UNIVERSIDADE FEDERAL DO CEARÁ
CURSO: CIÊNCIA DA COMPUTAÇÃO
DISCIPLINA: ESTRUTURA DE DADOS (2023.2)
PROF. ARNALDO BARRETO VILA NOVA

KAUAN COELHO VASCONCELOS

ÁRVORE GENEALÓGICA

Crateús-CE, 2023

Sumário

1 - Descrição do TAD 3
tad.h....3
tad.c.... 4
main.c.... 9
2 - Explicação das Funções.... 12
Função inicializarFila....12
Função enfileirarIrmao....12
Função adicionarIrmao....13
Função imprimirIrmaos....13
Função calcularTamanhoArvore....14
Função removerPessoa....14
Função criarPessoa....15
Função adicionarPessoa....16
Função exibirArvoreGenealogica....16
Função buscarPessoa....17
Função liberarMemoria....17
Função imprimirIrmaosEFamilia.... 18
Função calcularAlturaArvore....19
3 - Dificuldades Encontradas.... 20

1 - Descrição do TAD

A Árvore Genealógica foi estruturada a partir de um tipo abstrato de dados, o uso de um TAD nesse contexto ajuda a encapsular as operações relacionadas à árvore genealógica, facilitando a organização e manutenção do código.

tad.h

```c
// arvoreGenealogica.h
#ifndef GENEALOGIA_H
#define GENEALOGIA_H

typedef struct Node {
   struct Pessoa *pessoa;
   struct Node *prox;
} Node;

typedef struct Fila {
   Node *frente;
   Node *tras;
} Fila;

typedef struct Pessoa {
   char nome[50];
   struct Pessoa *pai;
   struct Pessoa *mae;
   struct Fila *irmaos;
} Pessoa;

typedef struct ListaPessoas {
   Pessoa *pessoa;
   struct ListaPessoas *proximo;
} ListaPessoas;

void inicializarFila(Fila *fila);
void enfileirarIrmao(Fila *fila, Pessoa *irmao);
void adicionarIrmao(Fila *irmaos, Pessoa *irmao);
void imprimirIrmaos(Fila *irmaos, const char *nomePessoa);
int calcularTamanhoArvore(ListaPessoas *lista);
void removerPessoa(ListaPessoas **lista, const char *nome);
Pessoa *criarPessoa(const char *nome, Pessoa *pai, Pessoa *mae);
void adicionarPessoa(ListaPessoas **lista, Pessoa *pessoa);
void exibirArvoreGenealogica(Pessoa *pessoa, int nivel);
Pessoa *buscarPessoa(Pessoa *raiz, const char *nome);
void liberarMemoria(ListaPessoas *lista);
void imprimirIrmaosEFamilia(Pessoa *pessoa);
int calcularAlturaArvore(Pessoa *raiz);
```

```
#endif // GENEALOGIA_H
```

