Departamento de Computação

Programação Competitiva unesp

Árvores

Laboratório de Programação Competitiva I

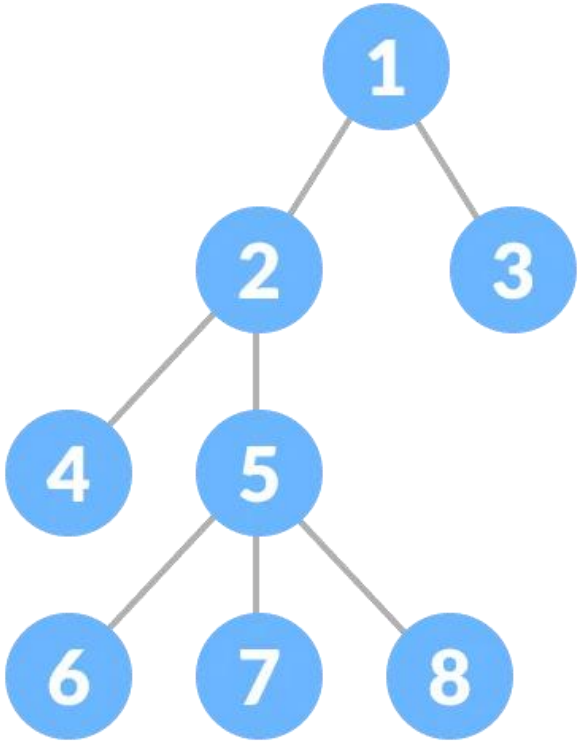
Pedro Henrique Paiola

Rene Pegoraro

Wilson M Yonezawa

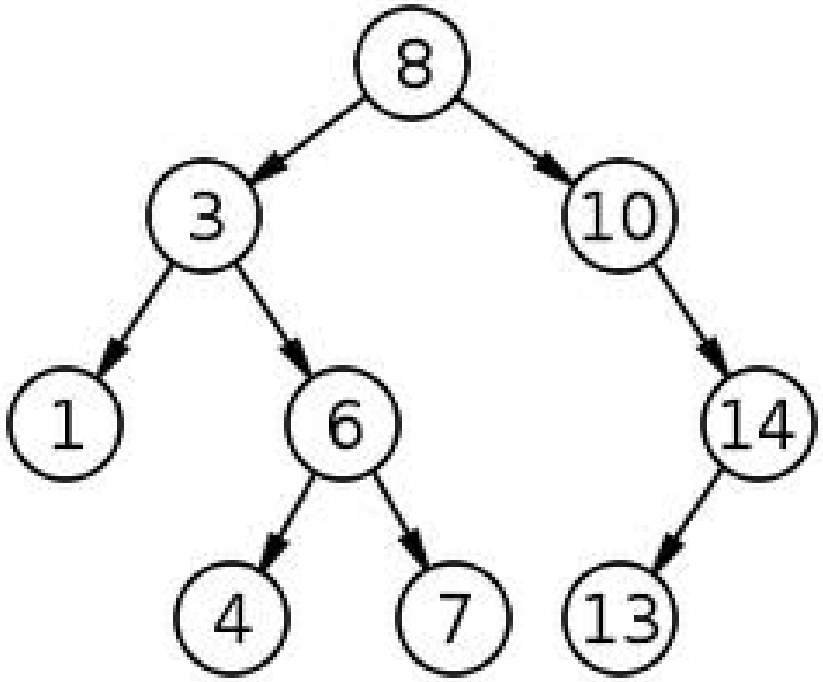
Árvore

- Uma árvore é formada por um conjunto de nós (ou vértices) e arestas em que existe exatamente um caminho conectando qualquer par de nós.
- Uma árvore qualquer possui
  - n nós
  - $n - 1$ arestas



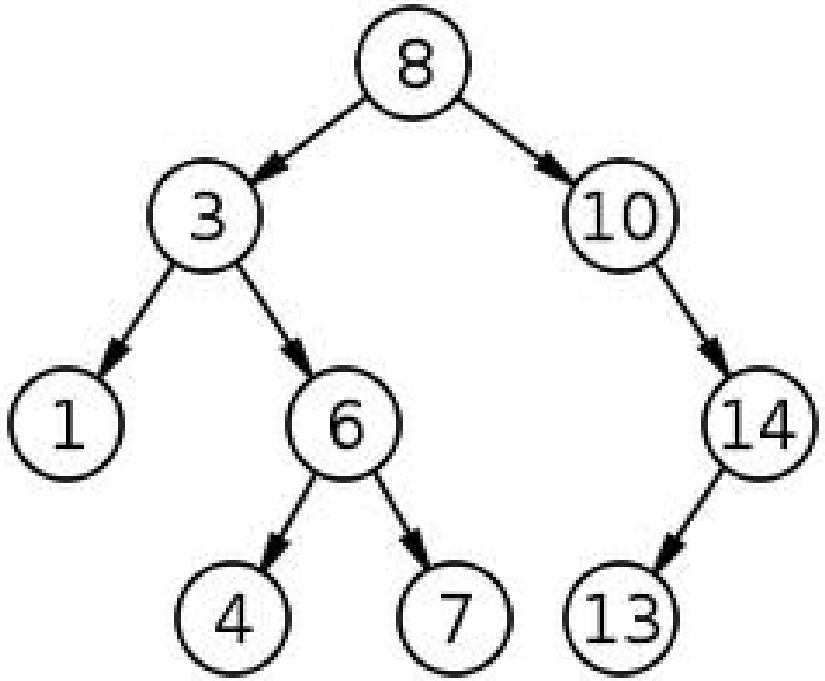
Árvore enraizada

- Uma árvore é dita enraizada quando há um nó especial, denominado **raiz**.
  - Com exceção da raiz, todo nó é ligado por uma aresta a um, e apenas um, nó (o pai)
  - Há um caminho único da raiz a cada nó



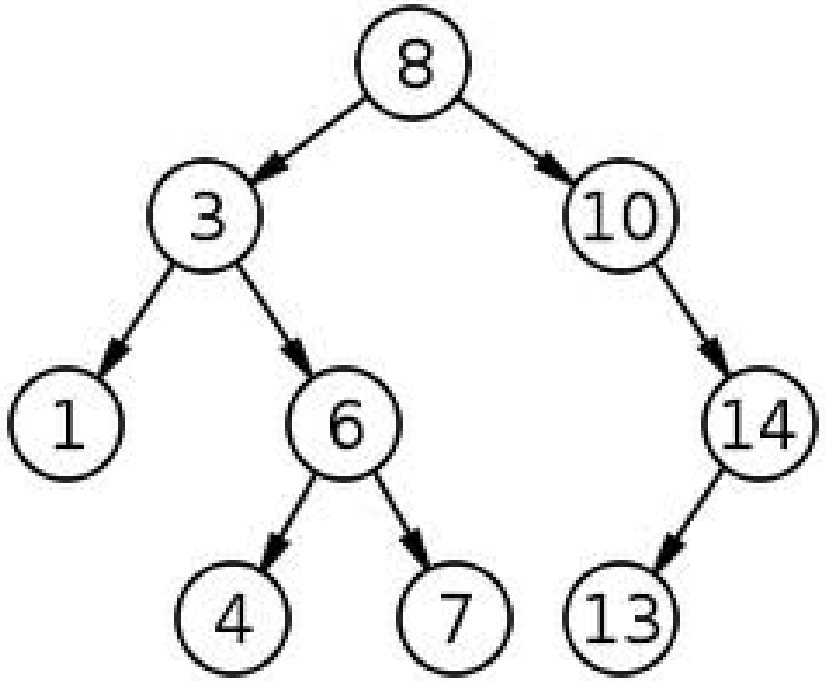
Terminologia

- **Raiz:** primeiro elemento, que dá origem aos demais. O único nó que não possui um antecessor/pai.



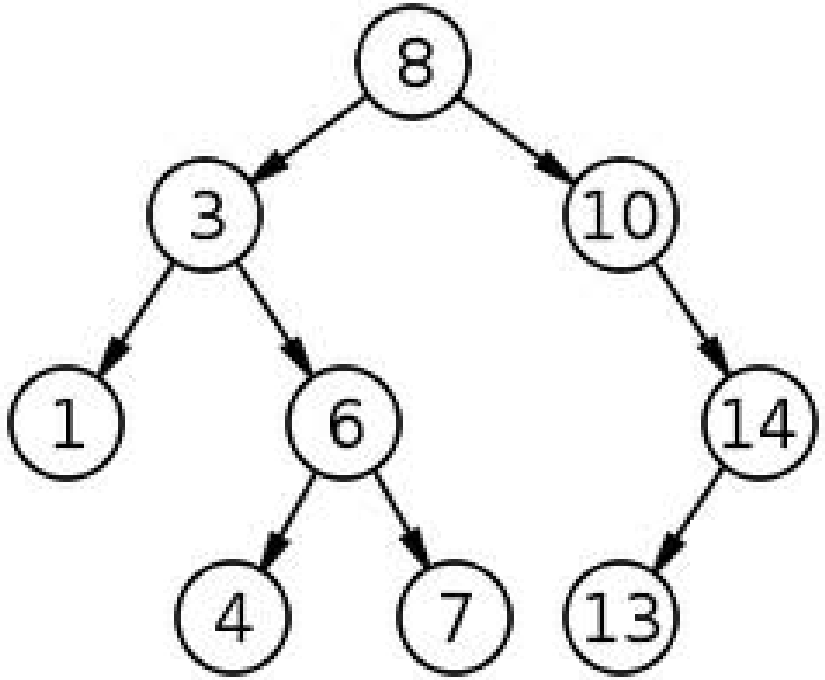
Terminologia

- **Nível de um nó:** distância que um nó tem em relação à raiz.
  - **Distância:** número de arestas entre os nós
- Exemplos
  - $nível(3) = 1$
  - $nível(7) = 3$
  - $nível(8) = 0$



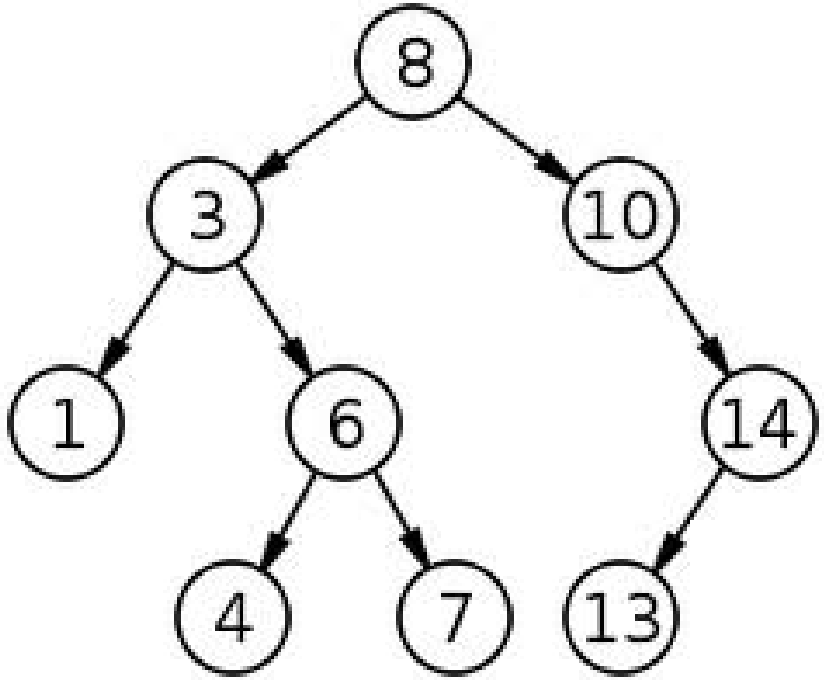
Terminologia

- **Altura de uma árvore:** quantidade de níveis a partir da raiz até o nó mais distante.
- Na árvore abaixo, $altura = 3$



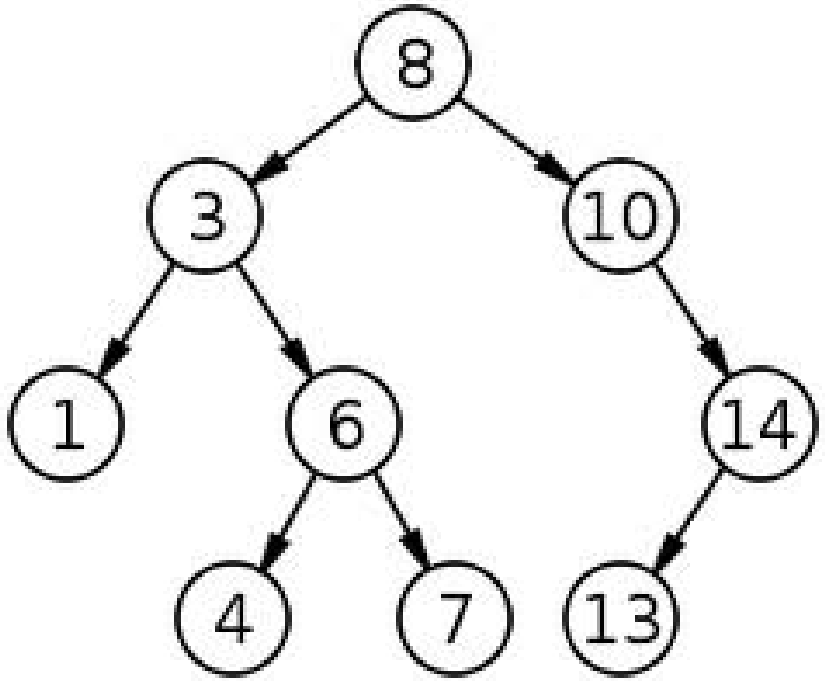
Terminologia

- **Filho:** sucessor de um determinado nó.
- **Pai:** antecessor (único) de um dado nó.



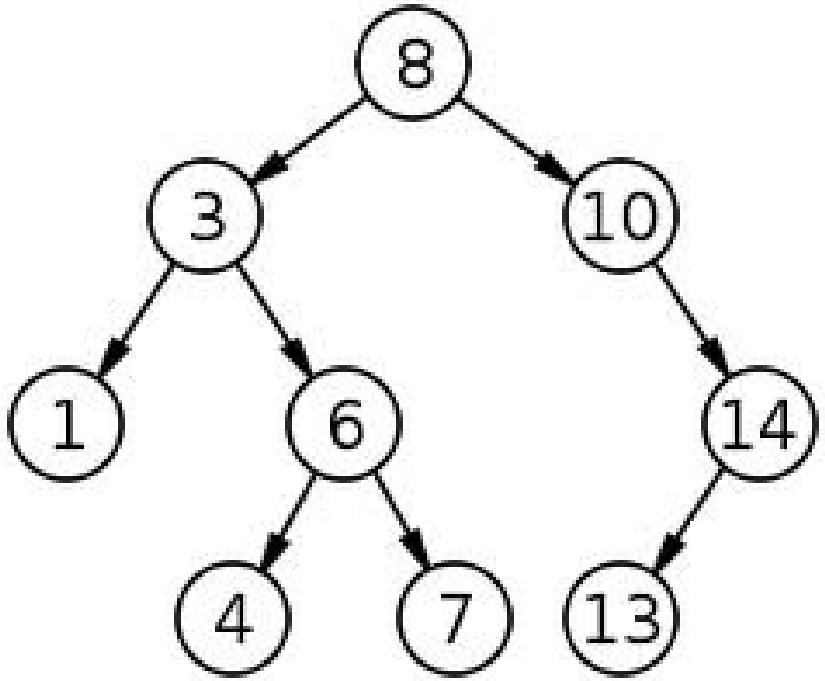
Terminologia

- **Folha:** nó que não possui filhos.



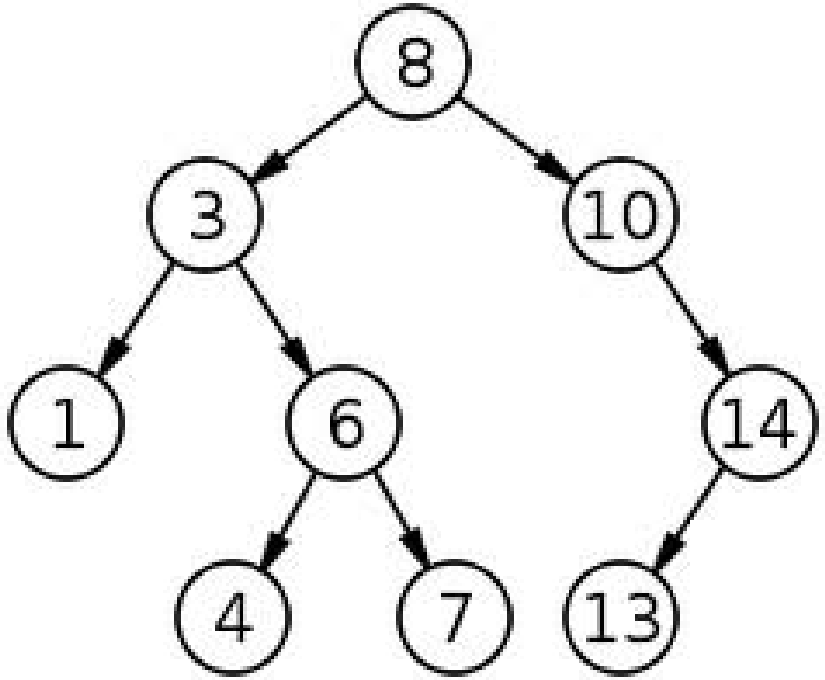
Terminologia

- **Grau de um nó:** número de ramificações, de filhos, de um nó.
- **Exemplo:**
  - $grau(3) = 2$
  - $grau(10) = 1$
  - $grau(1) = 0$



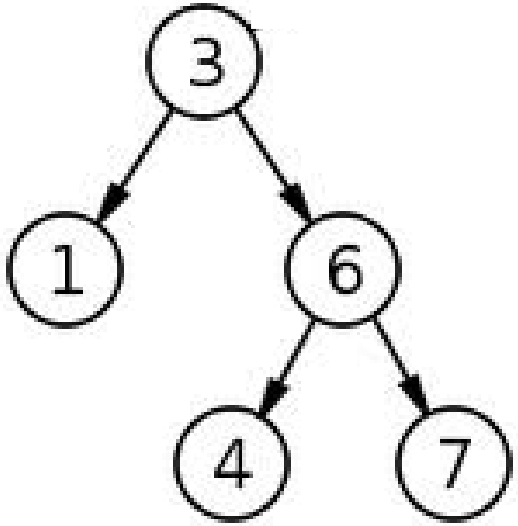
Terminologia

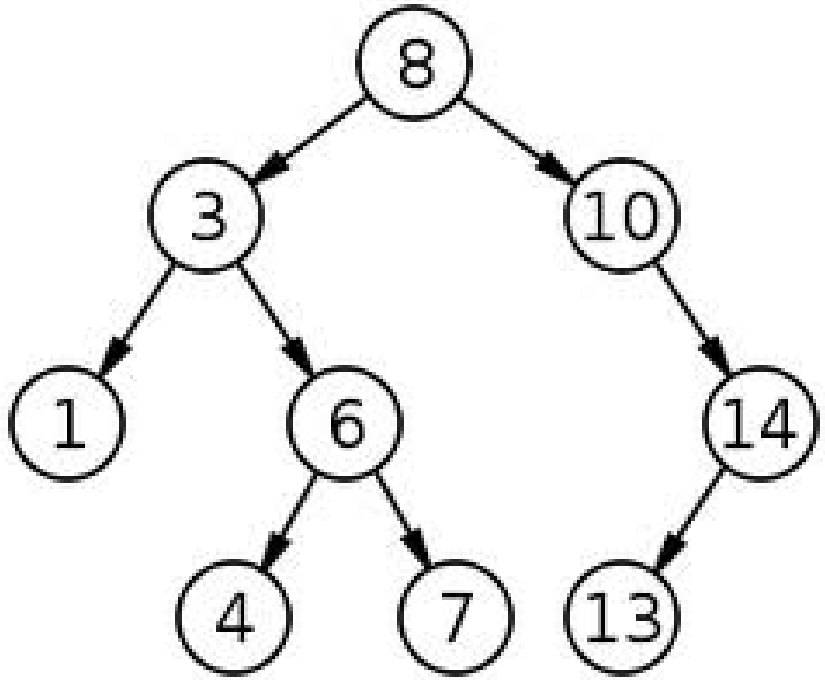
- **Grau de uma árvore:** número máximo de ramificações de um nó da árvore.
  - $grau\ da\ árvore\ = \max\{grau(nó) | \forall\ nó\ da\ árvore\}$



Terminologia

- **Sub-árvore:** uma sub-árvore é qualquer árvore contida na árvore original.
- Exemplo de sub-árvore





Implementações

- Existem várias formas de se implementar uma árvore, tanto com estruturas estáticas quanto dinâmicas.
- Qual a melhor representação depende muito do contexto em que a árvore será aplicada.
- Apresentaremos aqui o esboço de duas implementações:
  1. Versão genérica, semelhante a implementação de grafos. Aplicável na maioria dos casos.
  2. Árvores binárias (com grau 2) de busca utilizando ponteiros
     - Árvores binárias de busca possuem a seguinte propriedade:
       - Para cada nó x, todos os nós da sub-árvore esquerda possuem valor menores que x, e todos os nós da sub-árvore direita possuem valores maiores que x

Implementações

- Versão 1

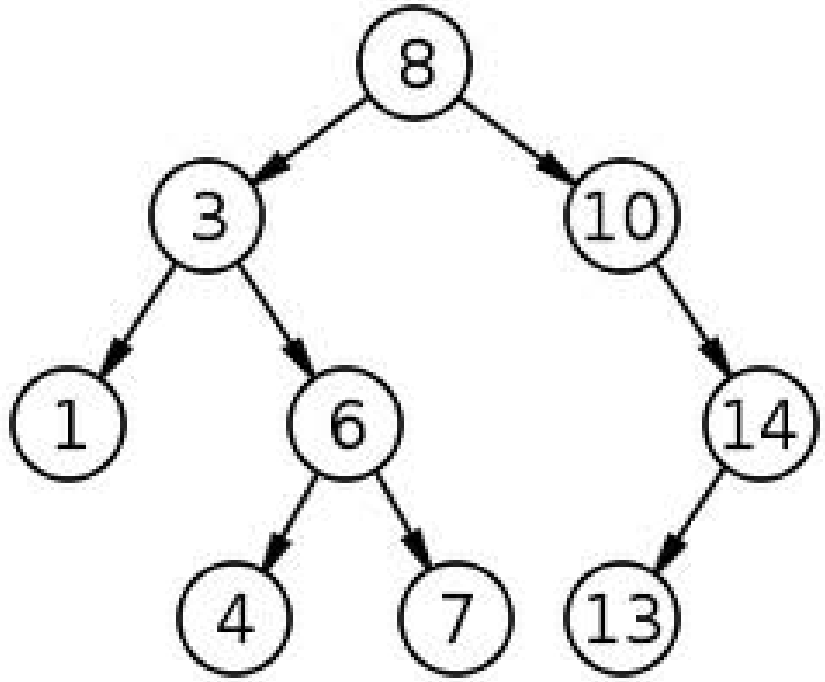
```
vector<int> arvore[QTDE_VERTICES];

int pai[QTDE_VERTICES];  //nem sempre é necessário
//Se for necessário, podem ser armazenadas informações adicionais para
cada nó, em vetores auxiliares

void aresta(int u, int v) //u é pai de v
{
    arvore[u].push_back(v);
    pai[v] = u;
}
```

Implementações

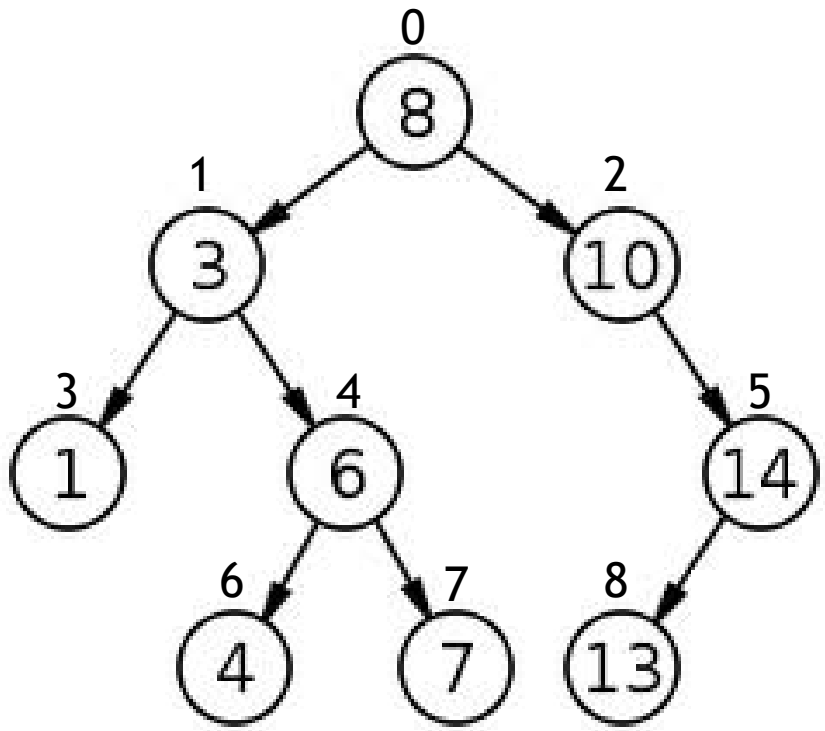
- Exemplo



i	val[i]	arvore[i]
0		
1		
2		
3		
4		
5		
6		
7		
8		

Implementações

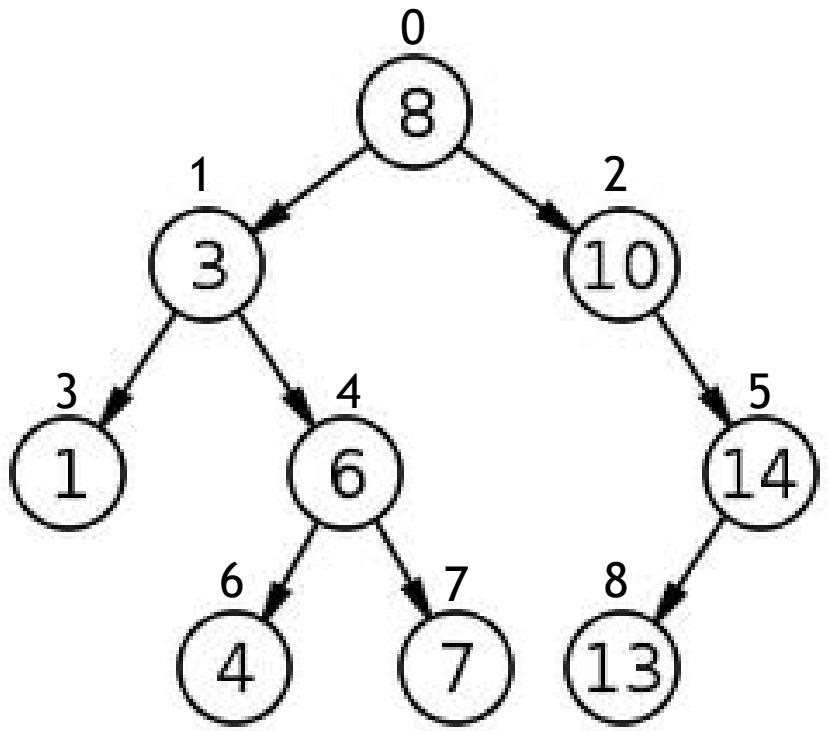
- Exemplo



i	val[i]	arvore[i]
0	8	
1	3	
2	10	
3	1	
4	6	
5	14	
6	4	
7	7	
8	13	

Implementações

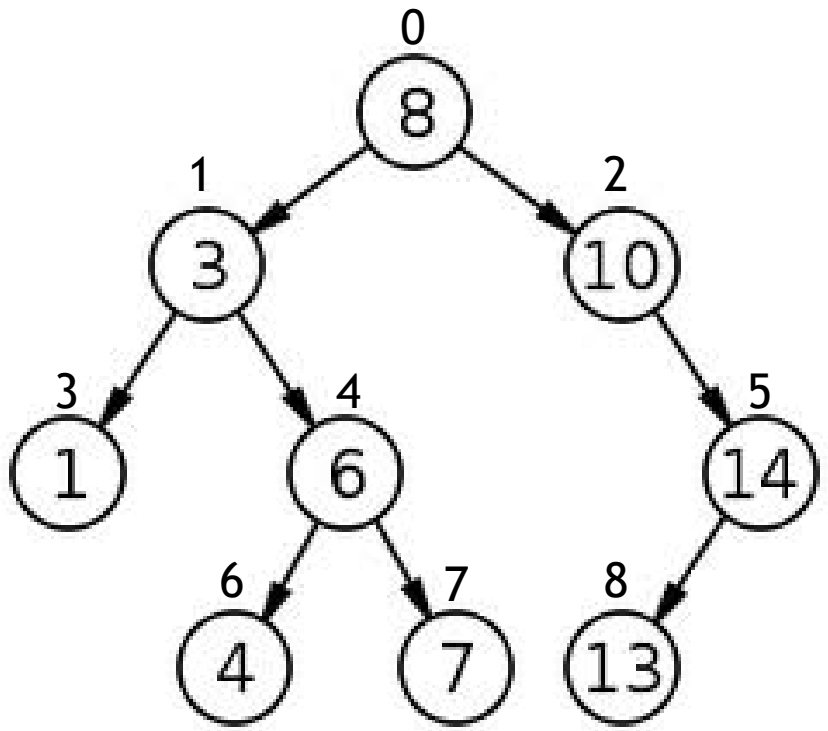
- Exemplo



i	val[i]	arvore[i]
0	8	[1, 2]
1	3	[3, 4]
2	10	[5]
3	1	[]
4	6	[6, 7]
5	14	[8]
6	4	[]
7	7	[]
8	13	[]

Implementações

- Exemplo



i	val[i]	arvore[i]	pai[i]
0	8	[1, 2]	-
1	3	[3, 4]	0
2	10	[5]	0
3	1	[]	1
4	6	[6, 7]	1
5	14	[8]	2
6	4	[]	4
7	7	[]	4
8	13	[]	5

Implementações

- Versão 2

```
struct no{
    int v;
    no *esq;
    no *dir;
};
typedef no* def_arvore;

void cria_no(def_arvore &arvore, int valor){
    arvore = new no();
    arvore->v = valor;
    arvore->dir = arvore->esq = NULL;
}
```

Implementações

- Versão 2

```
void insere(def_arvore &arvore, int valor){
    if (arvore != NULL)
    {
        if (valor > arvore->v)
            insere(arvore->dir, valor);
        else
            insere(arvore->esq, valor);
    }
    else
        cria_no(arvore, valor);
}
```

Implementações

- Exemplo:

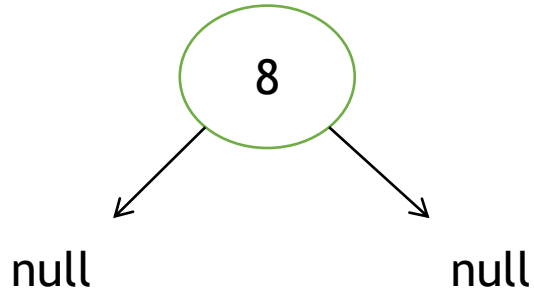
null

```
insere(arvore, 8)
```

Implementações

- Exemplo:

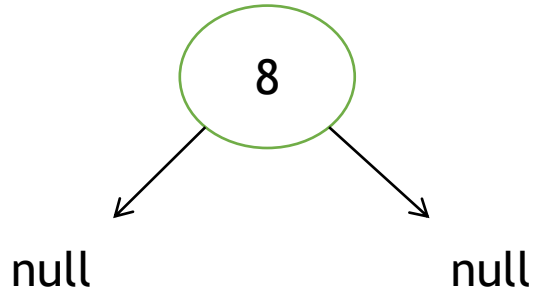
```
insere(arvore, 8)
```



Implementações

- Exemplo:

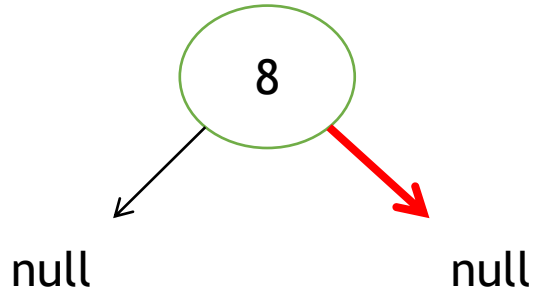
```
insere(arvore, 10)
```



Implementações

- Exemplo:

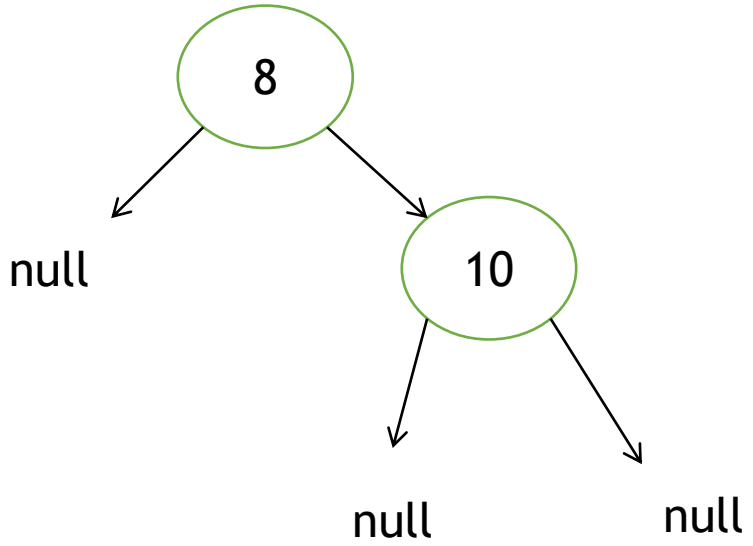
```
insere(arvore, 10)
```



Implementações

- Exemplo:

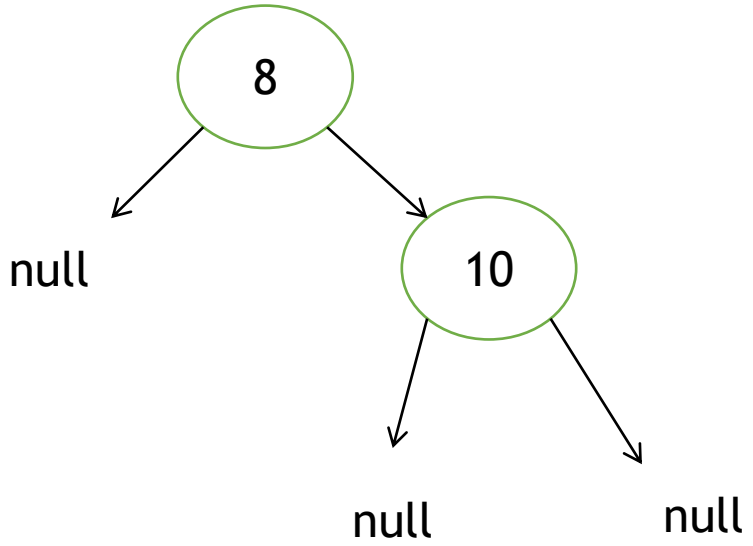
```
insere(arvore, 10)
```



Implementações

- Exemplo:

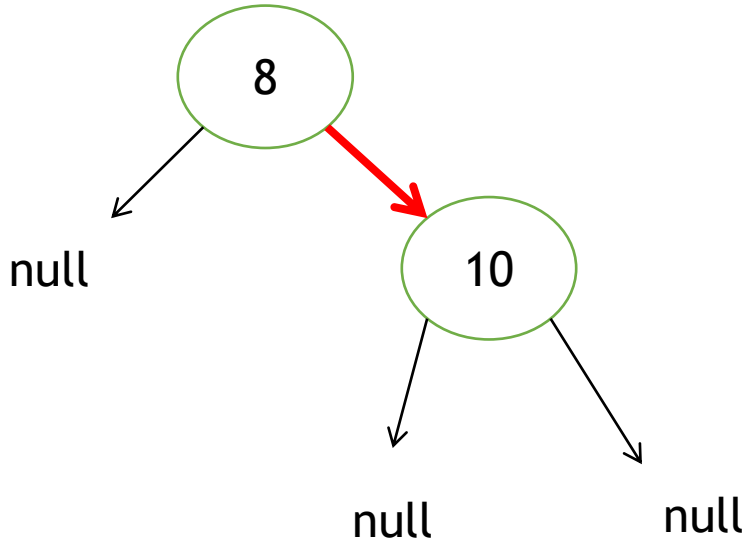
```
insere(arvore, 14)
```



Implementações

- Exemplo:

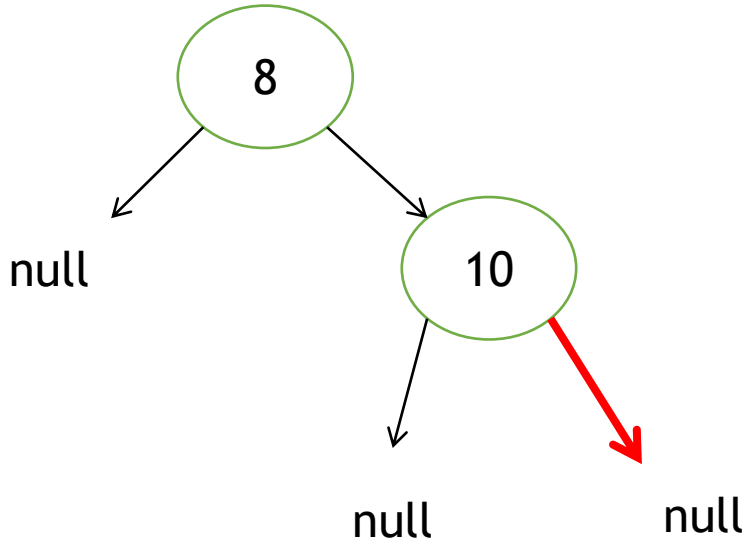
```
insere(arvore, 14)
```



Implementações

- Exemplo:

