Roteiro 9

Sávio Francisco - 202050045

1 Árvores AVL

1.1 Implementação utilizando o código fornecido

1.1.1 Implementação (.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;
    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){
   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);
    (*raiz)->fb = FB(*raiz);
```

```
    *raiz = ffd;
}


void avl_RotDirEsq(NO** raiz){

    printf("Rotacao Dupla DIREITA-ESQUERDA!\n");

    NO* fd; //filho direito
    NO* ffe; //filho filho esquerdo

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

    fd->esq = ffe->dir;
    ffe->dir = fd;

    (*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);
}

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;
                    }
                }
            }
            else if(elem > (*raiz)->info){
                ok = insereRec(&(*raiz)->dir, 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_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);
        }
    }

    int qtd_elementos(NO *raiz, int *cont){
```

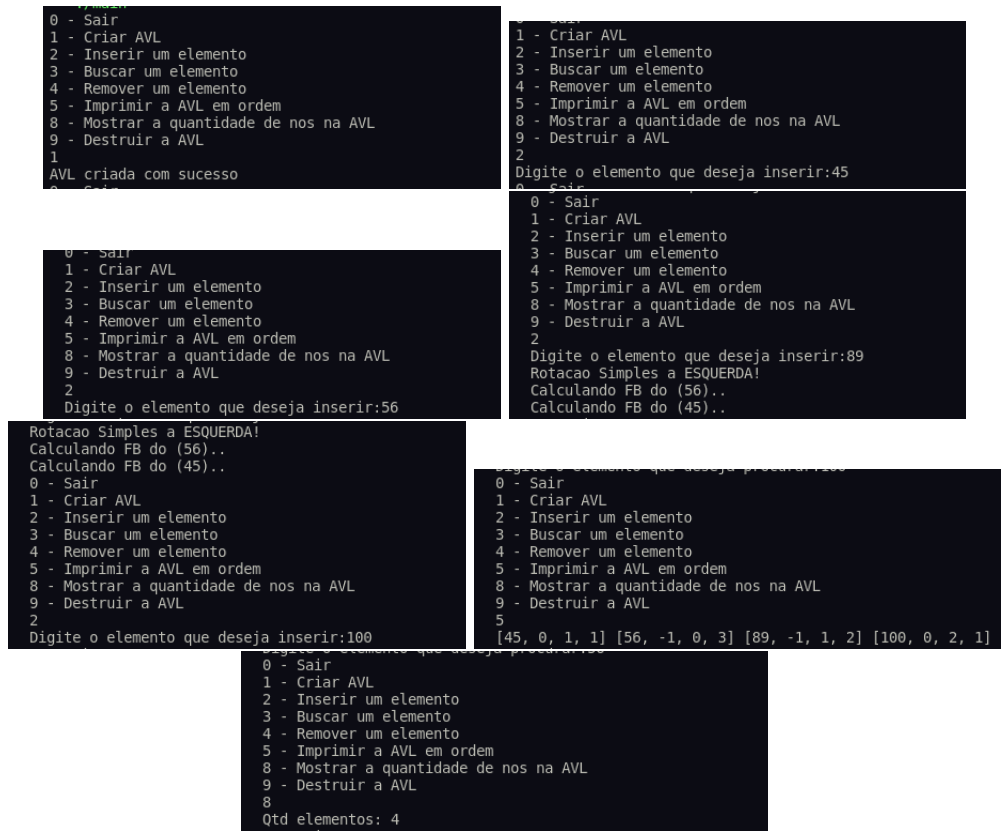
```
    if(raiz != NULL){
        (*cont)++;
        qtd_elementos(raiz->esq, cont);
        qtd_elementos(raiz->dir, cont);
    }

    return *cont;
}

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
```

1.1.2 Implementação (Main.c)

```
#include "Avl.h"
#define endl printf("\n")

int main(){

    AVL *A;
    int op;
    int *cont = (int*) calloc(1,sizeof(int));

    do{

        int elem;

        printf("0 - Sair");
        endl;
        printf("1 - Criar AVL");
        endl;
        printf("2 - Inserir um elemento");
        endl;
        printf("3 - Buscar um elemento");
        endl;
        printf("4 - Remover um elemento");
        endl;
        printf("5 - Imprimir a AVL em ordem");
        endl;
        printf("8 - Mostrar a quantidade de nos na AVL");
        endl;
        printf("9 - Destruir a AVL");
        endl;

        scanf("%d", &op);

        switch (op) {
            case 1:
                A = criaAVL();
                printf("AVL criada com sucesso");
                endl;
                break;
            case 2:
                printf("Digite o elemento que deseja inserir:");
                scanf("%d", &elem);
                insereElem(A, elem);
                break;
            case 3:
                printf("Digite o elemento que deseja procurar:");
                scanf("%d", &elem);
```

