

CMP1054 – EDI Java

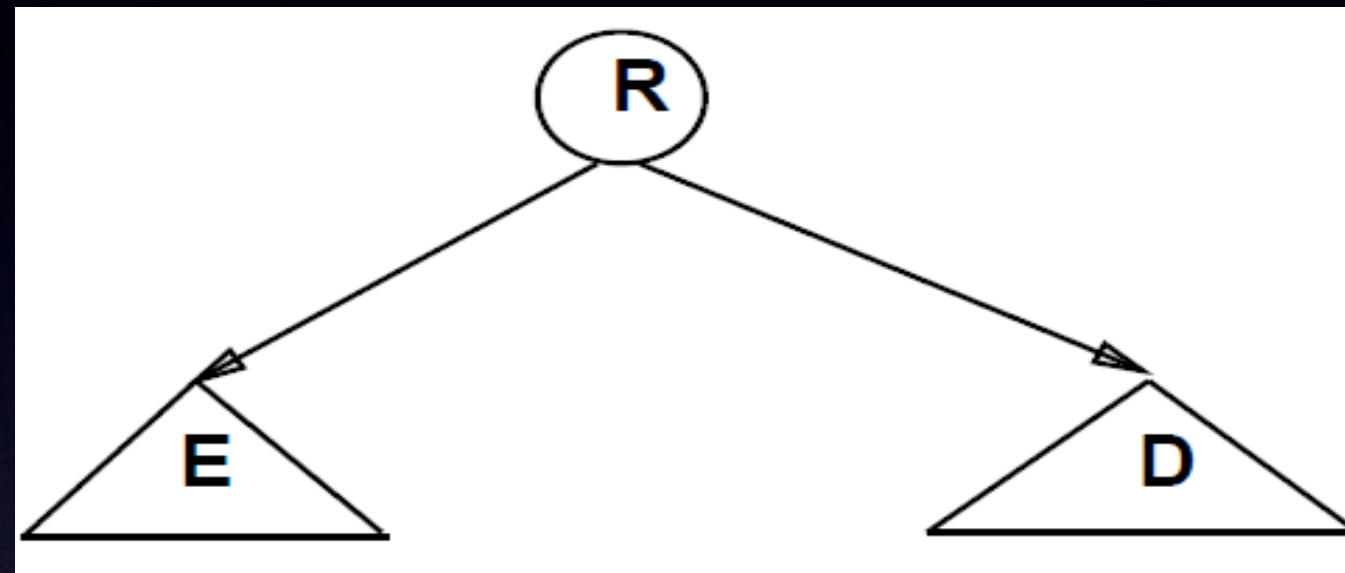
Árvores Binárias de Pesquisa.

Imprimir e testar integridade.

