

roteiro-09/ex01-01.h

```
1  #ifndef AVL_H
2  #define AVL_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
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     return (NO*)malloc(sizeof(NO));
19 }
20
21 void liberarNO(NO* q) {
22     free(q);
23 }
24
25 AVL* criaAVL() {
26     AVL* raiz = (AVL*)malloc(sizeof(AVL));
27     if (raiz != NULL)
28         *raiz = NULL;
29     return raiz;
30 }
31
32 void destroiRec(NO* no) {
33     if (no == NULL) return;
34     destroiRec(no->esq);
35     destroiRec(no->dir);
36     liberarNO(no);
37     no = NULL;
38 }
39
40 void destroiAVL(AVL** raiz) {
41     if (*raiz != NULL) {
42         destroiRec(**raiz);
43         free(*raiz);
44         *raiz = NULL;
45     }
46 }
47
48 int estaVazia(AVL* raiz) {
49     if (raiz == NULL) return 0;
50     return (*raiz == NULL);
51 }
52
53 // Calcula FB
54 int altura(NO* raiz) {
55     if (raiz == NULL) return 0;
56     if (raiz->alt > 0)
```

```

57     return raiz->alt;
58 else {
59     // printf("Calculando altura do (%d)..\n", raiz->info);
60     return MAIOR(altura(raiz->esq), altura(raiz->dir)) + 1;
61 }
62 }
63
64 int FB(NO* raiz) {
65     if (raiz == NULL) return 0;
66     printf("Calculando FB do (%d)..\n", raiz->info);
67     return altura(raiz->esq) - altura(raiz->dir);
68 }
69
70 // Funcoes de Rotacao Simples
71 void avl_RotDir(NO** raiz) {
72     printf("Rotacao Simples a DIREITA!\n");
73     NO* aux;
74     aux = (*raiz)->esq;
75     (*raiz)->esq = aux->dir;
76     aux->dir = *raiz;
77
78     (*raiz)->alt = aux->alt = -1;
79     aux->alt = altura(aux);
80     (*raiz)->alt = altura(*raiz);
81     aux->fb = FB(aux);
82     (*raiz)->fb = FB(*raiz);
83
84     *raiz = aux;
85 }
86
87 void avl_RotEsq(NO** raiz) {
88     printf("Rotacao Simples a ESQUERDA!\n");
89     NO* aux;
90     aux = (*raiz)->dir;
91     (*raiz)->dir = aux->esq;
92     aux->esq = *raiz;
93
94     (*raiz)->alt = aux->alt = -1;
95     aux->alt = altura(aux);
96     (*raiz)->alt = altura(*raiz);
97     aux->fb = FB(aux);
98     (*raiz)->fb = FB(*raiz);
99
100     *raiz = aux;
101 }
102
103 void avl_RotEsqDir(NO** raiz) {
104     printf("Rotacao Dupla ESQUERDA-DIREITA!\n");
105     NO* fe; // filho esquerdo
106     NO* ffd; // filho filho direito
107
108     fe = (*raiz)->esq;
109     ffd = fe->dir;
110
111     fe->dir = ffd->esq;
112     ffd->esq = fe;
113
114     (*raiz)->esq = ffd->dir;
115     ffd->dir = *raiz;

```

```

116
117     (*raiz)->alt = fe->alt = ffd->alt = -1;
118     fe->alt = altura(fe);
119     ffd->alt = altura(ffd);
120     (*raiz)->alt = altura(*raiz);
121     fe->fb = FB(fe);
122     ffd->fb = FB(ffd);
123     (*raiz)->fb = FB(*raiz);
124
125     *raiz = ffd;
126 }
127
128 void avl_RotDirEsq(NO** raiz) {
129     printf("Rotacao Dupla DIREITA-ESQUERDA!\n");
130     NO* fd;    // filho direito
131     NO* ffe;   // filho filho esquerdo
132
133     fd = (*raiz)->dir;
134     ffe = fd->esq;
135
136     fd->esq = ffe->dir;
137     ffe->dir = fd;
138
139     (*raiz)->dir = ffe->esq;
140     ffe->esq = *raiz;
141
142     (*raiz)->alt = fd->alt = ffe->alt = -1;
143     fd->alt = altura(fd);
144     ffe->alt = altura(ffe);
145     (*raiz)->alt = altura(*raiz);
146     fd->fb = FB(fd);
147     ffe->fb = FB(ffe);
148     (*raiz)->fb = FB(*raiz);
149
150     *raiz = ffe;
151 }
152
153 void avl_RotEsqDir2(NO** raiz) {
154     printf("Rotacao Dupla 2 ESQUERDA-DIREITA!\n");
155     avl_RotEsq(&(*raiz)->esq);
156     avl_RotDir(raiz);
157 }
158
159 void avl_RotDirEsq2(NO** raiz) {
160     printf("Rotacao Dupla 2 DIREITA-ESQUERDA!\n");
161     avl_RotDir(&(*raiz)->dir);
162     avl_RotEsq(raiz);
163 }
164
165 void avl_AuxFE(NO** raiz) {
166     NO* fe;
167     fe = (*raiz)->esq;
168     if (fe->fb == +1) /* Sinais iguais e positivo*/
169         avl_RotDir(raiz);
170     else /* Sinais diferentes*/
171         avl_RotEsqDir(raiz);
172 }
173
174 void avl_AuxFD(NO** raiz) {

