arvores

November 25, 2021

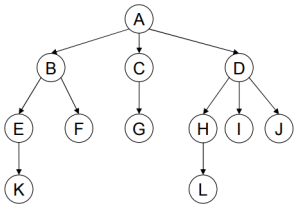
1 Árvores

Estrura recursiva:

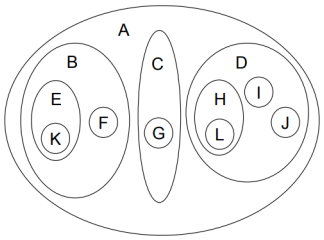
- Formada por subárvores
- Conjunto finito de um ou mais nós
- Nó raiz

1.1 Representações:

1.1.1 Grafos



1.1.2 Conjuntos Aninhados



1.1.3 Parênteses Aninhados

((A (B (E (K) (F)) C (G)
D (H (L) (I) (J))))

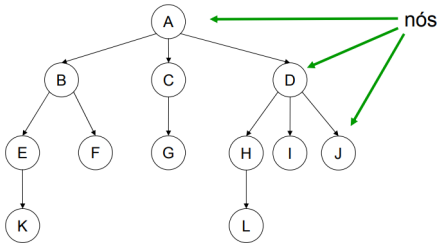
1.1.4 Paragrafação

```
A
  B
    E
      K
    F
  C
    G
  D
    H
      L
    I
    J
```

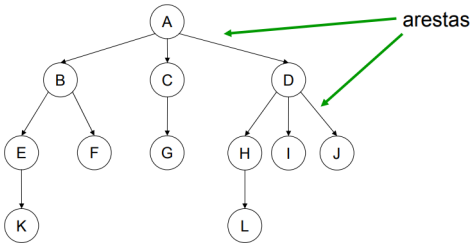
```
A
..B
....E
......K
....F
..C
....G
..D
....H
......L
....I
....J
```

1.2 Estrututra:

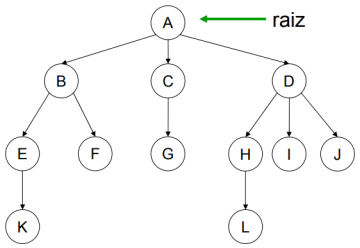
1.2.1 Nós:



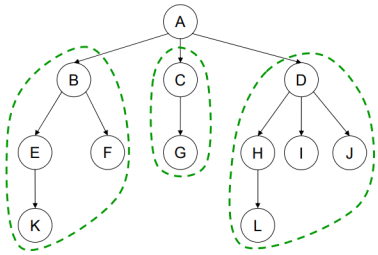
1.2.2 Arestas:



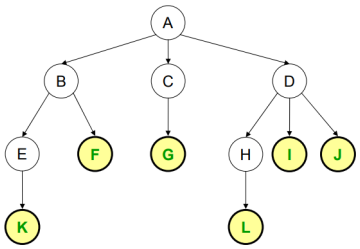
1.2.3 Raiz:



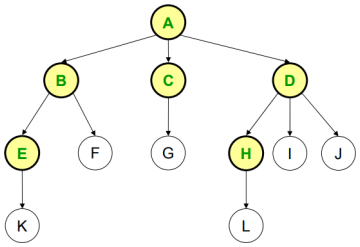
1.2.4 Subárvores



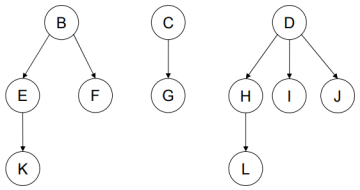
1.2.5 Folha



1.2.6 Não-folha



1.2.7 Floresta



1.3 Grau de um Nó

Nó

Grau

A

3

B

2

C

1

D

3

E

1

F

0

G

0

H

1

I

0

J

0

K

0

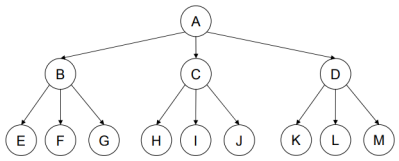
L

0

1.4 Grau de uma árvore

- Grau máximo atingido pelos nós de uma árvore

1.5 Árvore Completa



1.6 Pai

Nó Pai

Nós Filhos

A

B, C, D

B

E, F

C

G

D

H, I, J

E

K

F

-

G

-

H

L

I

-

J

-

K

-

L

-

1.7 Irmão

Irmãos

B, C, D

E, F

H, I, J

1.8 Avô e Demais Parentes

Nós

Avô

E, F, G, H, I ,J

A

K

B

L

D

Nós

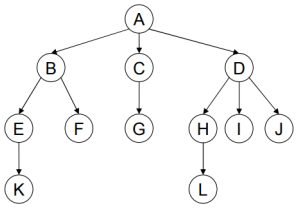
Bisavô

K, L

A

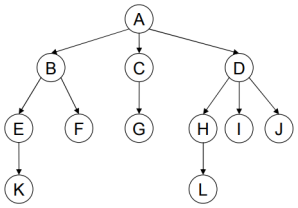
1.9 Caminho

- A, D, H, L é um caminho entre A e L com comprimento de 3



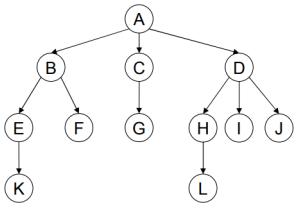
1.10 Antecessores

- Antecessores ou antepassados
- Todos os nós no caminho entre a raiz e o respectivo nó
- No exemplo são antecessores de L: A, D e H



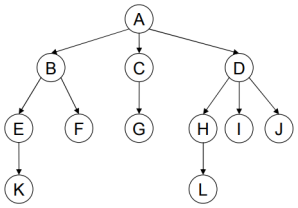
1.11 Nível

- Raiz: podemos considerar nível 0 ou nível 1
- Nível 0: A
- Nível 1: B, C e D
- Nível 2: E, F, G, H, I, J
- Nível 3: K e L



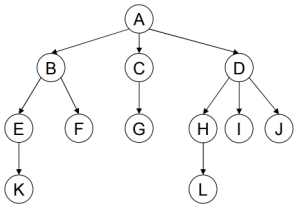
1.12 Altura de um Nó

- Número de arestas no maior caminho até um de seus descendentes
- Folhas tem altura 0
- No exemplo:
  - Altura 0: K, F, G, L, I e J
  - Altura 1: E, C e H
  - Altura 2: B e D
  - Altura 3: A