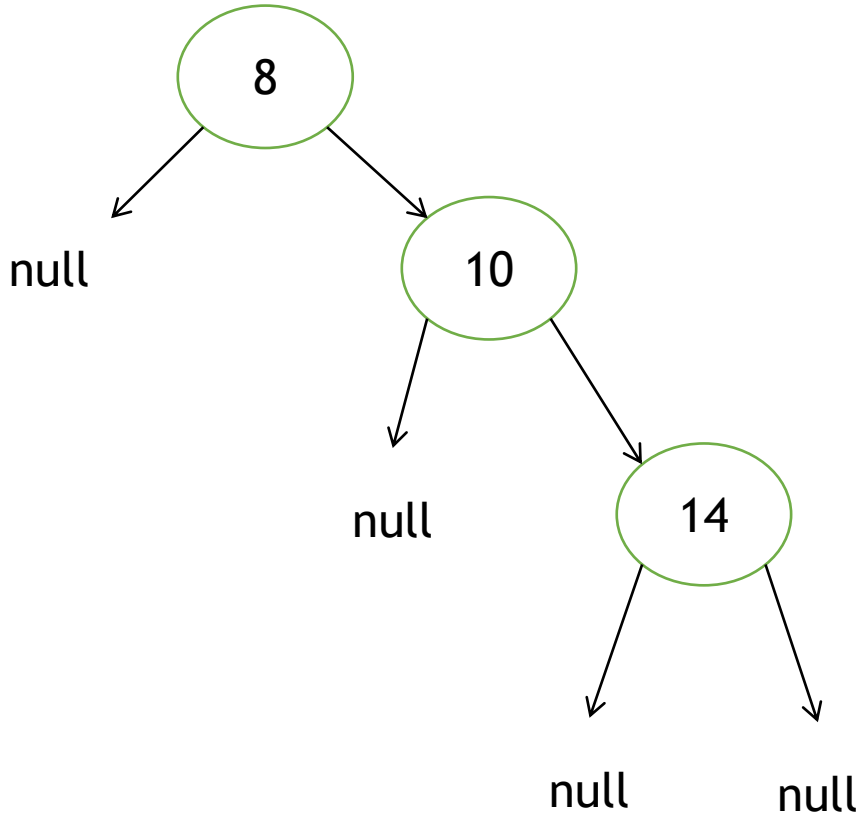
```
insere(arvore, 14)
```



Implementações

- Exemplo:

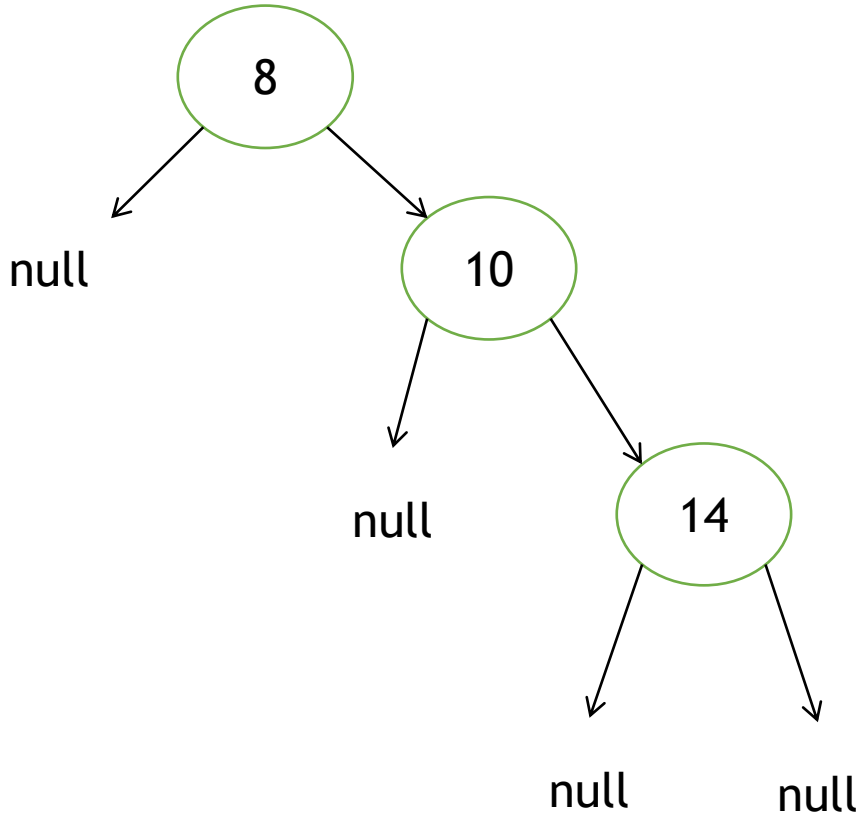
```
insere(arvore, 14)
```



Implementações

- Exemplo:

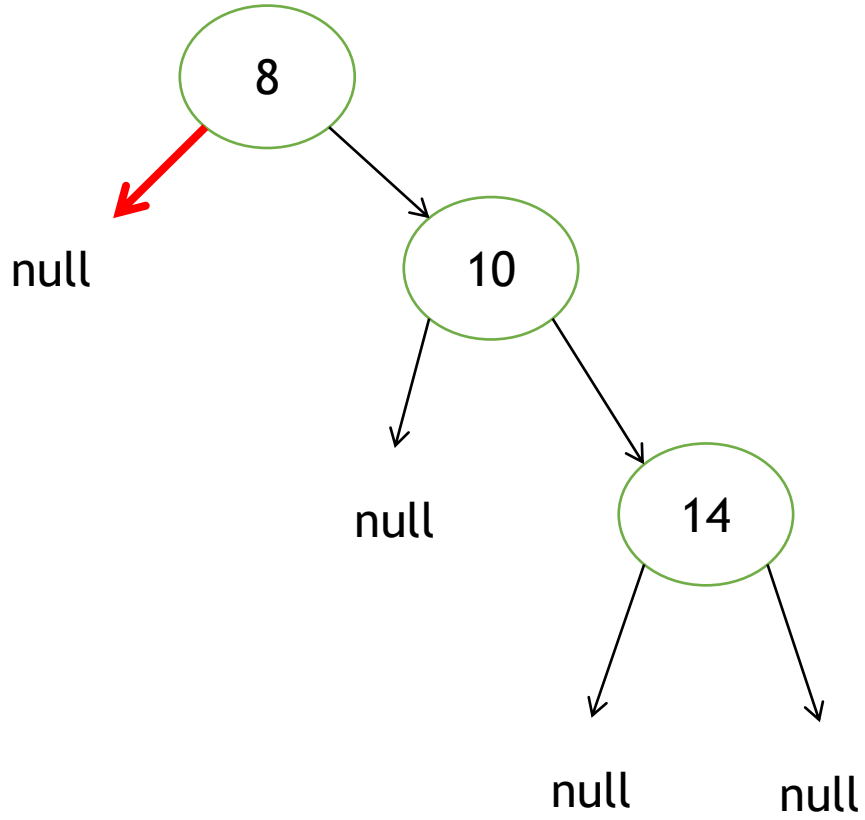
```
insere(arvore, 3)
```



Implementações

- Exemplo:

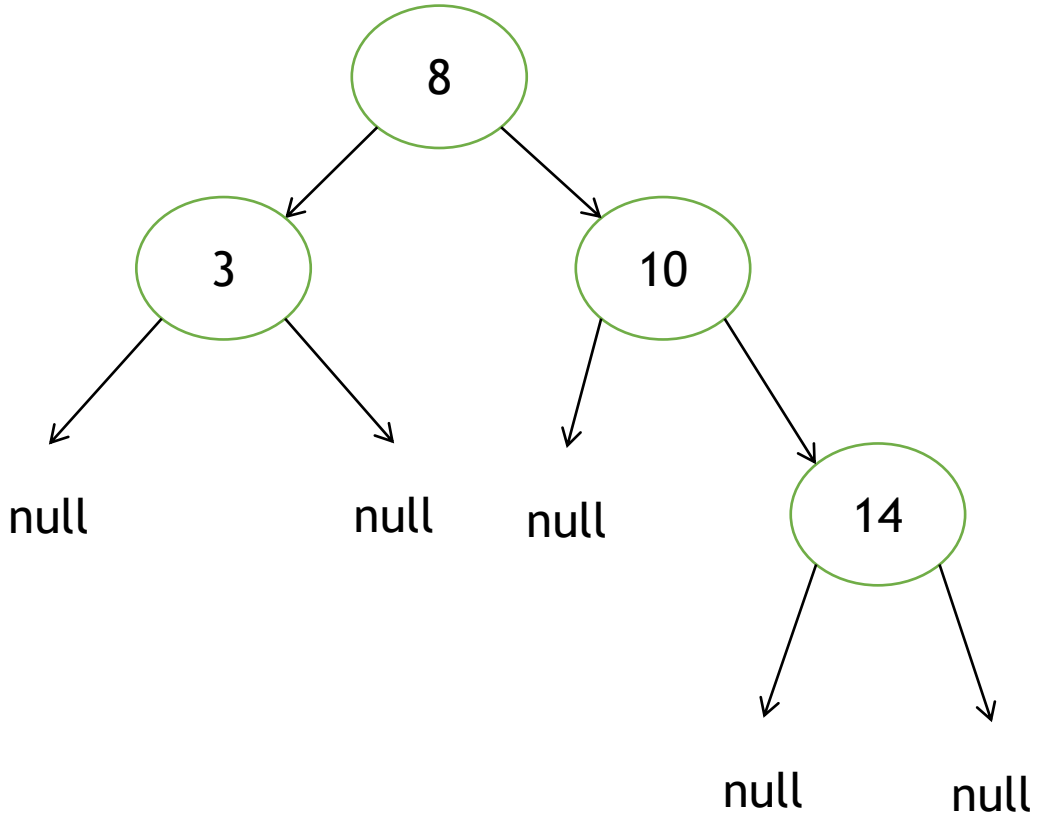
```
insere(arvore, 3)
```



Implementações

- Exemplo:

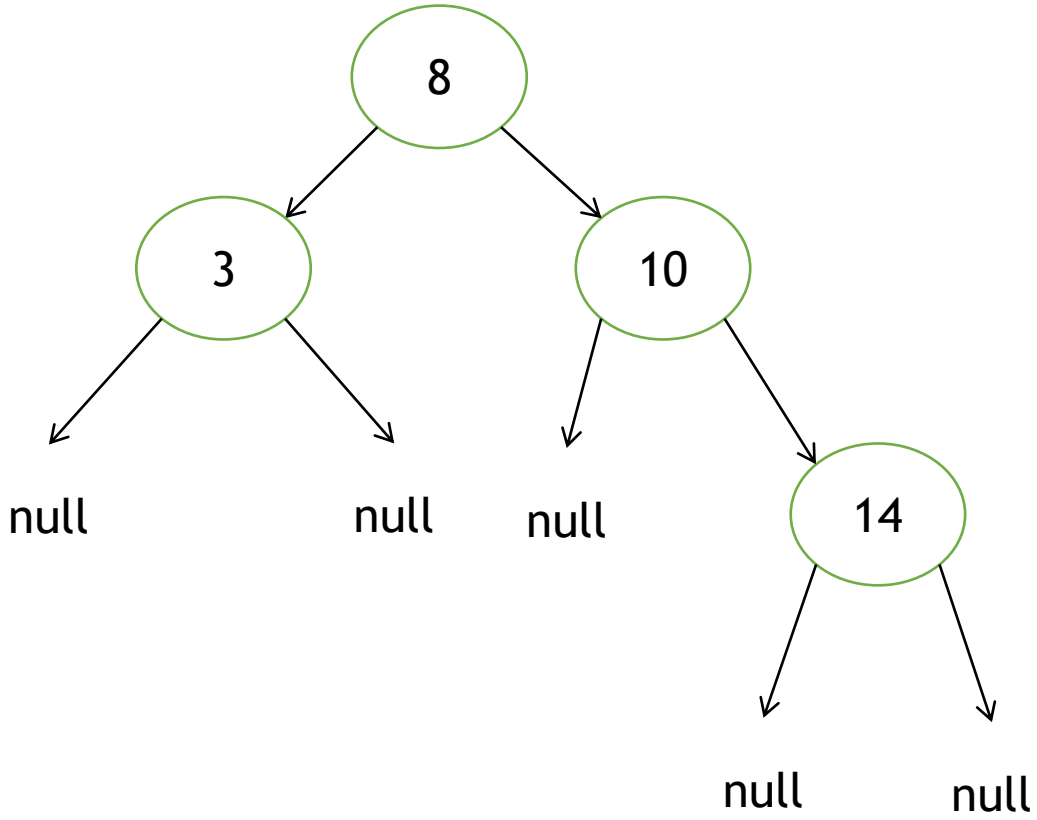
```
insere(arvore, 3)
```



Implementações

- Exemplo:

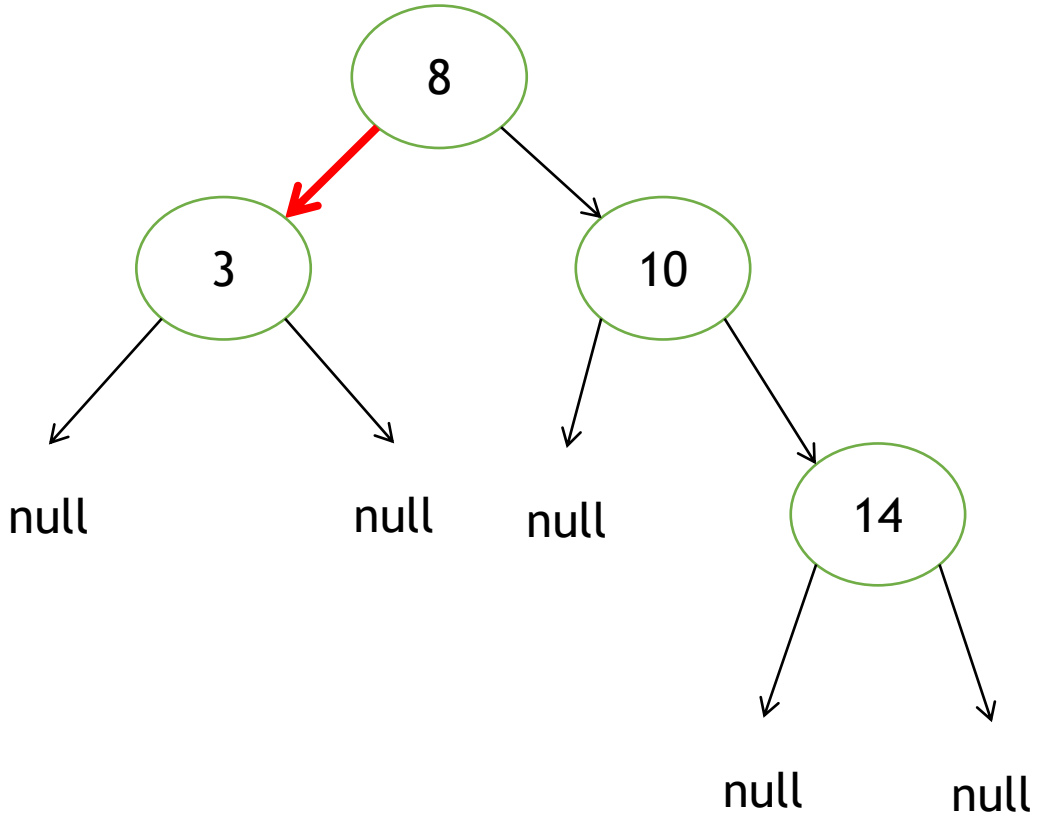
```
insere(arvore, 6)
```



Implementações

- Exemplo:

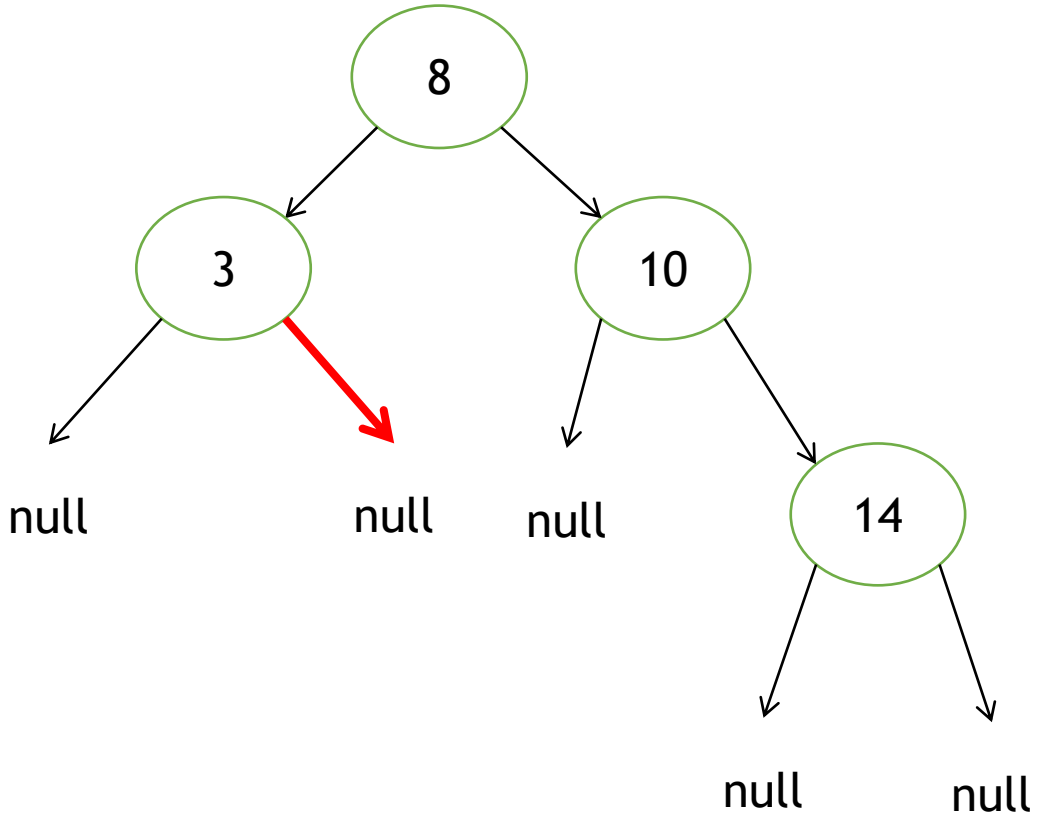
```
insere(arvore, 6)
```



Implementações

- Exemplo:

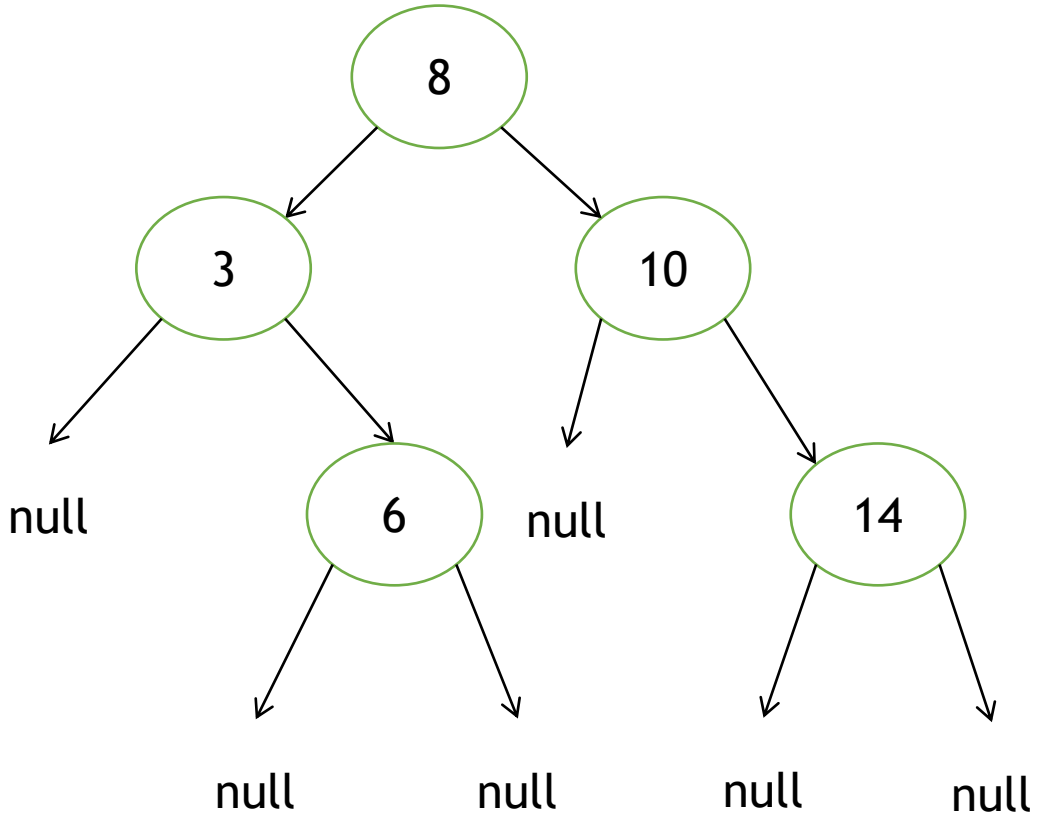
```
insere(arvore, 6)
```



Implementações

- Exemplo:

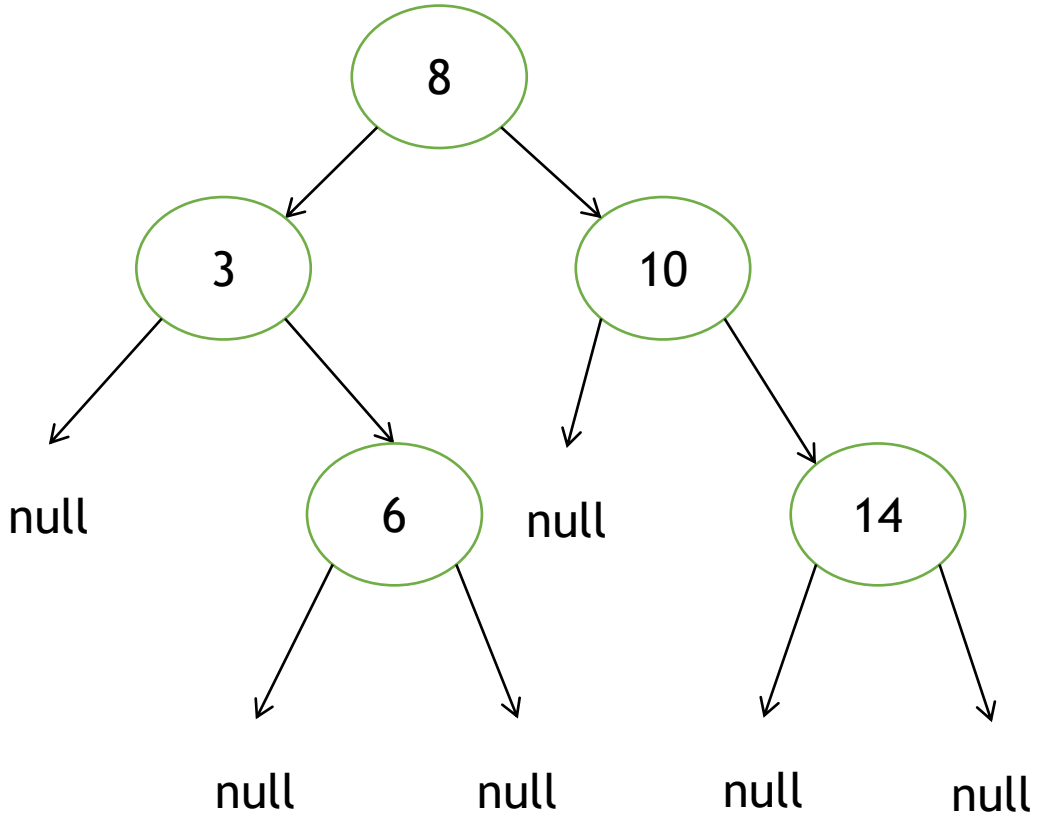
```
insere(arvore, 6)
```



Implementações

- Exemplo:

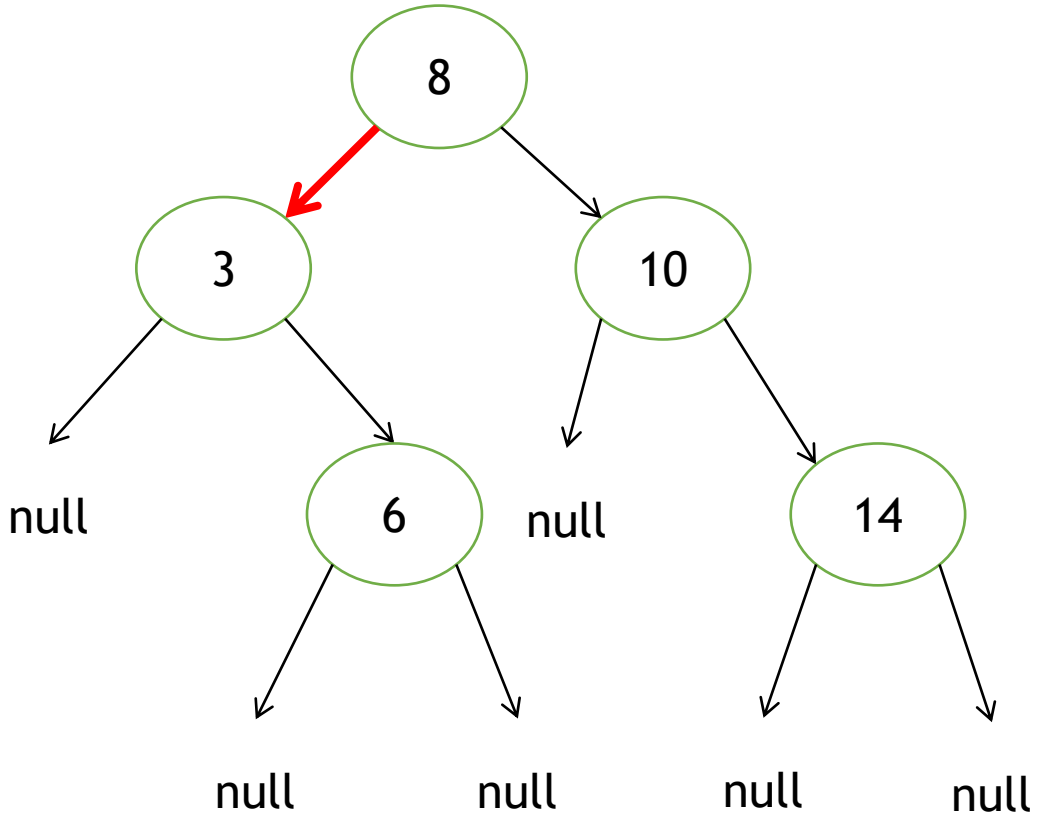
```
insere(arvore, 1)
```



Implementações

- Exemplo:

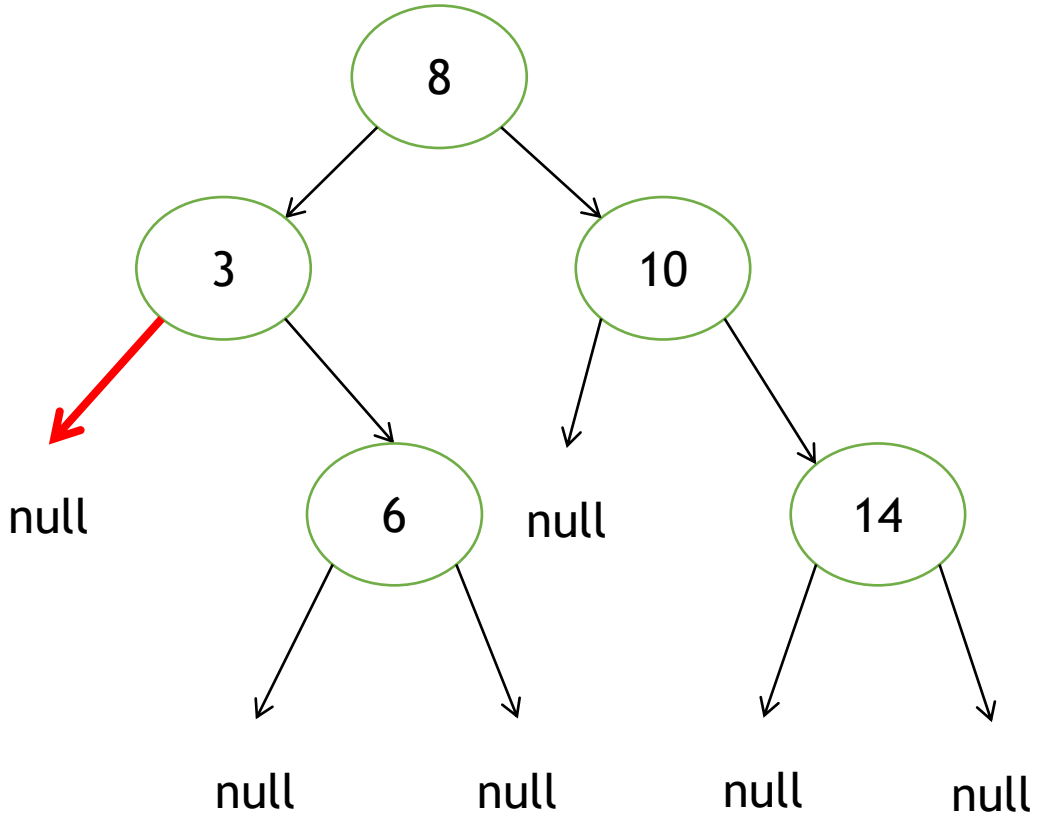
```
insere(arvore, 1)
```



Implementações

- Exemplo:

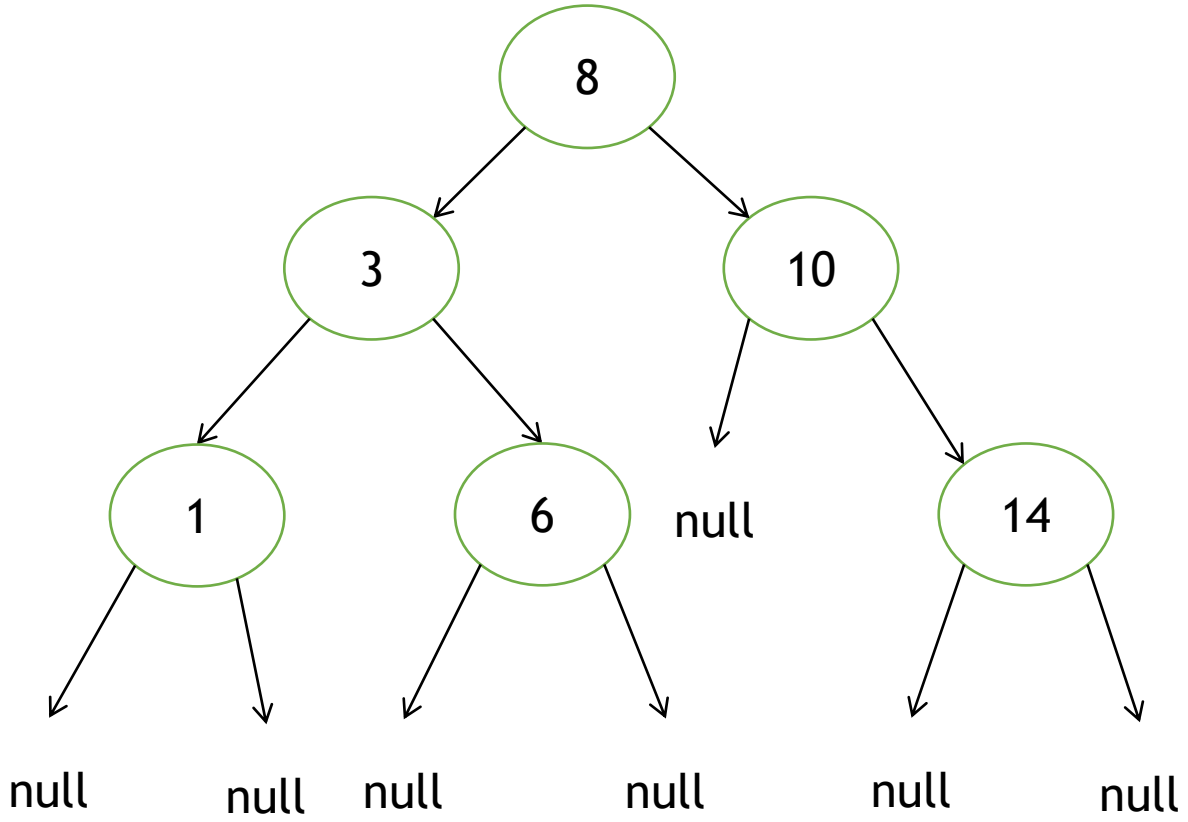
```
insere(arvore, 1)
```



Implementações

- Exemplo:

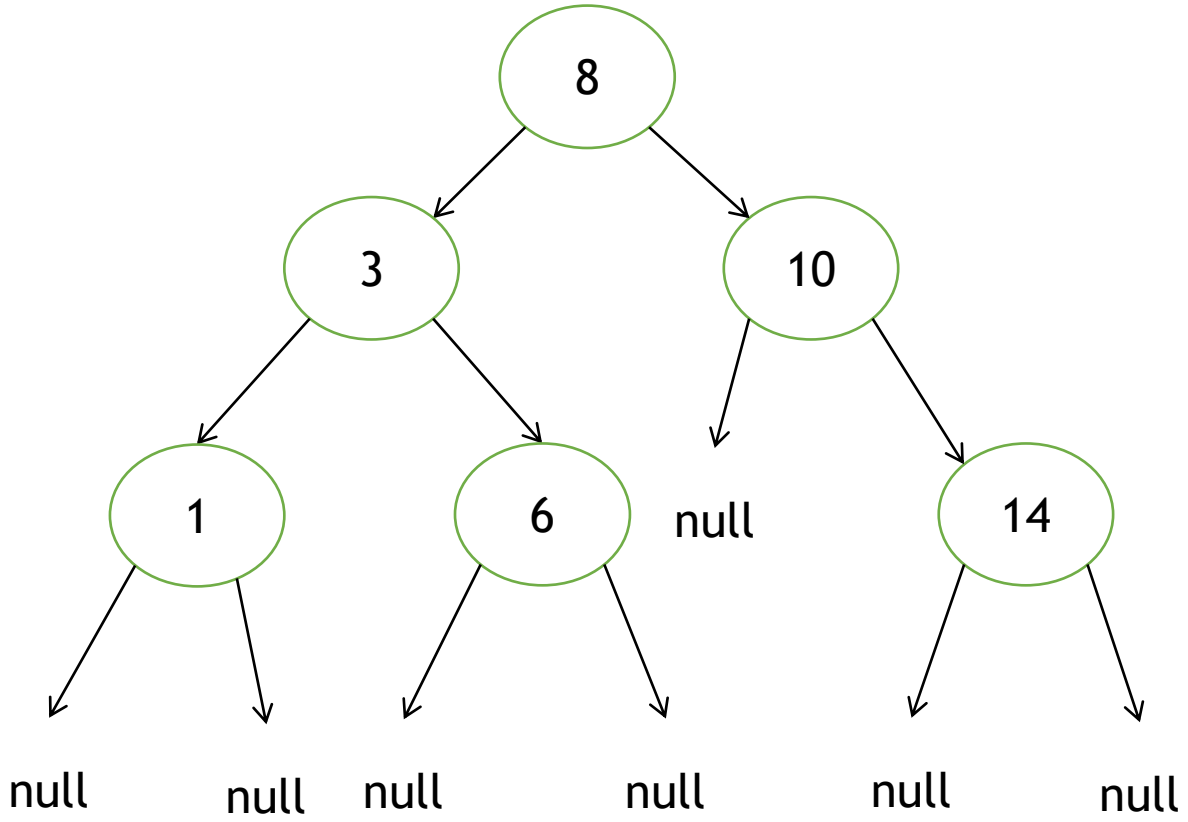
```
insere(arvore, 1)
```



Implementações

- Exemplo:

