Lista 9 de Laboratório de Programação II
Nome : Messias Feres Curi Melo
Matrícula : 2022003764

Problema 1
1.1

main.c

atv1.1 > C main.c > ...

```c
#include <stdio.h>
#include "AVL.h"

int main(){
    AVL* A;
    int escolha, resultado, elemento, nivel;

    printf("-=x Menu x=-\n");
    printf("1 - Criar AVL\n");
    printf("2 - Inserir um elemento\n");
    printf("3 - Buscar um elemento\n");
    printf("4 - Remover um elemento\n");
    printf("5 - Imprimir a AVL em ordem\n");
    printf("6 - Mostrar a quantidade de nós na AVL\n");
    printf("7 - Destruir a AVL\n");
    printf("8 - Sair\n");

    do{
        printf("\nInsira sua escolha: ");
        scanf("%d", &escolha);
        switch(escolha){
            case 1:
                A = criaAVL();
                printf("-> AVL criada com sucesso!");
                break;
            case 2:
                printf("- Insira o elemento a ser adicionado: ");
                scanf("%d", &elemento);
                resultado = insereElem(A, elemento);
                if(resultado){
                    printf("-> Elemento adicionado com sucesso!");
                }else{
                    printf("-> Elemento não foi adicionado!");
                }
                break;
            case 3:
                printf("- Insira o elemento a ser buscado: ");
                scanf("%d", &elemento);
                resultado = pesquisa(A, elemento);
```

```c
                if(resultado){
                    printf("-> Elemento encontrado!");
                }else{
                    printf("-> Elemento não encontrado!");
                }
                break;
            case 4:
                printf("- Insira o elemento a ser removido: ");
                scanf("%d", &elemento);
                resultado = removeElem(A, elemento);
                if(resultado){
                    printf("-> Elemento removido com sucesso!!");
                }else{
                    printf("-> Elemento não foi removido!");
                }
                break;
            case 5:
                printf("- Insira o nível: ");
                scanf("%d", &nivel);
                em_ordem(*A, nivel);
                break;
            case 6:
                resultado = contaNos(A);
                printf("-> Quantidade de nós: %d", resultado);
                break;
            case 7:
                destroiAVL(A);
                printf("-> AVL destruida com sucesso!");
                break;
            default:
                break;
        }
    }while(escolha < 8);

    return 0;
}
```

AVL.h

atv1.1 > C AVL.h > ...

```c
/*---------------- File: AVL.h  ------------------------+
|Arvore AVL                                             |
|                                                       |
|                                                       |
| Implementado por Guilherme C. Pena em 23/10/2023      |
+-------------------------------------------------------+ */

#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(){
    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;
```

```c
    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;
    return (*raiz == NULL);
}

//Calcula FB
int altura(NO* raiz){
    if(raiz == NULL) return 0;
    if(raiz->alt > 0)
        return raiz->alt;
    else{
        //printf("Calculando altura do (%d)..\n", raiz->info);
        return MAIOR(altura(raiz->esq), altura(raiz->dir)) + 1;
    }
}

int FB(NO* raiz){
    if(raiz == NULL) return 0;
    printf("Calculando FB do (%d)..\n", raiz->info);
    return altura(raiz->esq) - altura(raiz->dir);
}

//Funcoes de Rotacao Simples
void avl_RotDir(NO** raiz){
```

```
77      printf("Rotacao Simples a DIREITA!\n");
78      NO *aux;
79      aux = (*raiz)->esq;
80      (*raiz)->esq = aux->dir;
81      aux->dir = *raiz;
82
83      //Acertando alturas e FB
84      //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  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     //Acertando alturas e Fatores de Balanceamento dos NOs afetados
102      (*raiz)->alt = aux->alt = -1;
