
    int matricula;
    double nota;
}Aluno;

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

typedef struct NO* ABP;

NO* alocarNO();
void liberarNO(NO* q);
ABP* criaABP();
Aluno* criaAluno();
void destroiRec(NO* no);
void destroiABP(ABP* raiz);
int estaVazia(ABP* raiz);
int insereRec(NO** raiz, Aluno *aluno);
int insereIte(NO** raiz, Aluno *aluno);
int insereElem(ABP* raiz, Aluno *aluno);
int pesquisaRec(NO** raiz, Aluno
    *aluno);
int pesquisaIte(NO** raiz, Aluno
    *aluno);
int pesquisa(ABP* raiz, Aluno *aluno);
int removeRec(NO** raiz, Aluno *aluno);
NO* removeAtual(NO* atual);
int removeIte(NO** raiz, Aluno *aluno);
int removeElem(ABP* raiz, Aluno *aluno);
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 tipo);

void aguardaLimpa();
void contador(NO* raiz, int nivel, int
    *cont);
char* maior_nt(NO* raiz, int nivel, int
    *maior, char* aluno_maior);
char* menor_nt(NO* raiz, int nivel, int
    *menor, char* aluno_menor);

#endif
```

codigos/questao12/questao12.h

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "questao12.h"

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;
}


Aluno* criaAluno(){
    Aluno* aluno = (Aluno*) malloc
        (sizeof(Aluno));
    if(aluno != NULL){
        aluno->nome = (char*)malloc(30
            * sizeof(char));
        aluno->matricula = 0;
        aluno->nota = 0.0;
    }
    return aluno;
}

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, Aluno *aluno){
    if(*raiz == NULL){
        NO* novo = alocarNO();
        if(novo == NULL) return 0;
        novo->aluno = aluno;
        novo->esq = NULL; novo->dir = NULL;
        *raiz = novo;
    }else{
        int cmp = strcmp(aluno->nome, (*raiz)->aluno->nome);
        if(cmp == 0){
            printf("Aluno Existente!\n");
            return 0;
        }
        if(cmp < 0)
            return insereRec(&(*raiz)->esq, aluno);
        else if(cmp > 0)
            return insereRec(&(*raiz)->dir, aluno);
    }
    return 1;
}

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

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

    Aluno* novoAluno = criaAluno();
    if(novoAluno == NULL) return 0;

    strcpy(novoAluno->nome, aluno->nome);
    novoAluno->matricula = aluno->matricula;
    novoAluno->nota = aluno->nota;

    return insereRec(raiz, novoAluno);
}


int pesquisaRec(NO** raiz, Aluno *aluno){
    if(*raiz == NULL) return 0;
    int cmp = strcmp(aluno->nome, (*raiz)->aluno->nome);
    if(cmp == 0){
        printf("O nome do aluno eh: %s\nA matricula do aluno eh: %d"
               "\nA nota do aluno eh: %lf\n",
               (*raiz)->aluno->nome, (*raiz)->aluno->matricula, (*raiz)->aluno->nota);
        return 1;
    }
    if(cmp < 0)
        return pesquisaRec(&(*raiz)->esq, aluno);
    else
        return pesquisaRec(&(*raiz)->dir, aluno);
}


int pesquisaIte(NO** raiz, Aluno *aluno){
    NO* aux = *raiz;
    while(aux != NULL){
        if(aux->aluno->nome == aluno->nome) return 1;
        if(aluno->nome < aux->aluno->nome)
```

```
            aux = aux->esq;
        else
            aux = aux->dir;
    }
    return 0;
}

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

int removeRec(NO** raiz, Aluno *aluno){
    if(*raiz == NULL) return 0;
    if(strcmp(aluno->nome, (*raiz)->aluno->nome) == 0){
        NO* aux;
        if((*raiz)->esq == NULL && (*raiz)->dir == NULL){
            //Caso 1 - NO sem filhos
            printf("Caso 1: Liberando %s..\n", (*raiz)->aluno->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)->aluno->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)->aluno->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)->aluno->nome);
            //Estrategia 3.1:
            NO* Filho = (*raiz)->esq;
            while(Filho->dir != NULL)//Localiza o MAIOR valor da subarvore esquerda
                Filho = Filho->dir;
            (*raiz)->aluno = Filho->aluno;
            Filho->aluno = aluno;
            return removeRec(&(*raiz)->esq, aluno);
        }
        return 1;
    }else if(aluno->nome < (*raiz)->aluno->nome)