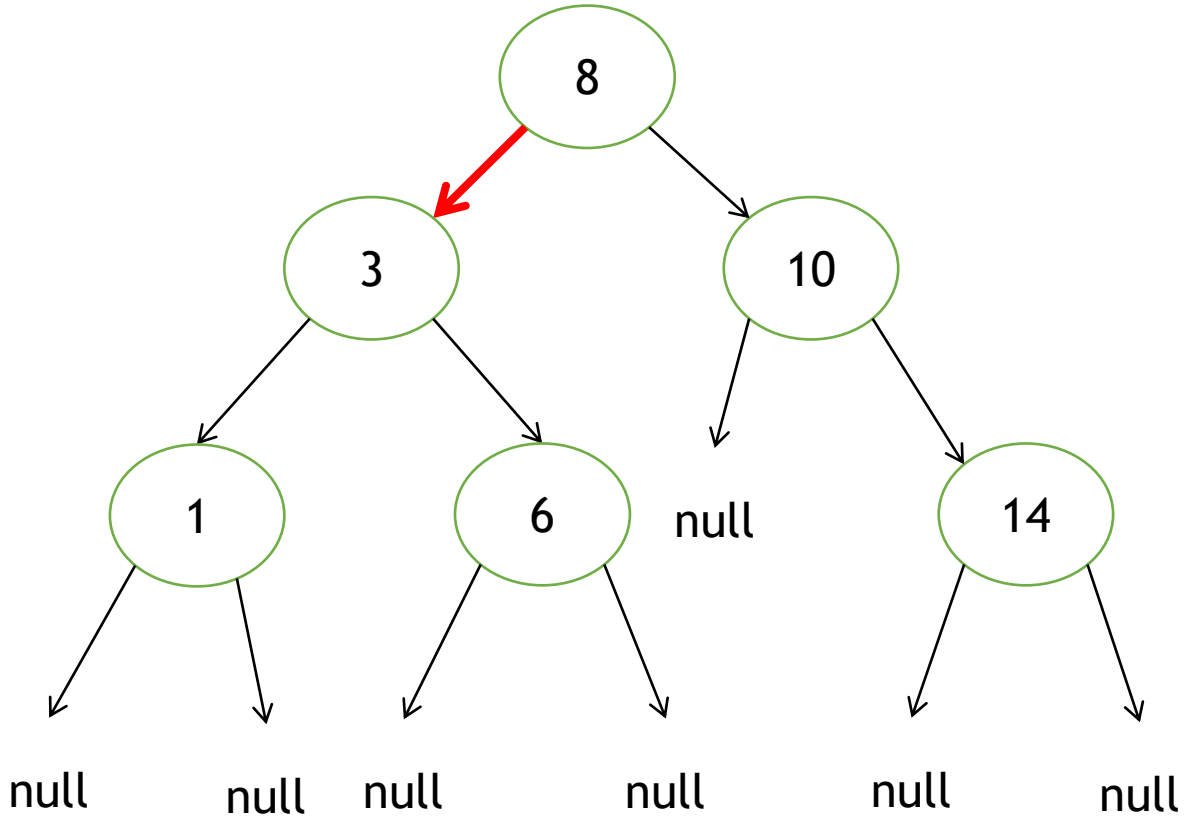
```
insere(arvore, 4)
```



Implementações

- Exemplo:

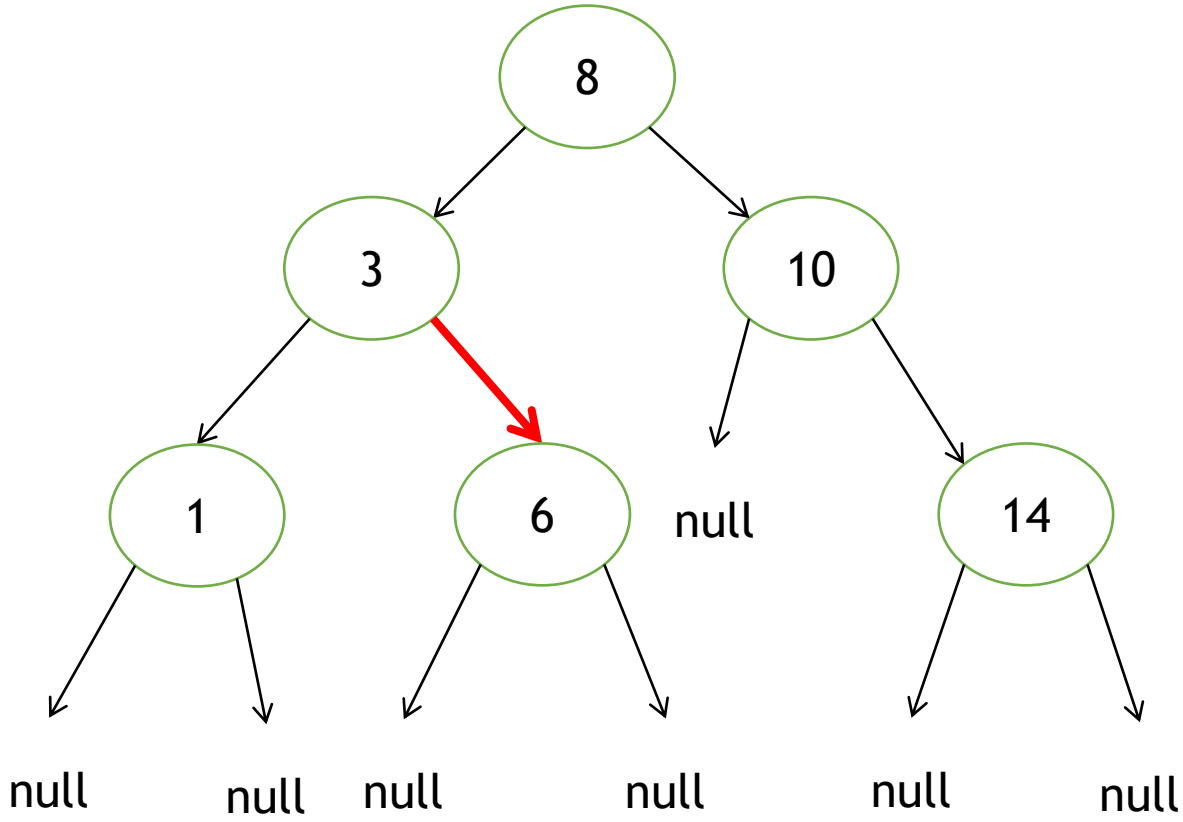
```
insere(arvore, 4)
```



Implementações

- Exemplo:

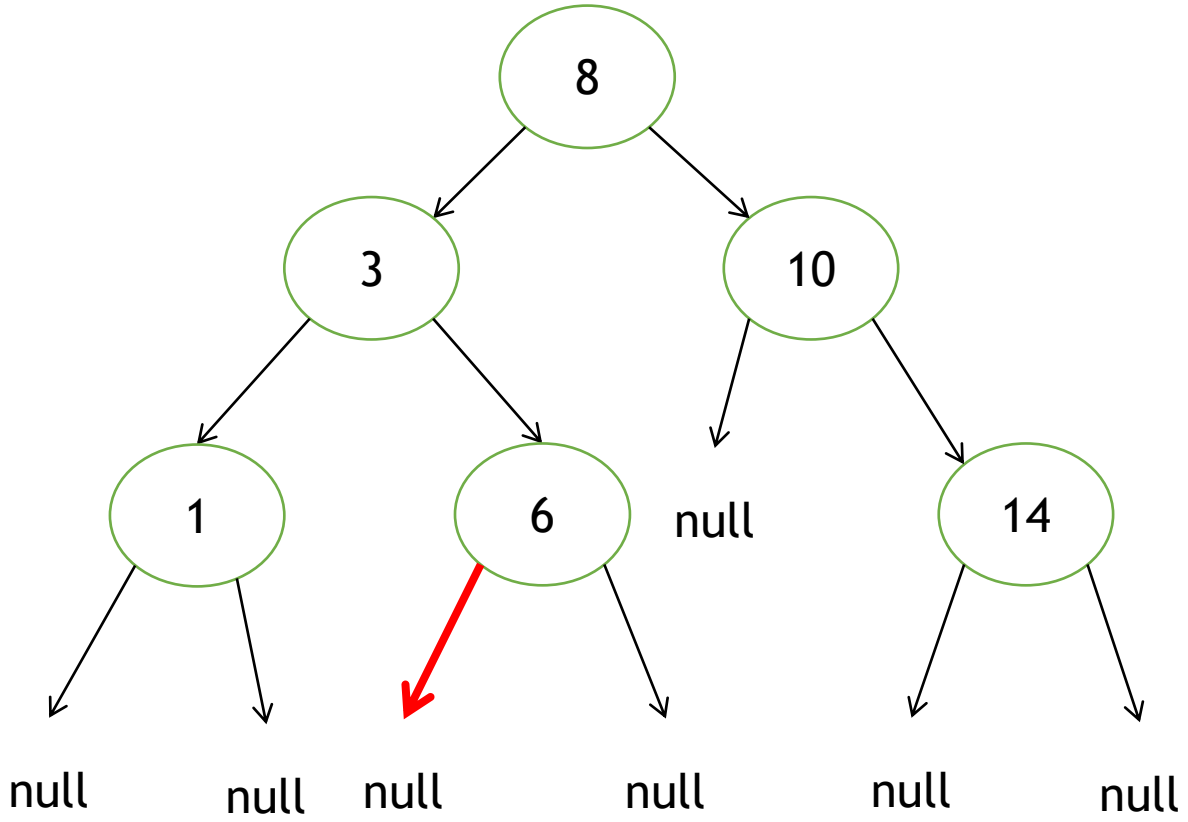
```
insere(arvore, 4)
```



Implementações

- Exemplo:

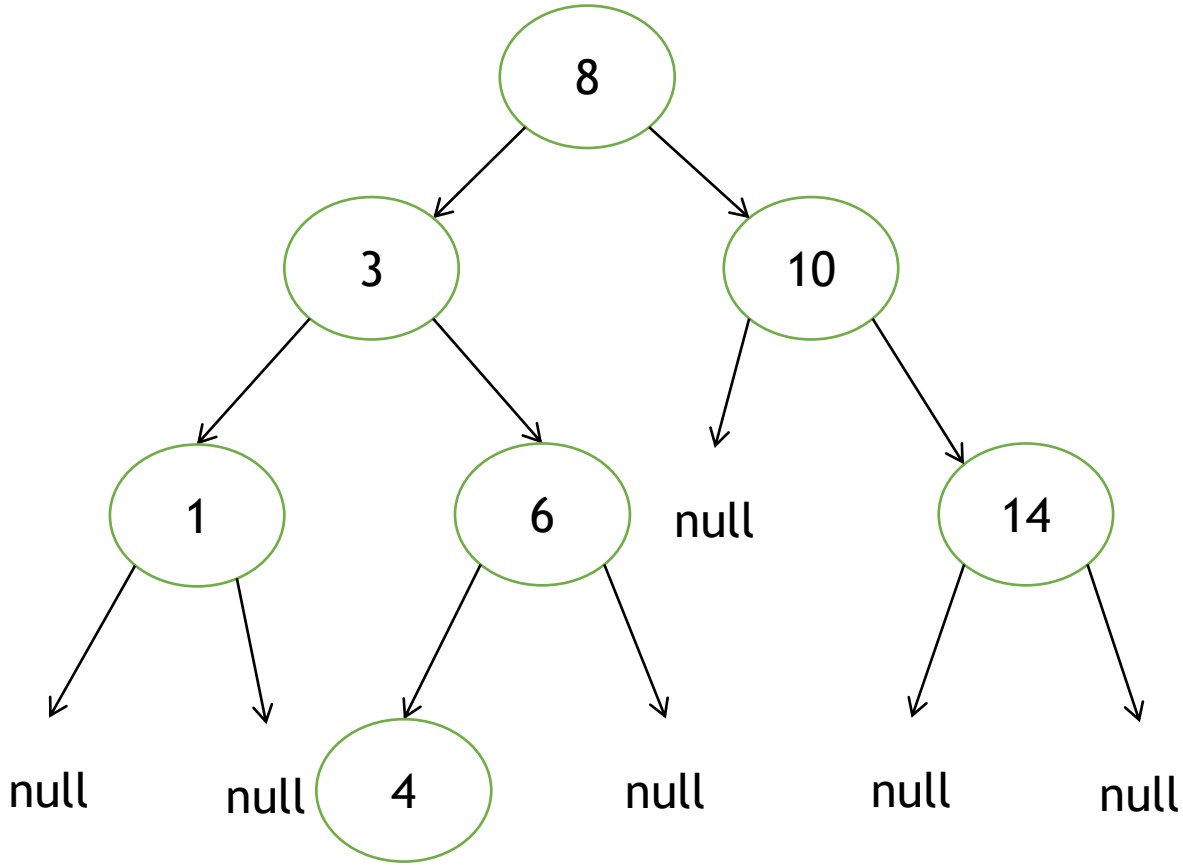
```
insere(arvore, 4)
```



Implementações

- Exemplo:

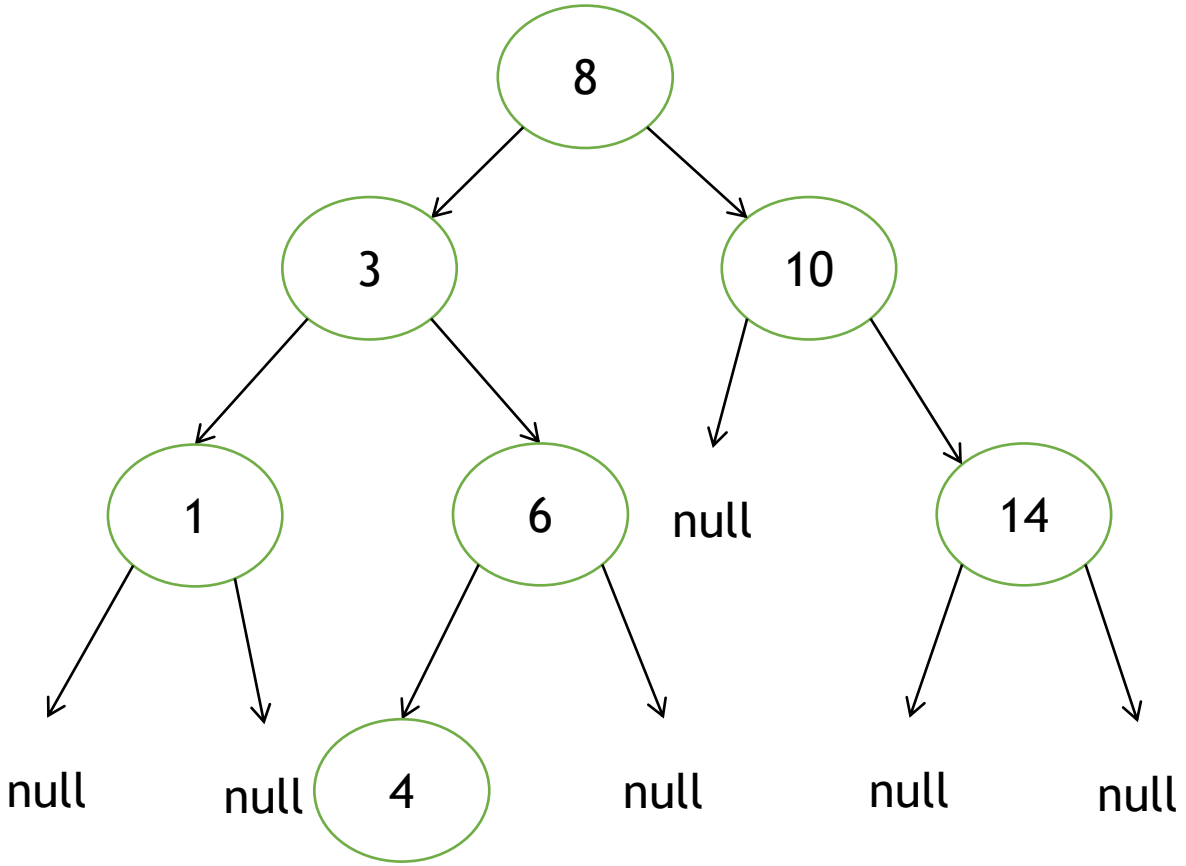
```
insere(arvore, 4)
```



Implementações

- Exemplo:

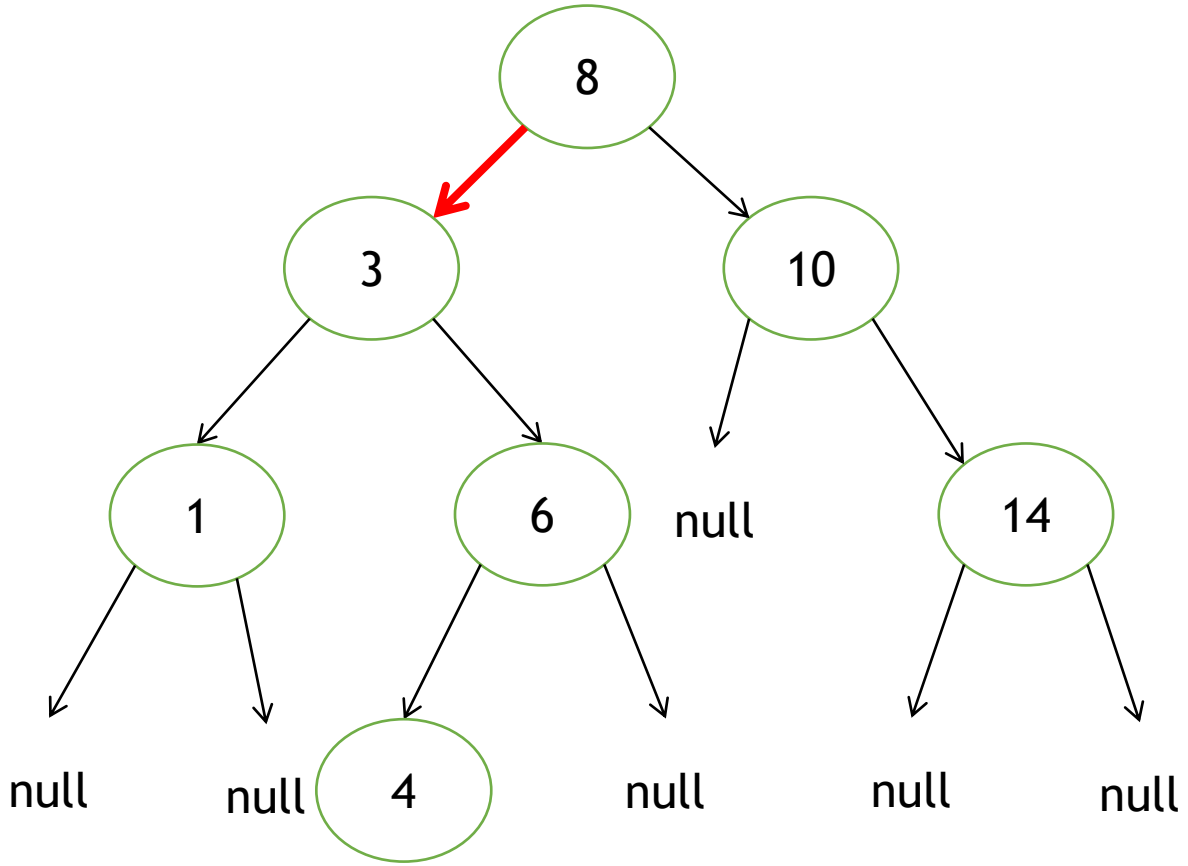
```
insere(arvore, 7)
```



Implementações

- Exemplo:

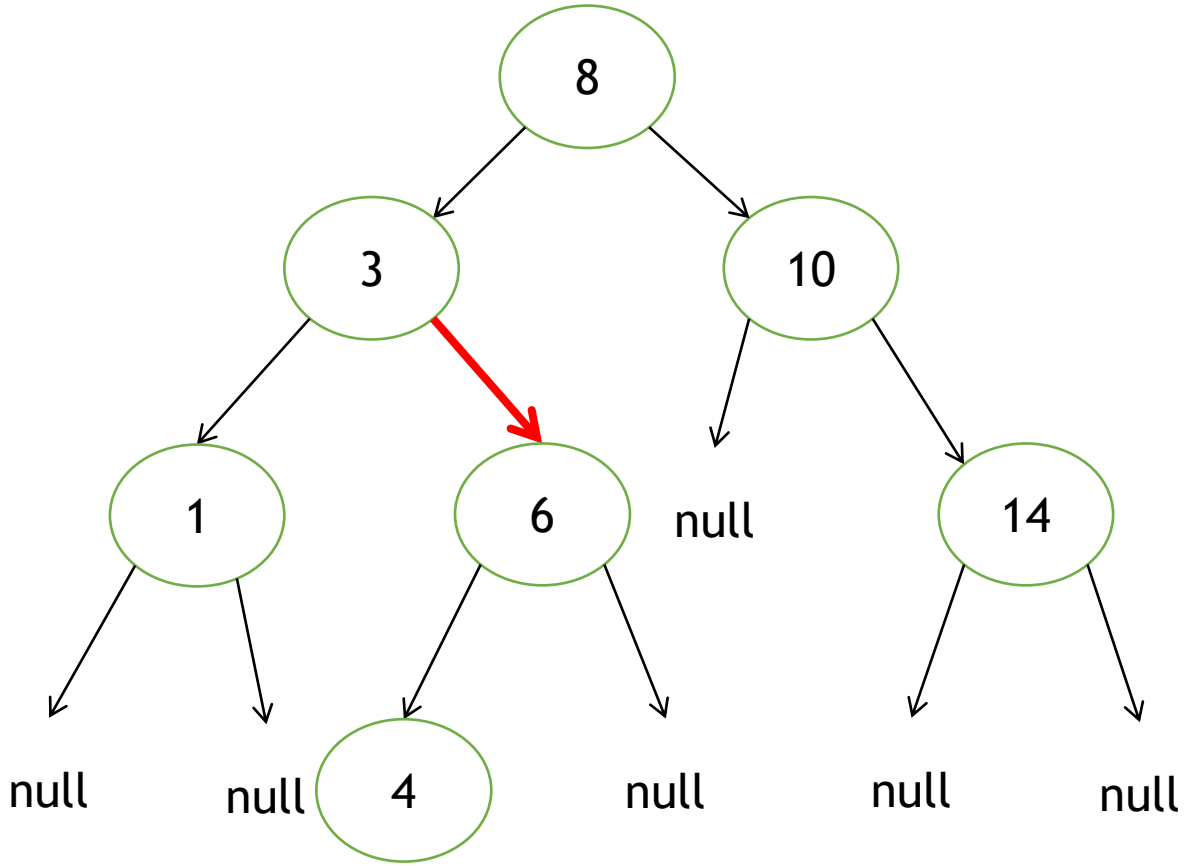
```
insere(arvore, 7)
```



Implementações

- Exemplo:

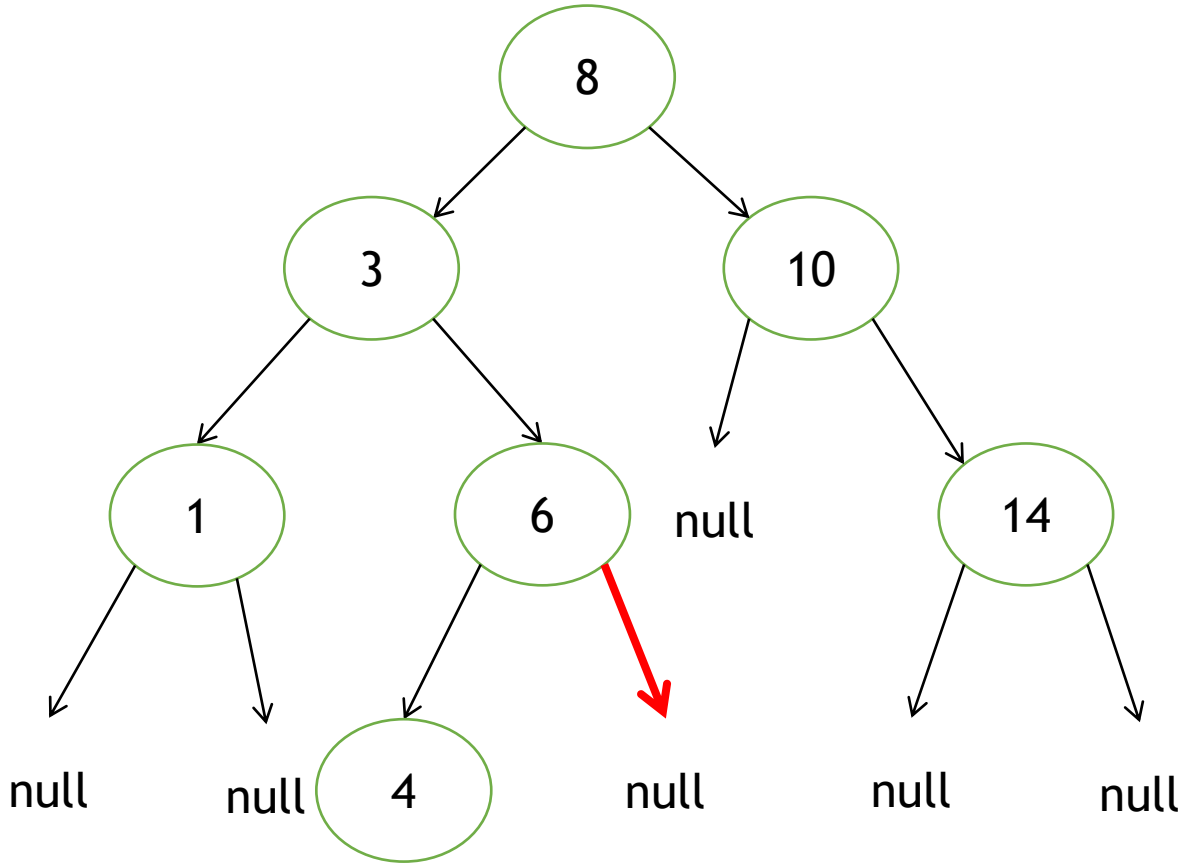
```
insere(arvore, 7)
```



Implementações

- Exemplo:

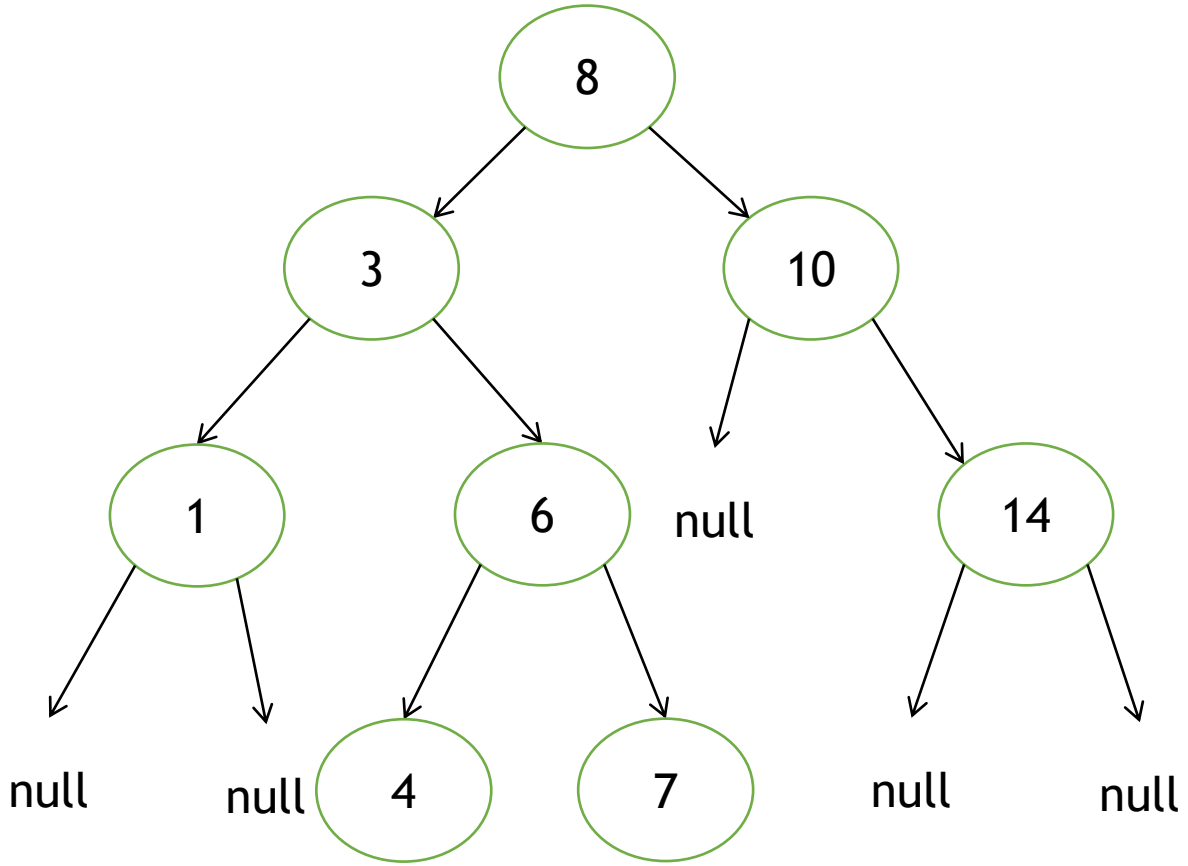
```
insere(arvore, 7)
```



Implementações

- Exemplo:

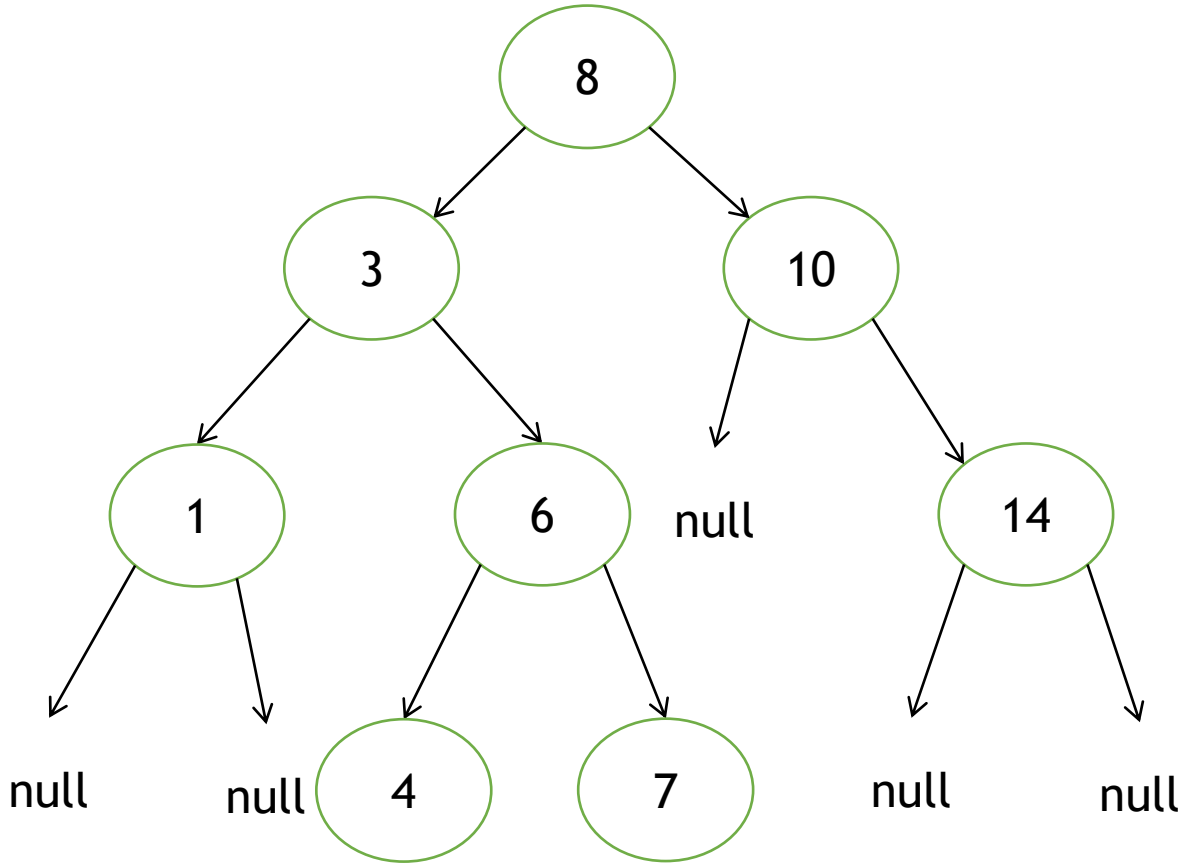
```
insere(arvore, 7)
```



Implementações

- Exemplo:

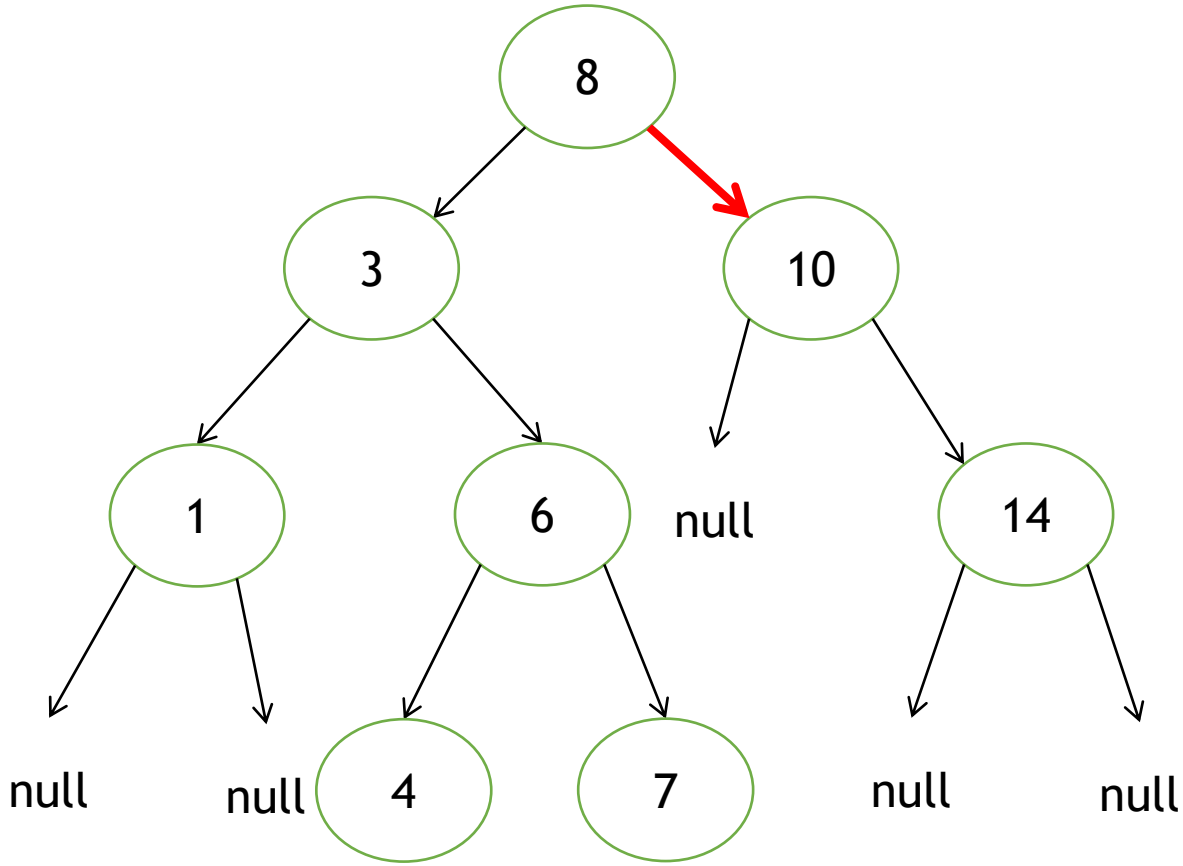
```
insere(arvore, 13)
```



Implementações

- Exemplo:

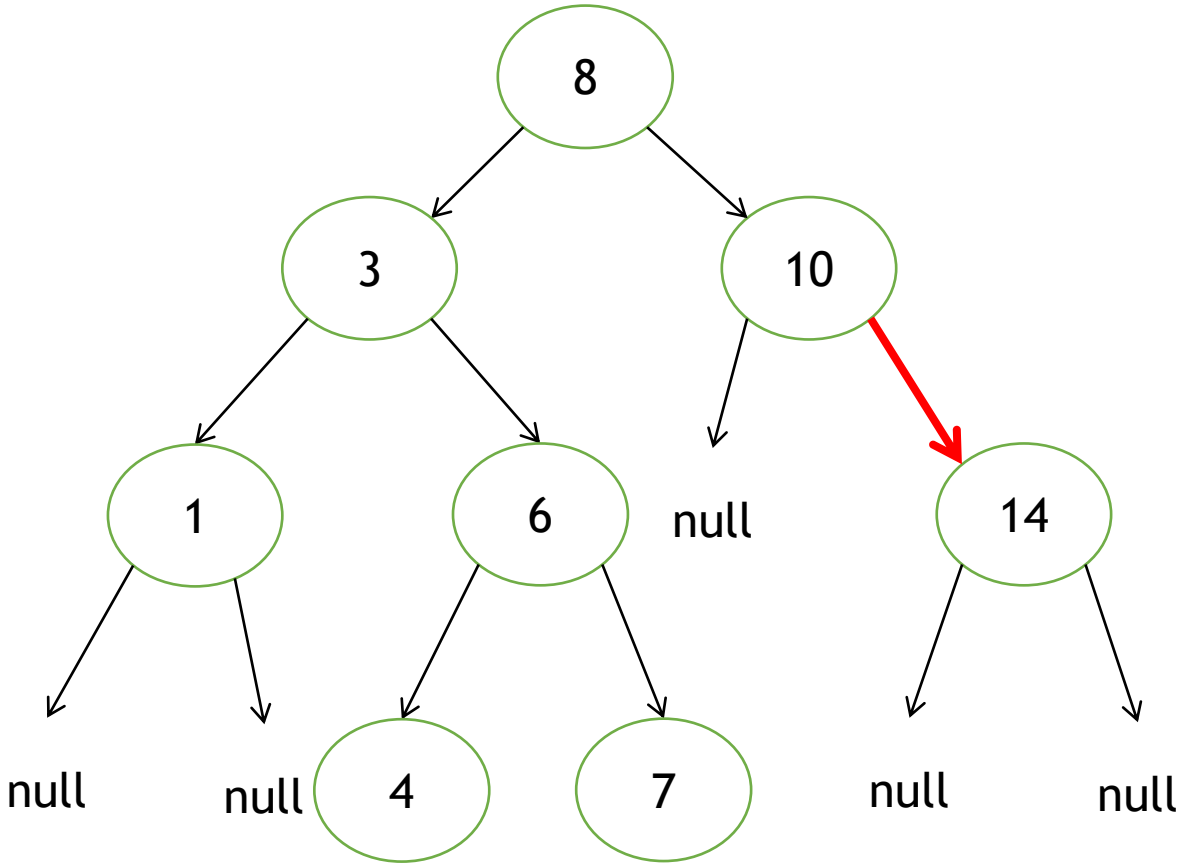
```
insere(arvore, 13)
```



Implementações

- Exemplo:

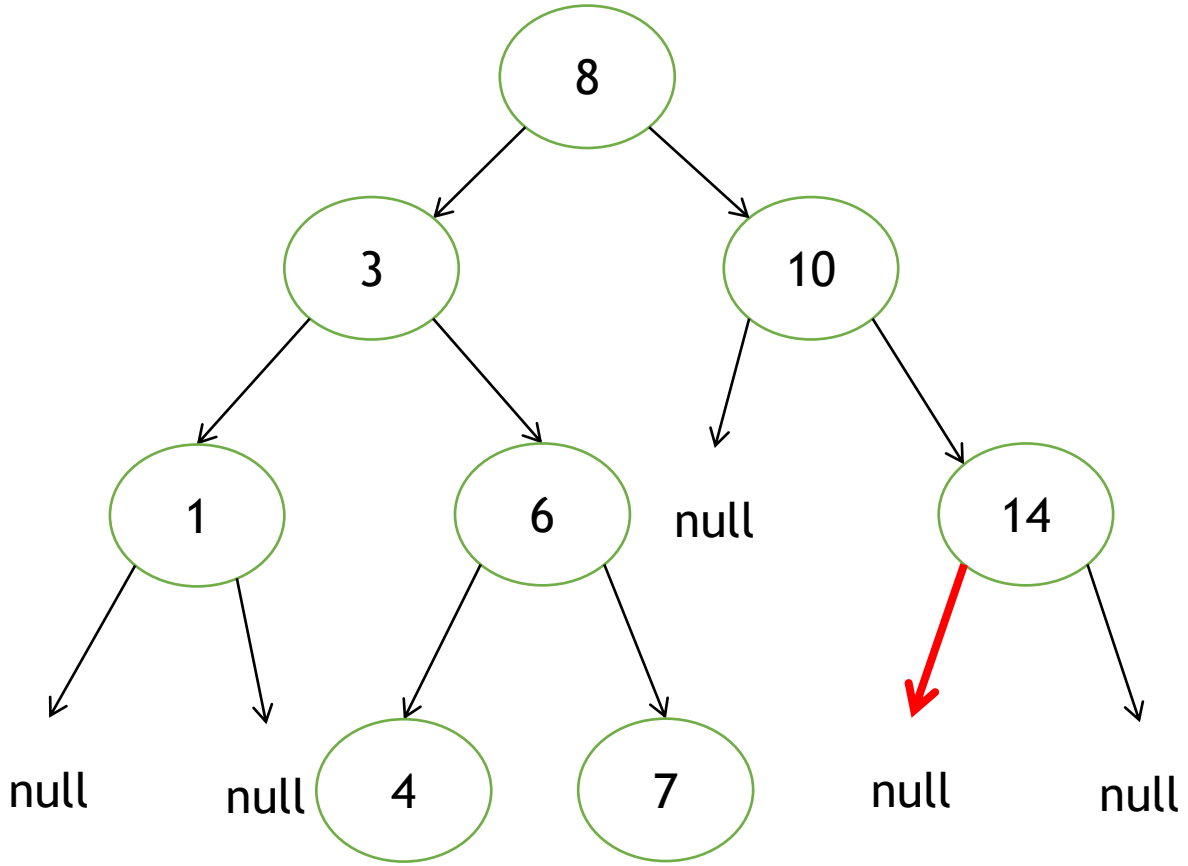
```
insere(arvore, 13)
```



Implementações

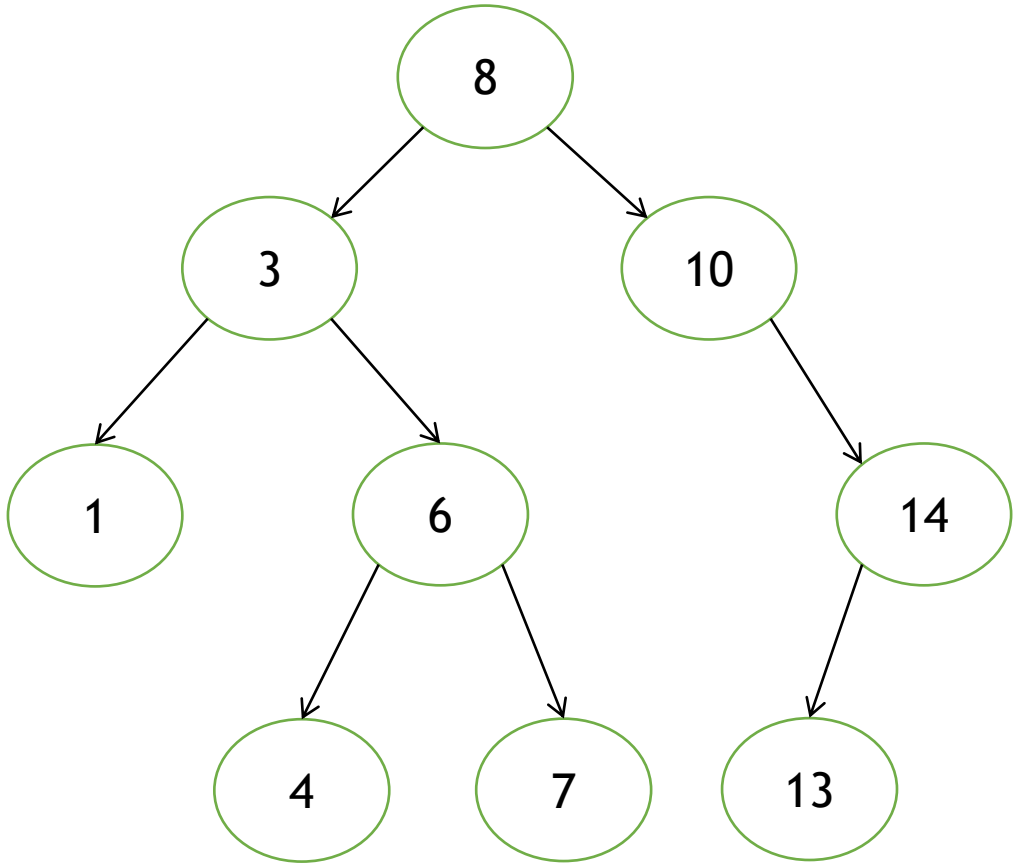
- Exemplo:

```
insere(arvore, 13)
```



Implementações

- Exemplo:

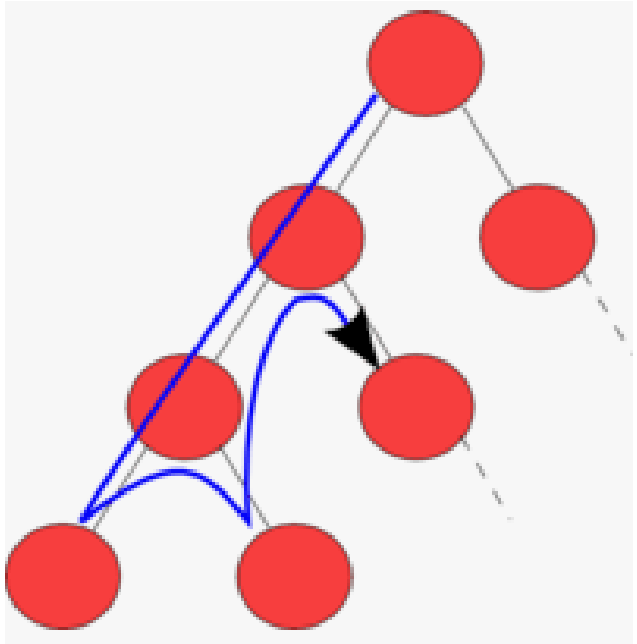


Busca em árvores

- Um algoritmo de busca (ou de varredura) é um algoritmo que visita todos os nós de um grafo ou árvore, andando pelas arestas de um vértice a outro.
- Uma varredura, por si só, não resolve um problema específico, mas ela serve de base para a resolução eficiente de vários problemas concretos.

Busca em Profundidade (DFS)

- Na busca em profundidade (*depth-first search*), o algoritmo começa na raiz da árvore e explora tanto quanto possível cada um dos seus ramos antes de retroceder (ideia semelhante ao *backtracking*).



Busca em Profundidade (DFS)

- Implementação 1 – versão recursiva

```
void dfs(int no)
{
    processa(no);
    for(int i = 0; i < arvore[no].size(); i++)
        dfs(arvore[no][i]);
}
```

Busca em Profundidade (DFS)

- Implementação 1 – versão recursiva 2

```
void dfs(int no)
{
    processa(no);
    for(auto v: arvore[no])
        dfs(v);
}
```

Busca em Profundidade (DFS)

- Implementação 2 – versão recursiva

```
void dfs(def_arvore no)
{
    if (no == NULL)
        return;
    processa(no);
    dfs(no->esq);
    dfs(no->dir);
}
```

Busca em Profundidade (DFS)

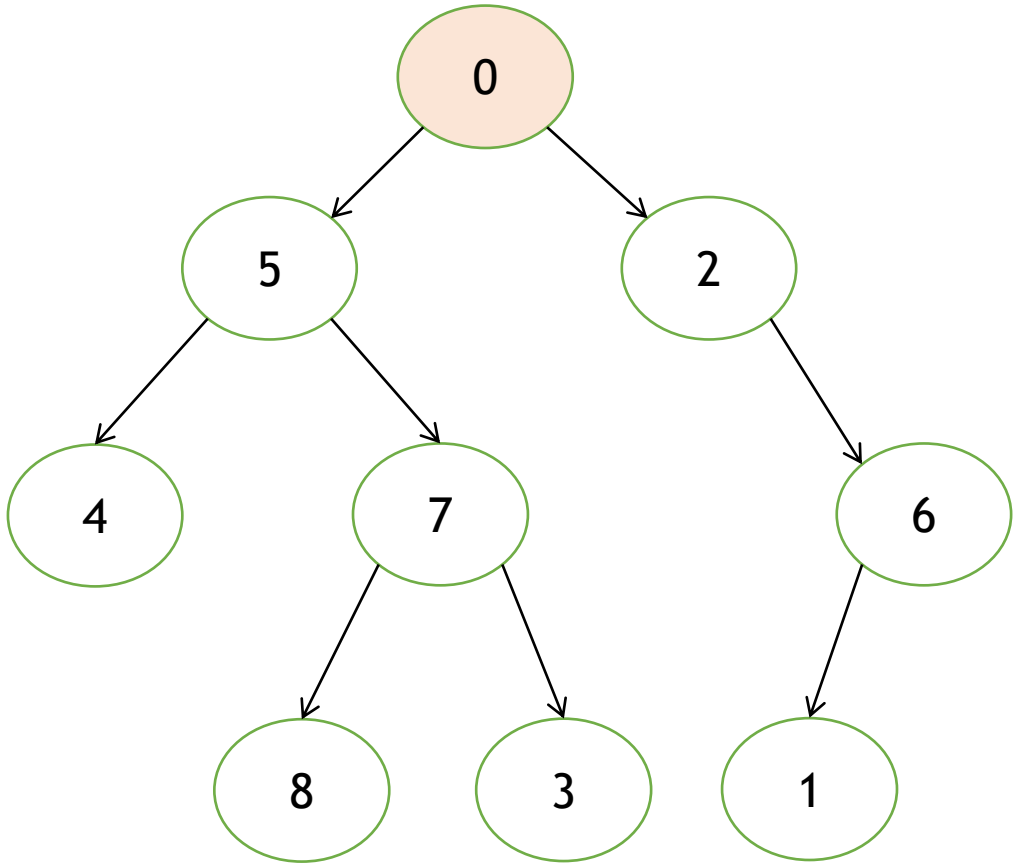
- Implementação 1 – versão iterativa

```
void dfs(int raiz){
    stack<int> pilha;
    int no;
    pilha.push(raiz);
    while(!pilha.empty()){
        no = pilha.top();
        pilha.pop();
        processa(no);
        for(auto v : arvore[no])
            pilha.push(v);
    }
}
```

Busca em Profundidade (DFS)

Pilha = {0}

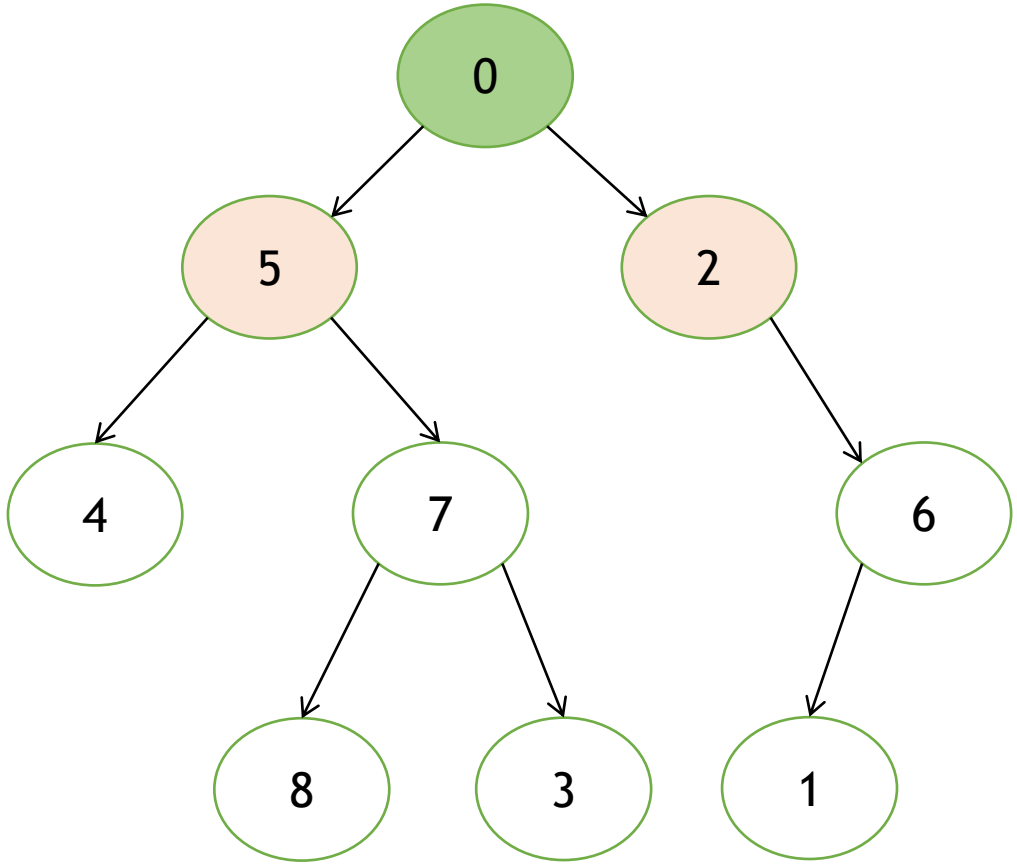
Nó	Ordem de visitação
0	
1	
2	
3	
4	
5	
6	
7	
8	



Busca em Profundidade (DFS)

Pilha = {2, 5}

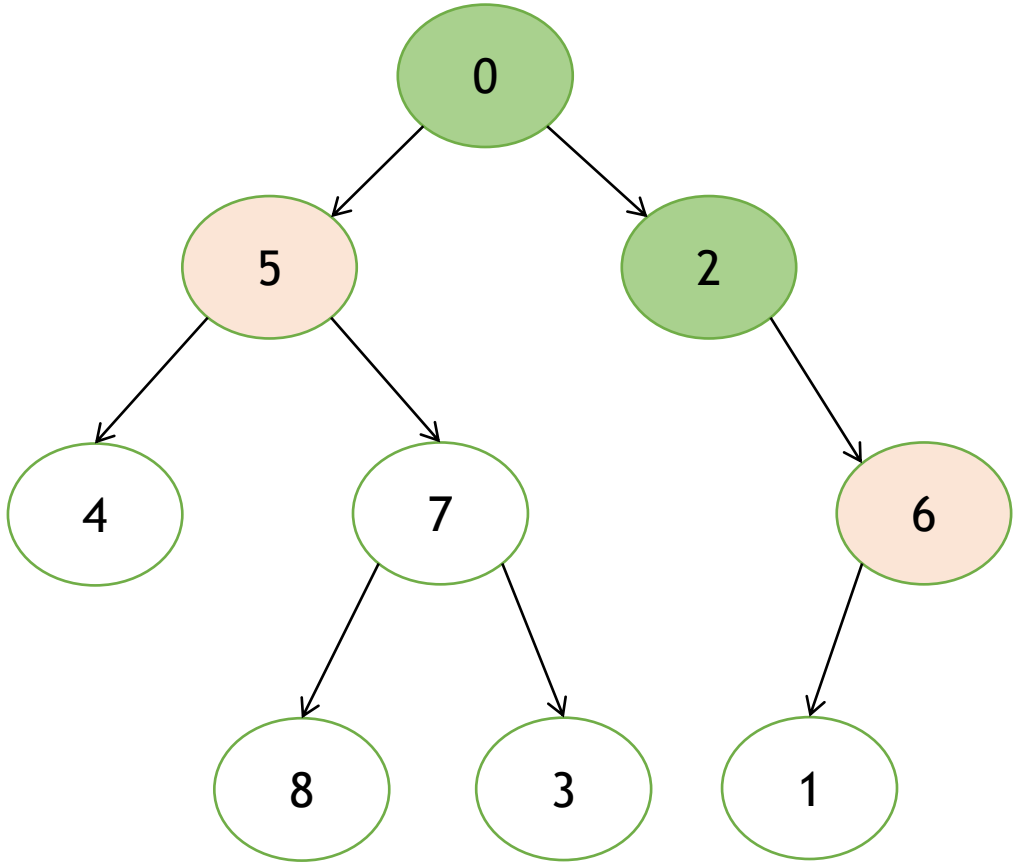
Nó	Ordem de visitação
0	1
1	
2	
3	
4	
5	
6	
7	
8	



Busca em Profundidade (DFS)

Pilha = {6, 5}

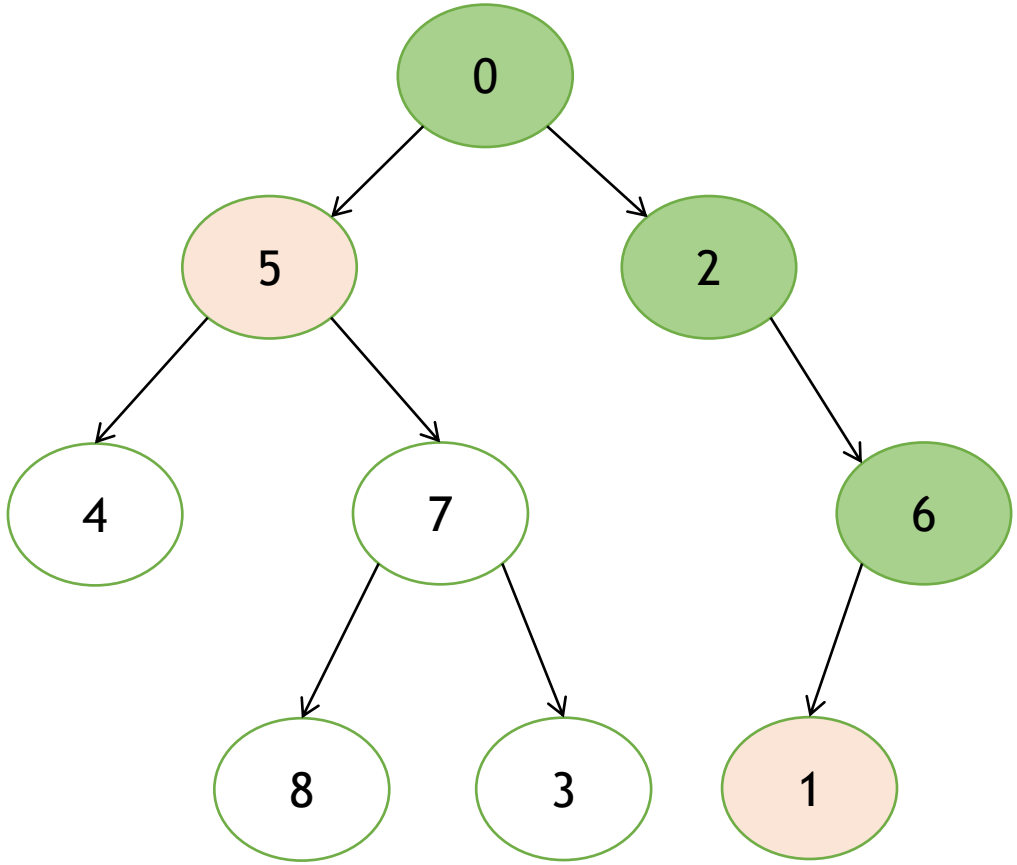
Nó	Ordem de visitação
0	1
1	
2	2
3	
4	
5	
6	
7	
8	



Busca em Profundidade (DFS)

Pilha = {1, 5}

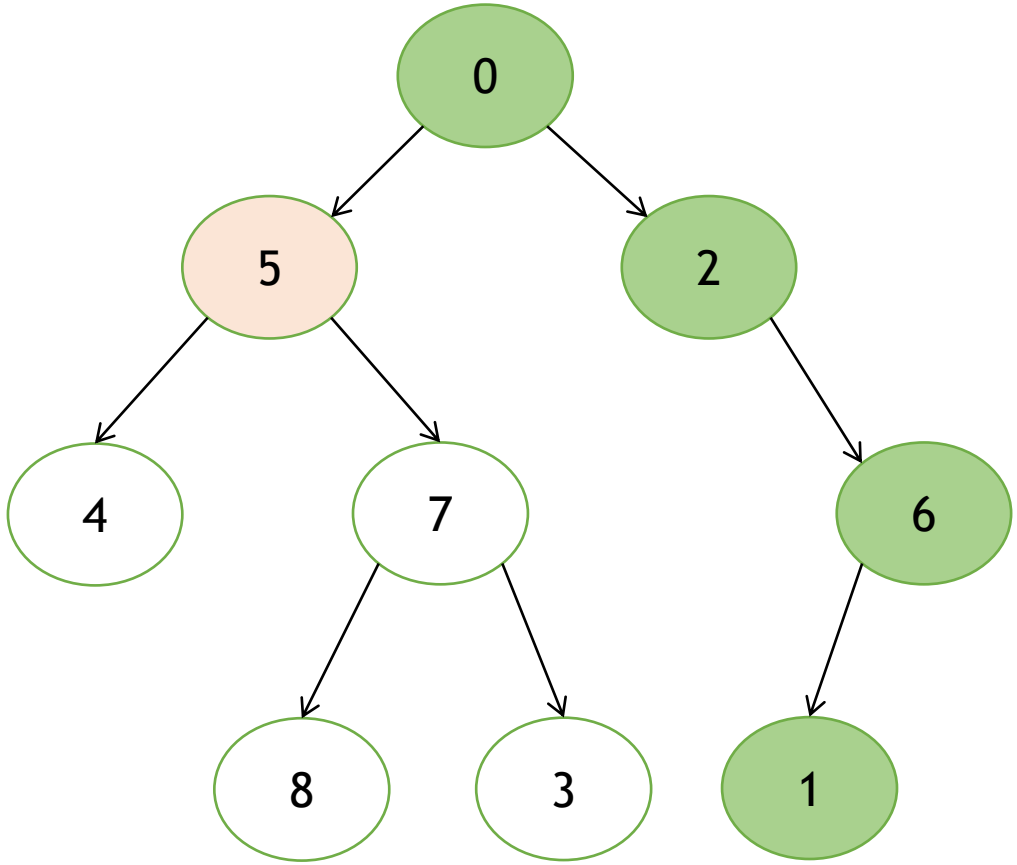
Nó	Ordem de visitação
0	1
1	
2	2
3	
4	
5	
6	3
7	
8	



Busca em Profundidade (DFS)

Pilha = {5}

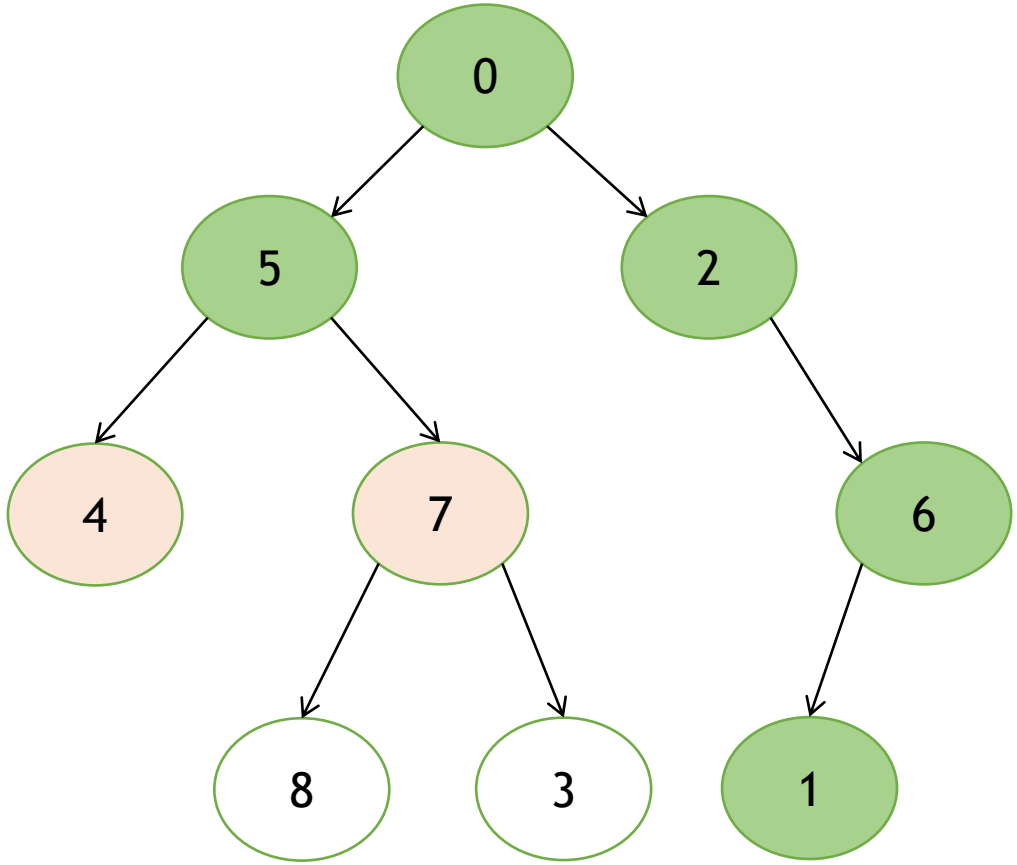
Nó	Ordem de visitação
0	1
1	4
2	2
3	
4	
5	
6	3
7	
8	



Busca em Profundidade (DFS)

Pilha = {7, 4}

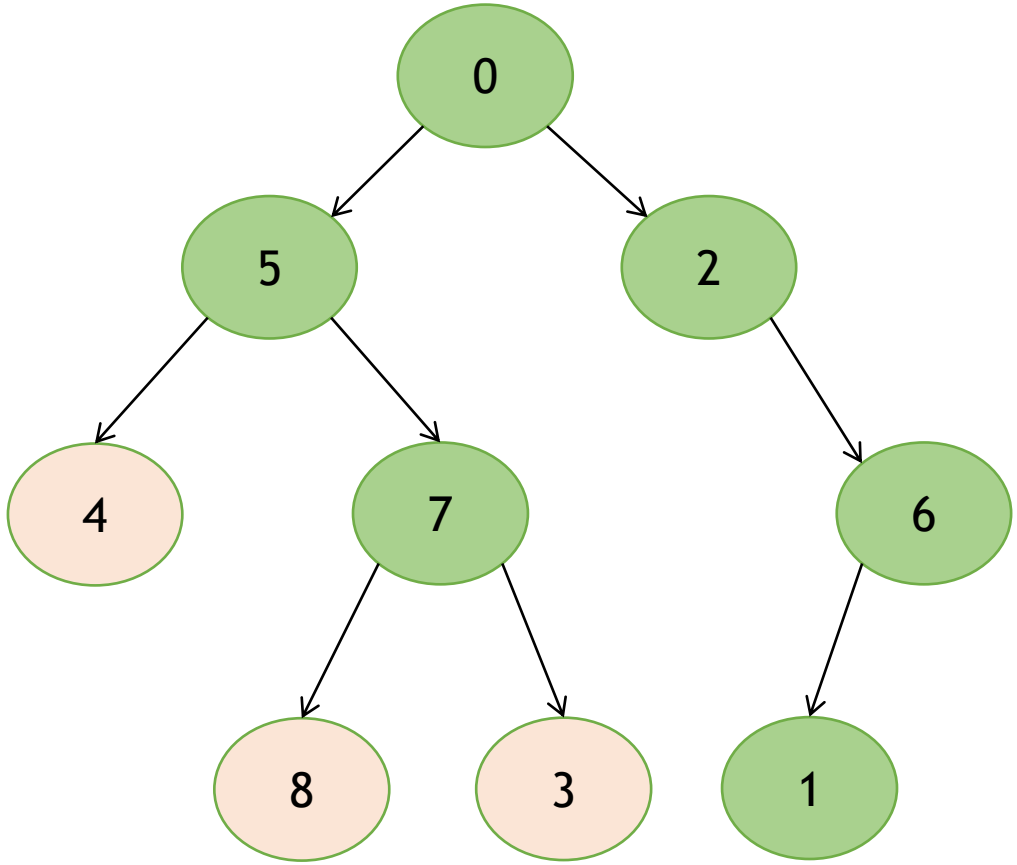
Nó	Ordem de visitação
0	1
1	4
2	2
3	
4	
5	5
6	3
7	
8	



Busca em Profundidade (DFS)

Pilha = {3, 8, 4}

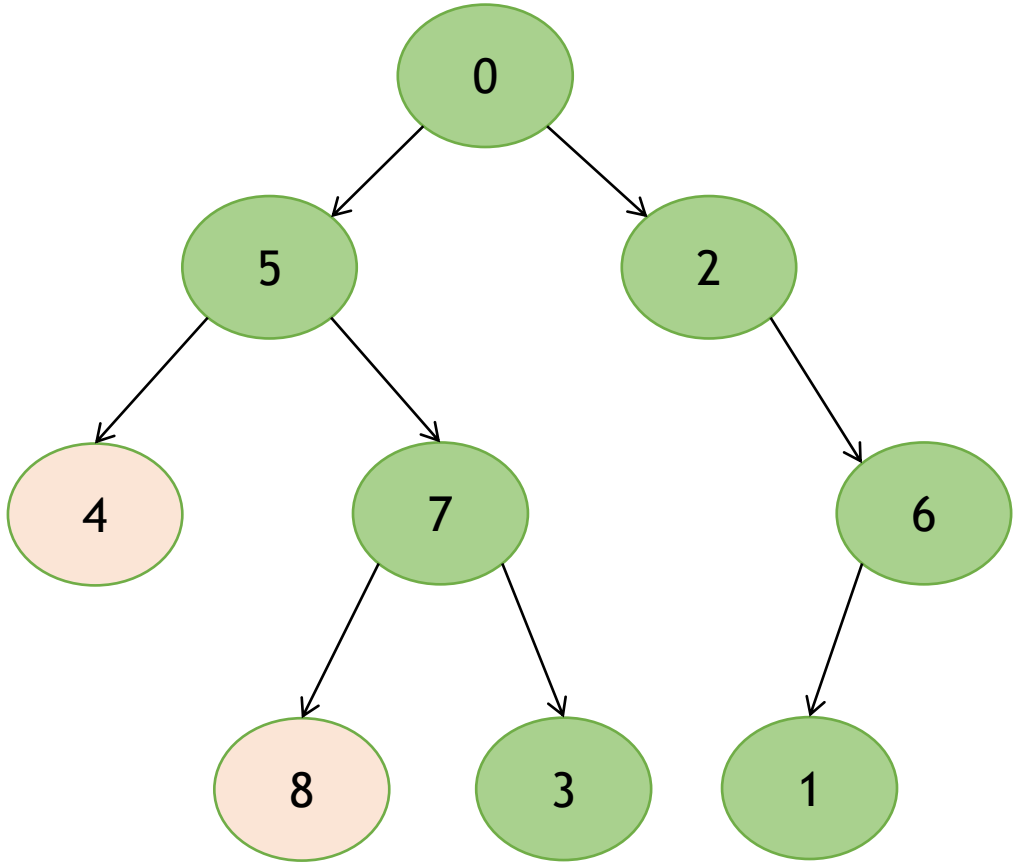
Nó	Ordem de visitação
0	1
1	4
2	2
3	
4	
5	5
6	3
7	6
8	



Busca em Profundidade (DFS)

Pilha = {8, 4}

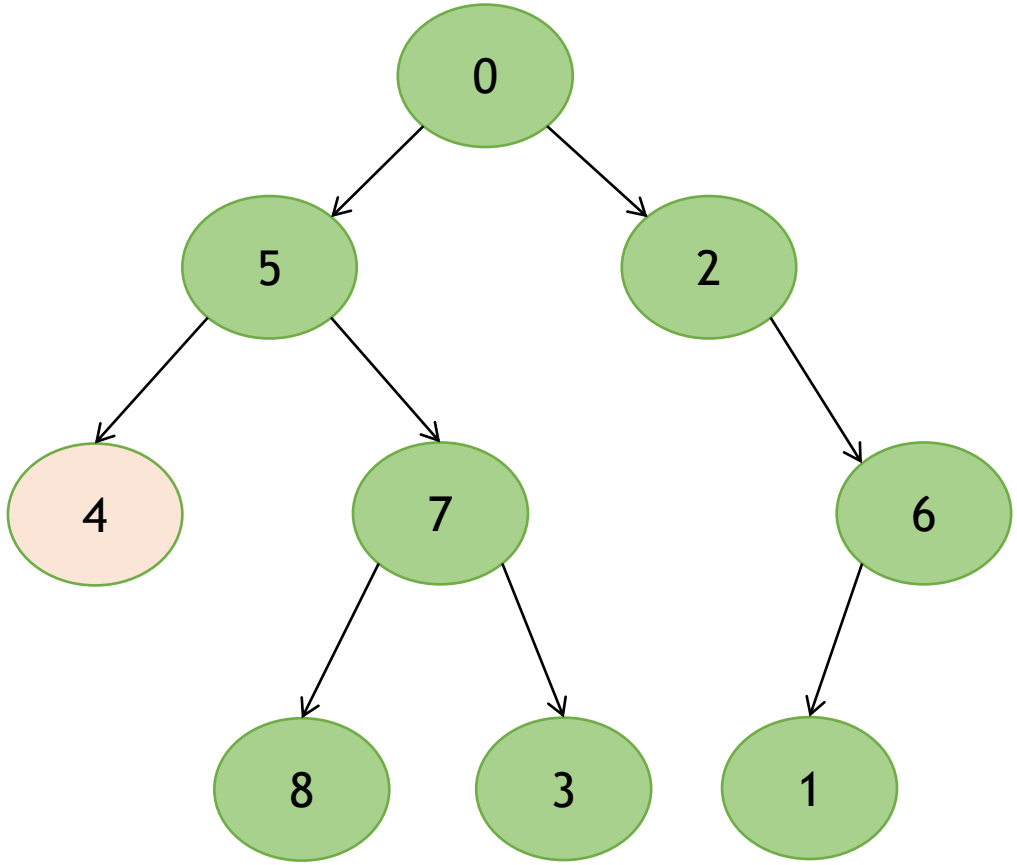
Nó	Ordem de visitação
0	1
1	4
2	2
3	7
4	
5	5
6	3
7	6
8	



Busca em Profundidade (DFS)

Pilha = {4}

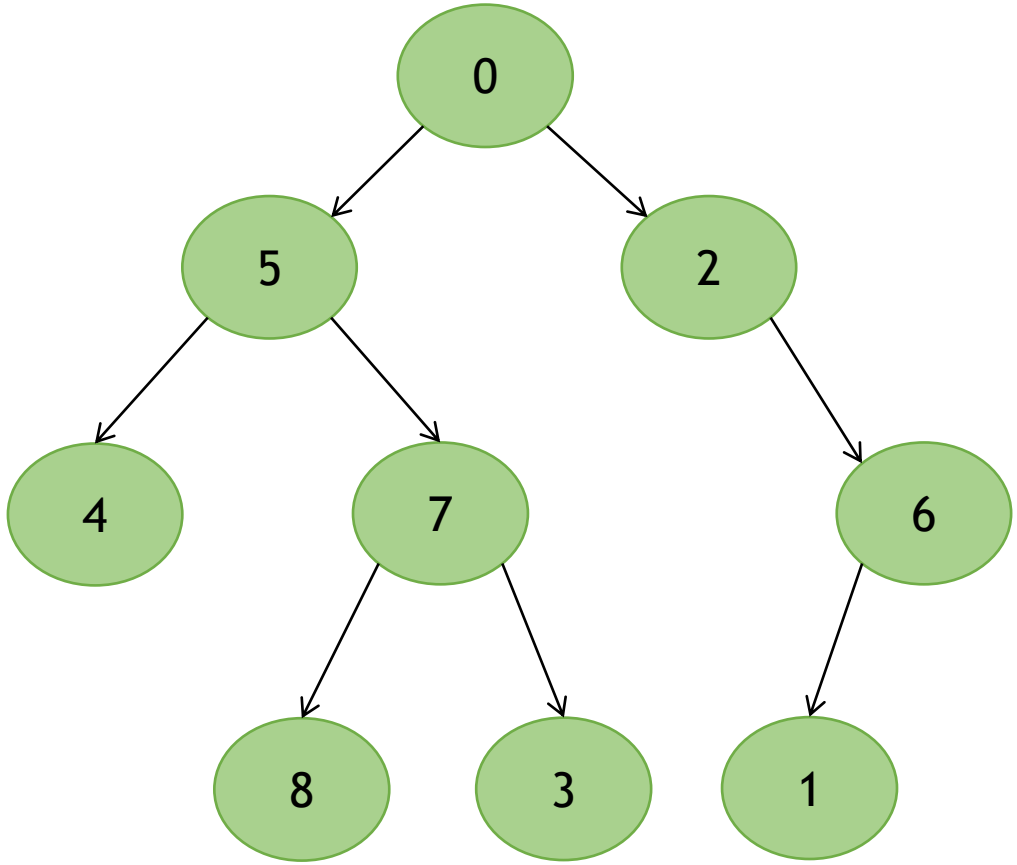
Nó	Ordem de visitação
0	1
1	4
2	2
3	7
4	
5	5
6	3
7	6
8	8



