UFSJ - Ciências da Computação

Laboratório de Programação 2

Roteiro 9

Nome: Geraldo Arthur Detomi

1.1) TAD: Arvore AVL

AVL.h

```c
#ifndef AVL_H
#define AVL_H

#include <math.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);

// Calcula FB
int altura(NO *raiz);

int FB(NO *raiz);

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

void avl_RotEsq(NO **raiz);

// Funcoes de Rotacao Dupla
void avl_RotEsqDir(NO **raiz);

void avl_RotDirEsq(NO **raiz);

void avl_RotEsqDir2(NO **raiz);

void avl_RotDirEsq2(NO **raiz);

// Funcoes Auxiliares referentes a cada filho
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);
```

```
66
67 void pre_ordem(NO *raiz, int nivel);
68
69 void pos_ordem(NO *raiz, int nivel);
70
71 void imprime(AVL *raiz);
72
73 #endif
```

AVL.c

```
1  #include "AVL.h"
2
3  NO *alocarNO() { return (NO *)malloc(sizeof(NO)); }
4
5  void liberarNO(NO *q) { free(q); }
6
7  AVL *criaAVL() {
8    AVL *raiz = (AVL *)malloc(sizeof(AVL));
9    if (raiz != NULL)
10     *raiz = NULL;
11   return raiz;
12 }
13
14 void destroiRec(NO *no) {
15   if (no == NULL)
16     return;
17   destroiRec(no->esq);
18   destroiRec(no->dir);
19   liberarNO(no);
20   no = NULL;
21 }
22
23 void destroiAVL(AVL *raiz) {
24   if (raiz != NULL) {
25     destroiRec(*raiz);
26     free(raiz);
27   }
28 }
29
30 int estaVazia(AVL *raiz) {
31   if (raiz == NULL)
32     return 0;
33   return (*raiz == NULL);
34 }
35
36 // Calcula FB
37 int altura(NO *raiz) {
38   if (raiz == NULL)
39     return 0;
40   if (raiz->alt > 0)
41     return raiz->alt;
42   else {
43     // printf("Calculando altura do (%d)..\n", raiz->info);
44     return MAIOR(altura(raiz->esq), altura(raiz->dir)) + 1;
45   }
46 }
47
48 int FB(NO *raiz) {
49   if (raiz == NULL)
50     return 0;
51   printf("Calculando FB do (%d)..\n", raiz->info);
52   return altura(raiz->esq) - altura(raiz->dir);
53 }
54
55 // Funcoes de Rotacao Simples
56 void avl_RotDir(NO **raiz) {
57   printf("Rotacao Simples a DIREITA!\n");
58   NO *aux;
59   aux = (*raiz)->esq;
60   (*raiz)->esq = aux->dir;
61   aux->dir = *raiz;
62
63   // Acertando alturas e FB
64   // dos NOs afetados
65   (*raiz)->alt = aux->alt = -1;
66   aux->alt = altura(aux);
67   (*raiz)->alt = altura(*raiz);
68   aux->fb = FB(aux);
69   (*raiz)->fb = FB(*raiz);
70
71   *raiz = aux;
```

```c
}

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);
```

```
153   avl_RotEsq(raiz);
154 }
155
156 // Funcoes Auxiliares referentes a cada filho
157 void avl_AuxFE(NO **raiz) {
158   NO *fe;
159   fe = (*raiz)->esq;
160   if (fe->fb == +1) /* Sinais iguais e positivo*/
161     avl_RotDir(raiz);
162   else /* Sinais diferentes*/
163     avl_RotEsqDir(raiz);
164 }
165
166 void avl_AuxFD(NO **raiz) {
167   NO *fd;
168   fd = (*raiz)->dir;
169   if (fd->fb == -1) /* Sinais iguais e negativos*/
170     avl_RotEsq(raiz);
171   else /* Sinais diferentes*/
172     avl_RotDirEsq(raiz);
173 }
174
175 int insereRec(NO **raiz, int elem) {
176   int ok; // Controle para as chamadas recursivas
177   if (*raiz == NULL) {
178     NO *novo = alocarNO();
179     if (novo == NULL)
180       return 0;
181     novo->info = elem;
182     novo->fb = 0, novo->alt = 1;
183     novo->esq = NULL;
184     novo->dir = NULL;
185     *raiz = novo;
186     return 1;
187   } else {
188     if ((*raiz)->info == elem) {
189       printf("Elemento Existente!\n");
190       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;
198           ok = 0;
199           break;
200         case 0:
201           (*raiz)->fb = +1;
202           (*raiz)->alt++;
203           break;
204         case +1:
205           avl_AuxFE(raiz);
206           ok = 0;
207           break;
208         }
209       }
210     } else if (elem > (*raiz)->info) {
211       ok = insereRec(&(*raiz)->dir, elem);
212       if (ok) {
213         switch ((*raiz)->fb) {
214         case +1:
215           (*raiz)->fb = 0;
216           ok = 0;
217           break;
218         case 0:
219           (*raiz)->fb = -1;
220           (*raiz)->alt++;
221           break;
222         case -1:
223           avl_AuxFD(raiz);
224           ok = 0;
225           break;
226         }
227       }
228     }
229   }
230   return ok;
231 }
232
233 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");
}

int get_quantidade_nos(AVL *raiz) {
  if (*raiz == NULL) {
    return 0;
  }

  int qtd = 1;

  qtd += get_quantidade_nos(&(*raiz)->esq);

  qtd += get_quantidade_nos(&(*raiz)->dir);

  return qtd;
}
```

