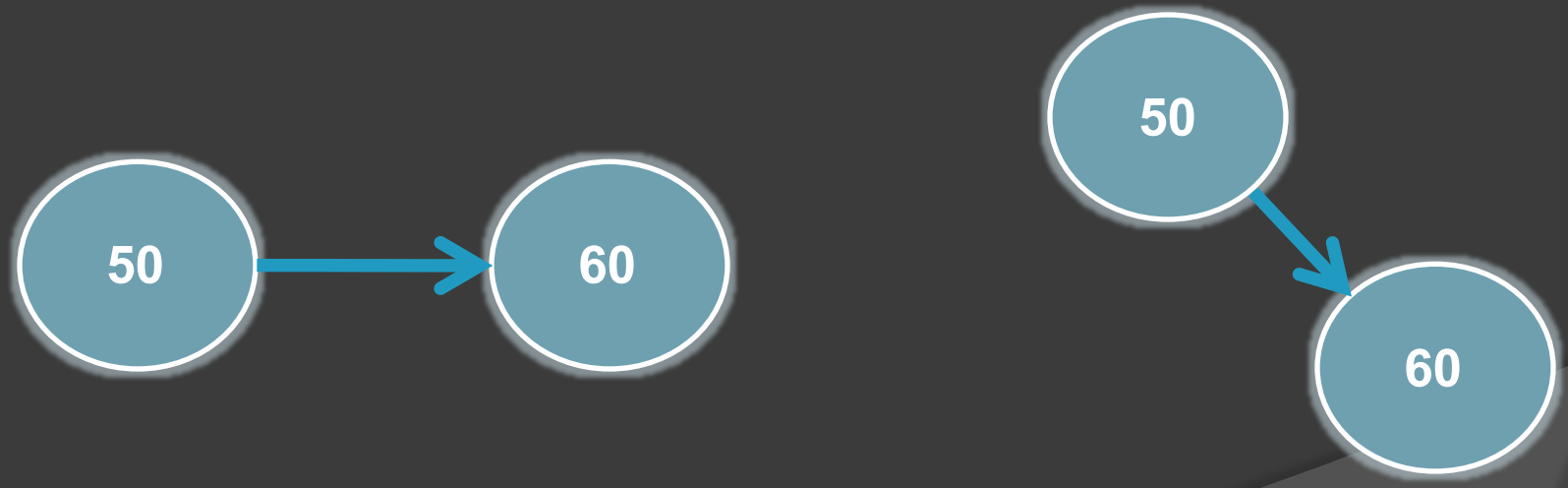
INSERÇÃO NA ÁRVORE SBB #2
PENSANDO NO ALGORITMO

Objetivo

- Ter embasamento necessário para codificar o algoritmo de inserção da árvore SBB.