Busca em Profundidade (DFS)

Pilha = {}

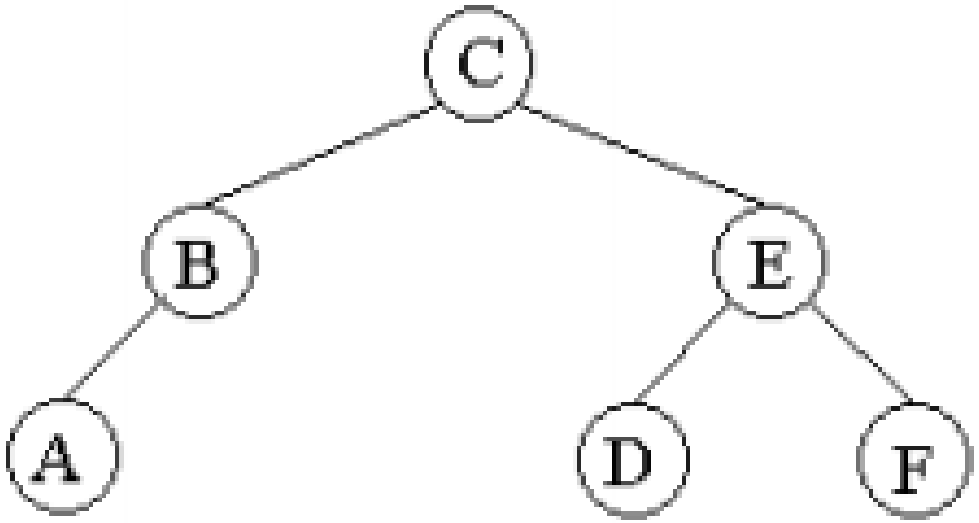
Nó	Ordem de visitação
0	1
1	4
2	2
3	7
4	9
5	5
6	3
7	6
8	8



Percurso em árvore binária

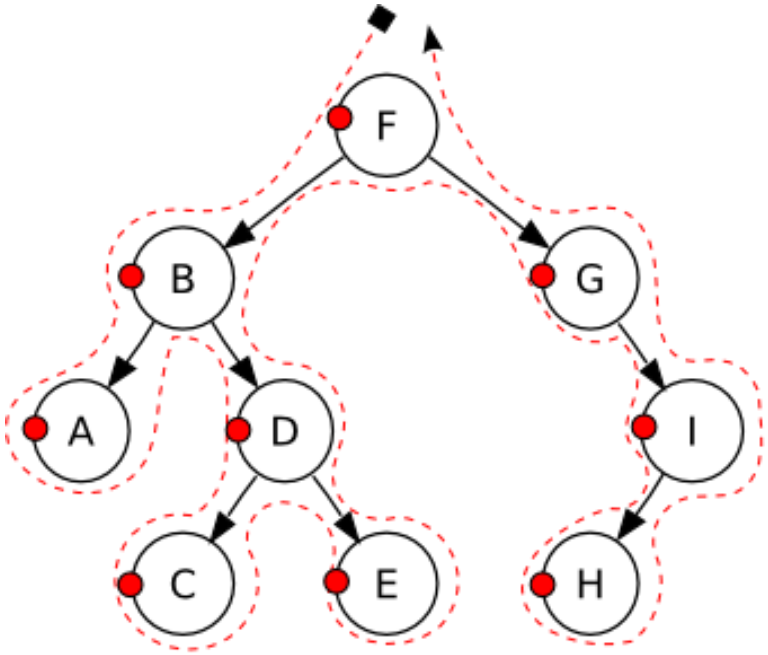
- Ao aplicar uma busca em profundidade em uma árvore binária, temos três opções de percurso, considerando a ordem de visitação da raiz (R), da subárvore esquerda (E) e da subárvore direita(D):
  - Pré-ordem ou prefixo: R, E, D
  - Em-ordem ou infixo: E, R, D
  - Pós-ordem ou posfixo: E, D, R

Percurso em árvore binária

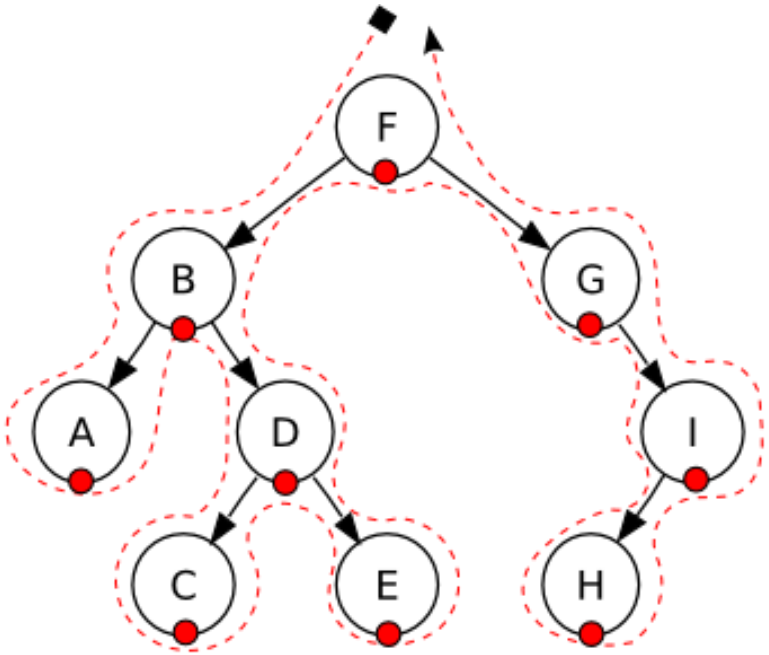


Pré-ordem:	C B A E D F
Em-ordem:	A B C D E F
Pós-ordem:	A B D F E C

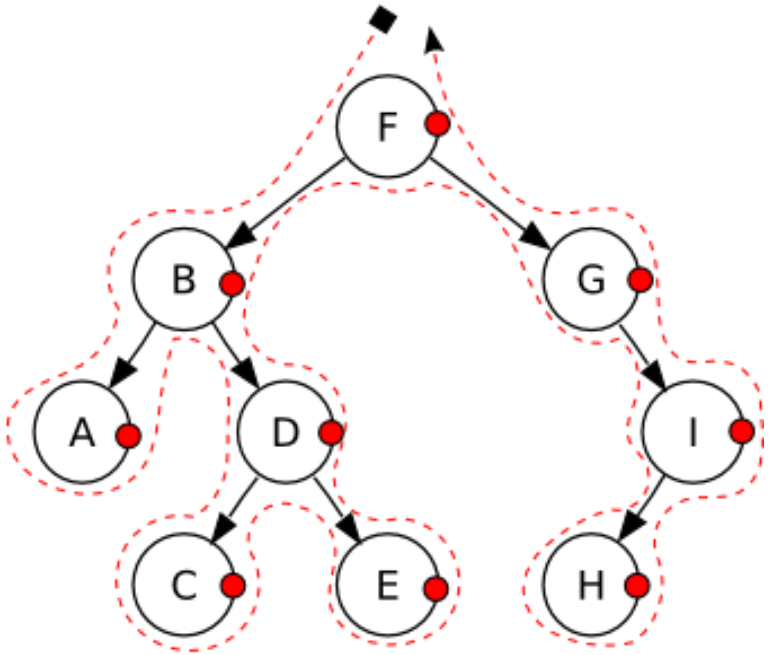
Percurso em árvore binária



Pré-ordem: F, B, A, D, C, E, G, I, H



Ordem simétrica: A, B, C, D, E, F, G, H, I



Pós-ordem: A, C, E, D, B, H, I, G, F

Percurso em árvore binária

- Exemplo: árvore de expressões
  - Notação infixa (convencional)

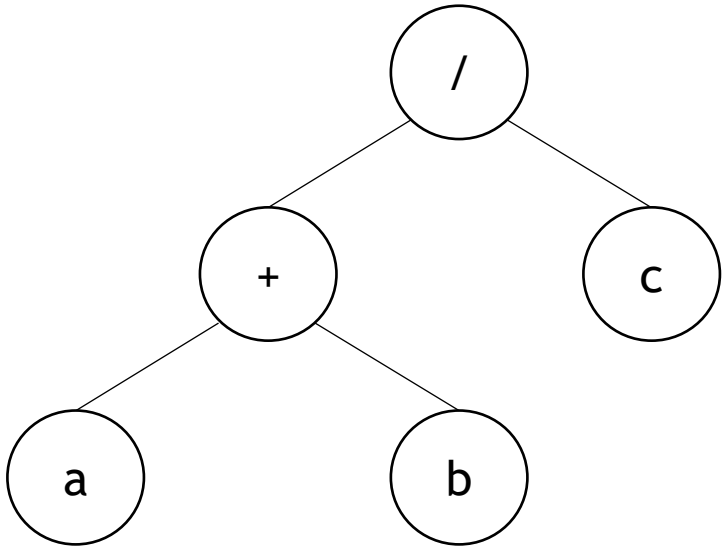
$(a + b)/c$

  - Notação prefixa (polonesa)

$/ + a\ b\ c$

  - Notação posfixa (polonesa reversa)

$a\ b + c\ /$



Busca em Largura (BFS)

- Diferente da busca em profundidade, que faz a exploração ramo por ramo, a busca em largura (*breadth-first search*) faz a busca nível por nível.
- Inicialmente a raiz é processada (nível 0), a seguir todos os seus filhos (nível 1), então os filhos dos seus filhos (nível 2) e assim por diante.
- A implementação da BFS é muito semelhante a versão iterativa da DFS, mas utilizando uma fila ao invés de uma pilha.

Busca em Largura (BFS)

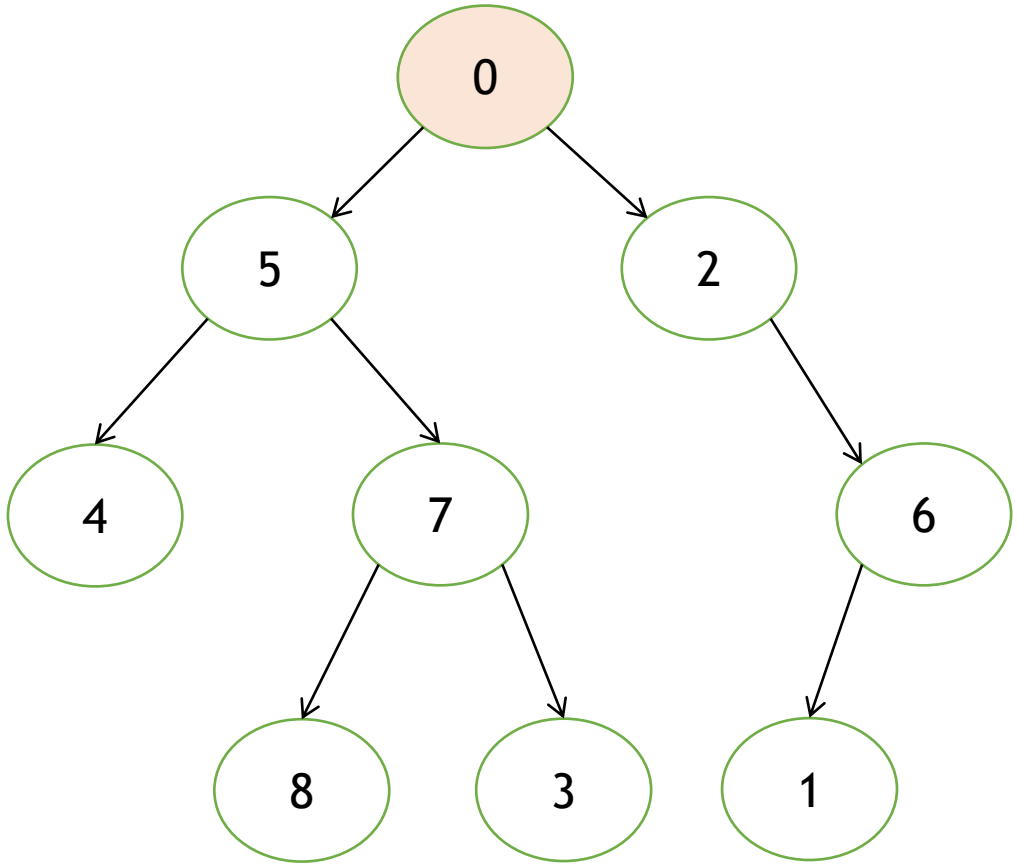
- Implementação

```
void bfs(int raiz){
    queue<int> fila;
    int no;
    fila.push(raiz);
    while(!fila.empty()){
        no = fila.front();
        fila.pop();
        processa(no);
        for(auto v : arvore[no])
            fila.push(v);
    }
}
```

Busca em Largura (BFS)

Fila = {0}

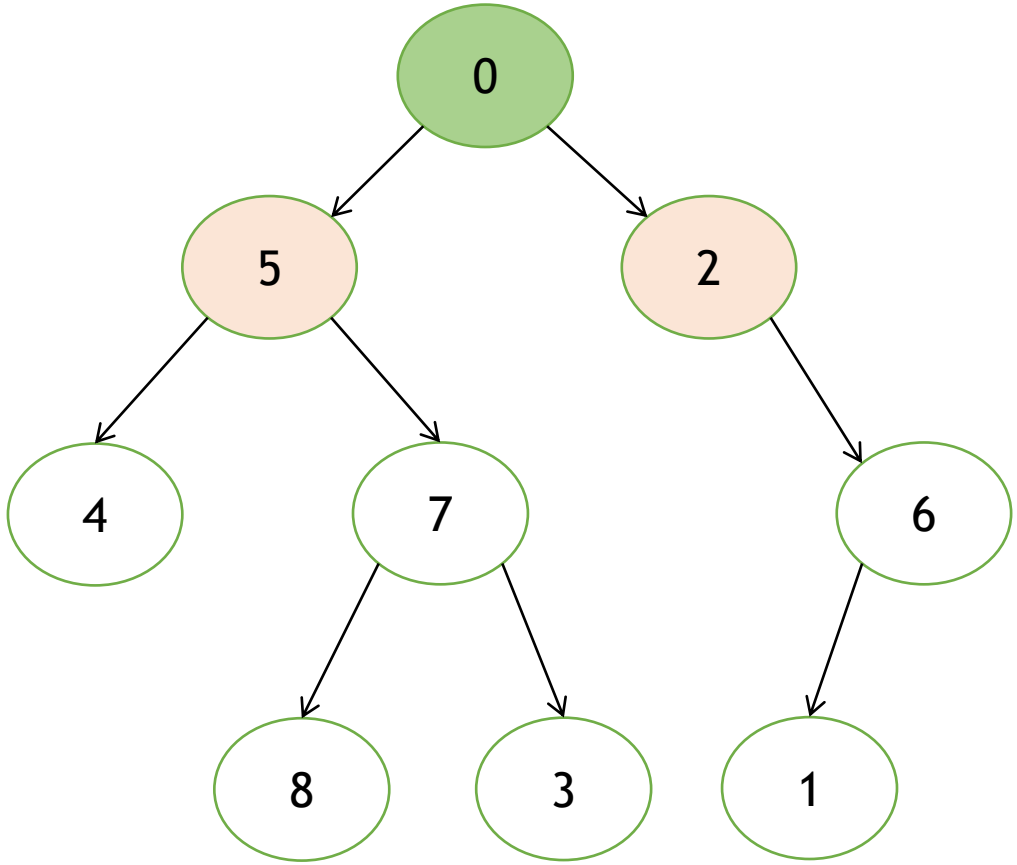
Nó	Ordem de visitação
0	
1	
2	
3	
4	
5	
6	
7	
8	



Busca em Largura (BFS)

Fila = {5, 2}

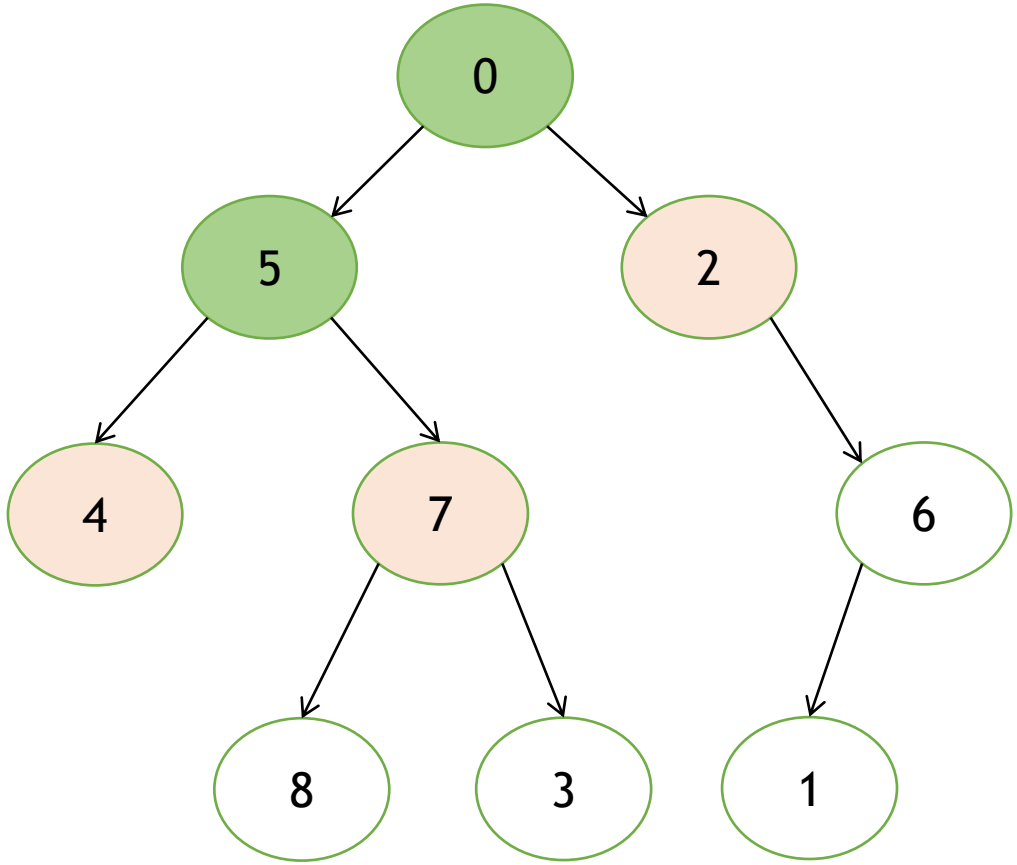
Nó	Ordem de visitação
0	1
1	
2	
3	
4	
5	
6	
7	
8	



Busca em Largura (BFS)

Fila = {2, 4, 7}

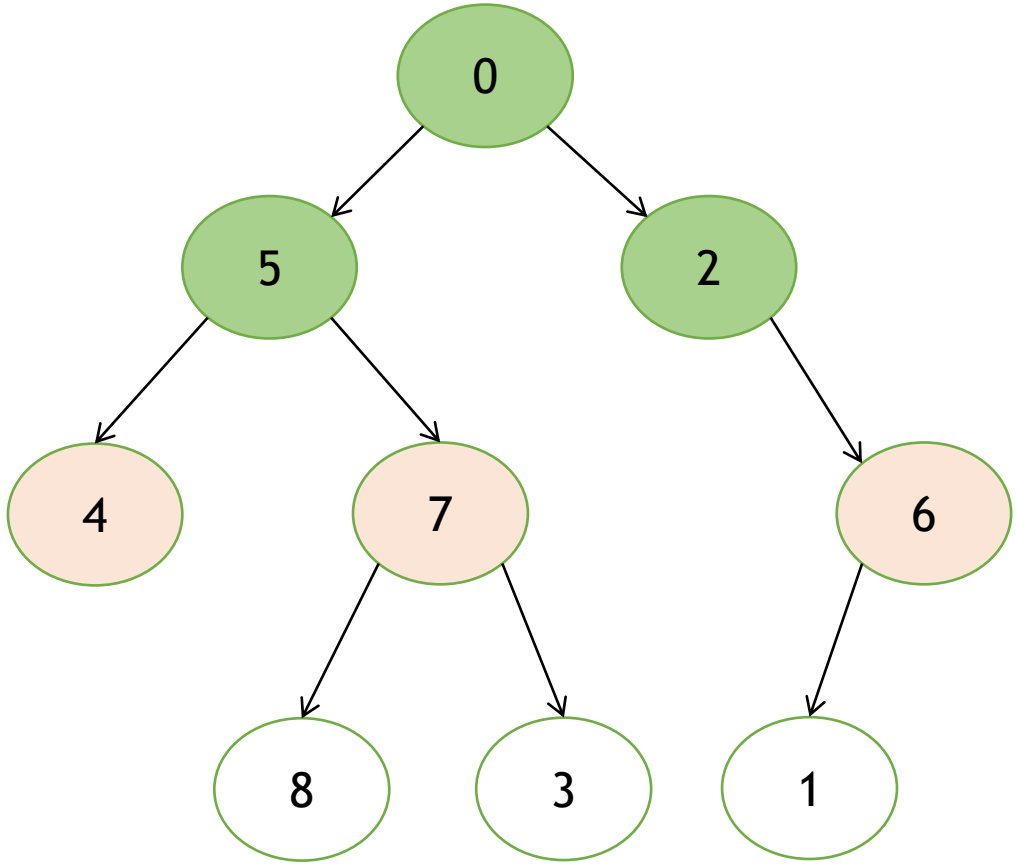
Nó	Ordem de visitação
0	1
1	
2	
3	
4	
5	2
6	
7	
8	



Busca em Largura (BFS)

Fila = {4, 7, 6}

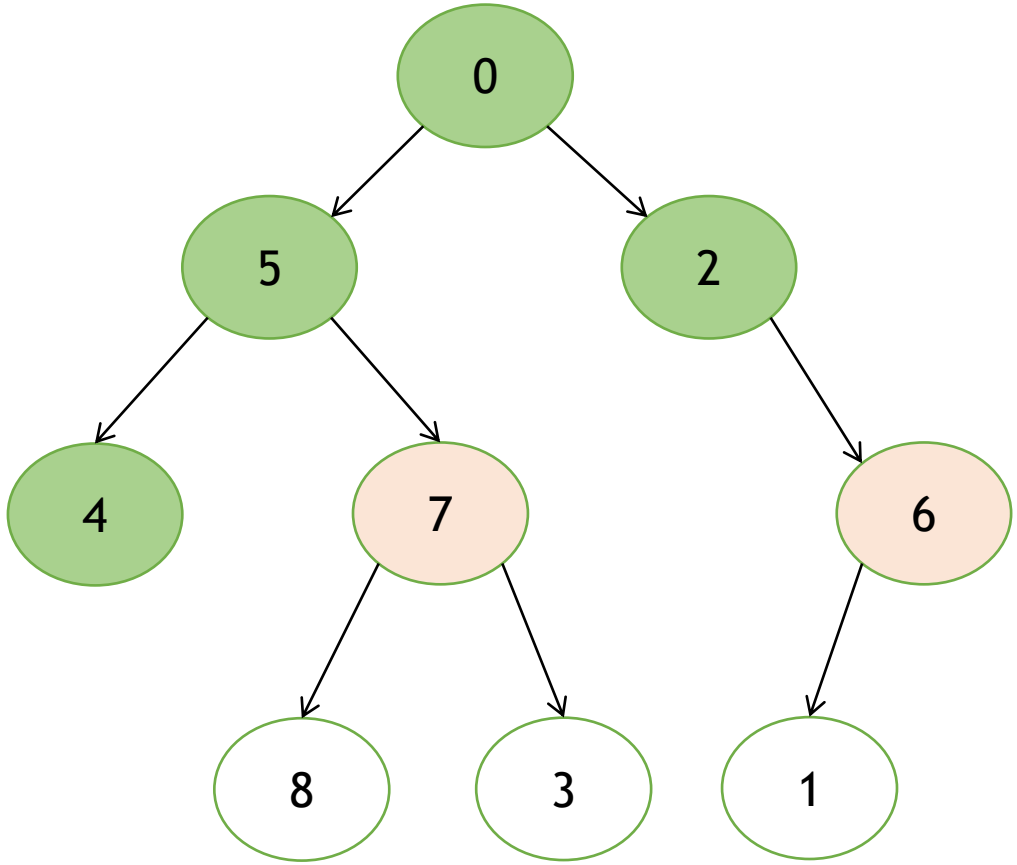
Nó	Ordem de visitação
0	1
1	
2	3
3	
4	
5	2
6	
7	
8	



Busca em Largura (BFS)

Fila = {7, 6}

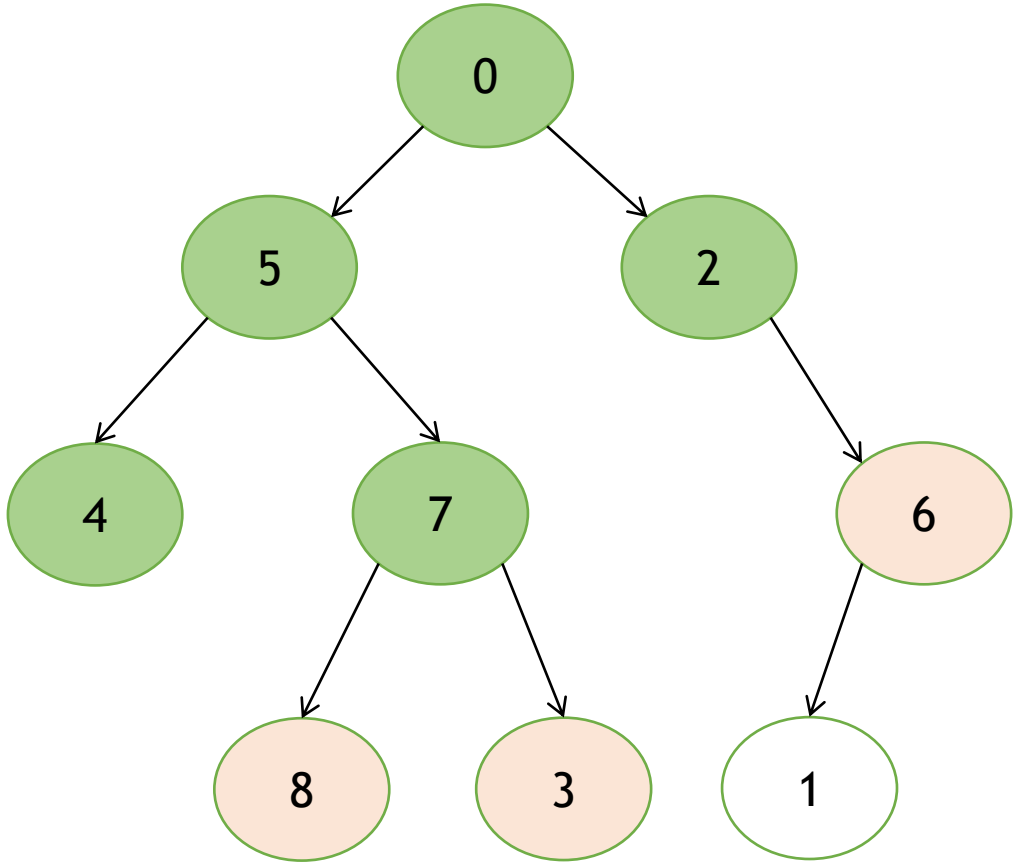
Nó	Ordem de visitação
0	1
1	
2	3
3	
4	4
5	2
6	
7	
8	



Busca em Largura (BFS)

Fila = {6, 8, 3}

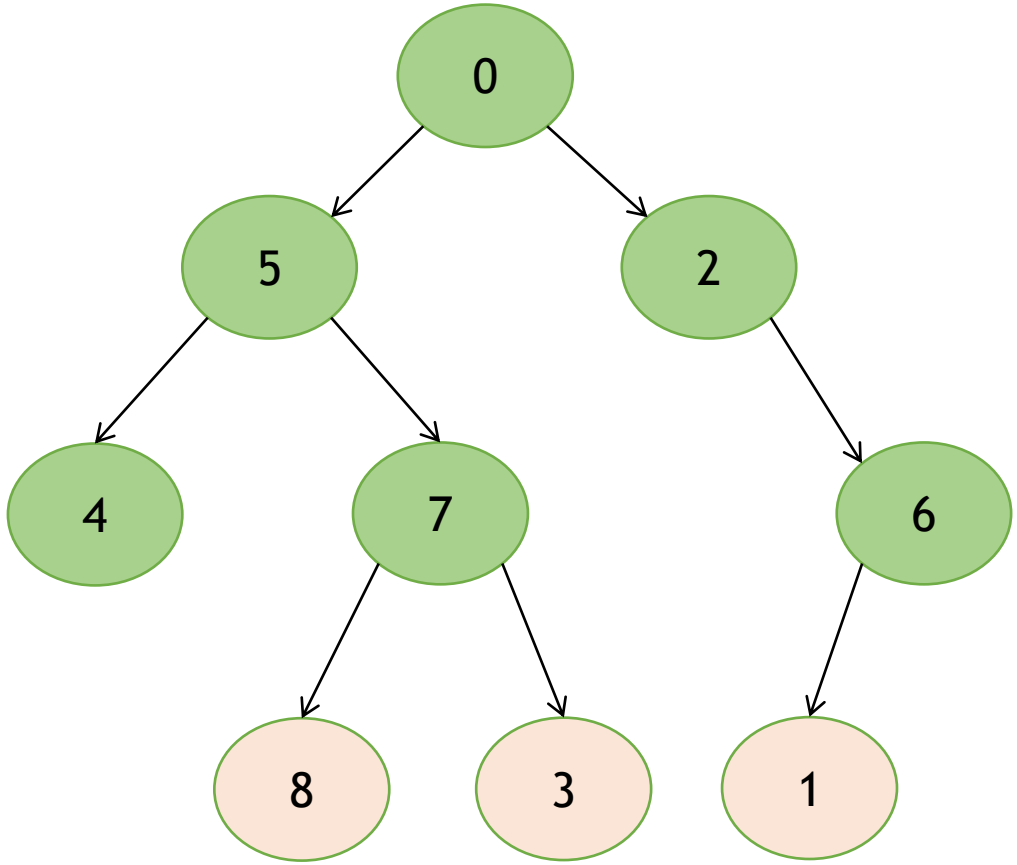
Nó	Ordem de visitação
0	1
1	
2	3
3	
4	4
5	2
6	
7	5
8	



Busca em Largura (BFS)

Fila = {8, 3, 1}

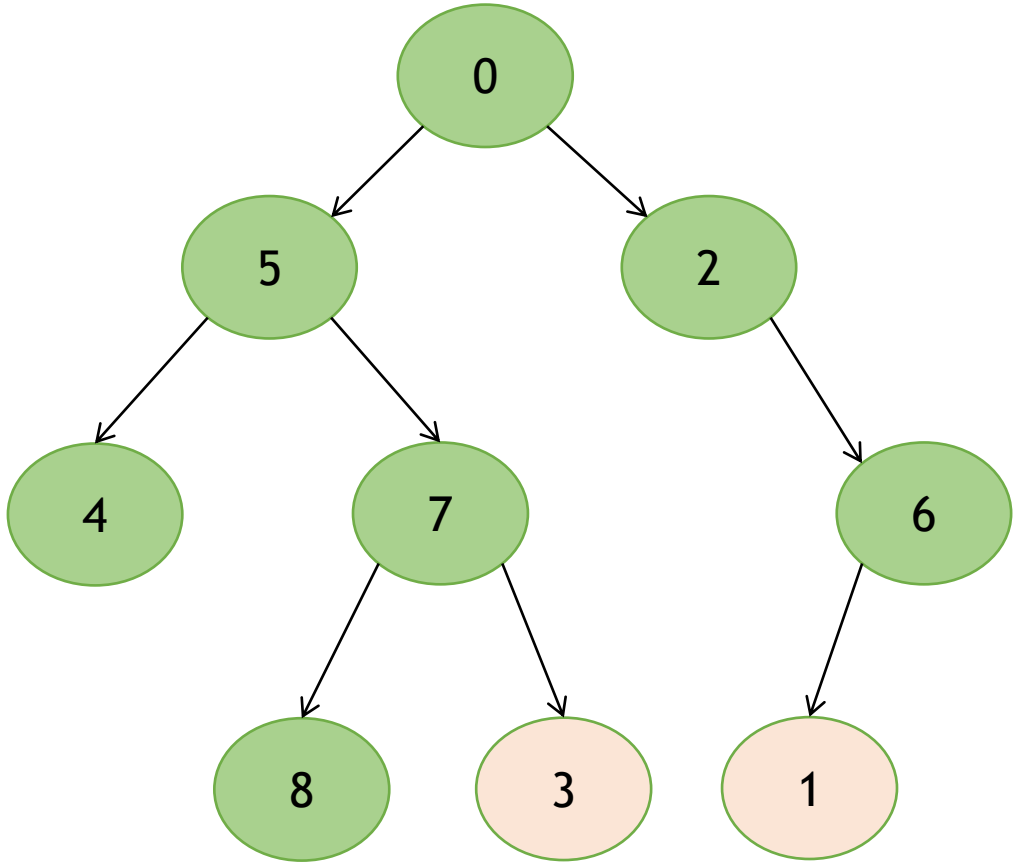
Nó	Ordem de visitação
0	1
1	
2	3
3	
4	4
5	2
6	6
7	5
8	



Busca em Largura (BFS)

Fila = {3, 1}

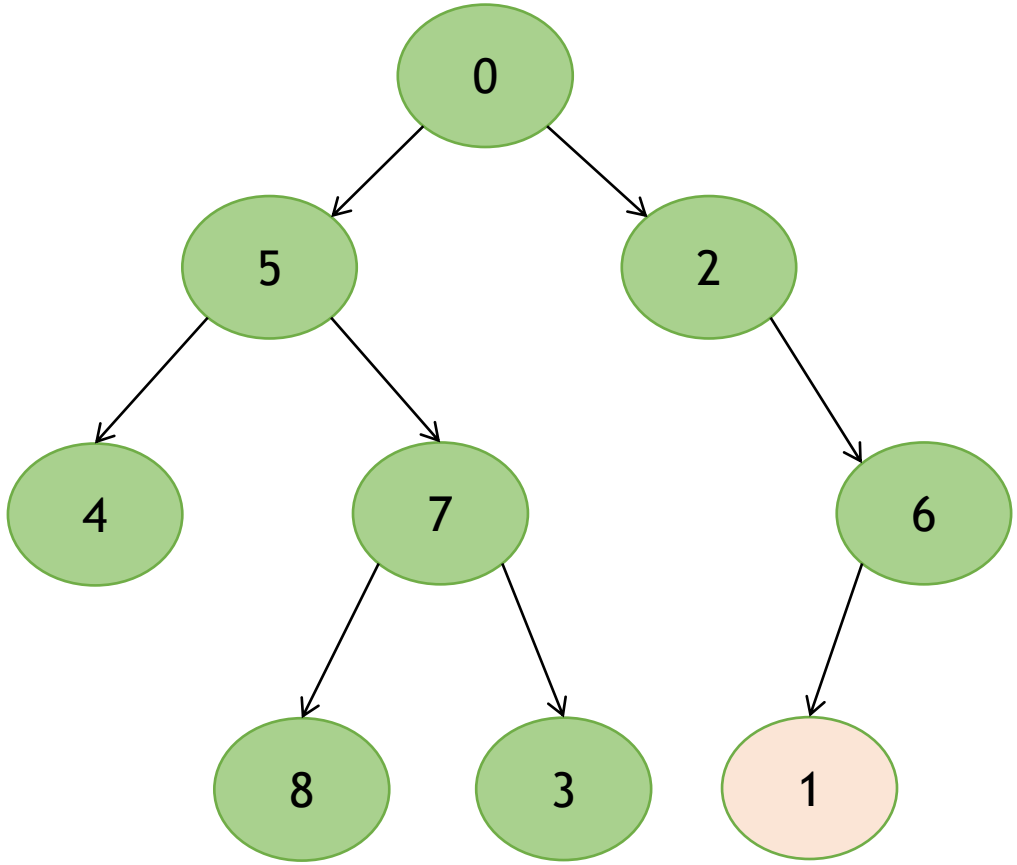
Nó	Ordem de visitação
0	1
1	
2	3
3	
4	4
5	2
6	6
7	5
8	7



Busca em Largura (BFS)

