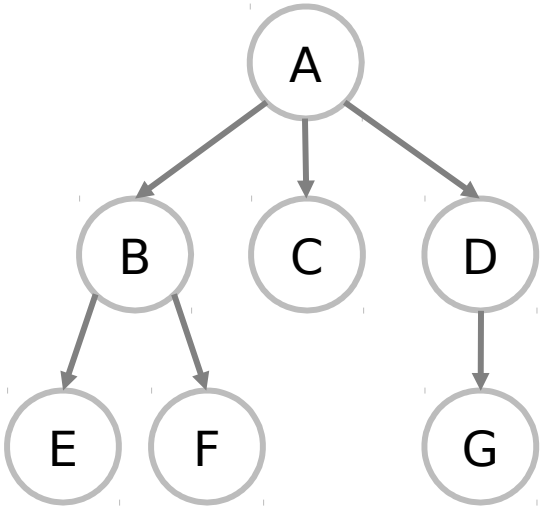
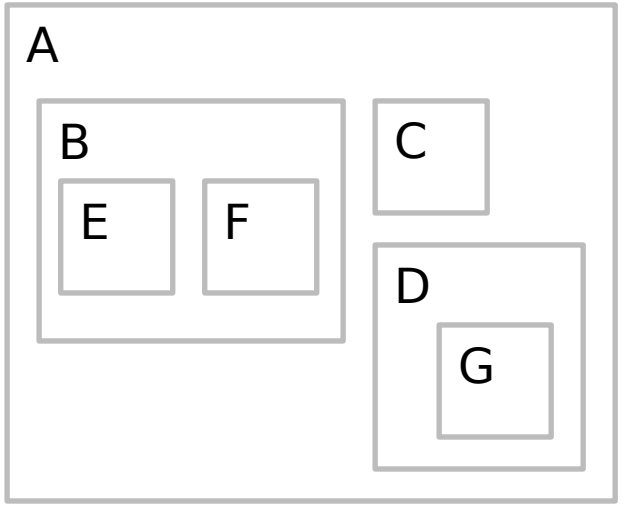
Estruturas de Dados Avançadas (INF1010)

Árvores Binárias

Árvore

Estrutura Hierárquica





(A (B (E, F)), C, (D (G)))

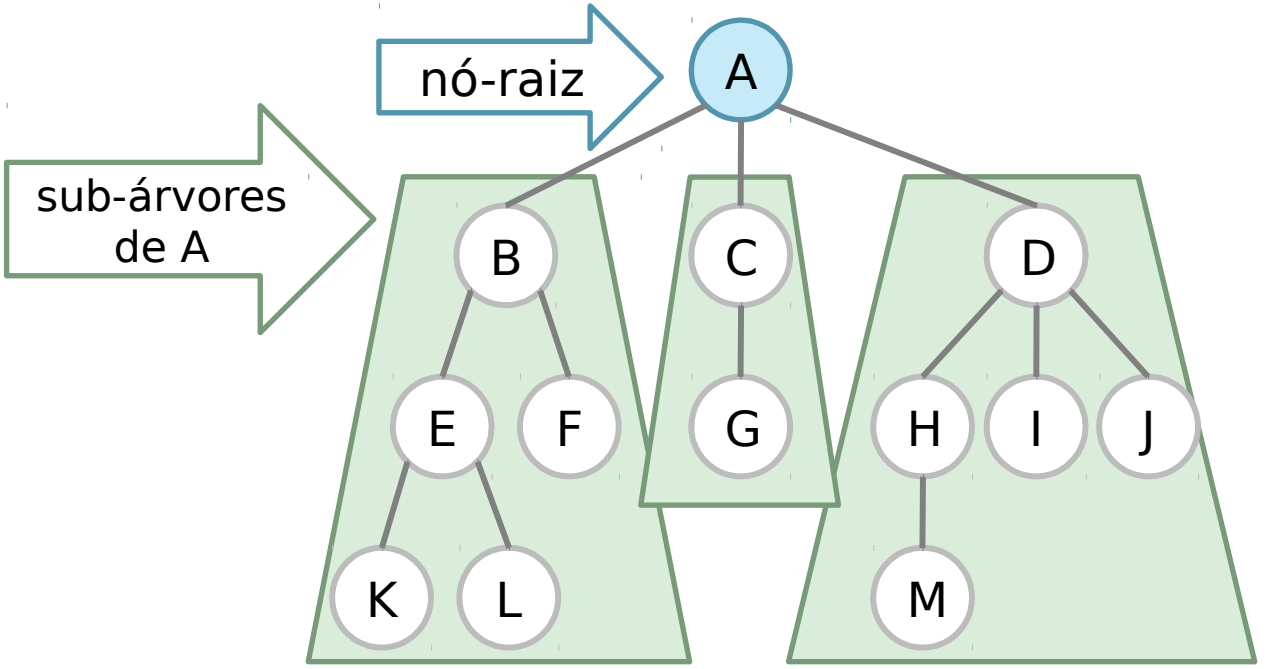
Árvore: nós e subárvores

árvore:

- nó raiz
- sub-árvores

nó:

- informação
- ramos



DEPARTAMENTO DE INFORMÁTICA PUC·RIO

Definições: grau, folha

Grau de um nó

- número de sub-árvores do nó

se grau = 0

- nó **folha** ou **terminal**

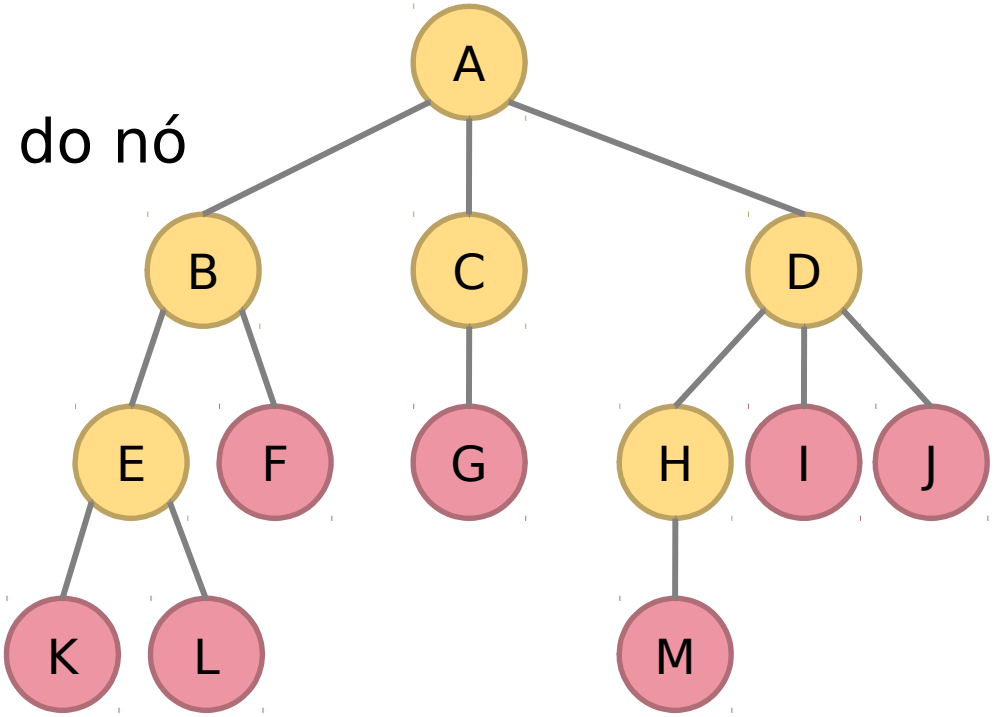
se grau > 0

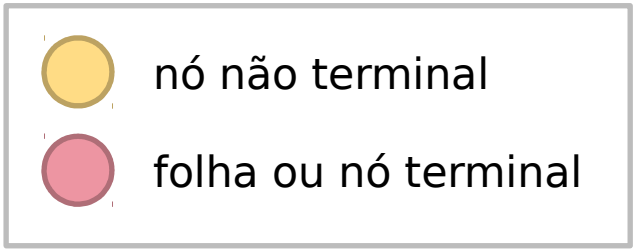
- nó **não-terminal**

Grau da árvore

- maior dentre os graus dos nós

  grau da árvore de exemplo = 3





Definições: pai, filhos, irmãos

Filhos de um nó

