INF213 - Estruturas de Dados

Trabalho Final

Prof. Marcos Henrique Fonseca Ribeiro

Junho/Julho de 2015

1 Observações Gerais Sobre o Trabalho

1. Data do início da entrega do trabalho: **01/07/2015**, quarta-feira, às 08:00.
2. Data limite de entrega do trabalho: **05/07/2015**, domingo, às 23:59.
3. Formato de entrega: arquivo *.zip*, contendo os arquivos pedidos no fim do roteiro.
4. Meio de entrega: http://www.dpi.ufv.br/~inf213
5. Máximo de alunos por grupo: 2 (dois). O trabalho também pode ser feito individualmente. Outras quantidades de membros no grupo não serão permitidas.
6. Valor da atividade: 15 pontos
7. Não esquecer:
   - De identificar o(s) aluno(s) no início dos arquivos em comentários, com nome e número de matrícula
   - Se achar(em) pertinente, incluir no pacote zip arquivo de texto (txt, doc, odt ou pdf) com observações gerais

2 Especificações do Programa

2.1 Introdução

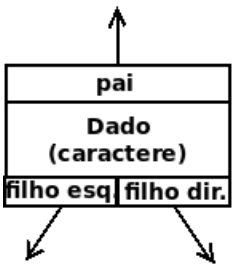
A exemplo do que vem sendo desenvolvido nas aulas práticas nas últimas semanas, este trabalho consiste em desenvolver um programa para gerenciamento de Árvores de Busca Binária, bem como seu subtipo Árvores AVL. No software, haverá um menu inicial onde o usuário escolherá uma entre várias possíveis operações, sendo encaminhado para a tela correspondente àquela funcionalidade e, após a execução da mesma, o sistema retornará para o menu inicial, repetindo este processo até que o usuário informe que deseja sair do programa. Algumas funcionalidades "periféricas", como a classe que faz a visualização gráfica de uma árvore e algumas funções auxiliares já se encontram implementadas e foram disponibilizadas para o grupo juntamente com este arquivo.

O desenvolvimento do programa principal, bem como das classes especificadas mais adiante, caberá ao grupo. Este roteiro também inclui exemplos de telas do sistema que devem ser desenvolvidas. O grupo é livre para escolher em qual plataforma (Sistema Operacional, Ferramenta de Programação, Compilador) irá desenvolver o trabalho. No entanto a função de montagem do menu, já fornecida, terá melhor visualização em um console Linux, maximizado, e com resolução de tela de 1366 x 768 ou superior. As seções a seguir trazem as especificações do trabalho.

2.2 Classe *No*

Esta classe, cujo nome deverá ser *No*, é a implementação de um nó de uma árvore de busca binária. Cada nó armazena um caractere como informação e possui ponteiros tanto para os filhos à esquerda e à direita quanto para o nó que o antecede na árvore, isto é, seu nó pai. Graficamente, temos o nó como ilustrado na Figura 1.

Figura 1: Estrutura de dado do nó

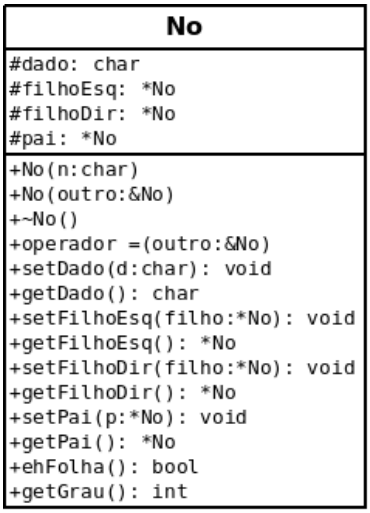


Divida a implementação da classe em dois arquivos, um arquivo de cabeçalho (*header file*) *.h*, contendo a declaração da classe e um arquivo *.cpp* contendo a implementação propriamente dita das funcionalidades da classe. A especificação UML da classe do nó pode ser encontrada na Figura 2. **Importante**: a especificação UML dada determina os requisitos **mínimos** que sua implementação deve ter. No entanto, o grupo está livre para expandir a classe, adicionando funções ou membros de dados auxiliares para as soluções dadas, da forma como desejar.

O grupo deve estar atento à visibilidade de cada membro da classe, seja função ou membro de dados. Basicamente, a classe deve conter:

- Dois construtores, um que recebe um caracter como parâmetro e um construtor de cópia, que recebe outro nó como parâmetro. Um ponto importante aqui é que ao se copiar um nó, a cópia não pode copiar os ponteiros do nó original, apenas o dado.
- Um destrutor, que aponta os ponteiros do objeto para o endereço nulo.
- Um operador de atribuição, que limpa os valores dos membros de dados da classe e copia o dado do objeto informado como parâmetro.
- *Getters* e *setters* para cada membro de dado.

Figura 2: Diagrama de Classes UML para a classe *No*



- Método *ehFolha*, que retorna verdadeiro, para o caso do nó ser uma folha da árvore que o contém, ou falso, caso contrário.
- Método *getGrau*, que informa o grau daquele nó. Estejam atentos à definição de grau, para evitar confusões com outros conceitos de árvores.

2.3 Classe *ArvoreBin*

Esta classe, cujo nome deverá ser *ArvoreBin*, é a implementação de uma árvore de busca binária (ABB) básica. A árvore será composta por múltiplos nós da classe *No*, conforme ilustrado na Figura 3. Na Figura, as setas pontilhadas representam os ponteiros para os nós pai que, conceitualmente e a rigor, não comporiam a estrutura de dados, mas que estão presentes na implementação para facilitar a programação de algumas funcionalidades que são mais facilmente solucionadas com a capacidade de se "navegar" na árvore no sentido folhas-raiz.