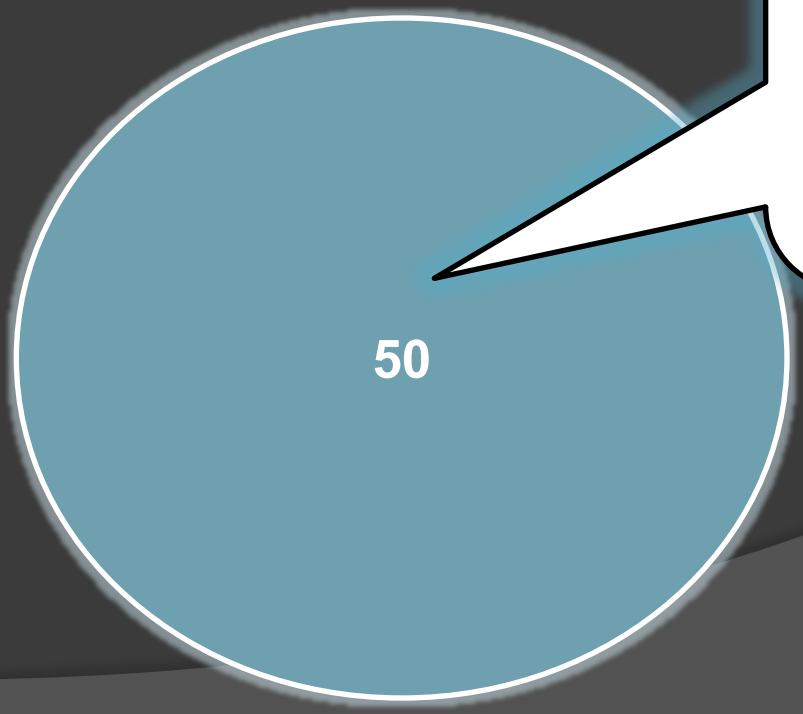
Introdução

- Toda árvore SBB leva em consideração a inclinação do ponteiro.
  - ...mas como fazer isto?



Introdução

- Toda árvore SBB leva em consideração a inclinação do ponteiro.
  - ...mas como fazer isto?



Vamos desmistificar a estrutura do nó!

Introdução

Em uma estrutura normal de uma árvore precisamos de uma chave, um dado, ponteiro para esquerda e direita.

Introdução


```
template <typename TIPO>
struct TNo{
      int chave;
      TIPO dado;
      TNo<TIPO> * dir;
      TNo<TIPO> * esq;
};
```

Pensando o código no C++, teríamos:

Introdução


Agora precisamos armazernar a orientação dos ponteiros!
Chave: 50
Dado: {um dado qualquer}
orientE
ponteiro esquerda
orientD
ponteiro direita

Introdução


```
const char VERTICAL = 'v';
const char HORIZONTAL = 'h';
template <typename TIPO>
struct TNo{
      int chave;
      TIPO dado;
      TNo<TIPO> * dir;
      char orientD;
      TNo<TIPO> * esq;
      char orientE;
};
```

No código, podemos pensar em algo deste tipo!

Inserção

- Mas como inserimos em uma árvore SBB?
  - levando em consideração a montagem da árvore, iremos criar o algoritmo de inserção.

Inserção

- Que tal lembrarmos a implementação da inserção em uma árvore de busca binária no C++?
  - ...

```
template <typename TIPO>
void inserir (TNo<TIPO> *&no, int chave, TIPO dado){
	if(no == NULL){
		no = new TNo<TIPO>;
		no->chave = chave;
		no->dado  = dado;
		no->esq   = NULL;
		no->dir   = NULL;
	}else{
		if(chave > no->chave){
			inserir(no->dir, chave, dado);
		}else{
			if(chave < no->chave){
				inserir(no->esq, chave, dado);
			}
		}
	}
}
```

Inserção

- Estudamos que na inserção em uma árvore SBB há movimentos que devemos fazer.
- Vamos esquecê-los um pouco e nos concentrarmos em criar uma inserção capaz de trabalhar com a violação

Inserção

- Relembrando:
  - uma violação ocorre quando há dois nós horizontais consecutivos

Inserção

- Note que a violação deve ser tratada pelo nó do início da violação.
  Logo, este nó deve detectar esta necessidade na inserção!

corre quando há dois nós
ecutivos

Podemos realizar esta detecção de várias formas.

Mas uma coisa sabemos: só conseguiremos verificar se houve ou não violação, após a inserção do nó

quando há dois nós
tivos

Inserção

- Como estamos trabalhando com funções recursivas, devemos realizar esta verificação após as chamadas de recursividade.
  - ou seja...

```
template <typename TIPO>
void inserir (TNo<TIPO> *&no, int chave
      if(no == NULL){
            no = new TNo<TIPO>;
            no->chave = chave;
            no->dado  = dado;
            no->esq   = NULL;
            no->dir   = NULL;
      }else{
            if(chave > no->chave){
                  inserir(no->dir, chave, dado);
                  //(aqui...)
            }else{
                  if(chave < no->chave){
                        inserir(no->esq, chave, dado);
                        //(aqui...)
                  }
            }
      }
}
```

