



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

Roteiro 9

Rodrigo José Zonzin Esteves
212050002

1 Árvore AVL

1.1 Reimplantação

Código

```
1 #ifndef AVL_H
2 #define AVL_H
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 #define MAIOR(a, b) ((a > b) ? (a) :
    (b))
7
8 typedef struct NO{
9     int info, fb, alt;
10     struct NO* esq;
11     struct NO* dir;
12 }NO;
13
14 typedef struct NO* AVL;
15
16 NO* alocarNO();
17 void liberarNO(NO* q);
18 AVL* criaAVL();
19 void destroiRec(NO* no);
20 void destroiAVL(AVL* raiz);
21 int estaVazia(AVL* raiz);
22 int altura(NO* raiz);
23 int FB(NO* raiz);
24 void avl_RotDir(NO** raiz);
25 void avl_RotEsq(NO** raiz);
26 void avl_RotEsqDir(NO** raiz);
27 void avl_RotDirEsq(NO** raiz);
28 void avl_RotEsqDir2(NO** raiz);
29 void avl_RotDirEsq2(NO** raiz);
30 void avl_AuxFE(NO **raiz);
31 void avl_AuxFD(NO **raiz);
32 int insereRec(NO** raiz, int elem);
33 int insereElem(AVL* raiz, int elem);
34 int pesquisaRec(NO** raiz, int elem);
35 int pesquisa(AVL* raiz, int elem);
36 int removeRec(NO** raiz, int elem);
37 int removeElem(AVL* raiz, int elem);
38 void em_ordem(NO* raiz, int nivel);
39 void pre_ordem(NO* raiz, int nivel);
40 void pos_ordem(NO* raiz, int nivel);
41 void imprime(AVL* raiz);
42 void aguardaLimpa();
43 void contador(NO* raiz, int nivel, int
    *cont);
44
45 #endif
```

codigos/ex11/ex11.h

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "ex11.h"
4
5 NO* alocarNO(){
6     return (NO*) malloc (sizeof(NO));
7 }
8
9 void liberarNO(NO* q){
10     free(q);
11 }
12
13 AVL* criaAVL(){
14     AVL* raiz = (AVL*) malloc
        (sizeof(AVL));
15     if(raiz != NULL)
```

```

16     *raiz = NULL;
17     return raiz;
18 }
19
20 void destroiRec(NO* no){
21     if(no == NULL) return;
22     destroiRec(no->esq);
23     destroiRec(no->dir);
24     liberarNO(no);
25     no = NULL;
26 }
27
28 void destroiAVL(AVL* raiz){
29     if(raiz != NULL){
30         destroiRec(*raiz);
31         free(raiz);
32     }
33 }
34
35 int estaVazia(AVL* raiz){
36     if(raiz == NULL) return 0;
37     return (*raiz == NULL);
38 }
39
40 //Calcula FB
41 int altura(NO* raiz){
42     if(raiz == NULL) return 0;
43     if(raiz->alt > 0)
44         return raiz->alt;
45     else{
46         //printf("Calculando altura do
47             (%d)..\\n", raiz->info);
48         return MAIOR(altura(raiz->esq),
49             altura(raiz->dir)) + 1;
50     }
51 }
52
53 int FB(NO* raiz){
54     if(raiz == NULL) return 0;
55     printf("Calculando FB do (%d)..\\n",
56         raiz->info);
57     return altura(raiz->esq) -
58         altura(raiz->dir);
59 }
60
61 //Funcoes de Rotacao Simples
62 void avl_RotDir(NO** raiz){
63     printf("Rotacao Simples a
64         DIREITA!\\n");
65     NO *aux;
66     aux = (*raiz)->esq;
67     (*raiz)->esq = aux->dir;
68     aux->dir = *raiz;
69
70     //Acertando alturas e FB
71     //dos NOs afetados
72     (*raiz)->alt = aux->alt = -1;
73     aux->alt = altura(aux);
74     (*raiz)->alt = altura(*raiz);
75
76     *raiz = NULL;
77     return raiz;
78 }
79
80 void avl_RotEsq(NO** raiz){
81     printf("Rotacao Simples a
82         ESQUERDA!\\n");
83     NO *aux;
84     aux = (*raiz)->dir;
85     (*raiz)->dir = aux->esq;
86     aux->esq = *raiz;
87
88     //Acertando alturas e Fatores de
89     //Balanceamento dos NOs afetados
90     (*raiz)->alt = aux->alt = -1;
91     aux->alt = altura(aux);
92     (*raiz)->alt = altura(*raiz);
93
94     *raiz = NULL;
95     return raiz;
96 }
97
98 //Funcoes de Rotacao Dupla
99 void avl_RotEsqDir(NO** raiz){
100     printf("Rotacao Dupla
101         ESQUERDA-DIREITA!\\n");
102     NO *fe; //filho esquerdo
103     NO *ffd; //filho filho direito
104
105     fe = (*raiz)->esq;
106     ffd = fe->dir;
107
108     fe->dir = ffd->esq;
109     ffd->esq = fe;
110
111     (*raiz)->esq = ffd->dir;
112     ffd->dir = *raiz;
113
114     //Acertando alturas e Fatores de
115     //Balanceamento dos NOs afetados
116     (*raiz)->alt = fe->alt = ffd->alt =
117         -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

```

