UNIVERSIDADE FEDERAL DO CEARÁ - CAMPUS CRATEÚS

CURSO: CIÊNCIA DA COMPUTAÇÃO

DISCIPLINA: ESTRUTURA DE DADOS (2022.2)

PROF. ARNALDO BARRETO VILA NOVA

EQUIPE: FRANCISCO LUCAS FARIAS DA SILVA

LUCAS EDUARDO MOTA

MARIA CLARA PEREIRA DE SOUSA

ÁRVORE GENEALÓGICA

Crateús – CE, 2022

Sumário

Descrição do TAD

Estruturamos nossa Árvore Genealógica com um TAD, ou seja, nosso código foi dividido da seguinte forma:

tad.h

```h
#ifndef TAD_H_INCLUDED
#define TAD_H_INCLUDED

typedef struct arvore_genealogica {
    char nome[50];
    char genero; //F para feminino e H para masculino
    struct arvore_genealogica *pai, *mae;
}Arvg;

Arvg* adicionar(Arvg* pessoa, char* nome, char genero);
void listarTudo(Arvg* pessoa);
int altura(Arvg* pessoa);
int contaPessoas(Arvg* pessoa);
void apagarArvore(Arvg* pessoa);
Arvg* buscarPessoa(Arvg* pessoa, char* nome, char genero);
void listarAncestrais(Arvg* pessoa, char* nome, char genero);
Arvg* removerPessoa(Arvg* pessoa, char* nome, char genero);


#endif // TAD_H_INCLUDED
```

tad.c

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

Arvg* adicionar(Arvg* pessoa, char* nome, char genero){
    if(pessoa == NULL){
        Arvg* novo = (Arvg*) malloc(sizeof(Arvg));
        strcpy(novo->nome,nome);
        novo->genero = genero;
        novo->mae = NULL;
        novo->pai = NULL;
        return novo;
    }
    else{
        if(genero == 'F' || genero == 'f'){
            pessoa->mae = adicionar(pessoa->mae, nome, 'f'); // esquerda
```

```
        }
        else if(genero == 'M' || genero == 'm'){
            pessoa->pai = adicionar(pessoa->pai, nome, 'm'); // direita
        }
        return pessoa;
    }
}
void listarTudo(Arvg* pessoa){
    if(pessoa){
        printf("%s ", pessoa->nome);
        listarTudo(pessoa->mae);
        listarTudo(pessoa->pai);
    }
}

int altura(Arvg* pessoa){
    if(pessoa == NULL){
        return -1;
    }
    else{
        int esquerda = altura(pessoa->mae);
        int direita = altura(pessoa->pai);
        if(esquerda > direita){
            return esquerda+1;
        }
        else{
            return direita+1;
        }
    }
}

int contaPessoas(Arvg* pessoa){
    if(pessoa == NULL){
        return 0;
    }
    else{
        return 1 + contaPessoas(pessoa->mae) + contaPessoas(pessoa->pai);
    }
}

void apagarArvore(Arvg* pessoa)
{
    if (pessoa!=NULL)
    {
        apagarArvore(pessoa->mae);
        apagarArvore(pessoa->pai);
        pessoa->nome[0] = '\0';
        pessoa->genero = '\0';
        pessoa->mae = NULL;
```

```
        pessoa->pai = NULL;
        free(pessoa);
    }
}

Arvg* buscarPessoa(Arvg* pessoa, char* nome, char genero){
    while(pessoa){
        if(strcasecmp(pessoa->nome,nome) != 0 && genero == 'F'){
            pessoa = pessoa->mae;
        }
        else if(strcasecmp(pessoa->nome,nome) != 0 && genero == 'M'){
            pessoa = pessoa->pai;
        }
        else{
            return pessoa;
        }
    }
    return NULL;
}

void listarAncestrais(Arvg* pessoa, char* nome, char genero){
    Arvg* busca = buscarPessoa(pessoa,nome,genero);

    if(busca){
        listarTudo(busca);
    }
    else{
        printf("Pessoa fora da árvore genealógica");
    }
}

Arvg* removerPessoa(Arvg* pessoa, char* nome, char genero){
    if(pessoa == NULL){
        printf("Pessoa não encontrada");
        return NULL;
    }
    else{
        if(strcasecmp(pessoa->nome,nome) == 0){
                //Remove uma pessoa sem pai e mãe
            if(pessoa->mae == NULL && pessoa->pai == NULL){
                free(pessoa);
                return NULL;
            }
            else{
                //Remove uma pessoa que possui somente 1 pai ou mãe
                if(pessoa->mae == NULL || pessoa->pai == NULL){
                    Arvg* aux;
                    if(pessoa->mae != NULL){
                        aux = pessoa->mae;
```