        return removeRec(&(*raiz)->esq, aluno);
    else
        return removeRec(&(*raiz)->dir, aluno);
}

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, Aluno *aluno){
    if(*raiz == NULL) return 0;
    NO* atual = *raiz, *ant = NULL;
    while(atual != NULL){
```

```
222         if(strcmp(aluno->nome,
                atual->aluno->nome) == 0){
223             if(atual == *raiz)
224                 *raiz =
                        removeAtual(atual);
225             else{
226                 if(ant->dir == atual)
227                     ant->dir =
                            removeAtual(atual);
228                 else
229                     ant->esq =
                            removeAtual(atual);
230             }
231             return 1;
232         }
233         ant = atual;
234         if(strcmp(aluno->nome,
                atual->aluno->nome) < 0)
235             atual = atual->esq;
236         else
237             atual = atual->dir;
238     }
239     return 0;
240 }
241
242
243 int removeElem(ABP* raiz, Aluno *aluno){
244     if(pesquisa(raiz, aluno) == 0){
245         printf("Aluno inexistente!\n");
246         return 0;
247     }
248     //return removeRec(raiz, aluno);
249     return removeIte(raiz, aluno);
250 }
251
252 void em_ordem(NO* raiz, int nivel){
253     if(raiz != NULL){
254         em_ordem(raiz->esq, nivel+1);
255         printf("[(%s,%d,%lf), %d] ",
                raiz->aluno->nome,
256             raiz->aluno->matricula,
                    raiz->aluno->nota,nivel);
257         em_ordem(raiz->dir, nivel+1);
258     }
259 }
260
261 void pre_ordem(NO* raiz, int nivel){
262     if(raiz != NULL){
263         printf("[(%s,%d,%lf), %d] ",
                raiz->aluno->nome,
264             raiz->aluno->matricula,
                    raiz->aluno->nota,nivel);
265         pre_ordem(raiz->esq, nivel+1);
266         pre_ordem(raiz->dir, nivel+1);
267     }
268 }
269
270 void pos_ordem(NO* raiz, int nivel){
271     if(raiz != NULL){
272         pos_ordem(raiz->esq, nivel+1);
273         pos_ordem(raiz->dir, nivel+1);
274         printf("[(%s,%d,%lf), %d] ",
                raiz->aluno->nome,
275             raiz->aluno->matricula,
                    raiz->aluno->nota,nivel);
276     }
277 }
278
279 void imprime(ABP* raiz, int tipo){
280     if(raiz == NULL) return;
281     if(estaVazia(raiz)){
282         printf("Arvore Vazia!\n");
283         return;
284     }
285     switch (tipo){
286     case 0:
287         printf("\nEm Ordem: ");
                em_ordem(*raiz, 0);
288         break;
289
290     case 1:
291         printf("\nPre Ordem: ");
                pre_ordem(*raiz, 0);
292         break;
293
294     case 2:
295         printf("\nPos Ordem: ");
                pos_ordem(*raiz, 0);
296         break;
297     default:
298         break;
299     }
300     printf("\n");
301 }
302
303 /* Funcao para esperar resposta do
        usuario e depois limpar*/
304 void aguardaLimpa(){
305     getchar();
306     printf("\n\nAperte qualquer tecla
            para continuar\n");
307     getchar();
308     system("clear");
309 }
310
311
312 char* maior_nt(NO* raiz, int nivel, int
        *maior, char* aluno_maior){
313     if(raiz != NULL){
314         maior_nt(raiz->esq, nivel+1,
                maior, aluno_maior);
315         if(*maior < raiz->aluno->nota){
316             *maior = raiz->aluno->nota;
317             strcpy(aluno_maior,
                    raiz->aluno->nome);
318         }
319         maior_nt(raiz->dir, nivel+1,
                maior, aluno_maior);
```

```
320     }
321     return aluno_maior;
322 }
323
324
325 char* menor_nt(NO* raiz, int nivel, int
        *menor, char* aluno_menor){
326     if(raiz != NULL){
327         menor_nt(raiz->esq, nivel+1,
                menor, aluno_menor);
328         if(*menor > raiz->aluno->nota){
329             *menor = raiz->aluno->nota;
330             strcpy(aluno_menor,
                    raiz->aluno->nome);
331         }
332         menor_nt(raiz->dir, nivel+1,
                menor, aluno_menor);
333     }
334     return aluno_menor;
335 }
```