```

124     DIREITA-ESQUERDA!\n");
125     NO* fd; //filho direito
126     NO* ffe; //filho filho esquerdo
127     fd = (*raiz)->dir;
128     ffe = fd->esq;
129
130     fd->esq = ffe->dir;
131     ffe->dir = fd;
132
133     (*raiz)->dir = ffe->esq;
134     ffe->esq = *raiz;
135
136     //Acertando alturas e Fatores de
137     //Balanceamento dos NOs afetados
138     (*raiz)->alt = fd->alt = ffe->alt = 187
139     -1;
140     fd->alt = altura(fd);
141     ffe->alt = altura(fffe);
142     (*raiz)->alt = altura(*raiz);
143     fd->fb = FB(fd);
144     ffe->fb = FB(fffe);
145     (*raiz)->fb = FB(*raiz);
146
147     *raiz = ffe;
148 }
149 void avl_RotEsqDir2(NO** raiz){
150     printf("Rotacao Dupla 2
151     ESQUERDA-DIREITA!\n");
152     avl_RotEsq(&(*raiz)->esq);
153     avl_RotDir(raiz);
154 }
155 void avl_RotDirEsq2(NO** raiz){
156     printf("Rotacao Dupla 2
157     DIREITA-ESQUERDA!\n");
158     avl_RotDir(&(*raiz)->dir);
159     avl_RotEsq(raiz);
160 }
161 //Funcoes Auxiliares referentes a cada
162 //filho
163 void avl_AuxFE(NO **raiz){
164     NO* fe;
165     fe = (*raiz)->esq;
166     if(fe->fb == +1) /* Sinais iguais e
167     positivo*/
168     avl_RotDir(raiz);
169     else /* Sinais diferentes*/
170     avl_RotEsqDir(raiz);
171 }
172 void avl_AuxFD(NO **raiz){
173     NO* fd;
174     fd = (*raiz)->dir;
175     if(fd->fb == -1) /* Sinais iguais e
176     negativos*/
177     avl_RotEsq(raiz);
178     else /* Sinais diferentes*/
179     avl_RotDirEsq(raiz);
180 }
181 int insereRec(NO** raiz, int elem){
182     int ok; //Controle para as chamadas
183     recursivas
184     if(*raiz == NULL){
185         NO* novo = alocarNO();
186         if(novo == NULL) return 0;
187         novo->info = elem; novo->fb =
188         0, novo->alt = 1;
189         novo->esq = NULL; novo->dir =
190         NULL;
191         *raiz = novo; return 1;
192     }else{
193         if((*raiz)->info == elem){
194             printf("Elemento
195             Existente!\n"); ok = 0;
196         }
197         if(elem < (*raiz)->info){
198             ok =
199             insereRec(&(*raiz)->esq,
200             elem);
201             if(ok){
202                 switch((*raiz)->fb){
203                     case -1:
204                         (*raiz)->fb =
205                         0; ok = 0;
206                         break;
207                     case 0:
208                         (*raiz)->fb =
209                         +1;
210                         (*raiz)->alt++;
211                         break;
212                     case +1:
213                         avl_AuxFE(raiz);
214                         ok = 0;
215                         break;
216                 }
217             }
218         }
219         else if(elem > (*raiz)->info){
220             ok =
221             insereRec(&(*raiz)->dir,
222             elem);
223             if(ok){
224                 switch((*raiz)->fb){
225                     case +1:
226                         (*raiz)->fb =
227                         0; ok = 0;
228                         break;
229                     case 0:
230                         (*raiz)->fb =
231                         -1;
232                         (*raiz)->alt++;
233                         break;
234                     case -1:

```

```

216             avl_AuxFD(raiz) 263
                ok = 0;        264
                break;         265
217         }                  266
218     }                        267
219 }                            268
220 }                            269
221 return ok;
222 }
223
224 int insereElem(AVL* raiz, int elem){ 270
225     if(raiz == NULL) return 0;
226     return insereRec(raiz, elem);
227 } 271
228
229 int pesquisaRec(NO** raiz, int elem){ 272
230     if(*raiz == NULL) return 0; 273
231     if((*raiz)->info == elem) return 1; 274
232     if(elem < (*raiz)->info)
233         return
                pesquisaRec(&(*raiz)->esq,
                elem); 275
234     else 276
235         return 277
                pesquisaRec(&(*raiz)->dir, 278
                elem);
236 }
237 279
238 int pesquisa(AVL* raiz, int elem){ 280
239     if(raiz == NULL) return 0; 281
240     if(estaVazia(raiz)) return 0; 282
241     return pesquisaRec(raiz, elem);
242 } 283
243 284
244 int removeRec(NO** raiz, int elem){ 285
245     if(*raiz == NULL) return 0; 286
246     int ok; 287
247     if((*raiz)->info == elem){
248         NO* aux;
249         if((*raiz)->esq == NULL &&
                (*raiz)->dir == NULL){ 288
250             //Caso 1 - NO sem filhos 289
251             printf("Caso 1: Liberando
                %d..\n", (*raiz)->info); 290
252             liberarNO(*raiz);
253             *raiz = NULL; 291
254         }else if((*raiz)->esq == NULL){ 292
255             //Caso 2.1 - Possui apenas 293
                uma subarvore direita
256             printf("Caso 2.1: Liberando 294
                %d..\n", (*raiz)->info); 295
257             aux = *raiz; 296
258             *raiz = (*raiz)->dir; 297
259             liberarNO(aux); 298
260         }else if((*raiz)->dir == NULL){
261             //Caso 2.2 - Possui apenas 299
                uma subarvore esquerda 300
262             printf("Caso 2.2: Liberando 301
                %d..\n", (*raiz)->info); 302
                aux = *raiz;
                *raiz = (*raiz)->esq;
                liberarNO(aux);
        }else{
            //Caso 3 - Possui as duas
            subarvores (esq e dir)
            //Duas estrategias:
            //3.1 - Substituir pelo NO
            com o MAIOR valor da
            subarvore esquerda
            //3.2 - Substituir pelo NO
            com o MENOR valor da
            subarvore direita
            printf("Caso 3: Liberando
            %d..\n", (*raiz)->info);
            //Estrategia 3.1:
            NO* Filho = (*raiz)->esq;
            while(Filho->dir !=
                NULL) //Localiza o MAIOR
                valor da subarvore
                esquerda
                Filho = Filho->dir;
            (*raiz)->info = Filho->info;
            Filho->info = elem;
            return
                removeRec(&(*raiz)->esq,
                elem);
        }
        return 1;
    }else if(elem < (*raiz)->info){
        ok = removeRec(&(*raiz)->esq,
            elem);
        if(ok){
            switch((*raiz)->fb){
                case +1:
                case 0:
                    //Acertando alturas
                    e Fatores de
                    Balanceamento
                    dos NOs afetados
                    (*raiz)->alt = -1;
                    (*raiz)->alt =
                        altura(*raiz);
                    (*raiz)->fb =
                        FB(*raiz);
                    break;
                case -1:
                    avl_AuxFD(raiz);
                    break;
            }
        }
    }
    }else{
        ok = removeRec(&(*raiz)->dir,
            elem);
        if(ok){
            switch((*raiz)->fb){
                case -1:
                case 0:

```

```

303          //Acertando alturas e Fatores de Balanceamento dos NOs afetados
304          (*raiz)->alt = -1;
305          (*raiz)->alt = altura(*raiz);
306          (*raiz)->fb = FB(*raiz);
307          break;
308          case +1:
309              avl_AuxFE(raiz);
310              break;
311      }
312  }
313  return ok;
314 }
315
316 int removeElem(AVL* raiz, int elem){
317     if(pesquisa(raiz, elem) == 0){
318         printf("Elemento inexistente!\n");
319         return 0;
320     }
321     return removeRec(raiz, elem);
322 }
323
324 void em_ordem(NO* raiz, int nivel){
325     if(raiz != NULL){
326         em_ordem(raiz->esq, nivel+1);
327         //printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
328         printf("[%d, %d, %d, %d] ", raiz->info, raiz->fb, nivel, raiz->alt);
329         em_ordem(raiz->dir, nivel+1);
330     }
331 }
332
333 void pre_ordem(NO* raiz, int nivel){
334     if(raiz != NULL){
335         printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
336         pre_ordem(raiz->esq, nivel+1);
337         pre_ordem(raiz->dir, nivel+1);
338     }
339 }
340
341 void pos_ordem(NO* raiz, int nivel){
342     if(raiz != NULL){
343         pos_ordem(raiz->esq, nivel+1);
344         pos_ordem(raiz->dir, nivel+1);
345         printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
346     }
347 }
348
349 void imprime(AVL* raiz){
350     if(raiz == NULL) 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: ");
359     pre_ordem(*raiz, 0);
360     //printf("\nPos Ordem: ");
361     pos_ordem(*raiz, 0);
362     printf("\n");
363 }
364 /* Funcao para esperar resposta do usuario e depois limpar*/
365 void aguardaLimpa(){
366     getchar();
367     printf("\n\nAperte qualquer tecla para continuar\n");
368     getchar();
369     system("clear");
370 }
371
372 void contador(NO* raiz, int nivel, int *cont){
373     if(raiz != NULL){
374         contador(raiz->esq, nivel+1, cont);
375         (*cont)++;
376         contador(raiz->dir, nivel+1, cont);
377     }
378 }

```

codigos/ex11/ex11.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "ex11.h"
4
5 #include <stdio.h>
6 #include <stdlib.h>
7
8 // Assuming AVL and other function declarations here
9
10 int main() {
11     int *cont =

