Sistema de Catálogo Nacional de Telefones e Endereços
Relatório de Estrutura de Dados I

Camila Andrade de Sena

Centro de Ciências e Tecnologia
Universidade Estadual do Ceará (UECE) – Fortaleza, CE – Brazil

andrade.sena@aluno.uece.br

Abstract. *This report aims to describe the decisions and code structures revolving around the implementation of a National Telephone and Address Directory System. In this regard, it dealt with the description of each structure used, as well as the functions and why they were used at any given time. Therefore, an overview of the developed programming project will be presented.*

Resumo. *O presente relatório tem como objetivo descrever as decisões e estruturas de códigos revolvendo em torno da implementação de um Sistema de Catálogo Nacional de Telefones e Endereços. Neste sentido, tratou-se da descrição de cada estrutura utilizada, assim como as funções e o porquê do uso das mesmas em cada dado momento. Portanto, uma visão panorâmica do projeto de programação desenvolvido será apresentada.*

1. Descrição do ambiente de desenvolvimento

O projeto foi desenvolvido no ambiente com a descrição a seguir.

a) Processador: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz

b) Quantidade de memória: 8,00 GB

c) Sistema operacional: Windows 11 64-bit

d) Linguagem de programação: C

e) Compilador (versão e IDE): Code::Blocks - Versão 20.03-r11983

2. As Estruturas

Inicialmente, foi criado um projeto de aplicação de console, contendo três arquivos: uma biblioteca (.h) chamada "TrabalhoED1.h", um .c chamado "TrabalhoED1.c" e finalmente o main.c. Na biblioteca, ficarão armazenados os protótipos das funções das três estruturas que serão descritas, além das funções que foram criadas em adição a elas. No .c serão declarados os tipos de dados e implementadas todas as funções. O objetivo dessa separação de arquivos é implementar um Tipo Abstrado de Dados, ou TAD.

Ademais, como o compilador foi o Code::Blocks, no começo da implementação houve a formação de um arquivo ".cbp", que quando aberto contém todos os arquivos pertencentes ao projeto.

2.1. Árvore AVL

As primeiras funções a serem criadas foram as da Árvore AVL, que é um tipo de Árvore Binária balanceada com relação à altura de suas subárvores. Assim como a Árvore Binária, cada nó terá dois filhos, o da esquerda com valor menor ao seu e o da direita com valor maior.

Para o código que será desenvolvido ao longo desse trabalho, cada Árvore possuirá um nó raiz com um DDD existente no Brasil, e não serão permitidas repetições de DDDs. Cada nó subsequente dessa Árvore conterá um número de 8 dígitos pertencente à esse DDD, o nome do usuário correspondente e o seu endereço. Visto isso, cada vez que a função de inserção for chamada para um novo usuário, o DDD será solicitado antes. Não serão permitidas repetições de número e nome também.

As funções intrínsecas da Árvore, como por exemplo criação, destruição, inserção e outras do tipo foram implementadas de forma semelhante ao que está contido no livro "Estrutura de dados descomplicada em linguagem C", do autor André Ricardo Backes. No entanto, cada nó conterá uma struct, e não um inteiro.

Na Árvore ordenada por número, as comparações serão feitas com o número de telefone que está dentro da struct, do tipo inteiro. Já na Árvore ordenada por nome, essas comparações são feitas com o nome de usuário, uma string também contida na struct. Nesse caso, será utilizada a função **strcasecmp()**, que não é case sensitive e compara uma string com outra.

2.2. Lista Dinâmica

As funções implementadas em sequência foram as de Lista Dinâmica, que novamente seguiram o padrão apresentado no livro do autor André Backes. Contudo, cada elemento dessa lista será o endereço de uma Árvore criada previamente, e não uma struct como está na literatura citada. Como a raiz da Árvore muda devido ao balanceamento e deixa de ser o DDD, também foi guardado na lista um inteiro que recebe o DDD e será usado para a busca no futuro.

Essas funções serão usadas apenas dentro dos códigos referentes à Tabela Hash, uma vez que o tratamento de colisão será feito por Encadeamento Separado.