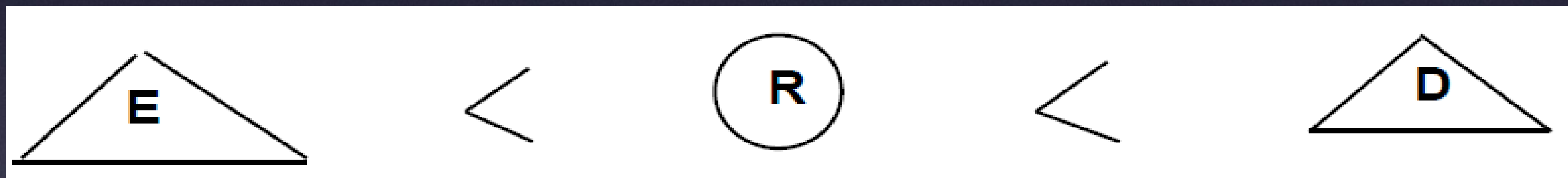
Prof. Dr. José Olímpio Ferreira

Árvores Binárias de Pesquisa

- Para qualquer nó que contenha um registro



- Temos a relação invariante



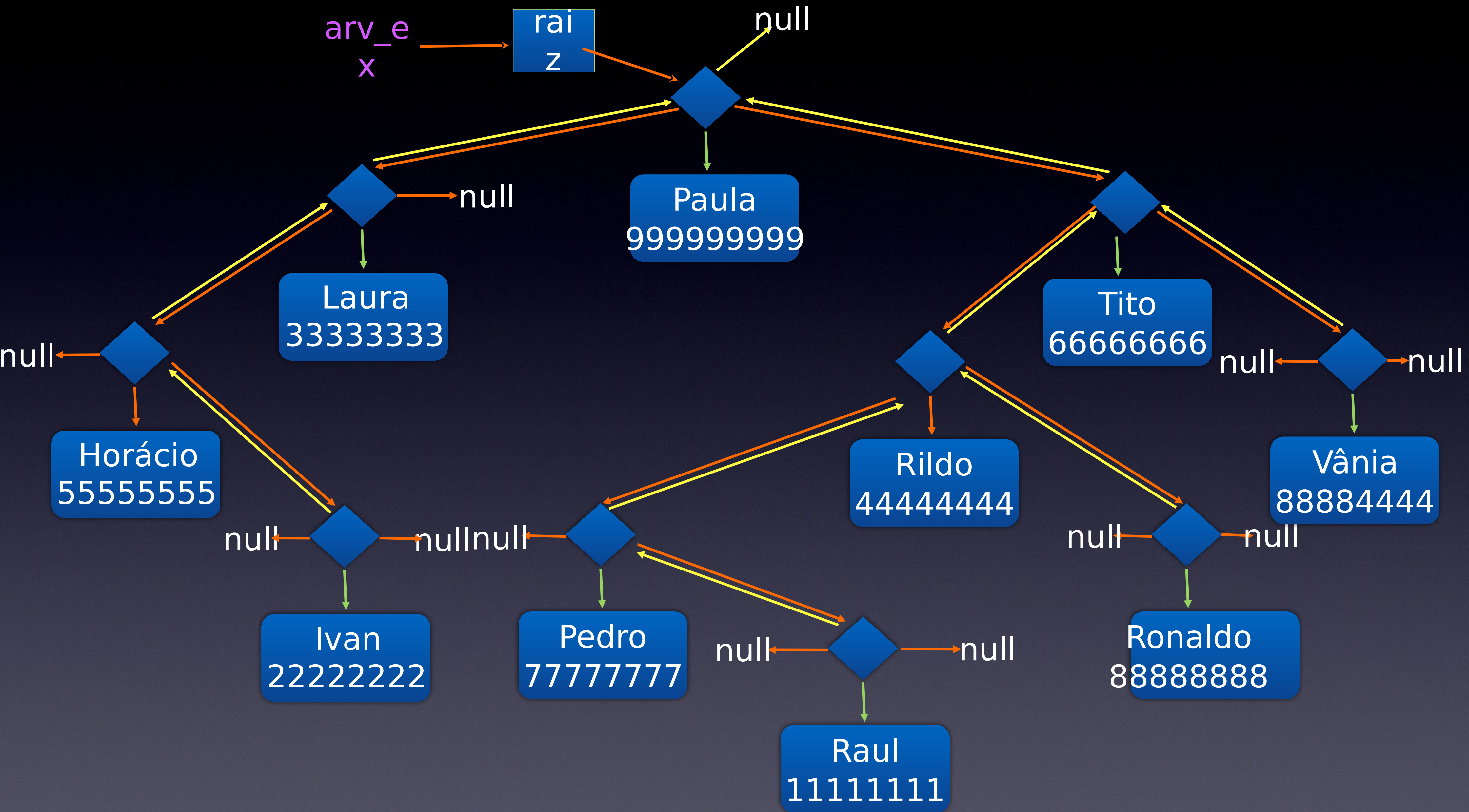
- Todos os registros com chaves menores estão na sub-árvore à esquerda.
- Todos os registros com chaves maiores estão na sub-árvore à direita.

classe Abp

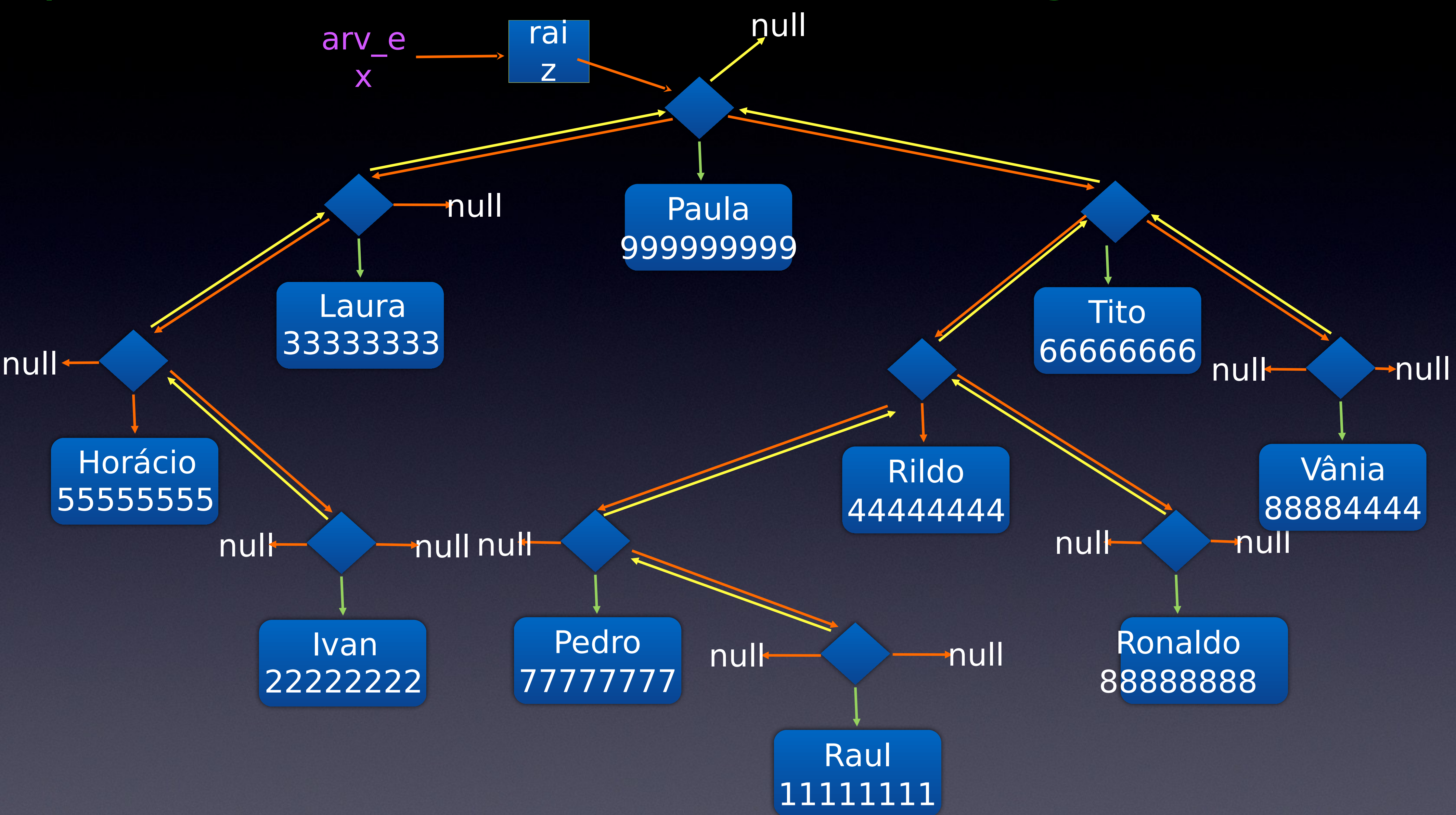
```
package ed1;
public class Abp {
    public class No {
        private Item dados;
        private No fd, fe, pai;
        public No(Item aux) {
            dados = aux;
            fd = fe = pai = null;
        }
    }
    private No raiz;
    private int tamanho;
    public Abp() {
        raiz = null;
    }
    public int getTamanho() {
        return tamanho;
    }
    public boolean vazia() {
        return (raiz == null);
    }
    private No consultar(Item obj) {
        // implementar depois
        return null;
    }
    public Item pesquisar(Item obj) {
        // implementar depois
        return null;
    }
    public boolean inserir(Item obj) {
        // implementar depois
        return true;
    }
}
```

```
private No maximo(No obj) {
    // implementar depois
    return obj;
}
private No minimo(No obj) {
    // implementar depois
    return obj;
}
private No antecessor(No obj) {
    // implementar depois
    return obj;
}
private No sucessor(No obj) {
    // implementar depois
    return obj;
}
public Item retirar(Item obj) {
    // implementar depois
    return obj;
}
public void visitaEmOrdem(StringBuffer aux) {
    // chamar método recursivo
}
public void visitaEmPreOrdem(StringBuffer aux) {
    // chamar método recursivo
}
public void visitaEmPosOrdem(StringBuffer aux) {
    // chamar método recursivo
}
public void testaIntegridade(StringBuffer aux) {
    // chamar método recursivo
}
}
```