1-1.c

```
#include "AVL.h"
```

```c

#define MAX_OPTIONS 8

enum options {
  CRIAR = 0,
  INSERIR,
  BUSCAR,
  REMOVER,
  IMPRIMIR_ORDEM,
  QUANTIDADE_NOS,
  DESTRUIR,
  SAIR,
};

int get_option() {
  int opt = -1;
  do {
    printf("\tOperacoes\n");
    printf("[%d] Criar AVL, ", CRIAR);
    printf("[%d] Inserir um elemento, ", INSERIR);
    printf("[%d] Buscar um elemento, ", BUSCAR);
    printf("[%d] Remover um elemento, ", REMOVER);
    printf("[%d] Imprimir a AVL em ordem, ", IMPRIMIR_ORDEM);
    printf("[%d] Mostrar a quantidade de nós na AVL, ", QUANTIDADE_NOS);
    printf("[%d] Destruir a AVL, ", DESTRUIR);
    printf("[%d] Sair do programa \n", SAIR);

    printf("\nInsira a opção desejada: ");
    scanf("%d", &opt);
    printf("\n");

    if (opt < 0 || opt >= MAX_OPTIONS) {
      printf("Opção escolhida inválida!\n");
    }

  } while (opt < 0 || opt >= MAX_OPTIONS);

  return opt;
}

int get_valor(char *msg) {
  int value;

  printf("%s", msg);
  scanf("%d", &value);

  return value;
}

int main() {
  int opt, valor;

  AVL *arvore = NULL;

  do {
    opt = get_option();

    switch (opt) {
    case CRIAR:
      printf("Executando comando...\n");

      if (arvore == NULL) {
        arvore = criaAVL();
      } else {
        destroiAVL(arvore);
        arvore = NULL;
        arvore = criaAVL();
      }

      printf("Árvore AVL criada com sucesso!\n");
      break;
    case INSERIR:
      printf("Executando comando...\n");

      if (arvore == NULL) {
        printf("Árvore AVL não incializada impossível realizar operação..\n");
        break;
      }

      valor = get_valor("Insira um valor: ");

```

```
      if (insereElem(arvore, valor)) {
        printf("Elemento inserido na árvore com sucesso!\n");
      } else {
        printf("Falha ao inserir elemento!\n");
      }

      break;
    case BUSCAR:
      printf("Executando comando...\n");

      if (arvore == NULL) {
        printf("Árvore AVL não incializada impossível realizar operação..\n");
        break;
      }

      valor = get_valor("Insira o valor para buscar: ");

      if (pesquisa(arvore, valor)) {
        printf("Elemento está presente na árvore AVL!\n");
      } else {
        printf("Elemento não está presente na árvore AVL!\n");
      }

      break;
    case REMOVER:
      printf("Executando comando...\n");

      if (arvore == NULL) {
        printf("Árvore AVL não incializada impossível realizar operação..\n");
        break;
      }

      valor = get_valor("Elemento a se remover: ");

      if (removeElem(arvore, valor)) {
        printf("Elemento removido com sucesso!\n");
      }

      break;
    case IMPRIMIR_ORDEM:
      printf("Executando comando...\n");

      if (arvore == NULL) {
        printf("Árvore AVL não incializada impossível realizar operação..\n");
        break;
      }

      em_ordem(*arvore, 0);

      printf("Árvore imprimida com sucesso!\n");

      break;
    case QUANTIDADE_NOS:
      printf("Executando comando...\n");

      if (arvore == NULL) {
        printf("Árvore AVL não incializada impossível realizar operação..\n");
        break;
      }

      int qtd_nos = get_quantidade_nos(arvore);

      printf("A quantidade de nós é %d\n", qtd_nos);

      break;
    case DESTRUIR:
      printf("Executando comando...\n");

      if (arvore == NULL) {
        printf("Árvore AVL não incializada impossível realizar operação..\n");
        break;
      }

      destroiAVL(arvore);
      arvore = NULL;

      printf("Árvore AVL destruida com sucesso!");

      break;
    case SAIR:
      if (arvore != NULL) {
```

```
164         destroiAVL(arvore);
165         arvore = NULL;
166       }
167
168       printf("Finalizando programa! Até mais!\n");
169       break;
170     }
171   } while (opt != SAIR);
172
173   return 0;
174 }
```