```

```

175     NO* fd;
176     fd = (*raiz)->dir;
177     if (fd->fb == -1) /* Sinais iguais e negativos*/
178         avl_RotEsq(raiz);
179     else /* Sinais diferentes*/
180         avl_RotDirEsq(raiz);
181 }
182
183 int insereRec(NO** raiz, int elem) {
184     int ok; // Controle para as chamadas recursivas
185     if (*raiz == NULL) {
186         NO* novo = alocarNO();
187         if (novo == NULL) return 0;
188         novo->info = elem;
189         novo->fb = 0, novo->alt = 1;
190         novo->esq = NULL;
191         novo->dir = NULL;
192         *raiz = novo;
193         return 1;
194     } else {
195         if ((*raiz)->info == elem) {
196             printf("Elemento Existente!\n");
197             ok = 0;
198         }
199         if (elem < (*raiz)->info) {
200             ok = insereRec(&(*raiz)->esq, elem);
201             if (ok) {
202                 switch ((*raiz)->fb) {
203                     case -1:
204                         (*raiz)->fb = 0;
205                         ok = 0;
206                         break;
207                     case 0:
208                         (*raiz)->fb = +1;
209                         (*raiz)->alt++;
210                         break;
211                     case +1:
212                         avl_AuxFE(raiz);
213                         ok = 0;
214                         break;
215                 }
216             }
217         } else if (elem > (*raiz)->info) {
218             ok = insereRec(&(*raiz)->dir, elem);
219             if (ok) {
220                 switch ((*raiz)->fb) {
221                     case +1:
222                         (*raiz)->fb = 0;
223                         ok = 0;
224                         break;
225                     case 0:
226                         (*raiz)->fb = -1;
227                         (*raiz)->alt++;
228                         break;
229                     case -1:
230                         avl_AuxFD(raiz);
231                         ok = 0;
232                         break;
233                 }

```

```

234     }
235 }
236 }
237 return ok;
238 }
239
240 int insereElem(AVL* raiz, int elem) {
241     if (raiz == NULL) return 0;
242     return insereRec(raiz, elem);
243 }
244
245 int pesquisaRec(NO** raiz, int elem) {
246     if (*raiz == NULL) return 0;
247     if ((*raiz)->info == elem) return 1;
248     if (elem < (*raiz)->info)
249         return pesquisaRec(&(*raiz)->esq, elem);
250     else
251         return pesquisaRec(&(*raiz)->dir, elem);
252 }
253
254 int pesquisa(AVL* raiz, int elem) {
255     if (raiz == NULL) return 0;
256     if (estaVazia(raiz)) return 0;
257     return pesquisaRec(raiz, elem);
258 }
259
260 int removeRec(NO** raiz, int elem) {
261     if (*raiz == NULL) return 0;
262     int ok;
263     if ((*raiz)->info == elem) {
264         NO* aux;
265         if ((*raiz)->esq == NULL && (*raiz)->dir == NULL) {
266             // Caso 1 - NO sem filhos
267             printf("Caso 1: Liberando %d..\n", (*raiz)->info);
268             liberarNO(*raiz);
269             *raiz = NULL;
270         } else if ((*raiz)->esq == NULL) {
271             // Caso 2.1 - Possui apenas uma subarvore direita
272             printf("Caso 2.1: Liberando %d..\n", (*raiz)->info);
273             aux = *raiz;
274             *raiz = (*raiz)->dir;
275             liberarNO(aux);
276         } else if ((*raiz)->dir == NULL) {
277             // Caso 2.2 - Possui apenas uma subarvore esquerda
278             printf("Caso 2.2: Liberando %d..\n", (*raiz)->info);
279             aux = *raiz;
280             *raiz = (*raiz)->esq;
281             liberarNO(aux);
282         } else {
283             // Caso 3 - Possui as duas subarvoren (esq e dir)
284             // Duas estrategias:
285             // 3.1 - Substituir pelo NO com o MAIOR valor da subarvore esquerda
286             // 3.2 - Substituir pelo NO com o MENOR valor da subarvore direita
287             printf("Caso 3: Liberando %d..\n", (*raiz)->info);
288             // Estrategia 3.1:
289             NO* Filho = (*raiz)->esq;
290             while (Filho->dir != NULL) // Localiza o MAIOR valor da subarvore
291                 esquerda
                Filho = Filho->dir;

