

UFSJ - Ciências da Computação

Laboratório de Programação 2

Roteiro 9

Nome: Geraldo Arthur Detomi

1.1) TAD: Arvore AVL

AVL.h

```
1  #ifndef AVL_H
2  #define AVL_H
3
4  #include <math.h>
5  #include <stdio.h>
6  #include <stdlib.h>
7  #define MAIOR(a, b) ((a > b) ? (a) : (b))
8
9  typedef struct NO {
10     int info, fb, alt;
11     struct NO *esq;
12     struct NO *dir;
13 } NO;
14
15 typedef struct NO *AVL;
16
17 NO *alocarNO();
18
19 void liberarNO(NO *q);
20
21 AVL *criaAVL();
22
23 void destroiRec(NO *no);
24
25 void destroiAVL(AVL *raiz);
26
27 int estaVazia(AVL *raiz);
28
29 // Calcula FB
30 int altura(NO *raiz);
31
32 int FB(NO *raiz);
33
34 // Funcoes de Rotacao Simples
35 void avl_RotDir(NO **raiz);
36
37 void avl_RotEsq(NO **raiz);
38
39 // Funcoes de Rotacao Dupla
40 void avl_RotEsqDir(NO **raiz);
41
42 void avl_RotDirEsq(NO **raiz);
43
44 void avl_RotEsqDir2(NO **raiz);
45
46 void avl_RotDirEsq2(NO **raiz);
47
48 // Funcoes Auxiliares referentes a cada filho
49 void avl_AuxFE(NO **raiz);
50
51 void avl_AuxFD(NO **raiz);
52
53 int insereRec(NO **raiz, int elem);
54
55 int insereElem(AVL *raiz, int elem);
56
57 int pesquisaRec(NO **raiz, int elem);
58
59 int pesquisa(AVL *raiz, int elem);
60
61 int removeRec(NO **raiz, int elem);
62
63 int removeElem(AVL *raiz, int elem);
64
65 void em_ordem(NO *raiz, int nivel);
```

```

66
67 void pre_ordem(NO *raiz, int nivel);
68
69 void pos_ordem(NO *raiz, int nivel);
70
71 void imprime(AVL *raiz);
72
73 #endif

```

AVL.c

```

1  #include "AVL.h"
2
3  NO *alocarNO() { return (NO *)malloc(sizeof(NO)); }
4
5  void liberarNO(NO *q) { free(q); }
6
7  AVL *criaAVL() {
8      AVL *raiz = (AVL *)malloc(sizeof(AVL));
9      if (raiz != NULL)
10         *raiz = NULL;
11     return raiz;
12 }
13
14 void destroiRec(NO *no) {
15     if (no == NULL)
16         return;
17     destroiRec(no->esq);
18     destroiRec(no->dir);
19     liberarNO(no);
20     no = NULL;
21 }
22
23 void destroiAVL(AVL *raiz) {
24     if (raiz != NULL) {
25         destroiRec(*raiz);
26         free(raiz);
27     }
28 }
29
30 int estaVazia(AVL *raiz) {
31     if (raiz == NULL)
32         return 0;
33     return (*raiz == NULL);
34 }
35
36 // Calcula FB
37 int altura(NO *raiz) {
38     if (raiz == NULL)
39         return 0;
40     if (raiz->alt > 0)
41         return raiz->alt;
42     else {
43         // printf("Calculando altura do (%d)...\n", raiz->info);
44         return MAIOR(altura(raiz->esq), altura(raiz->dir)) + 1;
45     }
46 }
47
48 int FB(NO *raiz) {
49     if (raiz == NULL)
50         return 0;
51     printf("Calculando FB do (%d)...\n", raiz->info);
52     return altura(raiz->esq) - altura(raiz->dir);
53 }
54
55 // Funcoes de Rotacao Simples
56 void avl_RotDir(NO **raiz) {
57     printf("Rotacao Simples a DIREITA!\n");
58     NO *aux;
59     aux = (*raiz)->esq;
60     (*raiz)->esq = aux->dir;
61     aux->dir = *raiz;
62
63     // Acertando alturas e FB
64     // dos NOs afetados
65     (*raiz)->alt = aux->alt - 1;
66     aux->alt = altura(aux);
67     (*raiz)->alt = altura(*raiz);
68     aux->fb = FB(aux);
69     (*raiz)->fb = FB(*raiz);
70
71     *raiz = aux;

```

```

72 }
73
74 void avl_RotEsq(NO **raiz) {
75     printf("Rotacao Simples a ESQUERDA!\n");
76     NO *aux;
77     aux = (*raiz)->dir;
78     (*raiz)->dir = aux->esq;
79     aux->esq = *raiz;
80
81     // Acertando alturas e Fatores de Balanceamento dos NOs afetados
82     (*raiz)->alt = aux->alt = -1;
83     aux->alt = altura(aux);
84     (*raiz)->alt = altura(*raiz);
85     aux->fb = FB(aux);
86     (*raiz)->fb = FB(*raiz);
87
88     *raiz = aux;
89 }
90
91 // Funcoes de Rotacao Dupla
92 void avl_RotEsqDir(NO **raiz) {
93     printf("Rotacao Dupla ESQUERDA-DIREITA!\n");
94     NO *fe; // filho esquerdo
95     NO *ffd; // filho filho direito
96
97     fe = (*raiz)->esq;
98     ffd = fe->dir;
99
100     fe->dir = ffd->esq;
101     ffd->esq = fe;
102
103     (*raiz)->esq = ffd->dir;
104     ffd->dir = *raiz;
105
106     // Acertando alturas e Fatores de Balanceamento dos NOs afetados
107     (*raiz)->alt = fe->alt = ffd->alt = -1;
108     fe->alt = altura(fe);
109     ffd->alt = altura(ffd);
110     (*raiz)->alt = altura(*raiz);
111     fe->fb = FB(fe);
112     ffd->fb = FB(ffd);
113     (*raiz)->fb = FB(*raiz);
114
115     *raiz = ffd;
116 }
117
118 void avl_RotDirEsq(NO **raiz) {
119     printf("Rotacao Dupla DIREITA-ESQUERDA!\n");
120     NO *fd; // filho direito
121     NO *ffe; // filho filho esquerdo
122
123     fd = (*raiz)->dir;
124     ffe = fd->esq;
125
126     fd->esq = ffe->dir;
127     ffe->dir = fd;
128
129     (*raiz)->dir = ffe->esq;
130     ffe->esq = *raiz;
131
132     // Acertando alturas e Fatores de Balanceamento dos NOs afetados
133     (*raiz)->alt = fd->alt = ffe->alt = -1;
134     fd->alt = altura(fd);
135     ffe->alt = altura(ffe);
136     (*raiz)->alt = altura(*raiz);
137     fd->fb = FB(fd);
138     ffe->fb = FB(ffe);
139     (*raiz)->fb = FB(*raiz);
140
141     *raiz = ffe;
142 }
143
144 void avl_RotEsqDir2(NO **raiz) {
145     printf("Rotacao Dupla 2 ESQUERDA-DIREITA!\n");
146     avl_RotEsq(&(*raiz)->esq);
147     avl_RotDir(raiz);
148 }
149
150 void avl_RotDirEsq2(NO **raiz) {
151     printf("Rotacao Dupla 2 DIREITA-ESQUERDA!\n");
152     avl_RotDir(&(*raiz)->dir);

