

Árvores

Árvores Binárias de Busca: Busca e Travessia

Prof. Edson Alves - UnB/FGA

2018

1. Tamanho de árvores binárias de busca
2. Busca em árvores binárias de busca
3. Travessia em árvores binárias de busca

Tamanho de árvores binárias de busca

Tamanho de uma árvore binária de busca

- O tamanho de uma árvore corresponde ao número N de nós que ela possui
- O algoritmo abaixo computa este valor em $O(N)$:
 1. Comece no nó raiz e inicialize a variável tamanho com zero
 2. Para cada nó não nulo:
 - i. Incremente em uma unidade da variável tamanho
 - ii. Some à variável o tamanho da subárvore esquerda do nó
 - iii. Some à variável o tamanho da subárvore direita do nó
- O algoritmo acima é recursivo
- O caso base é a árvore vazia, que tem tamanho zero
- O passo 2 corresponde à chamada recursiva, uma vez que cada uma das subárvores são, de fato, árvores

Implementação do tamanho de uma BST em C++

```
1  template<typename T>
2  class BST {
3  private:
4      struct Node {
5          T info;
6          Node *left, *right;
7      };
8
9      Node *root;
10
11     int size(const Node *node) const
12     {
13         return node ? size(node->left) + size(node->right) + 1 : 0;
14     }
15
16 public:
17     BST() : root(nullptr) {}
18
19     int size() const { return size(root); }
20 };
```

Notas sobre o algoritmo de tamanho

- O algoritmo apresentado pode ser adaptado para computar o tamanho de árvores no caso geral
- Basta retornar a soma dos tamanho de todas as subárvore associadas ao nó atual
- Conforme dito, no pior caso a complexidade é $O(N)$
- Este é um exemplo de algoritmo cuja complexidade independente da forma da árvore, e é $O(N)$ mesmo em árvores balanceadas (pois é preciso visitar cada nó ao menos uma vez)
- Uma alternativa é adicionar um membro `size` na classe `BST`, e atualizá-la a cada inserção e a cada remoção
- Deste modo, o tamanho da árvore pode ser obtido em $O(1)$, bastando retornar o valor desta variável

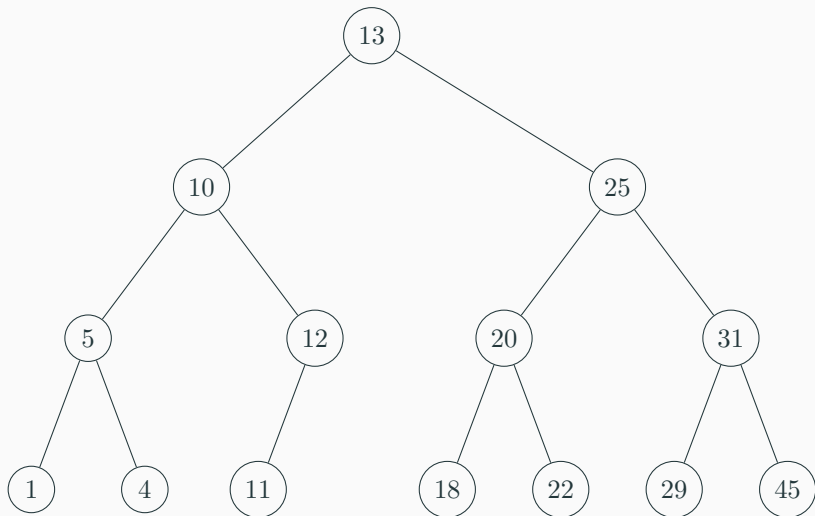
Busca em árvores binárias de busca

Busca em árvores binárias de busca

- A busca em uma árvore binária de busca procura responder a seguinte questão: a informação x está armazenada em algum dos nós da árvore?
- A importância desta operação nesta estrutura é tamanha que, de fato, a nomeia
- O algoritmo abaixo busca a informação x em uma árvore binária de busca:
 1. Comece no nó raiz
 2. Para cada nó não nulo:
 - 2.1 Se x está armazenado no nó, retorne verdadeiro
 - 2.2 Se x for menor do que o valor armazenado no nó, vá para a subárvore à esquerda
 - 2.3 Se x for maior do que o valor armazenado no nó, vá para a subárvore à direita
 3. Retorne falso