```

```

292         (*raiz)->info = Filho->info;
293         Filho->info = elem;
294         return removeRec(&(*raiz)->esq, elem);
295     }
296     return 1;
297 } else if (elem < (*raiz)->info) {
298     ok = removeRec(&(*raiz)->esq, elem);
299     if (ok) {
300         switch ((*raiz)->fb) {
301             case +1:
302             case 0:
303                 // Acertando alturas e Fatores de Balanceamento dos N0s
304                 afetados
305                 (*raiz)->alt = -1;
306                 (*raiz)->alt = altura(*raiz);
307                 (*raiz)->fb = FB(*raiz);
308                 break;
309             case -1:
310                 avl_AuxFD(raiz);
311                 break;
312         }
313     }
314 } else {
315     ok = removeRec(&(*raiz)->dir, elem);
316     if (ok) {
317         switch ((*raiz)->fb) {
318             case -1:
319             case 0:
320                 // Acertando alturas e Fatores de Balanceamento dos N0s
321                 afetados
322                 (*raiz)->alt = -1;
323                 (*raiz)->alt = altura(*raiz);
324                 (*raiz)->fb = FB(*raiz);
325                 break;
326             case +1:
327                 avl_AuxFE(raiz);
328                 break;
329         }
330     }
331 }
332 return ok;
333 }
334
335 int removeElem(AVL* raiz, int elem) {
336     if (pesquisa(raiz, elem) == 0) {
337         printf("Elemento inexistente!\n");
338         return 0;
339     }
340     return removeRec(raiz, elem);
341 }
342
343 void em_ordem(NO* raiz, int nivel) {
344     if (raiz != NULL) {
345         em_ordem(raiz->esq, nivel + 1);
346         // printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
347         printf("[%d, %d, %d, %d] ", raiz->info, raiz->fb, nivel, raiz->alt);
348         em_ordem(raiz->dir, nivel + 1);
349     }
350 }

```

```

349
350 void pre_ordem(N0* raiz, int nivel) {
351     if (raiz != NULL) {
352         printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
353         pre_ordem(raiz->esq, nivel + 1);
354         pre_ordem(raiz->dir, nivel + 1);
355     }
356 }
357
358 void pos_ordem(N0* raiz, int nivel) {
359     if (raiz != NULL) {
360         pos_ordem(raiz->esq, nivel + 1);
361         pos_ordem(raiz->dir, nivel + 1);
362         printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
363     }
364 }
365
366 void imprime(AVL* raiz) {
367     if (raiz == NULL) 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 tamanho(N0* raiz, int inicial) {
381     if (raiz == NULL) return 0;
382
383     int t = 1;
384     t += tamanho(raiz->esq, 0);
385     t += tamanho(raiz->dir, 0);
386     return t;
387 }
388
389 #endif

```

roteiro-09/ex01-01.c

```
1  #include "ex01-01.h"
2
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  enum {
7      EXIT = 0,
8      CREATE,
9      INSERT,
10     SEARCH,
11     REMOVE,
12     PRINT,
13     SIZE,
14     DESTROY
15 } Options;
16
17 int getOption() {
18     int option;
19
20     printf("\n=====\\n");
21     printf("(%) Criar\\n", CREATE);
22     printf("(%) Inserir elemento\\n", INSERT);
23     printf("(%) Buscar elemento\\n", SEARCH);
24     printf("(%) Remover elemento\\n", REMOVE);
25     printf("(%) Imprimir em ordem\\n", PRINT);
26     printf("(%) Tamanho\\n", SIZE);
27     printf("(%) Destruir\\n", DESTROY);
28     printf("(%) Sair\\n", EXIT);
29     printf("=====\\n");
30     printf("Operacao: ");
31
32     scanf("%d", &option);
33     printf("\\n");
34
35     return option;
36 }
37
38 int runMenu() {
39     AVL* avl = NULL;
40     int exit = 0, item;
41
42     do {
43         switch (getOption()) {
44             case CREATE:
45                 if (avl != NULL) {
46                     destroiAVL(&avl);
47                 }
48                 avl = criaAVL();
49                 break;
50
51             case INSERT:
52                 printf("Elemento a ser inserido: ");
53                 scanf("%d", &item);
54
55                 if (insereElem(avl, item)) {
56                     printf("Elemento inserido (%d)", item);
```



```

57         } else {
58             printf("Falha ao inserir (%d)", item);
59         }
60         break;
61
62     case SEARCH:
63         printf("Elemento a ser pesquisado: ");
64         scanf("%d", &item);
65
66         if (pesquisa(avl, item)) {
67             printf("Elemento presente (%d)", item);
68         } else {
69             printf("Elemento nao encontrado (%d)", item);
70         }
71         break;
72
73     case REMOVE:
74         printf("Elemento a ser removido: ");
75         scanf("%d", &item);
76
77         if (removeElem(avl, item)) {
78             printf("Elemento removido (%d)", item);
79         } else {
80             printf("Falha ao remover (%d)", item);
81         }
82         break;
83
84     case PRINT:
85         imprime(avl);
86         break;
87
88     case SIZE:
89         printf("Tamanho = %d", tamanho(*avl, 0));
90         break;
91
92     case DESTROY:
93         destroiAVL(&avl);
94         break;
95
96     case EXIT:
97         if (avl != NULL) {
98             destroiAVL(&avl);
99         }
100         printf("Programa encerrado");
101         exit = 1;
102         break;
103
104     default:
105         printf("Opcao desconhecida, tente novamente");
106     }
107     printf("\n");
108 } while (!exit);
109 }
110
111 int main() {
112     runMenu();
113     return 0;
114 }