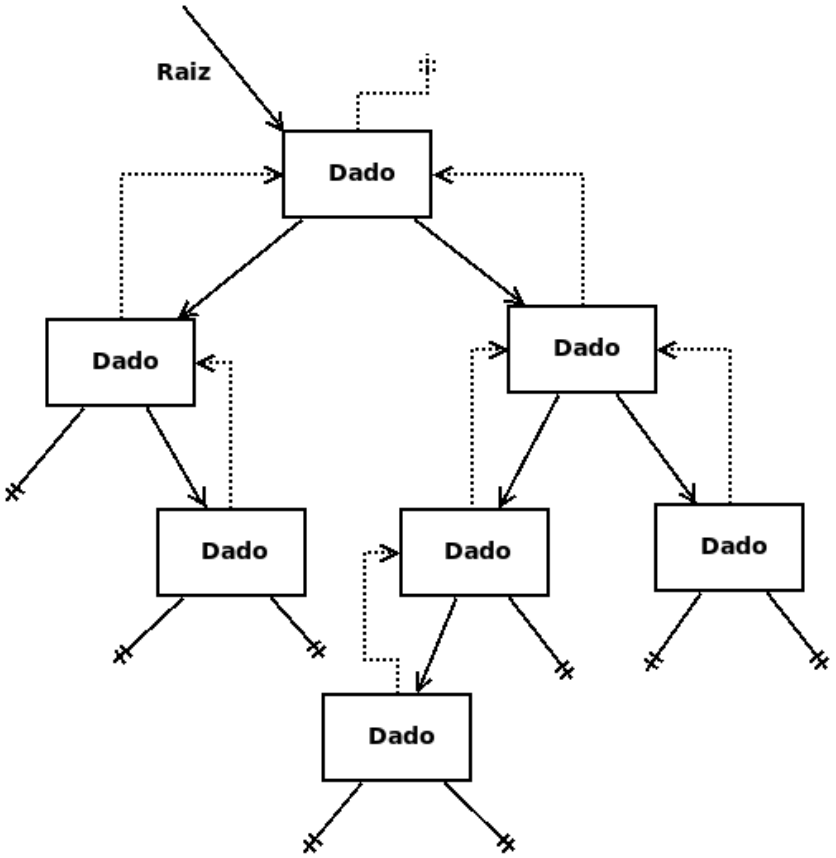
Assim como feito para a classe do nó, divida sua implementação da classe da ABB em dois arquivos, um arquivo de cabeçalho e um arquivo *.cpp*. A especificação UML da classe da ABB pode ser encontrada na Figura 4. **Importante**: a exemplo da classe anterior, a especificação UML dada determina os requisitos **mínimos** da implementação, podendo o grupo expandir a classe com membros auxiliares da forma que desejar.

Observação: algumas funcionalidades estão representadas no diagrama e na especificação apenas pela função principal, de visibilidade pública e, certamente, demandarão implementação de funções auxiliares. Cabe ao grupo identificar tais casos e adicionar as funções necessárias.

Mais uma vez, o grupo deve estar atento à visibilidade de cada membro da classe. Resumidamente, a classe deve conter:

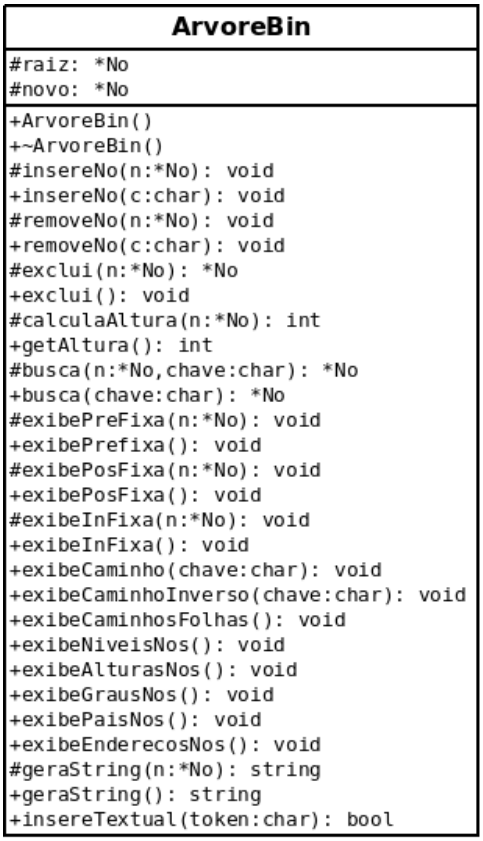
- Atributo *raiz*, com um ponteiro para o nó que está na raiz da árvore.

Figura 3: Estrutura de dado da árvore de busca binária



- Atributo *novo*, com um ponteiro para um objeto de nó, utilizado para auxiliar na criação e inserção de novos nós na árvore.
- Construtor padrão, que inicializa um objeto com os ponteiros apontando para o endereço nulo.
- Destrutor, que deve excluir todos os nós da árvore
- Funções *insereNo*, que juntas adicionam um nó à ABB, respeitando as regras de inserção deste tipo de árvore. A versão pública do método deve criar um novo nó contendo o caractere informado como parâmetro e evocar a versão protegida, passando para esta última a raiz da árvore.
- Funções *removeNo*, que juntas removem um nó da ABB, respeitando as regras de remoção de nós de uma árvore. A versão pública do método deve realizar uma busca na árvore, a partir da raiz (utilizar o método de busca) para encontrar o nó contendo a chave informada como parâmetro e então evocar a versão protegida do método informando este nó. Um algoritmo em alto nível para o mecanismo de remoção do nó em uma ABB pode

Figura 4: Diagrama de Classes UML para a classe *ArvoreBin*



ser encontrado na Seção 6.1. A versão protegida do método deve ser uma implementação deste algoritmo.

- Funções *exclui*, que juntas excluem todos os nós da ABB. Isto é, remove totalmente a árvore da memória, finalizando com o "aterramento" dos ponteiros da classe. A versão pública deve utilizar a versão protegida para, recursivamente, remover os nós no sentido folha-raiz.
- Função *calculaAltura*. Recebe como parâmetro um nó qualquer da árvore e calcula e retorna a altura do mesmo.
- Função *getAltura*. Calcula e retorna a altura da árvore.
- Funções *busca*, que juntas localizam o nó que contém determinado valor na árvore. Retornam o endereço do nó, caso a chave seja encontrada e o endereço nulo, caso contrário. A versão pública da função utiliza a versão protegida para, recursivamente, realizar a busca.
- Funções *exibePreFixa*, que juntas exibem os valores dos nós da árvore em notação pré-fixa, isto é, em pré-ordem. A versão pública da função utiliza a versão protegida para, recursivamente, realizar a exibição dos dados.

- Funções *exibePosFixa*, que juntas exibem os valores dos nós da árvore em notação pós-fixa, isto é, em pós-ordem. A versão pública da função utiliza a versão protegida para, recursivamente, realizar a exibição dos dados.

- Funções *exibeInFixa*, que juntas exibem os valores dos nós da árvore em notação infixa, isto é, em ordem central. A versão pública da função utiliza a versão protegida para, recursivamente, realizar a exibição dos dados.

- Função *exibeCaminho*. Exibe o caminho completo da raiz até um determinado nó da árvore, identificado através de uma chave de busca. Esta funcionalidade é a mesma implementada na aula prática 13, de 16/06/2015.

