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 Reimplentação

Código

```h
#ifndef AVL_H
#define AVL_H

#include <stdio.h>
#include <stdlib.h>
#define MAIOR(a, b) ((a > b) ? (a) : (b))

typedef struct NO{
    int info, fb, alt;
    struct NO* esq;
    struct NO* dir;
}NO;

typedef struct NO* AVL;

NO* alocarNO();
void liberarNO(NO* q);
AVL* criaAVL();
void destroiRec(NO* no);
void destroiAVL(AVL* raiz);
int estaVazia(AVL* raiz);
int altura(NO* raiz);
int FB(NO* raiz);
void avl_RotDir(NO** raiz);
void avl_RotEsq(NO** raiz);
void avl_RotEsqDir(NO** raiz);
void avl_RotDirEsq(NO** raiz);
void avl_RotEsqDir2(NO** raiz);
void avl_RotDirEsq2(NO** raiz);
void avl_AuxFE(NO **raiz);
void avl_AuxFD(NO **raiz);
int insereRec(NO** raiz, int elem);
int insereElem(AVL* raiz, int elem);
int pesquisaRec(NO** raiz, int elem);
int pesquisa(AVL* raiz, int elem);
int removeRec(NO** raiz, int elem);
int removeElem(AVL* raiz, int elem);
void em_ordem(NO* raiz, int nivel);
void pre_ordem(NO* raiz, int nivel);
void pos_ordem(NO* raiz, int nivel);
void imprime(AVL* raiz);
void aguardaLimpa();
void contador(NO* raiz, int nivel, int *cont);

#endif
```