103      aux->alt = altura(aux);
104      (*raiz)->alt = altura(*raiz);
105      aux->fb = FB(aux);
106      (*raiz)->fb = FB(*raiz);
107
108     *raiz = aux;
109 }
110
111
112 //Funcoes de Rotacao Dupla
113 void avl_RotEsqDir(NO** raiz){
```

```
114     printf("Rotacao Dupla ESQUERDA-DIREITA!\n");
115     NO *fe; //filho esquerdo
116     NO *ffd; //filho filho direito
117 
118     fe = (*raiz)->esq;
119     ffd = fe->dir;
120 
121     fe->dir = ffd->esq;
122     ffd->esq = fe;
123 
124     (*raiz)->esq = ffd->dir;
125     ffd->dir = *raiz;
126 
127     //Acertando alturas e Fatores de Balanceamento dos NOs afetados
128     (*raiz)->alt = fe->alt = ffd->alt = -1;
129     fe->alt = altura(fe);
130     ffd->alt = altura(ffd);
131     (*raiz)->alt = altura(*raiz);
132     fe->fb = FB(fe);
133     ffd->fb = FB(ffd);
134     (*raiz)->fb = FB(*raiz);
135 
136     *raiz = ffd;
137 }
138 
139 
140 void avl_RotDirEsq(NO** raiz){
141     printf("Rotacao Dupla DIREITA-ESQUERDA!\n");
142     NO* fd; //filho direito
143     NO* ffe; //filho filho esquerdo
144 
145     fd = (*raiz)->dir;
146     ffe = fd->esq;
147 
148     fd->esq = ffe->dir;
149     ffe->dir = fd;
```

```c

    (*raiz)->dir = ffe->esq;
    ffe->esq = *raiz;

    //Acertando alturas e Fatores de Balanceamento dos NOs afetados
    (*raiz)->alt = fd->alt = ffe->alt = -1;
    fd->alt = altura(fd);
    ffe->alt = altura(ffe);
    (*raiz)->alt = altura(*raiz);
    fd->fb = FB(fd);
    ffe->fb = FB(ffe);
    (*raiz)->fb = FB(*raiz);

    *raiz = ffe;
}

void avl_RotEsqDir2(NO** raiz){
    printf("Rotacao Dupla 2 ESQUERDA-DIREITA!\n");
    avl_RotEsq(&(*raiz)->esq);
    avl_RotDir(raiz);
}

void avl_RotDirEsq2(NO** raiz){
    printf("Rotacao Dupla 2 DIREITA-ESQUERDA!\n");
    avl_RotDir(&(*raiz)->dir);
    avl_RotEsq(raiz);
}


//Funcoes Auxiliares referentes a cada filho
void avl_AuxFE(NO **raiz){
   NO* fe;
   fe = (*raiz)->esq;
   if(fe->fb == +1) /* Sinais iguais e positivo*/
     avl_RotDir(raiz);
   else /* Sinais diferentes*/
     avl_RotEsqDir(raiz);
```

```c
}

void avl_AuxFD(NO **raiz){
   NO* fd;
   fd = (*raiz)->dir;
   if(fd->fb == -1) /* Sinais iguais e negativos*/
     avl_RotEsq(raiz);
   else /* Sinais diferentes*/
     avl_RotDirEsq(raiz);
}

int insereRec(NO** raiz, int elem){
    int ok; //Controle para as chamadas recursivas
    if(*raiz == NULL){
        NO* novo = alocarNO();
        if(novo == NULL) return 0;
        novo->info = elem; novo->fb = 0, novo->alt = 1;
        novo->esq = NULL; novo->dir = NULL;
        *raiz = novo; return 1;
    }else{
        if((*raiz)->info == elem){
            printf("Elemento Existente!\n"); ok = 0;
        }
        if(elem < (*raiz)->info){
            ok = insereRec(&(*raiz)->esq, elem);
            if(ok){
                switch((*raiz)->fb){
                    case -1:
                        (*raiz)->fb = 0; ok = 0; break;
                    case 0:
                        (*raiz)->fb = +1;
                        (*raiz)->alt++;
                        break;
                    case +1:
                        avl_AuxFE(raiz); ok = 0; break;
                }
            }
```

```
224         }
225         else if(elem > (*raiz)->info){
226             ok = insereRec(&(*raiz)->dir, elem);
227             if(ok){
228                 switch((*raiz)->fb){
229                     case +1:
230                         (*raiz)->fb = 0; ok = 0; break;
231                     case 0:
232                         (*raiz)->fb = -1; (*raiz)->alt++; break;
233                     case -1:
234                         avl_AuxFD(raiz); ok = 0; break;
235                 }
236             }
237         }
238     }
239     return ok;
240 }
241
242 int insereElem(AVL* raiz, int elem){
243     if(raiz == NULL) return 0;
244     return insereRec(raiz, elem);
245 }
246
247 int pesquisaRec(NO** raiz, int elem){
248     if(*raiz == NULL) return 0;
249     if((*raiz)->info == elem) return 1;
250     if(elem < (*raiz)->info)
251         return pesquisaRec(&(*raiz)->esq, elem);