2.3. Tabela Hash

Por último, foram implementadas as funções de Tabela Hash, o método de busca que será utilizado para encontrar rapidamente cada Árvore. Como serão trabalhadas duas Árvores, uma ordenada por número e outra por nome, foram criadas duas Tabelas Hash correspondentes. A obra literária utilizada para a criação das tabelas foi a mesma das estruturas anteriores, entretanto houveram mudanças maiores que apenas o tipo de dado em cada posição da tabela.

Como se foi solicitado, o tratamento de colisão será dado por Encadeamento Separado, o que significa que cada posição da tabela apontará para uma Lista Dinâmica. A função utilizada para calcular a chave será a de divisão, que retorna o resto da divisão entre um número e o tamanho da tabela. Esse número, para o trabalho a seguir, será sempre o DDD fornecido. O tamanho de ambas as tabelas foi definido como 101, pois é o maior número primo mais próximo de 99, o maior DDD nacional. Dessa maneira, serão evitadas ao máximo as possíveis colisões.

3. O Início do Programa

No programa principal, antes de tudo é chamada a função **srand(time(NULL))**, que não será usada imediatamente, mas apenas quando for solicitada a inserção de um número. Mais explicações serão dadas na subseção 4.1.

Então, são declaradas as variáveis que serão usadas imediatamente. Os inteiros "i" e "j" farão parte dos ciclos **for**, "quantidade" receberá o número de DDDs que o usuário deseja inserir, "opcao" receberá a opção de escolha do usuário quando apresentado o menu mais a frente, e "ddd" receberá o DDD que vai ser inserido em cada uma das árvores. O próximo passo será a criação da quantidade escolhida de Árvores e as duas Tabelas Hash mencionadas anteriormente.

```
    srand(time(NULL)); // Gera seeds diferentes para a função rand()
usada na funcao de insercao

    int i,j,quantidade,opcao,ddd;

    printf("\nQual a quantidade de DDDs que voce deseja inserir?\n");
    scanf("%i",&quantidade);

    ArvAVL *ordenadaNumero[quantidade];
    ArvAVL *ordenadaNome[quantidade];

    Hash* hashOrdenadoNumero=criaHash(MAX);
    Hash* hashOrdenadoNome=criaHash(MAX);
```

Figura 1. Declaração de variáveis e criação de estruturas

Quantas vezes indicar a quantidade inserida, um ciclo **for** acontecerá a seguir. Em cada rodada, um DDD é pedido. É feita uma consulta para ver se ele já não foi inserido previamente, se sim, o programa encerrará. Caso contrário, será feita uma tentativa de inserir o DDD em uma Árvore com a função **insere_ArvAVL_DDD**.

Essa função relaciona cada DDD nacional com uma macroregião e seu estado, com dados atualizados da Anatel. Por exemplo, se o usuário escolher o DDD 99, será inserida nas árvores uma struct com o número 99, o nome Imperatriz e o endereço Maranhão. O retorno será 1. Se o DDD não existir, por exemplo "03", o retorno será 0.

```
case 99:
    {
    struct usuario dddDados={99,"Imperatriz","Maranhao"};
    insere_ArvAVL(raiz,dddDados);
    insere_ArvAVL_porNome(raiz2,dddDados);
    return 1;
    }
default:
    printf("\nDDD inexistente no Brasil! \n");
    return 0;
    }
```

Figura 2. Exemplo de funcionamento da função de inserção de DDD

De volta ao ciclo, uma variável “teste” receberá esse retorno, e em caso de não existência o programa será encerrado. O endereço das Árvores são inseridos em suas Tabelas correspondentes ao final, apenas se os DDDs existirem. O ciclo acaba e uma mensagem de sucesso é exibida. O menu é chamado até que o usuário deseje sair ou escolha parar de imprimí-lo.

```
for(i=0;i<quantidade;i++){
    // Cria as Árvores AVL ordenadas
    ordenadaNumero[i]=cria_ArvAVL();
    ordenadaNome[i]=cria_ArvAVL();
    printf("\nInsira o DDD: \n");
    scanf("%i",&ddd); // Recebe o DDD que vai ser inserido na raiz
    // Encerra o programa caso o DDD inserido seja repetido
    for(j=0;j<i;j++){
        if(consulta_ArvAVL(ordenadaNumero[j],ddd)||
        consulta_ArvAVL(ordenadaNome[j],ddd))
        {
            printf("\nEsse DDD ja foi inserido!");
            return 0;
        }
    }
    //Insere o DDD nas arvores
    int teste=
    insere_ArvAVL_DDD(ordenadaNumero[i],ordenadaNome[i],ddd);
    //Encerra o programa caso o DDD inserido não exista no Brasil
    if(!teste){
        return 0;
    }
    // Insere as Árvores nas tabelas hash
   insereHash_EncadSeparado(hashOrdenadoNumero,ordenadaNumero[i]);
   insereHash_EncadSeparado(hashOrdenadoNome,ordenadaNome[i]);
}
```