```

```

153     avl_RotEsq(raiz);
154 }
155
156 // Funcoes Auxiliares referentes a cada filho
157 void avl_AuxFE(NO **raiz) {
158     NO *fe;
159     fe = (*raiz)->esq;
160     if (fe->fb == +1) /* Sinais iguais e positivo*/
161         avl_RotDir(raiz);
162     else /* Sinais diferentes*/
163         avl_RotEsqDir(raiz);
164 }
165
166 void avl_AuxFD(NO **raiz) {
167     NO *fd;
168     fd = (*raiz)->dir;
169     if (fd->fb == -1) /* Sinais iguais e negativos*/
170         avl_RotEsq(raiz);
171     else /* Sinais diferentes*/
172         avl_RotDirEsq(raiz);
173 }
174
175 int insereRec(NO **raiz, int elem) {
176     int ok; // Controle para as chamadas recursivas
177     if (*raiz == NULL) {
178         NO *novo = alocarNO();
179         if (novo == NULL)
180             return 0;
181         novo->info = elem;
182         novo->fb = 0, novo->alt = 1;
183         novo->esq = NULL;
184         novo->dir = NULL;
185         *raiz = novo;
186         return 1;
187     } else {
188         if ((*raiz)->info == elem) {
189             printf("Elemento Existente!\n");
190             ok = 0;
191         }
192         if (elem < (*raiz)->info) {
193             ok = insereRec(&(*raiz)->esq, elem);
194             if (ok) {
195                 switch ((*raiz)->fb) {
196                     case -1:
197                         (*raiz)->fb = 0;
198                         ok = 0;
199                         break;
200                     case 0:
201                         (*raiz)->fb = +1;
202                         (*raiz)->alt++;
203                         break;
204                     case +1:
205                         avl_AuxFE(raiz);
206                         ok = 0;
207                         break;
208                 }
209             }
210         } else if (elem > (*raiz)->info) {
211             ok = insereRec(&(*raiz)->dir, elem);
212             if (ok) {
213                 switch ((*raiz)->fb) {
214                     case +1:
215                         (*raiz)->fb = 0;
216                         ok = 0;
217                         break;
218                     case 0:
219                         (*raiz)->fb = -1;
220                         (*raiz)->alt++;
221                         break;
222                     case -1:
223                         avl_AuxFD(raiz);
224                         ok = 0;
225                         break;
226                 }
227             }
228         }
229     }
230     return ok;
231 }
232
233 int insereElem(AVL *raiz, int elem) {

```

```

234     if (raiz == NULL)
235         return 0;
236     return insereRec(raiz, elem);
237 }
238
239 int pesquisaRec(NO **raiz, int elem) {
240     if (*raiz == NULL)
241         return 0;
242     if ((*raiz)->info == elem)
243         return 1;
244     if (elem < (*raiz)->info)
245         return pesquisaRec(&(*raiz)->esq, elem);
246     else
247         return pesquisaRec(&(*raiz)->dir, elem);
248 }
249
250 int pesquisa(AVL *raiz, int elem) {
251     if (raiz == NULL)
252         return 0;
253     if (estaVazia(raiz))
254         return 0;
255     return pesquisaRec(raiz, elem);
256 }
257
258 int removeRec(NO **raiz, int elem) {
259     if (*raiz == NULL)
260         return 0;
261     int ok;
262     if ((*raiz)->info == elem) {
263         NO *aux;
264         if ((*raiz)->esq == NULL && (*raiz)->dir == NULL) {
265             // Caso 1 - NO sem filhos
266             printf("Caso 1: Liberando %d.\n", (*raiz)->info);
267             liberarNO(*raiz);
268             *raiz = NULL;
269         } else if ((*raiz)->esq == NULL) {
270             // Caso 2.1 - Possui apenas uma subarvore direita
271             printf("Caso 2.1: Liberando %d.\n", (*raiz)->info);
272             aux = *raiz;
273             *raiz = (*raiz)->dir;
274             liberarNO(aux);
275         } else if ((*raiz)->dir == NULL) {
276             // Caso 2.2 - Possui apenas uma subarvore esquerda
277             printf("Caso 2.2: Liberando %d.\n", (*raiz)->info);
278             aux = *raiz;
279             *raiz = (*raiz)->esq;
280             liberarNO(aux);
281         } else {
282             // Caso 3 - Possui as duas subarvoretas (esq e dir)
283             // Duas estrategias:
284             // 3.1 - Substituir pelo NO com o MAIOR valor da subarvore esquerda
285             // 3.2 - Substituir pelo NO com o MENOR valor da subarvore direita
286             printf("Caso 3: Liberando %d.\n", (*raiz)->info);
287             // Estrategia 3.1:
288             NO *Filho = (*raiz)->esq;
289             while (Filho->dir != NULL) // Localiza o MAIOR valor da subarvore esquerda
290                 Filho = Filho->dir;
291             (*raiz)->info = Filho->info;
292             Filho->info = elem;
293             return removeRec(&(*raiz)->esq, elem);
294         }
295         return 1;
296     } else if (elem < (*raiz)->info) {
297         ok = removeRec(&(*raiz)->esq, elem);
298         if (ok) {
299             switch ((*raiz)->fb) {
300                 case +1:
301                     case 0:
302                         // Acertando alturas e Fatores de Balanceamento dos NOs afetados
303                         (*raiz)->alt = -1;
304                         (*raiz)->alt = altura(*raiz);
305                         (*raiz)->fb = FB(*raiz);
306                         break;
307                     case -1:
308                         avl_AuxFD(raiz);
309                         break;
310             }
311         }
312     } else {
313         ok = removeRec(&(*raiz)->dir, elem);
314         if (ok) {