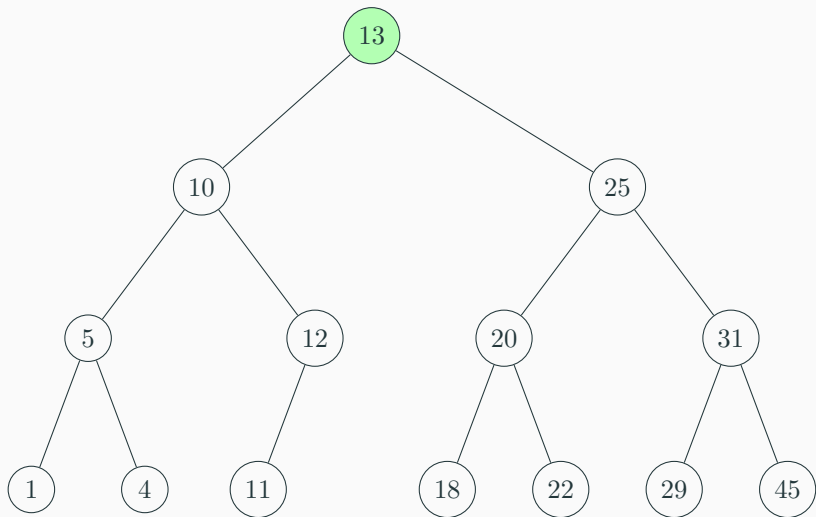
Exemplo de inserção em árvore binária de busca

Elemento a ser localizado: 24



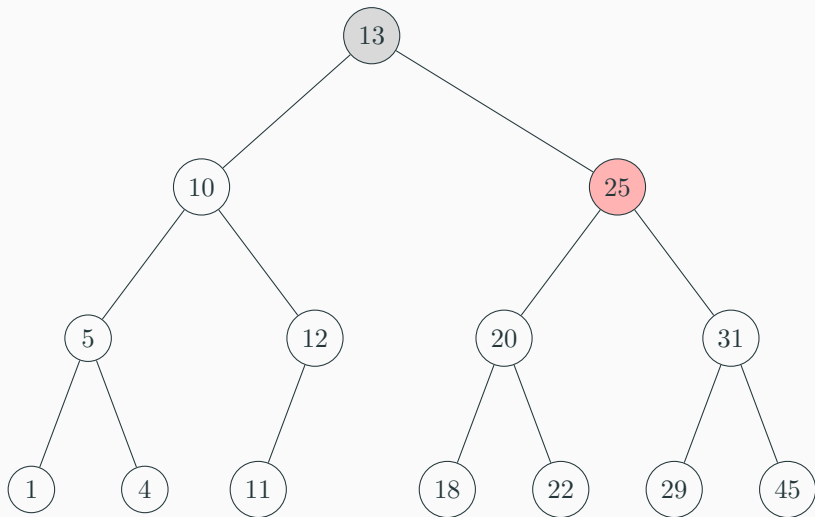
Exemplo de inserção em árvore binária de busca

Elemento a ser localizado: 24



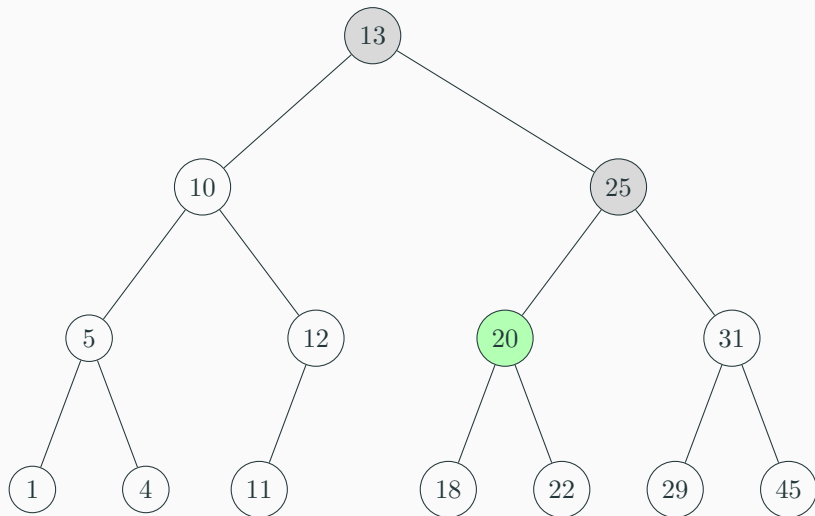
Exemplo de inserção em árvore binária de busca