codigos/ex11/ex11.h

```c
#include <stdio.h>
#include <stdlib.h>
#include "ex11.h"

NO* alocarNO(){
    return (NO*) malloc (sizeof(NO));
}

void liberarNO(NO* q){
    free(q);
}

AVL* criaAVL(){
    AVL* raiz = (AVL*) malloc (sizeof(AVL));
    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
                (%d)..\n", raiz->info);
47          return MAIOR(altura(raiz->esq),
                altura(raiz->dir)) + 1;
48      }
49  }
50
51  int FB(NO* raiz){
52      if(raiz == NULL) return 0;
53      printf("Calculando FB do (%d)..\n",
            raiz->info);
54      return altura(raiz->esq) -
            altura(raiz->dir);
55  }
56
57  //Funcoes de Rotacao Simples
58  void avl_RotDir(NO** raiz){
59     printf("Rotacao Simples a
           DIREITA!\n");
60     NO *aux;
61     aux = (*raiz)->esq;
62     (*raiz)->esq = aux->dir;
63     aux->dir = *raiz;
64
65     //Acertando alturas e FB
66     //dos NOs afetados
67      (*raiz)->alt = aux->alt = -1;
68      aux->alt = altura(aux);
69      (*raiz)->alt = altura(*raiz);
70      aux->fb = FB(aux);
71      (*raiz)->fb = FB(*raiz);
72
73     *raiz = aux;
74  }
75
76  void avl_RotEsq(NO** raiz){
77     printf("Rotacao Simples a
           ESQUERDA!\n");
78     NO *aux;
79     aux = (*raiz)->dir;
80     (*raiz)->dir = aux->esq;
81     aux->esq = *raiz;
82
83     //Acertando alturas e Fatores de
           Balanceamento dos NOs afetados
84      (*raiz)->alt = aux->alt = -1;
85      aux->alt = altura(aux);
86      (*raiz)->alt = altura(*raiz);
87      aux->fb = FB(aux);
88      (*raiz)->fb = FB(*raiz);
89
90     *raiz = aux;
91  }
92
93
94  //Funcoes de Rotacao Dupla
95  void avl_RotEsqDir(NO** raiz){
96      printf("Rotacao Dupla
            ESQUERDA-DIREITA!\n");
97      NO *fe; //filho esquerdo
98      NO *ffd; //filho filho direito
99
100     fe = (*raiz)->esq;
101     ffd = fe->dir;
102
103     fe->dir = ffd->esq;
104     ffd->esq = fe;
105
106     (*raiz)->esq = ffd->dir;
107     ffd->dir = *raiz;
108
109     //Acertando alturas e Fatores de
            Balanceamento dos NOs afetados
110     (*raiz)->alt = fe->alt = ffd->alt =
            -1;
111     fe->alt = altura(fe);
112     ffd->alt = altura(ffd);
113     (*raiz)->alt = altura(*raiz);
114     fe->fb = FB(fe);
115     ffd->fb = FB(ffd);
116     (*raiz)->fb = FB(*raiz);
117
118     *raiz = ffd;
119 }
120
121
122 void avl_RotDirEsq(NO** raiz){
123     printf("Rotacao Dupla
```

```
        DIREITA-ESQUERDA!\n");
124     NO* fd; //filho direito
125     NO* ffe; //filho filho esquerdo
126
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
            Balanceamento dos NOs afetados
137     (*raiz)->alt = fd->alt = ffe->alt =
            -1;
138     fd->alt = altura(fd);
139     ffe->alt = altura(ffe);
140     (*raiz)->alt = altura(*raiz);
141     fd->fb = FB(fd);
142     ffe->fb = FB(ffe);
143     (*raiz)->fb = FB(*raiz);
144
145     *raiz = ffe;
146 }
147
148 void avl_RotEsqDir2(NO** raiz){
149     printf("Rotacao Dupla 2
            ESQUERDA-DIREITA!\n");
150     avl_RotEsq(&(*raiz)->esq);
151     avl_RotDir(raiz);
152 }
153
154 void avl_RotDirEsq2(NO** raiz){
155     printf("Rotacao Dupla 2
            DIREITA-ESQUERDA!\n");
156     avl_RotDir(&(*raiz)->dir);
157     avl_RotEsq(raiz);
158 }
159
160
161 //Funcoes Auxiliares referentes a cada
        filho
162 void avl_AuxFE(NO **raiz){
163    NO* fe;
164    fe = (*raiz)->esq;
165    if(fe->fb == +1) /* Sinais iguais e
           positivo*/
166      avl_RotDir(raiz);
167    else /* Sinais diferentes*/
168      avl_RotEsqDir(raiz);
169 }
170
171 void avl_AuxFD(NO **raiz){
172    NO* fd;
173    fd = (*raiz)->dir;
174    if(fd->fb == -1) /* Sinais iguais e
           negativos*/
175      avl_RotEsq(raiz);
176    else /* Sinais diferentes*/
177      avl_RotDirEsq(raiz);
178 }
179
180 int insereRec(NO** raiz, int elem){
181     int ok; //Controle para as chamadas
            recursivas
182     if(*raiz == NULL){
183         NO* novo = alocarNO();
184         if(novo == NULL) return 0;
185         novo->info = elem; novo->fb =
                0, novo->alt = 1;
186         novo->esq = NULL; novo->dir =
                NULL;
187         *raiz = novo; return 1;
188     }else{
189         if((*raiz)->info == elem){
190             printf("Elemento
                    Existente!\n"); 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; ok = 0;
                                break;
198                     case 0:
199                         (*raiz)->fb =
                                +1;
200                         (*raiz)->alt++;