Saída do terminal:

```
arthurdetomi at arthurdetomi-System-Product-Name in ~/Documents/UFSJ-Graduacao/UFSJ-2025_1/Lab_Prog_2/roteiro_9 on main*** 25-05-29 - 19:07:59
.:: ./1-1.out
       Operacoes
[0] Criar AVL, [1] Inserir um elemento, [2] Buscar um elemento, [3] Remover um elemento, [4] Imprimir a AVL em ordem, [5] Mostrar a quantidade de nós na

Insira a opção desejada: 0

Executando comando...
Arvore AVL criada com sucesso!
       Operacoes
[0] Criar AVL, [1] Inserir um elemento, [2] Buscar um elemento, [3] Remover um elemento, [4] Imprimir a AVL em ordem, [5] Mostrar a quantidade de nós na

Insira a opção desejada: 1

Executando comando...
Insira um valor: 2
Elemento inserido na árvore com sucesso!
       Operacoes
[0] Criar AVL, [1] Inserir um elemento, [2] Buscar um elemento, [3] Remover um elemento, [4] Imprimir a AVL em ordem, [5] Mostrar a quantidade de nós na

Insira a opção desejada: 1

Executando comando...
Insira um valor: 5
Elemento inserido na árvore com sucesso!
       Operacoes
[0] Criar AVL, [1] Inserir um elemento, [2] Buscar um elemento, [3] Remover um elemento, [4] Imprimir a AVL em ordem, [5] Mostrar a quantidade de nós na

Insira a opção desejada: 1

Executando comando...
Insira um valor: 7
Rotacao Simples a ESQUERDA!
Calculando FB do (5)..
Calculando FB do (2)..
Falha ao inserir elemento!
       Operacoes
[0] Criar AVL, [1] Inserir um elemento, [2] Buscar um elemento, [3] Remover um elemento, [4] Imprimir a AVL em ordem, [5] Mostrar a quantidade de nós na

Insira a opção desejada: 4

Executando comando...
[2, 0, 1, 1] [5, 0, 0, 2] [7, 0, 1, 1] Árvore imprimida com sucesso!
       Operacoes
[0] Criar AVL, [1] Inserir um elemento, [2] Buscar um elemento, [3] Remover um elemento, [4] Imprimir a AVL em ordem, [5] Mostrar a quantidade de nós na

Insira a opção desejada: 5

Executando comando...
A quantidade de nós é 3
       Operacoes
[0] Criar AVL, [1] Inserir um elemento, [2] Buscar um elemento, [3] Remover um elemento, [4] Imprimir a AVL em ordem, [5] Mostrar a quantidade de nós na

Insira a opção desejada: 7

Finalizando programa! Até mais!
```