Árvore Binária de Pesquisa

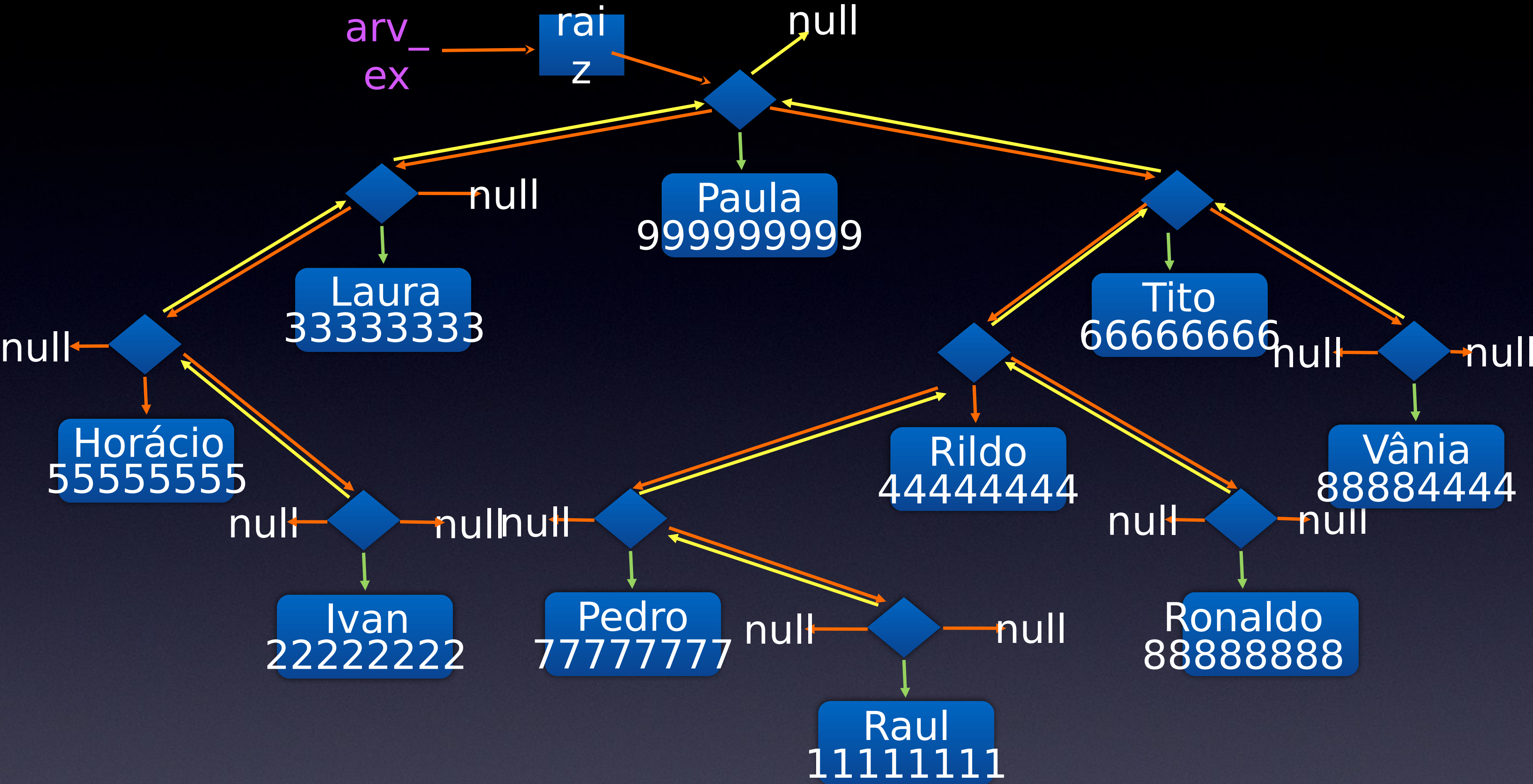


public void visitaEmPreOrdem(StringBuffer aux)