```

```

315     switch ((*raiz)->fb) {
316     case -1:
317     case 0:
318         // Acertando alturas e Fatores de Balanceamento dos NOs afetados
319         (*raiz)->alt = -1;
320         (*raiz)->alt = altura(*raiz);
321         (*raiz)->fb = FB(*raiz);
322         break;
323     case +1:
324         avl_AuxFE(raiz);
325         break;
326     }
327 }
328 }
329 return ok;
330 }
331
332 int removeElem(AVL *raiz, int elem) {
333     if (pesquisa(raiz, elem) == 0) {
334         printf("Elemento inexistente!\n");
335         return 0;
336     }
337     return removeRec(raiz, elem);
338 }
339
340 void em_ordem(NO *raiz, int nivel) {
341     if (raiz != NULL) {
342         em_ordem(raiz->esq, nivel + 1);
343         // printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
344         printf("[%d, %d, %d, %d] ", raiz->info, raiz->fb, nivel, raiz->alt);
345         em_ordem(raiz->dir, nivel + 1);
346     }
347 }
348
349 void pre_ordem(NO *raiz, int nivel) {
350     if (raiz != NULL) {
351         printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
352         pre_ordem(raiz->esq, nivel + 1);
353         pre_ordem(raiz->dir, nivel + 1);
354     }
355 }
356
357 void pos_ordem(NO *raiz, int nivel) {
358     if (raiz != NULL) {
359         pos_ordem(raiz->esq, nivel + 1);
360         pos_ordem(raiz->dir, nivel + 1);
361         printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
362     }
363 }
364
365 void imprime(AVL *raiz) {
366     if (raiz == NULL)
367         return;
368     if (estaVazia(raiz)) {
369         printf("Arvore Vazia!\n");
370         return;
371     }
372     // printf("\nEm Ordem: [INFO, FB, NIVEL]\n");
373     printf("\nEm Ordem: [INFO, FB, NIVEL, altura]\n");
374     em_ordem(*raiz, 0);
375     // printf("\nPre Ordem: "); pre_ordem(*raiz, 0);
376     // printf("\nPos Ordem: "); pos_ordem(*raiz, 0);
377     printf("\n");
378 }
379
380 int get_quantidade_nos(AVL *raiz) {
381     if (*raiz == NULL) {
382         return 0;
383     }
384
385     int qtd = 1;
386
387     qtd += get_quantidade_nos(&(*raiz)->esq);
388
389     qtd += get_quantidade_nos(&(*raiz)->dir);
390
391     return qtd;
392 }

```

1-1.c

1 | #include "AVL.h"

```
2
3 #define MAX_OPTIONS 8
4
5 enum options {
6     CRIAR = 0,
7     INSERIR,
8     BUSCAR,
9     REMOVER,
10    IMPRIMIR_ORDEM,
11    QUANTIDADE_NOS,
12    DESTRUIR,
13    SAIR,
14 };
15
16 int get_option() {
17     int opt = -1;
18     do {
19         printf("\tOperacoes\n");
20         printf("[%d] Criar AVL, ", CRIAR);
21         printf("[%d] Inserir um elemento, ", INSERIR);
22         printf("[%d] Buscar um elemento, ", BUSCAR);
23         printf("[%d] Remover um elemento, ", REMOVER);
24         printf("[%d] Imprimir a AVL em ordem, ", IMPRIMIR_ORDEM);
25         printf("[%d] Mostrar a quantidade de nós na AVL, ", QUANTIDADE_NOS);
26         printf("[%d] Destruir a AVL, ", DESTRUIR);
27         printf("[%d] Sair do programa \n", SAIR);
28
29         printf("\nInsira a opção desejada: ");
30         scanf("%d", &opt);
31         printf("\n");
32
33         if (opt < 0 || opt >= MAX_OPTIONS) {
34             printf("Opção escolhida inválida!\n");
35         }
36
37     } while (opt < 0 || opt >= MAX_OPTIONS);
38
39     return opt;
40 }
41
42 int get_valor(char *msg) {
43     int value;
44
45     printf("%s", msg);
46     scanf("%d", &value);
47
48     return value;
49 }
50
51 int main() {
52     int opt, valor;
53
54     AVL *arvore = NULL;
55
56     do {
57         opt = get_option();
58
59         switch (opt) {
60             case CRIAR:
61                 printf("Executando comando...\n");
62
63                 if (arvore == NULL) {
64                     arvore = criaAVL();
65                 } else {
66                     destroiAVL(arvore);
67                     arvore = NULL;
68                     arvore = criaAVL();
69                 }
70
71                 printf("Árvore AVL criada com sucesso!\n");
72                 break;
73             case INSERIR:
74                 printf("Executando comando...\n");
75
76                 if (arvore == NULL) {
77                     printf("Árvore AVL não inicializada impossível realizar operação..\n");
78                     break;
79                 }
80
81                 valor = get_valor("Insira um valor: ");
82
```

```
83     if (insereElem(arvore, valor)) {
84         printf("Elemento inserido na árvore com sucesso!\n");
85     } else {
86         printf("Falha ao inserir elemento!\n");
87     }
88
89     break;
90 case BUSCAR:
91     printf("Executando comando...\n");
92
93     if (arvore == NULL) {
94         printf("Árvore AVL não inicializada impossível realizar operação..\n");
95         break;
96     }
97
98     valor = get_valor("Insira o valor para buscar: ");
99
100    if (pesquisa(arvore, valor)) {
101        printf("Elemento está presente na árvore AVL!\n");
102    } else {
103        printf("Elemento não está presente na árvore AVL!\n");
104    }
105
106    break;
107 case REMOVER:
108     printf("Executando comando...\n");
109
110     if (arvore == NULL) {
111         printf("Árvore AVL não inicializada impossível realizar operação..\n");
112         break;
113     }
114
115     valor = get_valor("Elemento a se remover: ");
116
117     if (removeElem(arvore, valor)) {
118         printf("Elemento removido com sucesso!\n");
119     }
120
121     break;
122 case IMPRIMIR_ORDEM:
123     printf("Executando comando...\n");
124
125     if (arvore == NULL) {
126         printf("Árvore AVL não inicializada impossível realizar operação..\n");
127         break;
128     }
129
130     em_ordem(*arvore, 0);
131
132     printf("Árvore imprimida com sucesso!\n");
133
134     break;
135 case QUANTIDADE_NOS:
136     printf("Executando comando...\n");
137
138     if (arvore == NULL) {
139         printf("Árvore AVL não inicializada impossível realizar operação..\n");
140         break;
141     }
142
143     int qtd_nos = get_quantidade_nos(arvore);
144
145     printf("A quantidade de nós é %d\n", qtd_nos);
146
147     break;
148 case DESTRUIR:
149     printf("Executando comando...\n");
150
151     if (arvore == NULL) {
152         printf("Árvore AVL não inicializada impossível realizar operação..\n");
153         break;
154     }
155
156     destroiAVL(arvore);
157     arvore = NULL;
158
159     printf("Árvore AVL destruída com sucesso!");
160
161     break;
162 case SAIR:
163     if (arvore != NULL) {
```



```

164     destroiAVL(arvore);
165     arvore = NULL;
166 }
167
168     printf("Finalizando programa! Até mais!\n");
169     break;
170 }
171 } while (opt != SAIR);
172
173     return 0;
174 }