Figura 3. Ciclo de inserção das árvores

4. O Menu

O menu contém seis opções, cinco delas correspondentes a cada tipo de acesso à informação pedidos nas especificações do trabalho. Não segue a mesma ordem.

1) Insere um usuário no DDD escolhido caso não haja repetição de número e nome. Caso o número já exista na Árvore, três números que ainda não foram usados serão disponibilizados para escolha. Após a escolha, o usuário é inserido.

2) Remove um usuário do DDD escolhido e mostra as alturas de ambas as árvores, ordenadas por número e nome, antes e depois da remoção.

3) Encontra um usuário do DDD escolhido com base no telefone informado, após isso mostra todas suas informações.

4) Mostra todos os usuários do DDD escolhido, de acordo com todas as três ordens disponíveis para percurso na Árvore.

5) Mostra a lista telefônica completa, por DDD. No caso, imprime todas as Árvores de todos os DDDs, ordenadas por número e nome.

0) Sai e encerra o programa

```
void imprimirMenu(){
    printf("\nO que voce deseja fazer?\n");
    printf("\n|-----------------------------------------------------------------|\n");
    printf("|                                                                 |\n");
    printf("|          ( 1 ) Inserir numero de telefone                       |\n");
    printf("|                                                                 |\n");
    printf("|          ( 2 ) Remover numero de telefone                       |\n");
    printf("|                                                                 |\n");
    printf("|          ( 3 ) Buscar numero de telefone                        |\n");
    printf("|                                                                 |\n");
    printf("|          ( 4 ) Mostrar numeros por DDD                          |\n");
    printf("|                                                                 |\n");
    printf("|          ( 5 ) Mostrar lista telefonica completa |\n");
    printf("|                                                                 |\n");
    printf("|          ( 0 ) Sair                                             |\n");
    printf("|                                                                 |\n");
    printf("|-----------------------------------------------------------------|\n");
    printf("\nInsira a opcao:\n");
}
```

Figura 4. Função que imprime o menu

Ao fim de cada escolha, exceto a última, o programa pergunta ao usuário se ele deseja ver o menu novamente. Na última o programa é apenas encerrado.

4.1. Opção 1 – Inserir e mostrar 3 números não utilizados

Quando a opção 1 é escolhida, inicialmente a tela é limpada, o DDD é pedido e armazenado na variável “ddd”. Então a primeira função da opção 1, com o nome de **opcao1Menu**, é chamada. Ela recebe como parâmetro as duas Tabelas Hash e o DDD que foi lido.

```
if(opcao==1){
    system("cls");
    printf("\nInsira o DDD:\n");
    scanf("%d",&ddd);
    // Insere o número nas arvores e mostra 3 opções de
    números aleatórios em caso de repetição
    opcao1Menu(hashOrdenadoNumero,hashOrdenadoNome,ddd);
}
```

Figura 5. Primeira parte da opção 1

Dentro dessa função são criadas duas Árvores que irão receber o retorno da busca do DDD na Tabela Hash. Uma será a ordenada por número e a outra por nome. Caso a busca tenha sido um sucesso, ou seja, o DDD tiver sido inserido previamente, a execução da função continuará. Caso contrário, uma mensagem de erro aparecerá, a função encerrará e será perguntado ao usuário se ele gostaria que o menu fosse imprimido novamente.

