



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 Reimplantação

Código

```
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 void liberarNO(NO* q);
17 ABP* criaABP();
18 void destroiRec(NO* no);
19 void destroiABP(ABP* raiz);
20 int estaVazia(ABP* raiz);
21 int insereRec(NO** raiz, int elem);
22 int insereIte(NO** raiz, int elem);
23
24 int insereElem(ABP* raiz, int elem);
25 int pesquisaRec(NO** raiz, int elem);
26 int pesquisaIte(NO** raiz, int elem);
27 int pesquisa(ABP* raiz, int elem);
28 int removeRec(NO** raiz, int elem);
29 NO* removeAtual(NO* atual);
30 int removeIte(NO** raiz, int elem);
31 int removeElem(ABP* raiz, int elem);
32 void em_ordem(NO* raiz, int nivel);
33 void pre_ordem(NO* raiz, int nivel);
34 void pos_ordem(NO* raiz, int nivel);
35 void imprime(ABP* raiz, int tipo);
36
37 void aguardaLimpa();
38 void contador(NO* raiz, int nivel, int
39             *cont);
40 #endif
41
42
43 #include <stdio.h>
44 #include <stdlib.h>
45 #include "questao11.h"
46
47 NO* alocarNO(){
48     return (NO*) malloc (sizeof(NO));
49 }
50
51 void liberarNO(NO* q){
52     free(q);
53 }
54
55 ABP* criaABP(){
56     ABP* raiz = (ABP*) malloc
57         (sizeof(ABP));
58     if(raiz != NULL)
59         *raiz = NULL;
60     return raiz;
61 }
62
63 void destroiRec(NO* no){
64     if(no == NULL) return;
```

codigos/questao11/questao11.h

```

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
51                 Existente!\n");
52             return 0;
53         }
54         if(elem < (*raiz)->info)
55             return
56                 insereRec(&(*raiz)->esq,
57                     elem);
58         else if(elem > (*raiz)->info)
59             return
60                 insereRec(&(*raiz)->dir,
61                     elem);
62     }
63     return 1;
64 }
65
66 int insereIte(NO** raiz, int elem){
67     NO *aux = *raiz, *ant = NULL;
68     while (aux != NULL){
69         ant = aux;
70         if(aux->info == elem){
71             printf("Elemento
72                 Existente!\n");
73             return 0;
74         }
75         if(elem < aux->info) aux =
76             aux->esq;
77         else aux = aux->dir;
78     }
79     NO* novo = alocarNO();
80
81     if(novo == NULL) return 0;
82     novo->info = elem;
83     novo->esq = NULL; novo->dir = NULL;
84     if(ant == NULL){
85         *raiz = novo;
86     }else{
87         if(elem < ant->info) ant->esq =
88             novo;
89         else ant->dir = novo;
90     }
91     return 1;
92 }
93
94 int insereElem(ABP* raiz, int elem){
95     if(raiz == NULL) return 0;
96     return insereRec(raiz, elem);
97     //return insereIte(raiz, elem);
98 }
99
100 int pesquisaRec(NO** raiz, int elem){
101     if(*raiz == NULL) return 0;
102     if((*raiz)->info == elem) return 1;
103     if(elem < (*raiz)->info)
104         return
105             pesquisaRec(&(*raiz)->esq,
106                 elem);
107     else
108         return
109             pesquisaRec(&(*raiz)->dir,
110                 elem);
111 }
112
113 int pesquisaIte(NO** raiz, int elem){
114     NO* aux = *raiz;
115     while(aux != NULL){
116         if(aux->info == elem) return 1;
117         if(elem < aux->info)
118             aux = aux->esq;
119         else
120             aux = aux->dir;
121     }
122     return 0;
123 }
124
125 int pesquisa(ABP* raiz, int elem){
126     if(raiz == NULL) return 0;
127     if(estaVazia(raiz)) return 0;
128     return pesquisaRec(raiz, elem);
129     //return pesquisaIte(raiz, elem);
130 }
131
132 int removeRec(NO** raiz, int elem){
133     if(*raiz == NULL) return 0;
134     if((*raiz)->info == elem){
135         NO* aux;
136         if((*raiz)->esq == NULL &&
137             (*raiz)->dir == NULL){
138             //Caso 1 - NO sem filhos
139             printf("Caso 1: Liberando