- Função *exibeCaminhoInverso*. Exibe o caminho completo de um determinado nó da árvore, identificado através de uma chave de busca, até a raiz. Esta funcionalidade é a mesma implementada na aula prática 13, de 16/06/2015.

- Função *exibeCaminhosFolhas*. Exibe todos os caminhos completo da raiz até cada uma das folhas da árvore. Esta funcionalidade é a mesma implementada na aula prática 13, de 16/06/2015.

- Função *exibeNiveisNos*. Exibe listagem, em caminhamento pré-ordem, do valor contido na chave do nó, seguido pelo nível que o mesmo se encontra na árvore. Exibir as informações de cada nó em uma linha da tela. Exemplos da tela com esta funcionalidade se encontram na Seção 7. **Adote o nível da raiz como sendo zero**.

- Função *exibeAlturasNos*. Exibe uma listagem semelhante à do caso anterior, porém ao invés do par dado / nível, exibir o par dado / altura do nó. Exemplos da tela com esta funcionalidade se encontram na Seção 7.

- Função *exibeGrausNos*. Exibe uma listagem semelhante à do caso anterior, porém ao invés do par dado / altura, exibir o par dado / grau do nó. Exemplos da tela com esta funcionalidade se encontram na Seção 7.

- Função *exibePaisNos*. Exibe uma listagem semelhante à do caso anterior, porém ao invés do par dado / grau, exibir o par dado / dado do pai do nó. Exemplos da tela com esta funcionalidade se encontram na Seção 7.

- Função *exibeEnderecosNos*. Exibe uma listagem semelhante à do caso anterior, porém ao invés do par dado / dado do pai, exibir o par dado / endereço do nó. Exemplos da tela com esta funcionalidade se encontram na Seção 7.

- Funções *geraString*. Juntas, são responsáveis por gerar e retornar uma string contendo a representação textual da árvore. Mais sobre a representação textual pode ser encontrada na Seção 2.3.1.

- Função *insereTextual*. Responsável por processar, caractere a caractere, uma string contendo a notação textual de uma árvore binária e reconstruí-la em memória. **Obs.:** a função deve, antes de processar a string, excluir da memória qualquer árvore já contida no objeto. Um algoritmo em alto nível para a construção da árvore a partir da notação textual pode ser encontrado na Seção 6.2.

2.3.1 Representação Textual de Uma Árvore Binária

Considere a sigla *SAE*, para representar a subárvore esquerda de um nó qualquer de uma árvore binária e a sigla *SAD*, para representar a subárvore direita deste mesmo nó. Considere ainda a sigla *INFO*, para representar a informação contida em um nó da árvore. A representação textual de um nó qualquer da árvore pode ser feita da seguinte forma:

```
<SAE INFO SAD>
```

. Onde as subárvore vão seguir a mesma notação aqui apresentada. Quando uma subárvore não existir, deve ser utilizada a notação:

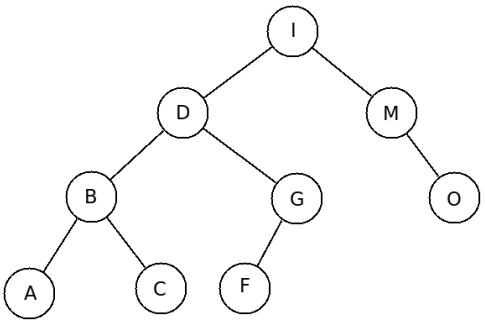
```
< >
```

para representar uma subárvore vazia.

Desta maneira, a árvore da Figura 5 pode ser representada pela seguinte notação textual a seguir.

```
<<<<< >A< >>B<< >C< >>>D<<< >F< >>G< >>>I<< >M<< >O< >>>>
```

Figura 5: Árvore Binária do Exemplo de Notação Textual



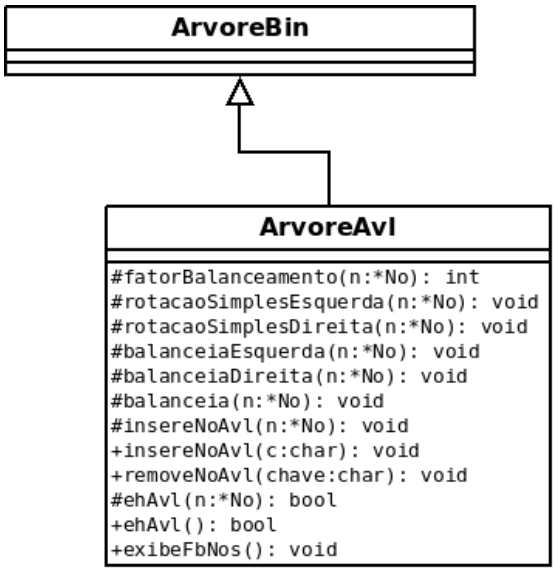
2.4 Classe *ArvoreAvl*

Esta classe, cujo nome deverá ser *ArvoreAvl*, é a implementação de uma árvore de busca binária básica auto-balanceável, também conhecida como Árvore AVL. Sendo este tipo de árvore um tipo especial de ABB, pode ser modelada como uma especialização da classe *ArvoreBin*, portanto, herdará desta última classe suas características, acrescentando as funcionalidades específicas para que a árvore agora se comporte como uma árvore AVL.

Assim como feito para as classes anteriores, divida sua implementação da classe da Árvore AVL em dois arquivos, um arquivo de cabeçalho e um arquivo *.cpp*. A especificação UML da classe da Árvore AVL pode ser encontrada na Figura 6. **Importante**: a exemplo das classes anteriores, a especificação UML dada determina os requisitos **mínimos** da implementação, podendo o grupo expandir a classe com membros auxiliares da forma que desejar.

Observação: novamente, algumas funcionalidades estão representadas no diagrama e na especificação apenas pela função principal, de visibilidade pública e, certamente, demandarão implementação de funções auxiliares. Cabe ao grupo identificar tais casos e adicionar as funções necessárias.