Continuando a execução, um número ficará sendo pedido enquanto ele não possuir 8 dígitos. Esse é o formato de número nacional. Uma vez atingido essa formatação de 8 dígitos, a função **insere_ArvAVL_semRepeticao** é chamada. Ela receberá como parâmetro as duas Árvores que foram retornadas anteriormente e o número digitado.

```
void opcao1Menu(Hash *hashAVLNumero, Hash *hashAVLNome, int ddd){
    int numeroTelefone;
    ArvAVL *numero,*nome;
    numero=cria_ArvAVL();
    nome=cria_ArvAVL();
    int teste=buscaHash_EncadSeparado(hashAVLNumero,ddd,&numero);
    int teste2=buscaHash_EncadSeparado(hashAVLNome,ddd,&nome);

    // Pede o número se o DDD existir em alguma Árvore
    if(teste&&teste2){
        printf("\nInsira o numero de 8 digitos:\n");
        char numeroTelefoneString[10];
        do
        {
            scanf("%10s", numeroTelefoneString);
            if(strnlen(numeroTelefoneString, 10) != 8)
            {
                printf("\nEsse numero nao possui 8 digitos!\n");
                printf("\nInsira o numero de 8 digitos:\n");
            }
        }while(strlen(numeroTelefoneString)!= 8);
        numeroTelefone=atoi(numeroTelefoneString);
        // Insere o número nas duas Árvores
        insere_ArvAVL_semRepeticao(numero,nome,numeroTelefone);
    }
    else{
        printf("\nEsse DDD nao foi inserido previamente!\n");
    }
}
```

Figura 6. Segunda parte da opção 1

Dentro dessa nova função, é checado se o número recebido já existe dentro da Árvore. Se ele não existir, é pedido então o nome do usuário (que também não pode ser repetido, a checagem também é feita) e o seu endereço. Essas informações são copiadas para uma struct e essa struct é então inserida em ambas as Árvores.

Vale relembrar que daqui para frente, sempre que for falado de inserção ou remoção em uma Árvore ordenada por nome, as funções usadas não serão as mesmas da Árvore ordenada por número. Os dois processos são feitos separadamente. O motivo disso é que como dito anteriormente, as comparações não são mais feitas pelo número de telefone quando se trata da ordem por nome, e sim pela string do nome.

Para esse fim será usada a função **strcasecmp()**, uma função de comparação que não diferencia letras maiúsculas e minúsculas, e recebe como parâmetro duas strings. Se a primeira string for menor que a segunda, ela retorna <0. Se for maior, >0. E se forem iguais, retorna 0.

```
int insere_ArvAVL_semRepeticao(ArvAVL *raiz, ArvAVL *raiz2,int
numero){
    char nome[200], endereco[200];
    // Insere o número nas Árvores caso ele não exista
    if(!consulta_ArvAVL(raiz,numero)){
            struct usuario novo;
            novo.numero=numero;
            printf("\nInsira o nome:\n");
            fflush(stdin);
            gets(nome);
            while(consulta_ArvAVL_porNome(raiz2,nome)){
                  printf("\nEsse nome ja foi utilizado! Insira
                  outro:\n");
                  gets(nome);
            }
            strcpy(novo.nome,nome);
            printf("\nInsira o endereco:\n");
            fflush(stdin);
            gets(endereco);
            strcpy(novo.endereco,endereco);
            insere_ArvAVL(raiz,novo);
            insere_ArvAVL_porNome(raiz2,novo);
            printf("\nNumero inserido com sucesso!\n");
            return 1;
    }
```

Figura 7. Começo da função de inserção sem repetição nas Árvores. No primeiro teste, o número ainda não existe na árvore.

Caso contrário, se o número já existir, são gerados três números aleatórios diferentes entre si e do número que foi inserido previamente. Tais números são gerados com a função **geradorNumero8Digitos**, que funciona gerando duas partes de quatro dígitos com a função **rand**. Essa última função, **rand**, é o motivo pelo qual **srand(time(NULL))** foi chamado bem no começo da main. Se isso não fosse feito, todos os números aleatórios gerados em tempo de execução seriam iguais.

```
int geradorNumero8Digitos(){
    int parte1= 1000+(rand()%9000);
    int parte2= 1000+(rand()%9000);
    char parte1string[4];
    char parte2string[4];
    sprintf(parte1string, "%i", parte1);
    sprintf(parte2string, "%i", parte2);
    strcat(parte1string,parte2string);
    int numero=atoi(parte1string);
    return numero;
}
```

