Árvores AVL

Estruturas de Dados e Algoritmos

Árvores AVL

1. Uma árvore AVL é uma árvore de pesquisa binária que é de altura balanceada
2. Para cada nó x, as alturas das subárvores da esquerda e direita de x diferem por no máximo 1.

Árvores AVL

1. No caso da inserção em uma árvore AVL, primeiro um nó é inserido no lugar apropriado na ordem da árvore de pesquisa binária
2. As alturas dos filhos da esquerda e da direita de algum nó podem diferir por 2. Nesse momento, é preciso realizar o balanceamento da árvore por meio de rotações.

Rotações

Rotações

1. São operações em árvores que alteram a estrutura dos ponteiros
2. No caso de árvores AVL, as rotações são utilizadas para balancear árvores desbalanceadas
3. Existem **quatro** rotações que são classificadas em dois tipos:
   - Rotações **simples**: Rotação para a direita, rotação para a esquerda
   - Rotações **duplas**: Rotação direita esquerda, rotação esquerda direita

1. Rotação para a Direita

```
202     NodoArvoreAvl rotacaoDireita(NodoArvoreAvl a){
203         NodoArvoreAvl aux = a.nodoEsquerda;
204         a.nodoEsquerda = aux.nodoDireita;
205         aux.nodoDireita = a;
206         return aux;
207     }
```

1. Rotação para a Direita

- Guarda-se uma cópia do nodo da esquerda em uma variável auxiliar

```
202     NodoArvoreAvl rotacaoDireita(NodoArvoreAvl a){
203         NodoArvoreAvl aux = a.nodoEsquerda;
204         a.nodoEsquerda = aux.nodoDireita;
205         aux.nodoDireita = a;
206         return aux;
207     }
```

1. Rotação para a Direita

- Guarda-se uma cópia do nodo da esquerda em uma variável auxiliar
- Agora, a posição do nodo da esquerda está liberada para receber o nodo da direita da própria variável auxiliar

```
202     NodoArvoreAvl rotacaoDireita(NodoArvoreAvl a){
203         NodoArvoreAvl aux = a.nodoEsquerda;
204         a.nodoEsquerda = aux.nodoDireita;
205         aux.nodoDireita = a;
206         return aux;
207     }
```

1. Rotação para a Direita

- Guarda-se uma cópia do nodo da esquerda em uma variável auxiliar
- Agora, a posição do nodo da esquerda está liberada para receber o nodo da direita da própria variável auxiliar
- O nodo da direita da variável auxiliar deve então receber o nodo original enviado como parâmetro

```
202     NodoArvoreAvl rotacaoDireita(NodoArvoreAvl a){
203         NodoArvoreAvl aux = a.nodoEsquerda;
204         a.nodoEsquerda = aux.nodoDireita;
205         aux.nodoDireita = a;
206         return aux;
207     }
```

Rotações

1. Rotação para a Direita

- Guarda-se uma cópia do nodo da esquerda em uma variável auxiliar
- Agora, a posição do nodo da esquerda está liberada para receber o nodo da direita da própria variável auxiliar
- O nodo da direita da variável auxiliar deve então receber o nodo original enviado como parâmetro
- Por último deve-se retornar o nodo auxiliar como resultado

```
202     NodoArvoreAvl rotacaoDireita(NodoArvoreAvl a){
203         NodoArvoreAvl aux = a.nodoEsquerda;
204         a.nodoEsquerda = aux.nodoDireita;
205         aux.nodoDireita = a;
206         return aux;
207     }
```

1. Rotação para a Esquerda

- Segue a mesma lógica da rotação para a direita. O código é o mesmo somente que com todos os nodos invertidos (o que era nodoEsquerda vira nodoDireita, e vice-versa)

```
100     NodoArvoreAvl rotacaoEsquerda(NodoArvoreAvl a){
101         NodoArvoreAvl aux = a.direita;
102         a.direita = aux.esquerda;
103         aux.esquerda = a;
104         return aux;
105     }
```

