**Relatório Estrutura de Dados**

**Aluno:** Thales Yahya

1. **Árvore binária**
   1. *Classe Árvore Binária*

Texto

Descrição gerada automaticamente

* 1. *Implementação dos Métodos iniciais*

Declara uma variável de instância chamada raiz, que representa o nó raiz da árvore.



Em seguida cria um construtor da classe ArvoreBinaria. O construtor é chamado quando um novo objeto ArvoreBinaria é criado.

Texto

Descrição gerada automaticamente

* 1. *Implementação do Método Inserir*

*Texto

Descrição gerada automaticamente*

* + 1. Explicação do Código

Texto

Descrição gerada automaticamente

Um novo nó no é criado com o valor dado que deseja ser inserido na árvore.

Texto

Descrição gerada automaticamente

Aqui, o código verifica se a árvore está vazia (a raiz é null). Se a árvore estiver vazia, o novo nó “no” se torna a raiz da árvore.

Texto

Descrição gerada automaticamente

Nessa parte do código caso o dado que desejamos inserir for menor que o valor do nó atual (atual.getValor()), então verificamos se o nó à esquerda (atual.getEsquerdo()) do nó atual é nulo. Se for nulo, inserimos o novo nó à esquerda do nó atual e saímos do loop.

Caso contrário, se o dado for maior ou igual ao valor do nó atual, verificamos se o nó à direita (atual.getDireito()) do nó atual é nulo. Se for nulo, inserimos o novo nó à direita do nó atual e saímos do loop.

Se ambos os lados não forem nulos, continuamos navegando para a esquerda ou direita na árvore, dependendo do valor do dado em relação ao valor do nó atual. Isso se repete até encontrarmos um local vazio para inserir o novo nó.

* 1. *Métodos para imprimir a Árvore*

Texto

Descrição gerada automaticamente

Esta função é usada para imprimir os nós da árvore em três diferentes formas: pré-ordem, in-ordem e pós-ordem. Ela chama as funções apropriadas e imprime os valores dos nós na sequência desejada.

Texto

Descrição gerada automaticamente

Esta função imprime os nós da árvore em pré-ordem, o que significa que ela primeira visita o nó atual, em seguida, o nó à esquerda e, por fim, o nó à direita. Isso é feito de forma recursiva, e a função verifica se o nó não é nulo antes de imprimir seu valor.

Texto

Descrição gerada automaticamente

Essa função imprime os nós da árvore em in-ordem, o que significa que ela visita o nó à esquerda, depois o nó atual e, por fim, o nó à direita. Como as outras funções, ela também é feita de forma recursiva e verifica se o nó não é nulo antes de imprimir o valor.

Texto

Descrição gerada automaticamente

Essa função imprime os nós da árvore em pós-ordem, o que significa que ela primeira visita o nó à esquerda, depois o nó à direita e, por fim, o nó atual. Também é feita de forma recursiva que verifica se o nó não é nulo antes de imprimir o valor.

* 1. *Métodos para verificar se é folha*

Texto

Descrição gerada automaticamente

Essa função Folha(NodeArvore no) verifica se o nó no é uma folha na árvore. Uma folha é um nó que não tem filhos, ou seja, nem à esquerda (no.getEsquerdo()) nem à direita (no.getDireito()) há outros nós

* 1. *Método para Buscar na Árvore Binária*

Texto

Descrição gerada automaticamente

Inicializa um nó chamado noAtual com a raiz da árvore para iniciar a busca.

Entra em um loop que continua enquanto noAtual não for nulo. Isso significa que ele irá percorrer a árvore até encontrar o valor desejado ou até não haver mais nós para verificar.

Dentro do loop:

Verifica se o valor do noAtual é igual ao valor procurado (valor).

Se forem iguais, imprime uma mensagem indicando que o valor foi encontrado na árvore e retorna true, indicando que o valor foi encontrado com sucesso.

Caso o valor seja menor do que o valor do noAtual:

Atualiza o noAtual para ser o nó à esquerda, pois valores menores são encontrados à esquerda em uma árvore binária de busca.

Caso o valor seja maior do que o valor do noAtual:

Atualiza o noAtual para ser o nó à direita, pois valores maiores são encontrados à direita em uma árvore binária de busca.