252     else
253         return pesquisaRec(&(*raiz)->dir, elem);
254 }
255
256 int pesquisa(AVL* raiz, int elem){
257     if(raiz == NULL) return 0;
258     if(estaVazia(raiz)) return 0;
259     return pesquisaRec(raiz, elem);
260 }
```

```c

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);
        }
```

```
298             return 1;
299         }else if(elem < (*raiz)->info){
300             ok = removeRec(&(*raiz)->esq, elem);
301             if(ok){
302                 switch((*raiz)->fb){
303                     case +1:
304                     case 0:
305                         //Acertando alturas e Fatores de Balanceamento dos NOs afetados
306                         (*raiz)->alt = -1;
307                         (*raiz)->alt = altura(*raiz);
308                         (*raiz)->fb = FB(*raiz);
309                         break;
310                     case -1:
311                         avl_AuxFD(raiz); break;
312                 }
313             }
314         }
315         else{
316             ok = removeRec(&(*raiz)->dir, elem);
317             if(ok){
318                 switch((*raiz)->fb){
319                     case -1:
320                     case 0:
321                         //Acertando alturas e Fatores de Balanceamento dos NOs afetados
322                         (*raiz)->alt = -1;
323                         (*raiz)->alt = altura(*raiz);
324                         (*raiz)->fb = FB(*raiz);
325                         break;
326                     case +1:
327                         avl_AuxFE(raiz); break;
328                 }
329             }
330         }
331         return ok;
332     }
333
334     int removeElem(AVL* raiz, int elem){
```

```
335     if(pesquisa(raiz, elem) == 0){
336         printf("Elemento inexistente!\n");
337         return 0;
338     }
339     return removeRec(raiz, elem);
340 }
341 
342 void em_ordem(NO* raiz, int nivel){
343     if(raiz != NULL){
344         em_ordem(raiz->esq, nivel+1);
345         //printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
346         printf("[%d, %d, %d, %d] ", raiz->info, raiz->fb, nivel, raiz->alt);
347         em_ordem(raiz->dir, nivel+1);
348     }
349 }
350 
351 void pre_ordem(NO* raiz, int nivel){
352     if(raiz != NULL){
353         printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
354         pre_ordem(raiz->esq, nivel+1);
355         pre_ordem(raiz->dir, nivel+1);
356     }
357 }
358 
359 void pos_ordem(NO* raiz, int nivel){
360     if(raiz != NULL){
361         pos_ordem(raiz->esq, nivel+1);
362         pos_ordem(raiz->dir, nivel+1);
363         printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
364     }
365 }
366 
367 int contaNosRec(NO* raiz) {
368     if (raiz == NULL) {
369         return 0;
370     }
371     return 1 + contaNosRec(raiz->esq) + contaNosRec(raiz->dir);
```

```c
    return 1 + contaNosRec(raiz->esq) + contaNosRec(raiz->dir);
}

int contaNos(AVL* raiz) {
    if (raiz == NULL || estaVazia(raiz)) {
        return 0;
    }
    return contaNosRec(*raiz);
}

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");
}


#endif
```

Console

```
messiasfcm@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/Roteiro 9/atv1.1$ gcc main.c AVL.h -o tp1
messiasfcm@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/Roteiro 9/atv1.1$ ./tp1
-=x Menu x=-
1 - Criar AVL
2 - Inserir um elemento
3 - Buscar um elemento
4 - Remover um elemento
5 - Imprimir a AVL em ordem
6 - Mostrar a quantidade de nós na AVL
7 - Destruir a AVL
8 - Sair

Insira sua escolha: 1
-> AVL criada com sucesso!
Insira sua escolha: 2
- Insira o elemento a ser adicionado: 10
-> Elemento adicionado com sucesso!
Insira sua escolha: 2
- Insira o elemento a ser adicionado: 18
-> Elemento adicionado com sucesso!
Insira sua escolha: 5
- Insira o nível: 0
[10, -1, 0, 2] [18, 0, 1, 1]
Insira sua escolha: 3
- Insira o elemento a ser buscado: 18
-> Elemento encontrado!
Insira sua escolha: 6
-> Quantidade de nós: 2
Insira sua escolha: 4
- Insira o elemento a ser removido: 18
Caso 1: Liberando 18..