Fila = {1}

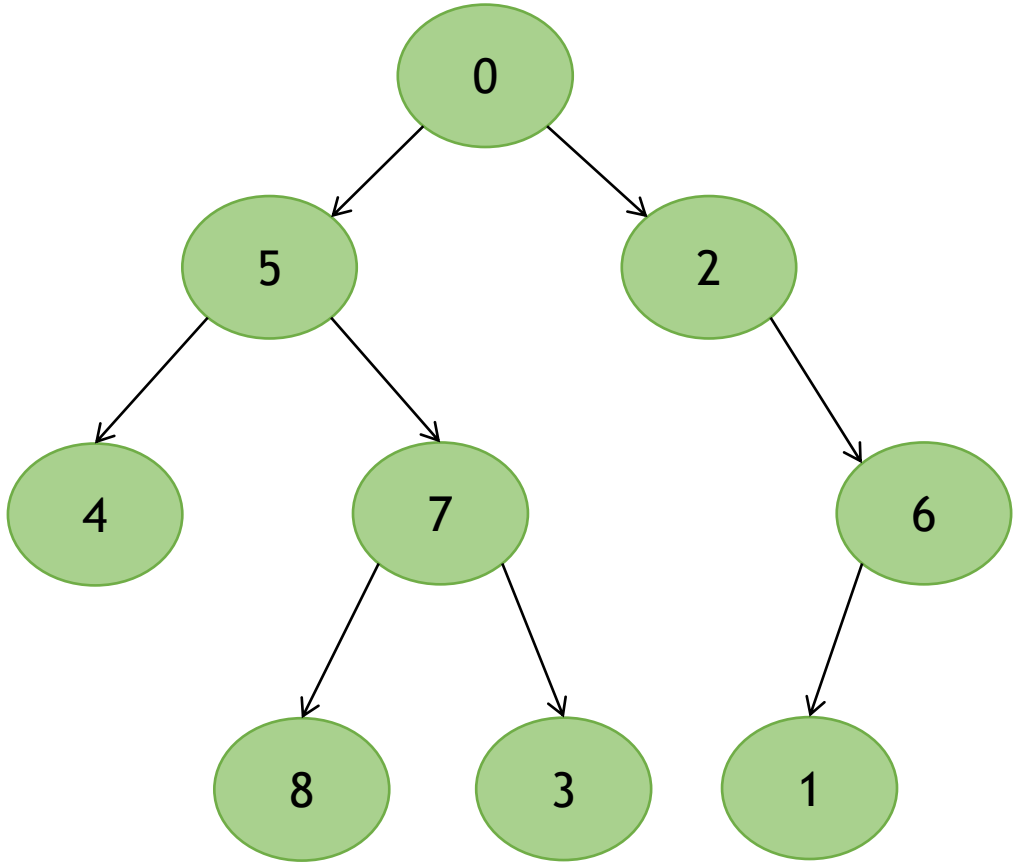
Nó	Ordem de visitação
0	1
1	
2	3
3	8
4	4
5	2
6	6
7	5
8	7



Busca em Largura (BFS)

Fila = {}

Nó	Ordem de visitação
0	1
1	9
2	3
3	8
4	4
5	2
6	6
7	5
8	7



Árvores + PD

- É comum a utilização de programação dinâmica para calcular certas informações durante uma varredura em uma árvore.
- Exemplo: determinar a quantidade de nós em cada sub-árvore

$$qtde(u) = \begin{cases} 1, & se\ u\ \text{é}\ folha \\ 1 + \sum_{v} qtde(v) & \forall v \mid v\ \text{é}\ filho\ de\ u \end{cases}$$

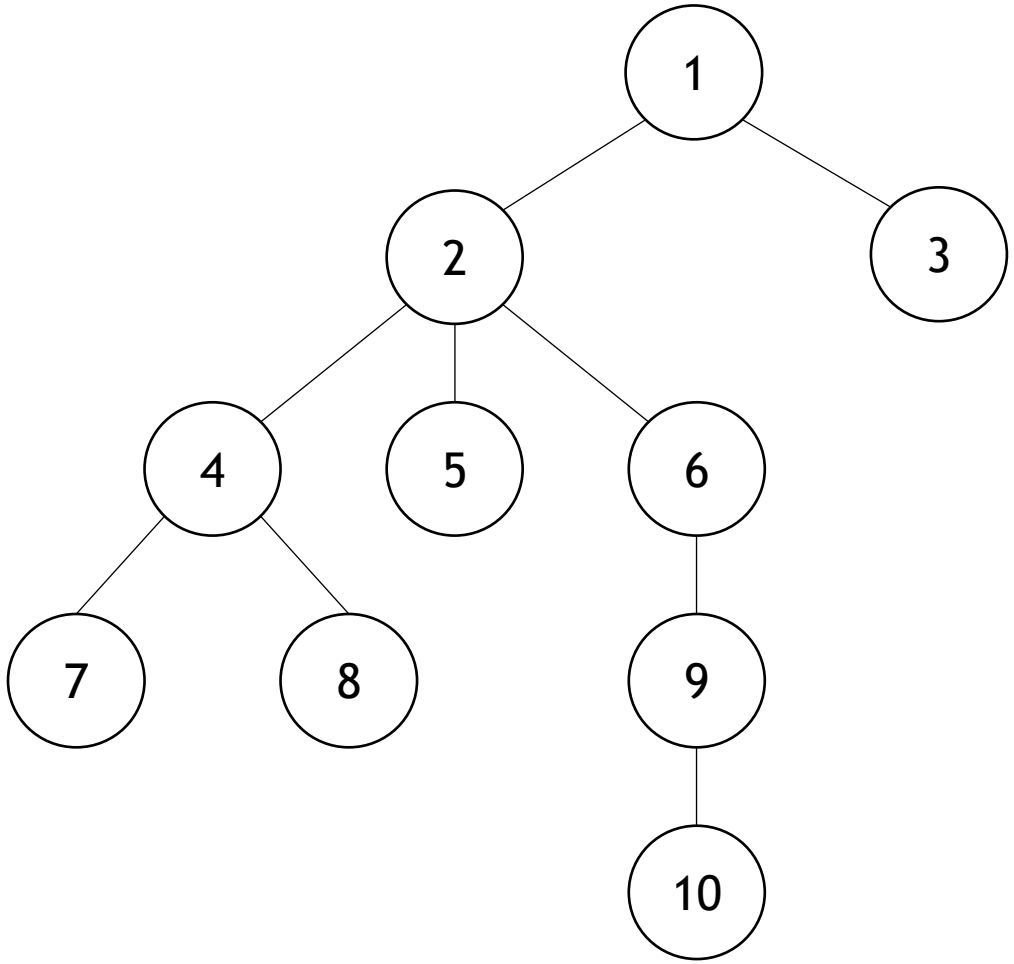
Árvores + PD

- Quantidade de nós na subárvore u

```
int dfs(int v)
{
    count[v] = 1;
    for(auto u : arvore[v])
        count[v] += dfs(u);
    return count[v];
}
```

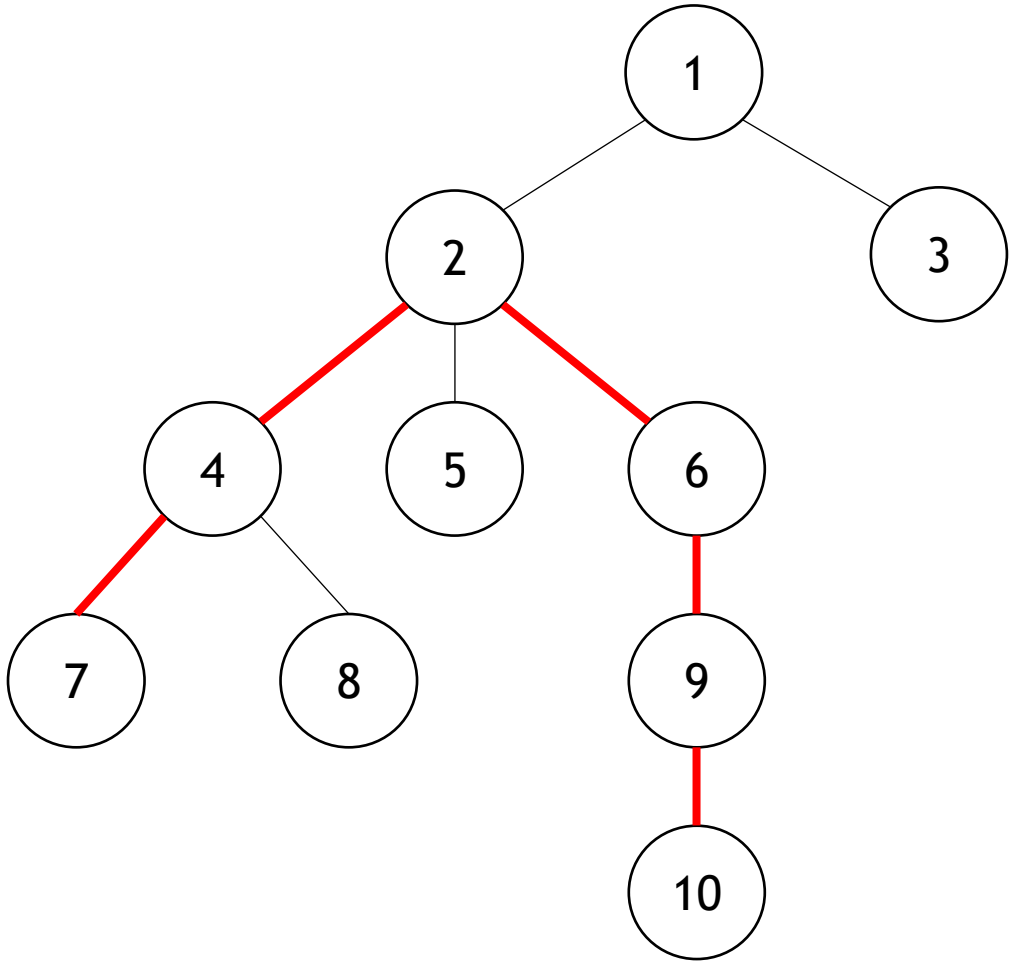
Diâmetro

- O **diâmetro** de uma árvore é o maior comprimento de um caminho entre dois nós da árvore.
- SPOJ PT07Z – *Longest path in a tree*



Diâmetro

- O **diâmetro** de uma árvore é o maior comprimento de um caminho entre dois nós da árvore.
- Diâmetro da árvore ao lado: 5



Diâmetro

- Para resolver este problema também usaremos PD.
- Uma observação importante é que qualquer caminho na árvore possui um “ponto mais alto”, o nó com menor nível que pertence a este caminho.
- Sendo assim, para cada nó, podemos determinar o maior caminho em que ele é o ponto mais alto.
  - A estratégia para fazer isto é simples: sabendo a altura das sub-árvores de cada filho, basta selecionar as duas maiores alturas.

Diâmetro

- Para resolver isso efetivamente, vamos calcular para cada nó as seguintes informações
  - height[u] = altura da sub-árvore u, ou simplesmente o comprimento máximo do caminho de u a qualquer folha.
  - maxLength[u] = o comprimento do maior caminho em que u é o ponto mais alto

Diâmetro

$$height(u) = \begin{cases} 1, & se\ u\ \text{é}\ folha \\ 1 + \max\{height(v)\} & \forall v \mid v\ \text{é}\ filho\ de\ u \end{cases}$$

$$maxLength(u) = \begin{cases} 0, & se\ u\ \text{é}\ folha \\ 1 + \max\{height(v_1) + height(v_2)\} & \forall v_1, v_2 \mid v_1\ e\ v_2\ \text{são}\ filhos\ distintos\ de\ u \end{cases}$$

Diâmetro

```
vector<int> arvore[MAXN];

int height[MAXN];
int maxLenght[MAXN];
int diameter=0;

void aresta(int u, int v)
{
    arvore[u].push_back(v);
    arvore[v].push_back(u);
}
```

Diâmetro

```
int dfs(int u, int pai){
    height[u] = 1;
    int maxA = 0, maxB = 0;
    for(auto v: arvore[u]){
        if (v == pai)
            continue;
        height[u] = max(height[u], 1 + dfs(v, u));
        if (height[v] > maxA){
            maxB = maxA;
            maxA = height[v];
        } else if (height[v] > maxB)
            maxB = height[v];
    }
    maxLenght[u] = maxA + maxB;
    diameter = max(diameter, maxLenght[u]);
    return height[u];
}
```

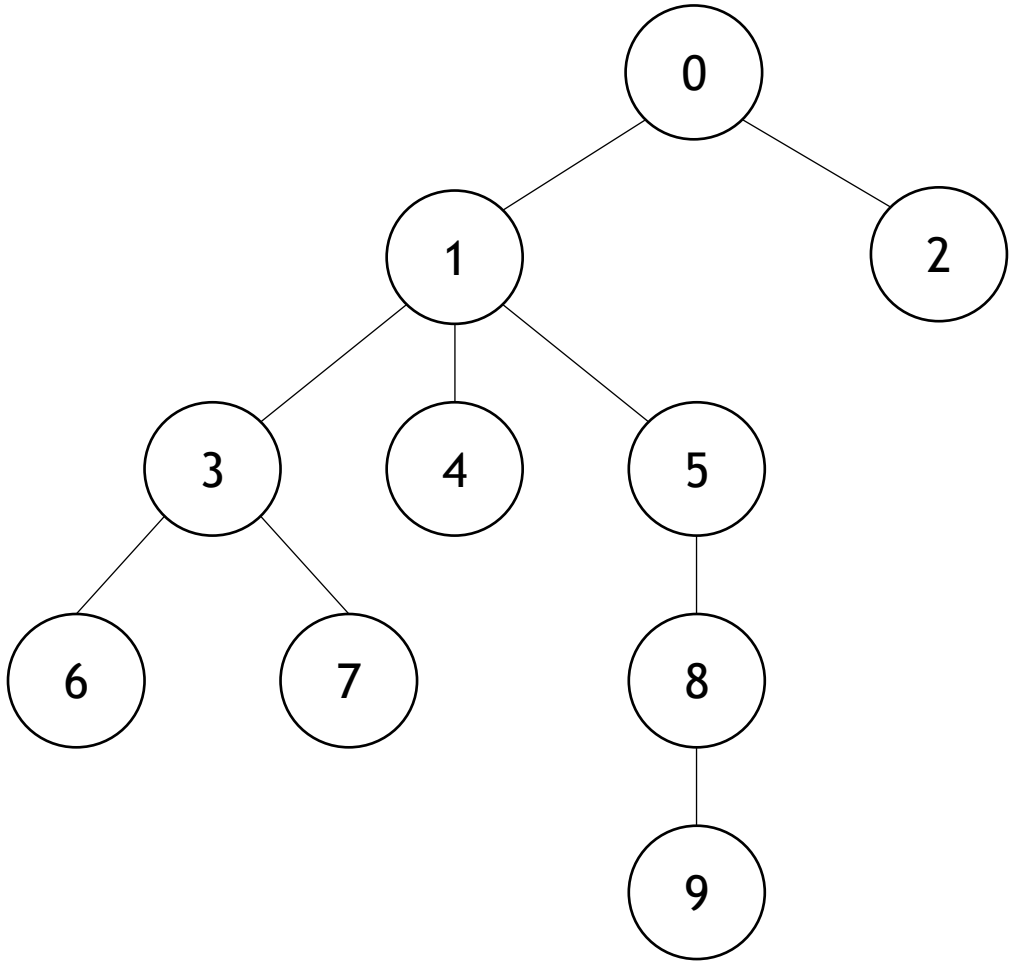
Diâmetro

```
int main(){
    int n, a, b;
    cin >> n;
    for(int i = 1; i < n; i++){
        cin >> a >> b;
        aresta(a-1, b-1);
    }
    dfs(0, -1);
    cout << diameter << endl;
}
```

Diâmetro

diameter = 0

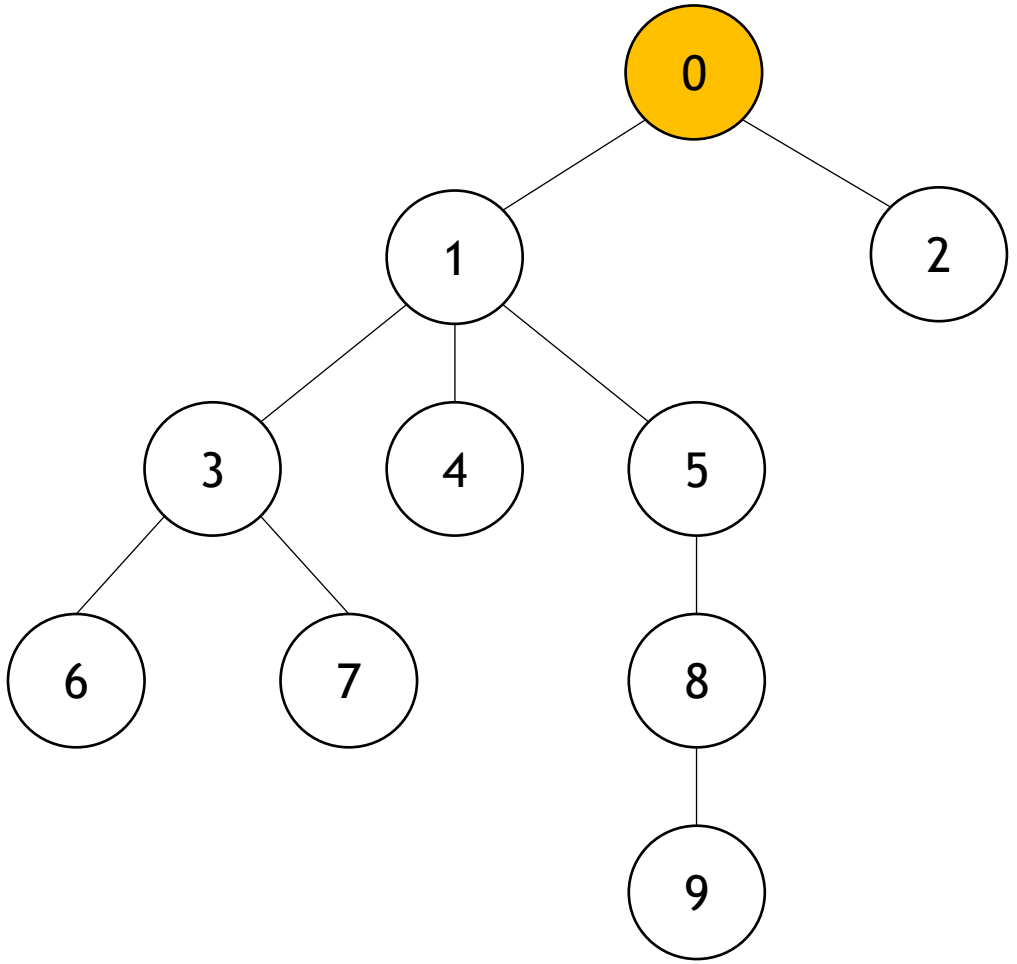
u	height[u]	maxLenght[u]
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		



Diâmetro

diameter = 0

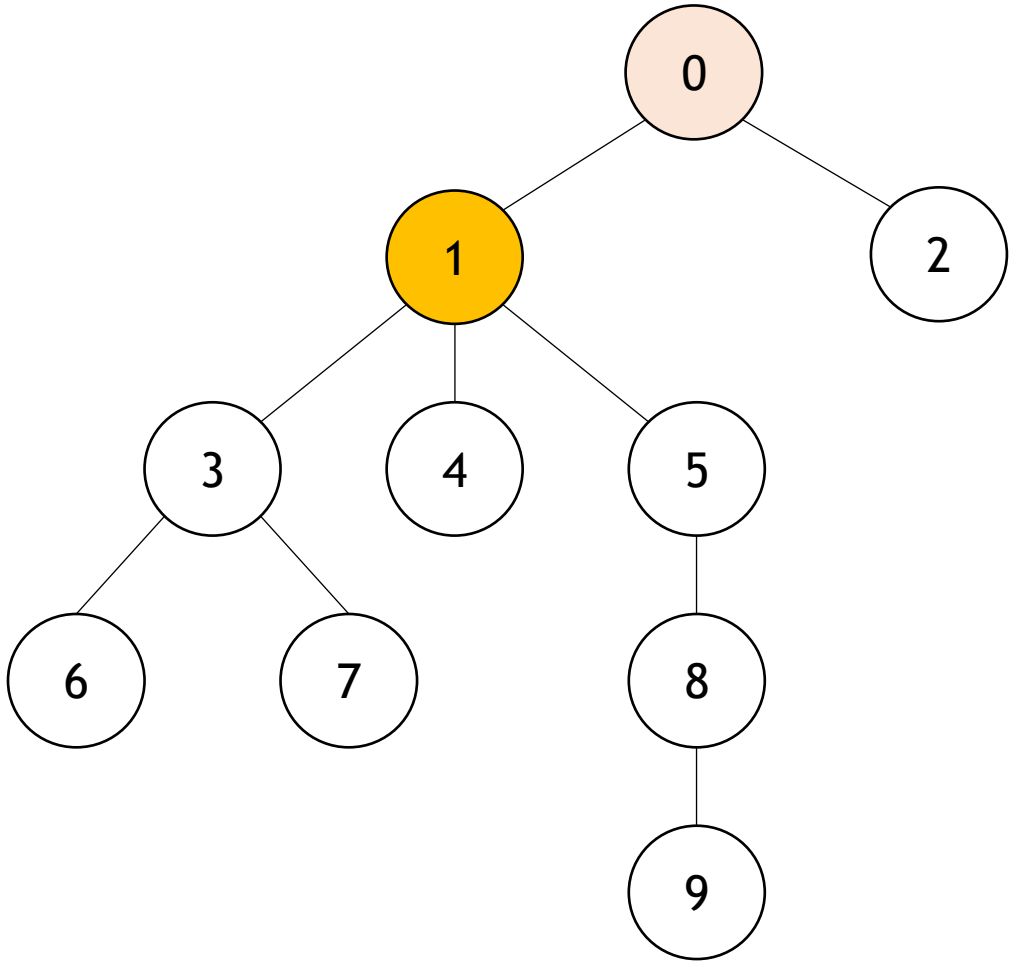
u	height[u]	maxLenght[u]
0	1	
1		
2		
3		
4		
5		
6		
7		
8		
9		



Diâmetro

diameter = 0

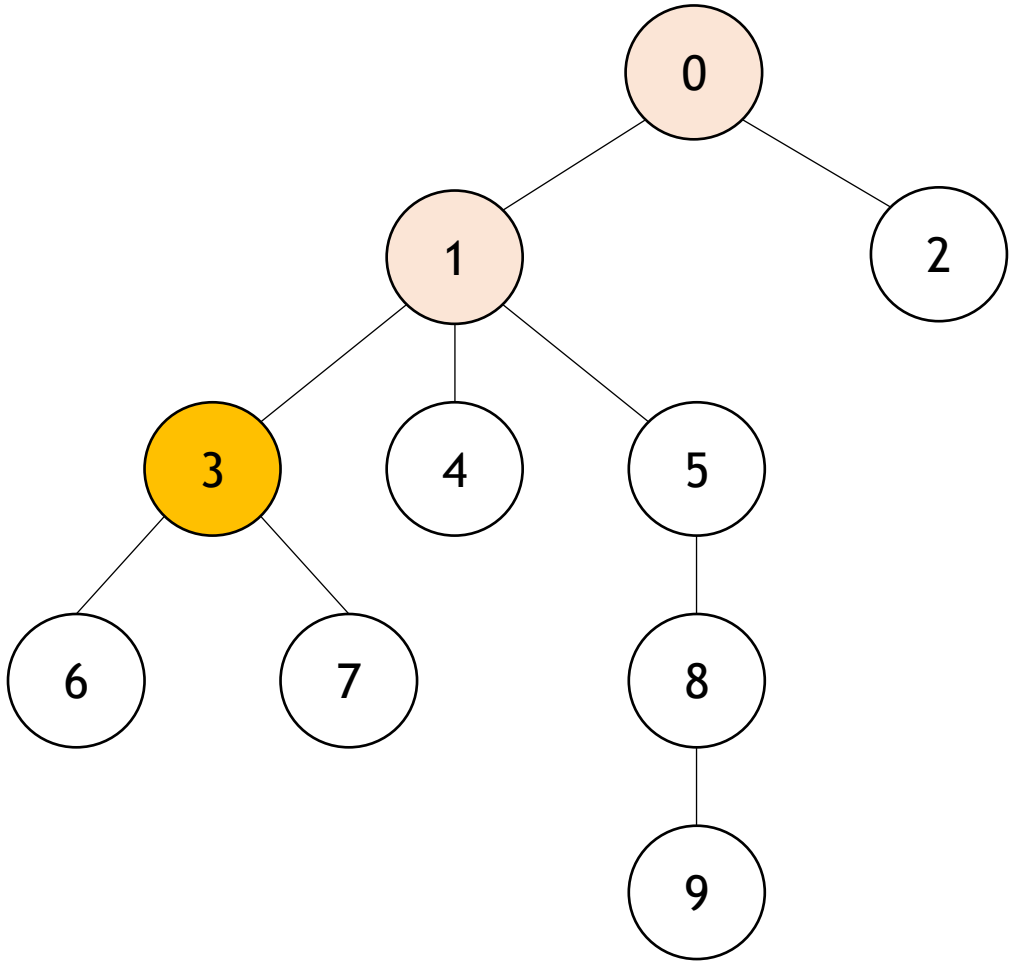
u	height[u]	maxLenght[u]
0	1	
1	1	
2		
3		
4		
5		
6		
7		
8		
9		



Diâmetro

diameter = 0

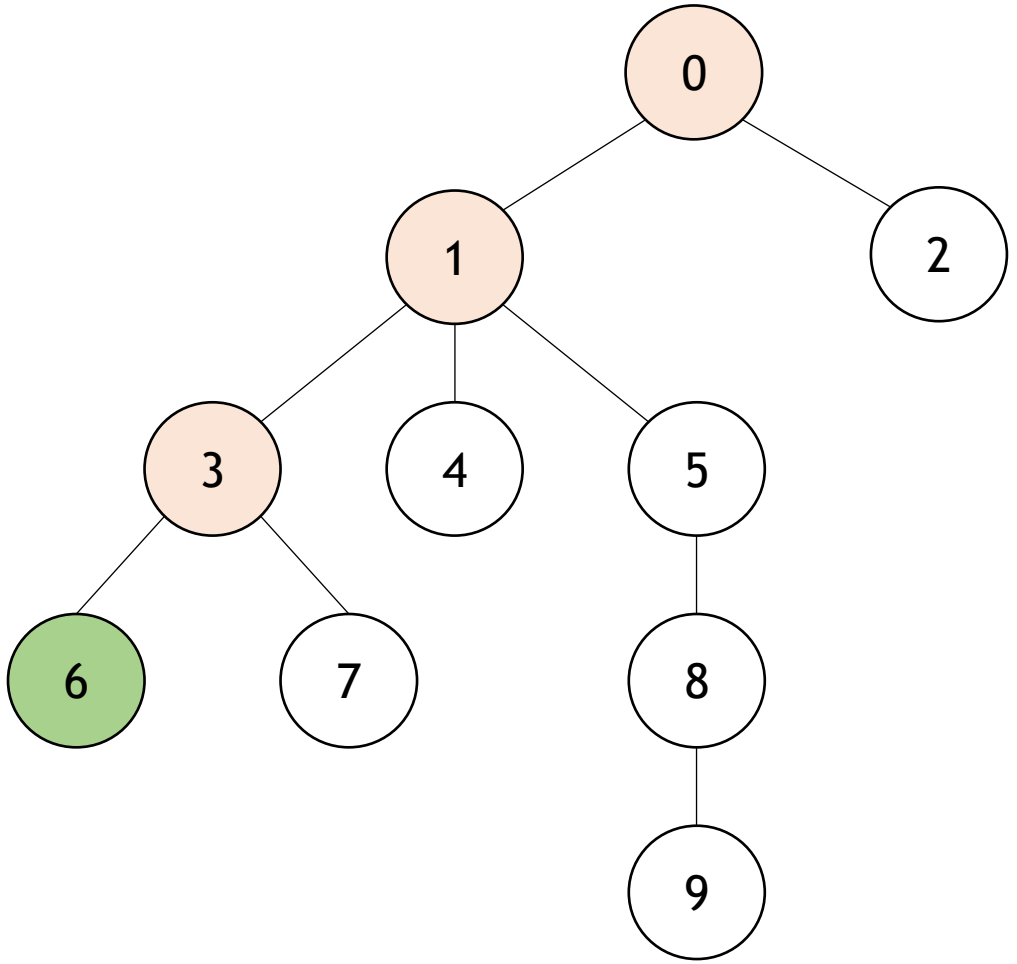
u	height[u]	maxLenght[u]
0	1	
1	1	
2		
3	1	
4		
5		
6		
7		
8		
9		



Diâmetro

diameter = 0

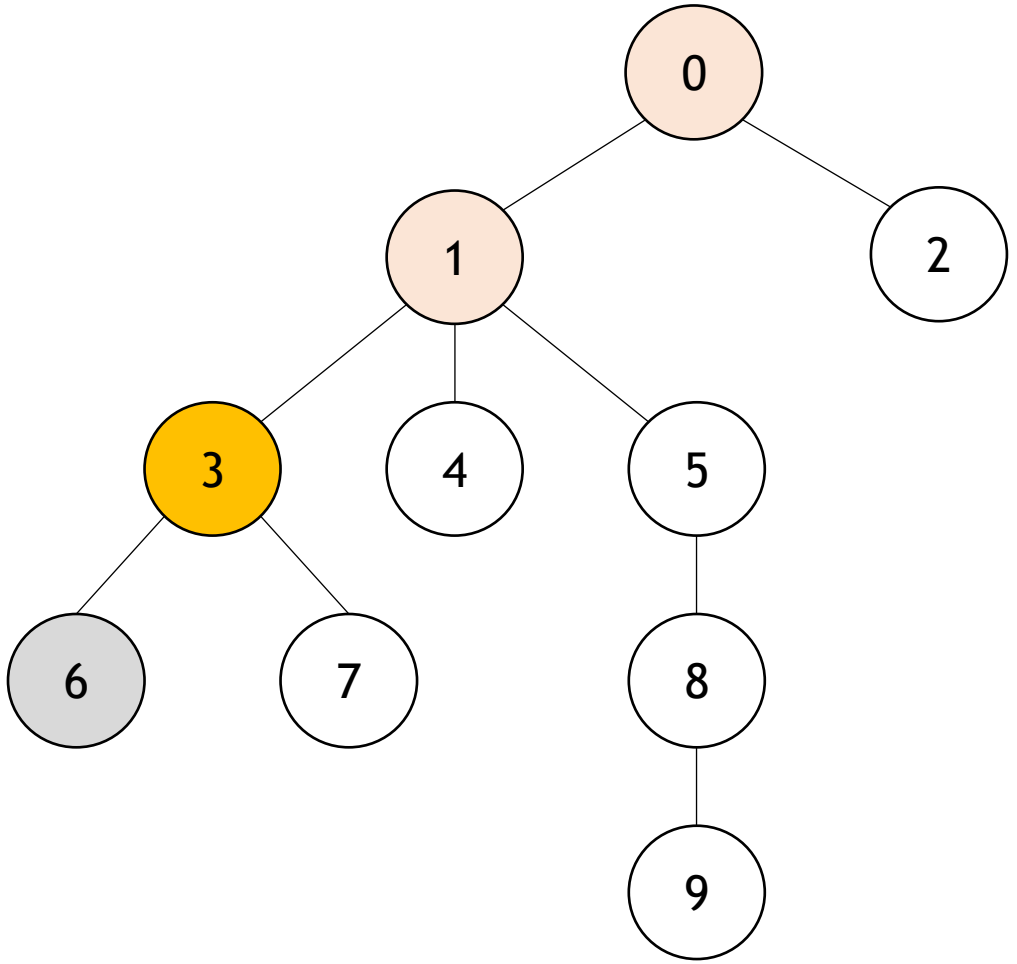
u	height[u]	maxLenght[u]
0	1	
1	1	
2		
3	1	
4		
5		
6	1	0
7		
8		
9		



Diâmetro

diameter = 0

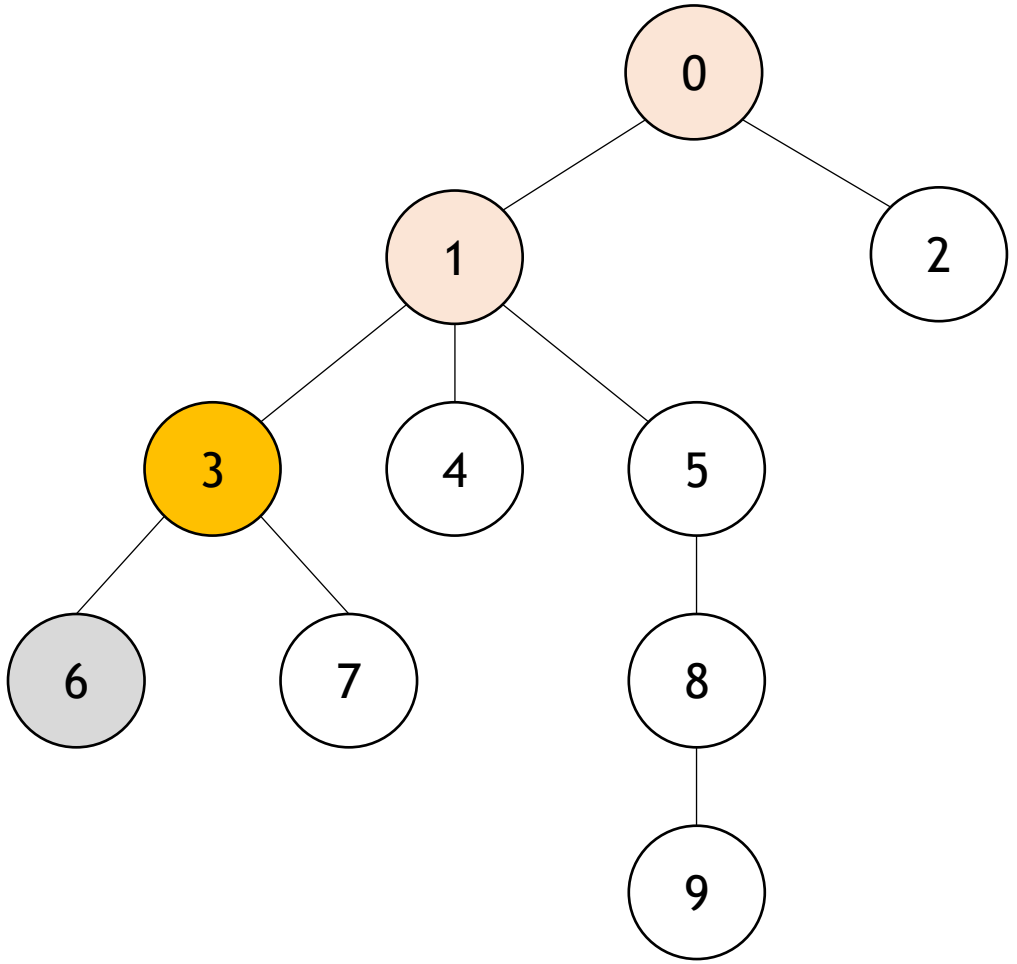
u	height[u]	maxLenght[u]
0	1	
1	1	
2		
3	max{1, 1 + 1}	
4		
5		
6	1	0
7		
8		
9		



Diâmetro

diameter = 0

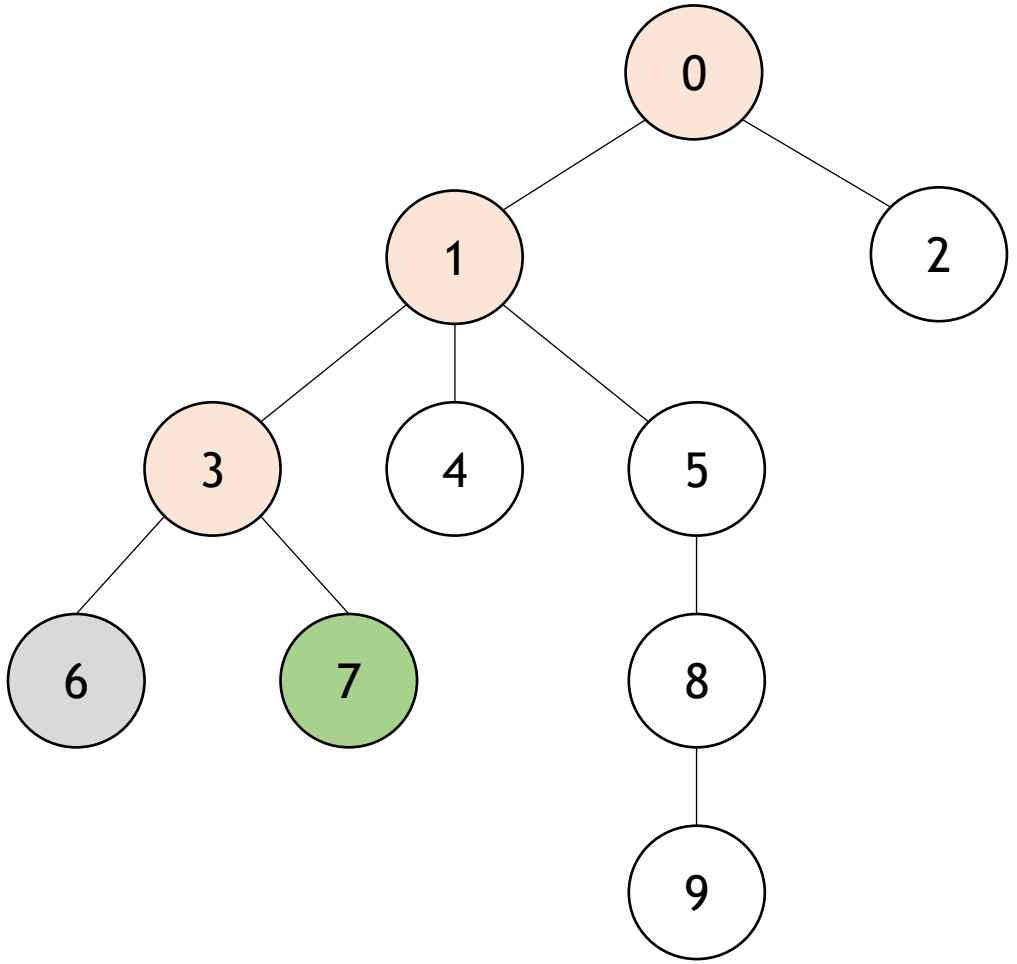
u	height[u]	maxLenght[u]
0	1	
1	1	
2		
3	2	
4		
5		
6	1	0
7		
8		
9		