Figura 6: Diagrama de Classes UML para a classe *ArvoreAvl*



Mais uma vez, o grupo deve estar atento à visibilidade de cada membro da classe. Resumidamente, a classe deve conter:

- Função *fatorBalanceamento*. Recebe como parâmetro um nó qualquer da árvore e calcula e retorna o fator de balanceamento calculado para aquele nó.
- Funções *rotacaoSimplesEsquerda* e *rotacaoSimplesDireita*. Úteis para as operações de balanceamento da árvore. Implementam, respectivamente, as rotações simples à esquerda e à direita, conforme mostrado em sala de aula.
- Funções *balanceiaEsquerda* e *balanceiaDireita*. Úteis para as operações de balanceamento da árvore. Implementam, respectivamente, as tomadas de decisão sobre qual tipo de rotação executar naquele lado da árvore. Decidem se a rotação será simples ou dupla à esquerda (ou à direita), conforme mostrado em sala de aula.
- Função *balanceia*. Analisa o fator de balanceamento de um nó e decide se haverá balanceamento ou não e, se houver, se será à direita ou à esquerda, chamando as funções *balanceiaEsquerda* ou *balanceiaDireita*.
- Funções *insereNoAvl*. Juntas, são responsáveis pela inserção de um nó na árvore, mantendo a mesma balanceada, conforme mostrado em sala de

aula. A versão pública do método deve criar um novo nó contendo o caractere informado como parâmetro e evocar a versão protegida, passando para esta última a raiz da árvore. Já a versão protegida, deve inserir o nó na árvore e efetuar o rebalanceamento, se necessário for.

- Função *removNoAvl*. Junto com a versão protegida da função *removeNo*, herdada da classe primitiva, é responsável por efetuar a remoção do nó da árvore e e efetuar o rebalanceamento, se necessário for. Seu funcionamento básico consiste em evocar a função de busca e localizar o nó a ser removido na árvore, remover o mesmo (via função herdada) e rebalancear a árvore, quando necessário.

- Funções *ehAvl*. Juntas, verificam se a árvore atualmente em memória é uma árvore AVL ou não. Uma árvore é AVL se for binária **de busca** e estiver balanceada, isto é, o fator de balanceamento fb para todos os nós da estrutura possuir valor $-1 \leq fb \leq 1$. A implementação do grupo deve contemplar ambos testes. A versão pública da função utiliza a versão protegida para, recursivamente, realizar a verificação das condições nó a nó.

- Função *exibeFbNos*. Exibe uma listagem, em caminhamento pré-ordem, do valor contido na chave do nó, seguido pelo fator de balanceamento calculado para aquele nó. Exibir as informações de cada nó em uma linha da tela. Exemplos da tela com esta funcionalidade se encontram na Seção 7.

Devido ao fato da classe *ArvoreAvl* ser uma especialização da classe *ArvoreBin* e não fazer sobreposição de nenhuma função, todos os recursos da classe primitiva estão disponíveis na classe derivada. Desta forma, recomenda-se, para fins de simplificação da implementação, que a árvore com a qual irá se trabalhar no programa seja um objeto instanciado da classe *ArvoreAvl*.

2.5 Classe *Visualizador*

Esta classe, cuja implementação já se encontra completa e fornecida juntamente com este roteiro, é responsável por gerar uma representação gráfica de uma árvore da classe *ArvoreAvl* utilizando caracteres Ascii. Não é necessário modificá-la e está disponível para ser utilizada como ferramenta para ser incorporada ao programa desenvolvido pelo grupo.

Para utilizá-la, são necessárias apenas 3 funções:

- Função *Visualizador*. Construtor da classe. Recebe como parâmetro um **ponteiro** para uma árvore avl.

- Função *exibe*. Sem retorno e sem parâmetros, exibe em tela a representação gráfica da árvore que está em memória.

- Função *retornaString*. Sem parâmetros, retorna uma string contendo a representação gráfica da árvore que está em memória.

Nos exemplos de tela da Seção 7, note que será necessário evocar a chamada à função *exibe*, da classe, por diversas vezes ao longo do software.

2.6 Funções "avulsas"

Algumas funções "avulsas", independentes de qualquer classe ou objeto, são importantes para o programa. Parte delas já se encontra implementada no arquivo *funcoes.cpp* fornecido juntamente com este roteiro e parte estão apenas declaradas no arquivo *funcoes.h* e caberá ao grupo desenvolvê-las.

Um ponto comum a todas estas funções é que elas têm relação com o funcionamento geral do programa e não consistem em funcionalidades específicas das entidades modeladas nas classes. No entanto, não possuem ligação conceitual entre si que justifique o seu agrupamento em novas classes e, por este motivo, ficaram implementadas como funções *standalone*.

2.6.1 Funções avulsas já implementadas

Estas funções já se encontram completamente implementadas e estão, assim como a classe *Visualizador*, disponíveis para seu uso como ferramentas. É recomendado, no entanto, que se faça uma breve análise do código das mesmas, pois este pode funcionar como dicas ou exemplos de soluções similares às que devem ser desenvolvidas no trabalho. A seguir, temos uma breve descrição de cada função já pronta.

- Função *limpaTela*. Sem retorno e parâmetros, emula a "limpeza de tela" para melhor exibição das informações.
- Função *leChar*. Sem parâmetros. Realiza a interação com o usuário para a leitura de chaves para inserção, remoção ou busca na árvore que está em memória. Também trata erros, impedindo o uso como chave dos caracteres reservados para a sintaxe de representação textual da árvore: $<$, $>$ e o espaço.
- Função *converteNum*. Função auxiliar para a implementação do tratamento de erros do menu inicial do programa. Não será utilizada pelo grupo.
- Função *exibeMenu*. Sem parâmetros. Exibe em tela o menu principal do programa (tela inicial), realiza a leitura da opção escolhida pelo usuário e retorna a mesma (valor inteiro) para uso no programa principal ou por outra função.
- Função *leNomeArquivo*. Sem parâmetros. Realiza a interação com o usuário para a leitura da string contendo um nome de arquivo que será utilizado para leitura ou escrita de dados da árvore. Retorna esta string.
- Função *salvaGrafico*. Recebe como parâmetros uma string contendo um nome de arquivo e um ponteiro para a árvore que está em memória. Executa a função *retornaString*, da classe *Visualizador* e salva a representação gráfica da árvore em um arquivo de texto, que pode ser aberto em um editor de textos qualquer. Retorna um valor booleano, indicando o sucesso (verdadeiro) ou fracasso (falso) da operação.
- Função *leDadosSequencial*. Recebe como parâmetros uma string contendo um nome de arquivo, um ponteiro para a árvore que está em memória e um valor booleano, indicando se a inserção na árvore será utilizando a função

insereNo, de uma árvore ABB comum ou a função *insereNoAvl*, de uma árvore AVL. O arquivo informado deve ser um arquivo de texto contendo uma sequência de caracteres válidos (diferentes de <, > e espaço) para serem inseridos na árvore, de acordo com o tipo (ABB ou AVL) desejado. A sequência pode estar contida em uma única ou dividida em múltiplas linhas no arquivo. As quebras de linha são ignoradas pela função. Antes de realizar a carga dos dados, deve excluir a árvore que está em memória, se houver.