1.2) TAD: Arvore AVL com campo info igual a struct Funcionario

AVL_FUNC.h

```
1  #ifndef AVL_FUNC_H
2  #define AVL_FUNC_H
3
4  #include <math.h>
5  #include <stdio.h>
6  #include <stdlib.h>
7  #define MAIOR(a, b) ((a > b) ? (a) : (b))
8
9  typedef struct Funcionario {
10   char nome[50];
11   double salario;
12   int ano;
13 } Funcionario;
14
15 typedef struct NO {
16   Funcionario info;
17   int fb, alt;
18   struct NO *esq;
19   struct NO *dir;
20 } NO;
21
22 typedef struct NO *AVL;
```

```
23
24 NO *alocarNO();
25
26 void liberarNO(NO *q);
27
28 AVL *criaAVL();
29
30 void destroiRec(NO *no);
31
32 void destroiAVL(AVL *raiz);
33
34 int estaVazia(AVL *raiz);
35
36 // Calcula FB
37 int altura(NO *raiz);
38
39 int FB(NO *raiz);
40
41 // Funcoes de Rotacao Simples
42 void avl_RotDir(NO **raiz);
43
44 void avl_RotEsq(NO **raiz);
45
46 // Funcoes de Rotacao Dupla
47 void avl_RotEsqDir(NO **raiz);
48
49 void avl_RotDirEsq(NO **raiz);
50
51 void avl_RotEsqDir2(NO **raiz);
52
53 void avl_RotDirEsq2(NO **raiz);
54
55 // Funcoes Auxiliares referentes a cada filho
56 void avl_AuxFE(NO **raiz);
57
58 void avl_AuxFD(NO **raiz);
59
60 int insereRec(NO **raiz, Funcionario elem);
61
62 int insereElem(AVL *raiz, Funcionario elem);
63
64 int pesquisa(AVL *raiz, Funcionario *elem);
65
66 int removeRec(NO **raiz, Funcionario elem);
67
68 int removeElem(AVL *raiz, Funcionario elem);
69
70 void em_ordem(NO *raiz, int nivel);
71
72 void pre_ordem(NO *raiz, int nivel);
73
74 void pos_ordem(NO *raiz, int nivel);
75
76 void imprime(AVL *raiz);
77
78 int get_quantidade_nos(AVL *raiz);
79
80 int buscar_elem_por_nome(AVL *raiz, char *nome, Funcionario *f);
81
82 int buscar_com_maior_salario(AVL *raiz, Funcionario *f);
83
84 int buscar_com_menor_salario(AVL *raiz, Funcionario *f);
85
86 #endif
```

AVL_FUNC.c

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

```c
  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;
  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

```

```c
  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 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.salario == elem.salario) {
      printf("Elemento Existente!\n");
      ok = 0;
    }
    if (elem.salario < (*raiz)->info.salario) {
      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.salario > (*raiz)->info.salario) {
      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, Funcionario elem) {
  if (raiz == NULL)
    return 0;
  return insereRec(raiz, elem);
}

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

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

```
259
260 int removeRec(NO **raiz, Funcionario elem) {
261   if (*raiz == NULL)
262     return 0;
263   int ok;
264   if ((*raiz)->info.salario == elem.salario) {
265     NO *aux;
266     if ((*raiz)->esq == NULL && (*raiz)->dir == NULL) {
267       // Caso 1 - NO sem filhos
268       liberarNO(*raiz);
269       *raiz = NULL;
270     } else if ((*raiz)->esq == NULL) {
271       // Caso 2.1 - Possui apenas uma subarvore direita
272       aux = *raiz;
273       *raiz = (*raiz)->dir;
274       liberarNO(aux);
275     } else if ((*raiz)->dir == NULL) {
276       // Caso 2.2 - Possui apenas uma subarvore esquerda
277       aux = *raiz;
278       *raiz = (*raiz)->esq;
279       liberarNO(aux);
280     } else {
281       // Caso 3 - Possui as duas subarvores (esq e dir)
282       // Duas estrategias:
283       // 3.1 - Substituir pelo NO com o MAIOR valor da subarvore esquerda
284       // 3.2 - Substituir pelo NO com o MENOR valor da subarvore direita
285       // Estrategia 3.1:
286       NO *Filho = (*raiz)->esq;
287       while (Filho->dir != NULL) // Localiza o MAIOR valor da subarvore esquerda
288         Filho = Filho->dir;
289       (*raiz)->info = Filho->info;
290       Filho->info = elem;
291       return removeRec(&(*raiz)->esq, elem);
292     }
293     return 1;
294   } else if (elem.salario < (*raiz)->info.salario) {
295     ok = removeRec(&(*raiz)->esq, elem);
296     if (ok) {
297       switch ((*raiz)->fb) {
298       case +1:
299       case 0:
300         // Acertando alturas e Fatores de Balanceamento dos NOs afetados
301         (*raiz)->alt = -1;
302         (*raiz)->alt = altura(*raiz);
303         (*raiz)->fb = FB(*raiz);
304         break;
305       case -1:
306         avl_AuxFD(raiz);
307         break;
308       }
309     }
310   } else {
311     ok = removeRec(&(*raiz)->dir, elem);
312     if (ok) {
313       switch ((*raiz)->fb) {
314       case -1:
315       case 0:
316         // Acertando alturas e Fatores de Balanceamento dos NOs afetados
317         (*raiz)->alt = -1;
318         (*raiz)->alt = altura(*raiz);
319         (*raiz)->fb = FB(*raiz);
320         break;
321       case +1:
322         avl_AuxFE(raiz);
323         break;
324       }
325     }
326   }
327   return ok;
328 }
329
330 int removeElem(AVL *raiz, Funcionario elem) {
331   if (pesquisa(raiz, &elem) == 0) {
332     printf("Elemento inexistente!\n");
333     return 0;
334   }
335   return removeRec(raiz, elem);
336 }
337
338 void em_ordem(NO *raiz, int nivel) {
339   if (raiz != NULL) {
```