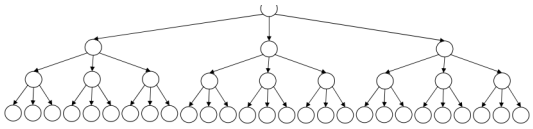
1.13 Altura de uma árvore

- Altura ou profundidade
- Altura máxima encontrada na árvore
- No exemplo: 3



1.14 Número máximo de nós

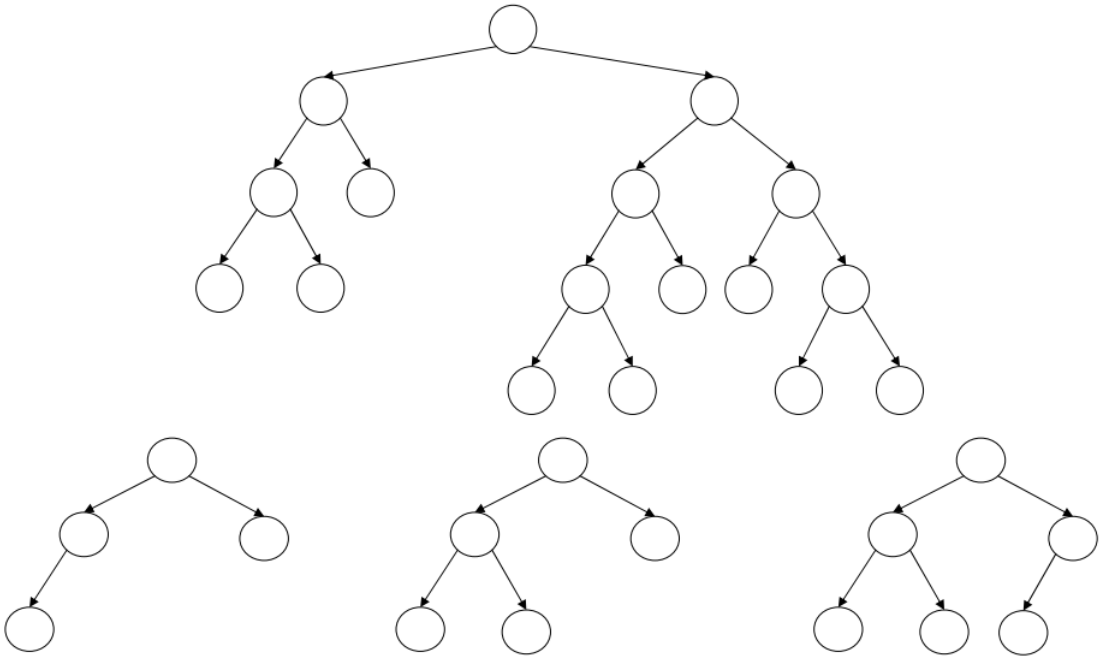
- Para uma árvore de grau d
- Quando todos os nós possuírem d subárvores, exceto suas folhas
- Exemplo: d = 3 (grau) e h = 3 (altura)
  - Nível 0: 1 nó
  - Nível 1: 3 nós
  - Nível 2: 9 nós
  - Nível 3: 27 nós
    - Total nós: 40