201                         break;
202                     case +1:
203                         avl_AuxFE(raiz);
                                ok = 0;
                                break;
204                 }
205             }
206         }
207         else if(elem > (*raiz)->info){
208             ok =
                    insereRec(&(*raiz)->dir,
                    elem);
209             if(ok){
210                 switch((*raiz)->fb){
211                     case +1:
212                         (*raiz)->fb =
                                0; ok = 0;
                                break;
213                     case 0:
214                         (*raiz)->fb =
                                -1;
                                (*raiz)->alt++;
                                break;
215                     case -1:
```

```
                        avl_AuxFD(raiz);
                        ok = 0;
                        break;
                }
            }
        }
    }
    return ok;
}

int insereElem(AVL* raiz, int elem){
    if(raiz == NULL) return 0;
    return insereRec(raiz, elem);
}

int pesquisaRec(NO** raiz, int elem){
    if(*raiz == NULL) return 0;
    if((*raiz)->info == elem) return 1;
    if(elem < (*raiz)->info)
        return
            pesquisaRec(&(*raiz)->esq,
            elem);
    else
        return
            pesquisaRec(&(*raiz)->dir,
            elem);
}

int pesquisa(AVL* raiz, int elem){
    if(raiz == NULL) return 0;
    if(estaVazia(raiz)) return 0;
    return pesquisaRec(raiz, elem);
}

int removeRec(NO** raiz, int elem){
    if(*raiz == NULL) return 0;
    int ok;
    if((*raiz)->info == elem){
        NO* aux;
        if((*raiz)->esq == NULL &&
            (*raiz)->dir == NULL){
            //Caso 1 - NO sem filhos
            printf("Caso 1: Liberando
                %d..\n", (*raiz)->info);
            liberarNO(*raiz);
            *raiz = NULL;
        }else if((*raiz)->esq == NULL){
            //Caso 2.1 - Possui apenas
                uma subarvore direita
            printf("Caso 2.1: Liberando
                %d..\n", (*raiz)->info);
            aux = *raiz;
            *raiz = (*raiz)->dir;
            liberarNO(aux);
        }else if((*raiz)->dir == NULL){
            //Caso 2.2 - Possui apenas
                uma subarvore esquerda
            printf("Caso 2.2: Liberando
                %d..\n", (*raiz)->info);
            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:
```

```
                        //Acertando alturas e Fatores de Balanceamento dos NOs afetados
                        (*raiz)->alt = -1;
                        (*raiz)->alt = altura(*raiz);
                        (*raiz)->fb = FB(*raiz);
                        break;
                    case +1:
                        avl_AuxFE(raiz);
                        break;
                }
            }
        }
        return ok;
}

int removeElem(AVL* raiz, int elem){
    if(pesquisa(raiz, elem) == 0){
        printf("Elemento inexistente!\n");
        return 0;
    }
    return removeRec(raiz, elem);
}

void em_ordem(NO* raiz, int nivel){
    if(raiz != NULL){
        em_ordem(raiz->esq, nivel+1);
        //printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
        printf("[%d, %d, %d, %d] ", raiz->info, raiz->fb, nivel, raiz->alt);
        em_ordem(raiz->dir, nivel+1);
    }
}

void pre_ordem(NO* raiz, int nivel){
    if(raiz != NULL){
        printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
        pre_ordem(raiz->esq, nivel+1);
        pre_ordem(raiz->dir, nivel+1);
    }
}

void pos_ordem(NO* raiz, int nivel){
    if(raiz != NULL){
        pos_ordem(raiz->esq, nivel+1);
        pos_ordem(raiz->dir, nivel+1);
        printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
    }
}

void imprime(AVL* raiz){
    if(raiz == NULL) return;
    if(estaVazia(raiz)){
        printf("Arvore Vazia!\n");
        return;
    }
    //printf("\nEm Ordem: [INFO, FB, NIVEL]\n");
    printf("\nEm Ordem: [INFO, FB, NIVEL, altura]\n");
    em_ordem(*raiz, 0);
    //printf("\nPre Ordem: ");
    pre_ordem(*raiz, 0);
    //printf("\nPos Ordem: ");
    pos_ordem(*raiz, 0);
    printf("\n");
}

/* Funcao para esperar resposta do usuario e depois limpar*/
void aguardaLimpa(){
    getchar();
    printf("\n\nAperte qualquer tecla para continuar\n");
    getchar();
    system("clear");
}

void contador(NO* raiz, int nivel, int *cont){
    if(raiz != NULL){
        contador(raiz->esq, nivel+1, cont);
        (*cont)++;
        contador(raiz->dir, nivel+1, cont);
    }
}
```