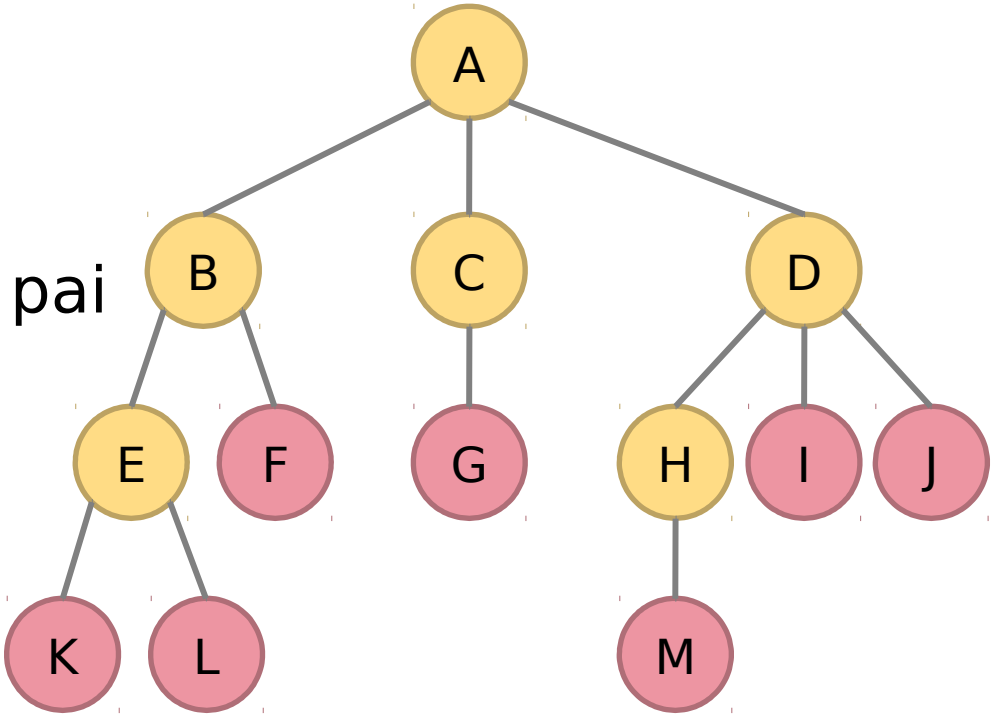
- raízes das sub-árvores

Pai de um nó

- nó é sub-árvore de seu pai

Irmãos de um nó

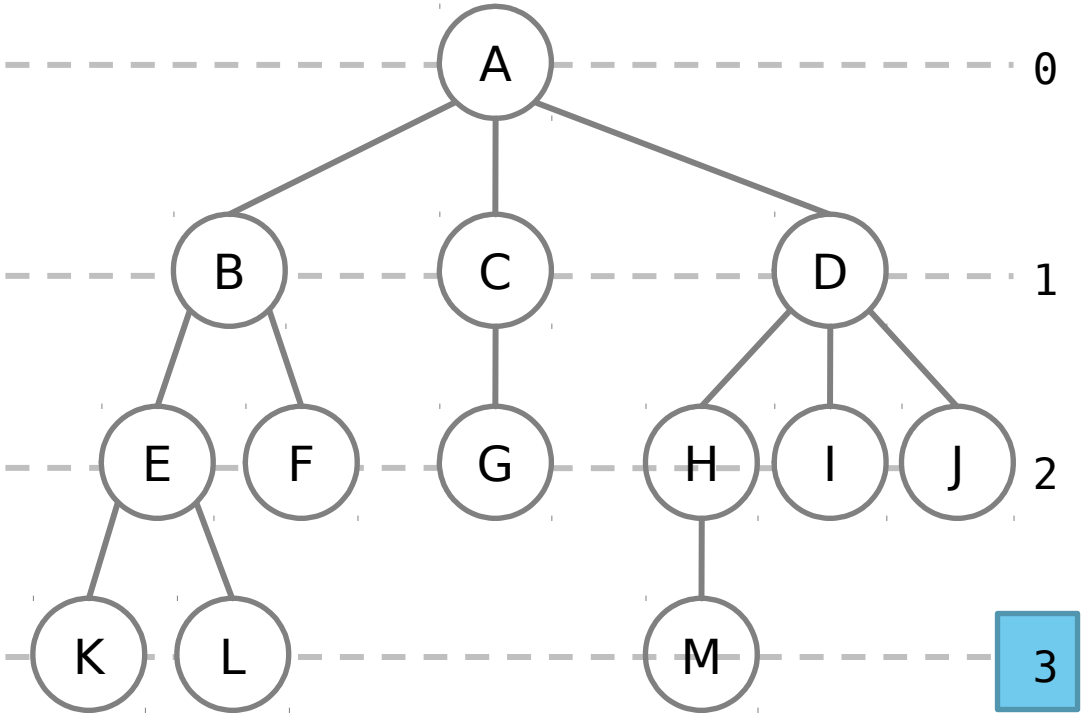
- tem o mesmo pai



Definições: nível, profundidade

nível (de um nó)

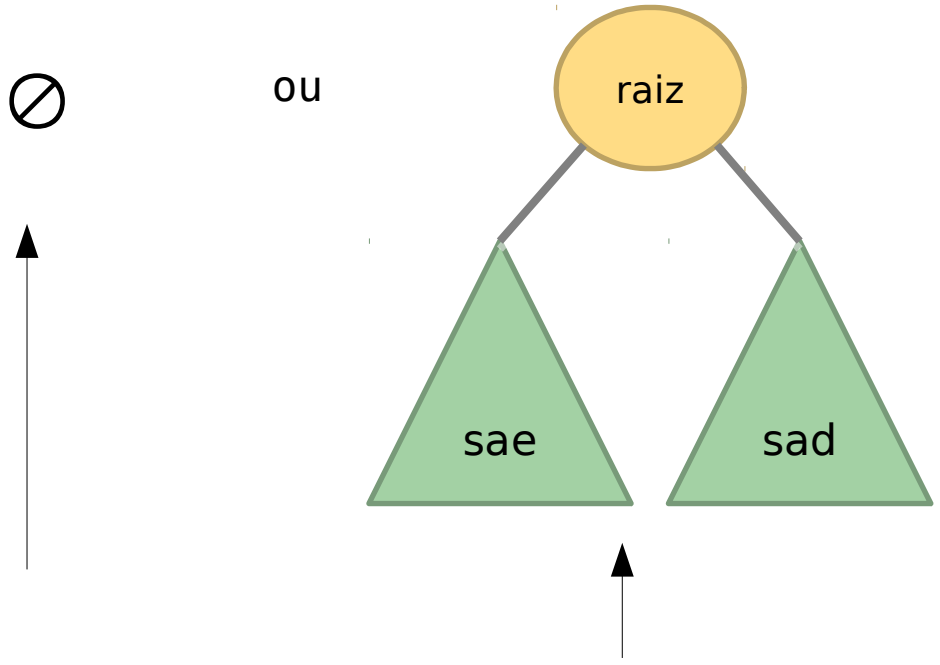
- raiz tem nível 0
- se nó X tem nível **n**, seus filhos têm nível **n+1**



altura ou **profundidade** (da árvore)
maior nível dentre todos os nós
no exemplo: **h = 3**

