roteiro-09/ex01-01.h

```c
#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);
        *raiz = NULL;
    }
}

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)
```

```
57          return raiz->alt;
58      else {
59          // printf("Calculando altura do (%d)..\n", raiz->info);
60          return MAIOR(altura(raiz->esq), altura(raiz->dir)) + 1;
61      }
62  }
63
64  int FB(NO* raiz) {
65      if (raiz == NULL) return 0;
66      printf("Calculando FB do (%d)..\n", raiz->info);
67      return altura(raiz->esq) - altura(raiz->dir);
68  }
69
70  // Funcoes de Rotacao Simples
71  void avl_RotDir(NO** raiz) {
72      printf("Rotacao Simples a DIREITA!\n");
73      NO* aux;
74      aux = (*raiz)->esq;
75      (*raiz)->esq = aux->dir;
76      aux->dir = *raiz;
77
78      (*raiz)->alt = aux->alt = -1;
79      aux->alt = altura(aux);
80      (*raiz)->alt = altura(*raiz);
81      aux->fb = FB(aux);
82      (*raiz)->fb = FB(*raiz);
83
84      *raiz = aux;
85  }
86
87  void avl_RotEsq(NO** raiz) {
88      printf("Rotacao Simples a ESQUERDA!\n");
89      NO* aux;
90      aux = (*raiz)->dir;
91      (*raiz)->dir = aux->esq;
92      aux->esq = *raiz;
93
94      (*raiz)->alt = aux->alt = -1;
95      aux->alt = altura(aux);
96      (*raiz)->alt = altura(*raiz);
97      aux->fb = FB(aux);
98      (*raiz)->fb = FB(*raiz);
99
100     *raiz = aux;
101 }
102
103 void avl_RotEsqDir(NO** raiz) {
104     printf("Rotacao Dupla ESQUERDA-DIREITA!\n");
105     NO* fe;   // filho esquerdo
106     NO* ffd;  // filho filho direito
107
108     fe = (*raiz)->esq;
109     ffd = fe->dir;
110
111     fe->dir = ffd->esq;
112     ffd->esq = fe;
113
114     (*raiz)->esq = ffd->dir;
115     ffd->dir = *raiz;
```

```
116
117     (*raiz)->alt = fe->alt = ffd->alt = -1;
118     fe->alt = altura(fe);
119     ffd->alt = altura(ffd);
120     (*raiz)->alt = altura(*raiz);
121     fe->fb = FB(fe);
122     ffd->fb = FB(ffd);
123     (*raiz)->fb = FB(*raiz);
124
125     *raiz = ffd;
126 }
127
128 void avl_RotDirEsq(NO** raiz) {
129     printf("Rotacao Dupla DIREITA-ESQUERDA!\n");
130     NO* fd;   // filho direito
131     NO* ffe;  // filho filho esquerdo
132
133     fd = (*raiz)->dir;
134     ffe = fd->esq;
135
136     fd->esq = ffe->dir;
137     ffe->dir = fd;
138
139     (*raiz)->dir = ffe->esq;
140     ffe->esq = *raiz;
141
142     (*raiz)->alt = fd->alt = ffe->alt = -1;
143     fd->alt = altura(fd);
144     ffe->alt = altura(ffe);
145     (*raiz)->alt = altura(*raiz);
146     fd->fb = FB(fd);
147     ffe->fb = FB(ffe);
148     (*raiz)->fb = FB(*raiz);
149
150     *raiz = ffe;
151 }
152
153 void avl_RotEsqDir2(NO** raiz) {
154     printf("Rotacao Dupla 2 ESQUERDA-DIREITA!\n");
155     avl_RotEsq(&(*raiz)->esq);
156     avl_RotDir(raiz);
157 }
158
159 void avl_RotDirEsq2(NO** raiz) {
160     printf("Rotacao Dupla 2 DIREITA-ESQUERDA!\n");
161     avl_RotDir(&(*raiz)->dir);
162     avl_RotEsq(raiz);
163 }
164
165 void avl_AuxFE(NO** raiz) {
166     NO* fe;
167     fe = (*raiz)->esq;
168     if (fe->fb == +1) /* Sinais iguais e positivo*/
169         avl_RotDir(raiz);
170     else /* Sinais diferentes*/
171         avl_RotEsqDir(raiz);
172 }
173
174 void avl_AuxFD(NO** raiz) {
```

```c
    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;
```

```
292             (*raiz)->info = Filho->info;
293             Filho->info = elem;
294             return removeRec(&(*raiz)->esq, elem);
295         }
296         return 1;
297     } else if (elem < (*raiz)->info) {
298         ok = removeRec(&(*raiz)->esq, elem);
299         if (ok) {
300             switch ((*raiz)->fb) {
301                 case +1:
302                 case 0:
303                     // Acertando alturas e Fatores de Balanceamento dos NOs
afetados
304                     (*raiz)->alt = -1;
305                     (*raiz)->alt = altura(*raiz);
306                     (*raiz)->fb = FB(*raiz);
307                     break;
308                 case -1:
309                     avl_AuxFD(raiz);
310                     break;
311             }
312         }
313     } else {
314         ok = removeRec(&(*raiz)->dir, elem);
315         if (ok) {
316             switch ((*raiz)->fb) {
317                 case -1:
318                 case 0:
319                     // Acertando alturas e Fatores de Balanceamento dos NOs
afetados
320                     (*raiz)->alt = -1;
321                     (*raiz)->alt = altura(*raiz);
322                     (*raiz)->fb = FB(*raiz);
323                     break;
324                 case +1:
325                     avl_AuxFE(raiz);
326                     break;
327             }
328         }
329     }
330     return ok;
331 }
332
333 int removeElem(AVL* raiz, int elem) {
334     if (pesquisa(raiz, elem) == 0) {
335         printf("Elemento inexistente!\n");
336         return 0;
337     }
338     return removeRec(raiz, elem);
339 }
340
341 void em_ordem(NO* raiz, int nivel) {
342     if (raiz != NULL) {
343         em_ordem(raiz->esq, nivel + 1);
344         // printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
345         printf("[%d, %d, %d, %d] ", raiz->info, raiz->fb, nivel, raiz->alt);
346         em_ordem(raiz->dir, nivel + 1);
347     }
348 }
```

```c

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 tamanho(NO* raiz, int inicial) {
    if (raiz == NULL) return 0;

    int t = 1;
    t += tamanho(raiz->esq, 0);
    t += tamanho(raiz->dir, 0);
    return t;
}

#endif
```

roteiro-09/ex01-01.c

```c
#include "ex01-01.h"

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

enum {
    EXIT = 0,
    CREATE,
    INSERT,
    SEARCH,
    REMOVE,
    PRINT,
    SIZE,
    DESTROY
} Options;

int getOption() {
    int option;

    printf("\n====================\n");
    printf("(%d) Criar\n", CREATE);
    printf("(%d) Inserir elemento\n", INSERT);
    printf("(%d) Buscar elemento\n", SEARCH);
    printf("(%d) Remover elemento\n", REMOVE);
    printf("(%d) Imprimir em ordem\n", PRINT);
    printf("(%d) Tamanho\n", SIZE);
    printf("(%d) Destruir\n", DESTROY);
    printf("(%d) Sair\n", EXIT);
    printf("====================\n");
    printf("Operacao: ");

    scanf("%d", &option);
    printf("\n");

    return option;
}

int runMenu() {
    AVL* avl = NULL;
    int exit = 0, item;

    do {
        switch (getOption()) {
            case CREATE:
                if (avl != NULL) {
                    destroiAVL(&avl);
                }
                avl = criaAVL();
                break;

            case INSERT:
                printf("Elemento a ser inserido: ");
                scanf("%d", &item);

                if (insereElem(avl, item)) {
                    printf("Elemento inserido (%d)", item);
```

```
                } else {
                    printf("Falha ao inserir (%d)", item);
                }
                break;

            case SEARCH:
                printf("Elemento a ser pesquisado: ");
                scanf("%d", &item);

                if (pesquisa(avl, item)) {
                    printf("Elemento presente (%d)", item);
                } else {
                    printf("Elemento nao encontrado (%d)", item);
                }
                break;

            case REMOVE:
                printf("Elemento a ser removido: ");
                scanf("%d", &item);

                if (removeElem(avl, item)) {
                    printf("Elemento removido (%d)", item);
                } else {
                    printf("Falha ao remover (%d)", item);
                }
                break;

            case PRINT:
                imprime(avl);
                break;

            case SIZE:
                printf("Tamanho = %d", tamanho(*avl, 0));
                break;

            case DESTROY:
                destroiAVL(&avl);
                break;

            case EXIT:
                if (avl != NULL) {
                    destroiAVL(&avl);
                }
                printf("Programa encerrado");
                exit = 1;
                break;

            default:
                printf("Opcao desconhecida, tente novamente");
        }
        printf("\n");
    } while (!exit);
}

int main() {
    runMenu();
    return 0;
}
```

```
gabriel-dp@gabriel-dp: ~/Desktop/dev/c/lab2/roteiro-09
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-09$ ./ex01-01

====================
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Tamanho
(7) Destruir
(0) Sair
====================
Operacao: 1



====================
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Tamanho
(7) Destruir
(0) Sair
====================
Operacao: 2

Elemento a ser inserido: 5
Elemento inserido (5)

====================
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Tamanho
(7) Destruir
(0) Sair
====================
Operacao: 2

Elemento a ser inserido: 10
Elemento inserido (10)

====================
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Tamanho
(7) Destruir
(0) Sair
====================
Operacao: 2

Elemento a ser inserido: 15
Rotacao Simples a ESQUERDA!