```

```

126         %d..\n", (*raiz)->info);
127         liberarNO(*raiz);
128         *raiz = NULL;
129     }else if((*raiz)->esq == NULL){
130         //Caso 2.1 - Possui apenas
131             uma subarvore direita
132         printf("Caso 2.1: Liberando
133             %d..\n", (*raiz)->info);
134         aux = *raiz;
135         *raiz = (*raiz)->dir;
136         liberarNO(aux);
137     }else if((*raiz)->dir == NULL){
138         //Caso 2.2 - Possui apenas
139             uma subarvore esquerda
140         printf("Caso 2.2: Liberando
141             %d..\n", (*raiz)->info);
142         aux = *raiz;
143         *raiz = (*raiz)->esq;
144         liberarNO(aux);
145     }else{
146         //Caso 3 - Possui as duas
147             subarvores (esq e dir)
148         //Duas estrategias:
149         //3.1 - Substituir pelo NO
150             com o MAIOR valor da
151             subarvore esquerda
152         //3.2 - Substituir pelo NO
153             com o MENOR valor da
154             subarvore direita
155         printf("Caso 3: Liberando
156             %d..\n", (*raiz)->info);
157         //Estrategia 3.1:
158         NO* Filho = (*raiz)->esq;
159         while(Filho->dir !=
160             NULL)//Localiza o MAIOR
161             valor da subarvore
162             esquerda
163         Filho = Filho->dir;
164         (*raiz)->info = Filho->info;
165         Filho->info = elem;
166         return
167             removeRec(&(*raiz)->esq,
168                 elem);
169     }
170     return 1;
171 }else if(elem < (*raiz)->info)
172     return removeRec(&(*raiz)->esq,
173         elem);
174 else
175     return removeRec(&(*raiz)->dir,
176         elem);
177 }
178
179 NO* removeAtual(NO* atual){
180     NO* no1, *no2;
181     //Ambos casos no if(atual->esq ==
182         NULL)
183     //Caso 1 - NO sem filhos
184     //Caso 2.1 - Possui apenas uma
185
186         subarvore direita
187     if(atual->esq == NULL){
188         no2 = atual->dir;
189         liberarNO(atual);
190         return no2;
191     }
192     //Caso 3 - Possui as duas
193         subarvores (esq e dir)
194     //Estrategia:
195     no1 = atual;
196     no2 = atual->esq;
197     while(no2->dir != NULL){
198         no1 = no2;
199         no2 = no2->dir;
200     }
201     if(no1 != atual){
202         no1->dir = no2->esq;
203         no2->esq = atual->esq;
204     }
205     no2->dir = atual->dir;
206     liberarNO(atual);
207     return no2;
208 }
209
210 int removeIte(NO** raiz, int elem){
211     if(*raiz == NULL) return 0;
212     NO* atual = *raiz, *ant = NULL;
213     while(atual != NULL){
214         if(elem == atual->info){
215             if(atual == *raiz)
216                 *raiz =
217                     removeAtual(atual);
218             else{
219                 if(ant->dir == atual)
220                     ant->dir =
221                         removeAtual(atual);
222                 else
223                     ant->esq =
224                         removeAtual(atual);
225             }
226             return 1;
227         }
228         ant = atual;
229         if(elem < atual->info)
230             atual = atual->esq;
231         else
232             atual = atual->dir;
233     }
234     return 0;
235 }
236
237 int removeElem(ABP* raiz, int elem){
238     if(pesquisa(raiz, elem) == 0){
239         printf("Elemento
240             inexistente!\n");
241         return 0;
242     }
243     //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] ",
226             raiz->info, nivel);
227         em_ordem(raiz->dir, nivel+1);
228     }
229 }
230 void pre_ordem(NO* raiz, int nivel){
231     if(raiz != NULL){
232         printf("[%d, %d] ",
233             raiz->info, nivel);
234         pre_ordem(raiz->esq, nivel+1);
235         pre_ordem(raiz->dir, nivel+1);
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] ",
243             raiz->info, nivel);
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: ");
255         em_ordem(*raiz, 0);
256         break;
257     case 1:
258         printf("\nPre Ordem: ");
259         pre_ordem(*raiz, 0);
260         break;
261     case 2:
262         printf("\nPos Ordem: ");
263         pos_ordem(*raiz, 0);
264         break;
265     default:
266         break;
267     }
268     printf("\n");
269 }
270 /* Funcao para esperar resposta do
271    usuario e depois limpar*/
272 void aguardaLimpa(){
273     getchar();
274     printf("\n\nAperte qualquer tecla
275     para continuar\n");
276     getchar();
277     system("clear");
278 }
279 void contador(NO* raiz, int nivel, int
280     *cont){
281     if(raiz != NULL){
282         contador(raiz->esq, nivel+1,
283             cont);
284         (*cont)++;
285         contador(raiz->dir, nivel+1,
286             cont);
287     }
288 }