1. Rotação Dupla Esquerda Direita

- Em uma rotação dupla EsquerdaDireita, primeiro se faz uma rotação para a esquerda no filho da esquerda da árvore enviada como parâmetro.
- Após isso, é feita uma rotação à direita na árvore enviada como parâmetro

```
188     NodoArvoreAvl rotacaoEsquerdaDireita(NodoArvoreAvl a){
189         a.nodoEsquerda = a.rotacaoEsquerda(a.nodoEsquerda);
190         a = a.rotacaoDireita(a);
191         return a;
192     }
```

1. Rotação Dupla Direita Esquerda

- Em uma rotação dupla DireitaEsquerda, primeiro se faz uma rotação para à direita no filho da direita da árvore enviada como parâmetro.
- Após isso, é feita uma rotação à esquerda na árvore enviada como parâmetro

```
87      NodoArvoreAvl rotacaoDireitaEsquerda(NodoArvoreAvl a){
88          a.direita = a.rotacaoDireita(a.direita);
89          a = a.rotacaoEsquerda(a);
90          return a;
91      }
```

Inserção Balanceada

```
NodoArvoreAvl insereBalanceado(NodoArvoreAvl a, int valor){
    if (a == null) {
        a = new NodoArvoreAvl(valor, null, null);
    }
    else if (valor < a.valor){ //insere na esquerda
         a.nodoEsquerda = insereBalanceado(a.nodoEsquerda, valor);
         if (this.altura(a.nodoEsquerda) -
             this.altura(a.nodoDireita) == 2){
                if (valor < a.nodoEsquerda.valor){
                    a = a.rotacaoDireita(a);}
                else{
                    a = a.rotacaoEsquerdaDireita(a);}
         }
    }
    else {//insere na direita
         a.nodoDireita = insereBalanceado(a.nodoDireita, valor);
         if (this.altura(a.nodoEsquerda)-
             this.altura(a.nodoDireita) == -2){
                if (valor > a.nodoDireita.valor){
                    a = a.rotacaoEsquerda(a);}
                else{
                    a = a.rotacaoDireitaEsquerda(a);}
         }
    }
    return a;
}
```

Inserção Balanceada

O método de inserção balanceada possui a mesma estrutura do método de inserção em árvores binárias.

Porém, após realizar uma inserção no nodo da direita ou da esquerda é realizada uma verificação para avaliar se essa inserção irá acarretar em desbalanceamento

```
NodoArvoreAvl insereBalanceado(NodoArvoreAvl a, int valor){
    if (a == null) {
        a = new NodoArvoreAvl(valor, null, null);
    }
    else if (valor < a.valor){ //insere na esquerda
        a.nodoEsquerda = insereBalanceado(a.nodoEsquerda, valor);
        if (this.altura(a.nodoEsquerda) -
            this.altura(a.nodoDireita) == 2){
                if (valor < a.nodoEsquerda.valor){
                    a = a.rotacaoDireita(a);}
                else{
                    a = a.rotacaoEsquerdaDireita(a);}
        }
    }
    else {//insere na direita
        a.nodoDireita = insereBalanceado(a.nodoDireita, valor);
        if (this.altura(a.nodoEsquerda)-
            this.altura(a.nodoDireita) == -2){
                if (valor > a.nodoDireita.valor){
                    a = a.rotacaoEsquerda(a);}
                else{
                    a = a.rotacaoDireitaEsquerda(a);}
        }
    }
    return a;
}
```