1.15 Árvore balanceada

Para cada nó, a altura de suas subárvores diferem, no máximo, em uma unidade

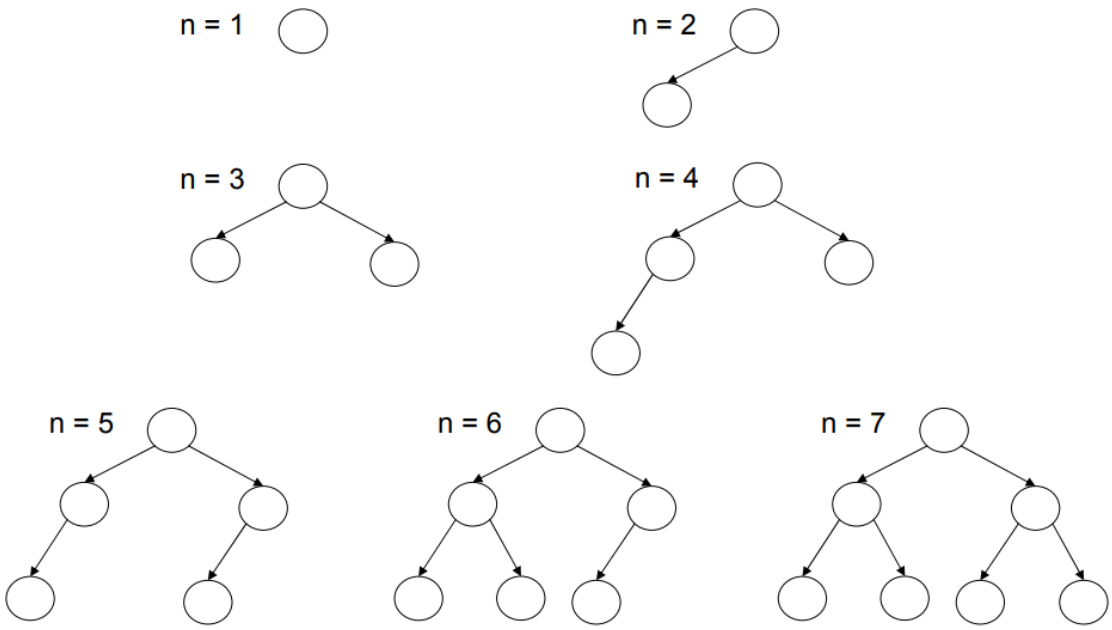
1.15.1 Árvores Balanceadas de Grau 2



1.16 Árvore Perfeitamente Balanceada

Para cada nó, os números de nós em suas subárvores diferem, no máximo, de uma unidade.

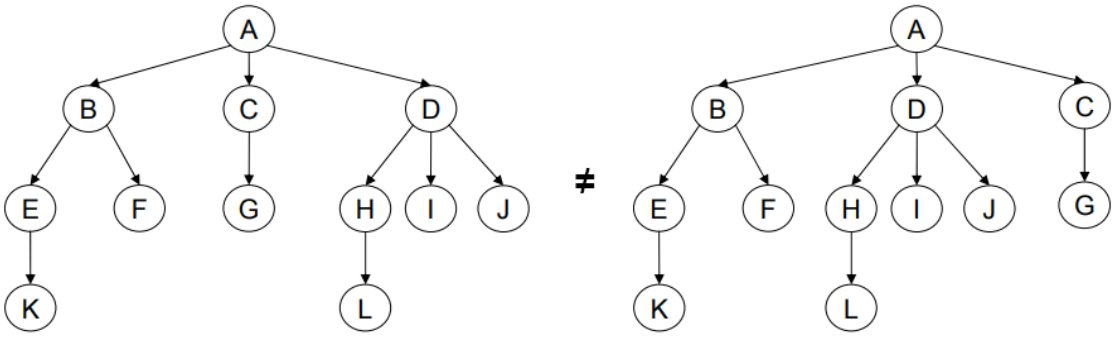
1.16.1 Árvores Perfeitamente Balanceadas de Grau 2



Todas as árvores pefeitamente balanceadas são árvores balanceadas.

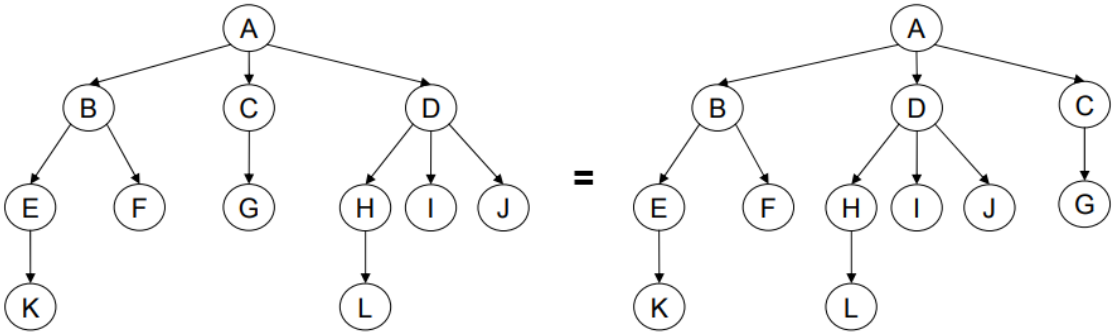
1.17 Árvore ordenada

Conjunto de subárvores ordenado. A ordenação é da esquerda para a direita.



1.18 Árvore Orientada

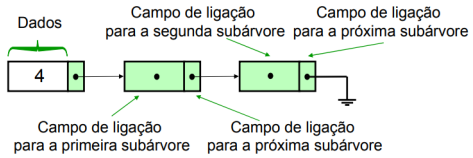
Apenas a orientação relativa dos nós é considerada (e não sua ordem).