Após o loop, se o valor não tiver sido encontrado, o método imprime uma mensagem indicando que o valor não foi encontrado na árvore e retorna false, indicando que o valor não foi encontrado com sucesso.

* 1. *Métodos para Remoção da Árvore Binária*

Texto

Descrição gerada automaticamente

Ela recebe dois argumentos: noAtual, que é o nó a ser removido, e pai, que é o nó pai do noAtual.

A função começa verificando se o noAtual é uma folha, o que significa que ele não tem filhos (nem à esquerda, nem à direita). Isso é verificado usando a função Folha(noAtual). Se for uma folha, o código entra no bloco condicional interno.

Dentro do bloco condicional, o código verifica se o noAtual é a raiz da árvore (raiz == noAtual). Se for a raiz, define a raiz da árvore como null, indicando que a árvore ficará vazia após a remoção. Caso contrário, verifica se o noAtual é o filho esquerdo do pai ou o filho direito e, em seguida, define o filho correspondente do pai como null, removendo assim o noAtual.

Se o noAtual não for uma folha (ou seja, ele tem um filho à esquerda), a função chama a si mesma recursivamente com o filho à esquerda do noAtual e atualiza o paicomo onoAtual`. Isso continua até que um nó folha seja encontrado, que é tratado no primeiro caso descrito.

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Verifica se a árvore está vazia (raiz == null). Se a árvore estiver vazia, não há nada a ser feito, e a função retorna.

Caso não esteja inicializa o noAtual como a raiz da árvore e cria variáveis pai e ehFilhoEsquerdo. pai será usado para rastrear o nó pai do noAtual, e ehFilhoEsquerdo indicará se o noAtual é um filho esquerdo ou direito do pai.

Inicia um loop que continua enquanto noAtual não for nulo e o valor do noAtual não for igual ao valor que deseja ser removido (dado). O loop percorre a árvore até encontrar o nó com o valor a ser removido.

Se o loop terminar e noAtual for nulo, significa que o valor a ser removido não está presente na árvore, e a função retorna sem fazer nada.

Se o noAtual a ser removido é uma folha (não tem filhos), ele é removido da árvore. O tratamento varia se o noAtual é a raiz, um filho esquerdo ou um filho direito do pai.

Se o noAtual a ser removido tiver um filho à direita, o nó é removido e o seu filho à direita substitui o noAtual. Mais uma vez, o tratamento varia se o noAtual é a raiz, um filho esquerdo ou um filho direito do pai.

Se o noAtual a ser removido tiver um filho à esquerda, o nó é removido e o seu filho à esquerda substitui o noAtual. O tratamento varia da mesma maneira que nos casos anteriores.

Se o noAtual a ser removido tiver dois filhos, é necessário encontrar o sucessor mais à esquerda do nó direito (o nó com o menor valor na subárvore direita). Esse sucessor é então copiado para o noAtual, e a função removerSucessor é chamada para remover o sucessor da subárvore direita.

1. **Árvore AVL**

**Leve em conta que alguns métodos da árvore Binária e Árvore AVL são os mesmos, porém alguns possuem pequenas modificações, os idênticos não irei citar aqui.**

Texto

Descrição gerada automaticamente

O código de inserir possuí a mesma lógica, porém ao finalizar uma inserção ele chama o método verificarBalanceamento();

Texto

Descrição gerada automaticamente

A função recebe um nó “no” como entrada e tem o objetivo de encontrar o pai desse nó na árvore. O pai de um nó é o nó que possui um filho igual a “no”.

A função começa verificando dois casos especiais:

Se o nó “no” for nulo ou se o nó “no” for a raiz da árvore (raiz), a função retorna null. Isso ocorre porque a raiz não tem pai ou, em geral, quando um nó é nulo, ele não tem pai.

Em seguida, a função cria duas variáveis percorre e pai. percorre é inicializada com a raiz da árvore, e pai é inicializada como null. Essas variáveis serão usadas para percorrer a árvore e rastrear o pai do nó alvo.

Entra em um loop que continua enquanto percorre não for nulo. O objetivo deste loop é percorrer a árvore até encontrar o pai do nó alvo.