```

```
gabriel-dp@gabriel-dp: ~/Desktop/dev/c/lab2/roteiro-09
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-09$ ./ex01-01

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Tamanho
(7) Destruir
(0) Sair
=====
Operacao: 1

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Tamanho
(7) Destruir
(0) Sair
=====
Operacao: 2

Elemento a ser inserido: 5
Elemento inserido (5)

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Tamanho
(7) Destruir
(0) Sair
=====
Operacao: 2

Elemento a ser inserido: 10
Elemento inserido (10)

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Tamanho
(7) Destruir
(0) Sair
=====
Operacao: 2

Elemento a ser inserido: 15
Rotacao Simples a ESQUERDA!
Calculando FB do (10)..
Calculando FB do (5)..
Falha ao inserir (15)

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Tamanho
(7) Destruir
(0) Sair
=====
Operacao: 5

Em Ordem: [INFO, FB, NIVEL, altura]
[5, 0, 1, 1] [10, 0, 0, 2] [15, 0, 1, 1]

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Tamanho
(7) Destruir
(0) Sair
=====
Operacao: 6

Tamanho = 3

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Tamanho
(7) Destruir
(0) Sair
=====
Operacao: 0

Programa encerrado
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-09$
```

roteiro-09/ex01-02.h

```
1  #ifndef AVL_H
2  #define AVL_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7
8  #define MAIOR(a, b) ((a > b) ? (a) : (b))
9
10 typedef struct {
11     char nome[50];
12     int contratacao;
13     double salario;
14 } Funcionario;
15
16 typedef struct NO {
17     int fb, alt;
18     Funcionario info;
19     struct NO* esq;
20     struct NO* dir;
21 } NO;
22
23 typedef struct NO* AVL;
24
25 NO* alocarNO() {
26     return (NO*)malloc(sizeof(NO));
27 }
28
29 void liberarNO(NO* q) {
30     free(q);
31 }
32
33 AVL* criaAVL() {
34     AVL* raiz = (AVL*)malloc(sizeof(AVL));
35     if (raiz != NULL)
36         *raiz = NULL;
37     return raiz;
38 }
39
40 void destroiRec(NO* no) {
41     if (no == NULL) return;
42     destroiRec(no->esq);
43     destroiRec(no->dir);
44     liberarNO(no);
45     no = NULL;
46 }
47
48 void destroiAVL(AVL** raiz) {
49     if (*raiz != NULL) {
50         destroiRec(**raiz);
51         free(*raiz);
52         *raiz = NULL;
53     }
54 }
55
56 int estaVazia(AVL* raiz) {
```

```

57     if (raiz == NULL) return 0;
58     return (*raiz == NULL);
59 }
60
61 // Calcula FB
62 int altura(NO* raiz) {
63     if (raiz == NULL) return 0;
64     if (raiz->alt > 0)
65         return raiz->alt;
66     else {
67         // printf("Calculando altura do (%d)..\\n", raiz->info);
68         return MAIOR(altura(raiz->esq), altura(raiz->dir)) + 1;
69     }
70 }
71
72 int FB(NO* raiz) {
73     if (raiz == NULL) return 0;
74     return altura(raiz->esq) - altura(raiz->dir);
75 }
76
77 // Funcoes de Rotacao Simples
78 void avl_RotDir(NO** raiz) {
79     printf("Rotacao Simples a DIREITA!\\n");
80     NO* aux;
81     aux = (*raiz)->esq;
82     (*raiz)->esq = aux->dir;
83     aux->dir = *raiz;
84
85     (*raiz)->alt = aux->alt = -1;
86     aux->alt = altura(aux);
87     (*raiz)->alt = altura(*raiz);
88     aux->fb = FB(aux);
89     (*raiz)->fb = FB(*raiz);
90
91     *raiz = aux;
92 }
93
94 void avl_RotEsq(NO** raiz) {
95     printf("Rotacao Simples a ESQUERDA!\\n");
96     NO* aux;
97     aux = (*raiz)->dir;
98     (*raiz)->dir = aux->esq;
99     aux->esq = *raiz;
100
101     (*raiz)->alt = aux->alt = -1;
102     aux->alt = altura(aux);
103     (*raiz)->alt = altura(*raiz);
104     aux->fb = FB(aux);
105     (*raiz)->fb = FB(*raiz);
106
107     *raiz = aux;
108 }
109
110 void avl_RotEsqDir(NO** raiz) {
111     printf("Rotacao Dupla ESQUERDA-DIREITA!\\n");
112     NO* fe; // filho esquerdo
113     NO* ffd; // filho filho direito
114
115     fe = (*raiz)->esq;