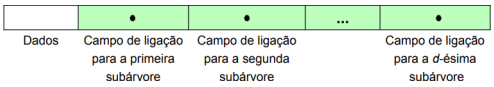
Alguns autores não distinguem árvores ordenadas de orientadas.

2 Implementação de Árvores

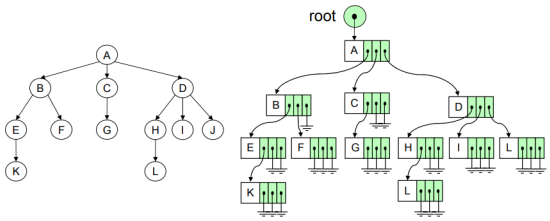
- Podem ser implementadas utilizando listas encadeadas



- Podem ser implementadas utilizando apenas estruturas quando se tem o número máximo de filhos pré-estabelecido.



Exemplo:



3 Implementação Árvore Binária

- Árvores ordenadas (os filhos de cada nó são ordenados) de grau 2
- Vazia ou possui 3 componentes:
  - raiz
  - subárvore esquerda
  - subárvore direita
- Suas subárvores devem ser binárias também

[1]:
```
#include <iostream>
#include <string>

using namespace std;
```

3.1 Definição da Classe

[2]:
```
class ArvoreBinaria {
    private:
        struct elemento {
            int valor;
            elemento *elementoEsquerda;
            elemento *elementoDireita;
        };
        typedef elemento *PonteiroElemento;
        PonteiroElemento raiz;
    public:
        ArvoreBinaria();
        void inserir(int x);
        void remover(int x);
        bool pesquisar(int x);
};
```

Inserção e remoção dependem do tipo de árvore binária a ser implementada.

Por hora, podemos definir alguns métodos genéricos que se aplicam a outros tipos de árvores.

[3]:
```
class ArvoreBinaria {
    private:
        struct elemento {
            int valor;
            elemento *elementoEsquerda;
            elemento *elementoDireita;
        };
        typedef elemento *PonteiroElemento;
        PonteiroElemento raiz;
        // outras operações: métodos auxiliares
        int totalElementos(PonteiroElemento &e);
        int totalElementos2(PonteiroElemento &e);
        int totalFolhas(PonteiroElemento &e);
        int totalFolhas2(PonteiroElemento &e);
        int altura(PonteiroElemento &e);
        void listarPreOrdem(PonteiroElemento &e);
    public:
        ArvoreBinaria();
        bool vazia();
        bool cheia();
```

```
        void inserir(int x);
        void remover(int x);
        bool pesquisar(int x);
        // outras operações
        int totalElementos();
        int totalFolhas();
        int altura();
        void listarPreOrdem();
};
```

3.2 Construtor

[4]:
```
ArvoreBinaria::ArvoreBinaria() {
    raiz = nullptr;
}
```

[5]:
```
ArvoreBinaria minhaArvore;
```

3.3 Vazia

[6]:
```
bool ArvoreBinaria::vazia() {
    return raiz == nullptr;
}
```

if (minhaArvore.vazia()) { cout « “Está vazia!”; }

3.4 Cheia

[7]:
```
bool ArvoreBinaria::cheia() {
    return false;
}
```

[8]:
```
if (! minhaArvore.cheia()) {
    cout << "Não está cheia!";
}
```

```
Não está cheia!
```

3.5 Inserir

Inserir “emprestado” da árvore de busca binária.

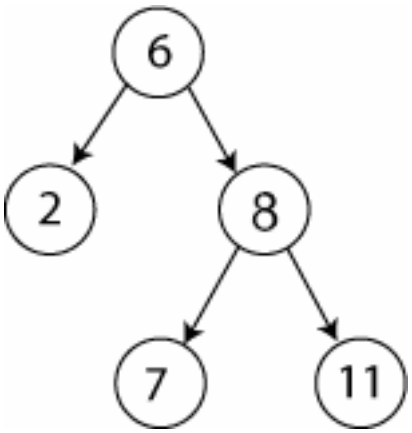
A intenção é criarmos uma árvore para testar os próximos metódos.