```
                    }
                    else{
                        aux = pessoa->pai;
                    }
                    free(pessoa);
                    return aux;
                }
                else{
                    printf("Remover o indivíduo principal da árvore
genealógica desconfigura a lógica pai e mãe...");
                    return pessoa;
                }
            }
        }
        else{
            if(genero == 'F'){
                pessoa->mae = removerPessoa(pessoa->mae, nome, 'F');
            }
            else{
                pessoa->pai = removerPessoa(pessoa->pai, nome, 'M');
            }
            return pessoa;
        }
    }
}
```

E a nossa classe main para testar a Árvore:

main.c

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

int main()
{
    /* É criado uma árvore genealógica que parte de um indivíduo,
    e são listados sua mãe, vó, bisavó, tataravó e etc.
    E seu pai, vô, bisavô, tataravô e etc.*/

    Arvg *pessoa = NULL;

    pessoa = adicionar(pessoa, "Lucas", 'M');
    pessoa = adicionar(pessoa, "Eduardo", 'M');
    pessoa = adicionar(pessoa, "Maria", 'F');
```

```
    pessoa = adicionar(pessoa, "Greta", 'F');
    pessoa = adicionar(pessoa, "Antonio ", 'M');
    pessoa = adicionar(pessoa, "Raimundo", 'M');
    pessoa = adicionar(pessoa, "Nilta", 'F');
    pessoa = adicionar(pessoa, "Josef", 'M');

    listarTudo(pessoa);

    printf("%d ",altura(pessoa));

    printf("%d ",contaPessoas(pessoa));

    buscarPessoa(pessoa,"Greta",'F');

    listarAncestrais(pessoa, "Maria", 'F');

    removerPessoa(pessoa, "Greta", 'F');


    return 0;
}
```

Explicação das Funções

Função adicionar

Insere uma pessoa na árvore. Primeiro é checado se o nó em questão é nulo, se sim, a pessoa é inserida naquela posição, se a posição já estiver ocupada serão feitas chamadas recursivas para encontrar uma posição que não esteja ocupada, se o gênero da pessoa a ser adicionada for feminino ("F ou f") a mesma será posicionada a esquerda da pessoa principal, se for masculina ("M ou m"), o mesmo será posicionado a direita da pessoa principal. Por fim é retornado o endereço de memória da posição ocupada.

```
Arvg* adicionar(Arvg* pessoa, char* nome, char genero){
    if(pessoa == NULL){
        Arvg* novo = (Arvg*) malloc(sizeof(Arvg));
        strcpy(novo->nome,nome);
        novo->genero = genero;
        novo->mae = NULL;
        novo->pai = NULL;
        return novo;
    }
```

```
    else{
        if(genero == 'F' || genero == 'f'){
            pessoa->mae = adicionar(pessoa->mae, nome, 'f'); // esquerda
        }
        else if(genero == 'M' || genero == 'm'){
            pessoa->pai = adicionar(pessoa->pai, nome, 'm'); // direita
        }
        return pessoa;
    }
}
```

Função listarTudo

Imprime no console as pessoas da árvore. Enquanto pessoa não for NULL, serão feitas chamadas recursivas, primeiro imprimindo a pessoa principal e suas matriarcas (esquerda da árvore) e depois os patriarcas da pessoa principal (direita da árvore).

```
void listarTudo(Arvg* pessoa){
    if(pessoa){
        printf("%s ", pessoa->nome);
        listarTudo(pessoa->mae);
        listarTudo(pessoa->pai);
    }
}
```

Função altura

Calcula a altura da árvore. Primeiramente, a altura de uma árvore, é a maior distância que existe entre o nó raiz e o nó folha. Assim, serão feitas chamadas recursivas para a esquerda e para a direita, após cada chamada, as variáveis esquerda e direita serão incrementadas (ou não) e no fim das chamadas recursivas, será checado qual a variável de maior valor e por fim será retornado a mesma mais 1.

```
int altura(Arvg* pessoa){
    if(pessoa == NULL){
        return -1;
    }
    else{
        int esquerda = altura(pessoa->mae);
        int direita = altura(pessoa->pai);
        if(esquerda > direita){
```

```
            return esquerda+1;
        }
        else{
            return direita+1;
        }
    }
}
```

Função contaPessoas

Conta a quantidade de pessoas na árvore. Primeiro é checado se a árvore é vazia, se sim, é retornado o valor 0, que indica que não há ninguém na árvore, se não, serão feitas chamadas recursivas para a esquerda (lado das mães) e para a direita (lados dos pais), no fim, será retornado a soma da pessoa principal mais as mães, mais os pais.

```
int contaPessoas(Arvg* pessoa){
    if(pessoa == NULL){
        return 0;
    }
    else{
        return 1 + contaPessoas(pessoa->mae) + contaPessoas(pessoa->pai);
    }
}
```

Função apagarArvore

Deleta todas as pessoas da árvore. Recursivamente, inicializa cada atributo de uma pessoa para um valor nulo e libera com a função free o endereço de memória que foi usado para alocá-la.

```
void apagarArvore(Arvg* pessoa)
{
    if (pessoa!=NULL)
    {
        apagarArvore(pessoa->mae);
        apagarArvore(pessoa->pai);
        pessoa->nome[0] = '\0';
        pessoa->genero = '\0';
        pessoa->mae = NULL;
        pessoa->pai = NULL;
        free(pessoa);
```

```
    }
}
```