tad.c

```
// arvoreGenealogica.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "arvore.h"

void inicializarFila(Fila *fila) {
   fila->frente = NULL;
   fila->tras = NULL;
}

void enfileirarIrmao(Fila *fila, Pessoa *irmao) {
   if (fila->tras == NULL) {
       inicializarFila(fila);
   }

   Node *novoNode = (Node *)malloc(sizeof(Node));
   if (novoNode == NULL) {
       printf("Erro ao alocar memória para o nó da fila.\n");
       exit(EXIT_FAILURE);
   }

   novoNode->pessoa = irmao;
   novoNode->prox = NULL;

   if (fila->tras == NULL) {
       fila->frente = novoNode;
       fila->tras = novoNode;
   } else {
       fila->tras->prox = novoNode;
       fila->tras = novoNode;
   }
}

void adicionarIrmao(Fila *irmaos, Pessoa *irmao) {
   enfileirarIrmao(irmaos, irmao);
}

void imprimirIrmaos(Fila *irmaos, const char *nomePessoa) {
   if (irmaos != NULL && irmaos->frente != NULL) {
```

```
        Node *atual = irmaos->frente;
        while (atual != NULL) {
            printf("Irmão de %s: %s\n", nomePessoa, atual->pessoa->nome);
            atual = atual->prox;
        }
    } else {
        printf("%s não tem irmãos.\n", nomePessoa);
    }
}

int calcularTamanhoArvore(ListaPessoas *lista) {
    int tamanho = 0;
    ListaPessoas *temp = lista;
    while (temp != NULL) {
        tamanho++;
        temp = temp->proximo;
    }
    return tamanho;
}

void removerPessoa(ListaPessoas **lista, const char *nome) {
    ListaPessoas *atual = *lista;
    ListaPessoas *anterior = NULL;

    while (atual != NULL && strcmp(atual->pessoa->nome, nome) != 0) {
        anterior = atual;
        atual = atual->proximo;
    }

    if (atual == NULL) {
        printf("Pessoa com o nome %s não encontrada na lista.\n", nome);
        return;
    }

    Node *nodeAtual = atual->pessoa->irmaos->frente;
    while (nodeAtual != NULL) {
        Node *temp = nodeAtual;
        nodeAtual = nodeAtual->prox;
        free(temp);
    }

    if (anterior == NULL) {
        *lista = atual->proximo;
    } else {
        anterior->proximo = atual->proximo;
    }
```

```
    free(atual->pessoa->irmaos);
    free(atual->pessoa);
    free(atual);

    printf("Pessoa removida com sucesso!\n");
}

Pessoa *criarPessoa(const char *nome, Pessoa *pai, Pessoa *mae) {
    Pessoa *novaPessoa = (Pessoa *)malloc(sizeof(Pessoa));
    strcpy(novaPessoa->nome, nome);
    novaPessoa->pai = pai;
    novaPessoa->mae = mae;
    novaPessoa->irmaos = (Fila *)malloc(sizeof(Fila));
    inicializarFila(novaPessoa->irmaos);
    return novaPessoa;
}

void adicionarPessoa(ListaPessoas **lista, Pessoa *pessoa) {
    ListaPessoas *novoNode = (ListaPessoas *)malloc(sizeof(ListaPessoas));
    novoNode->pessoa = pessoa;
    novoNode->proximo = *lista;
    *lista = novoNode;
}

void exibirArvoreGenealogica(Pessoa *pessoa, int nivel) {
    if (pessoa != NULL) {
        for (int i = 0; i < nivel; i++) {
            printf("  ");
        }
        printf("%s", pessoa->nome);

        if (pessoa->irmaos != NULL && pessoa->irmaos->frente != NULL) {
            Node *atual = pessoa->irmaos->frente;
            printf(" - Irmãos - ");
            while (atual != NULL) {
                printf("%s ", atual->pessoa->nome);
                atual = atual->prox;
            }
        }

        printf("\n");

        exibirArvoreGenealogica(pessoa->pai, nivel + 1);
        exibirArvoreGenealogica(pessoa->mae, nivel + 1);
    }
```

```
}

Pessoa *buscarPessoa(Pessoa *raiz, const char *nome) {
   if (raiz == NULL) {
       return NULL;
   }

   if (strcmp(raiz->nome, nome) == 0) {
       return raiz;
   }

   Pessoa *pessoaEncontrada = buscarPessoa(raiz->pai, nome);
   if (pessoaEncontrada == NULL) {
       pessoaEncontrada = buscarPessoa(raiz->mae, nome);
   }

   return pessoaEncontrada;
}

void liberarMemoria(ListaPessoas *lista) {
   while (lista != NULL) {
       ListaPessoas *temp = lista;
       lista = lista->proximo;
       free(temp->pessoa->irmaos);
       free(temp->pessoa);
       free(temp);
   }
}

void imprimirIrmaosEFamilia(Pessoa *pessoa) {
   if (pessoa != NULL) {
       printf("Família de %s:\n", pessoa->nome);
       printf("  Pai: %s\n", (pessoa->pai != NULL) ? pessoa->pai->nome :
"Desconhecido");
       printf("  Mãe: %s\n", (pessoa->mae != NULL) ? pessoa->mae->nome :
"Desconhecido");
       printf("  Irmãos: ");

       if (pessoa->irmaos != NULL && pessoa->irmaos->frente != NULL) {
           Node *atual = pessoa->irmaos->frente;
           while (atual != NULL) {
               printf("%s ", atual->pessoa->nome);
               atual = atual->prox;
           }
       } else {
           printf("Nenhum irmão");
```

```
        }

        printf("\n");
    } else {
        printf("Pessoa não encontrada.\n");
    }
}

int calcularAlturaArvore(Pessoa *raiz) {
    if (raiz == NULL) {
        return -1;
    }

    int alturaPai = calcularAlturaArvore(raiz->pai);
    int alturaMae = calcularAlturaArvore(raiz->mae);

    return 1 + ((alturaPai > alturaMae) ? alturaPai : alturaMae);
}
```