2.6.2 Funções avulsas não implementadas

Estas funções se encontram apenas declaradas no arquivo *funcoes.h* e devem ser implementadas no arquivo *funcoes.cpp*. Embora sejam desenvolvidas pelo grupo, seu ojbetivo é o mesmo daquelas descritas na Seção anterior: seu uso como ferramentas. A seguir, temos uma breve descrição de cada uma delas.

- Função *salva*. Recebe como parâmetros uma string contendo um nome de arquivo e um ponteiro para a árvore que está em memória. Executa a função *geraString*, disponível no objeto da árvore e salva a representação textual da mesma em um arquivo de texto, que pode ser aberto em um editor de textos qualquer e também pode ser utilizado para a funcionalidade de carregar uma árvore salva em arquivo para a memória. Retorna um valor booleano, indicando o sucesso (verdadeiro) ou fracasso (falso) da operação.

- Função *leDados*. Recebe como parâmetros uma string contendo um nome de arquivo e um ponteiro para a árvore que está em memória. O funcionamento geral é análogo ao da função *leDadosSequencial*, porém ao invés de realizar a inserção de nós típica, deve construir em memória exatamente a árvore que está representada no arquivo de texto, utilizando a notação textual. Para isto, deve utilizar a função *insereTextual*, do objeto da árvore. Antes de realizar a carga dos dados, deve excluir a árvore que está em memória, se houver.

- Função *executaOpcao*. Recebe como parâmetros um inteiro contendo a opção escolhida pelo usuário no menu principal, um ponteiro para a árvore que está em memória e uma variável booleana, passada por referência, utilizada como sentinela para controlar o tipo de árvore com a qual se está trabalhando no momento. Se o valor do parâmetro booleano for verdadeiro, indica que deve se seguir as funcionalidades de uma árvore AVL, se falso, deve se seguir as funcionalidades de uma ABB comum. A função deve analisar o primeiro parâmetro, da opção escolhida e realizar a execução da operação desejada. Não tem retorno.

A função *executaOpcao* deve ser responsável pela seleção da operação. Se a operação de fato vai ser implementada internamente nesta mesma função ou em outras, apenas chamadas por esta, é uma decisão de projeto que fica a critério do grupo. Uma das funcionalidades previstas consiste exatamente em alterar a sentinela do tipo de árvore e, por esta razão, o parâmetro referente à mesma é passado por referência, para que a mudança se reflita externamente, no programa principal.

Obs.: Por fim, vale lembrar que o grupo está livre para extender este arquivo criando novas funções bem como desejar. Só se pede que continue se mantendo o padrão utilizado no restante do programa de se colocar a declaração (protótipo) das funções criadas no arquivo *funcoes.h*.

2.7 Programa Principal

A implementação do arquivo *main.cpp*, contendo o programa principal, fica inteiramente a cargo do grupo, que deve se atentar às inclusões de arquivos de cabeçalho que devem ser feitas para que o mesmo funcione corretamente. A exigência é que este arquivo tenha este nome e seja o mais "enxuto" possível, isto é, contenha apenas a função *main*.

3 Documentação do Projeto

A documentação do projeto será bastante simples, devendo apenas algumas recomendações serem seguidas. Pede-se, portanto que se siga as especificações a seguir.

3.1 Comentários

Primeiramente, comente ao máximo o código produzido. Especialmente quando alguma solução não intuitiva ou de compreensão mais difícil for adotada. Além disso, pede-se que o seguinte padrão seja respeitado. Para cada arquivo de código, seja ele *.h* ou *.cpp*, inclua-se um comentário inicial, **no início do arquivo**, contendo os dados do grupo, conforme exemplificado abaixo.

```
/***************
** ARQUIVO: main.cpp
** GRUPO:
** Fulano da Silva Sauro - 88888
** Beltrana Souza Souza - 77654
***************/
```

Um padrão semelhante deve ser seguido para se comentar cada função desenvolvida, conforme exemplo abaixo.

```
/***************
** NOME:
**      busca
** DESCRICAO:
**      funcao recursiva que realiza a busca de uma chave em uma ABB.
** PARAMETROS:
**      n: ponteiro para o noh corrente da arvore
**      chave: representa o valor procurado na arvore
** RETORNO:
**      ponteiro para o noh que contem a chave
**      ou endereco nulo, caso nao encontrado
***************/
```

3.2 Documentação UML

Pede-se que se entregue um arquivo *.pdf*, contendo o Diagrama de Classes UML (apenas esse) da versão final das classes feitas pelo grupo, considerando os relacionamentos entre elas. Inclua todos os membros das classes, isto é, funções e atributos. Mesmo aqueles que já foram especificados aqui. Pode-se utilizar os diagramas deste roteiro como modelo a ser seguido.

4 Compilação

O programa resultante do trabalho estará distribuído em diversos arquivos com códigos-fonte. Minimamente, dois arquivos para a classe *No*, dois arquivos para a classe *ArvoreBin*, dois arquivo para a classe *ArvoreAvl*, um para a classe *Visualizador*, dois para as funções avulsas e um para o programa principal, em um total de 10 arquivos, desconsiderando-se eventuais arquivos adicionais que o grupo possa decidir criar. Desta forma, a compilação pode ficar desorganizada.

Este problema deve ser solucionado com a utilzação da ferramenta *make*, mostrada em sala de aula e utilizada por diversas vezes nas aulas práticas. Para isto, crie um arquivo chamado *makefile* com os alvos:

- **avl**. Alvo principal, que compila o programa executável com o nome *avl* e cujas dependências são o arquivo *Visualizador.h* e os demais alvos do arquivo, à exceção do alvo *clean*.
- **clean**. Alvo que realiza a limpeza de binários da pastas. Deve remover os arquivos *.o*, bem como o programa *avl* propriamente dito.
- **Alvos para os demais arquivos *.cpp***. Cada arquivo *.cpp* deve ter seu próprio alvo, cuja dependência é ele próprio. Estes alvos devem ser compilados com a opção *-c*.

5 Entrega

O arquivo compactado *.zip* a ser enviado deve conter:

- Todos os arquivos *.cpp* resultantes da versão final do programa desenvolvido pelo grupo.
- Todos os arquivos *.h* resultantes da versão final do programa desenvolvido pelo grupo.
- O arquivo *makefile*, para compilação.
- O arquivo *.pdf* com a especificação UML.
- Arquivo de texto (txt, doc, odt ou pdf) com observações ou explicações gerais, se o grupo achar pertinente.