```

```

116     ffd = fe->dir;
117
118     fe->dir = ffd->esq;
119     ffd->esq = fe;
120
121     (*raiz)->esq = ffd->dir;
122     ffd->dir = *raiz;
123
124     (*raiz)->alt = fe->alt = ffd->alt = -1;
125     fe->alt = altura(fe);
126     ffd->alt = altura(ffd);
127     (*raiz)->alt = altura(*raiz);
128     fe->fb = FB(fe);
129     ffd->fb = FB(ffd);
130     (*raiz)->fb = FB(*raiz);
131
132     *raiz = ffd;
133 }
134
135 void avl_RotDirEsq(NO** raiz) {
136     printf("Rotacao Dupla DIREITA-ESQUERDA!\n");
137     NO* fd;    // filho direito
138     NO* ffe;   // filho filho esquerdo
139
140     fd = (*raiz)->dir;
141     ffe = fd->esq;
142
143     fd->esq = ffe->dir;
144     ffe->dir = fd;
145
146     (*raiz)->dir = ffe->esq;
147     ffe->esq = *raiz;
148
149     (*raiz)->alt = fd->alt = ffe->alt = -1;
150     fd->alt = altura(fd);
151     ffe->alt = altura(fffe);
152     (*raiz)->alt = altura(*raiz);
153     fd->fb = FB(fd);
154     ffe->fb = FB(fffe);
155     (*raiz)->fb = FB(*raiz);
156
157     *raiz = ffe;
158 }
159
160 void avl_RotEsqDir2(NO** raiz) {
161     printf("Rotacao Dupla 2 ESQUERDA-DIREITA!\n");
162     avl_RotEsq(&(*raiz)->esq);
163     avl_RotDir(raiz);
164 }
165
166 void avl_RotDirEsq2(NO** raiz) {
167     printf("Rotacao Dupla 2 DIREITA-ESQUERDA!\n");
168     avl_RotDir(&(*raiz)->dir);
169     avl_RotEsq(raiz);
170 }
171
172 void avl_AuxFE(NO** raiz) {
173     NO* fe;
174     fe = (*raiz)->esq;

```

```

175     if (fe->fb == +1) /* Sinais iguais e positivo*/
176         avl_RotDir(raiz);
177     else /* Sinais diferentes*/
178         avl_RotEsqDir(raiz);
179 }
180
181 void avl_AuxFD(NO** raiz) {
182     NO* fd;
183     fd = (*raiz)->dir;
184     if (fd->fb == -1) /* Sinais iguais e negativos*/
185         avl_RotEsq(raiz);
186     else /* Sinais diferentes*/
187         avl_RotDirEsq(raiz);
188 }
189
190 int insereRec(NO** raiz, Funcionario elem) {
191     int ok; // Controle para as chamadas recursivas
192     if (*raiz == NULL) {
193         NO* novo = alocarNO();
194         if (novo == NULL) return 0;
195         novo->info = elem;
196         novo->fb = 0, novo->alt = 1;
197         novo->esq = NULL;
198         novo->dir = NULL;
199         *raiz = novo;
200         return 1;
201     } else {
202         if ((*raiz)->info.salario == elem.salario) {
203             printf("Elemento Existente!\n");
204             ok = 0;
205         }
206         if (elem.salario < (*raiz)->info.salario) {
207             ok = insereRec(&(*raiz)->esq, elem);
208             if (ok) {
209                 switch ((*raiz)->fb) {
210                     case -1:
211                         (*raiz)->fb = 0;
212                         ok = 0;
213                         break;
214                     case 0:
215                         (*raiz)->fb = +1;
216                         (*raiz)->alt++;
217                         break;
218                     case +1:
219                         avl_AuxFE(raiz);
220                         ok = 0;
221                         break;
222                 }
223             }
224         } else if (elem.salario > (*raiz)->info.salario) {
225             ok = insereRec(&(*raiz)->dir, elem);
226             if (ok) {
227                 switch ((*raiz)->fb) {
228                     case +1:
229                         (*raiz)->fb = 0;
230                         ok = 0;
231                         break;
232                     case 0:
233                         (*raiz)->fb = -1;

