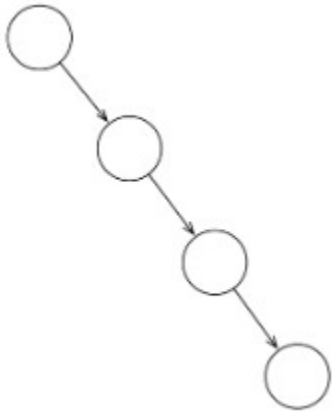
Relatório Entrega 1 e 2

1. A função geraArvoreDegenerada gera uma árvore degenerada, o que significa que todos os nós têm no máximo um filho. Isso é alcançado adicionando os nós sequencialmente à árvore, o que leva à formação de uma cadeia linear de nós.

O que queremos dizer com uma cadeia linear de nós é que, como o método geraArvoreDegenerada sempre incrementa 1 na matrícula, o próximo nó sempre é maior que o nó raiz, logo inserindo sempre a direita.

Exemplo da árvore formada com o método geraArvoreDegenerada.



2. O pior caso para 100 nós, seria o método percorrer os 100 nós. A mesma coisa para 200, 500, 1000, etc... Isso acontece pois o método gera uma árvore encadeada sequencialmente, não tendo necessidade de verificar outros nós, apenas o próximo. Portanto, o número de elementos na árvore determina o número máximo de nós que serão percorridos durante a busca no pior caso.

3. Chegamos na conclusão que a ordem de complexidade identificada é O(1) para o melhor caso, sendo então o nó pesquisado como o primeiro nó da árvore e O(n) para o pior caso, sendo o nó buscado o último nó da árvore. Isso acontece devido à necessidade de percorrer todos os nós da árvore para o pior caso.

4.

```java
//---Este é o método citado na questão 4 do primeiro relatório
    public void geraArvoreDegenerada(int n, IArvoreBinaria<Aluno> arv){
        //inicio matricula com o valor da constante matriculaBase
        int i,matricula= matriculaBase;
        String nome;
        for(i=1;i<=n;i++){
            //Cada vez que entra a matrícula é incrementada em 1.
            matricula++;
            nome = geraNomeCompleto();
            //Aqui crio um aluno com os dados gerados e o adiciono na árvore.
            arv.adicionar(new Aluno(matricula,nome));
        }
    }
```

Esse é o método que gera a árvore degenerada.

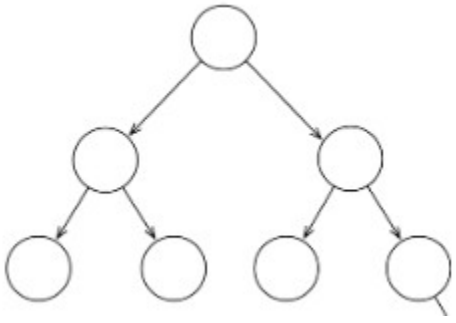
O método recebe como parâmetro a árvore que será inserida, e a quantidade de nós que serão inseridos na árvore.

No começo as variáveis são inicializadas, tendo i como um contador para o loop, matricula recebendo a propriedade da classe que o método faz parte e o nome do aluno.

O método executa um loop, tendo como condição de parada o contador "i" sendo o tamanho de "n", que é a quantidade de nós que serão inseridos na árvore.

O número de operações básicas é proporcional a "n", fazendo com que o loop seja chamado n vezes, ou seja O(n).

5. O método geraArvorePerfeitamenteBalanceada cria uma árvore binária perfeitamente balanceada, onde cada nó tem dois filhos e a diferença na altura entre os subárvores esquerda e direita de qualquer nó é no máximo 1.



6. Em uma árvore perfeitamente balanceada podemos contar o número máximo de nós percorridos pela altura da árvore. Em uma árvore binária de busca balanceada, a busca avança em cada nível da árvore com base na comparação do valor buscado com o valor do nó atual, reduzindo pela metade o número de nós a serem considerados em cada nível. Logo, podemos calcular sendo $\log_2(n)$:

Arvore de 100 elementos:

$\log_2(100) = 6.64$ → A altura será 6 porque a altura de uma árvore completa é a parte inteira do logaritmo de base 2 do número de nós. Logo, no pior caso, será preciso percorrer 6 nós.