Inserção Balanceada

Os novos códigos com relação ao código da árvore binária de pesquisa estão marcados ao lado

```
NodoArvoreAvl insereBalanceado(NodoArvoreAvl a, int valor){
    if (a == null) {
        a = new NodoArvoreAvl(valor, null, null);
    }
    else if (valor < a.valor){ //insere na esquerda
        a.nodoEsquerda = insereBalanceado(a.nodoEsquerda, valor);
        if (this.altura(a.nodoEsquerda) -
            this.altura(a.nodoDireita) == 2){
                if (valor < a.nodoEsquerda.valor){
                    a = a.rotacaoDireita(a);}
                else{
                    a = a.rotacaoEsquerdaDireita(a);}
        }
    }
    else {//insere na direita
        a.nodoDireita = insereBalanceado(a.nodoDireita, valor);
        if (this.altura(a.nodoEsquerda)-
            this.altura(a.nodoDireita) == -2){
                if (valor > a.nodoDireita.valor){
                    a = a.rotacaoEsquerda(a);}
                else{
                    a = a.rotacaoDireitaEsquerda(a);}
        }
    }
    return a;
}
```

Inserção Balanceada

A lógica é a seguinte:

1. É realizada uma subtração da altura do nodo da esquerda pela altura do nodo da direita (isso é o calculo do fator de balanceamento ou fator h)

```
NodoArvoreAvl insereBalanceado(NodoArvoreAvl a, int valor){
    if (a == null) {
        a = new NodoArvoreAvl(valor, null, null);
    }
    else if (valor < a.valor){ //insere na esquerda
        a.nodoEsquerda = insereBalanceado(a.nodoEsquerda, valor);
        if (this.altura(a.nodoEsquerda) -
            this.altura(a.nodoDireita) == 2){
                if (valor < a.nodoEsquerda.valor){
                    a = a.rotacaoDireita(a);}
                else{
                    a = a.rotacaoEsquerdaDireita(a);}
        }
    }
    else {//insere na direita
        a.nodoDireita = insereBalanceado(a.nodoDireita, valor);
        if (this.altura(a.nodoEsquerda)-
            this.altura(a.nodoDireita) == -2){
                if (valor > a.nodoDireita.valor){
                    a = a.rotacaoEsquerda(a);}
                else{
                    a = a.rotacaoDireitaEsquerda(a);}
        }
    }
    return a;
}
```

Inserção Balanceada

A lógica é a seguinte:

1. No **primeiro** caso, se o fator h for 2 significa que haverá desbalanceamento com a inserção
2. Avalia-se então se a inserção está acontecendo à esquerda do nodo da esquerda (rotação simples) ou a à direita do nodo da esquerda (rotação dupla)

```
NodoArvoreAvl insereBalanceado(NodoArvoreAvl a, int valor){
    if (a == null) {
        a = new NodoArvoreAvl(valor, null, null);
    }
    else if (valor < a.valor){ //insere na esquerda
        a.nodoEsquerda = insereBalanceado(a.nodoEsquerda, valor);
        if (this.altura(a.nodoEsquerda) -
            this.altura(a.nodoDireita) == 2){
                if (valor < a.nodoEsquerda.valor){
                    a = a.rotacaoDireita(a);}
                else{
                    a = a.rotacaoEsquerdaDireita(a);}
        }
    }
    else {//insere na direita
        a.nodoDireita = insereBalanceado(a.nodoDireita, valor);
        if (this.altura(a.nodoEsquerda)-
            this.altura(a.nodoDireita) == -2){
                if (valor > a.nodoDireita.valor){
                    a = a.rotacaoEsquerda(a);}
                else{
                    a = a.rotacaoDireitaEsquerda(a);}
        }
    }
    return a;
}
```

Inserção Balanceada

A lógica é a seguinte:

1. No **segundo** caso, se o fator h for -2 significa que haverá desbalanceamento com a inserção
2. Avalia-se então se a inserção está acontecendo à direita do nodo da direita (rotação simples) ou a à esquerda do nodo da direita (rotação dupla)

```
NodoArvoreAvl insereBalanceado(NodoArvoreAvl a, int valor){
    if (a == null) {
        a = new NodoArvoreAvl(valor, null, null);
    }
    else if (valor < a.valor){ //insere na esquerda
        a.nodoEsquerda = insereBalanceado(a.nodoEsquerda, valor);
        if (this.altura(a.nodoEsquerda) -
            this.altura(a.nodoDireita) == 2){
                if (valor < a.nodoEsquerda.valor){
                    a = a.rotacaoDireita(a);}
                else{
                    a = a.rotacaoEsquerdaDireita(a);}
        }
    }
    else {//insere na direita
        a.nodoDireita = insereBalanceado(a.nodoDireita, valor);
        if (this.altura(a.nodoEsquerda)-
            this.altura(a.nodoDireita) == -2){
                if (valor > a.nodoDireita.valor){
                    a = a.rotacaoEsquerda(a);}
                else{
                    a = a.rotacaoDireitaEsquerda(a);}
        }
    }
    return a;
}
```