6 Algoritmos

6.1 Algoritmo de Remoção de Nó em ABB

Algoritmo 1: Remoção de Nó em ABB

entrada: Nó n a ser removido

```
// A cada atualização de um nó filho, atualizar também as
   referências aos pais
```

1 **se** *n não é vazio* **então**
2 $g \leftarrow$ grau de n;
3 $d \leftarrow$ busca substituto de n (Algoritmo 2);
4 **se** *s foi encontrado* **então**
5 $p \leftarrow$ pai de s;
6 **se** *p foi encontrado* **então**
7 **se** $n = p$ *e* $g = 2$ **então**
8 faça filho à direita de s ser o atual filho à direita de n;
9 **senão se** $n \neq p$ **então**
`// O nó removido não é o pai do nó substituto`
10 **se** *s é folha* **então**
11 faça filho à direita de p ser vazio;
12 **senão**
13 faça filho à direita de p ser o atual filho à esquerda de s;
14 faça os filho à esquerda e à direita de s serem, respectivamente, os atuais filho à esquerda e à direita de n;
15 **se** *n é a raiz* **então**
16 faça a nova raiz ser s;
17 remova n;

Algoritmo 2: Busca por substituto

entrada: Nó n
saída : Nó da árvore

1 **se** *se subárvore esquerda de n não é vazia* **então**
2 **retorna** *nó mais à direita da subárvore esquerda*
3 **senão**
4 **retorna** *filho à direita de n*

6.2 Algoritmo de Construção de Árvore Binária a partir de Notação Textual

Algoritmo 3: Construção de Árvore Binária a partir de Notação Textual

entrada: *sequencia*: string
entrada: *arvore*: ArvoreAvl

```
1  ignorarProximo ← falso;
2  cursor ← raiz de arvore;
3  para cada caractere c em sequencia faça
4      se ignorarProximo = falso então
5          ignorarProximo ← insereTextual(c, cursor);
6      senão
7          ignorarProximo ← falso;
```

Algoritmo 4: Algoritmo da função *insereTextual*

entrada: *token*: caractere
entrada: *cursor*: ponteiro para nó
saída : Booleano, informando se próximo caractere da sequência deve ou não ser ignorado

```
   // A cada inserção de um nó filho, atualizar também a
      referência ao pai
1  se token ='>' então
2      move cursor para o nó pai;
3  senão se token ='<' então
4      cria novo nó novo;
5      se raiz ainda não definida então
6          raiz ← novo;
7      senão
8          faça novo ser filho à esquerda de cursor;
9      cursor ← novo;
10 senão se token =' ' então
11     temp ← pai de cursor;
12     faça ponteiro de filho de temp que apontava para o nó de cursor ser
       nulo;
13     remova da memória nó apontado por cursor;
14     cursor ← temp;
15     retorna verdadeiro
16 senão
17     armazene token na área de dados do nó apontado por cursor;
18     cria novo nó novo;
19     faça novo ser filho à direita de cursor;
20     cursor ← novo;
21     retorna verdadeiro
22 retorna falso
```

7 Execução

Esta Seção traz exemplos de telas da execução de cada operação a ser feita pelo programa, juntamente com algum exemplo de caso particular ou explicação adicional, quando necessário for. As telas são apresentadas na seguinte ordem. Começa-se pela tela do menu principal e, em seguida, apresenta-se as telas das operações na ordem em que estão no menu referido. Obs.: vale lembrar que ao término de cada operação, o programa sempre retorna ao menu principal, para que o usuário escolha a próxima operação a ser feita. Vale também destacar que o primeiro passo de cada tela é efetuar a limpeza da mesma. Na função *exibeMenu*, este passo já se encontra implementado. Antes de apresentar os exemplos de tela, porém, vamos apresentar exemplos de conteúdos de arquivos de texto que podem ser utilizados para os testes.

A sequência de caracteres a seguir gera a árvore utilizada na maioria dos exemplos aqui mostrados.

```
pfbsytwzhgr
```

A representação textual da ABB convencional gerada pela inserção dos dados da sequência anterior é mostrada abaixo.

```
<<<< >b< >>f<<< >g< >>h< >>>p<<< >r< >>s<<< >t<< >w< >>>y<< >z< >>>>>
```

Já a representação textual da AVL gerada pela inserção dos mesmos dados da referida sequência é mostrada a seguir.

```
<<<<< >b< >>f<< >g< >>>h<< >p<< >r< >>>>s<<< >t< >>w<< >y<< >z< >>>>>
```

7.1 Menu Principal (Já Implementado)

```
================================= ARVORES BINARIAS ==========================
MENU DE OPCOES:

--- TIPO DE ARVORE ---
 1) MUDAR O TIPO DE ARVORE
 2) EXIBIR TIPO DE ARVORE

--- ARQUIVOS ---
 3) CARREGAR DADOS DE ARQUIVO COM SEQUENCIA DE DADOS PARA ARVORE
 4) CARREGAR DADOS DE ARQUIVO COM NOTACAO TEXTUAL PARA ARVORE
 5) SALVAR ARVORE EM ARQUIVO
 6) SALVAR REPRESENTACAO GRAFICA DE ARVORE EM ARQUIVO

--- EDITAR ARVORE ---
 7) INSERIR MANUALMENTE UM VALOR NA ARVORE
 8) REMOVER UM VALOR DA ARVORE
 9) ESVAZIAR ARVORE

--- VISUALIZACAO DE DADOS DE NOS ---
10) EXIBIR NIVEIS DOS NOS DA ARVORE
11) EXIBIR ALTURAS DOS NOS DA ARVORE
12) EXIBIR GRAUS DOS NOS DA ARVORE
13) EXIBIR PAIS DOS NOS DA ARVORE
14) EXIBIR ENDERECOS DOS NOS DA ARVORE
15) EXIBIR FATORES DE BALANCEAMENTOS DOS NOS DA ARVORE

--- VISUALIZACAO DA ARVORE ---
16) VISUALIZAR ARVORE
17) EXIBIR ARVORE EM NOTACAO PREFIXA
18) EXIBIR ARVORE EM NOTACAO POSFIXA
19) EXIBIR ARVORE EM NOTACAO INFIXA
20) EXIBIR ALTURA DA ARVORE
```

```
--- CAMINHAMENTO ---
21) EXIBIR CAMINHO ATE NO DA ARVORE
22) EXIBIR CAMINHO DO NO DA ARVORE ATE RAIZ
23) EXIBIR CAMINHOS ATE FOLHAS DA ARVORE

-------
24) SAIR
```