```

```

    (int*)malloc(sizeof(int));
12 int escolha, elem, busca;
13 AVL *avl;
14 printf("0 que deseja fazer?\n1 -
    Criar AVL\n2 - Inserir um
    elemento\n"
15 "3 - Buscar um elemento\n4 -
    Remover um elemento\n5 -
    Imprimir a AVL em ordem"
16 "\n6 - Mostrar a quantidade de
    nos na AVL\n7 - Destruir a
    AVL\n8 - Sair\n");
17
18 scanf("%d", &escolha);
19
20 while (escolha != 8) {
21     if (escolha == 1) {
22         avl = criaAVL();
23         printf("Arvore criada!");
24         aguardaLimpa();
25     } else if (escolha == 2) {
26         printf("Digite o elemento
            que deseja inserir: ");
27         scanf("%d", &elem);
28         insereElem(avl, elem);
29         aguardaLimpa();
30     } else if (escolha == 3) {
31         printf("Digite o elemento
            que deseja consultar: ");
32         scanf("%d", &elem);
33         busca = pesquisa(avl, elem);
34         if (busca == 1)
35             printf("0 numero esta
                na arvore!");
36         else
37             printf("0 numero nao
                esta na arvore!");
38         aguardaLimpa();
39     } else if (escolha == 4) {
40         printf("Digite o elemento
            que deseja remover: ");
41         scanf("%d", &elem);
42         removeElem(avl, elem);
43         aguardaLimpa();
44     } else if (escolha == 5) {
45         imprime(avl);
46         aguardaLimpa();
47     } else if (escolha == 6) {
48         *cont = 0;
49         contador(*avl, 0, cont);
50         printf("0 numero de nos eh:
            %d!", *cont);
51         aguardaLimpa();
52     } else if (escolha == 7) {
53         destroiAVL(avl);
54         printf("Arvore destruida!");
55         aguardaLimpa();
56     } else {
57         printf("\nErro!\n\n");
58     }
59
60     printf("0 que deseja fazer?\n1
        - Criar AVL\n2 - Inserir um
        elemento\n"
        "3 - Buscar um elemento\n4
        - Remover um elemento\n5
        - Imprimir a AVL em
        ordem"
        "\n6 - Quantidade de nos na
        AVL\n7 - Destruir a
        AVL\n8 - Sair\n");
61
62     scanf("%d", &escolha);
63
64     }
65
66     free(cont); // Free the allocated
67                 memory
68
69     return 0;
70 }

```

codigos/ex11/main.c

```

1 all: ex11.o
2 gcc ex11.o main.c -o main
3
4 ex11.o: ex11.h ex11.c
5 gcc -c ex11.c

```

```

6
7 clean:
8 rm -f ex11.o main

```

codigos/ex11/Makefile

1.2 TAD Novo

Código

```
1 #ifndef AVL_H
2 #define AVL_H
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 #define MAIOR(a, b) ((a > b) ? (a) :
    (b))
7
8 typedef struct funcionario {
9     char nome[30];
10    double salario;
11    int anoContratacao;
12 } Funcionario;
13
14 typedef struct no{
15     Funcionario info;
16     int fb, alt;
17     struct NO* esq;
18     struct NO* dir;
19 }NO;
20
21 typedef struct no* AVL;
22
23 NO* alocarFuncionario();
24 void liberarFuncionario(NO* q);
25 AVL* criaAVL();
26 void destroiRec(NO* NO);
27 void destroiAVL(AVL* raiz);
28 int estaVazia(AVL* raiz);
29 int altura(NO* raiz);
30 int FB(NO* raiz);
31 void avl_RotDir(NO** raiz);
32 void avl_RotEsq(NO** raiz);
33 void avl_RotEsqDir(NO** raiz);
34 void avl_RotDirEsq(NO** raiz);
35 void avl_RotEsqDir2(NO** raiz);
36 void avl_RotDirEsq2(NO** raiz);
37 void avl_AuxFE(NO **raiz);
38 void avl_AuxFD(NO **raiz);
39 int insereRec(NO** raiz, double
    salario, char *nome, int
    ano_contratacao);
40 int insereElem(AVL* raiz, double
    salario, char *nome, int
    ano_contratacao);
41 int pesquisaRec(NO** raiz, double
    salario);
42 int pesquisa(AVL* raiz, double salario);
43 int removeRec(NO** raiz, double
    salario);
44 int removeElem(AVL* raiz, double
    salario);
45 void em_ordem(NO* raiz, int nivel);
46 void pre_ordem(NO* raiz, int nivel);
47 void pos_ordem(NO* raiz, int nivel);
48 void imprime(AVL* raiz);
49 void aguardaLimpa();
50 Funcionario maior(AVL* raiz);
51 Funcionario menor(AVL* raiz);
52
53 #endif
```

codigos/ex12/ex12.h

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "ex12.h"
4 #include <string.h>
5
6 NO* alocarNO(){
7     return (NO*) malloc (sizeof(NO));
8 }
9
10 void liberarNO(NO* q){
11     free(q);
12 }
13
14 AVL* criaAVL(){
15     AVL* raiz = (AVL*) malloc
        (sizeof(AVL));
16     if(raiz != NULL)
17         *raiz = NULL;
18     return raiz;
19 }
20
21 void destroiRec(NO* no){
22     if(no == NULL) return;
23     destroiRec(no->esq);
24     destroiRec(no->dir);
25     liberarNO(no);
26     no = NULL;
27 }
28
29 void destroiAVL(AVL* raiz){
30     if(raiz != NULL){
31         destroiRec(*raiz);
32         free(raiz);
33     }
34 }
35
36 int estaVazia(AVL* raiz){
37     if(raiz == NULL) return 0;
```