```

Saída do terminal:

```

[arthurdetomi@arthurdetomi-System-Product-Name ~]$ ./1-1.out
Operacoes
[0] Criar AVL, [1] Inserir um elemento, [2] Buscar um elemento, [3] Remover um elemento, [4] Imprimir a AVL em ordem, [5] Mostrar a quantidade de nós na
Insira a opção desejada: 0
Executando comando...
Árvore AVL criada com sucesso!
Operacoes
[0] Criar AVL, [1] Inserir um elemento, [2] Buscar um elemento, [3] Remover um elemento, [4] Imprimir a AVL em ordem, [5] Mostrar a quantidade de nós na
Insira a opção desejada: 1
Executando comando...
Insira um valor: 2
Elemento inserido na árvore com sucesso!
Operacoes
[0] Criar AVL, [1] Inserir um elemento, [2] Buscar um elemento, [3] Remover um elemento, [4] Imprimir a AVL em ordem, [5] Mostrar a quantidade de nós na
Insira a opção desejada: 1
Executando comando...
Insira um valor: 5
Elemento inserido na árvore com sucesso!
Operacoes
[0] Criar AVL, [1] Inserir um elemento, [2] Buscar um elemento, [3] Remover um elemento, [4] Imprimir a AVL em ordem, [5] Mostrar a quantidade de nós na
Insira a opção desejada: 1
Executando comando...
Insira um valor: 7
Rotacao Simples a ESQUERDA!
Calculando FB do (5)..
Calculando FB do (2)..
Falha ao inserir elemento!
Operacoes
[0] Criar AVL, [1] Inserir um elemento, [2] Buscar um elemento, [3] Remover um elemento, [4] Imprimir a AVL em ordem, [5] Mostrar a quantidade de nós na
Insira a opção desejada: 4
Executando comando...
[2, 0, 1, 1] [5, 0, 0, 2] [7, 0, 1, 1] Árvore imprimida com sucesso!
Operacoes
[0] Criar AVL, [1] Inserir um elemento, [2] Buscar um elemento, [3] Remover um elemento, [4] Imprimir a AVL em ordem, [5] Mostrar a quantidade de nós na
Insira a opção desejada: 5
Executando comando...
A quantidade de nós é 3
Operacoes
[0] Criar AVL, [1] Inserir um elemento, [2] Buscar um elemento, [3] Remover um elemento, [4] Imprimir a AVL em ordem, [5] Mostrar a quantidade de nós na
Insira a opção desejada: 7
Finalizando programa! Até mais!

```

1.2) TAD: Arvore AVL com campo info igual a struct Funcionario

AVL_FUNC.h

```

1  #ifndef AVL_FUNC_H
2  #define AVL_FUNC_H
3
4  #include <math.h>
5  #include <stdio.h>
6  #include <stdlib.h>
7  #define MAIOR(a, b) ((a > b) ? (a) : (b))
8
9  typedef struct Funcionario {
10     char nome[50];
11     double salario;
12     int ano;
13 } Funcionario;
14
15 typedef struct NO {
16     Funcionario info;
17     int fb, alt;
18     struct NO *esq;
19     struct NO *dir;
20 } NO;
21
22 typedef struct NO *AVL;

```

```

23
24 NO *alocarNO();
25
26 void liberarNO(NO *q);
27
28 AVL *criaAVL();
29
30 void destroiRec(NO *no);
31
32 void destroiAVL(AVL *raiz);
33
34 int estaVazia(AVL *raiz);
35
36 // Calcula FB
37 int altura(NO *raiz);
38
39 int FB(NO *raiz);
40
41 // Funcoes de Rotacao Simples
42 void avl_RotDir(NO **raiz);
43
44 void avl_RotEsq(NO **raiz);
45
46 // Funcoes de Rotacao Dupla
47 void avl_RotEsqDir(NO **raiz);
48
49 void avl_RotDirEsq(NO **raiz);
50
51 void avl_RotEsqDir2(NO **raiz);
52
53 void avl_RotDirEsq2(NO **raiz);
54
55 // Funcoes Auxiliares referentes a cada filho
56 void avl_AuxFE(NO **raiz);
57
58 void avl_AuxFD(NO **raiz);
59
60 int insereRec(NO **raiz, Funcionario elem);
61
62 int insereElem(AVL *raiz, Funcionario elem);
63
64 int pesquisa(AVL *raiz, Funcionario *elem);
65
66 int removeRec(NO **raiz, Funcionario elem);
67
68 int removeElem(AVL *raiz, Funcionario elem);
69
70 void em_ordem(NO *raiz, int nivel);
71
72 void pre_ordem(NO *raiz, int nivel);
73
74 void pos_ordem(NO *raiz, int nivel);
75
76 void imprime(AVL *raiz);
77
78 int get_quantidade_nos(AVL *raiz);
79
80 int buscar_elem_por_nome(AVL *raiz, char *nome, Funcionario *f);
81
82 int buscar_com_maior_salario(AVL *raiz, Funcionario *f);
83
84 int buscar_com_menor_salario(AVL *raiz, Funcionario *f);
85
86 #endif