Calculando FB do (10)..
-> Elemento removido com sucesso!!
Insira sua escolha: 6
-> Quantidade de nós: 1
Insira sua escolha: 7
-> AVL destruida com sucesso!
Insira sua escolha: 8
messiasfcm@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/Roteiro 9/atv1.1$
```

1.2

main.c

atv1.2 > C main.c > main()

```c
#include <stdio.h>
#include "AVL.h"

int main(){
    AVL* A;
    char nome[100];
    int escolha, resultado, anoDeContratacao, nivel;
    double salario;

    printf("-=x Menu x=-\n");
    printf("1 - Criar AVL\n");
    printf("2 - Inserir um Funcionário pelo salário\n");
    printf("3 - Buscar um Funcionário pelo salario e imprimir suas informações\n");
    printf("4 - Remover um Funcionário pelo nome\n");
    printf("5 - Imprimir a AVL em ordem\n");
    printf("6 - Imprimir as informações do Funcionário com o maior salário\n");
    printf("--> Qual a complexidade dessa operação?\n");
    printf("7 - Imprimir as informações do Funcionário com o menor salário\n");
    printf("--> Qual a complexidade dessa operação?\n");
    printf("8 - Destruir a AVL\n");
    printf("9 - Sair\n");

    do{
        printf("\nInsira sua escolha: ");
        scanf("%d", &escolha);
        switch(escolha){
            case 1:
                A = criaAVL();
                printf("-> AVL criada com sucesso!");
                break;
            case 2:
                printf("Insira o nome do funcionário: ");
                scanf("%s", nome);
                printf("Insira o salário do funcionário: ");
                scanf("%lf", &salario);
                printf("Insira o ano de contratação do funcionário: ");
                scanf("%d", &anoDeContratacao);
                resultado = insereFuncionario(A, nome, salario, anoDeContratacao);
                if(resultado){
```

```
                printf("-> Elemento adicionado com sucesso!");
            }else{
                printf("-> Elemento não foi adicionado!");
            }
            break;
        case 3:
            printf("- Insira o salário a ser buscado: ");
            scanf("%lf", &salario);
            resultado = pesquisaFuncionario(A, salario);
            if(resultado){
                printf("-> Elemento encontrado!");
            }else{
                printf("-> Elemento não encontrado!");
            }
            break;
        case 4:
            printf("- Insira o salário do funcionário a ser removido: ");
            scanf("%lf", &salario);
            resultado = removeElem(A, salario);
            if(resultado){
                printf("-> Elemento removido com sucesso!!");
            }else{
                printf("-> Elemento não foi removido!");
            }
            break;
        case 5:
            printf("- Insira o nível: ");
            scanf("%d", &nivel);
            em_ordem(*A, nivel);
            break;
        case 6:
            Funcionario maiorSalario = encontrarMaiorSalario(A);
            printf("- Funcionário com o maior salário:\n");
            printf("-> Nome: %s\n", maiorSalario.nome);
            printf("-> Salário: %.2lf reais\n", maiorSalario.salario);
            printf("-> Ano de Contratação: %d\n", maiorSalario.anoDeContratacao);
            break;
```

```
        case 7:
            Funcionario menorSalario = encontrarMenorSalario(A);
            printf("- Funcionário com o menor salário:\n");
            printf("-> Nome: %s\n", menorSalario.nome);
            printf("-> Salário: %.2lf reais\n", menorSalario.salario);
            printf("-> Ano de Contratação: %d\n", menorSalario.anoDeContratacao);
            break;
        case 8:
            destroiAVL(A);
            printf("-> AVL destruida com sucesso!");
            break;
        default:
            break;
    }
}while(escolha < 9);

return 0;
}
```

AVL.h

atv1.2 > C AVL.h > exibeInfoFuncionario(NO *, double)

```c
/*---------------- File: AVL.h  ------------------------+