```

38     return (*raiz == NULL);
39 }
40
41 //Calcula FB
42 int altura(NO* raiz){
43     if(raiz == NULL) return 0;
44     if(raiz->alt > 0)
45         return raiz->alt;
46     else{
47         //printf("Calculando altura do
48             (%d)..\\n", raiz->info);
49         return MAIOR(altura(raiz->esq),
50             altura(raiz->dir)) + 1;
51     }
52 }
53
54 int FB(NO* raiz){
55     if(raiz == NULL) return 0;
56     printf("Calculando FB do
57         (%lf)..\\n", raiz->info.salarior);
58     return altura(raiz->esq) -
59         altura(raiz->dir);
60 }
61
62 //Funcoes de Rotacao Simples
63 void avl_RotDir(NO** raiz){
64     printf("Rotacao Simples a
65         DIREITA!\\n");
66     NO *aux;
67     aux = (*raiz)->esq;
68     (*raiz)->esq = aux->dir;
69     aux->dir = *raiz;
70
71     //Acertando alturas e FB
72     //dos NOs afetados
73     (*raiz)->alt = aux->alt = -1;
74     aux->alt = altura(aux);
75     (*raiz)->alt = altura(*raiz);
76     aux->fb = FB(aux);
77     (*raiz)->fb = FB(*raiz);
78
79     *raiz = aux;
80 }
81
82 void avl_RotEsq(NO** raiz){
83     printf("Rotacao Simples a
84         ESQUERDA!\\n");
85     NO *aux;
86     aux = (*raiz)->dir;
87     (*raiz)->dir = aux->esq;
88     aux->esq = *raiz;
89
90     //Acertando alturas e Fatores de
91     //Balanceamento dos NOs afetados
92     (*raiz)->alt = aux->alt = -1;
93     aux->alt = altura(aux);
94     (*raiz)->alt = altura(*raiz);
95     aux->fb = FB(aux);
96     (*raiz)->fb = FB(*raiz);
97
98     *raiz = aux;
99 }
100
101 //Funcoes de Rotacao Dupla
102 void avl_RotEsqDir(NO** raiz){
103     printf("Rotacao Dupla
104         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     //Acertando alturas e Fatores de
118     //Balanceamento dos NOs afetados
119     (*raiz)->alt = fe->alt = ffd->alt =
120         -1;
121     fe->alt = altura(fe);
122     ffd->alt = altura(ffd);
123     (*raiz)->alt = altura(*raiz);
124     fe->fb = FB(fe);
125     ffd->fb = FB(ffd);
126     (*raiz)->fb = FB(*raiz);
127
128     *raiz = ffd;
129 }
130
131 void avl_RotDirEsq(NO** raiz){
132     printf("Rotacao Dupla
133         DIREITA-ESQUERDA!\\n");
134     NO* fd; //filho direito
135     NO* ffe; //filho filho esquerdo
136
137     fd = (*raiz)->dir;
138     ffe = fd->esq;
139
140     fd->esq = ffe->dir;
141     ffe->dir = fd;
142
143     (*raiz)->dir = ffe->esq;
144     ffe->esq = *raiz;
145
146     //Acertando alturas e Fatores de
147     //Balanceamento dos NOs afetados
148     (*raiz)->alt = fd->alt = ffe->alt =
149         -1;
150     fd->alt = altura(fd);
151     ffe->alt = altura(ffe);
152     (*raiz)->alt = altura(*raiz);
153     fd->fb = FB(fd);

```



```

143     ffe->fb = FB(ffe);
144     (*raiz)->fb = FB(*raiz);
145
146     *raiz = ffe;
147 }
148
149 void avl_RotEsqDir2(NO** raiz){
150     printf("Rotacao Dupla 2
151           ESQUERDA-DIREITA!\n");
152     avl_RotEsq(&(*raiz)->esq);
153     avl_RotDir(raiz);
154 }
155 void avl_RotDirEsq2(NO** raiz){
156     printf("Rotacao Dupla 2
157           DIREITA-ESQUERDA!\n");
158     avl_RotDir(&(*raiz)->dir);
159     avl_RotEsq(raiz);
160 }
161
162 //Funcoes Auxiliares referentes a cada
163 //filho
164 void avl_AuxFE(NO **raiz){
165     NO* fe;
166     fe = (*raiz)->esq;
167     if(fe->fb == +1) /* Sinais iguais e
168                     positivo*/
169         avl_RotDir(raiz);
170     else /* Sinais diferentes*/
171         avl_RotEsqDir(raiz);
172 }
173 void avl_AuxFD(NO **raiz){
174     NO* fd;
175     fd = (*raiz)->dir;
176     if(fd->fb == -1) /* Sinais iguais e
177                     negativos*/
178         avl_RotEsq(raiz);
179     else /* Sinais diferentes*/
180         avl_RotDirEsq(raiz);
181 }
182
183 int insereRec(NO** raiz, double
184             salario, char *nome, int
185             ano_contratacao){
186     int ok; //Controle para as chamadas
187             recursivas
188     if(*raiz == NULL){
189         NO* novo = alocarNO();
190         if(novo == NULL) return 0;
191         novo->info.salario = salario;
192         novo->fb = 0, novo->alt = 1;
193         strcpy(novo->info.nome, nome);
194         novo->info.anoContratacao =
195             ano_contratacao;
196         novo->esq = NULL; novo->dir =
197             NULL;
198         *raiz = novo; return 1;
199     }
200     }else{
201         if((*raiz)->info.salario ==
202             salario){
203             printf("Funcionario
204             Existente!\n"); ok = 0;
205         }
206         if(salario <
207             (*raiz)->info.salario){
208             ok =
209                 insereRec(&(*raiz)->esq,
210                     salario, nome, ano_contratacao);
211             if(ok){
212                 switch((*raiz)->fb){
213                     case -1:
214                         (*raiz)->fb =
215                             0; ok = 0;
216                         break;
217                     case 0:
218                         (*raiz)->fb =
219                             +1;
220                         (*raiz)->alt++;
221                         break;
222                     case +1:
223                         avl_AuxFE(raiz);
224                         ok = 0;
225                         break;
226                 }
227             }
228         }
229         else if(salario >
230             (*raiz)->info.salario){
231             ok =
232                 insereRec(&(*raiz)->dir,
233                     salario, nome, ano_contratacao);
234             if(ok){
235                 switch((*raiz)->fb){
236                     case +1:
237                         (*raiz)->fb =
238                             0; ok = 0;
239                         break;
240                     case 0:
241                         (*raiz)->fb =
242                             -1;
243                         (*raiz)->alt++;
244                         break;
245                     case -1:
246                         avl_AuxFD(raiz);
247                         ok = 0;
248                         break;
249                 }
250             }
251         }
252     }
253     return ok;
254 }
255
256 int insereElem(AVL* raiz, double
257             salario, char *nome, int
258             ano_contratacao){

```

228	if(raiz == NULL) return 0;	269
229	return insereRec(raiz, salario, nome, ano_contratacao);	270
230 }		271
231		272
232 int pesquisaRec(NO** raiz, double salario){		
233 if(*raiz == NULL) return 0;	273	
234 if((*raiz->info.salario == salario) return 1;		
235 if(salario < (*raiz->info.salario)	274	
236 return		
	pesquisaRec(&(*raiz->esq, salario);	275
237 else		276
238 return		277
	pesquisaRec(&(*raiz->dir, salario);	
239 }		
240		278
241 int pesquisa(AVL* raiz, double salario)	279	
242 if(raiz == NULL) return 0;	280	
243 if(estaVazia(raiz)) return 0;		
244 return pesquisaRec(raiz, salario);	281	
245 }		282
246		283
247 int removeRec(NO** raiz, double salario){		
248 if(*raiz == NULL) return 0;	284	
249 int ok;	285	
250 if((*raiz->info.salario == salario){	286	
251 NO* aux;	287	
252 if((*raiz->esq == NULL && (*raiz->dir == NULL){	288	
253 //Caso 1 - NO sem filhos	289	
254 printf("Caso 1: Liberando	290	
	%lf...\n",	291
	(*raiz->info.salario);	292
255 liberarNO(*raiz);		
256 *raiz = NULL;		
257 }else if((*raiz->esq == NULL){		
258 //Caso 2.1 - Possui apenas	293	
	uma subarvore direita	294
259 printf("Caso 2.1: Liberando	295	
	%lf...\n",	
	(*raiz->info.salario);	
260 aux = *raiz;	296	
261 *raiz = (*raiz->dir;	297	
262 liberarNO(aux);	298	
263 }else if((*raiz->dir == NULL){		
264 //Caso 2.2 - Possui apenas	299	
	uma subarvore esquerda	300
265 printf("Caso 2.2: Liberando	301	
	%lf...\n",	302
	(*raiz->info.salario);	303
266 aux = *raiz;		
267 *raiz = (*raiz->esq;	304	
268 liberarNO(aux);	305	