Figura 8. Gerador de número de 8 dígitos

De volta a função de inserção sem repetição, as três opções de números são mostradas e o usuário deve escolher entre uma das três. Feito isso, o mesmo processo de pedir as informações de nome, endereço, copiar para uma struct e colocar nas Árvores será repetido. Ao final, uma mensagem de sucesso será exibida.

```
else{
    // Mostra 3 opções de número que não existem na Árvore
    // Pede as informações do usuário após ele escolher uma
  opção
    printf("\nEsse numero ja existe! Escolha um desses 3:\n");
    int opcao,opcao1, opcao2, opcao3;
    // Gera 3 opções de números aleatórios
    do
    {
        opcao1=geradorNumero8Digitos();
    } while (opcao1==numero);
    do
    {
        opcao2=geradorNumero8Digitos();
    } while (opcao2==numero&&opcao2==opcao1);
    do
    {
        opcao3=geradorNumero8Digitos();
    } while (opcao3==numero&&opcao3==opcao1&&opcao3==opcao2);
    printf("(1) - %d\n",opcao1);
    printf("\n(2) - %d\n",opcao2);
    printf("\n(3) - %d\n",opcao3);
    printf("\nInsira a opcao referente ao numero desejado:\n");
    scanf("%d",&opcao);
    while(opcao!=1&&opcao!=2&&opcao!=3){
        printf("\nAs opcoes validas sao 1, 2 ou 3.\n");
        scanf("%d",&opcao);
    }
    if(opcao==1){
        struct usuario novo;
        novo.numero=opcao1;
        printf("\nInsira o nome:\n");
        fflush(stdin);
        gets(nome);
        while(consulta_ArvAVL_porNome(raiz2,nome)){
              printf("\nEsse nome ja foi utilizado! Insira
              outro:\n");
              gets(nome);
        }
        strcpy(novo.nome,nome);
        printf("\nInsira o endereco:\n");
        fflush(stdin);
        gets(endereco);
        strcpy(novo.endereco,endereco);
        insere_ArvAVL(raiz,novo);
        insere_ArvAVL_porNome(raiz2,novo);
        printf("\nNumero inserido com sucesso!\n");
        return 1;
    }
    else if(opcao==2){
        struct usuario novo;
        novo.numero=opcao2;
        printf("\nInsira o nome:\n");
        fflush(stdin);
        gets(nome);
        while(consulta_ArvAVL_porNome(raiz2,nome)){
              printf("\nEsse nome ja foi utilizado! Insira
              outro:\n");
              gets(nome);
        }
```

```
                strcpy(novo.nome,nome);
                printf("\nInsira o endereco:\n");
                fflush(stdin);
                gets(endereco);
                strcpy(novo.endereco,endereco);
                insere_ArvAVL(raiz,novo);
                insere_ArvAVL_porNome(raiz2,novo);
                printf("\nNumero inserido com sucesso!\n");
                return 1;
            }
            else if(opcao==3){
                struct usuario novo;
                novo.numero=opcao3;
                printf("\nInsira o nome:\n");
                fflush(stdin);
                gets(nome);
                while(consulta_ArvAVL_porNome(raiz2,nome)){
                      printf("\nEsse nome ja foi utilizado! Insira
                      outro:\n");
                      gets(nome);
                }
                strcpy(novo.nome,nome);
                printf("\nInsira o endereco:\n");
                fflush(stdin);
                gets(endereco);
                strcpy(novo.endereco,endereco);
                insere_ArvAVL(raiz,novo);
                insere_ArvAVL_porNome(raiz2,novo);
                printf("\nNumero inserido com sucesso!\n");
                return 1;
            }
        }
        return 0;
}
```

Figura 9. Final da função de inserção nas Árvores sem repetição

4.2. Opção 2 – Remover e mostrar altura antes e depois

Quando a opção 2 é escolhida, inicialmente a tela é limpada, o DDD é pedido e armazenado na variável “ddd”. Então a primeira função da opção 2, com o nome de **opcao2Menu**, é chamada. Ela recebe como parâmetro as duas Tabelas Hash e o DDD que foi lido.

```
        else if(opcao==2){
            system("cls");
            printf("\nInsira o DDD:\n");
            scanf("%d",&ddd);
            // Remove o número das Árvores e imprime as alturas de
            antes e depois da operação
            opcao2Menu(hashOrdenadoNumero,hashOrdenadoNome,ddd);
        }
```