Ou seja, ... nos locais em vermelho

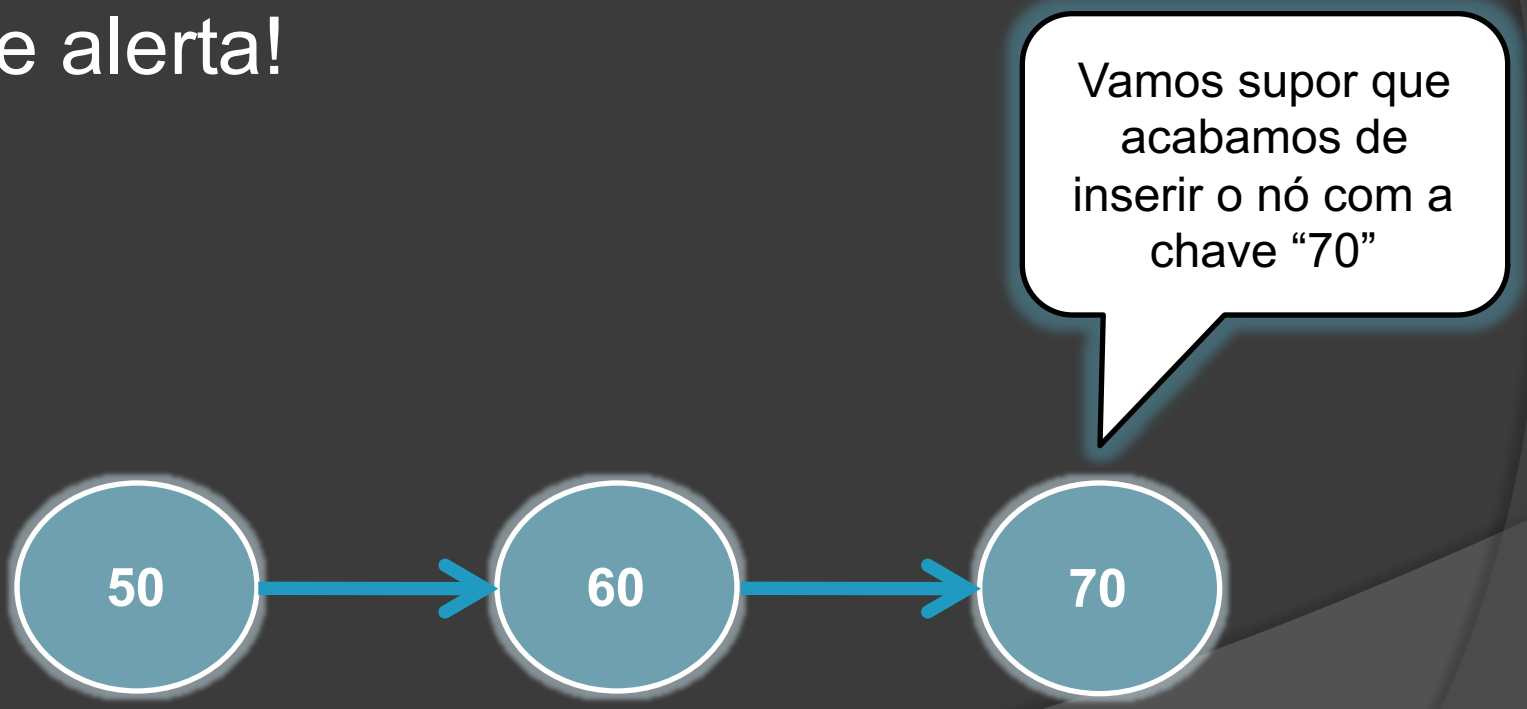
```
template <typename TIPO>
void inserir (TNo<TIPO> *&no, int chave
      if(no == NULL){
            no = new TNo<TIPO>;
            no->chave = chave;
            no->dado  = dado;
            no->esq   = NULL;
            no->dir   = NULL;
      }else{
            if(chave > no->chave){
                  inserir(no->dir, chave, dado);
                  //(aqui...)
            }else{
                  if(chave < no->chave){
                        inserir(no->esq, chave, dado);
                        //(aqui...)
                  }
            }
      }
}
```

Podemos trabalhar com um tipo de retorno.