Função buscarPessoa

Busca uma pessoa a partir de seu nome e gênero. Em um loop, enquanto pessoa não for nulo, é percorrida a árvore, buscando pelo nome da pessoa, se for uma pessoa do gênero feminino, a mesma será procurada a esquerda da árvore, assim como se for masculino será procurado a direita. Toda vez que o nome procurado não for igual ao nome passado, é feita a alteração de pessoa, e quando de fato o nome procurado condiz com o nome passado na função, é retornado pessoa para quem chamou. Se por fim o nome não for achado, chega ao fim o loop e é retornado nulo, significando que a pessoa não foi encontrada.

```
Arvg* buscarPessoa(Arvg* pessoa, char* nome, char genero){
    while(pessoa){
        if(strcasecmp(pessoa->nome,nome) != 0 && genero == 'F'){
            pessoa = pessoa->mae;
        }
        else if(strcasecmp(pessoa->nome,nome) != 0 && genero == 'M'){
            pessoa = pessoa->pai;
        }
        else{
            return pessoa;
        }
    }
    return NULL;
}
```

Função listarAncestrais

Busca uma pessoa e imprime dela em diante. É criada uma variável busca para armazenar a pessoa buscada, em seguida, é verificado se a pessoa buscada existe na árvore, se sim é chamada a função listarTudo passando como parâmetro a variável busca, para listar todas a pessoas a partir dela. Se a pessoa buscada não existir, é exibida a mensagem "Pessoa fora da árvore genealógica".

```
void listarAncestrais(Arvg* pessoa, char* nome, char genero){
    Arvg* busca = buscarPessoa(pessoa,nome,genero);

    if(busca){
        listarTudo(busca);
```

```
    }
    else{
        printf("Pessoa fora da árvore genealógica");
    }
}
```

Função removerPessoa

Remove uma pessoa da árvore. Primeiramente, se pessoa for nulo, significa que a mesma não foi encontrada na árvore, então, não há o que remover. Se não, é feita a verificação a fim de saber se o nome passado é igual ao nome da pessoa na árvore, se não for, serão feitas chamadas recursivas a fim de continuar procurando a pessoa a ser removida. Quando a pessoa é encontrada na árvore, será analisado se essa pessoa terá pai e mãe, se sim, a mensagem "Remover o indivíduo principal da árvore genealógica desconfigura a lógica pai e mãe...", pois remover a pessoa principal prejudica a organização lógica da árvore, caso a pessoa não tenha nem pai nem mãe, simplesmente é liberada com a função free e retornado nulo, caso tenha um pai ou uma mãe somente, é criado uma variável aux para segurar o conteúdo do pai ou mãe de quem vai ser removido, então, a pessoa é removida com a função free, e é retornado o aux.

```
Arvg* removerPessoa(Arvg* pessoa, char* nome, char genero){
    if(pessoa == NULL){
        printf("Pessoa não encontrada");
        return NULL;
    }
    else{
        if(strcasecmp(pessoa->nome,nome) == 0){
                //Remove uma pessoa sem pai e mãe
            if(pessoa->mae == NULL && pessoa->pai == NULL){
                free(pessoa);
                return NULL;
            }
            else{
                //Remove uma pessoa que possui somente 1 pai ou mãe
                if(pessoa->mae == NULL || pessoa->pai == NULL){
                    Arvg* aux;
                    if(pessoa->mae != NULL){
                        aux = pessoa->mae;
                    }
                    else{
                        aux = pessoa->pai;
                    }
                    free(pessoa);
                    return aux;
                }
                else{
```

```
                    printf("Remover o indivíduo principal da árvore
genealógica desconfigura a lógica pai e mãe...");
                    return pessoa;
                }
            }
        }
        else{
            if(genero == 'F'){
                pessoa->mae = removerPessoa(pessoa->mae, nome, 'F');
            }
            else{
                pessoa->pai = removerPessoa(pessoa->pai, nome, 'M');
            }
            return pessoa;
        }
    }
}
```

Divisão do Trabalho

Cada integrante da equipe pesquisou sobre, e depois discutimos como que iriamos implementar a Árvore Genealógica. Depois de decidir, cada um ficou responsável para implementar algumas das funções.

Francisco Lucas

Ficou responsável por implementar a função **listarTudo**, **buscarPessoa** e **listarAncestrais.**

Lucas Eduardo

Ficou responsável por implementar duas funções extras, e fez a função **altura** e **contaPessoas.**

Maria Clara

Ficou responsável por implementar a função r**emoverPessoa**, **removerArvore** e **adicionar**.

O restante foi desenvolvido em conjunto.

Dificuldades Encontradas

1. Encontramos dificuldades principalmente por acharmos a implementação de uma Árvore Genealógica difícil.
2. A criação de algumas funções, principalmente na de listar descendentes.