main.c

```
// main.c
#include <stdio.h>
#include <string.h>
#include "arvore.h"

int main() {
    ListaPessoas *lista = NULL;

    int opcao;
    do {
        printf("\nMenu:\n");
        printf("1. Criar Pessoa\n");
        printf("2. Exibir Árvore Genealógica\n");
        printf("3. Adicionar Irmão\n");
        printf("4. Imprimir Irmãos\n");
        printf("5. Buscar Pessoa\n");
        printf("6. Remover Pessoa\n");
        printf("7. Calcular Tamanho da Árvore\n");
        printf("8. Calcular Altura da Árvore\n");
        printf("0. Sair\n");
        printf("Escolha uma opção: ");
        scanf("%d", &opcao);

        switch (opcao) {
            case 1: {
```

```
            char nome[50];
            printf("Digite o nome da pessoa: ");
            scanf("%s", nome);

            printf("Pessoas disponíveis:\n");
            ListaPessoas *temp = lista;
            while (temp != NULL) {
                printf("%s\n", temp->pessoa->nome);
                temp = temp->proximo;
            }

            char nomePai[50], nomeMae[50];
            printf("Digite o nome do pai: ");
            scanf("%s", nomePai);
            printf("Digite o nome da mãe: ");
            scanf("%s", nomeMae);

            Pessoa *pai = NULL, *mae = NULL;
            temp = lista;
            while (temp != NULL) {
                if (strcmp(temp->pessoa->nome, nomePai) == 0) {
                    pai = temp->pessoa;
                }
                if (strcmp(temp->pessoa->nome, nomeMae) == 0) {
                    mae = temp->pessoa;
                }
                temp = temp->proximo;
            }

            Pessoa *novaPessoa = criarPessoa(nome, pai, mae);
            adicionarPessoa(&lista, novaPessoa);
            printf("Pessoa criada com sucesso!\n");
            break;
        }
        case 2:
            printf("Árvore Genealógica:\n");
            if (lista != NULL) {
                exibirArvoreGenealogica(lista->pessoa, 0);
            } else {
                printf("Árvore genealógica vazia.\n");
            }
            break;
        case 3: {
            char nome[50];
            printf("Digite o nome da pessoa para adicionar como irmão:
");
```

```
                scanf("%s", nome);

                ListaPessoas *pessoa = lista;
                while (pessoa != NULL) {
                    if (strcmp(pessoa->pessoa->nome, nome) == 0) {
                        break;
                    }
                    pessoa = pessoa->proximo;
                }

                if (pessoa != NULL) {
                    char nomeIrmao[50];
                    printf("Digite o nome do irmão: ");
                    scanf("%s", nomeIrmao);

                    Pessoa *novoIrmao = criarPessoa(nomeIrmao,
pessoa->pessoa->pai, pessoa->pessoa->mae);
                    adicionarIrmao(pessoa->pessoa->irmaos, novoIrmao);
                    printf("Irmão adicionado com sucesso!\n");
                } else {
                    printf("Pessoa com o nome %s não encontrada na
lista.\n", nome);
                }
                break;
            }
            case 4: {
                char nomeIrmaos[50];
                printf("Digite o nome da pessoa para imprimir os irmãos: ");
                scanf("%s", nomeIrmaos);

                ListaPessoas *pessoa = lista;
                while (pessoa != NULL) {
                    if (strcmp(pessoa->pessoa->nome, nomeIrmaos) == 0) {
                        break;
                    }
                    pessoa = pessoa->proximo;
                }

                if (pessoa != NULL) {
                    imprimirIrmaos(pessoa->pessoa->irmaos,
pessoa->pessoa->nome);
                } else {
                    printf("Pessoa com o nome %s não encontrada na
lista.\n", nomeIrmaos);
                }
```

```
                break;
            }
            case 5: {
                char nomeBusca[50];
                printf("Digite o nome da pessoa para buscar: ");
                scanf("%s", nomeBusca);

                Pessoa *pessoaEncontrada = buscarPessoa(lista->pessoa,
nomeBusca);
                imprimirIrmaosEFamilia(pessoaEncontrada);

                break;
            }
            case 6: {
                char nomeRemover[50];
                printf("Digite o nome da pessoa para remover: ");
                scanf("%s", nomeRemover);

                removerPessoa(&lista, nomeRemover);
                break;
            }
            case 7:
                printf("Tamanho da Árvore: %d pessoa(s)\n",
calcularTamanhoArvore(lista));
                break;
            case 8: {
                if (lista != NULL) {
                    int altura = calcularAlturaArvore(lista->pessoa);
                    printf("Altura da Árvore: %d\n", altura);
                } else {
                    printf("Árvore genealógica vazia.\n");
                }
                break;
            }
        }
    } while (opcao != 0);

    return 0;      }
```