Identificando quando teremos uma violação

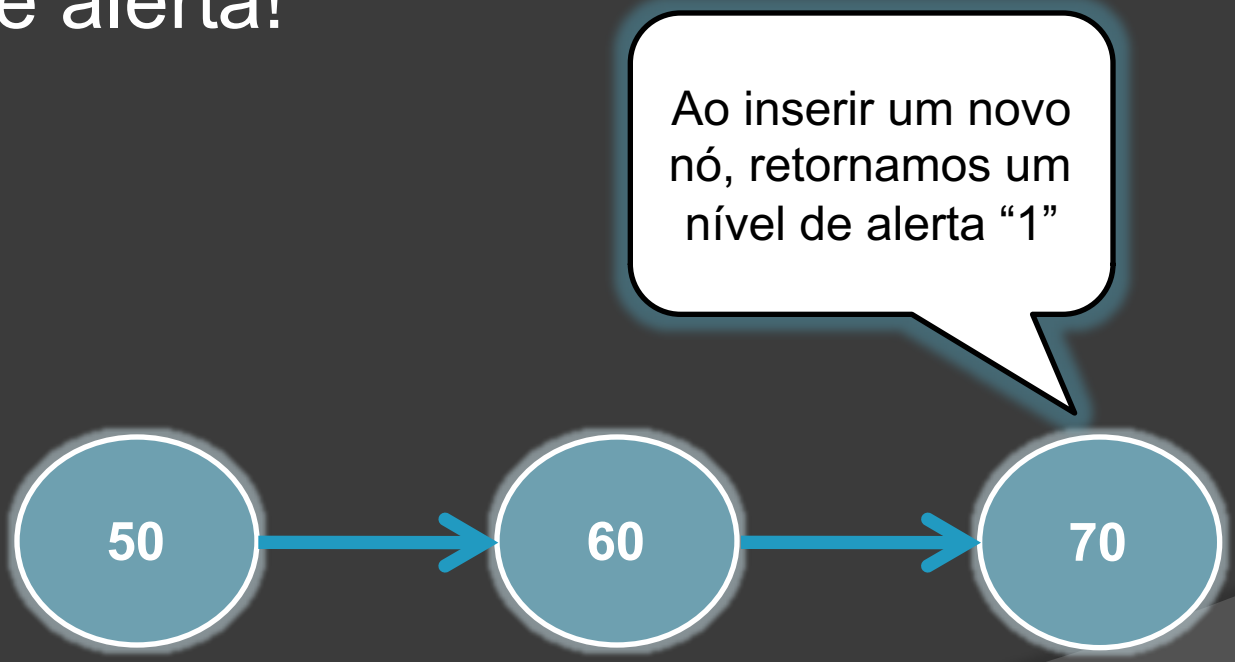
Inserção

- Imaginemos o tipo de retorno como um número inteiro, que representa o nível de alerta!