Figura 10. Primeira parte da opção 2

O começo dessa função é bem semelhante ao começo da função da opção 1, que foi descrita na subseção anterior. Duas Árvores são criadas para receber o retorno da busca nas Tabelas, caso a busca não ache nada, uma mensagem de erro aparece e em seguida é perguntado ao usuário se ele deseja voltar ao menu. Caso contrário, a busca sendo um sucesso, um número é pedido e a função **remove_mostraAltura_ArvAVL** é chamada. Essa função recebe as duas Árvores que foram retornadas na busca e o número desejado para a remoção.

```
void opcao2Menu(Hash *hashAVLNumero, Hash *hashAVLNome, int ddd){
    int numeroTelefone;
    ArvAVL *numero;
    ArvAVL *nome;
    numero=cria_ArvAVL();
    nome=cria_ArvAVL();
    int teste=buscaHash_EncadSeparado(hashAVLNumero,ddd,&numero);
    int teste2=buscaHash_EncadSeparado(hashAVLNome,ddd,&nome);
    // Pede o número se o DDD existir em alguma Árvore
    if(teste&&teste2){
        printf("\nInsira o numero:\n");
        scanf("%i",&numeroTelefone); // Recebe o número a ser removido
        // Remove o número das duas árvores
        // Imprime as alturas
        remove_mostraAltura_ArvAVL(numero,nome,numeroTelefone);
    }
    else{
        printf("\nEsse DDD nao foi inserido previamente!\n");
    }
}
```

Figura 11. Segunda parte da opção 2

Já dentro da função que foi chamada, é criada uma nova struct chamada **remocao**. Uma vez feito isso, é realizada uma consulta na Árvore ordenada por número, para buscar o número e devolver a struct a qual ele pertence para a struct **remocao** passada como parâmetro.

A função de consulta em questão é uma modificação da função de consulta em Árvore encontrada na literatura de referência citada na subseção 2.1. Enquanto a que está na literatura apenas retorna 1 ou 0 em caso de sucesso ou fracasso, respectivamente, a nova função faz esses retornos e também modifica a struct que foi passada como parâmetro.

Se esse número realmente existir na Árvore, a struct que foi modificada pela função de consulta é removida de ambas as Árvores, e suas alturas de antes e após desse processo são mostradas. Caso contrário, as alturas são mostradas juntamente com uma mensagem de erro.

```
int remove_mostraAltura_ArvAVL(ArvAVL *raiz, ArvAVL *raiz2,int
numero){
    struct usuario remocao; // Verifica se o número existe em alguma
    Árvore e devolve uma Struct com suas informações correspondentes
    int consulta=consulta_ArvAVL_devolveStruct(raiz,numero,&remocao);

      // Remove o número existente e imprime as alturas
    if(consulta){
        printf("\nAltura da arvore ordenada por numero antes: %i\n",
      altura_ArvAVL(raiz));
        printf("\nAltura da arvore ordenada por nome antes: %i\n",
      altura_ArvAVL(raiz2));
        remove_ArvAVL(raiz,remocao);
        remove_ArvAVL_porNome(raiz2,remocao);
        printf("\n------------------------------------------------------\n");
        printf("\n         Numero removido com sucesso!\n");
        printf("\n------------------------------------------------------\n");
        printf("\nAltura da arvore ordenada por numero depois: %i\n",
      altura_ArvAVL(raiz));
        printf("\nAltura da arvore ordenada por nome depois: %i\n",
      altura_ArvAVL(raiz2));
        return 1;
    }
      // Número não existe
    else{
        printf("\nAltura da arvore ordenada por numero antes: %i\n",
      altura_ArvAVL(raiz));
        printf("\nAltura da arvore ordenada por nome antes: %i\n",
      altura_ArvAVL(raiz2));
        printf("\n------------------------------------------------------\n");
        printf("\n         Esse numero nao existe na arvore!\n");
        printf("\n------------------------------------------------------\n");
        printf("\nAltura da arvore ordenada por numero depois: %i\n",
      altura_ArvAVL(raiz));
        printf("\nAltura da arvore ordenada por nome depois: %i\n",
      altura_ArvAVL(raiz2));
        return 1;
    }
    return 0;
}
```