codigos/ex11/ex11.c

```
#include <stdio.h>
#include <stdlib.h>
#include "ex11.h"

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


// Assuming AVL and other function declarations here

int main() {
    int *cont =
```

```c
        (int*)malloc(sizeof(int));
    int escolha, elem, busca;
    AVL *avl;
    printf("O 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 - Mostrar a quantidade de nos na AVL\n7 - Destruir a AVL\n8 - Sair\n");

    scanf("%d", &escolha);

    while (escolha != 8) {
        if (escolha == 1) {
            avl = criaAVL();
            printf("Arvore criada!");
            aguardaLimpa();
        } else if (escolha == 2) {
            printf("Digite o elemento que deseja inserir: ");
            scanf("%d", &elem);
            insereElem(avl, elem);
            aguardaLimpa();
        } else if (escolha == 3) {
            printf("Digite o elemento que deseja consultar: ");
            scanf("%d", &elem);
            busca = pesquisa(avl, elem);
            if (busca == 1)
                printf("O numero esta na arvore!");
            else
                printf("O numero nao esta na arvore!");
            aguardaLimpa();
        } else if (escolha == 4) {
            printf("Digite o elemento que deseja remover: ");
            scanf("%d", &elem);
            removeElem(avl, elem);
            aguardaLimpa();
        } else if (escolha == 5) {
            imprime(avl);
            aguardaLimpa();
        } else if (escolha == 6) {
            *cont = 0;
            contador(*avl, 0, cont);
            printf("O numero de nos eh: %d!", *cont);
            aguardaLimpa();
        } else if (escolha == 7) {
            destroiAVL(avl);
            printf("Arvore destruida!");
            aguardaLimpa();
        } else {
            printf("\nErro!\n\n");
        }

        printf("O 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");

        scanf("%d", &escolha);
    }

    free(cont);  // Free the allocated memory

    return 0;
}
```