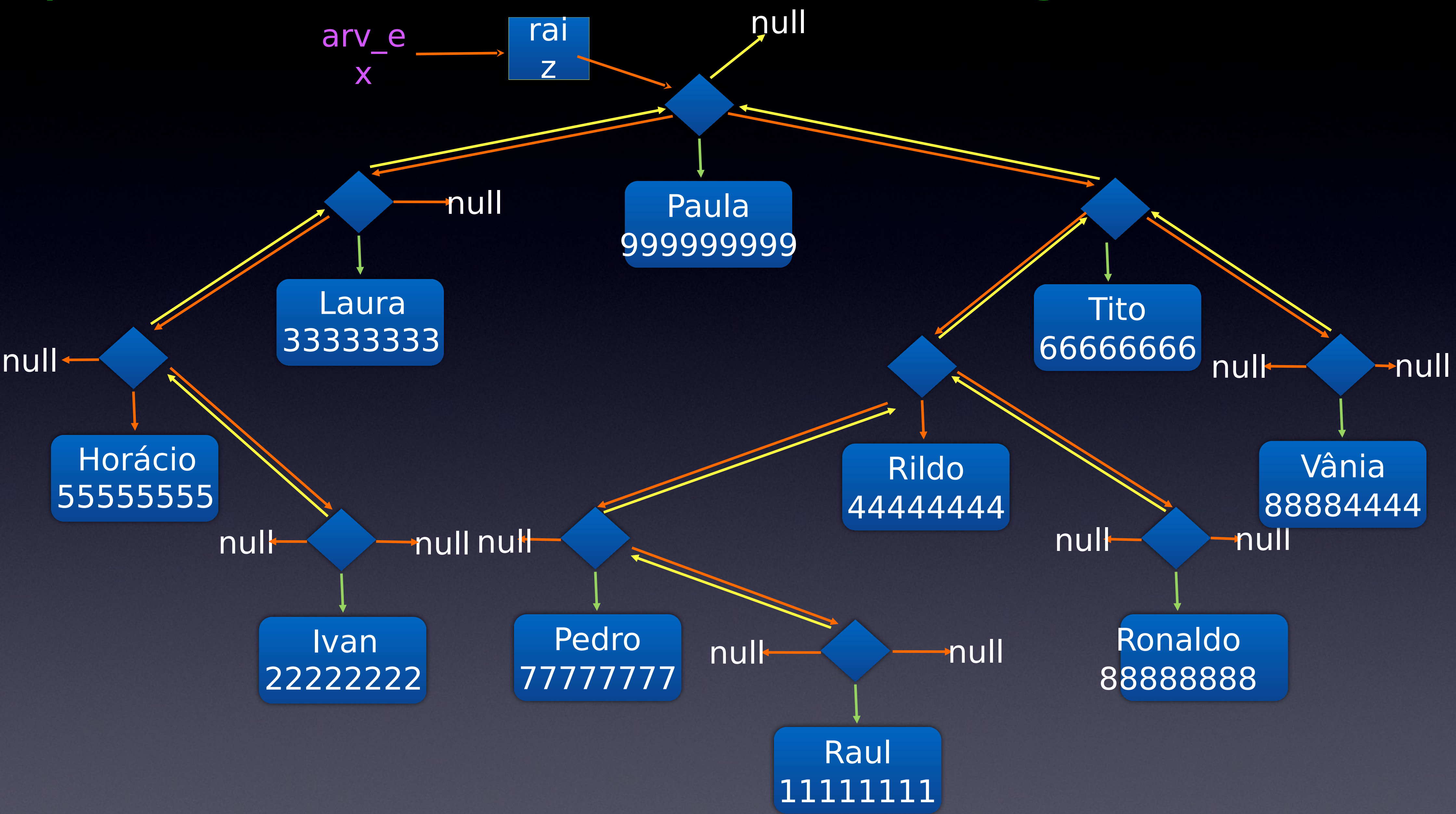

```
public void visitaEmPreOrdem(StringBuffer aux) {  
    // se a árvore estiver vazia não faz as chamadas  
recursivas  
    if(vazia()) { aux.append("Árvore vazia!"); }  
    // chamar método recursivo  
    else { visitaEmPreOrdem(aux, raiz); }  
}  
  
private void visitaEmPreOrdem(StringBuffer aux, No obj) {  
    if(obj != null) {  
        aux.append(obj.dados.toString());  
        visitaEmPreOrdem(aux, obj.fe);  
        visitaEmPreOrdem(aux, obj.fd);  
    }  
}
```


public void visitaEmPreOrdem(StringBuffer aux)