```

codigos/questao11/questao11.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "questao11.h"
4
5  int main(){
6      int *cont =
7          (int*)malloc(sizeof(int));
8      int escolha, elem, busca;
9      ABP *abt;
10     do{
11         printf("0 que deseja fazer?\n1
12             - Criar ABP\n2 - Inserir um
13             elemento\n3 - Buscar um elemento\n4 -
14             Remover um elemento\n5 -
15             Imprimir a ABP em ordem"
16             "\n6 - Imprimir a ABP em
17             pre-ordem\n7 - Imprimir a
18             ABP em pos-ordem\n8 -
19             Mostrar a quantidade de
20             nos na ABP\n9 - Destruir a
21             ABP\n10 - Sair\n");
22         scanf("%d",&escolha);
23         switch (escolha){
24         case 10:
25             break;
26         case 1:
27             abt = criaABP();
28             printf("Arvore criada!");
29             aguardaLimpa();

```

```

24         break;
25
26     case 2:
27         printf("Digite o elemento
28             que deseja inserir: ");
29         scanf("%d",&elem);
30         insereElem(abt,elem);
31         aguardaLimpa();
32         break;
33
34     case 3:
35         printf("Digite o elemento
36             que deseja consultar: ");
37         scanf("%d",&elem);
38         busca = pesquisa(abt,elem);
39         if(busca == 1)
40             printf("0 numero esta
41                 na arvore!");
42         else
43             printf("0 numero nao
44                 esta na arvore!");
45         aguardaLimpa();
46         break;
47
48     case 4:
49         printf("Digite o elemento
50             que deseja remover: ");
51         scanf("%d",&elem);
52         removeElem(abt,elem);
53         aguardaLimpa();
54         break;
55
56     case 5:
57         imprime(abt,0);
58         aguardaLimpa();
59         break;
60
61     case 6:
62         imprime(abt,1);
63         aguardaLimpa();
64         break;
65
66     case 7:
67         imprime(abt,2);
68         aguardaLimpa();
69         break;
70
71     case 8:
72         *cont = 0;
73         contador(*abt,0,cont);
74         printf("0 numero de nos eh:
75             %d!", *cont);
76         aguardaLimpa();
77         break;
78
79     case 9:
80         destroiABP(abt);
81         printf("Arvore destruida!");
82         aguardaLimpa();
83         break;
84
85     default:
86         printf("\nAlgo de errado
87             nao esta certo!\n\n");
88         break;
89     }
90 }while(escolha!=10);
91
92 return 0;
93 }

```

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

```

0 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

```

0 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

```

0 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

```

0 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

```

0 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

```

0 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
0 numero esta na arvore!