```c
    em_ordem(raiz->esq, nivel + 1);
    // printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
    printf("[%s, %.2lf,%d, %d, %d] ", raiz->info.nome, raiz->info.salario,
           raiz->fb, nivel, raiz->alt);
    em_ordem(raiz->dir, nivel + 1);
  }
}

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");
}

int get_quantidade_nos(AVL *raiz) {
  if (*raiz == NULL) {
    return 0;
  }

  int qtd = 1;

  qtd += get_quantidade_nos(&(*raiz)->esq);

  qtd += get_quantidade_nos(&(*raiz)->dir);

  return qtd;
}

int buscar_com_maior_salario(AVL *raiz, Funcionario *f) {
  if (*raiz == NULL) {
    return 0;
  }

  if ((*raiz)->dir == NULL) {
    *f = (*raiz)->info;
    return 1;
  }

  return buscar_com_maior_salario(&(*raiz)->dir, f);
}

int buscar_com_menor_salario(AVL *raiz, Funcionario *f) {
  if (*raiz == NULL) {
    return 0;
  }

  if ((*raiz)->esq == NULL) {
    *f = (*raiz)->info;
    return 1;
  }

  return buscar_com_menor_salario(&(*raiz)->esq, f);
}

int buscar_elem_por_nome(AVL *raiz, char *nome, Funcionario *f) {
  if (*raiz == NULL) {
    return 0;
  }

  if (strcmp(nome, (*raiz)->info.nome) == 0) {
    *f = (*raiz)->info;
    return 1;
  }