2 - Explicação das Funções

Função inicializarFila

Tem o propósito de preparar a fila para armazenar informações sobre os irmãos de uma pessoa na árvore genealógica. Recebendo um ponteiro para a estrutura Fila como parâmetro, a função atribui valores nulos aos ponteiros inicio e fim, indicando que a fila está vazia no momento da inicialização. Essa abordagem é crucial para garantir que a fila esteja pronta e consistente, proporcionando uma base sólida para futuras operações de adição e remoção de elementos na árvore genealógica

```
void inicializarFila(Fila *fila) {
   fila->frente = NULL;
   fila->tras = NULL;
}
```

Função enfileirarIrmao

É projetada para adicionar um irmão à fila de irmãos de uma pessoa na árvore genealógica. Recebendo como parâmetro um ponteiro para a estrutura "Fila", que representa a fila de irmãos, a função adiciona um novo irmão ao final da fila. Isso é feito por meio de alocação dinâmica de memória, configuração dos ponteiros apropriados e ajustes na estrutura da fila. Essa função é essencial para expandir a representação dos relacionamentos familiares na árvore genealógica, permitindo o registro eficiente dos irmãos de uma pessoa.

```
void enfileirarIrmao(Fila *fila, Pessoa *irmao) {
   if (fila->tras == NULL) {
       inicializarFila(fila);
   }

   Node *novoNode = (Node *)malloc(sizeof(Node));
   if (novoNode == NULL) {
       printf("Erro ao alocar memória para o nó da fila.\n");
       exit(EXIT_FAILURE);
   }

   novoNode->pessoa = irmao;
   novoNode->prox = NULL;

   if (fila->tras == NULL) {
       fila->frente = novoNode;
       fila->tras = novoNode;
   } else {
       fila->tras->prox = novoNode;
       fila->tras = novoNode;
   }
```

```
}
```

Função adicionarIrmao

É responsável por adicionar um novo irmão à árvore genealógica. Ao receber um ponteiro para a estrutura "Pessoa" que representa o indivíduo ao qual o novo irmão será adicionado, a função aloca dinamicamente memória para criar uma nova instância de "Pessoa", atribui os dados do novo irmão e o enfileira na fila de irmãos do indivíduo. Essa função facilita a expansão da árvore genealógica, registrando eficientemente informações sobre os irmãos de uma pessoa.

```
void adicionarIrmao(Fila *irmaos, Pessoa *irmao) {
   enfileirarIrmao(irmaos, irmao);}
```

Função imprimirIrmaos

Tem como objetivo exibir na tela os irmãos de uma pessoa na árvore genealógica. Ao receber um ponteiro para a estrutura "Pessoa" representando um indivíduo, a função percorre a fila de irmãos associada a essa pessoa e imprime os nomes dos irmãos. Essa função contribui para visualizar e compreender as relações familiares na árvore genealógica, apresentando de forma clara os irmãos de um determinado indivíduo.

```
void imprimirIrmaos(Fila *irmaos, const char *nomePessoa) {
   if (irmaos != NULL && irmaos->frente != NULL) {
       Node *atual = irmaos->frente;
       while (atual != NULL) {
           printf("Irmão de %s: %s\n", nomePessoa, atual->pessoa->nome);
           atual = atual->prox;
       }
   } else {
       printf("%s não tem irmãos.\n", nomePessoa);
   }
}
```

Função calcularTamanhoArvore

Responsável por calcular o tamanho total da árvore genealógica, contando recursivamente o número de indivíduos na árvore. Para cada pessoa, a função considera o próprio nó e chama recursivamente para os pais, acumulando o tamanho das subárvores. O resultado final é a soma do tamanho das subárvores e 1 para contar o nó atual. Essa função proporciona uma maneira eficaz de avaliar a dimensão da árvore genealógica.

```c
int calcularTamanhoArvore(ListaPessoas *lista) {
    int tamanho = 0;
    ListaPessoas *temp = lista;
    while (temp != NULL) {
        tamanho++;
        temp = temp->proximo;
    }
    return tamanho;
}
```