Figura 12. Função que remove e mostra as alturas das Árvores

4.3. Opção 3 – Encontrar nome e endereço a partir de número de telefone

A opção 3 é a primeira que apresenta uma mudança inicial em relação as duas apresentadas anteriormente. Quando escolhida, a tela é limpada, o DDD é pedido e armazenado na variável correspondente e a função **opcao3Menu** é chamada. No entanto, apenas uma Tabela Hash é passada como parâmetro, juntamente com o DDD. Isso acontece porque agora não serão feitas nenhuma operação de inserção ou remoção, apenas uma busca. Para qualquer uma das duas Tabela Hash o resultado seria o mesmo. A mesma lógica será seguida para a Opção 4, que será apresentada na próxima subseção.

```
        else if(opcao==3){
            system("cls");
            printf("\nInsira o DDD:\n");
            scanf("%d",&ddd);
            // Devolve número e endereço a partir do telefone
            opcao3Menu(hashOrdenadoNumero,ddd);
        }
```

Figura 13. Primeira parte da opção 3

Dentro dessa função, a Árvore correspondente ao DDD é buscada na Tabela Hash. Caso a busca falhe, uma mensagem de erro é mostrada e a opção de mostrar o menu novamente é dada ao usuário.

Caso contrário, a Árvore existindo, um número de telefone é pedido ao usuário e armazenado na variável correspondente. Então a mesma função modificada de consulta na Árvore que foi citada na subseção anterior é chamada, retornando uma struct com os dados do usuário correspondente ao número recebido. Se esse usuário existir, seus dados são impressos e a opção de retornar ao menu é dada. Se ele não existir, novamente uma mensagem de erro é mostrada juntamente com a opção de retornar ao menu.

```
void opcao3Menu(Hash *hashAVLNumero, int ddd){
    int numeroTelefone;
    ArvAVL *numero;
    numero=cria_ArvAVL();
    int teste=buscaHash_EncadSeparado(hashAVLNumero,ddd,&numero);
    // Pede o número caso o DDD exista em alguma Árvore
    if(teste){
        printf("\nInsira o numero:\n");
        scanf("%d",&numeroTelefone);
        struct usuario dados;
        // Busca o usuário do telefone e devolve suas
        informações
        int    teste2=consulta_ArvAVL_devolveStruct
        (numero,numeroTelefone,&dados);
        if(teste2){
            printf("\nAqui esta o usuario buscado:\n\n");
            printf("Nome: %s\n",dados.nome);
            printf("Endereco: %s\n",dados.endereco);
            printf("Numero: %d\n",dados.numero);
        }
        else{
        printf("\nNao ha nenhum usuario com esse numero!\n");
      // Usuário não existe
        }
    }
    else{
      printf("\nEsse DDD nao foi inserido previamente!\n"); // DDD
      não existe
    }
}
```

Figura 14. Segunda parte da opção 3

4.4. Opção 4 – Lista os usuários de um DDD pelos três percursos

Uma vez que a opção 4 é escolhida, a tela é limpada, o DDD é solicitado e armazenado em sua variável correspondente e a função **opcao4Menu** é chamada. Como dito na subseção anterior, apenas a Tabela Hash correspondente as Árvores ordenadas por número será passada como parâmetro, juntamente com DDD lido.

```
else if(opcao==4){
            system("cls");
            printf("\nInsira o DDD:\n");
            scanf("%d",&ddd);
            // Lista todos os números de um DDD por 3 percursos
            opcao4Menu(hashOrdenadoNumero,ddd);
        }
```