|Arvore AVL                                             |
|                                                       |
|                                                       |
| Implementado por Guilherme C. Pena em 23/10/2023      |
+-------------------------------------------------------+ */

#ifndef AVL_H
#define AVL_H

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

typedef struct Funcionario {
    char nome[100];
    double salario;
    int anoDeContratacao;
}Funcionario;

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

typedef struct NO* AVL;

NO* alocarNO(char nome[], double salario, int anoDeContratacao){
    NO* novoFuncionario = (NO*) malloc (sizeof(NO));
    if(novoFuncionario != NULL){
        strcpy(novoFuncionario->funcionario.nome, nome);
        novoFuncionario->funcionario.salario = salario;
        novoFuncionario->funcionario.anoDeContratacao = anoDeContratacao;
        novoFuncionario->esq = NULL;
        novoFuncionario->dir = NULL;
    }
```

```c
    return novoFuncionario;
}

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;
    return (*raiz == NULL);
}

//Calcula FB
int altura(NO* raiz){
    if(raiz == NULL) return 0;
```

```c
    if(raiz->alt > 0)
        return raiz->alt;
    else{
        //printf("Calculando altura do (%d)..\n", raiz->info);
        return MAIOR(altura(raiz->esq), altura(raiz->dir)) + 1;
    }
}

int FB(NO* raiz){
    if(raiz == NULL) return 0;
    printf("- Calculando FB do (%lf)..\n", raiz->funcionario.salario);
    return altura(raiz->esq) - altura(raiz->dir);
}

//Funcoes de Rotacao Simples
void avl_RotDir(NO** raiz){
   printf("- Rotacao Simples a DIREITA!\n");
   NO *aux;
   aux = (*raiz)->esq;
   (*raiz)->esq = aux->dir;
   aux->dir = *raiz;

   //Acertando alturas e FB
   //dos NOs afetados
    (*raiz)->alt = aux->alt = -1;
    aux->alt = altura(aux);
    (*raiz)->alt = altura(*raiz);
    aux->fb = FB(aux);
    (*raiz)->fb = FB(*raiz);

   *raiz = aux;
}

void avl_RotEsq(NO** raiz){
   printf("- Rotacao Simples a ESQUERDA!\n");
   NO *aux;
   aux = (*raiz)->dir;
```

```
    (*raiz)->dir = aux->esq;
    aux->esq = *raiz;

    //Acertando alturas e Fatores de Balanceamento dos NOs afetados
     (*raiz)->alt = aux->alt = -1;
     aux->alt = altura(aux);
     (*raiz)->alt = altura(*raiz);
     aux->fb = FB(aux);
     (*raiz)->fb = FB(*raiz);

    *raiz = aux;
}


//Funcoes de Rotacao Dupla
void avl_RotEsqDir(NO** raiz){
    printf("- Rotacao Dupla ESQUERDA-DIREITA!\n");
    NO *fe; //filho esquerdo
    NO *ffd; //filho filho direito

    fe = (*raiz)->esq;
    ffd = fe->dir;

    fe->dir = ffd->esq;
    ffd->esq = fe;

    (*raiz)->esq = ffd->dir;
    ffd->dir = *raiz;

    //Acertando alturas e Fatores de Balanceamento dos NOs afetados
    (*raiz)->alt = fe->alt = ffd->alt = -1;
    fe->alt = altura(fe);
    ffd->alt = altura(ffd);
    (*raiz)->alt = altura(*raiz);
    fe->fb = FB(fe);
    ffd->fb = FB(ffd);
```

```
150     (*raiz)->fb = FB(*raiz);
151
152     *raiz = ffd;
153 }
154
155
156 void avl_RotDirEsq(NO** raiz){
157     printf("- Rotacao Dupla DIREITA-ESQUERDA!\n");
158     NO* fd; //filho direito
159     NO* ffe; //filho filho esquerdo
160
161     fd = (*raiz)->dir;
162     ffe = fd->esq;
163
164     fd->esq = ffe->dir;
165     ffe->dir = fd;
166
167     (*raiz)->dir = ffe->esq;
168     ffe->esq = *raiz;
169
170     //Acertando alturas e Fatores de Balanceamento dos NOs afetados
171     (*raiz)->alt = fd->alt = ffe->alt = -1;
172     fd->alt = altura(fd);
173     ffe->alt = altura(ffe);
174     (*raiz)->alt = altura(*raiz);
175     fd->fb = FB(fd);
176     ffe->fb = FB(ffe);
177     (*raiz)->fb = FB(*raiz);
178
179     *raiz = ffe;
180 }
181
182 void avl_RotEsqDir2(NO** raiz){
183     printf("- Rotacao Dupla 2 ESQUERDA-DIREITA!\n");
184     avl_RotEsq(&(*raiz)->esq);
185     avl_RotDir(raiz);
186 }
```

```
187