```

Figura 6: Questão 2.1 - Busca elemento

```

0 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

```

0 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

```

0 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
0 numero de nos eh: 1!

```

Figura 9: Questão 2.1 - Quantidade de elementos na árvore

```

0 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

```
1 #ifndef ABP_H_ALUNO_
2 #define ABP_H_ALUNO_
3
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 typedef struct aluno{
8     char* nome;
9     int matricula;
10    double nota;
11 }Aluno;
12
13 typedef struct NO{
14     Aluno *aluno;
15     struct NO* esq;
16     struct NO* dir;
17 }NO;
18
19 typedef struct NO* ABP;
20
21 NO* alocarNO();
22 void liberarNO(NO* q);
23 ABP* criaABP();
24 Aluno* criaAluno();
25 void destroiRec(NO* no);
26 void destroiABP(ABP* raiz);
27 int estaVazia(ABP* raiz);
28 int insereRec(NO** raiz, Aluno *aluno);
29
30 int insereIte(NO** raiz, Aluno *aluno);
31 int insereElem(ABP* raiz, Aluno *aluno);
32 int pesquisaRec(NO** raiz, Aluno
    *aluno);
33 int pesquisaIte(NO** raiz, Aluno
    *aluno);
34 int pesquisa(ABP* raiz, Aluno *aluno);
35 int removeRec(NO** raiz, Aluno *aluno);
36 NO* removeAtual(NO* atual);
37 int removeIte(NO** raiz, Aluno *aluno);
38 int removeElem(ABP* raiz, Aluno *aluno);
39 void em_ordem(NO* raiz, int nivel);
40 void pre_ordem(NO* raiz, int nivel);
41 void pos_ordem(NO* raiz, int nivel);
42 void imprime(ABP* raiz, int tipo);
43
44 void aguardaLimpa();
45 void contador(NO* raiz, int nivel, int
    *cont);
46 char* maior_nt(NO* raiz, int nivel, int
    *maior, char* aluno_maior);
47 char* menor_nt(NO* raiz, int nivel, int
    *menor, char* aluno_menor);
48
49 #endif
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

codigos/questao12/questao12.h


```

44     }
45 }
46
47 int estaVazia(ABP* raiz){
48     if(raiz == NULL) return 0;
49     return (*raiz == NULL);
50 }
51
52
53 int insereRec(NO** raiz, Aluno *aluno){
54     if(*raiz == NULL){
55         NO* novo = alocarNO();
56         if(novo == NULL) return 0;
57         novo->aluno = aluno;
58         novo->esq = NULL; novo->dir =
            NULL;
59         *raiz = novo;
60     }else{
61         int cmp = strcmp(aluno->nome,
62             (*raiz)->aluno->nome);
63         if(cmp == 0){
64             printf("Aluno
65                 Existente!\n");
66             return 0;
67         }
68         if(cmp < 0)
69             return
70                 insereRec(&(*raiz)->esq,
71                     aluno);
72         else if(cmp > 0)
73             return
74                 insereRec(&(*raiz)->dir,
75                     aluno);
76     }
77     return 1;
78 }
79
80 int insereIte(NO** raiz, Aluno *aluno){
81     NO *aux = *raiz, *ant = NULL;
82     while (aux != NULL){
83         ant = aux;
84         if(aux->aluno->nome ==
85             aluno->nome){
86             printf("Aluno
87                 Existente!\n");
88             return 0;
89         }
90         if(aluno->nome <
91             aux->aluno->nome) aux =
92                 aux->esq;
93         else aux = aux->dir;
94     }
95     NO* novo = alocarNO();
96     if(novo == NULL) return 0;
97     novo->aluno = aluno;
98     novo->esq = NULL; novo->dir = NULL;
99     *raiz = novo;
100 }else{
101     int cmp = strcmp(aluno->nome,
102         (*raiz)->aluno->nome);
103     if(cmp == 0){
104         printf("Aluno
105             Existente!\n");
106         return 0;
107     }
108     if(cmp < 0)
109         return
110             insereRec(&(*raiz)->esq,
111                 aluno);
112     else if(cmp > 0)
113         return
114             insereRec(&(*raiz)->dir,
115                 aluno);
116 }
117 return 1;
118 }
119
120 int pesquisaRec(NO** raiz, Aluno
121     *aluno){
122     if(*raiz == NULL) return 0;
123     int cmp = strcmp(aluno->nome,
124         (*raiz)->aluno->nome);
125     if(cmp == 0){
126         printf("O nome do aluno eh:
127             %s\nA matricula do aluno eh:
128             %d"
129                 "\nA nota do aluno eh:
130                 %lf\n",
131             (*raiz)->aluno->nome,
132             (*raiz)->aluno->matricula,
133             (*raiz)->aluno->nota);
134         return 1;
135     }
136     if(cmp < 0)
137         return
138             pesquisaRec(&(*raiz)->esq,
139                 aluno);
140     else
141         return
142             pesquisaRec(&(*raiz)->dir,
143                 aluno);
144 }
145
146 int pesquisaIte(NO** raiz, Aluno
147     *aluno){
148     NO* aux = *raiz;
149     while(aux != NULL){
150         if(aux->aluno->nome ==
151             aluno->nome) return 1;
152         if(aluno->nome <
153             aux->aluno->nome)
154             aux = aux->esq;
155         else aux = aux->dir;
156     }
157     return 0;
158 }