Paula - Laura - Horácio - Ivan - Tito - Rildo - Pedro - Raul - Ronaldo - Vânia

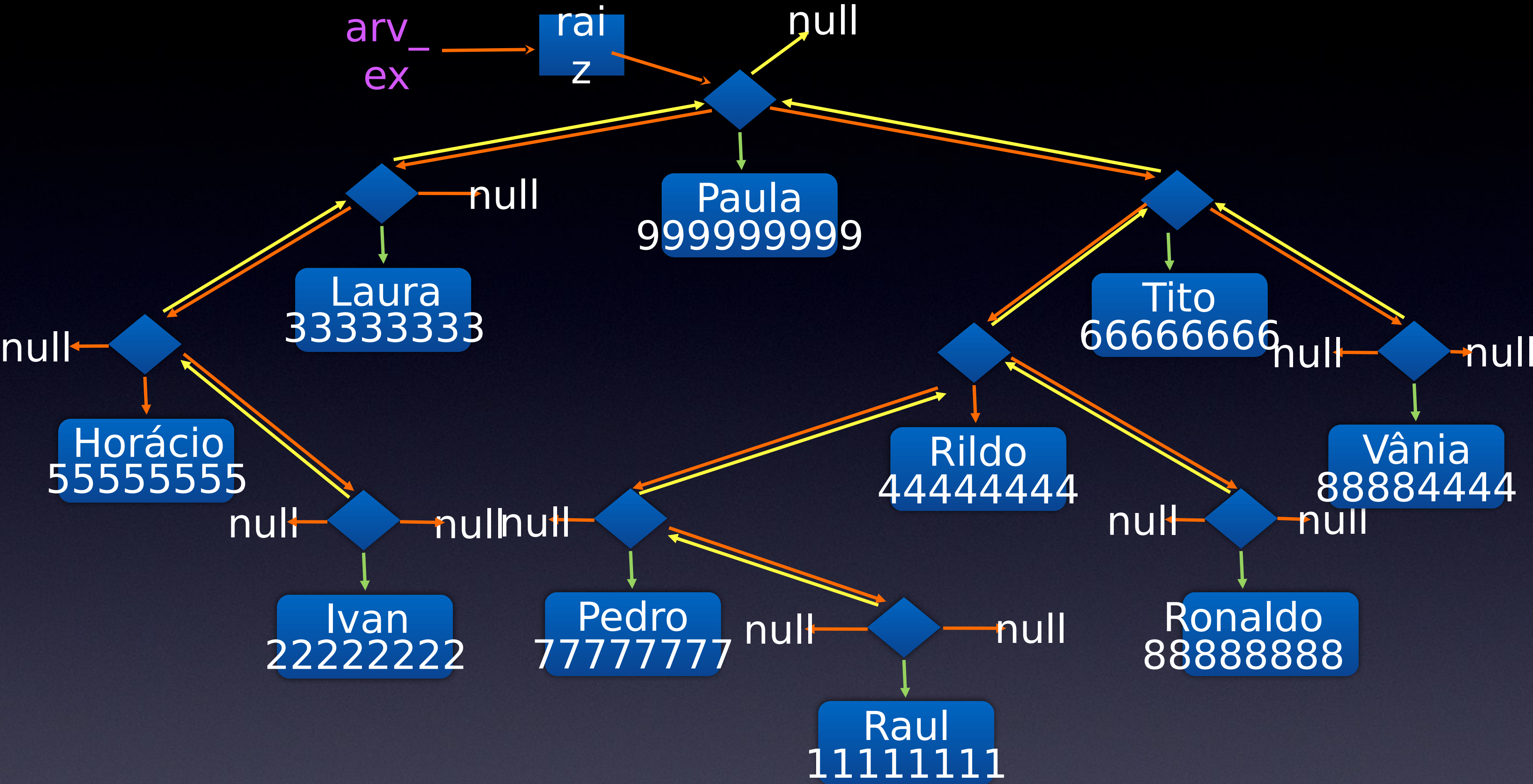
public void visitaEmOrdem(StringBuffer aux)




```
public void visitaEmOrdem(StringBuffer aux) {  
    // se a árvore estiver vazia não faz as chamadas  
recursivas  
    if(vazia()) { aux.append("Árvore vazia!"); }  
    // chamar método recursivo  
    else { visitaEmOrdem(aux, raiz); }  
}
```