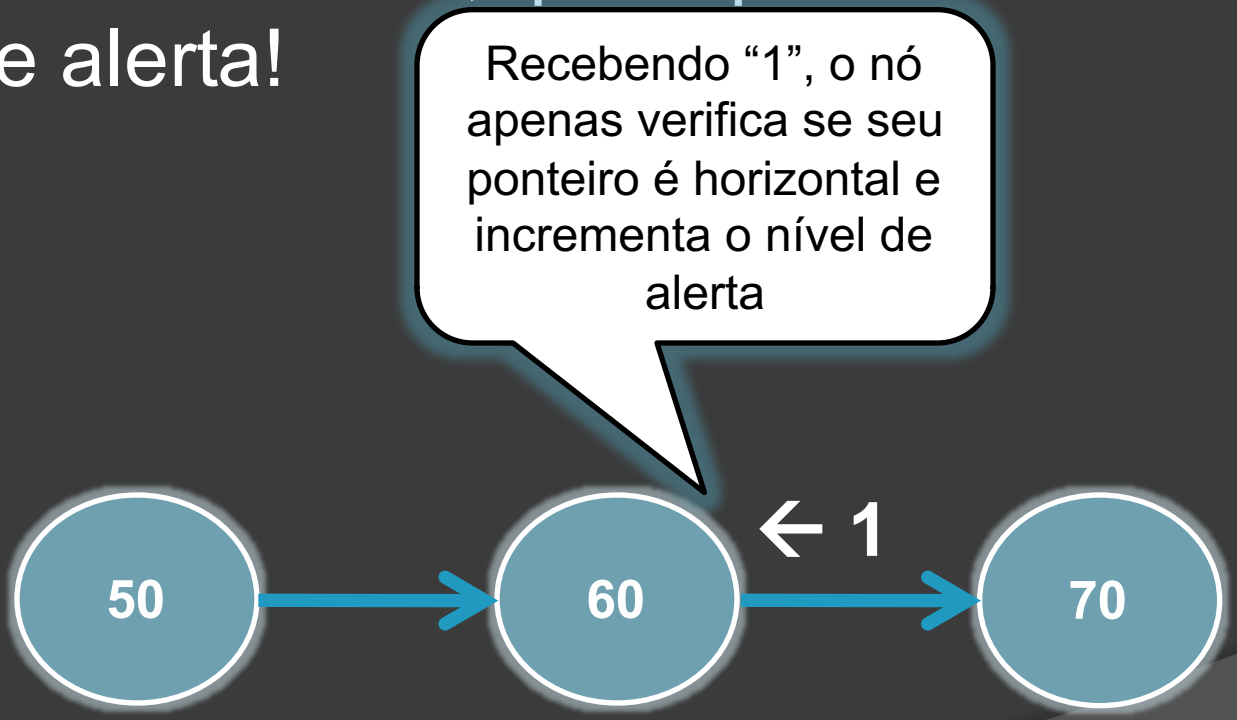
Inserção

- Imaginemos o tipo de retorno como um número inteiro, que representa o nível de alerta!