```

```

133         aux = aux->esq;           175
134     else                           176
135         aux = aux->dir;           177
136 }
137 return 0;
138 }
139                                     178
140 int pesquisa(ABP* raiz, Aluno *aluno){ 179
141     if(raiz == NULL) return 0;
142     if(estaVazia(raiz)) return 0;      180
143     return pesquisaRec(raiz, aluno);   181
144     //return pesquisaIte(raiz, aluno);
145 }
146                                     182
147                                     183
148 int removeRec(NO** raiz, Aluno *aluno){ 184
149     if(*raiz == NULL) return 0;
150     if(strcmp(aluno->nome,           185
151         (*raiz)->aluno->nome) == 0){
152         NO* aux;                     186
153         if((*raiz)->esq == NULL &&   187
154             (*raiz)->dir == NULL){
155             //Caso 1 - NO sem filhos  188 }
156             printf("Caso 1: Liberando  189
157                 %s..\n",           190 NO* removeAtual(NO* atual){
158                 (*raiz)->aluno->nome); 191 NO* no1, *no2;
159             liberarNO(*raiz);       192 //Ambos casos no if(atual->esq ==
160             *raiz = NULL;           NULL)
161         }else if((*raiz)->esq == NULL){ 193 //Caso 1 - NO sem filhos
162             //Caso 2.1 - Possui apenas 194 //Caso 2.1 - Possui apenas uma
163             uma subarvore direita    subarvore direita
164             printf("Caso 2.1: Liberando 195
165                 %s..\n",           196 if(atual->esq == NULL){
166                 (*raiz)->aluno->nome); 197 no2 = atual->dir;
167             aux = *raiz;           198 liberarNO(atual);
168             *raiz = (*raiz)->dir;   199 return no2;
169             liberarNO(aux);         200 }
170         }else if((*raiz)->dir == NULL){ 201 //Caso 3 - Possui as duas
171             //Caso 2.2 - Possui apenas 202 subarvores (esq e dir)
172             uma subarvore esquerda   //Estrategia:
173             printf("Caso 2.2: Liberando 203
174                 %s..\n",           204 no1 = atual;
175                 (*raiz)->aluno->nome); 205 no2 = atual->esq;
176             aux = *raiz;           206 while(no2->dir != NULL){
177             *raiz = (*raiz)->esq;   207 no1 = no2;
178             liberarNO(aux);         208 no2 = no2->dir;
179         }else{                     209 }
180             //Caso 3 - Possui as duas 210 if(no1 != atual){
181             subarvores (esq e dir)   211 no1->dir = no2->esq;
182             //Duas estrategias:      212 no2->esq = atual->esq;
183             //3.1 - Substituir pelo NO 213 }
184             com o MAIOR valor da    214 no2->dir = atual->dir;
185             subarvore esquerda      215 liberarNO(atual);
186             //3.2 - Substituir pelo NO 216 }
187             com o MENOR valor da    217 return no2;
188             subarvore direita
189             printf("Caso 3: Liberando  218 int removeIte(NO** raiz, Aluno *aluno){
190                 %s..\n",           219 if(*raiz == NULL) return 0;
191                 (*raiz)->aluno->nome); 220 NO* atual = *raiz, *ant = NULL;
192                                     221 while(atual != NULL){