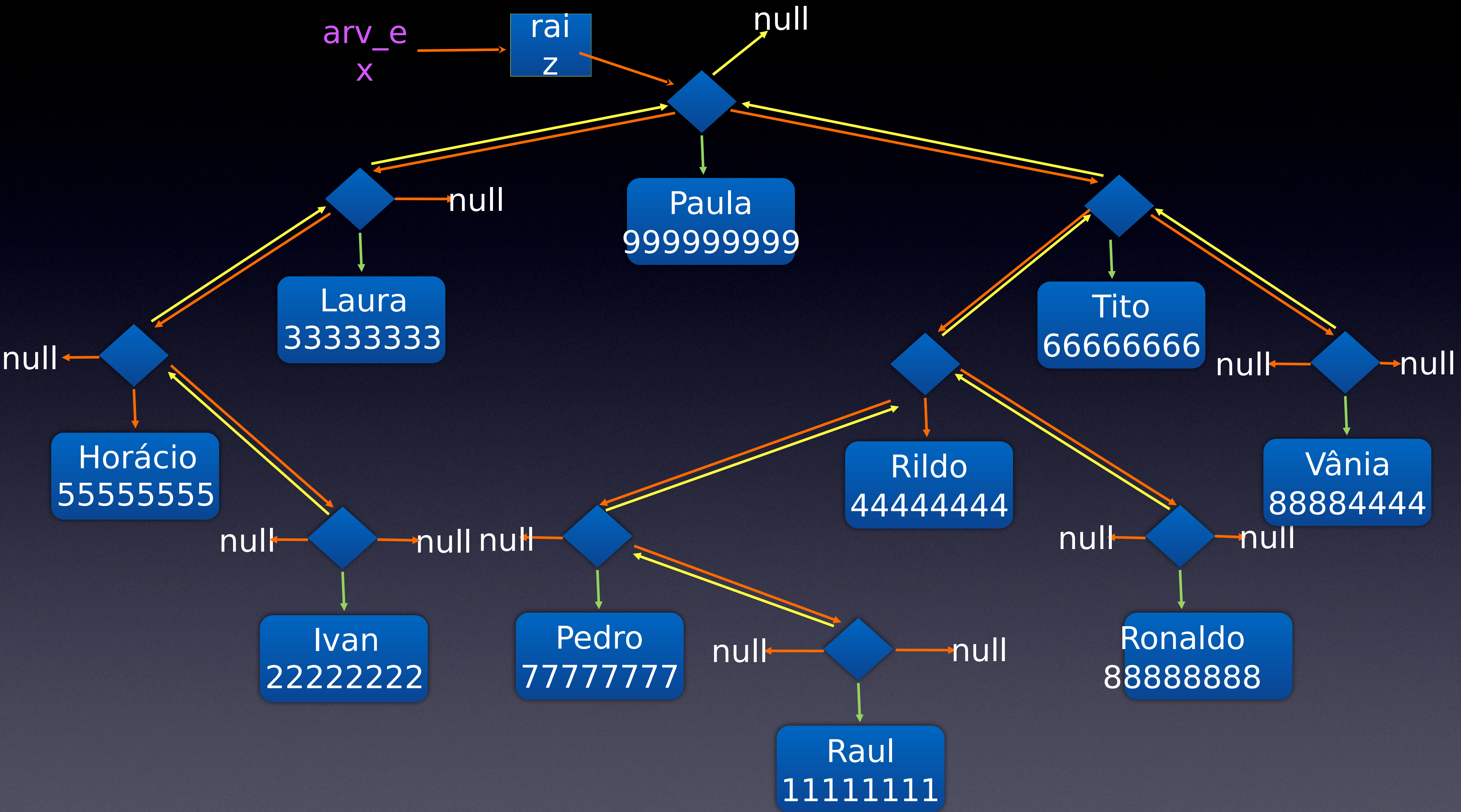
```
private void visitaEmOrdem(StringBuffer aux, No obj) {  
    if(obj != null) {  
        visitaEmOrdem(aux, obj.fe);  
        aux.append(obj.dados.toString());  
        visitaEmOrdem(aux, obj.fd);  
    }  
}
```


public void visitaEmOrdem(StringBuffer aux)



Horácio - Ivan - Laura - Paula - Pedro - Raul - Rildo - Ronaldo - Tito - Vânia

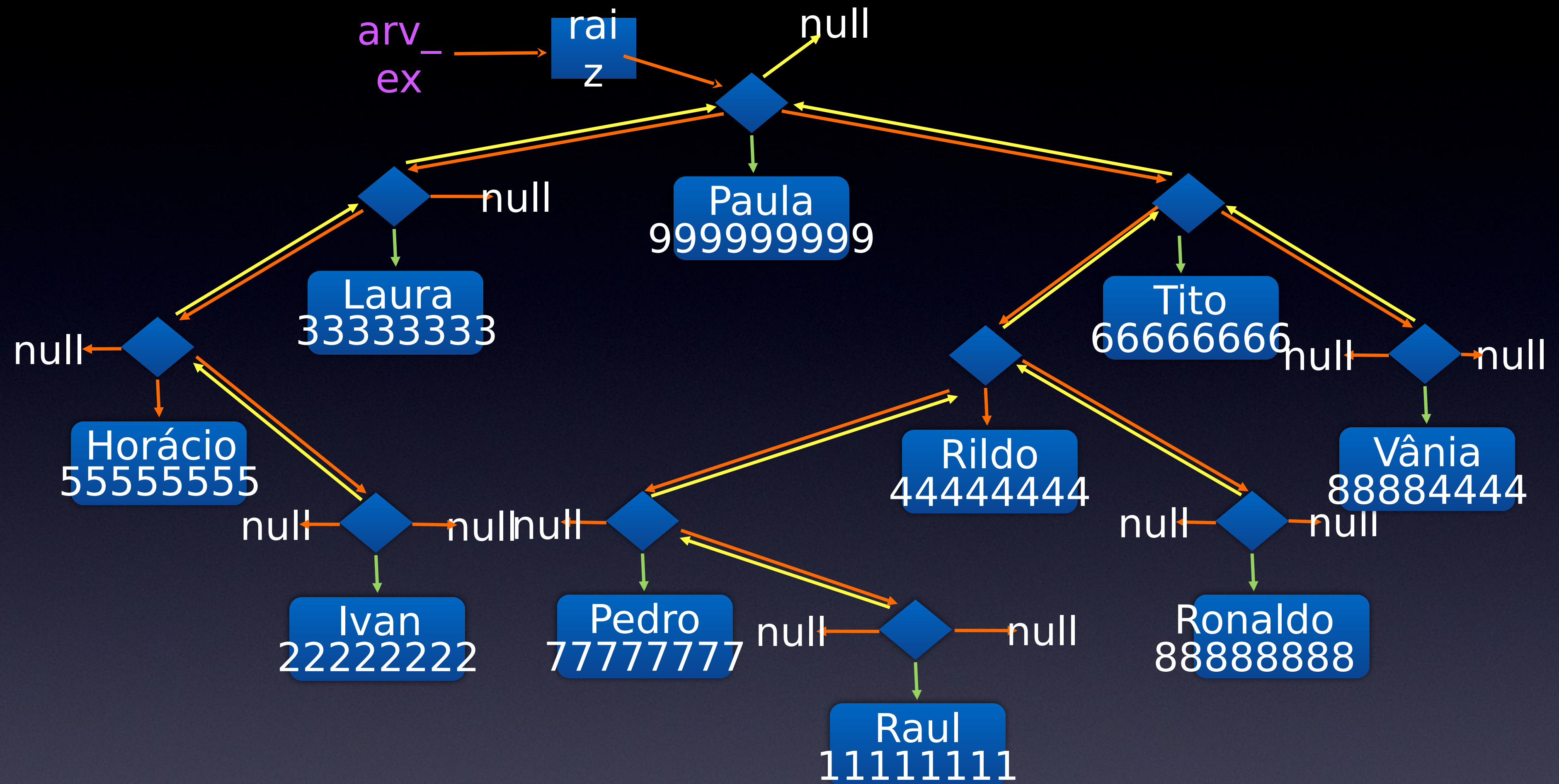

```
public void visitaEmPosOrdem(StringBuffer aux)
```




```
public void visitaEmPosOrdem(StringBuffer aux) {  
    // se a árvore estiver vazia não faz as chamadas  
recursivas  
    if(vazia()) { aux.append("Árvore vazia!"); }  
    // chamar método recursivo  
    else { visitaEmPosOrdem(aux, raiz); }  
}  
  
private void visitaEmPosOrdem(StringBuffer aux, No obj) {  
    if(obj != null) {  
        visitaEmPosOrdem(aux, obj.fe);  
        visitaEmPosOrdem(aux, obj.fd);  
        aux.append(obj.dados.toString());  
    }  
}
```