Árvore Binária

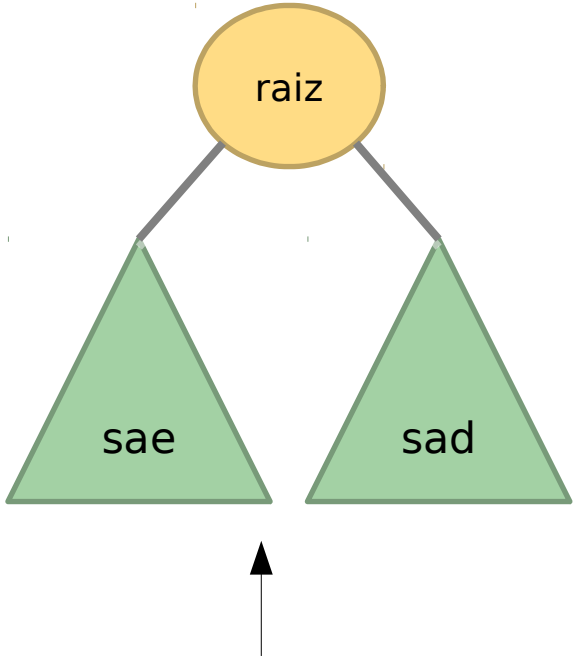
- ⊘ (árvore vazia)
- raiz, sub-árvore esquerda , sub-árvore direita



Árvore Binária

- ⊘ (árvore vazia)
- raiz, sub-árvore esquerda , sub-árvore direita

⊘ ou

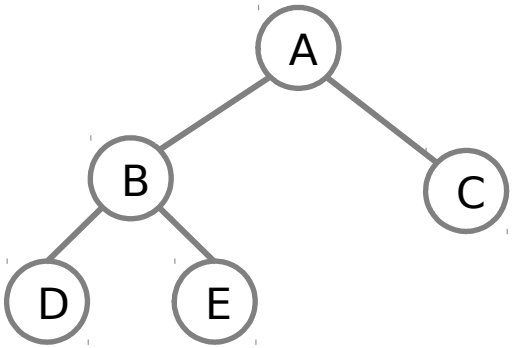


( ) (raiz sae sad)

```
/* nó da árvore binária */

struct arvbin {
  tdados info;
  struct arvbin *esq;
  struct arvbin *dir;
};
```

Exemplos



(A (B (D () ()) (E () ())) (C () ())

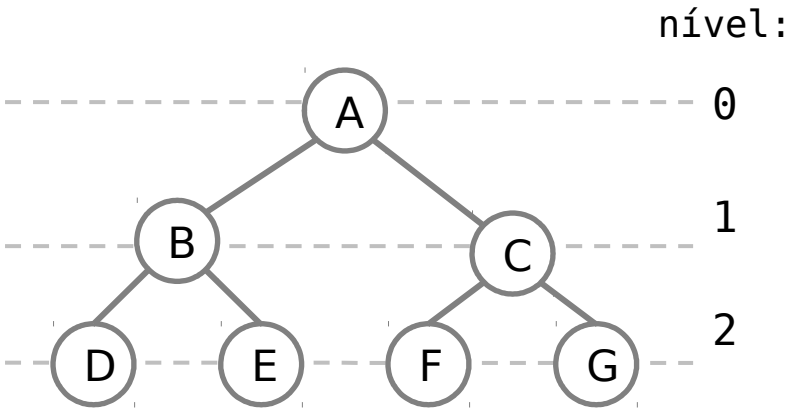
(A (B) ( ))

(A ( ) (B))

Árvore Binária Cheia

número máximo de nós no nível **i**

$$n_i = 2^i$$



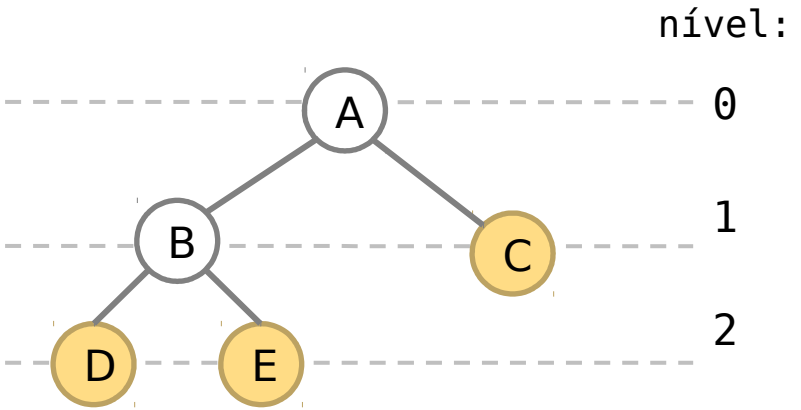
número máximo de nós na árvore de altura **k**

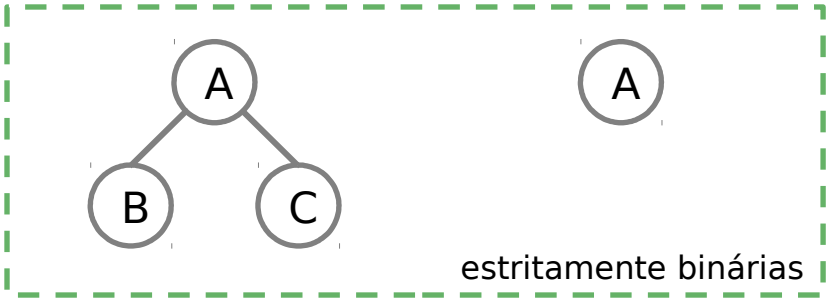
$$n_{max} = 2^k + \ldots + 2^2 + 2 + 1 = 2^{k+1} - 1$$