```

306         case -1:                                     raiz->fb, nivel));
307         case 0:                                     350     }
308         //Acertando alturas 351 }
309         e Fatores de 352
310         Balanceamento 353 void imprime(AVL* raiz){
311         dos NOs afetados 354     if(raiz == NULL) return;
312         (*raiz)->alt = -1; 355     if(estaVazia(raiz)){
313         (*raiz)->alt = 356         printf("Arvore Vazia!\n");
314         altura(*raiz); 357         return;
315         (*raiz)->fb = 358     }
316         FB(*raiz); 359     //printf("\nEm Ordem: [INFO, FB,
317         break;                                     NIVEL]\n");
318         case +1: 360     printf("\nEm Ordem: [SALARIO, NOME,
319         avl_AuxFE(raiz); 361         ANO DA CONTRATAcao, FB, NIVEL,
320         break;                                     altura]\n");
321     } 362     em_ordem(*raiz, 0);
322     } 363     //printf("\nPre Ordem: ");
323     return ok; 364     pre_ordem(*raiz, 0);
324     } 365     //printf("\nPos Ordem: ");
325     } 366     pos_ordem(*raiz, 0);
326     } 367     printf("\n");
327     } 368     void aguardaLimpa(){
328     } 369     getchar();
329     int removeElem(AVL* raiz, double 370     printf("\n\nAperte qualquer tecla
330     salario){ 371     para continuar\n");
331     if(pesquisa(raiz, salario) == 0){ 372     getchar();
332     printf("Funcionario 373     system("clear");
333     inexistente!\n"); 374
334     return 0; 375     Funcionario maior(AVL* raiz){
335     } 376     Funcionario funcionarioMaior = {"",
336     return removeRec(raiz, salario); 377     0.0, 0};
337     } 378     if ((*raiz) == NULL)
338     void em_ordem(NO* raiz, int nivel){ 379     return funcionarioMaior;
339     if(raiz != NULL){ 380
340     em_ordem(raiz->esq, nivel+1); 381     NO* aux = (*raiz);
341     printf("[%21f, %s, %d, %d, %d, 382     while(aux->dir != NULL)
342     %d] ", raiz->info.salarioraiz->info.nome, 383     aux = aux->dir;
343     raiz->info.anoContratacao, 384     funcionarioMaior.salarioraiz->fb, nivel, raiz->alt); 385     aux->info.salarioraiz->info.anoContratacao =
344     em_ordem(raiz->dir, nivel+1); 386     funcionarioMaior.anoContratacao =
345     } 387     strcpy(funcionarioMaior.nome,
346     } 388     aux->info.nome);
347     void pre_ordem(NO* raiz, int nivel){ 389     return funcionarioMaior;
348     if(raiz != NULL){ 390
349     printf("[%1f, %d, %d] ", 391     Funcionario menor(AVL* raiz){
350     raiz->info.salarioraiz->fb, nivel); 392     Funcionario funcionarioMenor = {"",
351     pre_ordem(raiz->esq, nivel+1); 393     0.0, 0};
352     pre_ordem(raiz->dir, nivel+1); 394
353     } 395     if ((*raiz) == NULL) {
354     } 396     return funcionarioMenor;
355     } 397
356     void pos_ordem(NO* raiz, int nivel){ 398
357     if(raiz != NULL){ 399
358     pos_ordem(raiz->esq, nivel+1); 400
359     pos_ordem(raiz->dir, nivel+1); 401
360     printf("[%1f, %d, %d] ", 402
361     raiz->info.salarioraiz->fb, nivel); 403
362     } 404
363     } 405
364     } 406
365     } 407
366     } 408
367     } 409
368     } 410
369     } 411
370     } 412
371     } 413
372     } 414
373     } 415
374     } 416
375     } 417
376     } 418
377     } 419
378     } 420
379     } 421
380     } 422
381     } 423
382     } 424
383     } 425
384     } 426
385     } 427
386     } 428
387     } 429
388     } 430
389     } 431
390     } 432
391     } 433
392     } 434
393     } 435
394     } 436
395     } 437
396     } 438
397     } 439
398     } 440
399     } 441
400     } 442
401     } 443
402     } 444
403     } 445
404     } 446
405     } 447
406     } 448
407     } 449
408     } 450
409     } 451
410     } 452
411     } 453
412     } 454
413     } 455
414     } 456
415     } 457
416     } 458
417     } 459
418     } 460
419     } 461
420     } 462
421     } 463
422     } 464
423     } 465
424     } 466
425     } 467
426     } 468
427     } 469
428     } 470
429     } 471
430     } 472
431     } 473
432     } 474
433     } 475
434     } 476
435     } 477
436     } 478
437     } 479
438     } 480
439     } 481
440     } 482
441     } 483
442     } 484
443     } 485
444     } 486
445     } 487
446     } 488
447     } 489
448     } 490
449     } 491
450     } 492
451     } 493
452     } 494
453     } 495
454     } 496
455     } 497
456     } 498
457     } 499
458     } 500
459     } 501
460     } 502
461     } 503
462     } 504
463     } 505
464     } 506
465     } 507
466     } 508
467     } 509
468     } 510
469     } 511
470     } 512
471     } 513
472     } 514
473     } 515
474     } 516
475     } 517
476     } 518
477     } 519
478     } 520
479     } 521
480     } 522
481     } 523
482     } 524
483     } 525
484     } 526
485     } 527
486     } 528
487     } 529
488     } 530
489     } 531
490     } 532
491     } 533
492     } 534
493     } 535
494     } 536
495     } 537
496     } 538
497     } 539
498     } 540
499     } 541
500     } 542
501     } 543
502     } 544
503     } 545
504     } 546
505     } 547
506     } 548
507     } 549
508     } 550
509     } 551
510     } 552
511     } 553
512     } 554
513     } 555
514     } 556
515     } 557
516     } 558
517     } 559
518     } 560
519     } 561
520     } 562
521     } 563
522     } 564
523     } 565
524     } 566
525     } 567
526     } 568
527     } 569
528     } 570
529     } 571
530     } 572
531     } 573
532     } 574
533     } 575
534     } 576
535     } 577
536     } 578
537     } 579
538     } 580
539     } 581
540     } 582
541     } 583
542     } 584
543     } 585
544     } 586
545     } 587
546     } 588
547     } 589
548     } 590
549     } 591
550     } 592
551     } 593
552     } 594
553     } 595
554     } 596
555     } 597
556     } 598
557     } 599
558     } 600
559     } 601
560     } 602
561     } 603
562     } 604
563     } 605
564     } 606
565     } 607
566     } 608
567     } 609
568     } 610
569     } 611
570     } 612
571     } 613
572     } 614
573     } 615
574     } 616
575     } 617
576     } 618
577     } 619
578     } 620
579     } 621