```

```

222         if(strcmp(aluno->nome,      272         pos_ordem(raiz->esq, nivel+1);
           atual->aluno->nome) == 0){ 273         pos_ordem(raiz->dir, nivel+1);
223             if(atual == *raiz)      274         printf("[(%s,%d,%lf), %d] ",
224                 *raiz =              raiz->aluno->nome,
           removeAtual(atual); 275         raiz->aluno->matricula,
225         else{                        raiz->aluno->nota,nivel);
226             if(ant->dir == atual) 276     }
227             ant->dir =              277 }
           removeAtual(atual); 278
228         else                          279 void imprime(ABP* raiz, int tipo){
229             ant->esq =              280     if(raiz == NULL) return;
           removeAtual(atual); 281     if(estaVazia(raiz)){
230     }                                282         printf("Arvore Vazia!\n");
231         return 1;                  283         return;
232     }                                284     }
233     ant = atual;                    285     switch (tipo){
234     if(strcmp(aluno->nome,          286     case 0:
           atual->aluno->nome) < 0) 287         printf("\nEm Ordem: ");
           atual = atual->esq;        288         em_ordem(*raiz, 0);
235     else                            289         break;
236         atual = atual->dir;          290     case 1:
237     }                                291         printf("\nPre Ordem: ");
238     return 0;                      292         pre_ordem(*raiz, 0);
239 }                                  293         break;
240 }                                  294     case 2:
241                                     295         printf("\nPos Ordem: ");
242                                     296         pos_ordem(*raiz, 0);
243 int removeElem(ABP* raiz, Aluno *aluno) 297         break;
244 if(pesquisa(raiz, aluno) == 0){ 298     default:
245     printf("Aluno inexistente!\n"); 299         break;
246     return 0;                      300     }
247 }                                  301     printf("\n");
248 //return removeRec(raiz, aluno); 302 }
249 return removeIte(raiz, aluno);    303 /* Funcao para esperar resposta do
250 }                                  304     usuario e depois limpar*/
251                                     305 void aguardaLimpa(){
252 void em_ordem(NO* raiz, int nivel){ 306     getchar();
253     if(raiz != NULL){              307     printf("\n\nAperte qualquer tecla
254         em_ordem(raiz->esq, nivel+1); 308         para continuar\n");
255         printf("[(%s,%d,%lf), %d] ", 309     getchar();
           raiz->aluno->nome,        310     system("clear");
256         raiz->aluno->matricula,      311
           raiz->aluno->nota,nivel); 312
257         em_ordem(raiz->dir, nivel+1); 313
258     }                                314 char* maior_nt(NO* raiz, int nivel, int
259 }                                  315     *maior, char* aluno_maior){
260                                     316     if(raiz != NULL){
261 void pre_ordem(NO* raiz, int nivel){ 317         maior_nt(raiz->esq, nivel+1,
262     if(raiz != NULL){              318         maior, aluno_maior);
263         printf("[(%s,%d,%lf), %d] ", 319         if(*maior < raiz->aluno->nota){
           raiz->aluno->nome,        320             *maior = raiz->aluno->nota;
264         raiz->aluno->matricula,      321             strcpy(aluno_maior,
           raiz->aluno->nota,nivel); 322                 raiz->aluno->nome);
265         pre_ordem(raiz->esq, nivel+1); 323         }
266         pre_ordem(raiz->dir, nivel+1); 324     }
267     }                                325     maior_nt(raiz->dir, nivel+1,
268 }                                  326     maior, aluno_maior);
269                                     327
270 void pos_ordem(NO* raiz, int nivel){ 328
271     if(raiz != NULL){              329