Calculando FB do (10)..
Calculando FB do (5)..
Falha ao inserir (15)

====================
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Tamanho
(7) Destruir
(0) Sair
====================
Operacao: 5


Em Ordem: [INFO, FB, NIVEL, altura]
[5, 0, 1, 1] [10, 0, 0, 2] [15, 0, 1, 1]


====================
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Tamanho
(7) Destruir
(0) Sair
====================
Operacao: 6

Tamanho = 3

====================
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Tamanho
(7) Destruir
(0) Sair
====================
Operacao: 0

Programa encerrado
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-09$
```

roteiro-09/ex01-02.h

```c
#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 {
    char nome[50];
    int contratacao;
    double salario;
} Funcionario;

typedef struct NO {
    int fb, alt;
    Funcionario info;
    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);
        *raiz = NULL;
    }
}

int estaVazia(AVL* raiz) {
```

```c
    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;

    (*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;

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

    fe = (*raiz)->esq;
```

```
116     ffd = fe->dir;
117
118     fe->dir = ffd->esq;
119     ffd->esq = fe;
120
121     (*raiz)->esq = ffd->dir;
122     ffd->dir = *raiz;
123
124     (*raiz)->alt = fe->alt = ffd->alt = -1;
125     fe->alt = altura(fe);
126     ffd->alt = altura(ffd);
127     (*raiz)->alt = altura(*raiz);
128     fe->fb = FB(fe);
129     ffd->fb = FB(ffd);
130     (*raiz)->fb = FB(*raiz);
131
132     *raiz = ffd;
133 }
134
135 void avl_RotDirEsq(NO** raiz) {
136     printf("Rotacao Dupla DIREITA-ESQUERDA!\n");
137     NO* fd;   // filho direito
138     NO* ffe;  // filho filho esquerdo
139
140     fd = (*raiz)->dir;
141     ffe = fd->esq;
142
143     fd->esq = ffe->dir;
144     ffe->dir = fd;
145
146     (*raiz)->dir = ffe->esq;
147     ffe->esq = *raiz;
148
149     (*raiz)->alt = fd->alt = ffe->alt = -1;
150     fd->alt = altura(fd);
151     ffe->alt = altura(ffe);
152     (*raiz)->alt = altura(*raiz);
153     fd->fb = FB(fd);
154     ffe->fb = FB(ffe);
155     (*raiz)->fb = FB(*raiz);
156
157     *raiz = ffe;
158 }
159
160 void avl_RotEsqDir2(NO** raiz) {
161     printf("Rotacao Dupla 2 ESQUERDA-DIREITA!\n");
162     avl_RotEsq(&(*raiz)->esq);
163     avl_RotDir(raiz);
164 }
165
166 void avl_RotDirEsq2(NO** raiz) {
167     printf("Rotacao Dupla 2 DIREITA-ESQUERDA!\n");
168     avl_RotDir(&(*raiz)->dir);
169     avl_RotEsq(raiz);
170 }
171
172 void avl_AuxFE(NO** raiz) {
173     NO* fe;
174     fe = (*raiz)->esq;
```

```
175     if (fe->fb == +1) /* Sinais iguais e positivo*/
176         avl_RotDir(raiz);
177     else /* Sinais diferentes*/
178         avl_RotEsqDir(raiz);
179 }
180 
181 void avl_AuxFD(NO** raiz) {
182     NO* fd;
183     fd = (*raiz)->dir;
184     if (fd->fb == -1) /* Sinais iguais e negativos*/
185         avl_RotEsq(raiz);
186     else /* Sinais diferentes*/
187         avl_RotDirEsq(raiz);
188 }
189 
190 int insereRec(NO** raiz, Funcionario elem) {
191     int ok;  // Controle para as chamadas recursivas
192     if (*raiz == NULL) {
193         NO* novo = alocarNO();
194         if (novo == NULL) return 0;
195         novo->info = elem;
196         novo->fb = 0, novo->alt = 1;
197         novo->esq = NULL;
198         novo->dir = NULL;
199         *raiz = novo;
200         return 1;
201     } else {
202         if ((*raiz)->info.salario == elem.salario) {
203             printf("Elemento Existente!\n");
204             ok = 0;
205         }
206         if (elem.salario < (*raiz)->info.salario) {
207             ok = insereRec(&(*raiz)->esq, elem);
208             if (ok) {
209                 switch ((*raiz)->fb) {
210                     case -1:
211                         (*raiz)->fb = 0;
212                         ok = 0;
213                         break;
214                     case 0:
215                         (*raiz)->fb = +1;
216                         (*raiz)->alt++;
217                         break;
218                     case +1:
219                         avl_AuxFE(raiz);
220                         ok = 0;
221                         break;
222                 }
223             }
224         } else if (elem.salario > (*raiz)->info.salario) {
225             ok = insereRec(&(*raiz)->dir, elem);
226             if (ok) {
227                 switch ((*raiz)->fb) {
228                     case +1:
229                         (*raiz)->fb = 0;
230                         ok = 0;
231                         break;
232                     case 0:
233                         (*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) 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);
}

int removeRec(NO** raiz, Funcionario elem) {
    if (*raiz == NULL) return 0;
    int ok;
    if ((*raiz)->info.salario == elem.salario) {
        NO* aux;
        if ((*raiz)->esq == NULL && (*raiz)->dir == NULL) {
            // Caso 1 - NO sem filhos
            liberarNO(*raiz);
            *raiz = NULL;
        } else if ((*raiz)->esq == NULL) {
            // Caso 2.1 - Possui apenas uma subarvore direita
            aux = *raiz;
            *raiz = (*raiz)->dir;
            liberarNO(aux);
        } else if ((*raiz)->dir == NULL) {
            // Caso 2.2 - Possui apenas uma subarvore esquerda
            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
            // Estrategia 3.1:
            NO* Filho = (*raiz)->esq;
```

```
293             while (Filho->dir != NULL)  // Localiza o MAIOR valor da subarvore esquerda
294                 Filho = Filho->dir;
295             (*raiz)->info = Filho->info;
296             Filho->info = elem;
297             return removeRec(&(*raiz)->esq, elem);
298         }
299         return 1;
300     } else if (elem.salario < (*raiz)->info.salario) {
301         ok = removeRec(&(*raiz)->esq, elem);
302         if (ok) {
303             switch ((*raiz)->fb) {
304                 case +1:
305                 case 0:
306                     // Acertando alturas e Fatores de Balanceamento dos NOs afetados
307                     (*raiz)->alt = -1;
308                     (*raiz)->alt = altura(*raiz);
309                     (*raiz)->fb = FB(*raiz);
310                     break;
311                 case -1:
312                     avl_AuxFD(raiz);
313                     break;
314             }
315         }
316     } else {
317         ok = removeRec(&(*raiz)->dir, elem);
318         if (ok) {
319             switch ((*raiz)->fb) {
320                 case -1:
321                 case 0:
322                     // Acertando alturas e Fatores de Balanceamento dos NOs afetados
323                     (*raiz)->alt = -1;
324                     (*raiz)->alt = altura(*raiz);
325                     (*raiz)->fb = FB(*raiz);
326                     break;
327                 case +1:
328                     avl_AuxFE(raiz);
329                     break;
330             }
331         }
332     }
333     return ok;
334 }
335
336 int removeElem(AVL* raiz, Funcionario elem) {
337     if (pesquisa(raiz, elem) == 0) {
338         printf("Elemento inexistente!\n");
339         return 0;
340     }
341     return removeRec(raiz, elem);
342 }
343
344 void em_ordem(NO* raiz, int nivel) {
345     if (raiz != NULL) {
346         em_ordem(raiz->esq, nivel + 1);
347         // printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
348         printf("[%s, %d, %d, %d] ", raiz->info.nome, raiz->fb, nivel, raiz->alt);
```

```
349         em_ordem(raiz->dir, nivel + 1);
350     }
351 }
352 
353 void imprime(AVL* raiz) {
354     if (raiz == NULL) return;
355     if (estaVazia(raiz)) {
356         printf("Arvore Vazia!\n");
357         return;
358     }
359     // printf("\nEm Ordem: [INFO, FB, NIVEL]\n");
360     printf("\nEm Ordem: [INFO, FB, NIVEL, altura]\n");
361     em_ordem(*raiz, 0);
362     // printf("\nPre Ordem: "); pre_ordem(*raiz, 0);
363     // printf("\nPos Ordem: "); pos_ordem(*raiz, 0);
364     printf("\n");
365 }
366 
367 Funcionario* melhor(NO* raiz) {
368     if (raiz != NULL) {
369         Funcionario *dir = melhor(raiz->dir), *esq = melhor(raiz->esq), *maior;