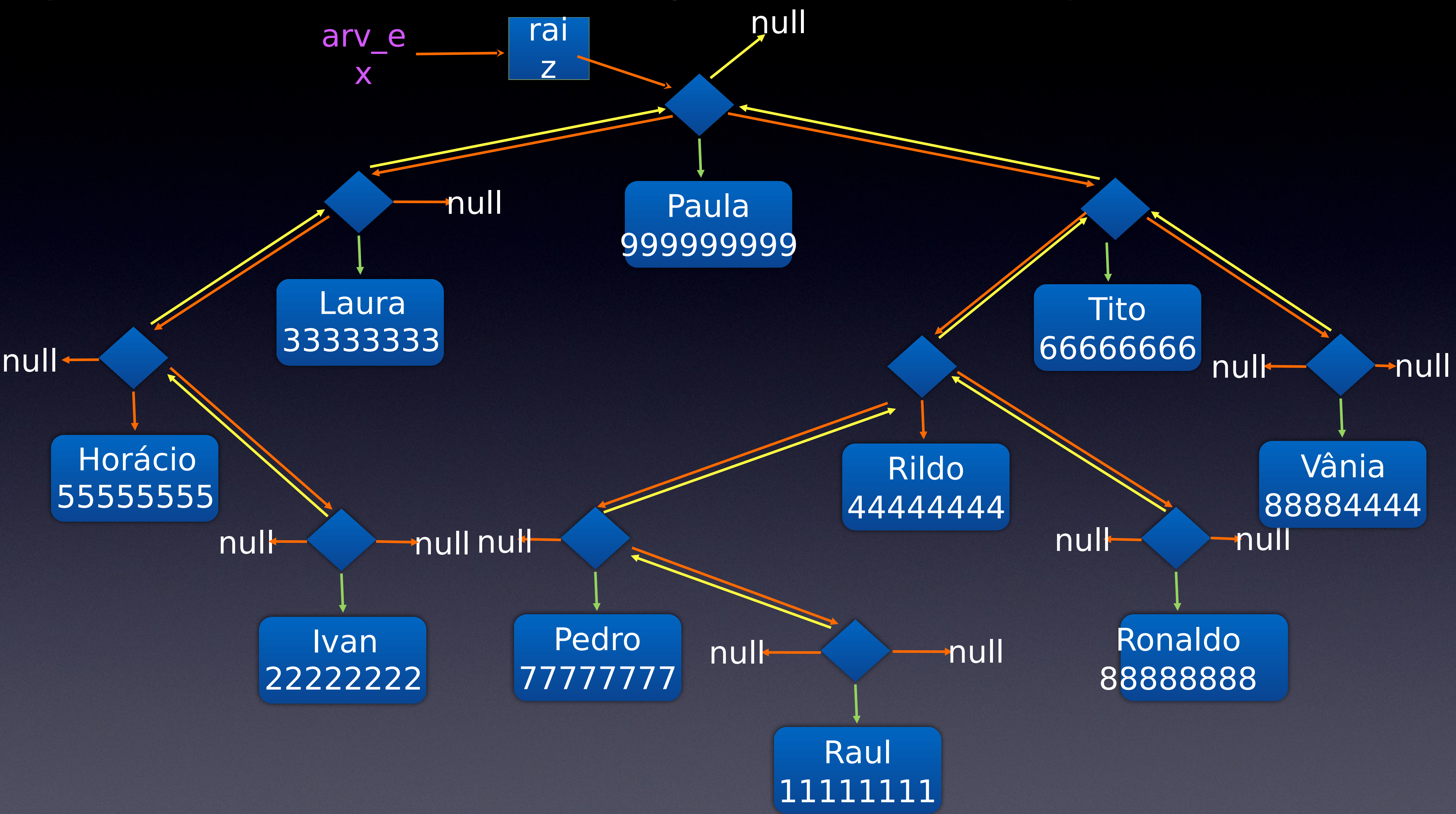


```
public void visitaEmPosOrdem(StringBuffer aux)
```