```
                    pesquisa(A, elem);
                    break;
                case 4:
                    printf("Digite o elemento que deseja remover:");
                    scanf("%d",&elem);
                    removeElem(A, elem);
                    break;
                case 5:
                    em_ordem(*A, 0);
                    endl;
                    break;
                case 8:
                    *cont = 0;
                    printf("Qtd elementos: %d", qtd_elementos(*A, cont));
                    endl;
                    break;
                case 9:
                    printf("Destruindo arvore!");
                    endl;
                    destroiAVL(A);
                    break;
            }

        }while(op != 0);

        free(cont);
        return 0;
    }
```

1.1.3 Saída do programa:



1.2 Implementação da AVL com alterações

Implementação com campo info de cada nó represente um **Funcionário** com nome(**string**), salário(**double**) e ano de contratação(**int**).

1.2.1 Implementação (.h)

```
#ifndef AVL_H
#define AVL_H

#include <bits/types/FILE.h>
#include <stdio.h>
#include <stdlib.h>

#define MAIOR(a, b) ((a > b) ? (a) : (b))

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

}Funcionario;

typedef struct NO{
    int fb, alt;
    Funcionario funcionario;
    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;
    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 (%s)..\n", raiz->funcionario.nome);

        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);
        (*raiz)->fb = FB(*raiz);

        *raiz = ffd;
    }
```

```
void avl_RotDirEsq(NO** raiz){

    printf("Rotacao Dupla DIREITA-ESQUERDA!\n");

    NO* fd; //filho direito
    NO* ffe; //filho filho esquerdo

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

    fd->esq = ffe->dir;
    ffe->dir = fd;

    (*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);
}

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, Funcionario funcionario){

    int ok; //Controle para as chamadas recursivas

    if(*raiz == NULL){

        NO* novo = alocarNO();

        if(novo == NULL){
            return 0;
        }

        novo->funcionario = funcionario; novo->fb = 0, novo->alt = 1;
        novo->esq = NULL; novo->dir = NULL;
        *raiz = novo;
```

```
            return 1;
        }else{

            if((*raiz)->funcionario.salario == funcionario.salario){
                printf("Elemento Existente!\n");
                ok = 0;
            }
            if(funcionario.salario < (*raiz)->funcionario.salario){
                ok = insereRec(&(*raiz)->esq, funcionario);

                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;
                    }
                }
            }else if(funcionario.salario > (*raiz)->funcionario.salario){
                ok = insereRec(&(*raiz)->dir, funcionario);

                if(ok){
                    switch((*raiz)->fb){
                        case +1:
                            (*raiz)->fb = 0; ok = 0;
                            break;
                        case 0:
                            (*raiz)->fb = -1; (*raiz)->alt++;
                            break;
                        case -1:
                            avl_AuxFD(raiz); ok = 0;
                            break;
                    }
                }
            }
        }
        return ok;
    }

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

    int pesquisaRec(NO** raiz, Funcionario funcionario){

        if(*raiz == NULL) return 0;

        if((*raiz)->funcionario.salario == funcionario.salario){
            return 1;
        }

        if(funcionario.salario < (*raiz)->funcionario.salario){
            return pesquisaRec(&(*raiz)->esq, funcionario);

        }else {
            return pesquisaRec(&(*raiz)->dir, funcionario);
        }
    }

    int pesquisa(AVL* raiz, Funcionario funcionario){

        if(raiz == NULL) return 0;

        if(estaVazia(raiz)) return 0;

        return pesquisaRec(raiz, funcionario);
    }

    int removeRec(NO** raiz, Funcionario funcionario){

        if(*raiz == NULL) return 0;
```

```
int ok;

if((*raiz)->funcionario.salario == funcionario.salario){
    NO* aux;

    if((*raiz)->esq == NULL && (*raiz)->dir == NULL){
        //Caso 1 - NO sem filhos
        printf("Caso 1: Liberando %s..\n", (*raiz)->funcionario.nome);
        printf("Caso 1: Liberando %lf..\n", (*raiz)->funcionario.salario);
        printf("Caso 1: Liberando %d..\n", (*raiz)->funcionario.ano);

        liberarNO(*raiz);
        *raiz = NULL;

    }else if((*raiz)->esq == NULL){
        //Caso 2.1 - Possui apenas uma subarvore direita

        printf("Caso 2.1: Liberando %s..\n", (*raiz)->funcionario.nome);
        printf("Caso 2.1: Liberando %lf..\n", (*raiz)->funcionario.salario);
        printf("Caso 2.1: Liberando %d..\n", (*raiz)->funcionario.ano);