370         if (dir == NULL || (esq != NULL && esq->salario > dir->salario)) {
371             maior = esq;
372         } else {
373             maior = dir;
374         }
375         if (maior == NULL || raiz->info.salario >= maior->salario) {
376             return &(raiz->info);
377         } else {
378             return maior;
379         }
380     }
381     return NULL;
382 }
383 
384 Funcionario* pior(NO* raiz) {
385     if (raiz != NULL) {
386         Funcionario *dir = melhor(raiz->dir), *esq = melhor(raiz->esq), *maior;
387         if (dir == NULL || (esq != NULL && esq->salario < dir->salario)) {
388             maior = esq;
389         } else {
390             maior = dir;
391         }
392         if (maior == NULL || raiz->info.salario <= maior->salario) {
393             return &(raiz->info);
394         } else {
395             return maior;
396         }
397     }
398     return NULL;
399 }
400 
401 void fixString(char str[50]) {
402     int length = strlen(str);
403     if (str[length - 1] == '\n') str[length - 1] = '\0';
404 }
405 
406 #endif
```

roteiro-09/ex01-02.c

```c
#include "ex01-02.h"

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

enum {
    EXIT = 0,
    CREATE,
    INSERT,
    SEARCH,
    REMOVE,
    PRINT,
    PRINT_BEST,
    PRINT_WORST,
    DESTROY
} Options;

int getOption() {
    int option;

    printf("\n====================\n");
    printf("(%d) Criar\n", CREATE);
    printf("(%d) Inserir elemento\n", INSERT);
    printf("(%d) Buscar elemento\n", SEARCH);
    printf("(%d) Remover elemento\n", REMOVE);
    printf("(%d) Imprimir em ordem\n", PRINT);
    printf("(%d) Imprimir funcionario com maior salario\n", PRINT_BEST);
    printf("(%d) Imprimir funcionario com menor salario\n", PRINT_WORST);
    printf("(%d) Destruir\n", DESTROY);
    printf("(%d) Sair\n", EXIT);
    printf("====================\n");
    printf("Operacao: ");

    scanf("%d", &option);
    printf("\n");

    return option;
}

int runMenu() {
    AVL* avl = NULL;
    int exit = 0;

    Funcionario item, *itemp;
    char nome[50];
    int contratacao;
    double salario;

    do {
        switch (getOption()) {
            case CREATE:
                if (avl != NULL) {
                    destroiAVL(&avl);
                }
                avl = criaAVL();
                break;
```

```
57
58              case INSERT:
59                  printf("Funcionario a ser inserido\n");
60                  printf("Nome (Max 50 caracteres): ");
61                  setbuf(stdin, NULL);
62                  fgets(item.nome, 50, stdin);
63                  fixString(item.nome);
64                  setbuf(stdin, NULL);
65                  printf("Ano de contratacao: ");
66                  scanf("%d", &item.contratacao);
67                  printf("Salario: ");
68                  scanf("%lf", &item.salario);
69
70                  if (insereElem(avl, item)) {
71                      printf("Funcionario inserido (%s)", item.nome);
72                  } else {
73                      printf("Falha ao inserir (%s)", item.nome);
74                  }
75                  break;
76
77              case SEARCH:
78                  printf("Salario para buscar: ");
79                  scanf("%lf", &item.salario);
80
81                  if (pesquisa(avl, item)) {
82                      printf("Funcionario presente (%.2lf)", item.salario);
83                  } else {
84                      printf("Funcionario nao encontrado (%.2lf)", item.salario);
85                  }
86                  break;
87
88              case REMOVE:
89                  printf("Nome para remover (Max 50 caracteres): ");
90                  setbuf(stdin, NULL);
91                  fgets(item.nome, 50, stdin);
92                  fixString(item.nome);
93                  setbuf(stdin, NULL);
94
95                  if (removeElem(avl, item)) {
96                      printf("Funcionario removido (%s)", item.nome);
97                  } else {
98                      printf("Falha ao remover (%s)", item.nome);
99                  }
100                 break;
101
102             case PRINT:
103                 imprime(avl);