Diâmetro

diameter = 0

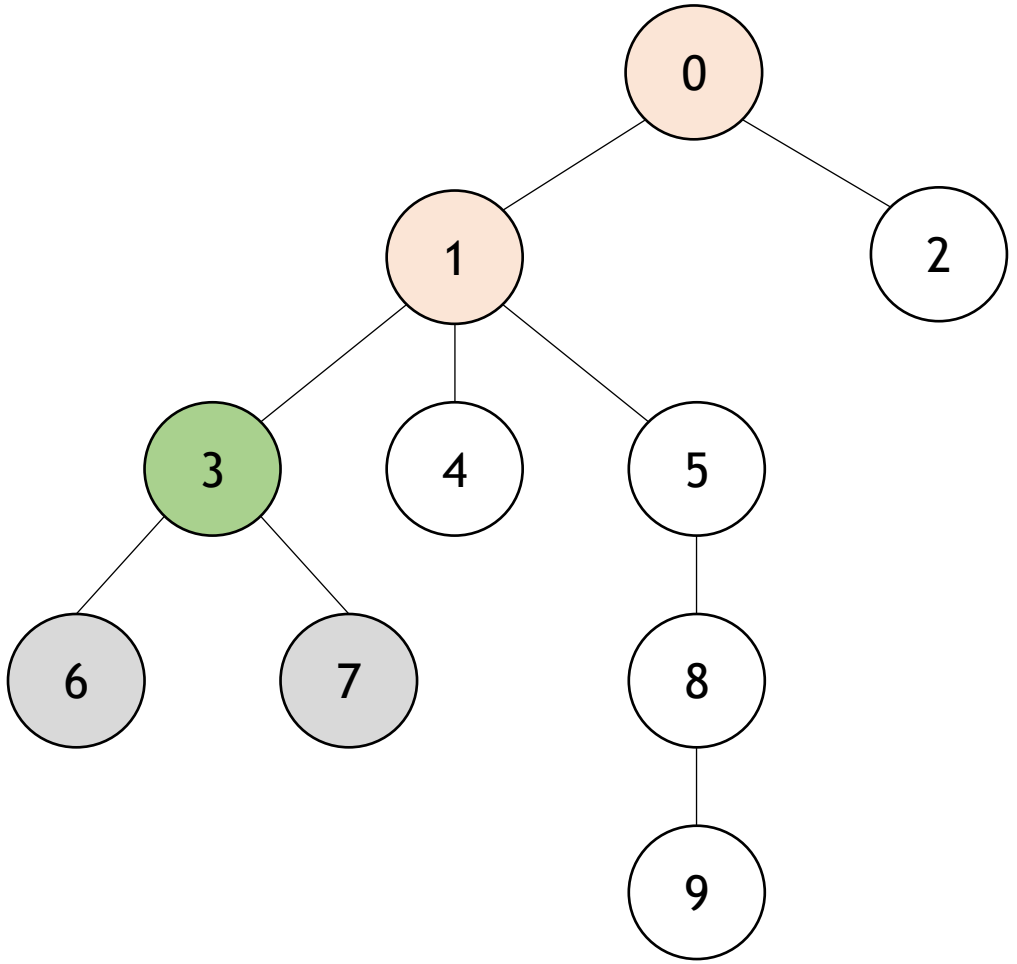
u	height[u]	maxLenght[u]
0	1	
1	1	
2		
3	2	
4		
5		
6	1	0
7	1	0
8		
9		



Diâmetro

diameter = 0

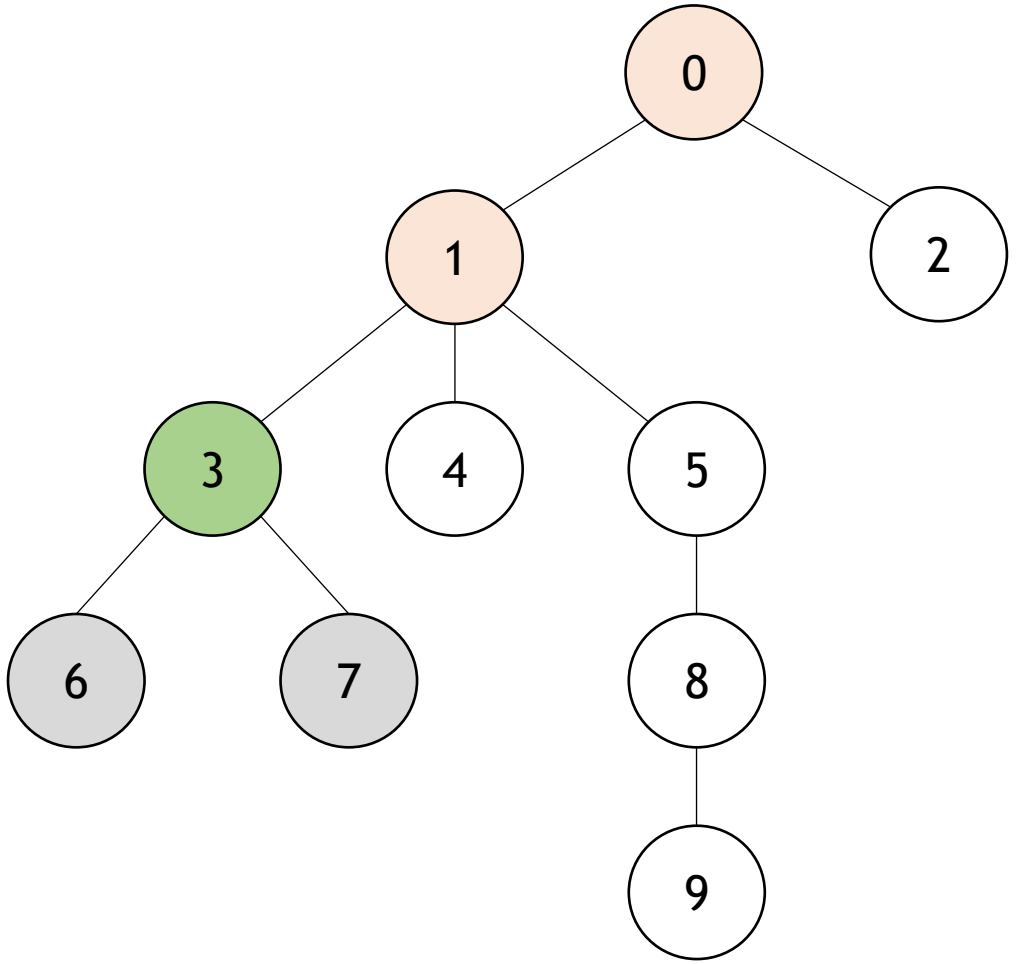
u	height[u]	maxLenght[u]
0	1	
1	1	
2		
3	max{2, 1 + 1}	1 + 1
4		
5		
6	1	0
7	1	0
8		
9		



Diâmetro

diameter = 2

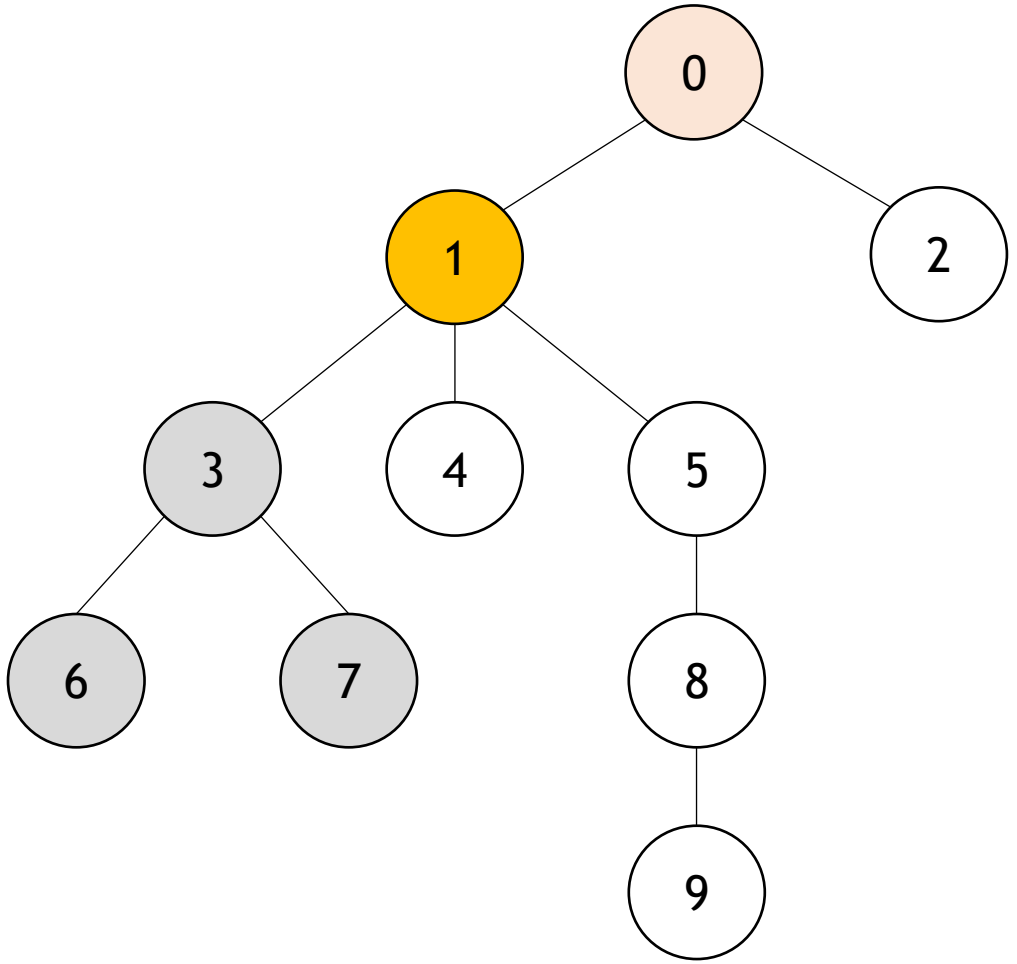
u	height[u]	maxLenght[u]
0	1	
1	1	
2		
3	2	2
4		
5		
6	1	0
7	1	0
8		
9		



Diâmetro

diameter = 2

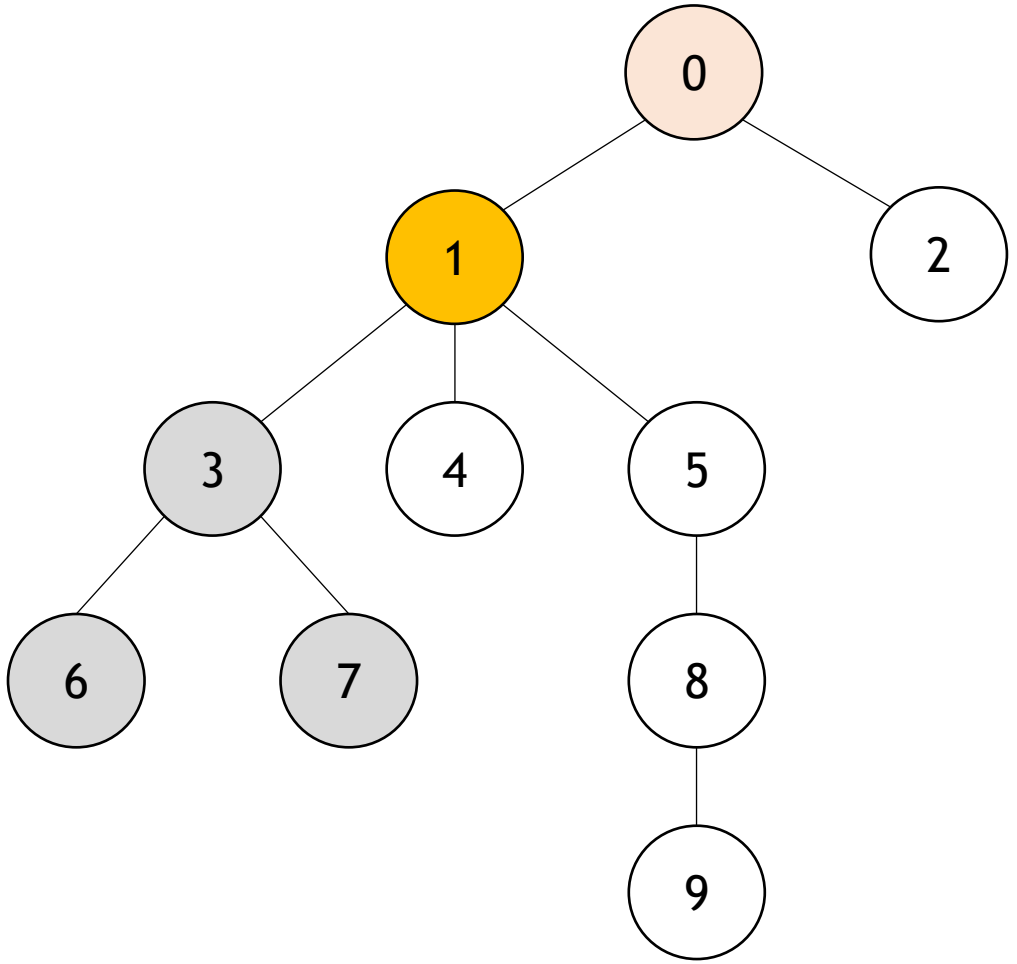
u	height[u]	maxLenght[u]
0	1	
1	max(1, 1 + 2)	
2		
3	2	2
4		
5		
6	1	0
7	1	0
8		
9		



Diâmetro

diameter = 2

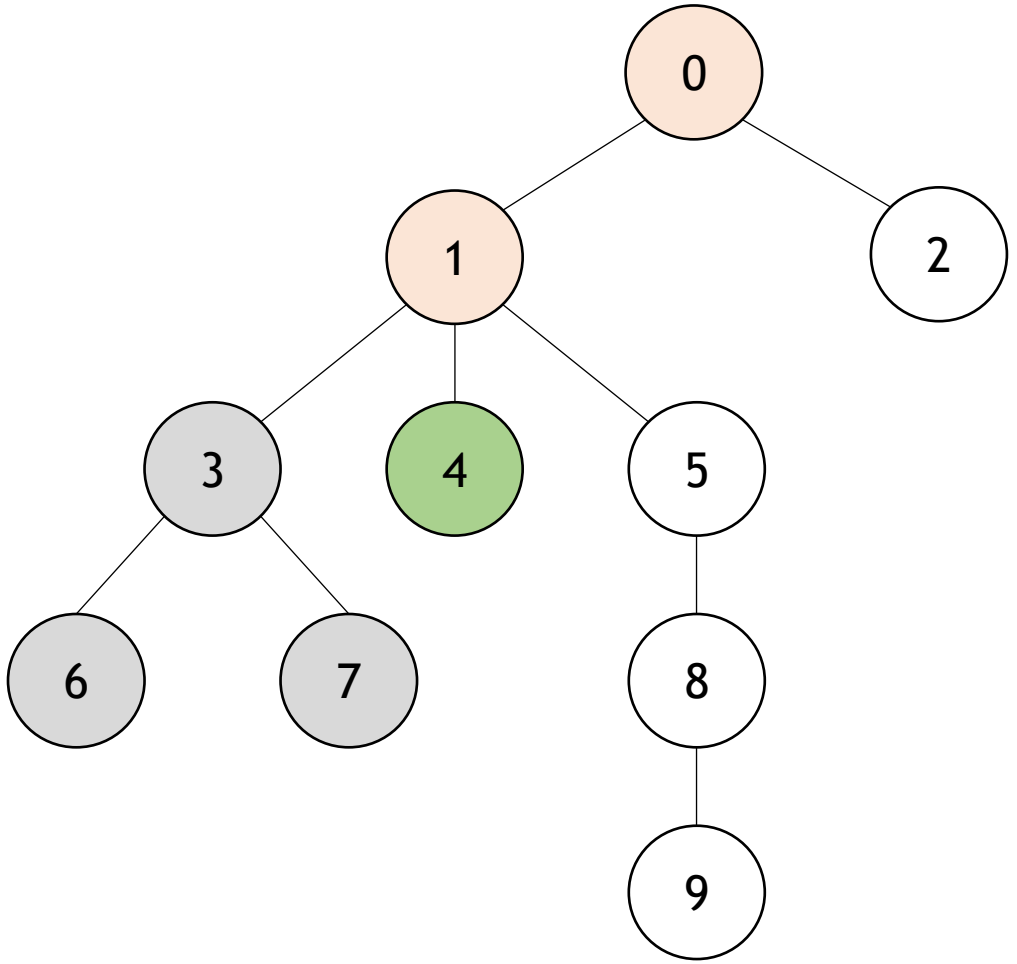
u	height[u]	maxLenght[u]
0	1	
1	3	
2		
3	2	2
4		
5		
6	1	0
7	1	0
8		
9		



Diâmetro

diameter = 2

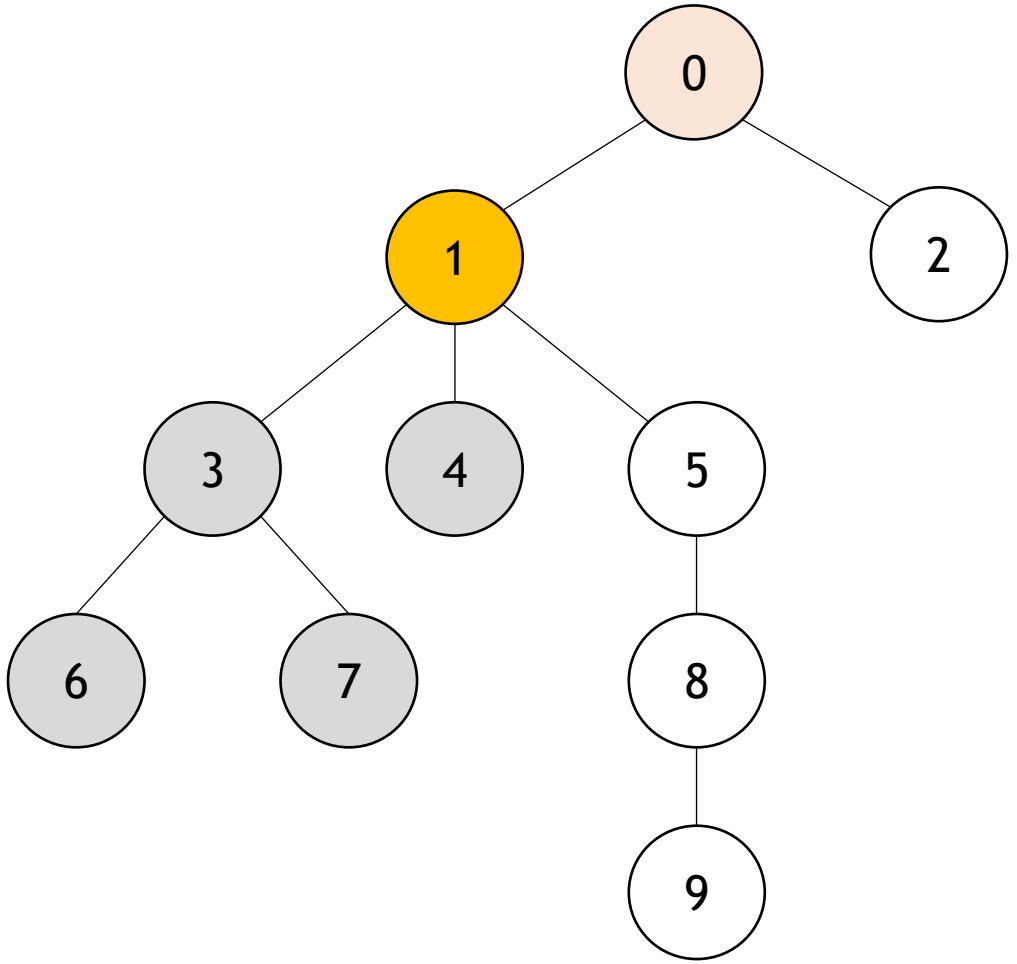
u	height[u]	maxLenght[u]
0	1	
1	3	
2		
3	2	2
4	1	0
5		
6	1	0
7	1	0
8		
9		



Diâmetro

diameter = 2

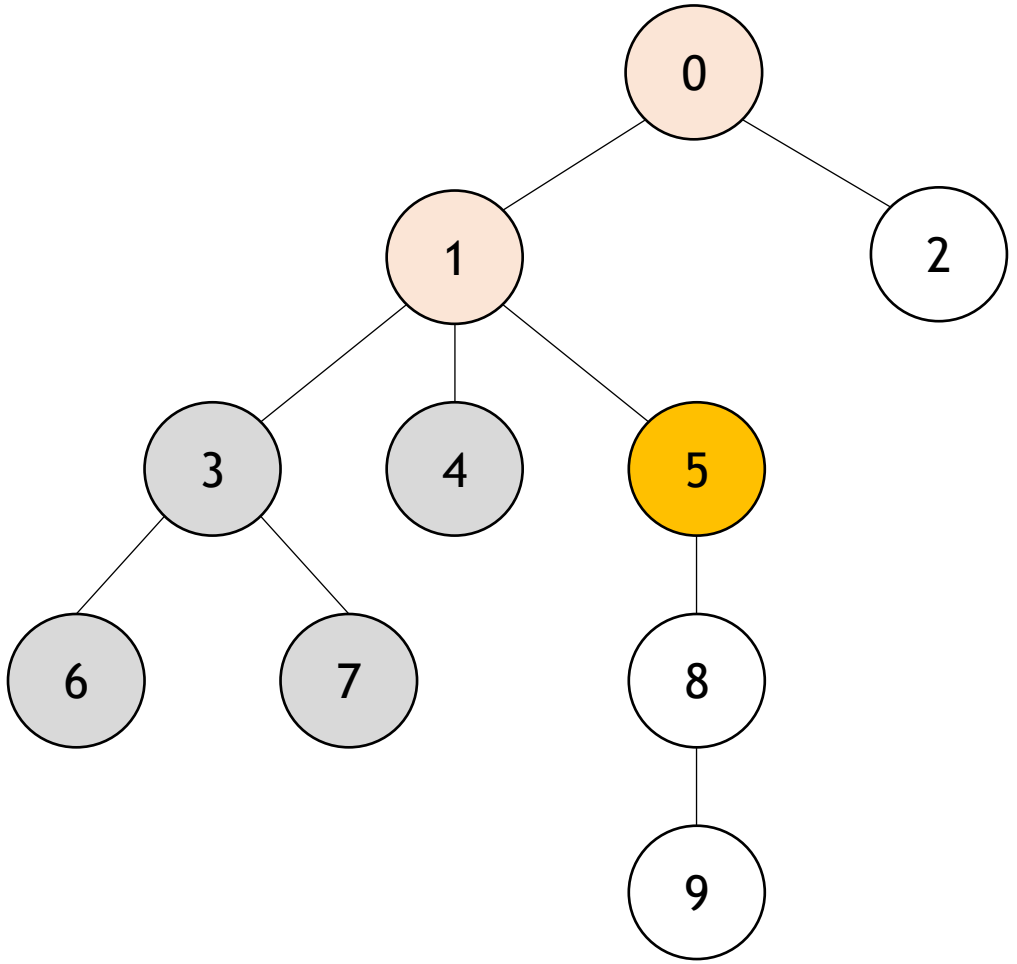
u	height[u]	maxLenght[u]
0	1	
1	max(3, 1+1)	
2		
3	2	2
4	1	0
5		
6	1	0
7	1	0
8		
9		



Diâmetro

diameter = 2

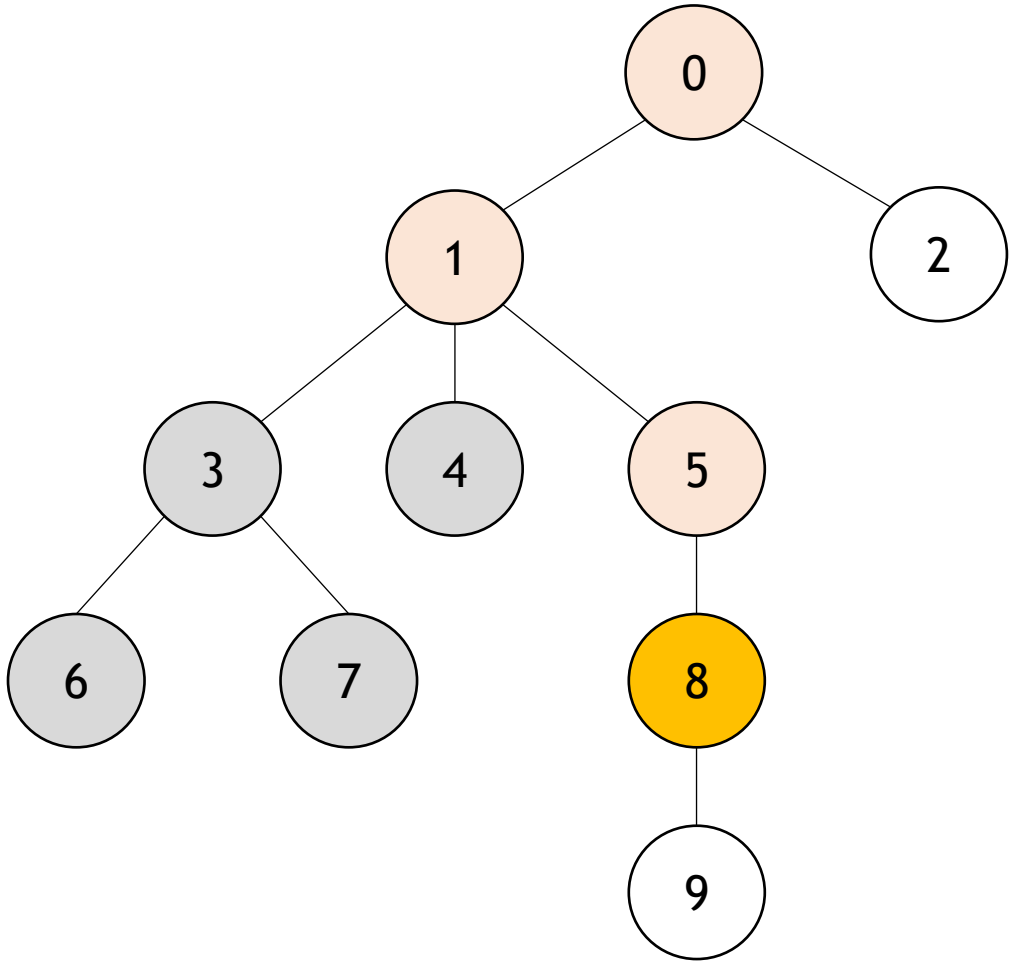
u	height[u]	maxLenght[u]
0	1	
1	3	
2		
3	2	2
4	1	0
5	1	
6	1	0
7	1	0
8		
9		



Diâmetro

diameter = 2

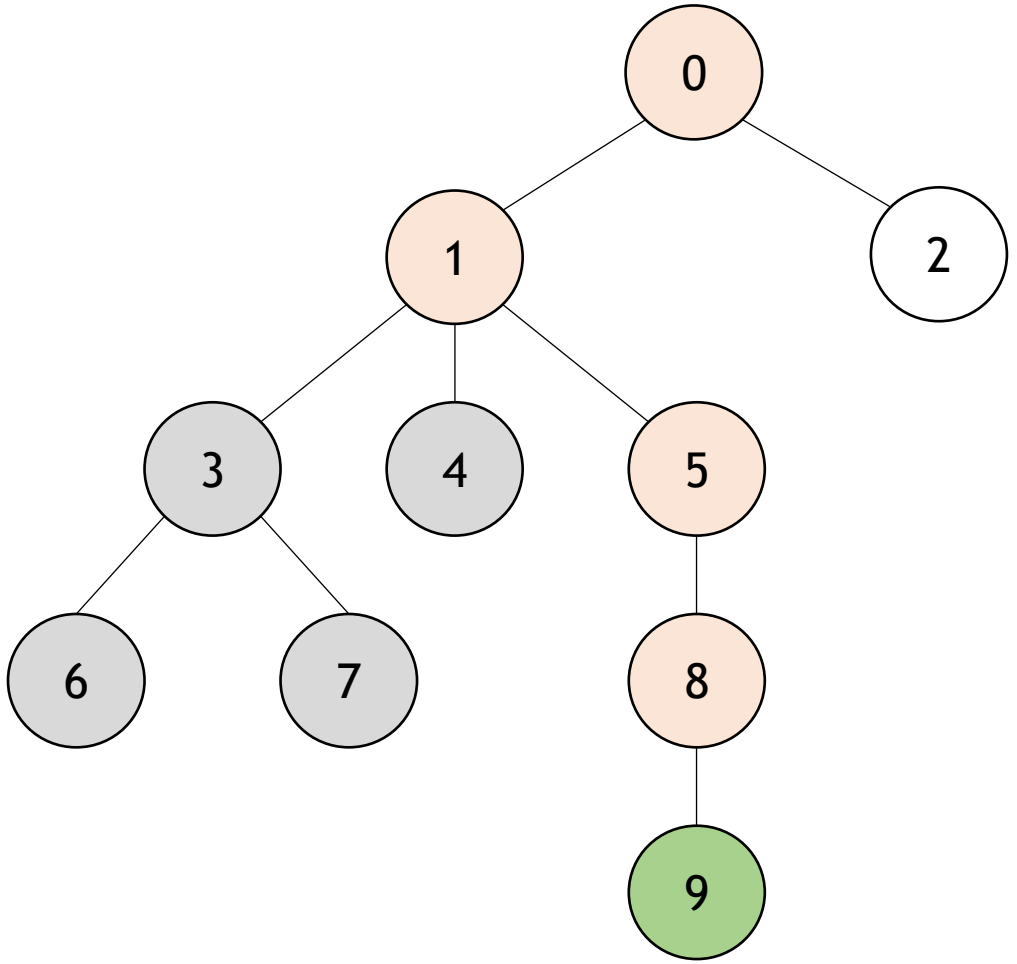
u	height[u]	maxLenght[u]
0	1	
1	3	
2		
3	2	2
4	1	0
5	1	
6	1	0
7	1	0
8	1	
9		



Diâmetro

diameter = 2

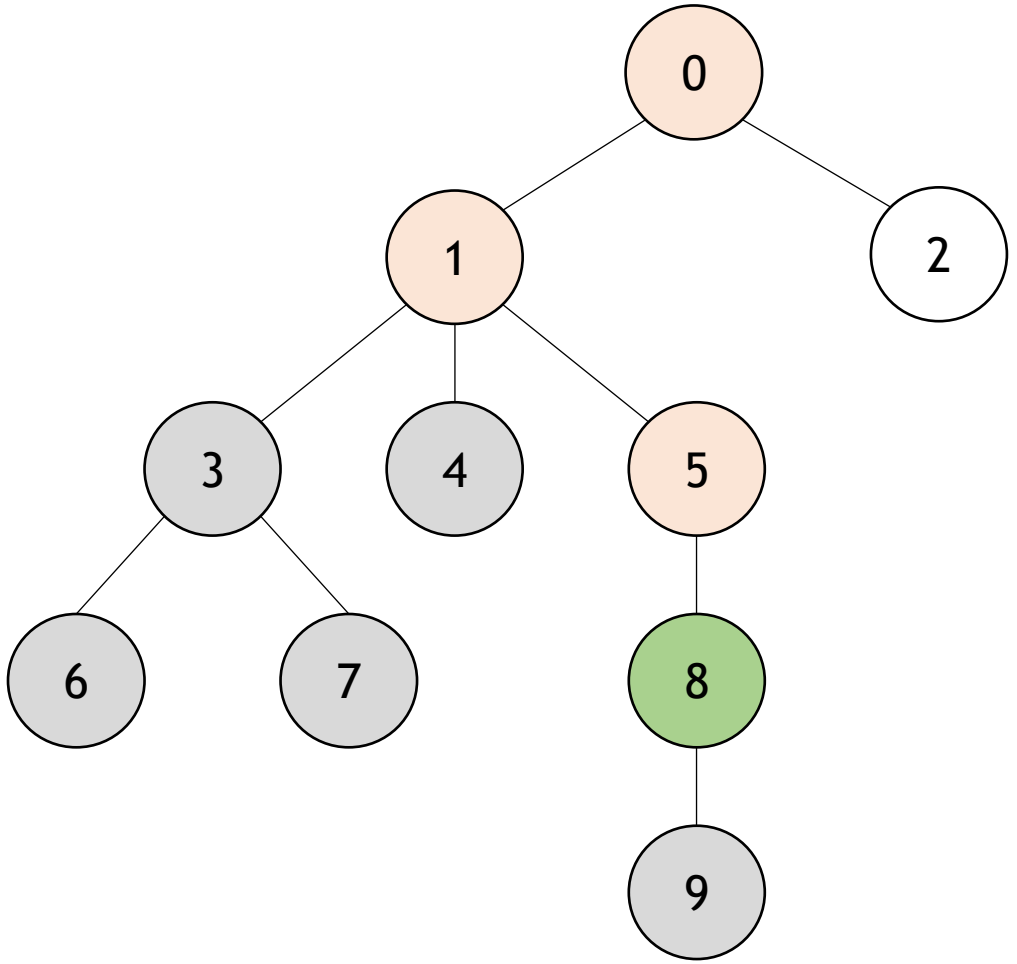
u	height[u]	maxLenght[u]
0	1	
1	3	
2		
3	2	2
4	1	0
5	1	
6	1	0
7	1	0
8	1	
9	1	0



Diâmetro

diameter = 2

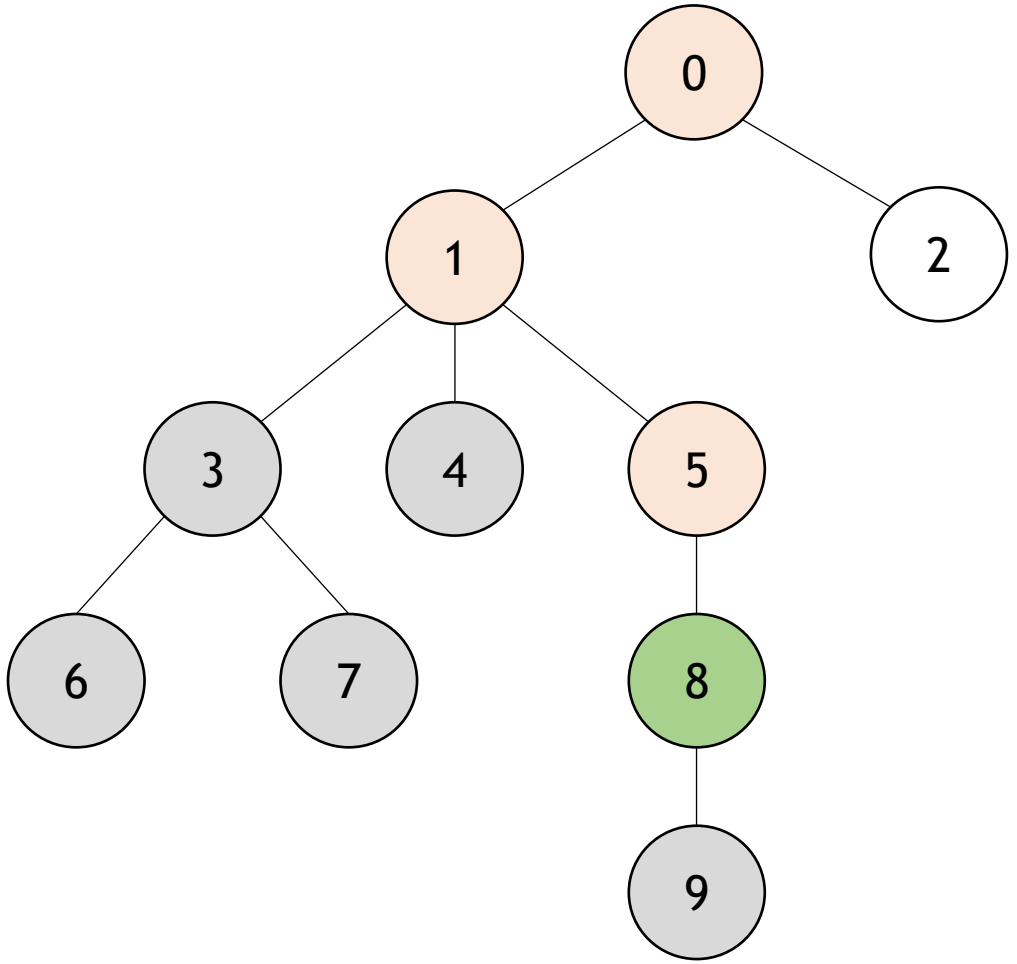
u	height[u]	maxLenght[u]
0	1	
1	3	
2		
3	2	2
4	1	0
5	1	
6	1	0
7	1	0
8	max(1, 1+1)	1 + 0
9	1	0