Figura 15. Primeira parte da opção 4

Dentro dessa função, o DDD é buscado dentro da Tabela e caso ele exista, sua Árvore é retornada e impressa em todos os percursos disponíveis. Caso contrário uma mensagem de erro é exibida e a opção de mostrar o menu novamente é dada ao usuário.

```
void opcao4Menu(Hash *hashAVLNumero, int ddd){
    ArvAVL *numero;
    numero=cria_ArvAVL();
    int teste=buscaHash_EncadSeparado(hashAVLNumero,ddd,&numero);
    // Imprime todas as informações de um DDD existente
    if(teste){
      printf("\n================================================\n");
      printf("    Percurso Pre-Ordem   ");
      printf("\n================================================\n");

      printf("\n-------------------------------------------------------------------\n");
      preOrdem_ArvAVL(numero);
      printf("-------------------------------------------------------------------\n");

      printf("\n================================================\n");
      printf("    Percurso Em-Ordem   ");
      printf("\n================================================\n");

      printf("\n-------------------------------------------------------------------\n");
      emOrdem_ArvAVL(numero);
      printf("-------------------------------------------------------------------\n");

      printf("\n================================================\n");
      printf("    Percurso Pos-Ordem   ");
      printf("\n================================================\n");

      printf("\n-------------------------------------------------------------------\n");
      posOrdem_ArvAVL(numero);
      printf("-------------------------------------------------------------------\n");
    }
    else{
      printf("\nEsse DDD nao foi inserido previamente!\n"); //ddd nao
      existente
    }
}
```

Figura 16. Segunda parte da opção 4

4.5. Opção 5 – Gerar uma lista telefônica ordenada por número e nome

Para a opção 5, o processo será levemente diferente dos anteriores. A lista telefônica completa ser impressa significa que todas as Árvores devem ser impressas, tanto em suas ordenações por número quanto por nome. Visto isso, não é necessário passar como parâmetro as Tabelas para uma busca específica, e sim as Árvores em sua totalidade. Portanto, entrando nessa opção, a tela é limpada e a função **opcao5Menu** é chamada, recebendo como parâmetro as Árvores e a quantidade em que foram criadas.

```
else if(opcao==5){
            system("cls");
            // Gera uma lista telefônica por DDD
            opcao5Menu(ordenadaNumero,ordenadaNome,quantidade);
        }
```

Figura 17. Primeira parte da opção 5

Dentro da função, dois ciclos **for** serão iterados até a quantidade de Árvores recebida. Em cada iteração, uma Árvore será impressa, primeiro em sua ordenação por número de telefone e depois por nome.

```
void opcao5Menu(ArvAVL *raiz[], ArvAVL *raiz2[], int quantidade){

      printf("\n================================================\n");
      printf("    Lista telefonica ordenada por numero de telefone");
      printf("\n================================================\n");

      for(int i=0;i<quantidade;i++){
        printf("\n--------------------------------------------------------------------\n");
        emOrdem_ArvAVL(raiz[i]);
        printf("--------------------------------------------------------------------\n");
      }

      printf("\n================================================\n");
      printf("    Lista telefonica ordenada por nome do usuario");
      printf("\n================================================\n");

      for(int i=0;i<quantidade;i++){
        printf("\n--------------------------------------------------------------------\n");
        emOrdem_ArvAVL(raiz2[i]);
        printf("--------------------------------------------------------------------\n");
    }
}
```

Figura 18. Segunda parte da opção 5

5. O Final do Programa

Evidentemente, se a opção escolhida não estiver no menu uma mensagem de erro será mostrada ao usuário, assim como a opção de voltar ao menu.

Tudo terminado, resta liberar a memória das estruturas criadas.

```
for(i=0;i<quantidade;i++){
        libera_ArvAVL(ordenadaNome[i]);
        libera_ArvAVL(ordenadaNumero[i]);
    }
    liberaHash(hashOrdenadoNumero);
    liberaHash(hashOrdenadoNome);
    return 0;
}
```

Figura 19. Memória sendo liberada

As funções de liberar Árvore são chamadas para cada uma delas, e ao final do ciclo as duas Tabelas também tem sua memória liberada. Por fim, **return 0** servirá pra indicar a ausência de erros no programa.

Referências

Backes, A. (2013) “Linguagem C: completa e descomplicada”, Edited by Elsevier, Rio de Janeiro.

Backes, A. (2016) “Estrutura de dados descomplicada: em linguagem C”, Edited by Elsevier, Rio de Janeiro.

Instinto Viajante. (2019) “DDDs do Brasil: lista de DDD completa e ATUALIZADA”, https://instintoviajante.com/lista-de-ddds-do-brasil-completa/, Janeiro.