codigos/ex11/main.c

```
all: ex11.o
  gcc ex11.o main.c -o main

ex11.o: ex11.h ex11.c
  gcc -c ex11.c

clean:
  rm -f ex11.o main
```

codigos/ex11/Makefile

1.2 TAD Novo

Código

```c
#ifndef AVL_H
#define AVL_H

#include <stdio.h>
#include <stdlib.h>
#define MAIOR(a, b) ((a > b) ? (a) : (b))

typedef struct funcionario {
    char nome[30];
    double salario;
    int anoContratacao;
} Funcionario;

typedef struct no{
    Funcionario info;
    int fb, alt;
    struct NO* esq;
    struct NO* dir;
}NO;

typedef struct no* AVL;

NO* alocarFuncionario();
void liberarFuncionario(NO* q);
AVL* criaAVL();
void destroiRec(NO* NO);
void destroiAVL(AVL* raiz);
int estaVazia(AVL* raiz);
int altura(NO* raiz);
int FB(NO* raiz);
void avl_RotDir(NO** raiz);
void avl_RotEsq(NO** raiz);
void avl_RotEsqDir(NO** raiz);
void avl_RotDirEsq(NO** raiz);
void avl_RotEsqDir2(NO** raiz);
void avl_RotDirEsq2(NO** raiz);
void avl_AuxFE(NO **raiz);
void avl_AuxFD(NO **raiz);
int insereRec(NO** raiz, double salario, char *nome, int ano_contratacao);
int insereElem(AVL* raiz, double salario, char *nome, int ano_contratacao);
int pesquisaRec(NO** raiz, double salario);
int pesquisa(AVL* raiz, double salario);
int removeRec(NO** raiz, double salario);
int removeElem(AVL* raiz, double salario);
void em_ordem(NO* raiz, int nivel);
void pre_ordem(NO* raiz, int nivel);
void pos_ordem(NO* raiz, int nivel);
void imprime(AVL* raiz);
void aguardaLimpa();
Funcionario maior(AVL* raiz);
Funcionario menor(AVL* raiz);

#endif
```