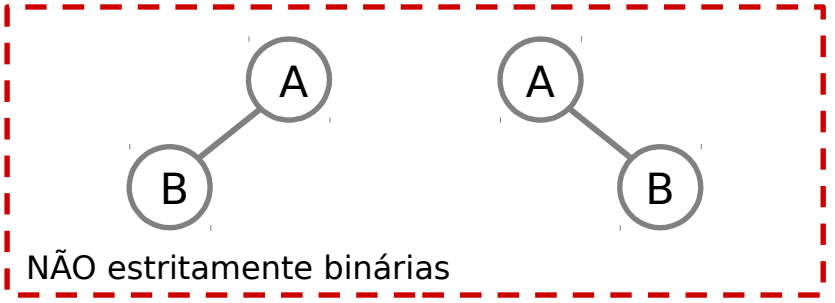
árvore binária **cheia** de altura **k** → $2^{k+1} - 1$ nós

Árvore Estritamente Binária

Cada nó tem 0 ou dois filhos



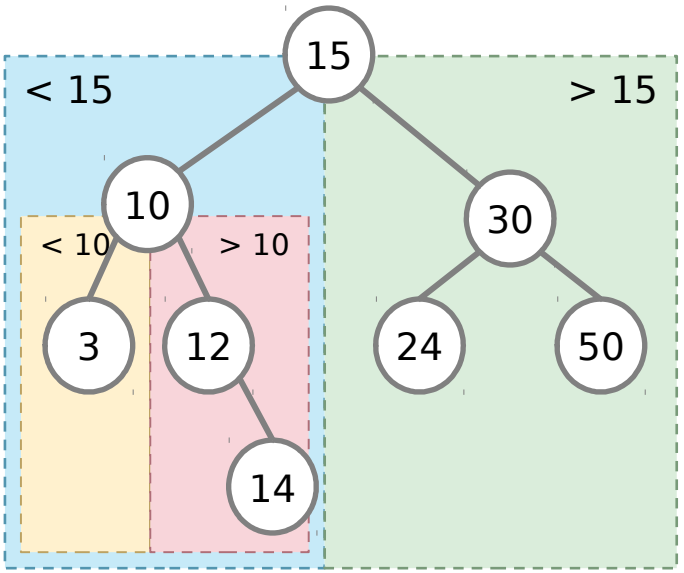




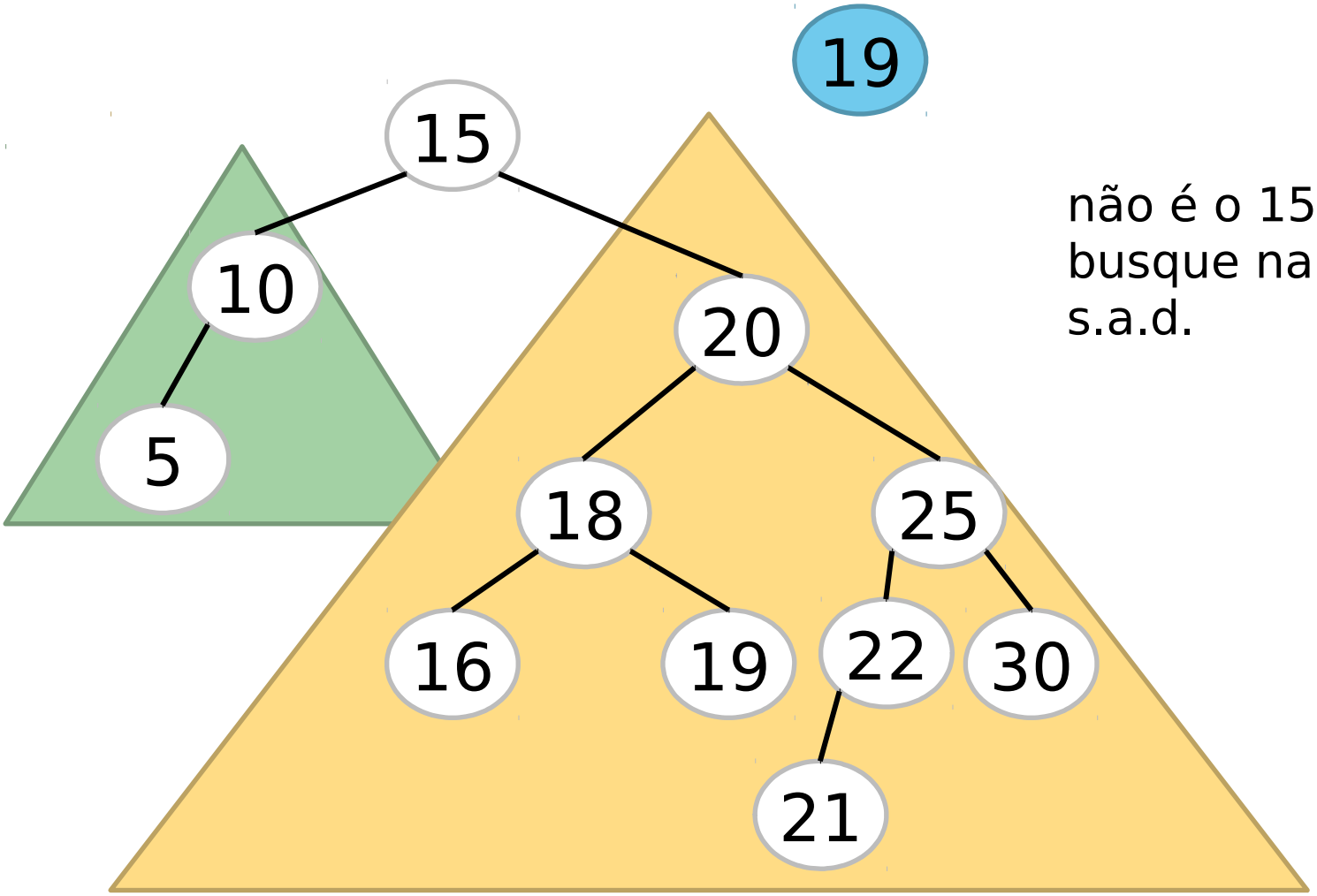
Árvore Binária de Busca

Uma ABB é uma árvore binária vazia, ou uma árvore tal que

- cada nó possui uma chave