580     } 622
581     } 623
582     } 624
583     } 625
584     } 626
585     } 627
586     } 628
587     } 629
588     } 630
589     } 631
590     } 632
591     } 633
592     } 634
593     } 635
594     } 636
595     } 637
596     } 638
597     } 639
598     } 640
599     } 641
600     } 642
601     } 643
602     } 644
603     } 645
604     } 646
605     } 647
606     } 648
607     } 649
608     } 650
609     } 651
610     } 652
611     } 653
612     } 654
613     } 655
614     } 656
615     } 657
616     } 658
617     } 659
618     } 660
619     } 661
620     } 662
621     } 663
622     } 664
623     } 665
624     } 666
625     } 667
626     } 668
627     } 669
628     } 670
629     } 671
630     } 672
631     } 673
632     } 674
633     } 675
634     } 676
635     } 677
636     } 678
637     } 679
638     } 680
639     } 681
640     } 682
641     } 683
642     } 684
643     } 685
644     } 686
645     } 687
646     } 688
647     } 689
648     } 690
649     } 691
650     } 692
651     } 693
652     } 694
653     } 695
654     } 696
655     } 697
656     } 698
657     } 699
658     } 700
659     } 701
660     } 702
661     } 703
662     } 704
663     } 705
664     } 706
665     } 707
666     } 708
667     } 709
668     } 710
669     } 711
670     } 712
671     } 713
672     } 714
673     } 715
674     } 716
675     } 717
676     } 718
677     } 719
678     } 720
679     } 721
680     } 722
681     } 723
682     } 724
683     } 725
684     } 726
685     } 727
686     } 728
687     } 729
688     } 730
689     } 731
690     } 732
691     } 733
692     } 734
693     } 735
694     } 736
695     } 737
696     } 738
697     } 739
698     } 740
699     } 741
700     } 742
701     } 743
702     } 744
703     } 745
704     } 746
705     } 747
706     } 748
707     } 749
708     } 750
709     } 751
710     } 752
711     } 753
712     } 754
713     } 755
714     } 756
715     } 757
716     } 758
717     } 759
718     } 760
719     } 761
720     } 762
721     } 763
722     } 764
723     } 765
724     } 766
725     } 767
726     } 768
727     } 769
728     } 770
729     } 771
730     } 772
731     } 773
732     } 774
733     } 775
734     } 776
735     } 777
736     } 778
737     } 779
738     } 780
739     } 781
740     } 782
741     } 783
742     } 784
743     } 785
744     } 786
745     } 787
746     } 788
747     } 789
748     } 790
749     } 791
750     } 792
751     } 793
752     } 794
753     } 795
754     } 796
755     } 797
756     } 798
757     } 799
758     } 800
759     } 801
760     } 802
761     } 803
762     } 804
763     } 805
764     } 806
765     } 807
766     } 808
767     } 809
768     } 810
769     } 811
770     } 812
771     } 813
772     } 814
773     } 815
774     } 816
775     } 817
776     } 818
777     } 819
778     } 820
779     } 821
780     } 822
781     } 823
782     } 824
783     } 825
784     } 826
785     } 827
786     } 828
787     } 829
788     } 830
789     } 831
790     } 832
791     } 833
792     } 834
793     } 835
794     } 836
795     } 837
796     } 838
797     } 839
798     } 840
799     } 841
800     } 842
801     } 843
802     } 844
803     } 845
804     } 846
805     } 847
806     } 848
807     } 849
808     } 850
809     } 851
810     } 852
811     } 853
812     } 854
813     } 855
814     } 856
815     } 857
816     } 858
817     } 859
818     } 860
819     } 861
820     } 862
821     } 863
822     } 864
823     } 865
824     } 866
825     } 867
826     } 868
827     } 869
828     } 870
829     } 871
830     } 872
831     } 873
832     } 874
833     } 875
834     } 876
835     } 877
836     } 878
837     } 879
838     } 880
839     } 881
840     } 882
841     } 883
842     } 884
843     } 885
844     } 886
845     } 887
846     } 888
847     } 889
848     } 890
849     } 891
850     } 892
851     } 893
852     } 894
853     } 895
854     } 896
855     } 897
856     } 898
857     } 899
858     } 900
859     } 901
860     } 902
861     } 903
862     } 904
863     } 905
864     } 906
865     } 907
866     } 908
867     } 909
868     } 910
869     } 911
870     } 912
871     } 913
872     } 914
873     } 915
874     } 916
875     } 917
876     } 918
877     } 919
878     } 920
879     } 921
880     } 922
881     } 923
882     } 924
883     } 925
884     } 926
885     } 927
886     } 928
887     } 929
888     } 930
889     } 931
890     } 932
891     } 933
892     } 934
893     } 935
894     } 936
895     } 937
896     } 938
897     } 939
898     } 940
899     } 941
900     } 942
901     } 943
902     } 944
903     } 945
904     } 946
905     } 947
906     } 948
907     } 949
908     } 950
909     } 951
910     } 952
911     } 953
912     } 954
913     } 955
914     } 956
915     } 957
916     } 958
917     } 959
918     } 960
919     } 961
920     } 962
921     } 963
922     } 964
923     } 965
924     } 966
925     } 967
926     } 968
927     } 969
928     } 970
929     } 971
930     } 972
931     } 973
932     } 974
933     } 975
934     } 976
935     } 977
936     } 978
937     } 979
938     } 980
939     } 981
940     } 982
941     } 983
942     } 984
943     } 985
944     } 986
945     } 987
946     } 988
947     } 989
948     } 990
949     } 991
950     } 992
951     } 993
952     } 994
953     } 995
954     } 996
955     } 997
956     } 998
957     } 999
958     } 1000
959     } 1001
960     } 1002
961     } 1003
962     } 1004
963     } 1005
964     } 1006
965     } 1007
966     } 1008
967     } 1009
968     } 1010
969     } 1011
970     } 1012
971     } 1013
972     } 1014
973     } 1015
974     } 1016
975     } 1017
976     } 1018
977     } 1019
978     } 1020
979     } 1021
980     } 1022
981     } 1023
982     } 1024
983     } 1025
984     } 1026
985     } 1027
986     } 1028
987     } 1029
988     } 1030
989     } 1031
990     } 1032
991     } 1033
992     } 1034
993     } 1035
994     } 1036
995     } 1037
996     } 1038
997     } 1039
998     } 1040
999     } 1041
1000     } 1042
1001     } 1043
1002     } 1044
1003     } 1045
1004     } 1046
1005     } 1047
1006     } 1048
1007     } 1049
1008     } 1050