```

```

234             (*raiz)->alt++;
235             break;
236         case -1:
237             avl_AuxFD(raiz);
238             ok = 0;
239             break;
240     }
241 }
242 }
243 }
244 return ok;
245 }
246
247 int insereElem(AVL* raiz, Funcionario elem) {
248     if (raiz == NULL) return 0;
249     return insereRec(raiz, elem);
250 }
251
252 int pesquisaRec(NO** raiz, Funcionario elem) {
253     if (*raiz == NULL) return 0;
254     if ((*raiz)->info.salario == elem.salario) return 1;
255     if (elem.salario < (*raiz)->info.salario)
256         return pesquisaRec(&(*raiz)->esq, elem);
257     else
258         return pesquisaRec(&(*raiz)->dir, elem);
259 }
260
261 int pesquisa(AVL* raiz, Funcionario elem) {
262     if (raiz == NULL) return 0;
263     if (estaVazia(raiz)) return 0;
264     return pesquisaRec(raiz, elem);
265 }
266
267 int removeRec(NO** raiz, Funcionario elem) {
268     if (*raiz == NULL) return 0;
269     int ok;
270     if ((*raiz)->info.salario == elem.salario) {
271         NO* aux;
272         if ((*raiz)->esq == NULL && (*raiz)->dir == NULL) {
273             // Caso 1 - NO sem filhos
274             liberarNO(*raiz);
275             *raiz = NULL;
276         } else if ((*raiz)->esq == NULL) {
277             // Caso 2.1 - Possui apenas uma subarvore direita
278             aux = *raiz;
279             *raiz = (*raiz)->dir;
280             liberarNO(aux);
281         } else if ((*raiz)->dir == NULL) {
282             // Caso 2.2 - Possui apenas uma subarvore esquerda
283             aux = *raiz;
284             *raiz = (*raiz)->esq;
285             liberarNO(aux);
286         } else {
287             // Caso 3 - Possui as duas subarvoretas (esq e dir)
288             // Duas estrategias:
289             // 3.1 - Substituir pelo NO com o MAIOR valor da subarvore esquerda
290             // 3.2 - Substituir pelo NO com o MENOR valor da subarvore direita
291             // Estrategia 3.1:
292             NO* Filho = (*raiz)->esq;

```

```

293     while (Filho->dir != NULL) // Localiza o MAIOR valor da subarvore
esquerda
294         Filho = Filho->dir;
295         (*raiz)->info = Filho->info;
296         Filho->info = elem;
297         return removeRec(&(*raiz)->esq, elem);
298     }
299     return 1;
300 } else if (elem.salario < (*raiz)->info.salario) {
301     ok = removeRec(&(*raiz)->esq, elem);
302     if (ok) {
303         switch ((*raiz)->fb) {
304             case +1:
305             case 0:
306                 // Acertando alturas e Fatores de Balanceamento dos N0s
afetados
307                 (*raiz)->alt = -1;
308                 (*raiz)->alt = altura(*raiz);
309                 (*raiz)->fb = FB(*raiz);
310                 break;
311             case -1:
312                 avl_AuxFD(raiz);
313                 break;
314         }
315     }
316 } else {
317     ok = removeRec(&(*raiz)->dir, elem);
318     if (ok) {
319         switch ((*raiz)->fb) {
320             case -1:
321             case 0:
322                 // Acertando alturas e Fatores de Balanceamento dos N0s
afetados
323                 (*raiz)->alt = -1;
324                 (*raiz)->alt = altura(*raiz);
325                 (*raiz)->fb = FB(*raiz);
326                 break;
327             case +1:
328                 avl_AuxFE(raiz);
329                 break;
330         }
331     }
332 }
333 return ok;
334 }
335
336 int removeElem(AVL* raiz, Funcionario elem) {
337     if (pesquisa(raiz, elem) == 0) {
338         printf("Elemento inexistente!\n");
339         return 0;
340     }
341     return removeRec(raiz, elem);
342 }
343
344 void em_ordem(NO* raiz, int nivel) {
345     if (raiz != NULL) {
346         em_ordem(raiz->esq, nivel + 1);
347         // printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
348         printf("[%s, %d, %d, %d] ", raiz->info.nome, raiz->fb, nivel, raiz->
alt);