188 void avl_RotDirEsq2(NO** raiz){
189     printf("- Rotacao Dupla 2 DIREITA-ESQUERDA!\n");
190     avl_RotDir(&(*raiz)->dir);
191     avl_RotEsq(raiz);
192 }
193
194
195 //Funcoes Auxiliares referentes a cada filho
196 void avl_AuxFE(NO **raiz){
197    NO* fe;
198    fe = (*raiz)->esq;
199    if(fe->fb == +1) /* Sinais iguais e positivo*/
200      avl_RotDir(raiz);
201    else /* Sinais diferentes*/
202      avl_RotEsqDir(raiz);
203 }
204
205 void avl_AuxFD(NO **raiz){
206    NO* fd;
207    fd = (*raiz)->dir;
208    if(fd->fb == -1) /* Sinais iguais e negativos*/
209      avl_RotEsq(raiz);
210    else /* Sinais diferentes*/
211      avl_RotDirEsq(raiz);
212 }
213
214 int insereRec(NO** raiz, char nome[], double salario, int anoDeContratacao){
215     int ok; //Controle para as chamadas recursivas
216     if(*raiz == NULL){
217         NO* novo = alocarNO(nome, salario, anoDeContratacao);
218         if(novo == NULL) return 0;
219         novo->fb = 0,
220         novo->alt = 1;
221         novo->esq = NULL; novo->dir = NULL;
222         *raiz = novo; return 1;
223     }else{
```

```
224         if((*raiz)->funcionario.salario == salario){
225             printf("- Elemento Existente!\n"); ok = 0;
226         }
227         if(salario < (*raiz)->funcionario.salario){
228             ok = insereRec(&(*raiz)->esq, nome, salario, anoDeContratacao);
229             if(ok){
230                 switch((*raiz)->fb){
231                     case -1:
232                         (*raiz)->fb = 0; ok = 0; break;
233                     case 0:
234                         (*raiz)->fb = +1;
235                         (*raiz)->alt++;
236                         break;
237                     case +1:
238                         avl_AuxFE(raiz); ok = 0; break;
239                 }
240             }
241         }
242         else if(salario > (*raiz)->funcionario.salario){
243             ok = insereRec(&(*raiz)->dir, nome, salario, anoDeContratacao);
244             if(ok){
245                 switch((*raiz)->fb){
246                     case +1:
247                         (*raiz)->fb = 0; ok = 0; break;
248                     case 0:
249                         (*raiz)->fb = -1; (*raiz)->alt++; break;
250                     case -1:
251                         avl_AuxFD(raiz); ok = 0; break;
252                 }
253             }
254         }
255     }
256     return ok;
257 }
258
259 int insereFuncionario(AVL* raiz, char nome[], double salario, int anoDeContratacao){
260     if(raiz == NULL) return 0;
```

```c
    return insereRec(raiz, nome, salario, anoDeContratacao);
}

int exibeInfoFuncionario(NO* raiz, double salario){
    if(raiz == NULL) return 0;
    if(raiz->funcionario.salario == salario){
        printf("Funcionário encontrado!\n");
        printf("Nome: %s\n", raiz->funcionario.nome);
        printf("Salário: %.2lf\n", raiz->funcionario.salario);
        printf("Ano de Contratação: %d\n", raiz->funcionario.anoDeContratacao);
        return 1;
    }else{
        if(salario < raiz->funcionario.salario){
            return exibeInfoFuncionario(raiz->esq, salario);
        }else{
            return exibeInfoFuncionario(raiz->dir, salario);
        }
    }
}

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

int pesquisaFuncionario(AVL* raiz, double salario){
    if(raiz == NULL) return 0;
    if(estaVazia(raiz)) return 0;
    int encontrado = pesquisaRec(raiz, salario);
    if(encontrado){
        exibeInfoFuncionario(*raiz, salario);
    }else{
        printf("- Funcionário não encontrado!\n");
```

```
298         }
299         return encontrado;
300     }
301 
302     int removeRec(NO** raiz, double salario){
303         if(*raiz == NULL) return 0;
304         int ok;
305         if((*raiz)->funcionario.salario == salario){
306             NO* aux;
307             if((*raiz)->esq == NULL && (*raiz)->dir == NULL){