Diâmetro

diameter = 2

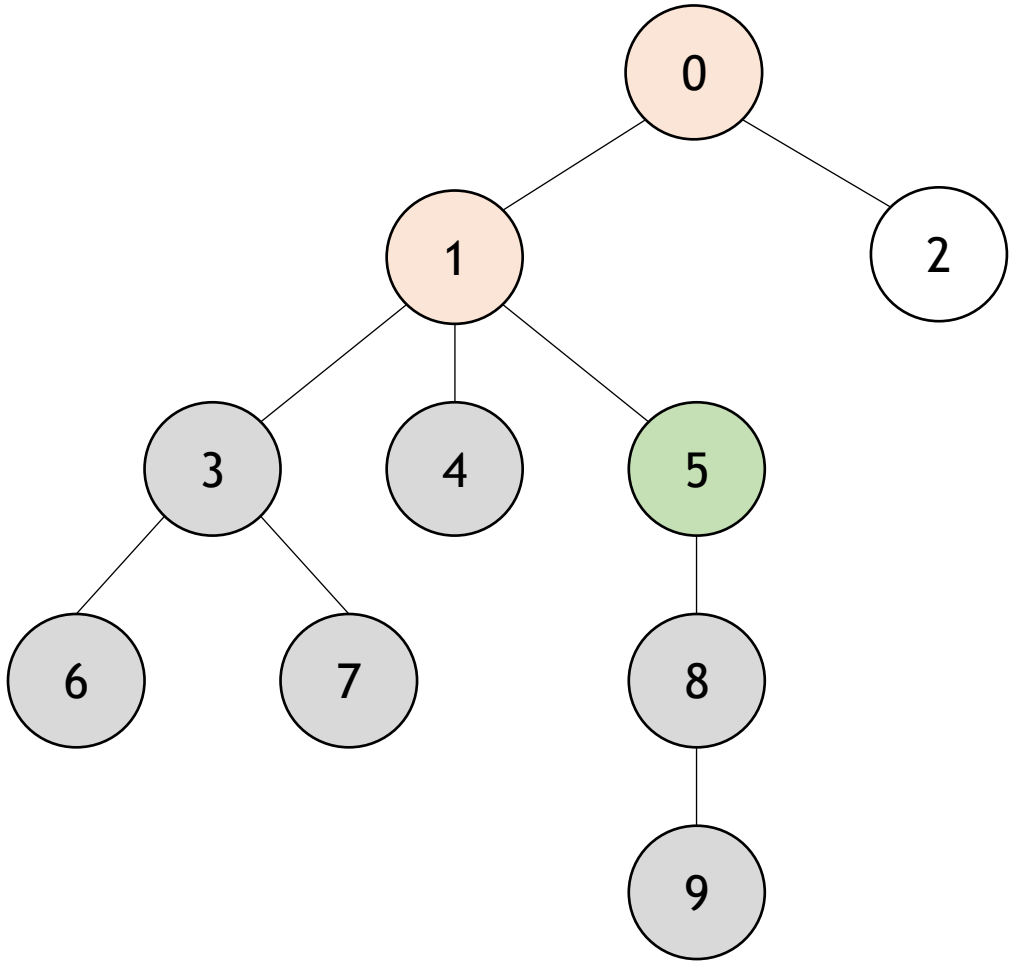
u	height[u]	maxLenght[u]
0	1	
1	3	
2		
3	2	2
4	1	0
5	1	
6	1	0
7	1	0
8	2	1
9	1	0



Diâmetro

diameter = 2

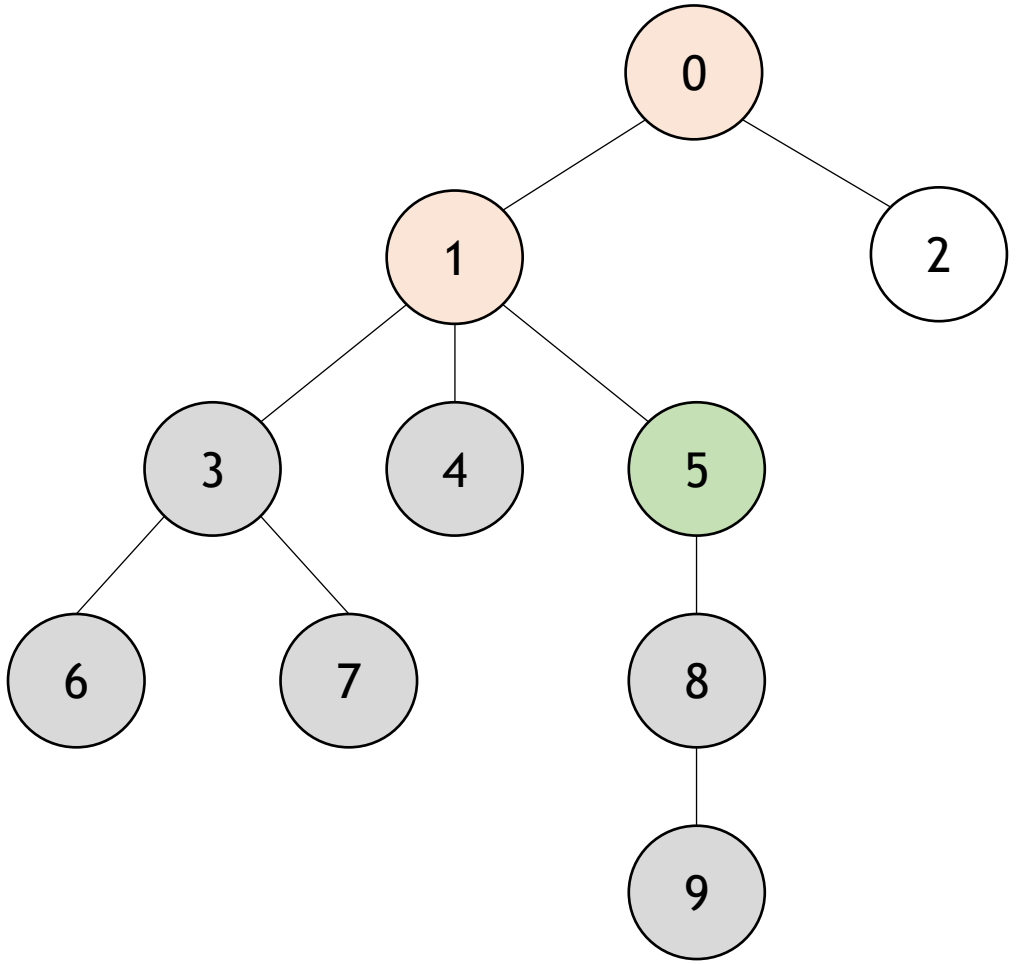
u	height[u]	maxLenght[u]
0	1	
1	3	
2		
3	2	2
4	1	0
5	max(1, 1+2)	2+0
6	1	0
7	1	0
8	2	1
9	1	0



Diâmetro

diameter = 2

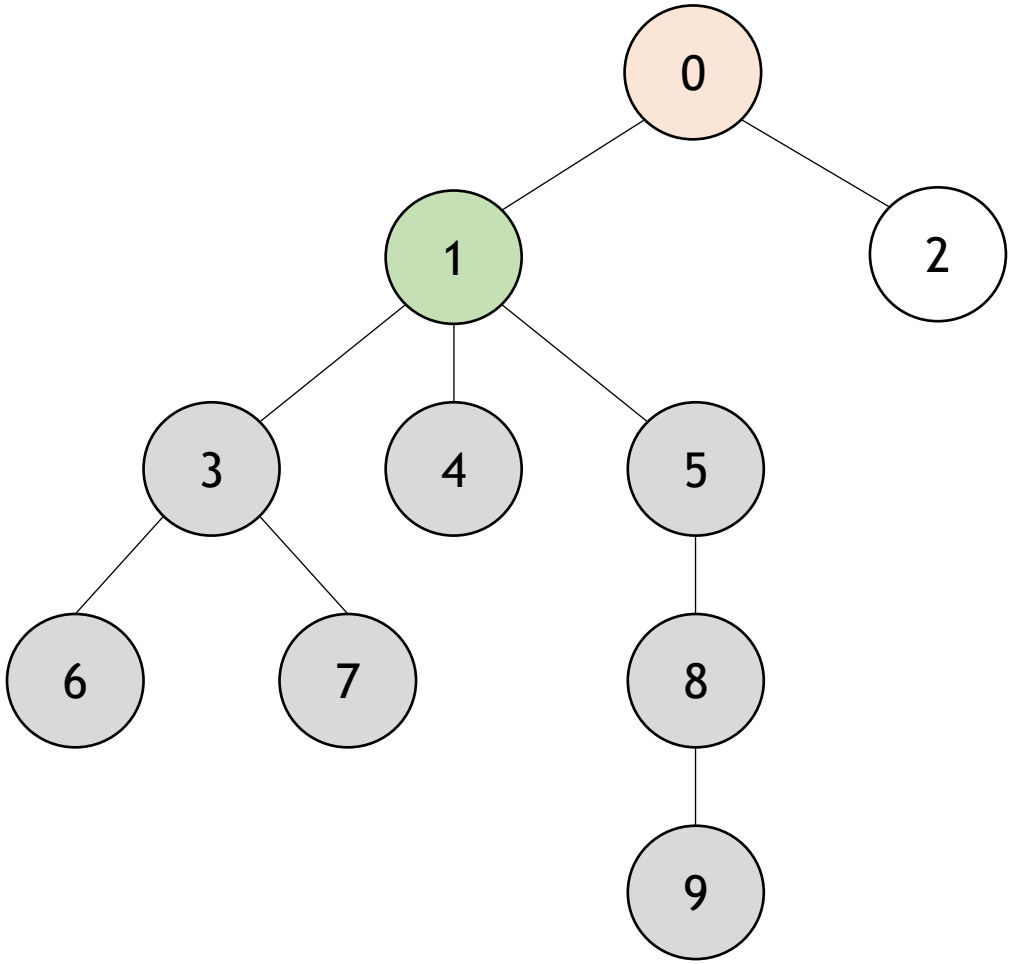
u	height[u]	maxLenght[u]
0	1	
1	3	
2		
3	2	2
4	1	0
5	3	2
6	1	0
7	1	0
8	2	1
9	1	0



Diâmetro

diameter = 2

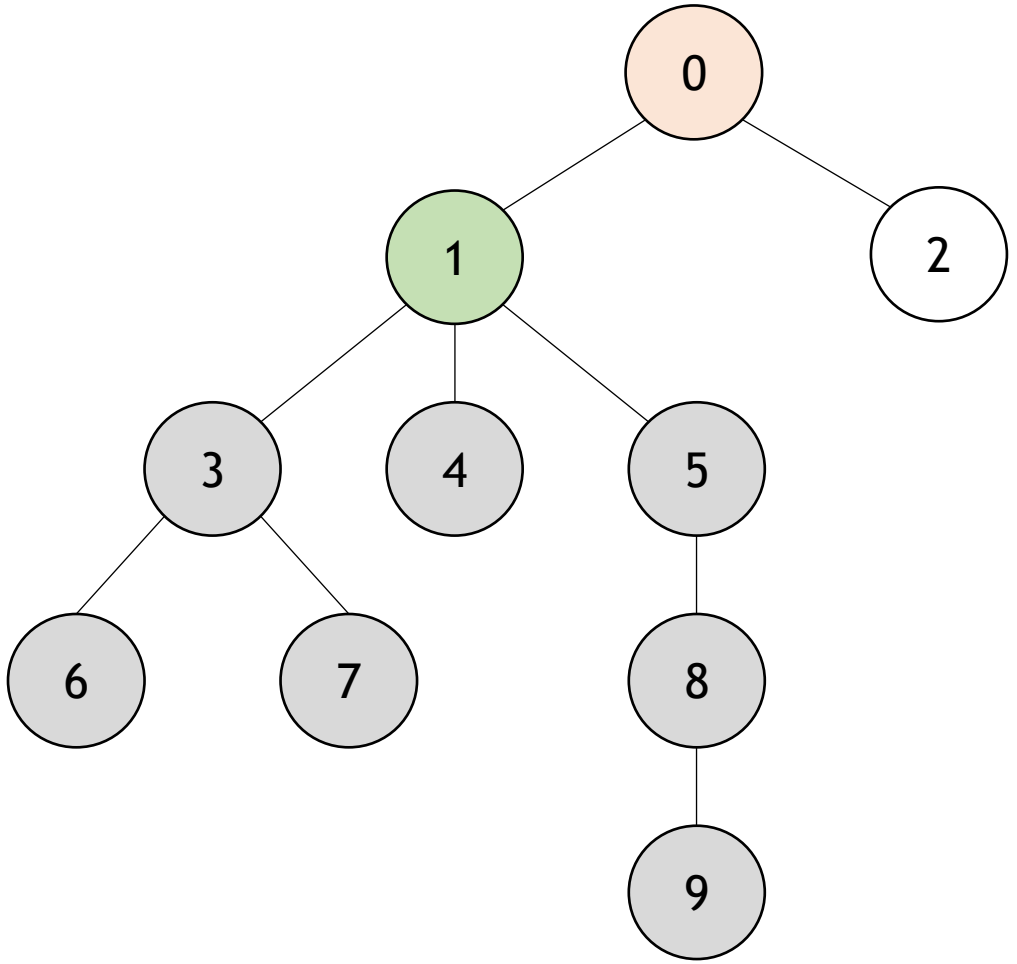
u	height[u]	maxLenght[u]
0	1	
1	max(3, 1+3)	3+2
2		
3	2	2
4	1	0
5	3	2
6	1	0
7	1	0
8	2	1
9	1	0



Diâmetro

diameter = 5

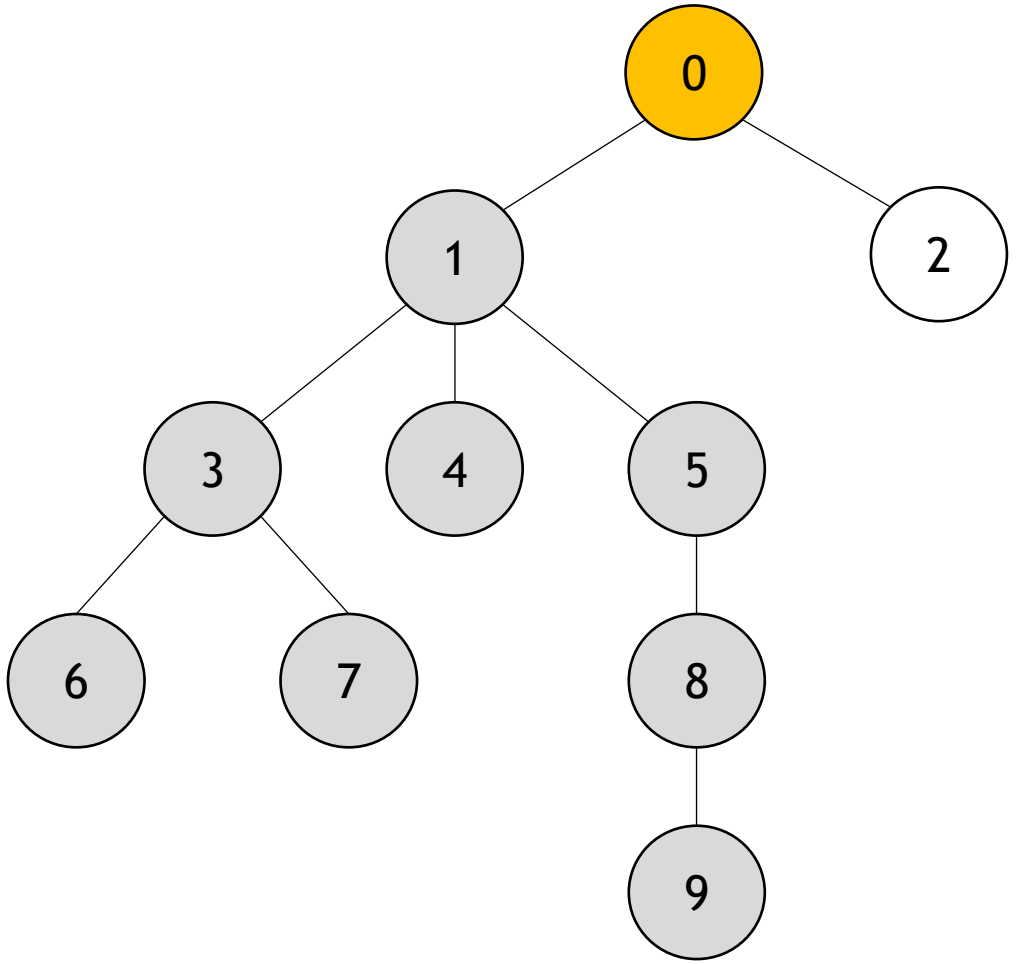
u	height[u]	maxLenght[u]
0	1	
1	4	5
2		
3	2	2
4	1	0
5	3	2
6	1	0
7	1	0
8	2	1
9	1	0



Diâmetro

diameter = 5

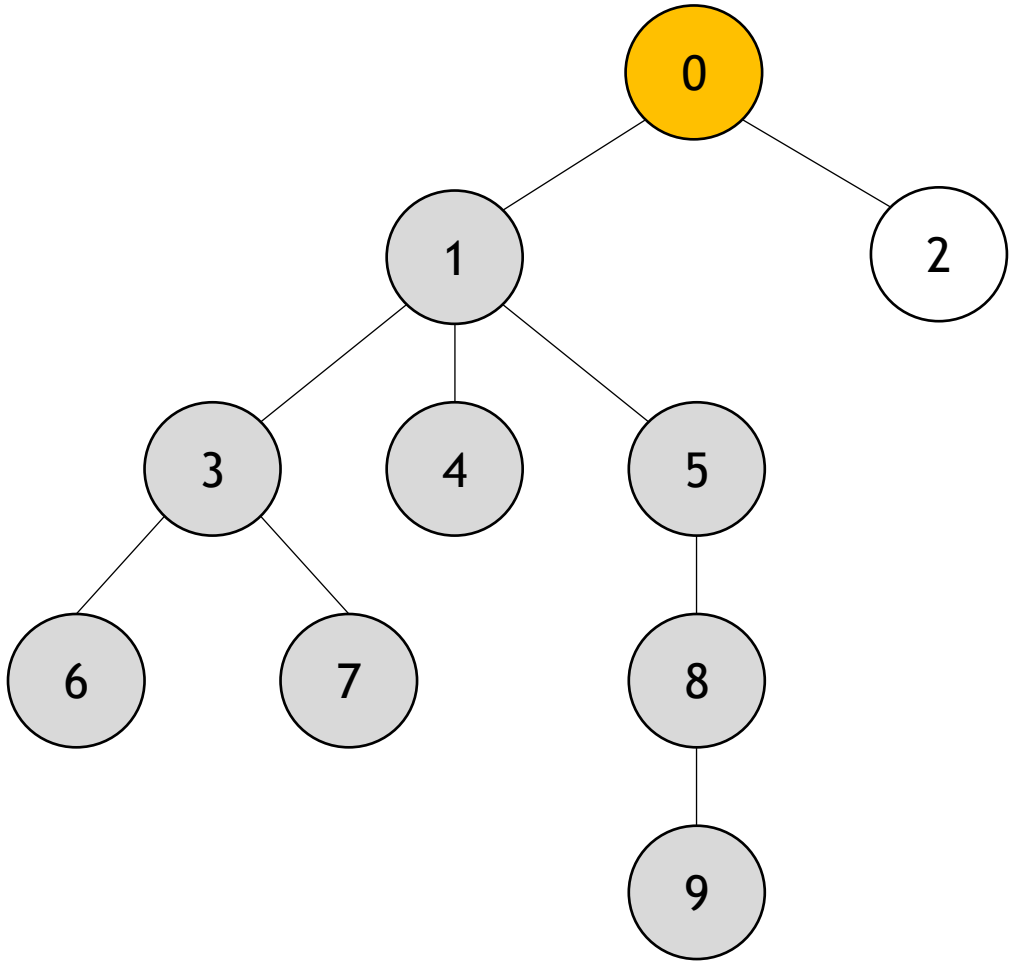
u	height[u]	maxLenght[u]
0	max(1, 1+4)	
1	4	5
2		
3	2	2
4	1	0
5	3	2
6	1	0
7	1	0
8	2	1
9	1	0



Diâmetro

diameter = 5

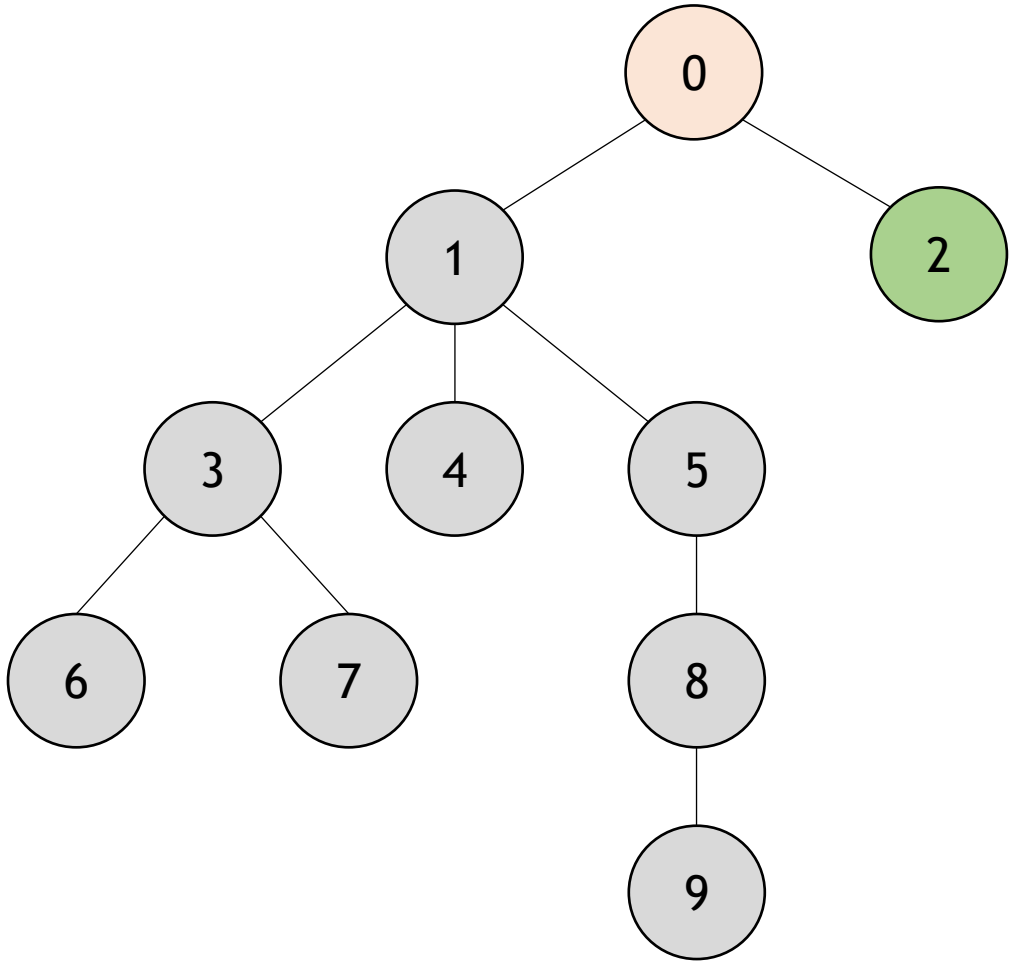
u	height[u]	maxLenght[u]
0	5	
1	4	5
2		
3	2	2
4	1	0
5	3	2
6	1	0
7	1	0
8	2	1
9	1	0



Diâmetro

diameter = 5

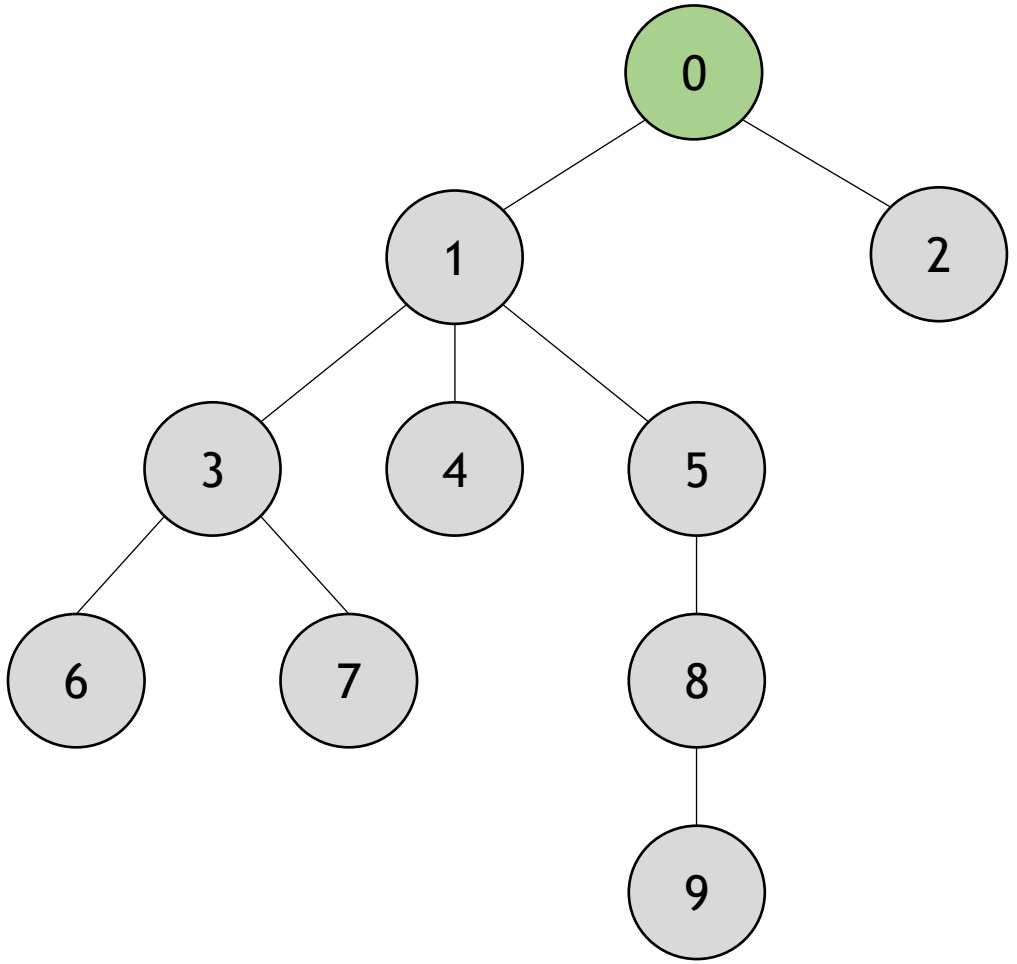
u	height[u]	maxLenght[u]
0	5	
1	4	5
2	1	0
3	2	2
4	1	0
5	3	2
6	1	0
7	1	0
8	2	1
9	1	0



Diâmetro

diameter = 5

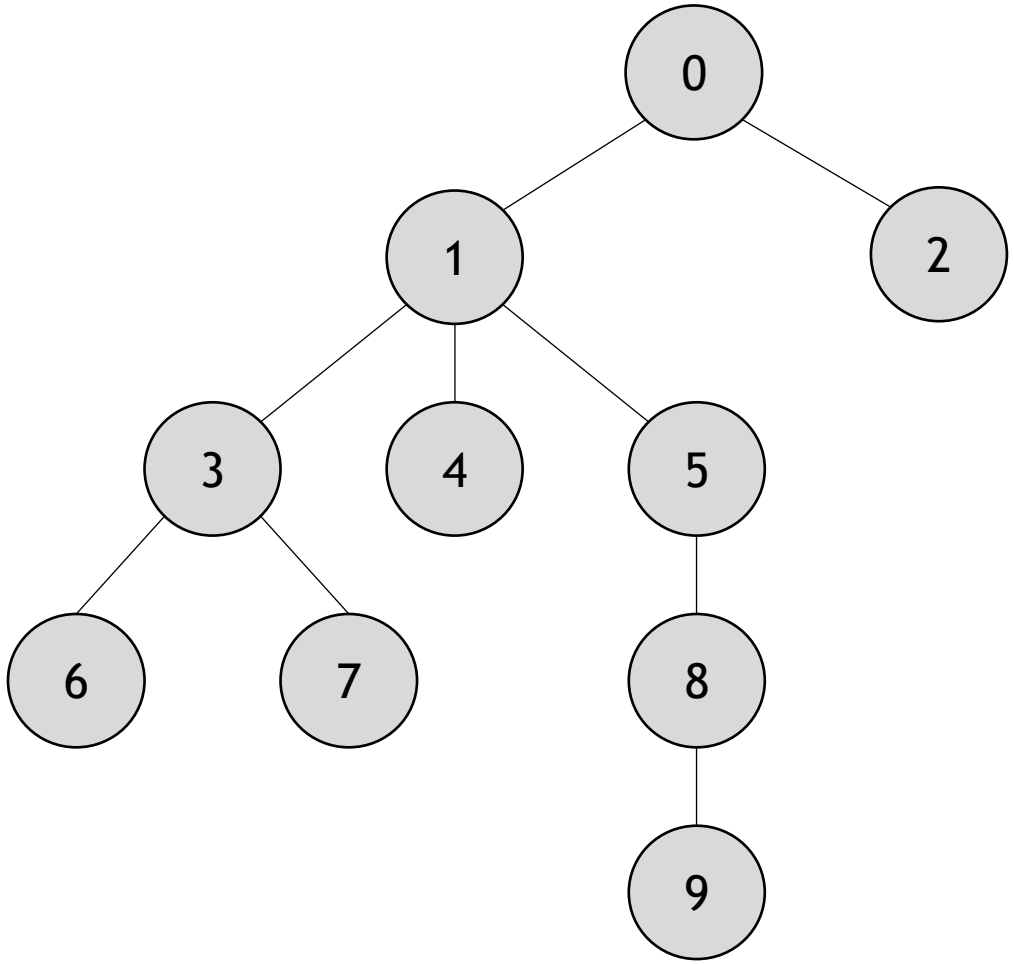
u	height[u]	maxLenght[u]
0	max(5, 1+1)	4+1
1	4	5
2	1	0
3	2	2
4	1	0
5	3	2
6	1	0
7	1	0
8	2	1
9	1	0



Diâmetro

diameter = 5

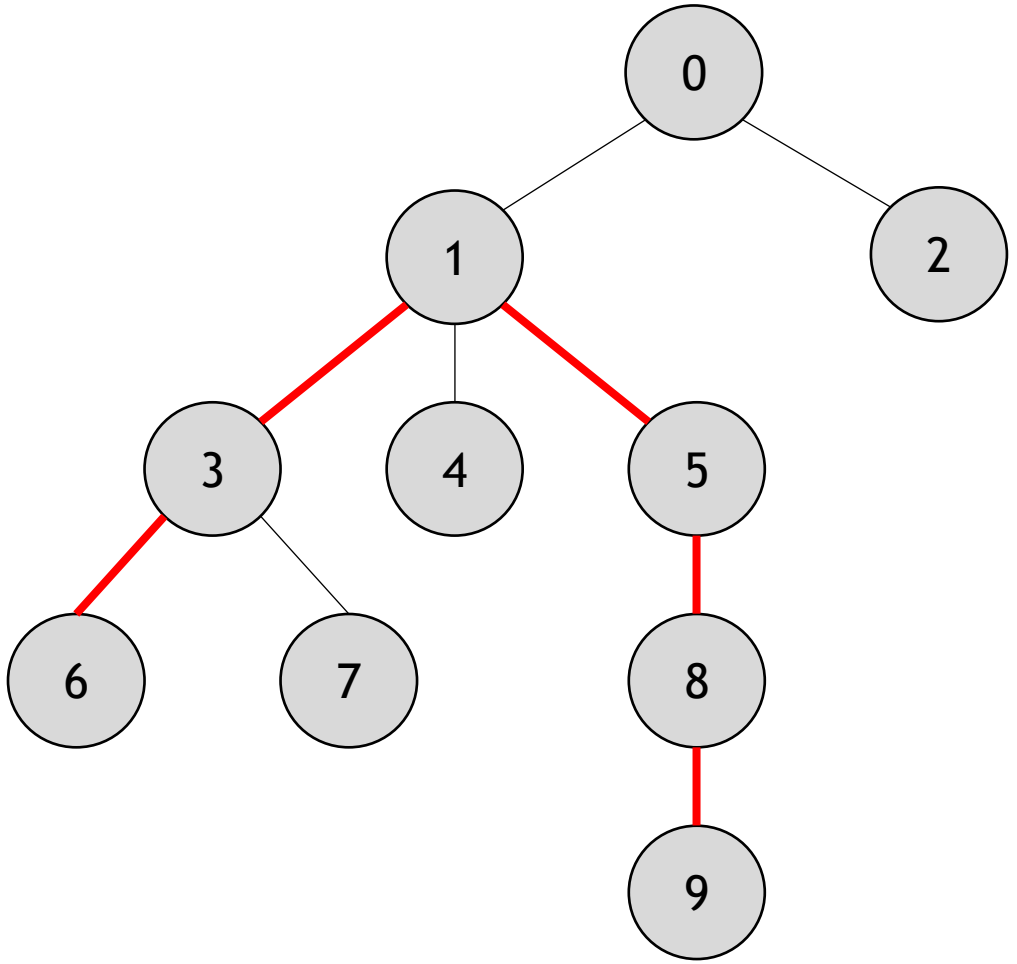
u	height[u]	maxLenght[u]
0	5	5
1	4	5
2	1	0
3	2	2
4	1	0
5	3	2
6	1	0
7	1	0
8	2	1
9	1	0



Diâmetro

diameter = 5

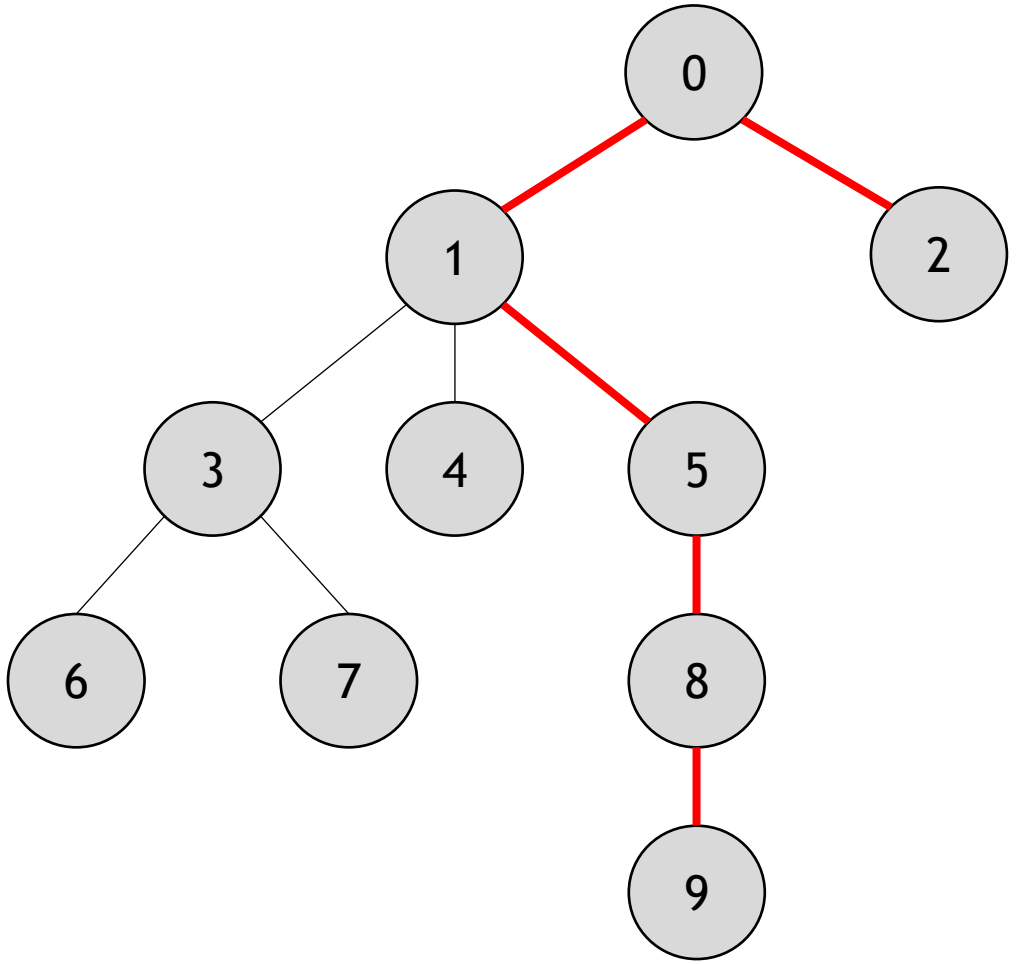
u	height[u]	maxLenght[u]
0	5	5
1	4	5
2	1	0
3	2	2
4	1	0
5	3	2
6	1	0
7	1	0
8	2	1
9	1	0



Diâmetro

diameter = 5

u	height[u]	maxLenght[u]
0	5	5
1	4	5
2	1	0
3	2	2
4	1	0
5	3	2
6	1	0
7	1	0
8	2	1
9	1	0



Referências

Aulas de Estrutura de Dados I da Profª Drª Simone das Graças Domingues Prado.

LAAKSONEN, Antii. Competitive Programmer's Handbook

https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/dfs.html

https://www.ime.usp.br/~song/mac5710/slides/05tree

https://web.fe.up.pt/~rcamacho/cadeiras/bioinformatica/docs/arvores.pdf

https://linux.ime.usp.br/~cef/mac499-05/monografias/daniel/apresentacao.pdf