Arvore de 200 elementos:

$\log_2(200) = 7.64$ → altura 7. Logo, pior caso percorrerá 7 nós.

Arvore de 1000 elementos:

$\log_2(1000) = 9.97$. → altura 9. Logo, pior caso percorrerá 9 nós.

7. Para uma árvore binária de busca perfeitamente balanceada com n nós, a altura h da árvore é O(log n). Isso ocorre porque, em cada comparação, a árvore é efetivamente dividida ao meio, reduzindo pela metade o número de nós a serem considerados em cada nível.

8.

-Divisão: O método começa verificando se min é menor ou igual a max (if(min<max)):

```
//---Este é o método citado na questão 8 do primeiro relatório
    public void geraArvorePerfeitamenteBalanceada(int min, int max, IArvoreBinaria<Aluno> arv){
        //Se o valor da menor matrícula for menor ou igual ou maior valor é sinal que ainda preciso inserir elementos na árvore
        //Senão essa recursão acabou...
        if (min <= max){
            //Calculo a matrícula média desta geração e insiro um aluno com essa matrícula na árvore
            int media = (min+max)/2;
            int matricula = matriculaBase+media;
            String nome = geraNomeCompleto();
            arv.adicionar(new Aluno(matricula,nome));
            //Chamo recursivamente para continuar inserindo os elementos com matrículas menores que a média
            geraArvorePerfeitamenteBalanceada(min,media-1,arv);
            //Chamo recursivamente para continuar inserindo os elementos com matrículas maiores que a média
            geraArvorePerfeitamenteBalanceada(media+1,max,arv);
        }
    }
```

Se essa condição for verdadeira, ele calcula o valor médio media (media = (min+max)/2).

```java
//---Este é o método citado na questão 8 do primeiro relatório
    public void geraArvorePerfeitamenteBalanceada(int min, int max, IArvoreBinaria<Aluno> arv){
        //Se o valor da menor matrícula for menor ou igual ou maior valor é sinal que ainda preciso inserir elementos na árvore
        //Senão essa recursão acabou...
        if (min <= max){
            //Calculo a matrícula média desta geração e insiro um aluno com essa matrícula na árvore
            int media = (min+max)/2;
            int matricula = matriculaBase+media;
            String nome = geraNomeCompleto();
            arv.adicionar(new Aluno(matricula,nome));
            //Chamo recursivamente para continuar inserindo os elementos com matrículas menores que a média
            geraArvorePerfeitamenteBalanceada(min,media-1,arv);
            //Chamo recursivamente para continuar inserindo os elementos com matrículas maiores que a média
            geraArvorePerfeitamenteBalanceada(media+1,max,arv);
        }
    }
```

-Adição de Nó: Em seguida, o método cria um novo nó (new Aluno(matricula,nome)):

```java
//---Este é o método citado na questão 8 do primeiro relatório
    public void geraArvorePerfeitamenteBalanceada(int min, int max, IArvoreBinaria<Aluno> arv){
        //Se o valor da menor matrícula for menor ou igual ou maior valor é sinal que ainda preciso inserir elementos na árvore
        //Senão essa recursão acabou...
        if (min <= max){
            //Calculo a matrícula média desta geração e insiro um aluno com essa matrícula na árvore
            int media = (min+max)/2;
            int matricula = matriculaBase+media;
            String nome = geraNomeCompleto();
            arv.adicionar(new Aluno(matricula,nome));
            //Chamo recursivamente para continuar inserindo os elementos com matrículas menores que a média
            geraArvorePerfeitamenteBalanceada(min,media-1,arv);
            //Chamo recursivamente para continuar inserindo os elementos com matrículas maiores que a média
            geraArvorePerfeitamenteBalanceada(media+1,max,arv);
        }
    }
```

com a matrícula e o nome gerados e o adiciona à árvore binária (arv.adicionar(...)).

```java
//---Este é o método citado na questão 8 do primeiro relatório
    public void geraArvorePerfeitamenteBalanceada(int min, int max, IArvoreBinaria<Aluno> arv){
        //Se o valor da menor matrícula for menor ou igual ou maior valor é sinal que ainda preciso inserir elementos na árvore
        //Senão essa recursão acabou...
        if (min <= max){
            //Calculo a matrícula média desta geração e insiro um aluno com essa matrícula na árvore
            int media = (min+max)/2;
            int matricula = matriculaBase+media;
            String nome = geraNomeCompleto();
            arv.adicionar(new Aluno(matricula,nome));
            //Chamo recursivamente para continuar inserindo os elementos com matrículas menores que a média
            geraArvorePerfeitamenteBalanceada(min,media-1,arv);
            //Chamo recursivamente para continuar inserindo os elementos com matrículas maiores que a média
            geraArvorePerfeitamenteBalanceada(media+1,max,arv);
        }
    }
```