Dentro do loop, a função verifica se o nó “no” é igual ao filho esquerdo (percorre.getEsquerdo()) ou ao filho direito (percorre.getDireito()) do nó atual (percorre). Se for igual a um dos filhos, significa que o nó no é o filho desse nó percorre, e, portanto, o nó percorre é o pai do nó alvo no. A função atualiza a variável pai com o valor de percorre e sai do loop usando break.

Se o nó “no” não for igual a nenhum dos filhos do nó atual, a função verifica se o valor do nó alvo “no” é maior ou igual ao valor do nó atual percorre. Se for maior ou igual, isso significa que o nó alvo está localizado à direita do nó atual, e a função atualiza percorre para ser o filho direito (percorre = percorre.getDireito()).

Caso contrário, o nó alvo está localizado à esquerda do nó atual, e a função atualiza percorre para ser o filho esquerdo (percorre = percorre.getEsquerdo()).

O loop continua a percorrer a árvore até encontrar o pai do nó alvo. Quando o pai é encontrado, a função sai do loop e retorna o nó pai.

Texto

Descrição gerada automaticamente

Esta função calcula a altura de um nó na árvore AVL. A altura de um nó é a quantidade de arestas no caminho mais longo entre esse nó e uma folha. Se o nó for nulo (ou seja, não existir na árvore), a função retorna -1 como indicativo de que esse nó não tem altura.

A função inicia com uma verificação: se o nó no for nulo, ela retorna -1 imediatamente.

Se o nó não for nulo, a função prossegue para calcular a altura. Ela faz isso de forma recursiva, ou seja, chamando a si mesma para calcular a altura dos filhos do nó no.

A função calcula a altura do nó à esquerda (no.getEsquerdo()) e a altura do nó à direita (no.getDireito()) chamando a função altura para esses nós filhos.

Ela compara as alturas da subárvore à esquerda e da subárvore à direita com a finalidade de determinar qual delas é maior.

A funçao retorna 1 mais a altura da subárvore mais alta entre a esquerda e a direita. Isso ocorre porque ao subir um nível na árvore, adiciona-se 1 à altura atual.

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Essa função ela recebe um nó no como entrada, que representa a raiz da subárvore que precisa ser rotacionada à esquerda.

A função começa criando uma referência chamada novaRaiz e a inicializa com o nó à direita do nó “no”. Isso efetivamente faz da novaRaiz a nova raiz da subárvore.

Em seguida, a função atualiza os ponteiros do nó “no”. O nó “no” deve ter seu filho à direita atualizado para ser o filho à esquerda da novaRaiz, para garantir que a estrutura da árvore seja mantida.

A novaRaiz deve ter seu filho à esquerda atualizado para ser o nó “no”, para garantir que o “no” se torne o filho à esquerda da novaRaiz.

Por fim, a função retorna a novaRaiz, que agora é a nova raiz da subárvore após a rotação.

Texto

Descrição gerada automaticamente

Ela recebe um nó “no” como entrada, que representa a raiz da subárvore que precisa ser rotacionada.

A função começa criando três referências: novaRaiz, pai, e vo. novaRaiz será a nova raiz da subárvore após a rotação dupla, pai será o pai do “no”, e vo será o próprio nó “no”.

A novaRaiz é definida como o filho esquerdo do filho direito do “no”, ou seja, o neto à esquerda do “no”.

O pai é definido como o filho direito do “no”.

Em seguida, a função realiza a rotação à direita “no” nó pai (que é um nó filho direito). Para isso, o filho direito do pai é atualizado para ser o filho direito do novaRaiz

A função realiza a rotação à esquerda no nó vo. Para isso, o filho direito do vo é atualizado para ser o filho esquerdo do novaRaiz.

Finalmente, a novaRaiz é definida como a nova raiz da subárvore após a rotação, com o vo como filho esquerdo e o pai como filho direito.

A função retorna a novaRaiz, que agora é a nova raiz da subárvore após a rotação dupla à esquerda.

Texto

Descrição gerada automaticamente  
  
Essas duas funções possuem a mesma lógica porem são invertidas as rotações.

Texto

Descrição gerada automaticamente

A função recebe um nó no como entrada, que representa o nó cujo balanceamento deve ser verificado.

Ela começa com uma verificação: se o nó “no” for nulo, a função retorna imediatamente. Isso ocorre porque não há nada a ser verificado ou corrigido em um nó nulo.