- as chaves na sub-árvore esquerda são menores do que a chave da raiz
- as chaves na sub-árvore direita são maiores do que a chave da raiz
- as sub-árvores esquerda e direita são árvores binárias de busca

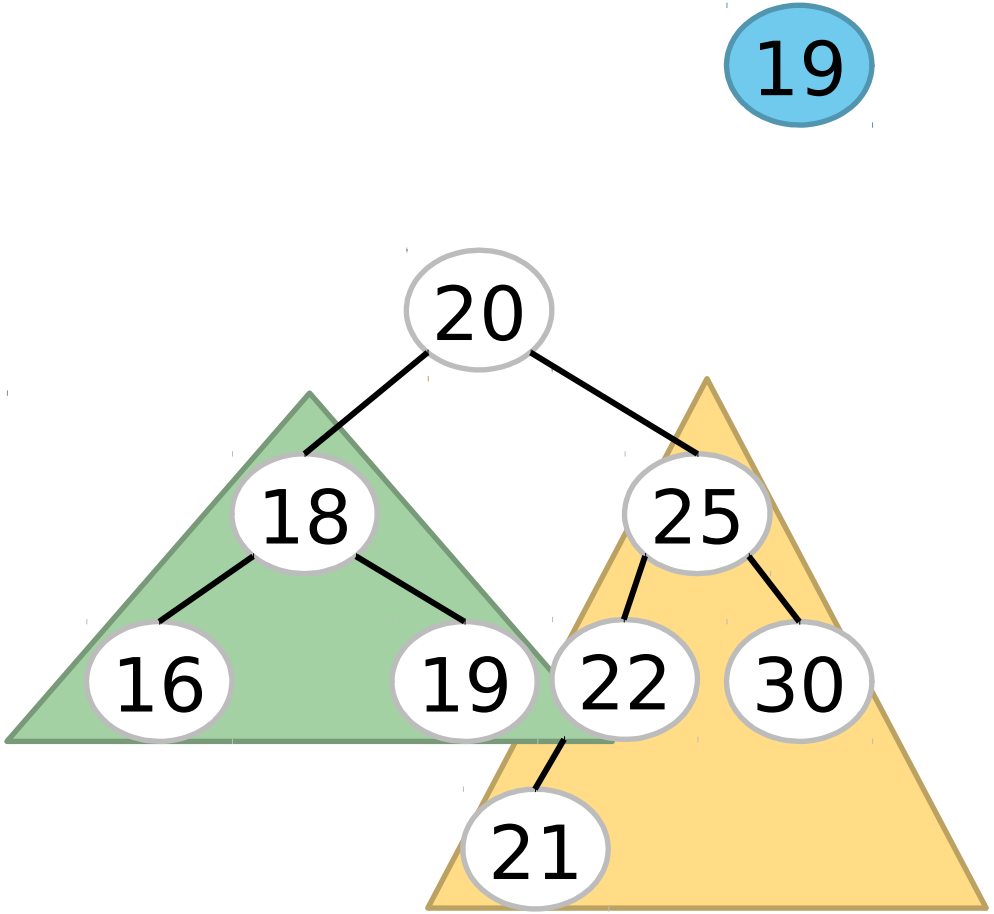


Busca em uma ABB



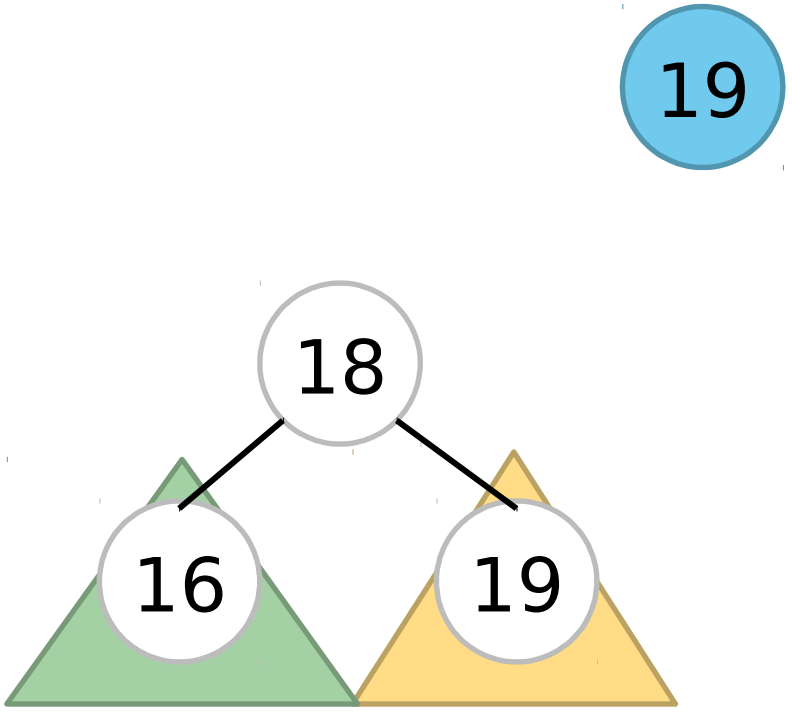
não é o 15, busque na s.a.d.