        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 %s..\n", (*raiz)->funcionario.nome);
        printf("Caso 2.2: Liberando %lf..\n", (*raiz)->funcionario.salario);
        printf("Caso 2.2: Liberando %d..\n", (*raiz)->funcionario.ano);

        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 %s..\n", (*raiz)->funcionario.nome);
        printf("Caso 3: Liberando %lf..\n", (*raiz)->funcionario.salario);
        printf("Caso 3: Liberando %d..\n", (*raiz)->funcionario.ano);

        //Estrategia 3.1:
        NO* Filho = (*raiz)->esq;

        //Localiza o MAIOR valor da subarvore esquerda
        while(Filho->dir != NULL){
            Filho = Filho->dir;
        }

        (*raiz)->funcionario = Filho->funcionario;
        Filho->funcionario = funcionario;

        return removeRec(&(*raiz)->esq, funcionario);
    }
    return 1;

}else if(funcionario.salario < (*raiz)->funcionario.salario){

    ok = removeRec(&(*raiz)->esq, funcionario);

    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, funcionario);

            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;
    }
    void maiorSalario(NO *raiz){

        if(raiz == NULL){
            return ;
        }

        while (raiz->dir != NULL){

            raiz = raiz->dir;
        }

        printf("Funcionario com maior salario:\n");
        printf("Nome: %s\n", raiz->funcionario.nome);
        printf("Salario: %.2f\n", raiz->funcionario.salario);
        printf("Ano de contratacao: %d\n", raiz->funcionario.ano);

    }
    void menorSalario(NO *raiz){

        if(raiz == NULL){
            return;
        }
        while (raiz ->esq != NULL){
            raiz = raiz->esq;
        }

        printf("Funcionario com menor salario:\n");
        printf("Nome: %s\n", raiz->funcionario.nome);
        printf("Salario: %.2f\n", raiz->funcionario.salario);
        printf("Ano de contratacao: %d\n", raiz->funcionario.ano);
    }

    int removeElem(AVL *raiz, Funcionario funcionario){

        if(pesquisa(raiz, funcionario) == 0){
            printf("Elemento inexistente!\n");

            return 0;
        }
        return removeRec(raiz, funcionario);
    }

    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("[%s, %lf, %d, %d, %d, %d] ", raiz->funcionario.nome, raiz->
funcionario.salario, raiz->funcionario.ano, raiz->fb, nivel, raiz->alt);
            em_ordem(raiz->dir, nivel+1);
        }
    }

    void pre_ordem(NO* raiz, int nivel){

        if(raiz != NULL){
```

```
            printf("[%s, %lf, %d, %d, %d, %d] ", raiz->funcionario.nome, raiz->
funcionario.salario, raiz->funcionario.ano, raiz->fb, nivel, raiz->alt);
            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("[%s, %lf, %d, %d, %d, %d] ", raiz->funcionario.nome, raiz->
funcionario.salario, raiz->funcionario.ano, raiz->fb, nivel, raiz->alt);
        }
    }

    int qtd_elementos(NO *raiz, int *cont){

        if(raiz != NULL){
            (*cont)++;
            qtd_elementos(raiz->esq, cont);
            qtd_elementos(raiz->dir, cont);
        }

        return *cont;
    }

    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
```

1.2.2 Implementação (Main.c)

A complexidade tanto para procurar o funcionário com o **maior** salário e o **menor**, ambos são *O(log n)*. Pois para se encontrar o maior, deve ser percorrer até o último nó folha a direita cuja a complexidade é *O(log n)* e para o menor salário o mesmo se aplica so que para a esquerda da árvore.

```
    #include "Avl.h"
    #include <string.h>

    #define endl printf("\n")

    int main(){

        AVL* A;
        Funcionario funcionario;

        int *cont = (int*) calloc(1,sizeof(int));

        char nome[100];
        double salario = 0.0;
        int op, ano;

        do{

            int elem;

            printf("0 - Sair");
            endl;
            printf("1 - Criar AVL");
            endl;
            printf("2 - Inserir um Funcion rio pelo sal rio");
            endl;
            printf("3 - Buscar um Funcion rio pelo salario e imprimir suas
informa  es");
            endl;
            printf("4 - Remover um Funcion rio pelo nome");
            endl;
            printf("5 - Imprimir a AVL em ordem");
            endl;
            printf("6 - Imprimir as informa  es do Funcion rio com o maior sal rio")
;
            endl;
            printf("7 - Imprimir as informa  es do Funcion rio com o menor sal rio")
;
            endl;
            printf("8 - Mostrar a quantidade de nos na AVL");
            endl;
            printf("9 - Destruir a AVL");
            endl;

            scanf("%d", &op);

            switch (op) {
                case 1:
                   A = criaAVL();
                   printf("AVL criada com sucesso");
                   endl;
                    break;
                case 2:

                    printf("Digite o nome do funcionario:");
                    endl;
                    while (getchar() != '\n');
                    fgets(nome, sizeof(nome), stdin);

                    printf("Digite o salario do funcionario:");
                    endl;
                    scanf("%lf", &salario);

                    printf("Digite o ano de contratacao do funcionario:");
                    endl;
                    scanf("%d", &ano);

                    strcpy(funcionario.nome, nome);

                    funcionario.salario = salario;
```

```
                    funcionario.ano = ano;

                    insereElem(A, funcionario);
                    break;
                case 3:

                    printf("Digite o nome do funcionario que deseja procurar:");
                    while (getchar() != '\n');
                    fgets(nome, sizeof(nome), stdin);

                    strcpy(funcionario.nome, nome);

                    if(pesquisa(A, funcionario)){
                        printf("Nome funcionario: %s \nSalario: %lf \nAno de contratacao
: %d", funcionario.nome, funcionario.salario, funcionario.ano);
                        endl;
                    }else{
                        printf("Funcionario nao encontrado");
                        endl;
                    }
                    break;
                case 4:
                    printf("Digite o nome do funcionario que deseja remover:");
                    while (getchar() != '\n');
                    fgets(nome, sizeof(nome), stdin);

                    strcpy(funcionario.nome, nome);

                    removeElem(A, funcionario);
                    break;
                case 5:
                    em_ordem(*A, 0);
                    endl;
                    break;
                case 6:
                    menorSalario(*A);
                    endl;
                    break;
                case 7:
                    maiorSalario(*A);
                    endl;
                    break;
                case 8:
                    *cont = 0;
                    printf("Qtd elementos: %d", qtd_elementos(*A, cont));
                    endl;
                    break;
                case 9:
                    printf("Destruindo arvore!");
                    endl;
                    destroiAVL(A);
                    break;
            }

        }while(op != 0);

        free(cont);

        return 0;
    }
```

1.2.3 Saída do programa:

```
0 - Sair
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
7 - Imprimir as informações do Funcionário com o menor salário
8 - Mostrar a quantidade de nos na AVL
9 - Destruir a AVL
1
AVL criada com sucesso
```

```
0 - Sair
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
7 - Imprimir as informações do Funcionário com o menor salário
8 - Mostrar a quantidade de nos na AVL
9 - Destruir a AVL
Digite o nome do funcionario:
rodrigo
Digite o salario do funcionario:
123.2
Digite o ano de contratacao do funcionario:
2013
```

```
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
7 - Imprimir as informações do Funcionário com o menor salário
8 - Mostrar a quantidade de nos na AVL
9 - Destruir a AVL
2
Digite o nome do funcionario:
Arim
Digite o salario do funcionario:
1233.2
Digite o ano de contratacao do funcionario:
2010
Rotacao Simples a ESQUERDA!
Calculando FB do (rodrigo
)..
Calculando FB do (Rogerio
)..
```

```
Calculando FB do (rodrigo
)..
Calculando FB do (Rogerio
)..
0 - Sair
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
7 - Imprimir as informações do Funcionário com o menor salário
8 - Mostrar a quantidade de nos na AVL
9 - Destruir a AVL
5
[Rogerio
, 123.000000, 0, 0, 1, 1] [rodrigo
, 123.200000, 2013, 0, 0, 2] [Arim
, 1233.200000, 2010, 0, 1, 1]
```

```
, 123.200000, 2013, 0, 0, 2] [Arim
, 1233.200000, 2010, 0, 1, 1]
0 - Sair
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
7 - Imprimir as informações do Funcionário com o menor salário
8 - Mostrar a quantidade de nos na AVL
9 - Destruir a AVL
6
Funcionario com menor salario:
Nome: Rogerio

Salario: 123.00
Ano de contratacao: 0
```

```
8 - Mostrar a quantidade de nos na AVL
9 - Destruir a AVL
6
Funcionario com menor salario:
Nome: Rogerio

Salario: 123.00
Ano de contratacao: 0

0 - Sair
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
7 - Imprimir as informações do Funcionário com o menor salário
8 - Mostrar a quantidade de nos na AVL
9 - Destruir a AVL
7
Funcionario com maior salario:
Nome: Arim

Salario: 1233.20
Ano de contratacao: 2010
```

```
0 - Sair
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
7 - Imprimir as informações do Funcionário com o menor salário
8 - Mostrar a quantidade de nos na AVL
9 - Destruir a AVL
8
Qtd elementos: 3
```

```
0 - Sair
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
7 - Imprimir as informações do Funcionário com o menor salário
8 - Mostrar a quantidade de nos na AVL
9 - Destruir a AVL
9
Destruindo árvore!
```