```

```

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 =
9      (int*)malloc(sizeof(int));
10     int *maior =
11     (int*)malloc(sizeof(int));
12     double nota;
13     int escolha, matricula, busca;
14     char* nome = (char*)malloc(30 *
15     sizeof(char));
16     ABP *abt;
17     Aluno *aluno = criaAluno();
18
19     do{
20         printf("0 que deseja fazer?\n1
21         - Criar ABP\n2 - Inserir um
22         Aluno\n"
23         "3 - Buscar um Aluno pelo nome
24         e imprimir suas
25         informacoes\n"
26         "4 - Remover um Aluno pelo
27         nome\n5 - Imprimir a ABP em
28         ordem"
29         "\n6 - Imprimir as informacoes
30         do aluno com a maior nota"
31         "\n7 - Imprimir as informacoes
32         do aluno com a menor nota\n"
33         "8 - Destruir a ABP\n9 -
34         Sair\n");
35         scanf("%d",&escolha);
36
37         switch (escolha){
38             case 9:
39                 break;
40
41             case 1:
42                 abt = criaABP();
43                 printf("Arvore criada!");
44                 aguardaLimpa();
45                 break;

```

```

46         case 2:
47             printf("Digite o elemento
48             que deseja inserir: ");
49             scanf("%s %d %lf", nome,
50             &matricula, &nota);
51             strcpy(aluno->nome, nome);
52             aluno->matricula =
53             matricula;
54             aluno->nota = nota;
55             insereElem(abt,aluno);
56             aguardaLimpa();
57             break;
58
59         case 3:
60             printf("Digite o nome do
61             aluno que deseja
62             consultar: ");
63             scanf("%s",nome);
64             strcpy(aluno->nome, nome);
65             busca = pesquisa(abt,aluno);
66             if(busca == 0)
67                 printf("0 aluno nao
68                 esta na arvore!");
69             aguardaLimpa();
70             break;
71
72         case 4:
73             printf("Digite o nome do
74             aluno que deseja
75             remover: ");
76             scanf("%s",nome);
77             strcpy(aluno->nome, nome);
78             removeElem(abt,aluno);
79             aguardaLimpa();
80             break;
81
82         case 5:
83             imprime(abt,0);
84             aguardaLimpa();
85             break;
86
87         case 6:
88             *maior = 0;

```

```

69         strcpy(aluno->nome,      83         printf("Arvore destruida!");
           maior_nt(*abt, 0, maior, 84         aguardaLimpa());
           nome));                85         break;
70     busca = pesquisa(abt,aluno);86
71     aguardaLimpa();            87     default:
72     break;                     88         printf("\nAlgo de errado
73                               89         nao esta certo!\n\n");
74     case 7:                    90         break;
75         *menor = 100000.0;      91     }
76     strcpy(aluno->nome,        92     }while(escolha!=9);
           menor_nt(*abt, 0, menor, 93
           nome));              94
77     busca = pesquisa(abt,aluno);95
78     aguardaLimpa();           96     return 0;
79     break;                    96 }
80
81     case 8:                    codigos/questao12/main.c
82     destroiABP(abt);

```

```

1 all: questao12.o              6
2 gcc questao12.o main.c -o main 7 clean:
3                               8 rm -f questao12.o main
4 questao12.o: questao12.h questao12.c
5 gcc -c questao12.c           codigos/questao12/Makefile

```

Saída

```

0 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

```

0 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

```

0 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

```

0 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

```

0 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
0 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

```

0 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
0 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

```

0 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
0 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

```

0 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
0 aluno nao esta na arvore!

```

Figura 18: Questão 2.1 - Busca Aluno

```

0 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