104                 break;
105
106             case PRINT_BEST:
107                 itemp = melhor(*avl);
108                 printf("Funcionario com o maior salario\n");
109                 printf("Nome = %s\nContratacao = %d\nSalario = %.2lf", itemp->
    nome, itemp->contratacao, itemp->salario);
110                 break;
111
112             case PRINT_WORST:
113                 itemp = pior(*avl);
114                 printf("Funcionario com o menor salario\n");
```

```c
                printf("Nome = %s\nContratacao = %d\nSalario = %.2lf", itemp->nome, itemp->contratacao, itemp->salario);
                break;

            case DESTROY:
                destroiAVL(&avl);
                break;

            case EXIT:
                if (avl != NULL) {
                    destroiAVL(&avl);
                }
                printf("Programa encerrado");
                exit = 1;
                break;

            default:
                printf("Opcao desconhecida, tente novamente");
        }
        printf("\n");
    } while (!exit);
}

int main() {
    runMenu();
    return 0;
}
```

```
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-09$ ./ex01-02

====================
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir funcionario com maior salario
(7) Imprimir funcionario com menor salario
(8) Destruir
(0) Sair
====================
Operacao: 1



====================
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir funcionario com maior salario
(7) Imprimir funcionario com menor salario
(8) Destruir
(0) Sair
====================
Operacao: 2

Funcionario a ser inserido
Nome (Max 50 caracteres): Gabriel
Ano de contratacao: 2022
Salario: 320
Funcionario inserido (Gabriel)

====================
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir funcionario com maior salario
(7) Imprimir funcionario com menor salario
(8) Destruir
(0) Sair
====================
Operacao: 2

Funcionario a ser inserido
Nome (Max 50 caracteres): Prenassi
Ano de contratacao: 2023
Salario: 700
Funcionario inserido (Prenassi)

====================
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir funcionario com maior salario
(7) Imprimir funcionario com menor salario
(8) Destruir
(0) Sair
====================
Operacao: 2

Funcionario a ser inserido
Nome (Max 50 caracteres): Gabriel de Paula
Ano de contratacao: 2023
Salario: 800
Rotacao Simples a ESQUERDA!
Falha ao inserir (Gabriel de Paula)

====================
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir funcionario com maior salario
(7) Imprimir funcionario com menor salario
(8) Destruir
(0) Sair
====================
Operacao: 3

Salario para buscar: 800
Funcionario presente (800.00)

====================
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir funcionario com maior salario
(7) Imprimir funcionario com menor salario
(8) Destruir
(0) Sair
====================
Operacao: 5


Em Ordem: [INFO, FB, NIVEL, altura]
[Gabriel, 0, 1, 1] [Prenassi, 0, 0, 2] [Gabriel de Paula, 0, 1, 1]


====================
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir funcionario com maior salario
(7) Imprimir funcionario com menor salario
(8) Destruir
(0) Sair
====================
Operacao: 6

Funcionario com o maior salario
Nome = Gabriel de Paula
Contratacao = 2023
Salario = 800.00

====================
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir funcionario com maior salario
(7) Imprimir funcionario com menor salario
(8) Destruir
(0) Sair
====================
Operacao: 7

Funcionario com o menor salario
Nome = Gabriel
Contratacao = 2022
Salario = 320.00

====================
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir funcionario com maior salario
(7) Imprimir funcionario com menor salario
(8) Destruir
(0) Sair
====================
Operacao: 0

Programa encerrado
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-09$
```

roteiro-09/ex01-02.txt

A complexidade da operação de encontrar os funcionários com maior
e menor salário é O(n), tendo em vista que as AVLs possuem a inserção
mais custosa para que a pesquisa seja mais eficiente do que em árvores
binárias comuns.