Função removerPessoa

Tem como objetivo retirar uma pessoa da árvore genealógica. Ao receber um ponteiro para o nó raiz e o nome da pessoa a ser removida, a função verifica se a pessoa atual é a desejada. Se encontrada, a lógica de remoção é implementada . A função chama recursivamente para nós pai e mãe, permitindo a busca em toda a árvore.

```c
void removerPessoa(ListaPessoas **lista, const char *nome) {
    ListaPessoas *atual = *lista;
    ListaPessoas *anterior = NULL;

    while (atual != NULL && strcmp(atual->pessoa->nome, nome) != 0) {
        anterior = atual;
        atual = atual->proximo;
    }

    if (atual == NULL) {
        printf("Pessoa com o nome %s não encontrada na lista.\n", nome);
        return;
    }

    Node *nodeAtual = atual->pessoa->irmaos->frente;
    while (nodeAtual != NULL) {
        Node *temp = nodeAtual;
        nodeAtual = nodeAtual->prox;
        free(temp);
    }

    if (anterior == NULL) {
        *lista = atual->proximo;
    } else {
        anterior->proximo = atual->proximo;
    }
```

```
    free(atual->pessoa->irmaos);
    free(atual->pessoa);
    free(atual);

    printf("Pessoa removida com sucesso!\n");
}
```

Função criarPessoa

A função criar Pessoa desempenha um papel fundamental na construção da árvore genealógica. Ela utiliza a alocação dinâmica de memória para criar uma nova instância da estrutura Pessoa. Em seguida, realiza a cópia do nome, atribui os pais, aloca e inicializa uma fila de irmãos. Ao final, retorna o ponteiro para a pessoa recém-criada. Essa abordagem simplifica o processo de adição de novos membros à árvore genealógica, cuidando dos detalhes de alocação e inicialização de forma eficiente.

```
Pessoa *criarPessoa(const char *nome, Pessoa *pai, Pessoa *mae) {
    Pessoa *novaPessoa = (Pessoa *)malloc(sizeof(Pessoa));
    strcpy(novaPessoa->nome, nome);
    novaPessoa->pai = pai;
    novaPessoa->mae = mae;
    novaPessoa->irmaos = (Fila *)malloc(sizeof(Fila));
    inicializarFila(novaPessoa->irmaos);
    return novaPessoa;
}
```

Função adicionarPessoa

Envolve a alocação dinâmica de memória para criar uma nova instância da estrutura Pessoa. Os dados dessa pessoa, como nome e outros atributos, são então atribuídos. Após a criação, a função configura os relacionamentos familiares, como pai, mãe e irmãos, se houver, e integra a nova pessoa à árvore. Essa função facilita a expansão da árvore genealógica, garantindo que os vínculos familiares sejam estabelecidos corretamente.

```
void adicionarPessoa(ListaPessoas **lista, Pessoa *pessoa) {
    ListaPessoas *novoNode = (ListaPessoas *)malloc(sizeof(ListaPessoas));
    novoNode->pessoa = pessoa;
    novoNode->proximo = *lista;
    *lista = novoNode;
}
```

Função exibirArvoreGenealogica

Percorre a estrutura hierárquica da árvore, começando pela raiz. Durante esse processo, ela imprime os dados de cada pessoa, como nome e informações relevantes. Utilizando uma abordagem recursiva, a função visita os descendentes de cada pessoa, garantindo que toda a árvore seja apresentada. Esse processo proporciona uma visão organizada e compreensível da estrutura genealógica, revelando os relacionamentos familiares e a hierarquia entre os indivíduos na árvore.

```
void exibirArvoreGenealogica(Pessoa *pessoa, int nivel) {
   if (pessoa != NULL) {
       for (int i = 0; i < nivel; i++) {
           printf("  ");
       }
       printf("%s", pessoa->nome);

       if (pessoa->irmaos != NULL && pessoa->irmaos->frente != NULL) {
           Node *atual = pessoa->irmaos->frente;
           printf(" - Irmãos - ");
           while (atual != NULL) {
               printf("%s ", atual->pessoa->nome);
               atual = atual->prox;
           }
       }

       printf("\n");

       exibirArvoreGenealogica(pessoa->pai, nivel + 1);
       exibirArvoreGenealogica(pessoa->mae, nivel + 1);
   }
}
```