[9]:
```
void ArvoreBinaria::inserir(int x) {
    PonteiroElemento p=NULL, q=raiz, r;
    while (q != NULL) {
        p = q;
```

```
        if (x < q->valor) {
            q = q->elementoEsquerda;
        } else {
            q = q->elementoDireita;
        }
    }
    r = new elemento;
    r->valor = x;
    r->elementoEsquerda = NULL;
    r->elementoDireita = NULL;

    if (p == NULL) {
        raiz = r;
    } else {
        if (x < p->valor) {
            p->elementoEsquerda = r;
        } else {
            p->elementoDireita = r;
        }
    }
}
```

[10]:
```
minhaArvore.inserir(6);
minhaArvore.inserir(2);
minhaArvore.inserir(8);
minhaArvore.inserir(11);
minhaArvore.inserir(7);
```



3.6 Número de Nós

[11]:
```
// Método Público
int ArvoreBinaria::totalElementos() {
    return totalElementos(raiz);
}
```

[12]:
```cpp
// Método Privado
int ArvoreBinaria::totalElementos(PonteiroElemento &e) {
    if (e == NULL) {
        return 0;
    } else {
        int E, D;
        E = totalElementos(e->elementoEsquerda);
        D = totalElementos(e->elementoDireita);
        return 1 + E + D;
    }
}
```

[13]:
```cpp
cout << "Total de elementos: " << minhaArvore.totalElementos();
```

```
Total de elementos: 5
```

```cpp
minhaArvore.totalElementos();
    totalElementos(raiz); // 6
    E = totalElementos(e->elementoEsquerda); // 2
        E = totalElementos(e->elementoEsquerda); // NULL
        // E = 0
        D = totalElementos(e->elementoDireita); // NULL
        // D = 0
        return 1 + E + D;
    // E = 1
    D = totalElementos(e->elementoDireita); // 8
        E = totalElementos(e->elementoEsquerda); // 7
            E = totalElementos(e->elementoEsquerda); // NULL
            // E = 0
            D = totalElementos(e->elementoDireita); // NULL
            // D = 0
            return 1 + E + D;
        // E = 1
        D = totalElementos(e->elementoDireita); // 11
            E = totalElementos(e->elementoEsquerda); // NULL
            // E = 0
            D = totalElementos(e->elementoDireita); // NULL
            // D = 0
            return 1 + E + D;
        // D = 1
        return 1 + E + D;
    // D = 3
    return 1 + E + D;
// final: 5
```

[14]:
```cpp
// Método Privado Otimizado
int ArvoreBinaria::totalElementos2(PonteiroElemento &e) {
    if (e == NULL) return 0;
```

```
    return 1 + totalElementos2(e->elementoEsquerda) +␣
 ↪totalElementos2(e->elementoDireita);
}
```

3.7 Número de Folhas

[15]:
```
// Método Público
int ArvoreBinaria::totalFolhas() {
    return totalFolhas(raiz);
}
```

[16]:
```
// Método Privado
int ArvoreBinaria::totalFolhas(PonteiroElemento &e) {
    if (e == NULL) {
        return 0;
    } else {
        if (e->elementoEsquerda == NULL and e->elementoDireita == NULL) {
            return 1;
        } else {
            int E, D;
            E = totalFolhas(e->elementoEsquerda);
            D = totalFolhas(e->elementoDireita);
            return E + D;
        }
    }
}
```

[17]:
```
cout << "Total de folhas: " << minhaArvore.totalFolhas();
```

```
Total de folhas: 3
```

```
minhaArvore.totalFolhas();
    totalFolhas(raiz); // 6
    E = totalFolhas(e->elementoEsquerda); // 2
    // E = 1
    D = totalFolhas(e->elementoDireita); // 8
        E = totalFolhas(e->elementoEsquerda); // 7
        // E = 1
        D = totalFolhas(e->elementoDireita); // 11
        // D = 1
        return E + D;
    // D = 2
    return E + D;
// Final: 3
```