A função calcula o fator de balanceamento do nó “no”. O fator de balanceamento é a diferença entre a altura da subárvore à esquerda e a altura da subárvore à direita. Isso é feito calculando a altura do filho esquerdo (altura(no.getEsquerdo())) e subtraindo a altura do filho direito (altura(no.getDireito())).

A função verifica se o balanceamento está fora do intervalo permitido de [-1, 1]. Isso significa que a árvore está desbalanceada, e a função precisa realizar uma ou duas rotações para corrigir o balanceamento.

Se o balanceamento for maior que 1, isso indica que a subárvore à esquerda é mais pesada. A função verifica a altura do filho à esquerda do filho à esquerda (altura(no.getEsquerdo().getEsquerdo())) em comparação com a altura do filho à esquerda do filho à direita. Com base nessa comparação, decide se deve realizar uma rotação simples à direita ou uma rotação dupla à direita para corrigir o balanceamento.

Se o balanceamento for menor que -1, isso indica que a subárvore à direita é mais pesada. Da mesma forma, a função verifica as alturas dos filhos à direita e decide se deve realizar uma rotação simples à esquerda ou uma rotação dupla à esquerda para corrigir o balanceamento.

Após a rotação, a função verifica o nó pai do nó “no” (usando a função encontrarPai(no)) para atualizar os ponteiros do pai corretamente após a rotação.

A função continua a verificar o balanceamento recursivamente subindo na árvore a partir do nó pai, chamando verificarBalanceamento(pai).

Se a raiz da árvore for afetada pela rotação (o que pode ocorrer se o nó “no” for a raiz original da árvore), a função atualiza a raiz da árvore para ser o novo nó “no”.

Interface gráfica do usuário, Texto

Descrição gerada automaticamente com confiança média

Não fui capaz de implementar essa lógica, realizei algumas pesquisas, algumas perguntas para o monitor, mas ao implementar a lógica, de alguma forma a árvore “quebrava”, talvez seja algum erro de lógica em outra parte do código, de qualquer forma não consegui realizar a implementação dessa função.

1. **Análise Crítica**
   1. *Árvore Binária:*

Professor acho que essa árvore aqui é perfeita, a implementação dela é parcialmente simples e mais direta também, além de ser mais legal de mexer e utilizar, o único problema é o fato do balanceamento mesmo que em si a árvore pode ficar toda torta para um lado só, parecendo um galho perdido, de resto me diverti montando esse código.

* 1. *Árvore AVL*

Professor odiei esse negócio aqui, descobri que sou burro de mais e que a vida de um programador desesperado me tirou boas noites de sono. Brincadeiras a parte a construção desse código foi realmente um desafio, e imagino que eu não tenha conseguido ainda, porém oque achei interessante dessa árvore é seu balanceamento em si, e manter uma árvore “bonita”, a única vantagem que percebo é a complexidade dela em si.

1. **Análise de Desempenho**
   1. *100 Valores*

Tempo de Inserção Árvore Binária:



Tempo De Inserção Árvore AVL:



* 1. *500 Valores*

Tempo de Inserção Árvore Binária:



Tempo De Inserção Árvore AVL:



* 1. *1000 Valores*

Tempo de Inserção Árvore Binária:



Tempo De Inserção Árvore AVL:

**

* 1. *10000 Valores*

Tempo de Inserção Árvore Binária:



Tempo De Inserção Árvore AVL:



* 1. *20000 Valores*

Tempo de Inserção Árvore Binária:



Tempo De Inserção Árvore AVL:



1. Análise de Busca
   1. Valores Inseridos nas duas árvores:

**14-15-4-9-7-18-3-5-16-4-20-17-9-14-5**

* 1. *Árvore Binária:*

Busca do Número 5

****

* 1. *Árvore AVL:*

Busca do Número 5



1. **Análise de Remoção**
   1. *Com os mesmo valores inseridos em cima*

**14-15-4-9-7-18-3-5-16-4-20-17-9-14-5**

* 1. *Árvore Binária:*

*Remoção do número 5*

*Texto

Descrição gerada automaticamente*

* 1. *Árvore AVL:*

**Infelizmente a remoção na árvore AVL não fui capaz de implementar.**