codigos/questao12/questao12.c

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4 #include "questao12.h"
 5
 6 int main(){
 7     int *menor =
           (int*)malloc(sizeof(int));
 8     int *maior =
           (int*)malloc(sizeof(int));
 9     double nota;
10     int escolha, matricula, busca;
11     char* nome = (char*)malloc(30 *
           sizeof(char));
12     ABP *abt;
13     Aluno *aluno = criaAluno();
14
15     do{
16         printf("O que deseja fazer?\n1
               - Criar ABP\n2 - Inserir um
               Aluno\n"
17         "3 - Buscar um Aluno pelo nome
               e imprimir suas
               informacoes\n"
18         "4 - Remover um Aluno pelo
               nome\n5 - Imprimir a ABP em
               ordem"
19         "\n6 - Imprimir as informacoes
               do aluno com a maior nota"
20         "\n7 - Imprimir as informacoes
               do aluno com a menor nota\n"
21         "8 - Destruir a ABP\n9 -
               Sair\n");
22         scanf("%d",&escolha);
23
24         switch (escolha){
25         case 9:
26             break;
27
28         case 1:
29             abt = criaABP();
30             printf("Arvore criada!");
31             aguardaLimpa();
32             break;
33
34         case 2:
35             printf("Digite o elemento
                   que deseja inserir: ");
36             scanf("%s %d %lf", nome,
                   &matricula, &nota);
37             strcpy(aluno->nome, nome);
38             aluno->matricula =
                   matricula;
39             aluno->nota = nota;
40             insereElem(abt,aluno);
41             aguardaLimpa();
42             break;
43
44         case 3:
45             printf("Digite o nome do
                   aluno que deseja
                   consultar: ");
46             scanf("%s",nome);
47             strcpy(aluno->nome, nome);
48             busca = pesquisa(abt,aluno);
49             if(busca == 0)
50                 printf("O aluno nao
                       esta na arvore!");
51             aguardaLimpa();
52             break;
53
54         case 4:
55             printf("Digite o nome do
                   aluno que deseja
                   remover: ");
56             scanf("%s",nome);
57             strcpy(aluno->nome, nome);
58             removeElem(abt,aluno);
59             aguardaLimpa();
60             break;
61
62         case 5:
63             imprime(abt,0);
64             aguardaLimpa();
65             break;
66
67         case 6:
68             *maior = 0;
```

```
            strcpy(aluno->nome,
                maior_nt(*abt, 0, maior,
                nome));
            busca = pesquisa(abt,aluno);
            aguardaLimpa();
            break;

        case 7:
            *menor = 100000.0;
            strcpy(aluno->nome,
                menor_nt(*abt, 0, menor,
                nome));
            busca = pesquisa(abt,aluno);
            aguardaLimpa();
            break;

        case 8:
            destroiABP(abt);
            printf("Arvore destruida!");
            aguardaLimpa();
            break;

        default:
            printf("\nAlgo de errado
                nao esta certo!\n\n");
            break;
        }
    }while(escolha!=9);


    return 0;
}
```

codigos/questao12/main.c

```
all: questao12.o
  gcc questao12.o main.c -o main

questao12.o: questao12.h questao12.c
  gcc -c questao12.c

clean:
  rm -f questao12.o main
```

codigos/questao12/Makefile

Saída

```
O que deseja fazer?
1 - Criar ABP
2 - Inserir um Aluno
3 - Buscar um Aluno pelo nome e imprimir suas informacoes
4 - Remover um Aluno pelo nome
5 - Imprimir a ABP em ordem
6 - Imprimir as informacoes do aluno com a maior nota
7 - Imprimir as informacoes do aluno com a menor nota
8 - Destruir a ABP
9 - Sair
1
Arvore criada!
```

Figura 11: Questão 2.1 - Cria árvore

```
O que deseja fazer?
1 - Criar ABP
2 - Inserir um Aluno
3 - Buscar um Aluno pelo nome e imprimir suas informacoes
4 - Remover um Aluno pelo nome
5 - Imprimir a ABP em ordem
6 - Imprimir as informacoes do aluno com a maior nota
7 - Imprimir as informacoes do aluno com a menor nota
8 - Destruir a ABP
9 - Sair
2
Digite o elemento que deseja inserir: Adelson
5
10
```

Figura 12: Questão 2.1 - Insere Elemento