1009     } 1051
1010     } 1052
1011     } 1053
1012     } 1054
1013     } 1055
1014     } 1056
1015     } 1057
1016     } 1058
1017     } 1059
1018     } 1060
1019     } 1061
1020     } 1062
1021     } 1063
1022     } 1064
1023     } 1065
1024     } 1066
1025     } 1067
1026     } 1068
1027     } 1069
1028     } 1070
1029     } 1071
1030     } 1072
1031     } 1073
1032     } 1074
1033     } 1075
1034     } 1076
1035     } 1077
1036     } 1078
1037     } 1079
1038     } 1080
1039     } 1081
1040     } 1082
1041     } 1083
1042     } 1084
1043     } 1085
1044     } 1086
1045     } 1087
1046     } 1088
1047     } 1089
1048     } 1090
1049     } 1091
1050     } 1092
1051     } 1093
1052     } 1094
1053     } 1095
1054     } 1096
1055     } 1097
1056     } 1098
1057     } 1099
1058     } 1100
1059     } 1101
1060     } 1102
1061     } 1103
1062     } 1104
1063     } 1105
1064     } 1106
1065     } 1107
1066     } 1108
1067     } 1109
1068     } 1110
1069     } 1111
1070     } 1112
1071     } 1113
1072     } 1114
1073     } 1115
1074     } 1116
1075     } 1117
1076     } 1118
1077     } 1119
1078     } 1120
1079     } 1121
1080     } 1122
1081     } 1123
1082     } 1124
1083     } 1125
1084     } 1126
1085     } 1127
1086     } 1128
1087     } 1129
1088     } 1130
1089     } 1131
1090     } 1132
1091     } 1133
1092     } 1134
1093     } 1135
1094     } 1136
1095     } 1137
1096     } 1138
1097     } 1139
1098     } 1140
1099     } 1141
1100     } 1142
1101     } 1143
1102     } 1144
1103     } 1145
1104     } 1146
1105     } 1147
1106     } 1148
1107     } 1149
1108     } 1150
1109     } 1151
1110     } 1152
1111     } 1153
1112     } 1154
1113     } 1155
1114     } 1156
1115     } 1157
1116     } 1158
1117     } 1159
1118     } 1160
1119     } 1161
1120     } 1162
1121     } 1163
1122     } 1164
1123     } 1165
1124     } 1166
1125     } 1167
1126     } 1168
1127     } 1169
1128     } 1170
1129     } 1171
1130     } 1172
1131     } 1173
1132     } 1174
1133     } 1175
1134     } 1176
1135     } 1177
1136     } 1178
1137     } 1179
1138     } 1180
1139     } 1181
1140     } 1182
1141     } 1183
1142     } 1184
1143     } 1185
1144     } 1186
1145     } 1187
1146     } 1188
1147     } 1189
1148     } 1190
1149     } 1191
1150     } 1192
1151     } 1193
1152     } 1194
1153     } 1195
1154     } 1196
1155     } 1197
1156     } 1198
1157     } 1199
1158     } 1200
1159     } 1201
1160     } 1202
1161     } 1203
1162     } 1204
1163     } 1205
1164     } 1206
1165     } 1207
1166     } 1208
1167     } 1209
1168     } 1210
1169     } 1211
1170     } 1212
1171     } 1213
1172     } 1214
1173     } 1215
1174     } 1216
1175     } 1217
1176     } 1218
1177     } 1219
1178     } 1220
1179     } 1221
1180     } 1222
1181     } 1223
1182     } 1224
1183     } 1225
1184     } 1226
1185     } 1227
1186     } 1228
1187     } 1229
1188     } 1230
1189     } 1231
1190     } 1232
1191     } 1233
1192     } 1234
1193     } 1235
1194     } 1236
1195     } 1237
1196     } 1238
1197     } 1239
1198     } 1240
1199     } 1241
1200     } 1242
1201     } 1243
1202     } 1244
1203     } 1245
1204     } 1246
1205     } 1247
1206     } 1248
1207     } 1249
1208     } 1250
1209     } 1251
1210     } 1252
1211     } 1253
1212     } 1254
1213     } 1255
1214     } 1256
1215     } 1257
1216     } 1258
1217     } 1259
1218     } 1260
1219     } 1261
1220     } 1262
1221     } 1263
1222     } 1264
1223     } 1265
1224     } 1266
1225     } 1267
1226     } 1268
1227     } 1269
1228     } 1270
1229     } 1271
1230     } 1272
1231     } 1273
1232     } 1274
1233     } 1275
1234     } 1276
1235     } 1277
1236     } 1278
1237     } 1279
1238     } 1280
1239     } 1281
1240     } 1282
1241     } 1283
1242     } 1284
1243     } 1285
1244     } 1286
1245     } 1287
1246     } 1288
1247     } 1289
1248     } 1290
1249     } 1291
1250     } 1292
1251     } 1293
1252     } 1294
1253     } 1295
1254     } 1296
1255     } 1297
1256     } 1298
1257     } 1299
1258     } 1300
1259     } 1301
1260     } 1302
1261     } 1303
1262     } 1304
1263     } 1305
1264     } 1306
1265     } 1307
1266     } 1308
1267     } 1309
1268     } 1310
1269     } 1311
1270     } 1312
1271     } 1313
1272     } 1314
1273     } 1315
1274     } 1316
1275     } 1317
1276     } 1318
1277     } 1319
1278     } 1320
1279     } 1321
1280     } 1322
1281     } 1323
1282     } 1324
1283     } 1325
1284     } 1326
1285     } 1327
1286     } 1328
1287     } 1329
1288     } 1330
1289     } 1331
1290     } 1332
1291     } 1333
1292     } 1334
1293     } 1335
1294     } 1336
1295     } 1337
1296     } 1338
1297     } 1339
1298     } 1340
1299     } 1341
1300     } 1342
1301     } 1343
1302     } 1344
1303     } 1345
1304     } 1346
1305     } 1347
1306     } 1348
1307     } 1349
1308     } 1350
1309     } 1351
1310     } 1352
1311     } 1353
1312     } 1354
1313     } 1355
1314     } 1356
1315     } 1357
1316     } 1358
1317     } 1359
1318     } 1360
1319     } 1361
1320     } 1362
1321     } 1363
1322     } 1364
1323     } 1365
1324     } 1366
1325     } 1367
1326     } 1368
1327     } 1369
1328     } 1370
1329     } 1371
1330     } 1372
1331     } 1373
1332     } 1374
1333     } 1375
1334     } 1376
1335     } 1377
1336     } 1378
1337     } 1379
1338     } 1380
1339     } 1381
1340     } 1382
1341     } 1383
1342     } 1384
1343     } 1385
1344     } 1386
1345     } 1387
1346     } 1388
1347     } 1389
1348     } 1390
1349     } 1391
1350     } 1392
1351     } 1393
1352     } 1394
1353     } 1395
1354     } 1396
1355     } 1397
1356     } 1398
1357     } 1399
1358     } 1400
1359     } 1401
1360     } 1402
1361     } 1403
1362     } 1404
1363     } 1405
1364     } 1406
1365     } 1407
1366     } 1408
1367     } 1409
1368     } 1410
1369     } 1411
1370     } 1412
1371     } 1413
1372     } 1414
1373     } 1415
1374     } 1416
1375     } 1417
1376     } 1418
1377     } 1419
1378     } 1420
1379     } 1421
1380     } 1422
1381     } 1423
1382     } 1424
1383     } 1425
1384     } 1426
1385     } 1427
1386     } 1428
1387
```