codigos/ex12/ex12.h

```c
#include <stdio.h>
#include <stdlib.h>
#include "ex12.h"
#include <string.h>

NO* alocarNO(){
    return (NO*) malloc (sizeof(NO));
}

void liberarNO(NO* q){
    free(q);
}

AVL* criaAVL(){
    AVL* raiz = (AVL*) malloc (sizeof(AVL));
    if(raiz != NULL)
        *raiz = NULL;
    return raiz;
}

void destroiRec(NO* no){
    if(no == NULL) return;
    destroiRec(no->esq);
    destroiRec(no->dir);
    liberarNO(no);
    no = NULL;
}

void destroiAVL(AVL* raiz){
    if(raiz != NULL){
        destroiRec(*raiz);
        free(raiz);
    }
}

int estaVazia(AVL* raiz){
    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
                (%d)..\n", raiz->info);
48          return MAIOR(altura(raiz->esq),
                altura(raiz->dir)) + 1;
49      }
50  }
51
52  int FB(NO* raiz){
53      if(raiz == NULL) return 0;
54      printf("Calculando FB do
            (%lf)..\n", raiz->info.salario);
55      return altura(raiz->esq) -
            altura(raiz->dir);
56  }
57
58  //Funcoes de Rotacao Simples
59  void avl_RotDir(NO** raiz){
60      printf("Rotacao Simples a
            DIREITA!\n");
61      NO *aux;
62      aux = (*raiz)->esq;
63      (*raiz)->esq = aux->dir;
64      aux->dir = *raiz;
65
66      //Acertando alturas e FB
67      //dos NOs afetados
68       (*raiz)->alt = aux->alt = -1;
69       aux->alt = altura(aux);
70       (*raiz)->alt = altura(*raiz);
71       aux->fb = FB(aux);
72       (*raiz)->fb = FB(*raiz);
73
74      *raiz = aux;
75  }
76
77  void avl_RotEsq(NO** raiz){
78      printf("Rotacao Simples a
            ESQUERDA!\n");
79      NO *aux;
80      aux = (*raiz)->dir;
81      (*raiz)->dir = aux->esq;
82      aux->esq = *raiz;
83
84      //Acertando alturas e Fatores de
            Balanceamento dos NOs afetados
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
95  //Funcoes de Rotacao Dupla
96  void avl_RotEsqDir(NO** raiz){
97      printf("Rotacao Dupla
            ESQUERDA-DIREITA!\n");
98      NO *fe; //filho esquerdo
99      NO *ffd; //filho filho direito
100
101     fe = (*raiz)->esq;
102     ffd = fe->dir;
103
104     fe->dir = ffd->esq;
105     ffd->esq = fe;
106
107     (*raiz)->esq = ffd->dir;
108     ffd->dir = *raiz;
109
110     //Acertando alturas e Fatores de
            Balanceamento dos NOs afetados
111     (*raiz)->alt = fe->alt = ffd->alt =
            -1;
112     fe->alt = altura(fe);
113     ffd->alt = altura(ffd);
114     (*raiz)->alt = altura(*raiz);
115     fe->fb = FB(fe);
116     ffd->fb = FB(ffd);
117     (*raiz)->fb = FB(*raiz);
118
119     *raiz = ffd;
120 }
121
122
123 void avl_RotDirEsq(NO** raiz){
124     printf("Rotacao Dupla
            DIREITA-ESQUERDA!\n");
125     NO* fd; //filho direito
126     NO* ffe; //filho filho esquerdo
127
128     fd = (*raiz)->dir;
129     ffe = fd->esq;
130
131     fd->esq = ffe->dir;
132     ffe->dir = fd;
133
134     (*raiz)->dir = ffe->esq;
135     ffe->esq = *raiz;
136
137     //Acertando alturas e Fatores de
            Balanceamento dos NOs afetados
138     (*raiz)->alt = fd->alt = ffe->alt =
            -1;
139     fd->alt = altura(fd);
140     ffe->alt = altura(ffe);
141     (*raiz)->alt = altura(*raiz);
142     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
            ESQUERDA-DIREITA!\n");
151     avl_RotEsq(&(*raiz)->esq);
152     avl_RotDir(raiz);
153 }
154
155 void avl_RotDirEsq2(NO** raiz){
156     printf("Rotacao Dupla 2
            DIREITA-ESQUERDA!\n");
157     avl_RotDir(&(*raiz)->dir);
158     avl_RotEsq(raiz);
159 }
160
161
162 //Funcoes Auxiliares referentes a cada
        filho
163 void avl_AuxFE(NO **raiz){
164    NO* fe;
165    fe = (*raiz)->esq;
166    if(fe->fb == +1) /* Sinais iguais e
           positivo*/
167      avl_RotDir(raiz);
168    else /* Sinais diferentes*/
169      avl_RotEsqDir(raiz);
170 }
171
172 void avl_AuxFD(NO **raiz){
173    NO* fd;
174    fd = (*raiz)->dir;
175    if(fd->fb == -1) /* Sinais iguais e
           negativos*/
176      avl_RotEsq(raiz);
177    else /* Sinais diferentes*/
178      avl_RotDirEsq(raiz);
179 }
180
181 int insereRec(NO** raiz, double
        salario, char *nome, int
        ano_contratacao){
182     int ok; //Controle para as chamadas
            recursivas
183     if(*raiz == NULL){
184         NO* novo = alocarNO();
185         if(novo == NULL) return 0;
186         novo->info.salario = salario;
                novo->fb = 0, novo->alt = 1;
187         strcpy(novo->info.nome, nome);
188         novo->info.anoContratacao =
                ano_contratacao;
189         novo->esq = NULL; novo->dir =
                NULL;
190         *raiz = novo; return 1;
191     }else{
192         if((*raiz)->info.salario ==
                salario){
193             printf("Funcionario
                    Existente!\n"); ok = 0;
194         }
195         if(salario <
                (*raiz)->info.salario){
196             ok =
                    insereRec(&(*raiz)->esq,
                    salario,nome,ano_contratacao);
197             if(ok){
198                 switch((*raiz)->fb){
199                     case -1:
200                         (*raiz)->fb =
                                0; ok = 0;
                                break;
201                     case 0:
202                         (*raiz)->fb =
                                +1;
203                         (*raiz)->alt++;
204                         break;
205                     case +1:
206                         avl_AuxFE(raiz);
                                ok = 0;
                                break;
207                 }
208             }
209         }
210         else if(salario >
                (*raiz)->info.salario){
211             ok =
                    insereRec(&(*raiz)->dir,
                    salario,nome,ano_contratacao);
212             if(ok){
213                 switch((*raiz)->fb){
214                     case +1:
215                         (*raiz)->fb =
                                0; ok = 0;
                                break;
216                     case 0:
217                         (*raiz)->fb =
                                -1;
                                (*raiz)->alt++;
                                break;
218                     case -1:
219                         avl_AuxFD(raiz);
                                ok = 0;
                                break;
220                 }
221             }
222         }
223     }
224     return ok;
225 }
226
227 int insereElem(AVL* raiz, double
        salario, char *nome, int
        ano_contratacao){
```