308                 //Caso 1 - NO sem filhos
309                 printf("- Caso 1: Liberando %lf..\n", (*raiz)->funcionario.salario);
310                 liberarNO(*raiz);
311                 *raiz = NULL;
312             }else if((*raiz)->esq == NULL){
313                 //Caso 2.1 - Possui apenas uma subarvore direita
314                 printf("- Caso 2.1: Liberando %lf..\n", (*raiz)->funcionario.salario);
315                 aux = *raiz;
316                 *raiz = (*raiz)->dir;
317                 liberarNO(aux);
318             }else if((*raiz)->dir == NULL){
319                 //Caso 2.2 - Possui apenas uma subarvore esquerda
320                 printf("- Caso 2.2: Liberando %lf..\n", (*raiz)->funcionario.salario);
321                 aux = *raiz;
322                 *raiz = (*raiz)->esq;
323                 liberarNO(aux);
324             }else{
325                 //Caso 3 - Possui as duas subarvores (esq e dir)
326                 //Duas estrategias:
327                 //3.1 - Substituir pelo NO com o MAIOR valor da subarvore esquerda
328                 //3.2 - Substituir pelo NO com o MENOR valor da subarvore direita
329                 printf("- Caso 3: Liberando %lf..\n", (*raiz)->funcionario.salario);
330                 //Estrategia 3.1:
331                 NO* Filho = (*raiz)->esq;
332                 while(Filho->dir != NULL)//Localiza o MAIOR valor da subarvore esquerda
333                     Filho = Filho->dir;
334                 (*raiz)->funcionario.salario = Filho->funcionario.salario;
```

```
335             Filho->funcionario.salario = salario;
336             return removeRec(&(*raiz)->esq, salario);
337         }
338         return 1;
339     }else if(salario < (*raiz)->funcionario.salario){
340         ok = removeRec(&(*raiz)->esq, salario);
341         if(ok){
342             switch((*raiz)->fb){
343                 case +1:
344                 case 0:
345                     //Acertando alturas e Fatores de Balanceamento dos NOs afetados
346                     (*raiz)->alt = -1;
347                     (*raiz)->alt = altura(*raiz);
348                     (*raiz)->fb = FB(*raiz);
349                     break;
350                 case -1:
351                     avl_AuxFD(raiz); break;
352             }
353         }
354     }
355     else{
356         ok = removeRec(&(*raiz)->dir, salario);
357         if(ok){
358             switch((*raiz)->fb){
359                 case -1:
360                 case 0:
361                     //Acertando alturas e Fatores de Balanceamento dos NOs afetados
362                     (*raiz)->alt = -1;
363                     (*raiz)->alt = altura(*raiz);
364                     (*raiz)->fb = FB(*raiz);
365                     break;
366                 case +1:
367                     avl_AuxFE(raiz); break;
368             }
369         }
370     }
```

```
371     return ok;
372 }
373
374 int removeElem(AVL* raiz, double salario){
375     if(pesquisaFuncionario(raiz, salario) == 0){
376         printf("- Elemento inexistente!\n");
377         return 0;
378     }
379     return removeRec(raiz, salario);
380 }
381
382 void em_ordem(NO* raiz, int nivel){
383     if(raiz != NULL){
384         em_ordem(raiz->esq, nivel+1);
385         //printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
386         printf("- [%s, %lf, %d, %d, %d]\n", raiz->funcionario.nome, raiz->funcionario.salario, raiz->fb, nivel, raiz->alt);
387         em_ordem(raiz->dir, nivel+1);
388     }
389 }
390
391 void pre_ordem(NO* raiz, int nivel){
392     if(raiz != NULL){
393         printf("- [%s, %lf, %d, %d]\n",  raiz->funcionario.nome, raiz->funcionario.salario, raiz->fb, nivel);
394         pre_ordem(raiz->esq, nivel+1);
395         pre_ordem(raiz->dir, nivel+1);
396     }
397 }
398
399 void pos_ordem(NO* raiz, int nivel){
400     if(raiz != NULL){