```

396
397 NO* aux = (*raiz);
398 while (aux->esq != NULL) {
399     aux = aux->esq;
400 }
401
402 funcionarioMenor.salario =
403     aux->info.salario;
404     strcpy(funcionarioMenor.nome,
405         aux->info.nome);
406     return funcionarioMenor;
407 }

```

codigos/ex12/ex12.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "ex12.h"
4 #include <string.h>
5
6 int main() {
7     int *cont =
8         (int*)malloc(sizeof(int));
9     int escolha, ano_contratacao, busca;
10    double salario;
11    char* nome = (char*)malloc(30 *
12        sizeof(char));
13    AVL avl = NULL;
14
15    printf("0 que deseja fazer?\n1 -
16        Criar AVL\n2 - Inserir um
17        salario do funcionario\n"
18        "3 - Buscar um
19        funcionario\n4 - Remover
20        um funcionario\n5 -
21        Imprimir a AVL em ordem"
22        "\n6 - Imprimir as
23        informacoes do
24        Funcionario com o maior
25        salario\n"
26        "7 - Imprimir as informacoes
27        do Funcionario com o
28        menor salario\n8 -
29        Destruir a AVL\n9 -
30        Sair\n");
31
32    while (1) {
33        scanf("%d", &escolha);
34
35        if (escolha == 9) {
36            break;
37        }
38
39        if (escolha == 1) {
40            if (avl == NULL) {
41                avl = criaAVL();
42                printf("Arvore
43                    criada!\n");
44            } else {
45                printf("A arvore ja foi
46                    criada!\n");
47            }
48        }
49
50        if (escolha == 2) {
51            if (avl == NULL) {
52                printf("Crie a arvore
53                    AVL antes de inserir
54                    um funcionario!\n");
55            } else {
56                printf("Digite o
57                    salario, o nome e o
58                    ano de contratacao
59                    do Funcionario que
60                    deseja inserir: ");
61                scanf("%lf", &salario);
62                getchar();
63                fgets(nome, 30, stdin);
64                nome[strcspn(nome,
65                    "\n")] = '\0';
66                scanf("%d",
67                    &ano_contratacao);
68
69                insereElem(avl,
70                    salario, nome,
71                    ano_contratacao);
72                aguardaLimpa();
73            }
74        } else if (escolha == 3) {
75            if (avl == NULL) {
76                printf("Crie a arvore
77                    AVL antes de buscar
78                    um funcionario!\n");
79            } else {
80                printf("Digite o
81                    salario do
82                    funcionario que
83                    deseja consultar: ");
84                scanf("%lf", &salario);
85                busca = pesquisa(avl,
86                    salario);
87                if (busca == 1)
88                    printf("0
89                        funcionario esta
90                        na arvore!\n");
91                else
92                    printf("0
93                        funcionario nao
94                        esta na
95                        arvore!\n");
96            }
97        }
98    }
99 }

```

```

58         aguardaLimpa();
59     }
60     } else if (escolha == 4) {
61         if (avl == NULL) {
62             printf("Crie a arvore
63                 AVL antes de remover
64                 um funcionario!\n");
65         } else {
66             printf("Digite o
67                 salario do
68                 funcionario que
69                 deseja remover: ");
70             scanf("%lf", &salario);
71             removeElem(avl,
72                 salario);
73             aguardaLimpa();
74         }
75     } else if (escolha == 5) {
76         if (avl == NULL) {
77             printf("Crie a arvore
78                 AVL antes de
79                 imprimir!\n");
80         } else {
81             imprime(avl);
82             aguardaLimpa();
83         }
84     } else if (escolha == 6) {
85         if (avl == NULL) {
86             printf("Crie a arvore
87                 AVL antes de
88                 imprimir!\n");
89         } else {
90             Funcionario func =
91                 maior(avl);
92             printf("0 funcionario
93                 de maior salario
94                 eh:%s\nSeu salario
95                 eh de %.2lf\n"
96                 "Ele entrou no ano
97                 func.nome,func.salario,func.anoContratacao);
98             aguardaLimpa();
99         }
100     } else if (escolha == 7) {
101         if (avl == NULL) {
102             printf("Crie a arvore
103                 AVL antes de
104                 imprimir!\n");
105         } else {
106             Funcionario func =
107                 menor(avl);
108             printf("0 funcionario
109                 de menor salario
110                 eh:%s\nSeu salario
111                 eh de %.2lf\n"
112                 "Ele entrou no ano
113                 func.nome,func.salario,func.anoContratacao);
114             aguardaLimpa();
115         }
116     } else if (escolha == 8) {
117         if (avl != NULL) {
118             destroiAVL(avl);
119             printf("arvore
120                 destruida!\n");
121         } else {
122             printf("A arvore ja foi
123                 destruida ou ainda
124                 nao foi criada!\n");
125         }
126         aguardaLimpa();
127     } else {
128         printf("\nOpcao
129             invalida!\n\n");
130     }
131     free(cont);
132     free(nome);
133     return 0;
134 }

```

```

1 all: ex12.o
2     gcc ex12.o main.c -o main
3
4 ex12.o: ex12.h ex12.c
5     gcc -c ex12.c

```

```

6
7 clean:
8     rm -f ex12.o main

```

codigos/ex12/Makefile

2 Código

<https://github.com/RodrigoZonzin/labaeds2>