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

```
306                 case -1:
307                 case 0:
308                     //Acertando alturas e Fatores de Balanceamento dos NOs afetados
309                     (*raiz)->alt = -1;
310                     (*raiz)->alt = altura(*raiz);
311                     (*raiz)->fb = FB(*raiz);
312                     break;
313                 case +1:
314                     avl_AuxFE(raiz);
                        break;
315             }
316         }
317     }
318     return ok;
319 }
320
321 int removeElem(AVL* raiz, double salario){
322     if(pesquisa(raiz, salario) == 0){
323         printf("Funcionario inexistente!\n");
324         return 0;
325     }
326     return removeRec(raiz, salario);
327 }
328
329 void em_ordem(NO* raiz, int nivel){
330     if(raiz != NULL){
331         em_ordem(raiz->esq, nivel+1);
332         printf("[%.2lf, %s, %d, %d, %d, %d] ", raiz->info.salario, raiz->info.nome, raiz->info.anoContratacao, raiz->fb, nivel, raiz->alt);
333         em_ordem(raiz->dir, nivel+1);
334     }
335 }
336
337 void pre_ordem(NO* raiz, int nivel){
338     if(raiz != NULL){
339         printf("[%lf, %d, %d] ", raiz->info.salario, raiz->fb, nivel);
340         pre_ordem(raiz->esq, nivel+1);
341         pre_ordem(raiz->dir, nivel+1);
342     }
343 }
344
345 void pos_ordem(NO* raiz, int nivel){
346     if(raiz != NULL){
347         pos_ordem(raiz->esq, nivel+1);
348         pos_ordem(raiz->dir, nivel+1);
349         printf("[%lf, %d, %d] ", raiz->info.salario, raiz->fb, nivel);
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: [SALARIO, NOME, ANO DA CONTRATACAO, 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 /* Funcao para esperar resposta do usuario e depois limpar*/
368 void aguardaLimpa(){
369     getchar();
370     printf("\n\nAperte qualquer tecla para continuar\n");
371     getchar();
372     system("clear");
373 }
374
375 Funcionario maior(AVL* raiz){
376     Funcionario funcionarioMaior = {"", 0.0, 0};
377     if ((*raiz) == NULL)
378         return funcionarioMaior;
379
380     NO* aux = (*raiz);
381     while(aux->dir != NULL)
382         aux = aux->dir;
383
384     funcionarioMaior.salario = aux->info.salario;
385     funcionarioMaior.anoContratacao = aux->info.anoContratacao;
386     strcpy(funcionarioMaior.nome, aux->info.nome);
387
388     return funcionarioMaior;
389 }
390 Funcionario menor(AVL* raiz){
391     Funcionario funcionarioMenor = {"", 0.0, 0};
392
393     if ((*raiz) == NULL) {
394         return funcionarioMenor;
395     }
```

```
396
397     NO* aux = (*raiz);
398     while (aux->esq != NULL) {
399         aux = aux->esq;
400     }
401
402     funcionarioMenor.salario =
            aux->info.salario;
403     funcionarioMenor.anoContratacao =
            aux->info.anoContratacao;
404     strcpy(funcionarioMenor.nome,
            aux->info.nome);
405
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 =
           (int*)malloc(sizeof(int));
 8     int escolha, ano_contratacao, busca;
 9     double salario;
10     char* nome = (char*)malloc(30 *
           sizeof(char));
11     AVL avl = NULL;
12
13     printf("O que deseja fazer?\n1 -
           Criar AVL\n2 - Inserir um
           salario do funcionario\n"
14            "3 - Buscar um
                  funcionario\n4 - Remover
                  um funcionario\n5 -
                  Imprimir a AVL em ordem"
15            "\n6 - Imprimir as
                  informacoes do
                  Funcionario com o maior
                  salario\n"
16            "7 - Imprimir as informacoes
                  do Funcionario com o
                  menor salario\n8 -
                  Destruir a AVL\n9 -
                  Sair\n");
17
18     while (1) {
19         scanf("%d", &escolha);
20
21         if (escolha == 9) {
22             break;
23         }
24
25         if (escolha == 1) {
26             if (avl == NULL) {
27                 avl = criaAVL();
28                 printf("Arvore
                       criada!\n");
29             } else {
30                 printf("A arvore ja foi
                       criada!\n");
31             }
32             aguardaLimpa();
33         } else if (escolha == 2) {
34             if (avl == NULL) {
35                 printf("Crie a arvore
                       AVL antes de inserir
                       um funcionario!\n");
36             } else {
37                 printf("Digite o
                       salario, o nome e o
                       ano de contratacao
                       do Funcionario que
                       deseja inserir: ");
38                 scanf("%lf", &salario);
39                 getchar();
40                 fgets(nome, 30, stdin);
41                 nome[strcspn(nome,
                       "\n")] = '\0';
42                 scanf("%d",
                       &ano_contratacao);
43
44                 insereElem(avl,
                       salario, nome,
                       ano_contratacao);
45                 aguardaLimpa();
46             }
47         } else if (escolha == 3) {
48             if (avl == NULL) {
49                 printf("Crie a arvore
                       AVL antes de buscar
                       um funcionario!\n");
50             } else {
51                 printf("Digite o
                       salario do
                       funcionario que
                       deseja consultar: ");
52                 scanf("%lf", &salario);
53                 busca = pesquisa(avl,
                       salario);
54                 if (busca == 1)
55                     printf("O
                           funcionario esta
                           na arvore!\n");
56                 else
57                     printf("O
                           funcionario nao
                           esta na
                           arvore!\n");
```

```
                aguardaLimpa();
            }
        } else if (escolha == 4) {
            if (avl == NULL) {
                printf("Crie a arvore
                    AVL antes de remover
                    um funcionario!\n");
            } else {
                printf("Digite o
                    salario do
                    funcionario que
                    deseja remover: ");
                scanf("%lf", &salario);
                removeElem(avl,
                    salario);
                aguardaLimpa();
            }
        } else if (escolha == 5) {
            if (avl == NULL) {
                printf("Crie a arvore
                    AVL antes de
                    imprimir!\n");
            } else {
                imprime(avl);
                aguardaLimpa();
            }
        } else if (escolha == 6) {
            if (avl == NULL) {
                printf("Crie a arvore
                    AVL antes de
                    imprimir!\n");
            } else {
                Funcionario func =
                    maior(avl);
                printf("O funcionario
                    de maior salario
                    eh:%s\nSeu salario
                    eh de %.2lf\n"
                "Ele entrou no ano
                    %d\n",
                    func.nome,func.salario,func.anoContratacao);
                aguardaLimpa();
            }
        } else if (escolha == 7) {
            if (avl == NULL) {
                printf("Crie a arvore
                    AVL antes de
                    imprimir!\n");
            } else {
                Funcionario func =
                    menor(avl);
                printf("O funcionario
                    de menor salario
                    eh:%s\nSeu salario
                    eh de %.2lf\n"
                "Ele entrou no ano %d",
                    func.nome,func.salario,func.ano
                aguardaLimpa();
            }
        } else if (escolha == 8) {
            if (avl != NULL) {
                destroiAVL(avl);
                printf("arvore
                    destruida!\n");
            } else {
                printf("A arvore ja foi
                    destruida ou ainda
                    nao foi criada!\n");
            }
            aguardaLimpa();
        } else {
            printf("\nOpcao
                invalida!\n\n");
        }
    }

    free(cont);
    free(nome);

    return 0;
}
```

codigos/ex12/main.c

```
all: ex12.o
  gcc ex12.o main.c -o main

ex12.o: ex12.h ex12.c
  gcc -c ex12.c

clean:
  rm -f ex12.o main
```

codigos/ex12/Makefile

2 Código

https://github.com/RodrigoZonzin/labaeds2