```
O que deseja fazer?
1 - Criar ABP
2 - Inserir um Aluno
3 - Buscar um Aluno pelo nome e imprimir suas informacoes
4 - Remover um Aluno pelo nome
5 - Imprimir a ABP em ordem
6 - Imprimir as informacoes do aluno com a maior nota
7 - Imprimir as informacoes do aluno com a menor nota
8 - Destruir a ABP
9 - Sair
2
Digite o elemento que deseja inserir: Fulano
4
5
```

Figura 13: Questão 2.1 - Insere elemento

```
O que deseja fazer?
1 - Criar ABP
2 - Inserir um Aluno
3 - Buscar um Aluno pelo nome e imprimir suas informacoes
4 - Remover um Aluno pelo nome
5 - Imprimir a ABP em ordem
6 - Imprimir as informacoes do aluno com a maior nota
7 - Imprimir as informacoes do aluno com a menor nota
8 - Destruir a ABP
9 - Sair
5

Em Ordem: [(Adelson,5,10.000000), 0] [(Fulano,4,5.000000), 1]
```

Figura 14: Questão 2.1 - Imprime árvore em ordem

```
O que deseja fazer?
1 - Criar ABP
2 - Inserir um Aluno
3 - Buscar um Aluno pelo nome e imprimir suas informacoes
4 - Remover um Aluno pelo nome
5 - Imprimir a ABP em ordem
6 - Imprimir as informacoes do aluno com a maior nota
7 - Imprimir as informacoes do aluno com a menor nota
8 - Destruir a ABP
9 - Sair
6
O nome do aluno eh: Adelson
A matricula do aluno eh: 5
A nota do aluno eh: 10.000000
```

Figura 15: Questão 2.1 - Imprime o maior Aluno

```
O que deseja fazer?
1 - Criar ABP
2 - Inserir um Aluno
3 - Buscar um Aluno pelo nome e imprimir suas informacoes
4 - Remover um Aluno pelo nome
5 - Imprimir a ABP em ordem
6 - Imprimir as informacoes do aluno com a maior nota
7 - Imprimir as informacoes do aluno com a menor nota
8 - Destruir a ABP
9 - Sair
7
O nome do aluno eh: Fulano
A matricula do aluno eh: 4
A nota do aluno eh: 5.000000
```

Figura 16: Questão 2.1 - Imprime o menor Aluno

```
O que deseja fazer?
1 - Criar ABP
2 - Inserir um Aluno
3 - Buscar um Aluno pelo nome e imprimir suas informacoes
4 - Remover um Aluno pelo nome
5 - Imprimir a ABP em ordem
6 - Imprimir as informacoes do aluno com a maior nota
7 - Imprimir as informacoes do aluno com a menor nota
8 - Destruir a ABP
9 - Sair
4
Digite o nome do aluno que deseja remover: Fulano
O nome do aluno eh: Fulano
A matricula do aluno eh: 4
A nota do aluno eh: 5.000000
```

Figura 17: Questão 2.1 - Remove Aluno

```
O que deseja fazer?
1 - Criar ABP
2 - Inserir um Aluno
3 - Buscar um Aluno pelo nome e imprimir suas informacoes
4 - Remover um Aluno pelo nome
5 - Imprimir a ABP em ordem
6 - Imprimir as informacoes do aluno com a maior nota
7 - Imprimir as informacoes do aluno com a menor nota
8 - Destruir a ABP
9 - Sair
3
Digite o nome do aluno que deseja consultar: Fulano
O aluno nao esta na arvore!
```

Figura 18: Questão 2.1 - Busca Aluno

```
O que deseja fazer?
1 - Criar ABP
2 - Inserir um Aluno
3 - Buscar um Aluno pelo nome e imprimir suas informacoes
4 - Remover um Aluno pelo nome
5 - Imprimir a ABP em ordem
6 - Imprimir as informacoes do aluno com a maior nota
7 - Imprimir as informacoes do aluno com a menor nota
8 - Destruir a ABP
9 - Sair
8
Arvore destruida!
```

Figura 19: Questão 2.1 - Destroi árvore