Busca em uma ABB



não é o 20, busque na s.a.e.

Busca em uma ABB

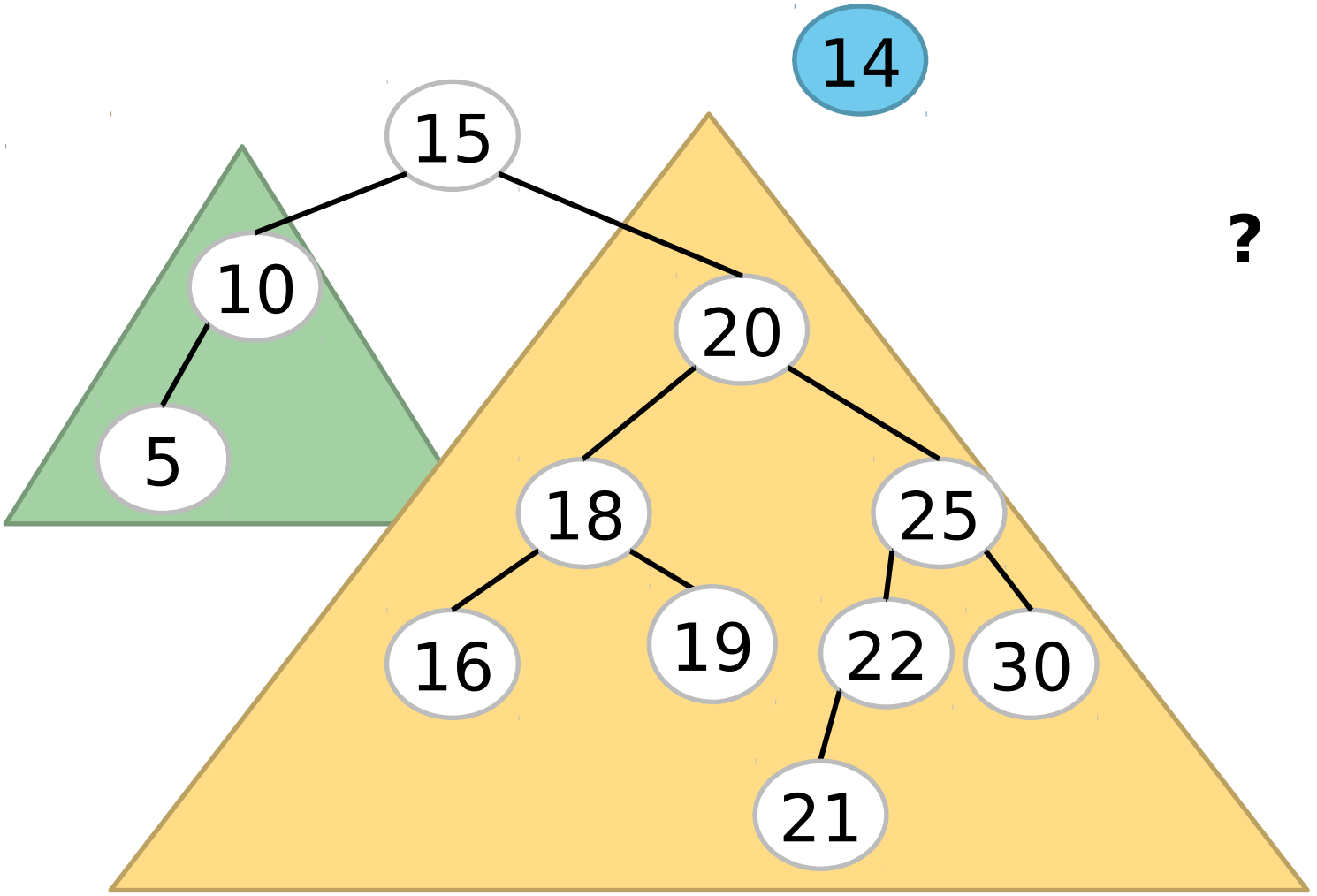


não é o 18,
busque na
s.a.d.

Busca em uma ABB

encontrou !