7.2 Opção 1: Mudar o Tipo de Árvore

Exemplo 1

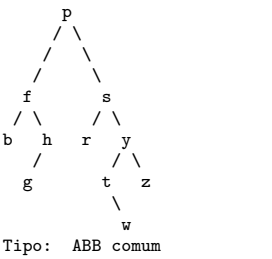
```
Tipo alterado para arvore AVL

Digite ENTER para continuar
```

Exemplo 2

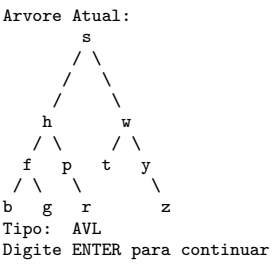
```
Tipo alterado para ABB comum

Digite ENTER para continuar
```

7.3 Opção 2: Exibir o Tipo de Árvore

Exemplo 1



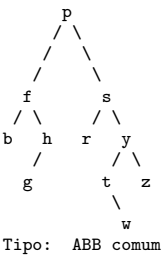
Exemplo 2



7.4 Opção 3: Carregar Dados de Arquivo Com Sequência de Dados

Exemplo 1

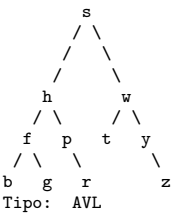
```
Informe o nome do arquivo: entrada.txt

Dados carregados com sucesso:

Arvore Atual:
```



```
Tipo:  ABB comum

Digite ENTER para continuar
```

Exemplo 2

```
Informe o nome do arquivo: entrada.txt

Dados carregados com sucesso:

Arvore Atual:
```



```
Tipo:  AVL

Digite ENTER para continuar
```

Exemplo 3

```
Informe o nome do arquivo: lixo.txt

Um erro ocorreu ao carregar os dados

Digite ENTER para continuar
```

7.5 Opção 4: Carregar Dados de Arquivo Com Notação Textual

Os exemplos são análogos aos da opção anterior.

7.6 Opção 5: Salvar Árvore em Arquivo

```
Informe o nome do arquivo: out.txt

Representacao gerada:
```

```
<<<< >b< >>f<<< >g< >>h< >>>p<<< >r< >>s<<< >t<< >w< >>>y<< >z< >>>>>

Arquivo salvo com sucesso

Digite ENTER para continuar
```

7.7 Opção 6: Salvar Representação Gráfica da Árvore em Arquivo

Exemplo de Tela

```
Informe o nome do arquivo: grafica.txt

Arquivo salvo com sucesso

Digite ENTER para continuar
```

Exemplo do Conteúdo do Arquivo Gerado

```
Arvore Atual:
      p
     / \
    /   \
   /     \
  f       s
 / \     / \
b   h   r   y
   /       / \
  g       t   z
           \
            w
Tipo:  ABB comum
```

7.8 Opção 7: Inserir Manualmente um Valor na Árvore

Exemplo 1

```
Arvore Atual:
      p
     / \
    /   \
   /     \
  f       s
 / \     / \
b   h   r   y
   /       / \
  g       t   z
           \
            w
Tipo:  ABB comum

Informe um caractere: m

Valor m inserido na arvore.


Arvore Atual:
      p
     / \
    /   \
   /     \
  f       s
 / \     / \
b   h   r   y
   / \     / \
  g   m   t   z
```

```
          \
           w
Tipo:  ABB comum
Digite ENTER para continuar
```

Exemplo 2

```
Arvore Atual:
      p
     / \
    /   \
   /     \
  f       s
 / \     / \
b   h   r   y
   / \     / \
  g   m   t   z
           \
            w
Tipo:  ABB comum

Informe um caractere: p

Valor p ja se encontra na arvore.

Digite ENTER para continuar
```

7.9 Opção 8: Remover um Valor na Árvore

Exemplo 1

```
Arvore Atual:
      p
     / \
    /   \
   /     \
  f       s
 / \     / \
b   h   r   y
   / \     / \
  g   m   t   z
           \
            w
Tipo:  ABB comum

Informe um caractere: h

Valor h removido da arvore.


Arvore Atual:
      p
     / \
    /   \
   /     \
  f       s
 / \     / \
b   g   r   y
     \     / \
      m   t   z
           \
            w
Tipo:  ABB comum
Digite ENTER para continuar
```

Exemplo 2

```
Arvore Atual:
      p
     / \
    /   \
   /     \
  f       s
 / \     / \
b   g   r   y
     \     / \
      m   t   z
           \
            w
Tipo:  ABB comum

Informe um caractere: n

O valor n nao se encontra na arvore.

Digite ENTER para continuar
```

7.10 Opção 9: Esvaziar Árvore

Exemplo

```
Arvore esvaziada.


Arvore Atual:
Arvore vazia.
Tipo:  AVL
Digite ENTER para continuar
```

7.11 Opção 10: Exibir o Nível de Cada Nó da Árvore

Exemplo

```
Arvore Atual:
        s
       / \
      /   \
     /     \
    h       w
   / \     / \
  f   p   t   y
 / \   \       \
b   g   r       z
Tipo:  AVL
Exibindo os niveis de cada no da arvore:
s: 0
h: 1
f: 2
b: 3
g: 3
p: 2
r: 3
w: 1
t: 2
y: 2
z: 3
Digite ENTER para continuar
```

7.12 Opção 11: Exibir a Altura de Cada Nó da Árvore

Exemplo

```
Arvore Atual:
        s
       / \
      /   \
     /     \
    h       w
   / \     / \
  f   p   t   y
 / \   \       \
b   g   r       z
Tipo:  AVL
Exibindo as alturas de cada no da arvore:
s: 4
h: 3
f: 2
b: 1
g: 1
p: 2
r: 1
w: 3
t: 1
y: 2
z: 1
Digite ENTER para continuar
```

7.13 Opção 12: Exibir o Grau de Cada Nó da Árvore

Exemplo

```
Arvore Atual:
        s
       / \
      /   \
     /     \
    h       w
   / \     / \
  f   p   t   y
 / \   \       \
b   g   r       z
Tipo:  AVL
Exibindo os graus de cada no da arvore:
s: 2
h: 2
f: 2
b: 0
g: 0
p: 1
r: 0
w: 2
t: 0
y: 1
z: 0
Digite ENTER para continuar
```

7.14 Opção 13: Exibir o Pai de Cada Nó da Árvore

Exemplo

```
Arvore Atual:
        s
       / \
      /   \
     /     \
    h       w
   / \     / \
  f   p   t   y
 / \   \       \
b   g   r       z
```

```
Tipo:  AVL
Exibindo os pais de cada no da arvore:
s: NULL
h: s
f: h
b: f
g: f
p: h
r: p
w: s
t: w
y: w
z: y
Digite ENTER para continuar
```