```



```

349     em_ordem(raiz->dir, nivel + 1);
350 }
351 }
352
353 void imprime(AVL* raiz) {
354     if (raiz == NULL) return;
355     if (estaVazia(raiz)) {
356         printf("Arvore Vazia!\n");
357         return;
358     }
359     // printf("\nEm Ordem: [INFO, FB, NIVEL]\n");
360     printf("\nEm Ordem: [INFO, FB, NIVEL, altura]\n");
361     em_ordem(*raiz, 0);
362     // printf("\nPre Ordem: "); pre_ordem(*raiz, 0);
363     // printf("\nPos Ordem: "); pos_ordem(*raiz, 0);
364     printf("\n");
365 }
366
367 Funcionario* melhor(NO* raiz) {
368     if (raiz != NULL) {
369         Funcionario *dir = melhor(raiz->dir), *esq = melhor(raiz->esq), *maior;
370         if (dir == NULL || (esq != NULL && esq->salario > dir->salario)) {
371             maior = esq;
372         } else {
373             maior = dir;
374         }
375         if (maior == NULL || raiz->info.salario >= maior->salario) {
376             return &(raiz->info);
377         } else {
378             return maior;
379         }
380     }
381     return NULL;
382 }
383
384 Funcionario* pior(NO* raiz) {
385     if (raiz != NULL) {
386         Funcionario *dir = melhor(raiz->dir), *esq = melhor(raiz->esq), *maior;
387         if (dir == NULL || (esq != NULL && esq->salario < dir->salario)) {
388             maior = esq;
389         } else {
390             maior = dir;
391         }
392         if (maior == NULL || raiz->info.salario <= maior->salario) {
393             return &(raiz->info);
394         } else {
395             return maior;
396         }
397     }
398     return NULL;
399 }
400
401 void fixString(char str[50]) {
402     int length = strlen(str);
403     if (str[length - 1] == '\n') str[length - 1] = '\0';
404 }
405
406 #endif

```

roteiro-09/ex01-02.c

```
1  #include "ex01-02.h"
2
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  enum {
7      EXIT = 0,
8      CREATE,
9      INSERT,
10     SEARCH,
11     REMOVE,
12     PRINT,
13     PRINT_BEST,
14     PRINT_WORST,
15     DESTROY
16 } Options;
17
18 int getOption() {
19     int option;
20
21     printf("\n=====\\n");
22     printf("(%) Criar\\n", CREATE);
23     printf("(%) Inserir elemento\\n", INSERT);
24     printf("(%) Buscar elemento\\n", SEARCH);
25     printf("(%) Remover elemento\\n", REMOVE);
26     printf("(%) Imprimir em ordem\\n", PRINT);
27     printf("(%) Imprimir funcionario com maior salario\\n", PRINT_BEST);
28     printf("(%) Imprimir funcionario com menor salario\\n", PRINT_WORST);
29     printf("(%) Destruir\\n", DESTROY);
30     printf("(%) Sair\\n", EXIT);
31     printf("=====\\n");
32     printf("Operacao: ");
33
34     scanf("%d", &option);
35     printf("\\n");
36
37     return option;
38 }
39
40 int runMenu() {
41     AVL* avl = NULL;
42     int exit = 0;
43
44     Funcionario item, *itemp;
45     char nome[50];
46     int contratacao;
47     double salario;
48
49     do {
50         switch (getOption()) {
51             case CREATE:
52                 if (avl != NULL) {
53                     destroiAVL(&avl);
54                 }
55                 avl = criaAVL();
56                 break;
```

```

57
58     case INSERT:
59         printf("Funcionario a ser inserido\n");
60         printf("Nome (Max 50 caracteres): ");
61         setbuf(stdin, NULL);
62         fgets(item.nome, 50, stdin);
63         fixString(item.nome);
64         setbuf(stdin, NULL);
65         printf("Ano de contratacao: ");
66         scanf("%d", &item.contratacao);
67         printf("Salario: ");
68         scanf("%lf", &item.salario);
69
70         if (insereElem(avl, item)) {
71             printf("Funcionario inserido (%s)", item.nome);
72         } else {
73             printf("Falha ao inserir (%s)", item.nome);
74         }
75         break;
76
77     case SEARCH:
78         printf("Salario para buscar: ");
79         scanf("%lf", &item.salario);
80
81         if (pesquisa(avl, item)) {
82             printf("Funcionario presente (%.2lf)", item.salario);
83         } else {
84             printf("Funcionario nao encontrado (%.2lf)", item.salario);
85         }
86         break;
87
88     case REMOVE:
89         printf("Nome para remover (Max 50 caracteres): ");
90         setbuf(stdin, NULL);
91         fgets(item.nome, 50, stdin);
92         fixString(item.nome);
93         setbuf(stdin, NULL);
94
95         if (removeElem(avl, item)) {
96             printf("Funcionario removido (%s)", item.nome);
97         } else {
98             printf("Falha ao remover (%s)", item.nome);
99         }
100         break;
101
102     case PRINT:
103         imprime(avl);
104         break;
105
106     case PRINT_BEST:
107         itemp = melhor(*avl);
108         printf("Funcionario com o maior salario\n");
109         printf("Nome = %s\nContratacao = %d\nSalario = %.2lf", itemp->
nome, itemp->contratacao, itemp->salario);
110         break;
111
112     case PRINT_WORST:
113         itemp = pior(*avl);
114         printf("Funcionario com o menor salario\n");