Busca em uma ABB



Implementando Mapa com ABB

mapa.h

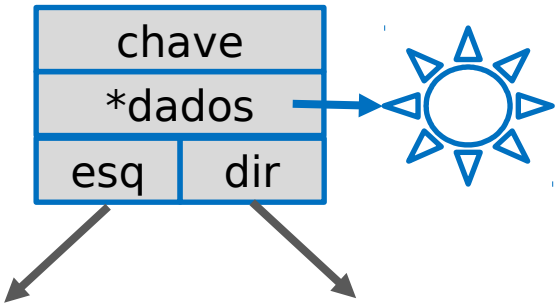
```
typedef struct smapa Mapa;

Mapa* cria (void);
Mapa* insere (Mapa* m, int chave,
                    tdados *novosdados);
tdados *busca (Mapa *m, int chave);
Mapa* retira (Mapa *m, int chave);
void destroi (Mapa *m);
```

```
Mapa* cria(void)
{
   return NULL;
}
```

mapa.c

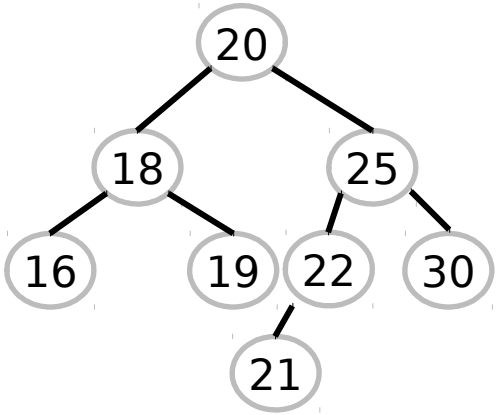
```
struct smapa {
   int chave;
   tdados *dados;
   Mapa* esq;
   Mapa* dir;
};
```



Implementando Liberação

```
void destroi (Mapa* raiz);
```

1. Começe pela raiz
2. Se a árvore for **vazia** então retorne
3. Destroi **sub-árvore à esquerda**
4. Destroi **sub-árvore à direita**
5. Libera o nó raiz

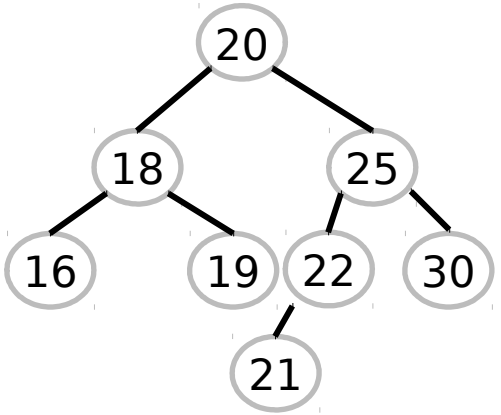



```
void destroi (Mapa* r) {
   if (r == NULL) return;
   destroi (r->esq);
   destroi (r->dir);
   free(r);
}
```

Implementando Busca

```
tdados* busca (Mapa* raiz, int chave);
```

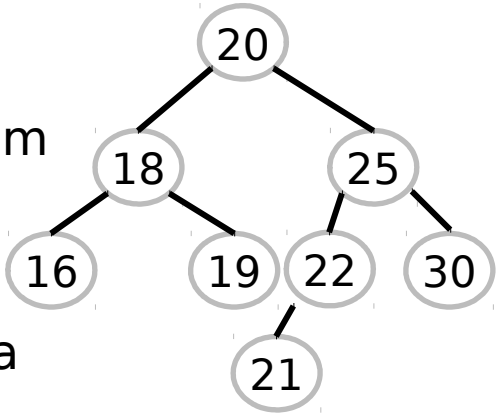
1. Começe com "visitado = nó raiz"
2. Se a árvore for **vazia** então retorne **valor especial (não presente)**
3. Se a chave procurada for **menor** que a chave do nó, **atualize visitado para sub-árvore à esquerda** e volte para passo 2
4. Se a chave procurada for **maior** que a chave do nó, **atualize visitado para sub-árvore à direita** e volte para passo 2
5. Se for **igual** retorne os dados associados



Implementando Busca com Recursão

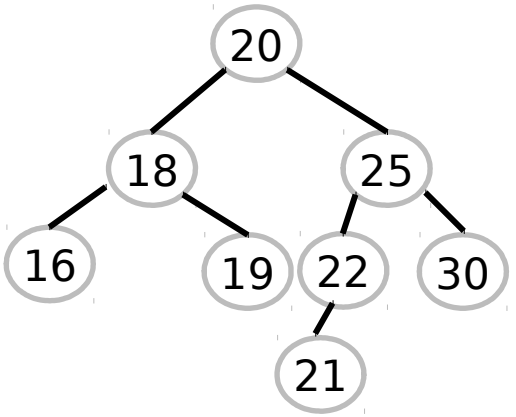
```
tdados* busca (Mapa* raiz, int chave);
```

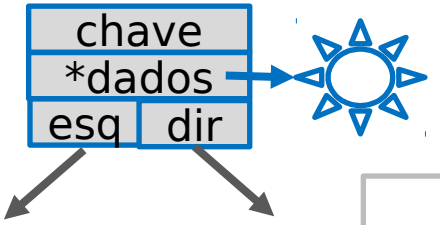
1. Começe pela raiz
2. Se a árvore for **vazia** então retorne **valor especial**
3. Se a chave procurada for **menor** que a chave do nó, **procure na sub-árvore à esquerda** e responda com a resposta recebida
4. Se a chave procurada for **maior** que a chave do nó, **procure na sub-árvore à direita** e responda com a resposta recebida
5. Se for **igual** retorne os dados associados



Implementando Busca com Recursão

```
tdados* busca (Mapa* r, int c) {
   if (r == NULL)
         return NULL;
   else if (c < r->chave)
         return busca (r->esq, c);
   else if (c > r->chave)
         return busca (r->dir, c);
   else return r->dados;
}
```






```
struct smapa {
   int chave;
   tdados *dados;
   Mapa* esq;
   Mapa* dir;
};
```