```

AVL_FUNC.c

```

1  #include "AVL_FUNC.h"
2  #include <string.h>
3
4  NO *alocarNO() { return (NO *)malloc(sizeof(NO)); }
5
6  void liberarNO(NO *q) { free(q); }
7
8  AVL *criaAVL() {
9      AVL *raiz = (AVL *)malloc(sizeof(AVL));
10     if (raiz != NULL)
11         *raiz = NULL;
12     return raiz;
13 }
14
15 void destroiRec(NO *no) {

```

```
16     if (no == NULL)
17         return;
18     destroiRec(no->esq);
19     destroiRec(no->dir);
20     liberarNO(no);
21     no = NULL;
22 }
23
24 void destroiAVL(AVL *raiz) {
25     if (raiz != NULL) {
26         destroiRec(*raiz);
27         free(raiz);
28     }
29 }
30
31 int estaVazia(AVL *raiz) {
32     if (raiz == NULL)
33         return 0;
34     return (*raiz == NULL);
35 }
36
37 // Calcula FB
38 int altura(NO *raiz) {
39     if (raiz == NULL)
40         return 0;
41     if (raiz->alt > 0)
42         return raiz->alt;
43     else {
44         // printf("Calculando altura do (%d)..\\n", raiz->info);
45         return MAIOR(altura(raiz->esq), altura(raiz->dir)) + 1;
46     }
47 }
48
49 int FB(NO *raiz) {
50     if (raiz == NULL)
51         return 0;
52     return altura(raiz->esq) - altura(raiz->dir);
53 }
54
55 // Funcoes de Rotacao Simples
56 void avl_RotDir(NO **raiz) {
57     printf("Rotacao Simples a DIREITA!\\n");
58     NO *aux;
59     aux = (*raiz)->esq;
60     (*raiz)->esq = aux->dir;
61     aux->dir = *raiz;
62
63     // Acertando alturas e FB
64     // dos NOs afetados
65     (*raiz)->alt = aux->alt - 1;
66     aux->alt = altura(aux);
67     (*raiz)->alt = altura(*raiz);
68     aux->fb = FB(aux);
69     (*raiz)->fb = FB(*raiz);
70
71     *raiz = aux;
72 }
73
74 void avl_RotEsq(NO **raiz) {
75     printf("Rotacao Simples a ESQUERDA!\\n");
76     NO *aux;
77     aux = (*raiz)->dir;
78     (*raiz)->dir = aux->esq;
79     aux->esq = *raiz;
80
81     // Acertando alturas e Fatores de Balanceamento dos NOs afetados
82     (*raiz)->alt = aux->alt - 1;
83     aux->alt = altura(aux);
84     (*raiz)->alt = altura(*raiz);
85     aux->fb = FB(aux);
86     (*raiz)->fb = FB(*raiz);
87
88     *raiz = aux;
89 }
90
91 // Funcoes de Rotacao Dupla
92 void avl_RotEsqDir(NO **raiz) {
93     printf("Rotacao Dupla ESQUERDA-DIREITA!\\n");
94     NO *fe; // filho esquerdo
95     NO *ffd; // filho filho direito
96 }
```

```

97     fe = (*raiz)->esq;
98     ffd = fe->dir;
99
100    fe->dir = ffd->esq;
101    ffd->esq = fe;
102
103    (*raiz)->esq = ffd->dir;
104    ffd->dir = *raiz;
105
106    // Acertando alturas e Fatores de Balanceamento dos NOs afetados
107    (*raiz)->alt = fe->alt = ffd->alt = -1;
108    fe->alt = altura(fe);
109    ffd->alt = altura(ffd);
110    (*raiz)->alt = altura(*raiz);
111    fe->fb = FB(fe);
112    ffd->fb = FB(ffd);
113    (*raiz)->fb = FB(*raiz);
114
115    *raiz = ffd;
116 }
117
118 void avl_RotDirEsq(NO **raiz) {
119     printf("Rotacao Dupla DIREITA-ESQUERDA!\n");
120     NO *fd; // filho direito
121     NO *ffe; // filho filho esquerdo
122
123     fd = (*raiz)->dir;
124     ffe = fd->esq;
125
126     fd->esq = ffe->dir;
127     ffe->dir = fd;
128
129     (*raiz)->dir = ffe->esq;
130     ffe->esq = *raiz;
131
132     // Acertando alturas e Fatores de Balanceamento dos NOs afetados
133     (*raiz)->alt = fd->alt = ffe->alt = -1;
134     fd->alt = altura(fd);
135     ffe->alt = altura(fffe);
136     (*raiz)->alt = altura(*raiz);
137     fd->fb = FB(fd);
138     ffe->fb = FB(fffe);
139     (*raiz)->fb = FB(*raiz);
140
141     *raiz = ffe;
142 }
143
144 void avl_RotEsqDir2(NO **raiz) {
145     printf("Rotacao Dupla 2 ESQUERDA-DIREITA!\n");
146     avl_RotEsq(&(*raiz)->esq);
147     avl_RotDir(raiz);
148 }
149
150 void avl_RotDirEsq2(NO **raiz) {
151     printf("Rotacao Dupla 2 DIREITA-ESQUERDA!\n");
152     avl_RotDir(&(*raiz)->dir);
153     avl_RotEsq(raiz);
154 }
155
156 // Funcoes Auxiliares referentes a cada filho
157 void avl_AuxFE(NO **raiz) {
158     NO *fe;
159     fe = (*raiz)->esq;
160     if (fe->fb == +1) /* Sinais iguais e positivo*/
161         avl_RotDir(raiz);
162     else /* Sinais diferentes*/
163         avl_RotEsqDir(raiz);
164 }
165
166 void avl_AuxFD(NO **raiz) {
167     NO *fd;
168     fd = (*raiz)->dir;
169     if (fd->fb == -1) /* Sinais iguais e negativos*/
170         avl_RotEsq(raiz);
171     else /* Sinais diferentes*/
172         avl_RotDirEsq(raiz);
173 }
174
175 int insereRec(NO **raiz, Funcionario elem) {
176     int ok; // Controle para as chamadas recursivas
177     if (*raiz == NULL) {