[18]:
```
// Método Privado Otimizado
int ArvoreBinaria::totalFolhas2(PonteiroElemento &e) {
```

```
    if (e == NULL) return 0;
    if (e->elementoEsquerda == NULL and e->elementoDireita == NULL) return 1;
    return totalFolhas2(e->elementoEsquerda) + totalFolhas2(e->elementoDireita);
}
```

3.8 Altura

- Árvore vazia: -1
- Folhas: 0

[19]:
```
// Método Público
int ArvoreBinaria::altura() {
    return altura(raiz);
}
```

[20]:
```
// Método Privado
int ArvoreBinaria::altura(PonteiroElemento &e) {
    if (e == NULL) {
        return -1;
    } else {
        int E, D;
        E = altura(e->elementoEsquerda);
        D = altura(e->elementoDireita);
        if (E > D) {
            return E + 1;
        }  else {
            return D + 1;
        }
    }
}
```

[21]:
```
cout << "Altura: " << minhaArvore.altura();
```

```
Altura: 2
```

```
minhaArvore.altura();
    altura(raiz); // 6
    E = altura(e->elementoEsquerda);  // 2
        E = altura(e->elementoEsquerda);  // NULL
        // E = -1;
        D = altura(e->elementoDireita);  // NULL
        // D = -1;
        return D + 1; // 0
    // E = 0
    D = altura(e->elementoDireita);  // 8
        E = altura(e->elementoEsquerda);  // 7
            E = altura(e->elementoEsquerda);  // NULL
            // E = -1;
```

```
            D = altura(e->elementoDireita);  // NULL
            // D = -1;
            return D + 1; // 0
        // E = 0
        D = altura(e->elementoDireita);  // 11
            E = altura(e->elementoEsquerda);  // NULL
            // E = -1;
            D = altura(e->elementoDireita);  // NULL
            // D = -1;
            return D + 1; // 0
        //D = 0
        return D + 1; // 1
    // D = 1
    return D + 1;
// Final: 2
```

3.9 Percurso

3.9.1 Pré-Ordem

raiz - esquerda - direita

[22]:
```
// Método Público
void ArvoreBinaria::listarPreOrdem() {
    listarPreOrdem(raiz);
}
```

[23]:
```
// Método Privado
void ArvoreBinaria::listarPreOrdem(PonteiroElemento &e)
{
    if (e != NULL) {
        cout << e->valor << "\n";
        listarPreOrdem(e->elementoEsquerda);
        listarPreOrdem(e->elementoDireita);
    }
}
```

[24]:
```
minhaArvore.listarPreOrdem();
```

```
6
2
8
7
11
```

```
minhaArvore.listarPreOrdem();
    listarPreOrdem(raiz); // 6
    listarPreOrdem(e->elementoEsquerda); // 2
        listarPreOrdem(e->elementoEsquerda); // NULL
```

```
        listarPreOrdem(e->elementoDireita); // NULL
    listarPreOrdem(e->elementoDireita); // 8
        listarPreOrdem(e->elementoEsquerda); // 7
            listarPreOrdem(e->elementoEsquerda); // NULL
            listarPreOrdem(e->elementoDireita); // NULL
        listarPreOrdem(e->elementoDireita); // 11
            listarPreOrdem(e->elementoEsquerda); // NULL
            listarPreOrdem(e->elementoDireita); // NULL
```

[25]:
```
minhaArvore.listarPreOrdem(); // raiz - esquerda - direita
```

```
6
2
8
7
11
```

3.10 In-Ordem

Esquerda - Raiz - Direita

[2][6][7][8][11]

3.11 Pós-Ordem

Esquerda - Direita - Raiz

[2][7][11][8][6]

2 6 7 8 11