  return buscar_elem_por_nome(&(*raiz)->dir, nome, f) ||
         buscar_elem_por_nome(&(*raiz)->esq, nome, f);
}
```

1-2.c

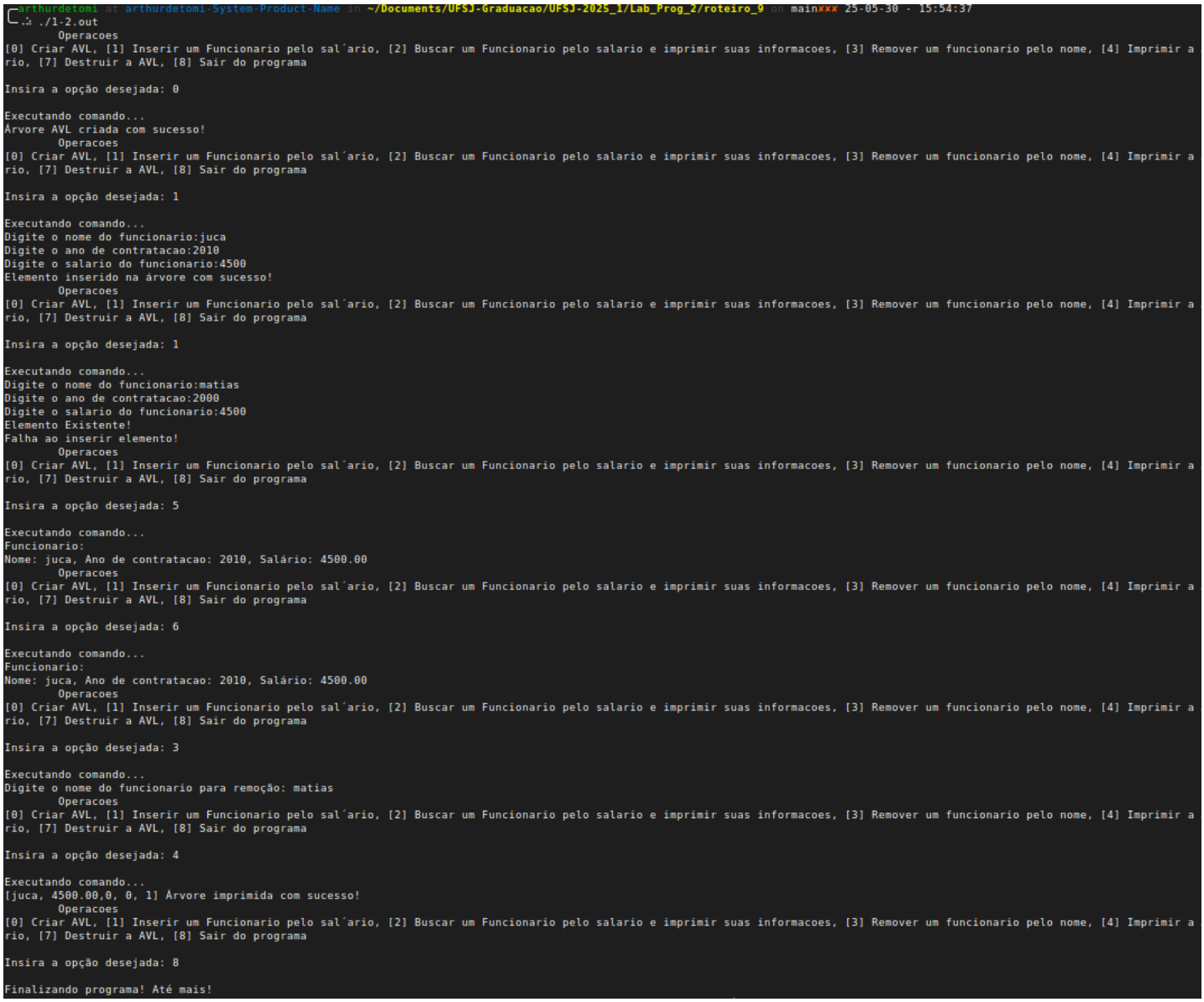
```c
#include "AVL_FUNC.h"

#define MAX_OPTIONS 9
```

```c

enum options {
  CRIAR = 0,
  INSERIR,
  BUSCAR,
  REMOVER,
  IMPRIMIR_ORDEM,
  MAIOR_SALARIO,
  MENOR_SALARIO,
  DESTRUIR,
  SAIR,
};

int get_option() {
  int opt = -1;
  do {
    printf("\tOperacoes\n");
    printf("[%d] Criar AVL, ", CRIAR);
    printf("[%d] Inserir um Funcionario pelo sal´ario, ", INSERIR);
    printf(
        "[%d] Buscar um Funcionario pelo salario e imprimir suas informacoes, ",
        BUSCAR);
    printf("[%d] Remover um funcionario pelo nome, ", REMOVER);
    printf("[%d] Imprimir a AVL em ordem, ", IMPRIMIR_ORDEM);
    printf("[%d]  Imprimir as informacoes do Funcionario com o maior salario, ",
           MAIOR_SALARIO);
    printf("[%d]  Imprimir as informacoes do Funcionario com o menor salario, ",
           MENOR_SALARIO);
    printf("[%d] Destruir a AVL, ", DESTRUIR);
    printf("[%d] Sair do programa \n", SAIR);

    printf("\nInsira a opção desejada: ");
    scanf("%d", &opt);
    printf("\n");

    if (opt < 0 || opt >= MAX_OPTIONS) {
      printf("Opção escolhida inválida!\n");
    }

  } while (opt < 0 || opt >= MAX_OPTIONS);

  return opt;
}

int get_valor(char *msg) {
  int value;

  printf("%s", msg);
  scanf("%d", &value);

  return value;
}

Funcionario get_funcionario() {
  Funcionario func;

  printf("Digite o nome do funcionario:");
  scanf("%49s", func.nome);
  printf("Digite o ano de contratacao:");
  scanf("%d", &func.ano);
  printf("Digite o salario do funcionario:");
  scanf("%lf", &func.salario);

  return func;
}

void print_funcionario(Funcionario *f) {
  printf("Funcionario:\n");
  printf("Nome: %s, Ano de contratacao: %d, Salário: %.2lf\n", f->nome, f->ano,
         f->salario);
}

double get_salario(char *msg) {
  double value;

  printf("%s", msg);
  scanf("%lf", &value);

  return value;
}

```

```c
int main() {
  int opt;

  double valor;
  Funcionario elem;