```

```

178     NO *novo = alocarNO();
179     if (novo == NULL)
180         return 0;
181     novo->info = elem;
182     novo->fb = 0, novo->alt = 1;
183     novo->esq = NULL;
184     novo->dir = NULL;
185     *raiz = novo;
186     return 1;
187 } else {
188     if ((*raiz)->info.salario == elem.salario) {
189         printf("Elemento Existente!\n");
190         ok = 0;
191     }
192     if (elem.salario < (*raiz)->info.salario) {
193         ok = insereRec(&(*raiz)->esq, elem);
194         if (ok) {
195             switch ((*raiz)->fb) {
196                 case -1:
197                     (*raiz)->fb = 0;
198                     ok = 0;
199                     break;
200                 case 0:
201                     (*raiz)->fb = +1;
202                     (*raiz)->alt++;
203                     break;
204                 case +1:
205                     avl_AuxFE(raiz);
206                     ok = 0;
207                     break;
208             }
209         }
210     } else if (elem.salario > (*raiz)->info.salario) {
211         ok = insereRec(&(*raiz)->dir, elem);
212         if (ok) {
213             switch ((*raiz)->fb) {
214                 case +1:
215                     (*raiz)->fb = 0;
216                     ok = 0;
217                     break;
218                 case 0:
219                     (*raiz)->fb = -1;
220                     (*raiz)->alt++;
221                     break;
222                 case -1:
223                     avl_AuxFD(raiz);
224                     ok = 0;
225                     break;
226             }
227         }
228     }
229 }
230 return ok;
231 }
232
233 int insereElem(AVL *raiz, Funcionario elem) {
234     if (raiz == NULL)
235         return 0;
236     return insereRec(raiz, elem);
237 }
238
239 int pesquisaRec(NO **raiz, Funcionario *elem) {
240     if (*raiz == NULL)
241         return 0;
242     if ((*raiz)->info.salario == elem->salario) {
243         *elem = (*raiz)->info;
244         return 1;
245     }
246     if (elem->salario < (*raiz)->info.salario)
247         return pesquisaRec(&(*raiz)->esq, elem);
248     else
249         return pesquisaRec(&(*raiz)->dir, elem);
250 }
251
252 int pesquisa(AVL *raiz, Funcionario *elem) {
253     if (raiz == NULL)
254         return 0;
255     if (estaVazia(raiz))
256         return 0;
257     return pesquisaRec(raiz, elem);
258 }

```

```

259
260 int removeRec(NO **raiz, Funcionario elem) {
261     if (*raiz == NULL)
262         return 0;
263     int ok;
264     if ((*raiz)->info.salario == elem.salario) {
265         NO *aux;
266         if ((*raiz)->esq == NULL && (*raiz)->dir == NULL) {
267             // Caso 1 - NO sem filhos
268             liberarNO(*raiz);
269             *raiz = NULL;
270         } else if ((*raiz)->esq == NULL) {
271             // Caso 2.1 - Possui apenas uma subarvore direita
272             aux = *raiz;
273             *raiz = (*raiz)->dir;
274             liberarNO(aux);
275         } else if ((*raiz)->dir == NULL) {
276             // Caso 2.2 - Possui apenas uma subarvore esquerda
277             aux = *raiz;
278             *raiz = (*raiz)->esq;
279             liberarNO(aux);
280         } else {
281             // Caso 3 - Possui as duas subarvoretas (esq e dir)
282             // Duas estrategias:
283             // 3.1 - Substituir pelo NO com o MAIOR valor da subarvore esquerda
284             // 3.2 - Substituir pelo NO com o MENOR valor da subarvore direita
285             // Estrategia 3.1:
286             NO *Filho = (*raiz)->esq;
287             while (Filho->dir != NULL) // Localiza o MAIOR valor da subarvore esquerda
288                 Filho = Filho->dir;
289             (*raiz)->info = Filho->info;
290             Filho->info = elem;
291             return removeRec(&(*raiz)->esq, elem);
292         }
293         return 1;
294     } else if (elem.salario < (*raiz)->info.salario) {
295         ok = removeRec(&(*raiz)->esq, elem);
296         if (ok) {
297             switch ((*raiz)->fb) {
298                 case +1:
299                 case 0:
300                     // Acertando alturas e Fatores de Balanceamento dos NOs afetados
301                     (*raiz)->alt = -1;
302                     (*raiz)->alt = altura(*raiz);
303                     (*raiz)->fb = FB(*raiz);
304                     break;
305                 case -1:
306                     avl_AuxFD(raiz);
307                     break;
308             }
309         }
310     } else {
311         ok = removeRec(&(*raiz)->dir, elem);
312         if (ok) {
313             switch ((*raiz)->fb) {
314                 case -1:
315                 case 0:
316                     // Acertando alturas e Fatores de Balanceamento dos NOs afetados
317                     (*raiz)->alt = -1;
318                     (*raiz)->alt = altura(*raiz);
319                     (*raiz)->fb = FB(*raiz);
320                     break;
321                 case +1:
322                     avl_AuxFE(raiz);
323                     break;
324             }
325         }
326     }
327     return ok;
328 }
329
330 int removeElem(AVL *raiz, Funcionario elem) {
331     if (pesquisa(raiz, &elem) == 0) {
332         printf("Elemento inexistente!\n");
333         return 0;
334     }
335     return removeRec(raiz, elem);
336 }
337
338 void em_ordem(NO *raiz, int nivel) {
339     if (raiz != NULL) {