```

```

115         printf("Nome = %s\nContratacao = %d\nSalario = %.2lf", itemp->
nome, itemp->contratacao, itemp->salario);
116         break;
117
118     case DESTROY:
119         destroiAVL(&avl);
120         break;
121
122     case EXIT:
123         if (avl != NULL) {
124             destroiAVL(&avl);
125         }
126         printf("Programa encerrado");
127         exit = 1;
128         break;
129
130     default:
131         printf("Opcao desconhecida, tente novamente");
132     }
133     printf("\n");
134 } while (!exit);
135 }
136
137 int main() {
138     runMenu();
139     return 0;
140 }

```

```
gabriel-dp@gabriel-dp: ~/Desktop/dev/c/lab2/roteiro-09
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-09$ ./ex01-02

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir funcionario com maior salario
(7) Imprimir funcionario com menor salario
(8) Destruir
(0) Sair
=====
Operacao: 1

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir funcionario com maior salario
(7) Imprimir funcionario com menor salario
(8) Destruir
(0) Sair
=====
Operacao: 2

Funcionario a ser inserido
Nome (Max 50 caracteres): Gabriel
Ano de contratacao: 2022
Salario: 320
Funcionario inserido (Gabriel)

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir funcionario com maior salario
(7) Imprimir funcionario com menor salario
(8) Destruir
(0) Sair
=====
Operacao: 2

Funcionario a ser inserido
Nome (Max 50 caracteres): Prenassi
Ano de contratacao: 2023
Salario: 700
Funcionario inserido (Prenassi)

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir funcionario com maior salario
(7) Imprimir funcionario com menor salario
(8) Destruir
(0) Sair
=====
Operacao: 2

Funcionario a ser inserido
Nome (Max 50 caracteres): Gabriel de Paula
Ano de contratacao: 2023
Salario: 800
Rotacao Simples a ESQUERDA!
Falha ao inserir (Gabriel de Paula)

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir funcionario com maior salario
(7) Imprimir funcionario com menor salario
(8) Destruir
(0) Sair
=====
Operacao: 3

Salario para buscar: 800
Funcionario presente (800.00)

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir funcionario com maior salario
(7) Imprimir funcionario com menor salario
(8) Destruir
(0) Sair
=====
Operacao: 5

Em Ordem: [INFO, FB, NIVEL, altura]
[Gabriel, 0, 1, 1] [Prenassi, 0, 0, 2] [Gabriel de Paula, 0, 1, 1]

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir funcionario com maior salario
(7) Imprimir funcionario com menor salario
(8) Destruir
(0) Sair
=====
Operacao: 6

Funcionario com o maior salario
Nome = Gabriel de Paula
Contratacao = 2023
Salario = 800.00

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir funcionario com maior salario
(7) Imprimir funcionario com menor salario
(8) Destruir
(0) Sair
=====
Operacao: 7

Funcionario com o menor salario
Nome = Gabriel
Contratacao = 2022
Salario = 320.00

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir funcionario com maior salario
(7) Imprimir funcionario com menor salario
(8) Destruir
(0) Sair
=====
Operacao: 0

Programa encerrado
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-09$
```

roteiro-09/ex01-02.txt

A complexidade da operação de encontrar os funcionários com maior e menor salário é $O(n)$, tendo em vista que as AVLs possuem a inserção mais custosa para que a pesquisa seja mais eficiente do que em árvores binárias comuns.