Função buscarPessoa

Utiliza um método recursivo para percorrer a árvore a partir do nó raiz. Ao receber o nome da pessoa a ser encontrada, a função compara o nome atual com o desejado. Se a pessoa é encontrada, seus dados são retornados. Caso contrário, a busca é realizada de forma recursiva nos nós pai e mãe. Essa função oferece uma maneira eficaz de localizar indivíduos na árvore genealógica com base em seus nomes, proporcionando uma visão clara das relações familiares.

```
Pessoa *buscarPessoa(Pessoa *raiz, const char *nome) {
    if (raiz == NULL) {
        return NULL;
    }

    if (strcmp(raiz->nome, nome) == 0) {
        return raiz;
    }

    Pessoa *pessoaEncontrada = buscarPessoa(raiz->pai, nome);
    if (pessoaEncontrada == NULL) {
        pessoaEncontrada = buscarPessoa(raiz->mae, nome);
    }

    return pessoaEncontrada;
}
```

Função liberarMemoria

Utiliza uma abordagem recursiva para percorrer a árvore, começando pelo nó raiz. Durante esse processo, a função libera dinamicamente a memória alocada para cada pessoa na árvore, garantindo que todos os nós sejam devidamente desalocados. Essa função é essencial para evitar vazamentos de memória, assegurando que todos os recursos alocados dinamicamente sejam liberados ao finalizar o programa.

```
void liberarMemoria(ListaPessoas *lista) {
    while (lista != NULL) {
        ListaPessoas *temp = lista;
        lista = lista->proximo;
        free(temp->pessoa->irmaos);
        free(temp->pessoa);
        free(temp);
    }
}
```

Função imprimirIrmaosEFamilia

Tem o objetivo de imprimir informações sobre a família de uma pessoa na árvore genealógica. Ela vai verificar se a pessoa fornecida não é nula, e, em caso positivo, imprime o nome da pessoa, o nome do pai e da mãe, e uma lista de irmãos, se

houver. Caso não haja informações sobre o pai, a mãe ou irmãos, a função vai lidar com esses casos de forma apropriada, indicando "Desconhecido" ou "Nenhum irmão". Se a pessoa fornecida for nula, a função irá imprimir uma mensagem informando que a pessoa não foi encontrada.

```
void imprimirIrmaosEFamilia(Pessoa *pessoa) {
   if (pessoa != NULL) {
       printf("Família de %s:\n", pessoa->nome);
       printf("  Pai: %s\n", (pessoa->pai != NULL) ? pessoa->pai->nome :
"Desconhecido");
       printf("  Mãe: %s\n", (pessoa->mae != NULL) ? pessoa->mae->nome :
"Desconhecido");
       printf("  Irmãos: ");

       if (pessoa->irmaos != NULL && pessoa->irmaos->frente != NULL) {
           Node *atual = pessoa->irmaos->frente;
           while (atual != NULL) {
               printf("%s ", atual->pessoa->nome);
               atual = atual->prox;
           }
       } else {
           printf("Nenhum irmão");
       }

       printf("\n");
   } else {
       printf("Pessoa não encontrada.\n");
   }
}
```

Função calcularAlturaArvore

Vai utilizar uma abordagem recursiva para percorrer a árvore, começando pelo nó raiz. A cada nível, a função determina a altura máxima entre as subárvores esquerda e direita. O processo continua até atingir as folhas da árvore. O resultado final é a altura total da árvore, proporcionando uma medida da profundidade e complexidade da estrutura genealógica. Essa função é essencial para avaliar a organização vertical da árvore e compreender a extensão das relações familiares.

```
int calcularAlturaArvore(Pessoa *raiz) {
   if (raiz == NULL) {
       return -1;
```

```
    }
    int alturaPai = calcularAlturaArvore(raiz->pai);
    int alturaMae = calcularAlturaArvore(raiz->mae);

    return 1 + ((alturaPai > alturaMae) ? alturaPai : alturaMae);
}
```

3 - Dificuldades Encontradas

- Uma das maiores dificuldades foi encontrar conteúdo sobre árvore genealógica, pois o seu conteúdo sobre esse assunto era bem escasso.

- Enfrentei certa dificuldade ao lidar com a formatação da impressão da árvore genealógica, resultando em saídas incorretas. Esse desafio exigiu um esforço significativo para identificar e corrigir o erro no processo de exibição.

- Tive bastante dificuldades durante a implementação da pilha de operações. Ao testarmos a função "desfazer", me deparei com erros que resultaram em bugs em todo o código, comprometendo seriamente a lógica da árvore genealógica. Infelizmente, apesar de meus esforços,não consegui desenvolver uma solução e não foi possível implementar tal função no código .