```

```

340     em_ordem(raiz->esq, nivel + 1);
341     // printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
342     printf("[%s, %.2lf,%d, %d, %d] ", raiz->info.nome, raiz->info.salario,
343           raiz->fb, nivel, raiz->alt);
344     em_ordem(raiz->dir, nivel + 1);
345 }
346 }
347
348 void imprime(AVL *raiz) {
349     if (raiz == NULL)
350         return;
351     if (estaVazia(raiz)) {
352         printf("Arvore Vazia!\n");
353         return;
354     }
355     // printf("\nEm Ordem: [INFO, FB, NIVEL]\n");
356     printf("\nEm Ordem: [INFO, FB, NIVEL, altura]\n");
357     em_ordem(*raiz, 0);
358     // printf("\nPre Ordem: "); pre_ordem(*raiz, 0);
359     // printf("\nPos Ordem: "); pos_ordem(*raiz, 0);
360     printf("\n");
361 }
362
363 int get_quantidade_nos(AVL *raiz) {
364     if (*raiz == NULL) {
365         return 0;
366     }
367
368     int qtd = 1;
369
370     qtd += get_quantidade_nos(&(*raiz)->esq);
371
372     qtd += get_quantidade_nos(&(*raiz)->dir);
373
374     return qtd;
375 }
376
377 int buscar_com_maior_salario(AVL *raiz, Funcionario *f) {
378     if (*raiz == NULL) {
379         return 0;
380     }
381
382     if ((*raiz)->dir == NULL) {
383         *f = (*raiz)->info;
384         return 1;
385     }
386
387     return buscar_com_maior_salario(&(*raiz)->dir, f);
388 }
389
390 int buscar_com_menor_salario(AVL *raiz, Funcionario *f) {
391     if (*raiz == NULL) {
392         return 0;
393     }
394
395     if ((*raiz)->esq == NULL) {
396         *f = (*raiz)->info;
397         return 1;
398     }
399
400     return buscar_com_menor_salario(&(*raiz)->esq, f);
401 }
402
403 int buscar_elem_por_nome(AVL *raiz, char *nome, Funcionario *f) {
404     if (*raiz == NULL) {
405         return 0;
406     }
407
408     if (strcmp(nome, (*raiz)->info.nome) == 0) {
409         *f = (*raiz)->info;
410         return 1;
411     }
412
413     return buscar_elem_por_nome(&(*raiz)->dir, nome, f) ||
414           buscar_elem_por_nome(&(*raiz)->esq, nome, f);
415 }

```

1-2.c

```

1 #include "AVL_FUNC.h"
2
3 #define MAX_OPTIONS 9

```

```
4
5 enum options {
6     CRIAR = 0,
7     INSERIR,
8     BUSCAR,
9     REMOVER,
10    IMPRIMIR_ORDEM,
11    MAIOR_SALARIO,
12    MENOR_SALARIO,
13    DESTRUIR,
14    SAIR,
15 };
16
17 int get_option() {
18     int opt = -1;
19     do {
20         printf("\tOperacoes\n");
21         printf("[%d] Criar AVL, ", CRIAR);
22         printf("[%d] Inserir um Funcionario pelo salário, ", INSERIR);
23         printf(
24             "[%d] Buscar um Funcionario pelo salario e imprimir suas informacoes, ",
25             BUSCAR);
26         printf("[%d] Remover um funcionario pelo nome, ", REMOVER);
27         printf("[%d] Imprimir a AVL em ordem, ", IMPRIMIR_ORDEM);
28         printf("[%d] Imprimir as informacoes do Funcionario com o maior salario, ",
29             MAIOR_SALARIO);
30         printf("[%d] Imprimir as informacoes do Funcionario com o menor salario, ",
31             MENOR_SALARIO);
32         printf("[%d] Destruir a AVL, ", DESTRUIR);
33         printf("[%d] Sair do programa \n", SAIR);
34
35         printf("\nInsira a opção desejada: ");
36         scanf("%d", &opt);
37         printf("\n");
38
39         if (opt < 0 || opt >= MAX_OPTIONS) {
40             printf("Opção escolhida inválida!\n");
41         }
42
43     } while (opt < 0 || opt >= MAX_OPTIONS);
44
45     return opt;
46 }
47
48 int get_valor(char *msg) {
49     int value;
50
51     printf("%s", msg);
52     scanf("%d", &value);
53
54     return value;
55 }
56
57 Funcionario get_funcionario() {
58     Funcionario func;
59
60     printf("Digite o nome do funcionario:");
61     scanf("%49s", func.nome);
62     printf("Digite o ano de contratacao:");
63     scanf("%d", &func.ano);
64     printf("Digite o salario do funcionario:");
65     scanf("%lf", &func.salario);
66
67     return func;
68 }
69
70 void print_funcionario(Funcionario *f) {
71     printf("Funcionario:\n");
72     printf("Nome: %s, Ano de contratacao: %d, Salário: %.2lf\n", f->nome, f->ano,
73         f->salario);
74 }
75
76 double get_salario(char *msg) {
77     double value;
78
79     printf("%s", msg);
80     scanf("%lf", &value);
81
82     return value;
83 }
84
```



```
85 int main() {
86     int opt;
87
88     double valor;
89     Funcionario elem;
90
91     AVL *arvore = NULL;
92
93     do {
94         opt = get_option();
95
96         switch (opt) {
97             case CRIAR:
98                 printf("Executando comando...\n");
99
100                 if (arvore == NULL) {
101                     arvore = criaAVL();
102                 } else {
103                     destroiAVL(arvore);
104                     arvore = NULL;
105                     arvore = criaAVL();
106                 }
107
108                 printf("Árvore AVL criada com sucesso!\n");
109                 break;
110             case INSERIR:
111                 printf("Executando comando...\n");
112
113                 if (arvore == NULL) {
114                     printf("Árvore AVL não inicializada impossível realizar operação..\n");
115                     break;
116                 }
117
118                 elem = get_funcionario();
119
120                 if (insereElem(arvore, elem)) {
121                     printf("Elemento inserido na árvore com sucesso!\n");
122                 } else {
123                     printf("Falha ao inserir elemento!\n");
124                 }
125
126                 break;
127             case BUSCAR:
128                 printf("Executando comando...\n");
129
130                 if (arvore == NULL) {
131                     printf("Árvore AVL não inicializada impossível realizar operação..\n");
132                     break;
133                 }
134
135                 valor = get_salario("Insira o salario do funcionario para busca: ");
136
137                 Funcionario f;
138                 f.salario = valor;
139
140                 if (pesquisa(arvore, &f)) {
141                     printf("Funcionario presente\n");
142                     print_funcionario(&f);
143                     printf("Elemento está presente na árvore AVL!\n");
144                 } else {
145                     printf("Elemento não está presente na árvore AVL!\n");
146                 }
147
148                 break;
149             case REMOVER:
150                 printf("Executando comando...\n");
151
152                 if (arvore == NULL) {
153                     printf("Árvore AVL não inicializada impossível realizar operação..\n");
154                     break;
155                 }
156
157                 char nome_pesquisa[50];
158                 printf("Digite o nome do funcionario para remoção: ");
159                 scanf("%49s", nome_pesquisa);
160
161                 int encontrado = buscar_elem_por_nome(arvore, nome_pesquisa, &elem);
162
163                 if (encontrado && removeElem(arvore, elem)) {
164                     printf("Elemento removido com sucesso!\n");
165                 }
```

```

166
167     break;
168 case IMPRIMIR_ORDEM:
169     printf("Executando comando...\n");
170
171     if (arvore == NULL) {
172         printf("Árvore AVL não inicializada impossível realizar operação..\n");
173         break;
174     }
175
176     em_ordem(*arvore, 0);
177
178     printf("Árvore imprimida com sucesso!\n");
179
180     break;
181 case MAIOR_SALARIO:
182     printf("Executando comando...\n");
183
184     if (arvore == NULL) {
185         printf("Árvore AVL não inicializada impossível realizar operação..\n");
186         break;
187     }
188
189     if (buscar_com_maior_salario(arvore, &elem)) {
190         print_funcionario(&elem);
191     } else {
192         printf("Falha ao encontrar funcionário com maior salário\n");
193     }
194
195     break;
196 case MENOR_SALARIO:
197     printf("Executando comando...\n");
198
199     if (arvore == NULL) {
200         printf("Árvore AVL não inicializada impossível realizar operação..\n");
201         break;
202     }
203
204     if (buscar_com_menor_salario(arvore, &elem)) {
205         print_funcionario(&elem);
206     } else {
207         printf("Falha ao encontrar funcionário com maior salário\n");
208     }
209
210     break;
211 case DESTRUIR:
212     printf("Executando comando...\n");
213
214     if (arvore == NULL) {
215         printf("Árvore AVL não inicializada impossível realizar operação..\n");
216         break;
217     }
218
219     destroiAVL(arvore);
220     arvore = NULL;
221
222     printf("Árvore AVL destruída com sucesso!");
223
224     break;
225 case SAIR:
226     if (arvore != NULL) {
227         destroiAVL(arvore);
228         arvore = NULL;
229     }
230
231     printf("Finalizando programa! Até mais!\n");
232     break;
233 }
234 } while (opt != SAIR);
235
236 return 0;
237 }