401         pos_ordem(raiz->esq, nivel+1);
402         pos_ordem(raiz->dir, nivel+1);
403         printf("- [%s, %lf, %d, %d]\n ",  raiz->funcionario.nome, raiz->funcionario.salario, raiz->fb, nivel);
404     }
405 }
406
407 int contaNosRec(NO* raiz) {
```

```
408     if (raiz == NULL) {
409         return 0;
410     }
411     return 1 + contaNosRec(raiz->esq) + contaNosRec(raiz->dir);
412 }
413
414 int contaNos(AVL* raiz) {
415     if (raiz == NULL || estaVazia(raiz)) {
416         return 0;
417     }
418     return contaNosRec(*raiz);
419 }
420
421 NO* encontrarMaiorSalarioRec(NO* raiz) {
422     if (raiz == NULL) {
423         return NULL;
424     }
425     NO* noMaiorSalario = raiz;
426     while (noMaiorSalario->dir != NULL) {
427         noMaiorSalario = noMaiorSalario->dir;
428     }
429     return noMaiorSalario;
430 }
431
432 Funcionario encontrarMaiorSalario(AVL* raiz){
433     NO* noMaiorSalario = encontrarMaiorSalarioRec(*raiz);
434     return noMaiorSalario->funcionario;
435 }
436
437 NO* encontrarMenorSalarioRec(NO* raiz) {
438     if (raiz == NULL) {
439         return NULL;
440     }
441     NO* noMenorSalario = raiz;
```

```
442     while (noMenorSalario->esq != NULL) {
443         noMenorSalario = noMenorSalario->esq;
444     }
445     return noMenorSalario;
446 }
447
448 Funcionario encontrarMenorSalario(AVL* raiz){
449     NO* noMenorSalario = encontrarMenorSalarioRec(*raiz);
450     return noMenorSalario->funcionario;
451 }
452
453 void imprime(AVL* raiz){
454     if(raiz == NULL) return;
455     if(estaVazia(raiz)){
456         printf("- Arvore Vazia!\n");
457         return;
458     }
459     //printf("\nEm Ordem: [INFO, FB, NIVEL]\n");
460     printf("\n- Em Ordem: [INFO, FB, NIVEL, altura]\n");
461     em_ordem(*raiz, 0);
462     //printf("\nPre Ordem: "); pre_ordem(*raiz, 0);
463     //printf("\nPos Ordem: "); pos_ordem(*raiz, 0);
464     printf("\n");
465 }
466
467
468 #endif
```

Console

```
messiasfcm@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/Roteiro 9/atv1.2$ gcc main.c AVL.h -o tp2
messiasfcm@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/Roteiro 9/atv1.2$ ./tp2
-=x Menu x=-
1 - Criar AVL
2 - Inserir um Funcionário pelo salário
3 - Buscar um Funcionário pelo salario e imprimir suas informações
4 - Remover um Funcionário pelo nome
5 - Imprimir a AVL em ordem
6 - Imprimir as informações do Funcionário com o maior salário
--> Qual a complexidade dessa operação?
7 - Imprimir as informações do Funcionário com o menor salário
--> Qual a complexidade dessa operação?
8 - Destruir a AVL
9 - Sair

Insira sua escolha: 1
-> AVL criada com sucesso!
Insira sua escolha: 2
Insira o nome do funcionário: Messias
Insira o salário do funcionário: 1300
Insira o ano de contratação do funcionário: 2004
-> Elemento adicionado com sucesso!
Insira sua escolha: 2
Insira o nome do funcionário: Luisa
Insira o salário do funcionário: 450
Insira o ano de contratação do funcionário: 2004
-> Elemento adicionado com sucesso!
Insira sua escolha: 2
Insira o nome do funcionário: Helen
Insira o salário do funcionário: 1750
Insira o ano de contratação do funcionário: 2004
-> Elemento não foi adicionado!
Insira sua escolha: 3
- Insira o salário a ser buscado: 1750
Funcionário encontrado!
Nome: Helen
Salário: 1750.00
Ano de Contratação: 2004
-> Elemento encontrado!
Insira sua escolha: 5
- Insira o nível: 0
- [Luisa, 450.000000, 0, 1, 1]
- [Messias, 1300.000000, 0, 0, 2]
- [Helen, 1750.000000, 0, 1, 1]

Insira sua escolha: 6
- Funcionário com o maior salário:
-> Nome: Helen
-> Salário: 1750.00 reais
-> Ano de Contratação: 2004

Insira sua escolha: 7
- Funcionário com o menor salário:
-> Nome: Luisa
-> Salário: 450.00 reais
-> Ano de Contratação: 2004

Insira sua escolha: 8
-> AVL destruida com sucesso!
Insira sua escolha: 9
messiasfcm@MessiasFCM:/mnt/c/Users/Messi/OneDrive/Área de Trabalho/Roteiro 9/atv1.2$
```