  AVL *arvore = NULL;

  do {
    opt = get_option();

    switch (opt) {
    case CRIAR:
      printf("Executando comando...\n");

      if (arvore == NULL) {
        arvore = criaAVL();
      } else {
        destroiAVL(arvore);
        arvore = NULL;
        arvore = criaAVL();
      }

      printf("Árvore AVL criada com sucesso!\n");
      break;
    case INSERIR:
      printf("Executando comando...\n");

      if (arvore == NULL) {
        printf("Árvore AVL não incializada impossível realizar operação..\n");
        break;
      }

      elem = get_funcionario();

      if (insereElem(arvore, elem)) {
        printf("Elemento inserido na árvore com sucesso!\n");
      } else {
        printf("Falha ao inserir elemento!\n");
      }

      break;
    case BUSCAR:
      printf("Executando comando...\n");

      if (arvore == NULL) {
        printf("Árvore AVL não incializada impossível realizar operação..\n");
        break;
      }

      valor = get_salario("Insira o salario do funcionario para busca: ");

      Funcionario f;
      f.salario = valor;

      if (pesquisa(arvore, &f)) {
        printf("Funcionario presente\n");
        print_funcionario(&f);
        printf("Elemento está presente na árvore AVL!\n");
      } else {
        printf("Elemento não está presente na árvore AVL!\n");
      }

      break;
    case REMOVER:
      printf("Executando comando...\n");

      if (arvore == NULL) {
        printf("Árvore AVL não incializada impossível realizar operação..\n");
        break;
      }

      char nome_pesquisa[50];
      printf("Digite o nome do funcionario para remoção: ");
      scanf("%49s", nome_pesquisa);

      int encontrado = buscar_elem_por_nome(arvore, nome_pesquisa, &elem);

      if (encontrado && removeElem(arvore, elem)) {
        printf("Elemento removido com sucesso!\n");
      }
```

```
166
167       break;
168     case IMPRIMIR_ORDEM:
169       printf("Executando comando...\n");
170
171       if (arvore == NULL) {
172         printf("Árvore AVL não inicializada impossível realizar operação..\n");
173         break;
174       }
175
176       em_ordem(*arvore, 0);
177
178       printf("Árvore imprimida com sucesso!\n");
179
180       break;
181     case MAIOR_SALARIO:
182       printf("Executando comando...\n");
183
184       if (arvore == NULL) {
185         printf("Árvore AVL não inicializada impossível realizar operação..\n");
186         break;
187       }
188
189       if (buscar_com_maior_salario(arvore, &elem)) {
190         print_funcionario(&elem);
191       } else {
192         printf("Falha ao encontrar funcionário com maior salario\n");
193       }
194
195       break;
196     case MENOR_SALARIO:
197       printf("Executando comando...\n");
198
199       if (arvore == NULL) {
200         printf("Árvore AVL não inicializada impossível realizar operação..\n");
201         break;
202       }
203
204       if (buscar_com_menor_salario(arvore, &elem)) {
205         print_funcionario(&elem);
206       } else {
207         printf("Falha ao encontrar funcionário com maior salario\n");
208       }
209
210       break;
211     case DESTRUIR:
212       printf("Executando comando...\n");
213
214       if (arvore == NULL) {
215         printf("Árvore AVL não incializada impossível realizar operação..\n");
216         break;
217       }
218
219       destroiAVL(arvore);
220       arvore = NULL;
221
222       printf("Árvore AVL destruida com sucesso!");
223
224       break;
225     case SAIR:
226       if (arvore != NULL) {
227         destroiAVL(arvore);
228         arvore = NULL;
229       }
230
231       printf("Finalizando programa! Até mais!\n");
232       break;
233     }
234   } while (opt != SAIR);
235
236   return 0;
237 }
```

Saída do terminal:



Resposta a respeito das complexidades

Como a AVL usa o salário como chave e mantém-se balanceada com as rotações, encontrar o funcionário com maior e menor salário leva tempo O(log n). Para encontrar o maior salário, basta seguir pela subárvore direita até chegar no último nó, que contém o maior valor. Já para o menor salário, é só fazer o mesmo caminho, mas seguindo sempre pela subárvore esquerda.