Inserção Balanceada

Melhorando o código:

1. Uma alternativa para deixar o código mais limpo, é criar uma função de balanceamento

```
NodoArvoreAvl balanceamento(NodoArvoreAvl a){
    if (this.altura(a.esquerda) - this.altura(a.direita) == -2) {
        if (this.altura(a.direita.esquerda) -
            this.altura(a.direita.direita) == -1){// dois sinai
            a = a.rotacaoEsquerda(a);}
        else {//rotacao direita esquerda - um sinal negativo e
            a = rotacaoDireitaEsquerda(a);}
    }
    else
     if (this.altura(a.esquerda) - this.altura(a.direita) == 2)
        if(this.altura(a.esquerda.esquerda) -
           this.altura(a.esquerda.direita) == 1){//dois sinais
            a = rotacaoDireita(a);}
        else {//um sinal positivo e um negativo
            a = rotacaoEsquerdaDireita(a);}
    }
    return a;
}
```

Inserção Balanceada

Melhorando o código:

1. Essa função é então chamada antes do final do retorno da inserção

```
NodoArvoreAvl insereBalanceado02(NodoArvoreAvl a, int valor){
    if (a == null) {
        a = new NodoArvoreAvl(valor, null, null);
    }
    else if (valor < a.valor){ //insere na esquerda
        a.esquerda = insereBalanceado02(a.esquerda, valor);
        System.out.println("inserindo "+valor+" em "+ a.valor)
    }
    else {//insere na direita
        a.direita = insereBalanceado02(a.direita, valor);
        System.out.println("inserindo "+valor+" em "+ a.valor)
    }
    a = balanceamento(a);
    return a;
}
```

```
NodoArvoreAvl retiraValor(NodoArvoreAvl a, int numero){
    if (a == null) return null;
    if (numero < a.valor) {
        a.nodoEsquerda = retiraValor(a.nodoEsquerda, numero);
        if (this.altura(a.nodoEsquerda) - this.altura(a.nodoDireita) == -2) {
            if (this.altura(a.nodoDireita.nodoEsquerda) -
                this.altura(a.nodoDireita.nodoDireita) == -1){// dois sinais
                a = a.rotacaoEsquerda(a);}
            else {//rotacao direita esquerda - um sinal negativo e um positi
                a = rotacaoDireitaEsquerda(a);}}}
    else if (numero > a.valor){
        a.nodoDireita = retiraValor(a.nodoDireita, numero);
        if (this.altura(a.nodoEsquerda) - this.altura(a.nodoDireita) == 2) {
            if(this.altura(a.nodoEsquerda.nodoEsquerda) -
               this.altura(a.nodoEsquerda.nodoDireita) == 1){//dois sinais po
                a = rotacaoDireita(a);}
            else {//um sinal positivo e um negativo
                a = rotacaoEsquerdaDireita(a);}}}
    else {
        if (a.nodoDireita == null && a.nodoEsquerda == null)
            return null;
        if (a.nodoDireita == null)  return a.nodoEsquerda;
        if (a.nodoEsquerda == null) return a.nodoDireita;
        NodoArvoreAvl temp = a.nodoEsquerda;
        while (temp.nodoDireita != null) {
            temp = temp.nodoDireita;
        }
        a.valor = temp.valor;
        temp.valor = numero;
        a.nodoEsquerda = retiraValor(a.nodoEsquerda, numero);}
    return a;
}
```

Exercícios

Exercícios

1. afasfasdf