van - Horácio - Laura - Raul - Pedro - Ronaldo - Rildo - Vânia - Tito - Paula

public void testaIntegridade(StringBuffer aux)




```
public void testaIntegridade(StringBuffer aux) {  
    // se a árvore estiver vazia não faz as chamadas recursivas  
    if(vazia()) { aux.append("Árvore vazia!"); }  
  
    // chamar método recursivo  
    else {  
        testaIntegridade(aux, raiz);  
        aux.append("Árvore provavelmente sem erros.\nVeja mensagens de erro a  
anteriores.\n");  
    }  
}
```



```
private void testaIntegridade(StringBuffer aux, No obj) {
    if(obj == null) { return; }
    if(obj.fe != null){
        if(obj.dados.getNome().compareTo(obj.fe.dados.getNome()) < 0) {
            aux.append("Erro!!! Pai menor que filho da esquerda.\n");
            aux.append("Pai --> " + obj.dados.getNome()+ "\n");
            aux.append("Filho da esquerda --> " + obj.fe.dados.getNome() + "\n");
        }
    }
    if(obj.fd != null){
        if(obj.dados.getNome().compareTo(obj.fd.dados.getNome()) > 0) {
            aux.append("Erro!!! Pai maior que filho da direita.\n");
            aux.append("Pai --> " + obj.dados.getNome()+ "\n");
            aux.append("Filho da direita --> " + obj.fd.dados.getNome() + "\n");
        }
    }
    testaIntegridade(aux, obj.fe);
    testaIntegridade(aux, obj.fd);
}
```


Lista de exercícios 02 - Árvores Binárias de Pesquisa.

1. Qual é a principal propriedade de uma Árvore binária de pesquisa (ABP)?
2. Escreva um método em Java para determinar o número de nós em uma árvore binária de pesquisa.
3. Escreva um método que conta o número de folhas de uma Árvores binária de pesquisa (ABP).
4. Duas Árvores binárias de pesquisa(ABP) são IGUAIS (ou similares) se são ambas vazias ou então se armazenam valores iguais em suas raízes, suas sub-árvores esquerdas são iguais, e suas sub-árvores direitas são iguais. Implemente um método que verifica se duas a árvores são iguais (ou similares).
5. Uma ABP é estritamente binária se todos os nós da árvore tem 2 filhos. Implemente um método que verifica se uma ABP é estritamente binária.
6. Implemente um método para testar se uma a Árvore binária é uma ABP.
7. Projete uma estrutura do tipo Árvore Binaria de Pesquisa para armazenar produtos de uma loja de conveniência. A árvore será ordenada pelo número de matrícula dos produtos, que serão guardados na estrutura composta matrícula, preço e descrição. Defina uma estrutura para abrigar um nó da árvore e defina-a como tipo No. Esta estrutura deve conter uma referência ao produto e referências aos nós filhos e ao pai.

Lista de exercícios 02 - Árvores Binárias de Pesquisa.

8. Considere uma árvore binária de pesquisa (ABP), inicialmente vazia, a qual armazena caracteres. Faça a inserção de todas as letras do seu nome completo na mesma sequência da escrita, desprezando os espaços em branco. Considere, também, todas as letras maiúsculas, sem acentuação e a ordenação da árvore de acordo com a ordem alfabética, onde a letra A é o menor valor e a letra Z é o maior.

Exemplo: Michel Miguel Elias Temer Lulia = MICHELMIGUELELIAS TEMER LULIA; inserir a letra M, em seguida a letra I, depois a letra C e assim por diante até a inserção da última letra que, no exemplo, é a letra A.

Obs: Letras repetidas devem ser inseridas à direita.

Alfabeto: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z. Em seguida, resolva:

- Desenhe a árvore.
- Qual é a altura da árvore?
- Dê o resultado de um percurso pós-fixado.
- Quem é o sucessor do nó de maior altura (caso existam dois ou mais, considere o menor deles) na árvore?
- Quem é o antecessor do quarto nó inserido na árvore?
- Indique (desenhando) o resultado da retirada do nó raiz da árvore.
- Indique (desenhando) a retirada do penúltimo nó.
- Indique (desenhando) a retirada do 7º nó inserido na ABP.

Lista de exercícios 02 - Árvores Binárias de Pesquisa.

9. Escreva uma função que imprima os valores armazenados em uma ABP com recuos de margem proporcionais à profundidade do nó na árvore. Imprima '-' para representar null.

Veja o exemplo abaixo:

Seja a ABP a seguir.

```
      555
      ^
     /\
    333 888
    /\  \
   111 444 999
```

Impresso.

```
555
 333
 111
 -
 -
444
 -
 -
888
 -
999
 -
 -
```