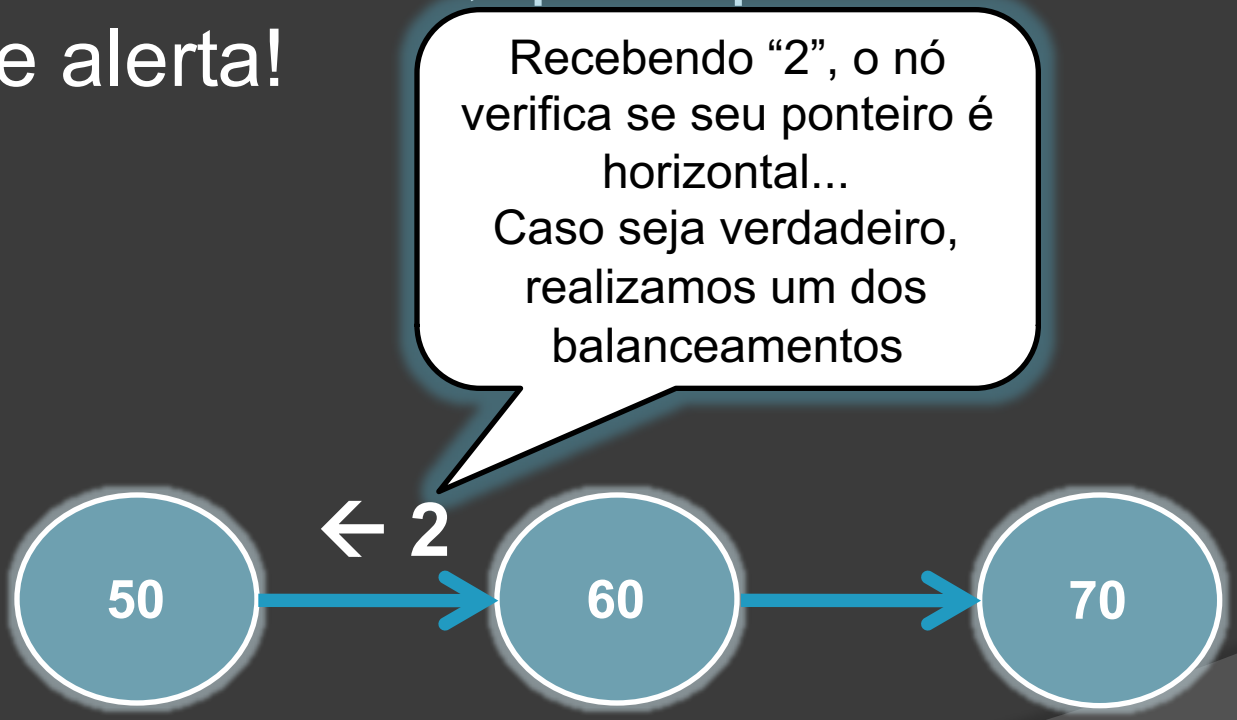
Inserção

- Imaginemos o tipo de retorno como um número inteiro, que representa o nível de alerta!



Inserção

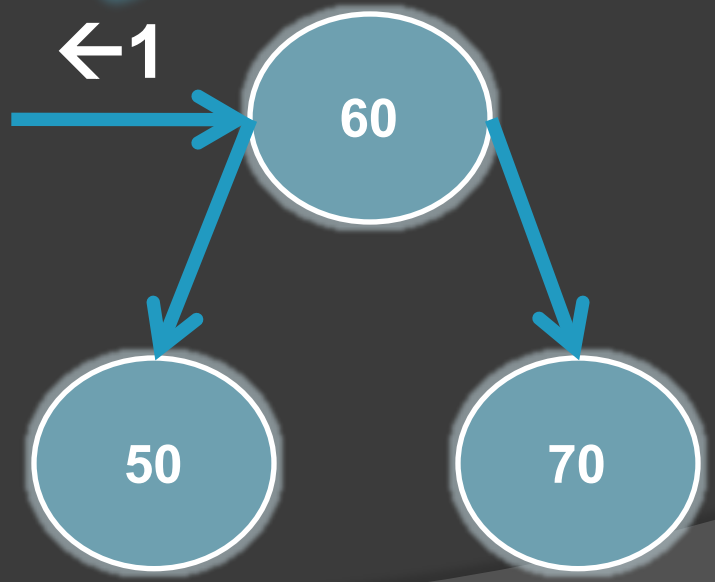
- Imaginemos o tipo de retorno como um número inteiro, que representa o nível de alerta!



Inserção

- Imagin… …um
  número… …el
  de alerta!

Após realizado o balanceamento, fazemos retornamos “1”, pois a cada subárvore recém balanceada de estar sendo referenciada com um apontamento horizontal



```
template <typename TIPO>
int inserir (TNo<TIPO> *&no, int chave,
      if(no == NULL){
            no = new TNo<TIPO>;
            no->chave = chave;
            no->dado  = dado;
            no->esq   = NULL;
            no->dir   = NULL;
            return 1;
      }else{
            if(chave > no->chave){
                  inserir(no->dir, chave, dado);
            }else{
                  if(chave < no->chave){
                        inserir(no->esq, chave, dado);
                  }
            }
      }
}
```

A cada inserção, retornamos “1”

```
template <typename TIPO>
int inserir (TNo<TIPO> *&no, int chave, TIPO dado){
   int n = 0;
   if(no == NULL){
      no = new TNo<TIPO>;
      no->chave = chave;
      no->dado  = dado;
      no->esq   = NULL;
      no->dir   = NULL;
      return 1;
   }else{
      if(chave > no->chave){
         n = inserir(no->dir, chave, dado);
         if (n == 1){
            no->orientD = HORIZONTAL;
            n++;
         }else{
            if(n == 2 && no->orientD == HORIZONTAL){
               n=1
               balanceaDir(no, chave);
            }else{
               n=0;
            }...return n;...}
```

Com a nossa ideia...

E agora?

- Mão na massa!