7.15 Opção 14: Exibir o Endereço de Cada Nó da Árvore

Exemplo 1

```
Arvore Atual:
        s
       / \
      /   \
     /     \
    h       w
   / \     / \
  f   p   t   y
 / \   \       \
b   g   r       z
Tipo:  AVL
Exibindo os enderecos de cada no da arvore:
s: 0x21f0210
h: 0x21f0810
f: 0x21f2b80
b: 0x21f00f0
g: 0x21f0860
p: 0x21f01c0
r: 0x21f08b0
w: 0x21f0010
t: 0x21f02d0
y: 0x21f0260
z: 0x21f0060
Digite ENTER para continuar
```

7.16 Opção 15: Exibir o Fator de Balanceamento de Cada Nó da Árvore

Exemplo 1

```
Arvore Atual:
      p
     / \
    /   \
   /     \
  f       s
 / \     / \
b   h   r   y
   /       / \
  g       t   z
           \
            w
Tipo:  ABB comum
Exibindo os fatores de balanceamento de cada no da arvore:
p: -1
f: -1
b: 0
h: 1
g: 0
```

```
s: -2
r: 0
y: 1
t: -1
w: 0
z: 0
Digite ENTER para continuar
```

Exemplo 2

```
Arvore Atual:
Arvore vazia.
Tipo:  AVL
Exibindo os fatores de balanceamento de cada no da arvore:
Digite ENTER para continuar
```

Observação: este segundo exemplo, da operação de listagem de dados de nós para uma árvore vazia, só foi ilustrado na operação 15, porém o análogo dele deve ser implementado para todas as operações do grupo "VISUALIZACAO DE DADOS DE NOS", isto é, as operações que vão de 10 a 15.

7.17 Opção 16: Visualização da Árvore

Exemplo 1

```
Arvore Atual:
      p
     / \
    /   \
   /     \
  f       s
 / \     / \
b   h   r   y
   /       / \
  g       t   z
           \
            w
Tipo:  ABB comum
Digite ENTER para continuar
```

Exemplo 2

```
Arvore Atual:
Arvore vazia.
Tipo:  AVL
Digite ENTER para continuar
```

7.18 Opção 17: Visualização da Árvore na Notação Prefixa

Exemplo

```
Arvore Atual:
      p
     / \
    /   \
   /     \
  f       s
 / \     / \
b   h   r   y
   /       / \
  g       t   z
```

```
          \
           w
Tipo:  ABB comum

Arvore na notacao prefixa:
p f b h g s r y t w z
Digite ENTER para continuar
```

7.19 Opção 18: Visualização da Árvore na Notação Pós-fixa

Exemplo

```
Arvore Atual:
      p
     / \
    /   \
   /     \
  f       s
 / \     / \
b   h   r   y
   /       / \
  g       t   z
           \
            w
Tipo:  ABB comum

Arvore na notacao posfixa:
b g h f r w t z y s p
Digite ENTER para continuar
```

7.20 Opção 19: Visualização da Árvore na Notação Infixa

Exemplo

```
Arvore Atual:
      p
     / \
    /   \
   /     \
  f       s
 / \     / \
b   h   r   y
   /       / \
  g       t   z
           \
            w
Tipo:  ABB comum

Arvore na notacao infixa:
b f g h p r s t w y z
Digite ENTER para continuar
```

Exemplo 2

```
Arvore Atual:
Arvore vazia.
Tipo:  AVL

Arvore na notacao infixa:

Digite ENTER para continuar
```

Observação: este segundo exemplo, da operação de exibição dos dados para uma árvore vazia, só foi ilustrado na operação 19, do caminhamento central, porém o análogo dele deve ser implementado também para as operações que exibem os dados com os caminhamentos em pré e pós ordem, isto é, as operações 17 e 18.

7.21 Opção 20: Visualização da Altura da Árvore

Exemplo 1

```
Arvore Atual:
      p
     / \
    /   \
   /     \
  f       s
 / \     / \
b   h   r   y
   /       / \
  g       t   z
           \
            w
Tipo:  ABB comum

Altura da arvore: 5

Digite ENTER para continuar
```

Exemplo 2

```
Arvore Atual:
Arvore vazia.
Tipo:  AVL

Altura da arvore: 0

Digite ENTER para continuar
```

7.22 Opção 21: Exibir Caminho da Raiz até Nó da Árvore

Exemplo 1

```
Arvore Atual:
      p
     / \
    /   \
   /     \
  f       s
 / \     / \
b   h   r   y
   /       / \
  g       t   z
           \
            w
Tipo:  ABB comum

Informe um caractere: t
Caminho ate a chave: p s y t

Digite ENTER para continuar
```

Exemplo 2

```
Arvore Atual:
      p
     / \
    /   \
   /     \
  f       s
 / \     / \
b   h   r   y
   /       / \
  g       t   z
           \
            w
Tipo:  ABB comum

Informe um caractere: m
Caminho ate a chave: chave nao encontrada.

Digite ENTER para continuar
```

7.23 Opção 22: Exibir Caminho do Nó da Árvore até Raiz

Exemplo 1

```
Arvore Atual:
      p
     / \
    /   \
   /     \
  f       s
 / \     / \
b   h   r   y
   /       / \
  g       t   z
           \
            w
Tipo:  ABB comum

Informe um caractere: h
Caminho da chave ate raiz: h f p

Digite ENTER para continuar
```

Exemplo 2

```
Arvore Atual:
      p
     / \
    /   \
   /     \
  f       s
 / \     / \
b   h   r   y
   /       / \
  g       t   z
           \
            w
Tipo:  ABB comum

Informe um caractere: q
Caminho da chave ate raiz: chave nao encontrada.

Digite ENTER para continuar
```

7.24 Opção 23: Exibir Caminhos da Raiz até cada Folha da Árvore

Exemplo 1

```
Arvore Atual:
      p
     / \
    /   \
   /     \
  f       s
 / \     / \
b   h   r   y
   /       / \
  g       t   z
           \
            w
Tipo:  ABB comum
p f b
p f h g
p s r
p s y t w
p s y z

Digite ENTER para continuar
```

Exemplo 2

```
Arvore Atual:
a
 \
  b
   \
    c
     \
      d
       \
        e
Tipo:  ABB comum
a b c d e

Digite ENTER para continuar
```

Exemplo 3

```
Arvore Atual:
Arvore vazia.
Tipo:  AVL

Digite ENTER para continuar
```

7.25 Opção 24: Sair do Programa

Exemplo

```
Programa encerrado
```

Bons estudos.
Prof. Marcos