-Recorrência: Depois de adicionar o nó à árvore, o método faz duas chamadas recursivas:

geraArvorePerfeitamenteBalanceada(min,media-1,arv): Esta chamada é feita para inserir os

elementos com matrículas menores que a média (media – 1).

```
//---Este é o método citado na questão 8 do primeiro relatório
    public void geraArvorePerfeitamenteBalanceada(int min, int max, IArvoreBinaria<Aluno> arv){
        //Se o valor da menor matricula for menor ou igual ou maior valor é sinal que ainda preciso inserir elementos na árvore
        //Senão essa recursão acabou...
        if (min <= max){
            //Calculo a matrícula média desta geração e insiro um aluno com essa matrícula na árvore
            int media = (min+max)/2;
            int matricula = matriculaBase+media;
            String nome = geraNomeCompleto();
            arv.adicionar(new Aluno(matricula,nome));
            //Chamo recursivamente para continuar inserindo os elementos com matrículas menores que a média
            geraArvorePerfeitamenteBalanceada(min,media-1,arv);
            //Chamo recursivamente para continuar inserindo os elementos com matrículas maiores que a média
            geraArvorePerfeitamenteBalanceada(media+1,max,arv);
        }
    }
```

geraArvorePerfeitamenteBalanceada(media+1,max,arv): Esta chamada é feita para inserir os elementos com matrículas maiores que a média (media + 1).

```
//---Este é o método citado na questão 8 do primeiro relatório
    public void geraArvorePerfeitamenteBalanceada(int min, int max, IArvoreBinaria<Aluno> arv){
        //Se o valor da menor matricula for menor ou igual ou maior valor é sinal que ainda preciso inserir elementos na árvore
        //Senão essa recursão acabou...
        if (min <= max){
            //Calculo a matrícula média desta geração e insiro um aluno com essa matrícula na árvore
            int media = (min+max)/2;
            int matricula = matriculaBase+media;
            String nome = geraNomeCompleto();
            arv.adicionar(new Aluno(matricula,nome));
            //Chamo recursivamente para continuar inserindo os elementos com matrículas menores que a média
            geraArvorePerfeitamenteBalanceada(min,media-1,arv);
            //Chamo recursivamente para continuar inserindo os elementos com matrículas maiores que a média
            geraArvorePerfeitamenteBalanceada(media+1,max,arv);
        }
    }
```

Estas chamadas recursivas são responsáveis por dividir e inserir os elementos na árvore de forma balanceada. A cada nível da recursão, o intervalo de matrículas é dividido em duas metades, e o elemento mediano é inserido como a raiz da subárvore. Esse processo se repete até que todos os elementos sejam inseridos.

A altura da árvore perfeitamente balanceada cresce logaritmicamente em relação ao número de elementos (`log2(n)`). Em cada nível da recursão, um número constante de operações é realizado. Como o número de operações por nível é constante e a altura da árvore cresce logaritmicamente, a complexidade total do algoritmo é **O(log n)**.

9.

No pior caso, em uma árvore binária não balanceada, você terá que percorrer todos os nós da árvore para encontrar o elemento desejado. Portanto, se a árvore tem n nós, você precisará percorrer n nós no pior caso. Logo a ordem de complexidade da busca neste caso é O(n).

10.

Ocorre o **erro de estouro de pilha** apenas na geração da **árvore degenerada** devido à sua estrutura e à quantidade de chamadas recursivas necessárias para gerá-la (complexidade O(n)). A **árvore perfeitamente balanceada**, com sua estrutura mais eficiente (complexidade O(log n)), não apresenta esse problema e pode ser gerada sem problemas.