```

Saída do terminal:

```
arthurdetomi@arthurdetomi-System-Product-Name: ~/Documents/UFSJ-Graduacao/UFSJ-2025_1/Lab_Prog_2/roteiro_9 on main*** 25-05-30 - 15:54:37
$ ./1-2.out
Operacoes
[0] Criar AVL, [1] Inserir um Funcionario pelo sal'ario, [2] Buscar um Funcionario pelo salario e imprimir suas informacoes, [3] Remover um funcionario pelo nome, [4] Imprimir a
rio, [7] Destruir a AVL, [8] Sair do programa

Insira a opção desejada: 0

Executando comando...
Árvore AVL criada com sucesso!
Operacoes
[0] Criar AVL, [1] Inserir um Funcionario pelo sal'ario, [2] Buscar um Funcionario pelo salario e imprimir suas informacoes, [3] Remover um funcionario pelo nome, [4] Imprimir a
rio, [7] Destruir a AVL, [8] Sair do programa

Insira a opção desejada: 1

Executando comando...
Digite o nome do funcionario:juca
Digite o ano de contratacao:2010
Digite o salario do funcionario:4500
Elemento inserido na árvore com sucesso!
Operacoes
[0] Criar AVL, [1] Inserir um Funcionario pelo sal'ario, [2] Buscar um Funcionario pelo salario e imprimir suas informacoes, [3] Remover um funcionario pelo nome, [4] Imprimir a
rio, [7] Destruir a AVL, [8] Sair do programa

Insira a opção desejada: 1

Executando comando...
Digite o nome do funcionario:matias
Digite o ano de contratacao:2000
Digite o salario do funcionario:4500
Elemento Existente!
Falha ao inserir elemento!
Operacoes
[0] Criar AVL, [1] Inserir um Funcionario pelo sal'ario, [2] Buscar um Funcionario pelo salario e imprimir suas informacoes, [3] Remover um funcionario pelo nome, [4] Imprimir a
rio, [7] Destruir a AVL, [8] Sair do programa

Insira a opção desejada: 5

Executando comando...
Funcionario:
Nome: juca, Ano de contratacao: 2010, Salário: 4500.00
Operacoes
[0] Criar AVL, [1] Inserir um Funcionario pelo sal'ario, [2] Buscar um Funcionario pelo salario e imprimir suas informacoes, [3] Remover um funcionario pelo nome, [4] Imprimir a
rio, [7] Destruir a AVL, [8] Sair do programa

Insira a opção desejada: 6

Executando comando...
Funcionario:
Nome: juca, Ano de contratacao: 2010, Salário: 4500.00
Operacoes
[0] Criar AVL, [1] Inserir um Funcionario pelo sal'ario, [2] Buscar um Funcionario pelo salario e imprimir suas informacoes, [3] Remover um funcionario pelo nome, [4] Imprimir a
rio, [7] Destruir a AVL, [8] Sair do programa

Insira a opção desejada: 3

Executando comando...
Digite o nome do funcionario para remoção: matias
Operacoes
[0] Criar AVL, [1] Inserir um Funcionario pelo sal'ario, [2] Buscar um Funcionario pelo salario e imprimir suas informacoes, [3] Remover um funcionario pelo nome, [4] Imprimir a
rio, [7] Destruir a AVL, [8] Sair do programa

Insira a opção desejada: 4

Executando comando...
[juca, 4500.00,0,0,1] Árvore imprimida com sucesso!
Operacoes
[0] Criar AVL, [1] Inserir um Funcionario pelo sal'ario, [2] Buscar um Funcionario pelo salario e imprimir suas informacoes, [3] Remover um funcionario pelo nome, [4] Imprimir a
rio, [7] Destruir a AVL, [8] Sair do programa

Insira a opção desejada: 8

Finalizando programa! Até mais!
```

Resposta a respeito das complexidades

Como a AVL usa o salário como chave e mantém-se balanceada com as rotações, encontrar o funcionário com maior e menor salário leva tempo $O(\log n)$. Para encontrar o maior salário, basta seguir pela subárvore direita até chegar no último nó, que contém o maior valor. Já para o menor salário, é só fazer o mesmo caminho, mas seguindo sempre pela subárvore esquerda.