Elemento a ser localizado: 24



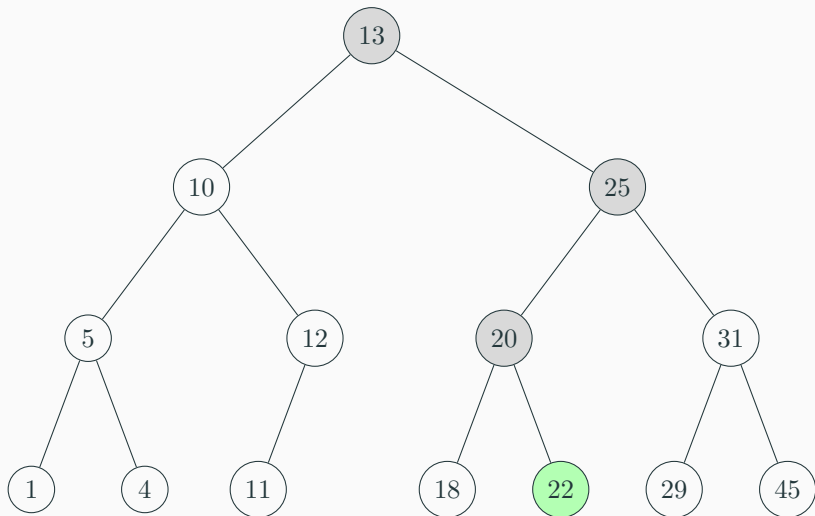
Exemplo de inserção em árvore binária de busca

Elemento a ser localizado: 24



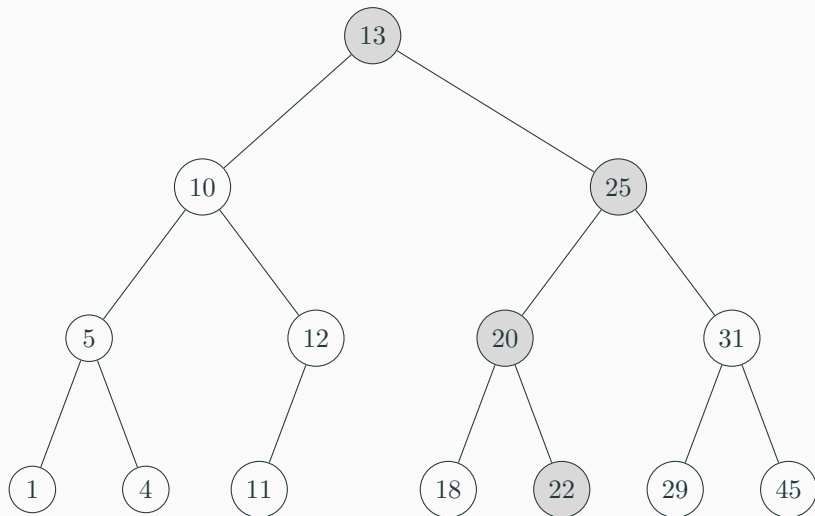
Exemplo de inserção em árvore binária de busca

Elemento a ser localizado: 24



Exemplo de inserção em árvore binária de busca

Elemento a ser localizado: 24



Implementação iterativa do algoritmo de busca

```
1  template<typename T>
2  class BST {
3  private:
4      struct Node {
5          T info;
6          Node *left, *right;
7      };
8
9      Node *root;
10
```

Implementação iterativa do algoritmo de busca

```
11  bool search(Node *node, const T& info)
12  {
13      while (node)
14      {
15          if (info == node->info)
16              return true;
17          else if (info < node->info)
18              node = node->left;
19          else
20              node = node->right;
21      }
22
23      return false;
24  }
25
26  public:
27      BST() : root(nullptr) {}
28
29      bool search(const T& info) const { return search(root, info); }
30  };
```


Notas sobre o algoritmo de busca

- O algoritmo de busca em árvores binárias de busca também pode ser implementado recursivamente:

```
1 bool search(Node *node, const T& info)
2 {
3     if (node == nullptr) return false;
4
5     if (node->info == info) return true;
6
7     return info < node->info ? search(node->left, info) :
8         search(node->right, info);
9 }
```

- Uma variante do algoritmo retorna o ponteiro para o elemento, se encontrado, ou um ponteiro nulo, caso contrário
- A ordem de complexidade, no pior caso, é $O(N)$
- Em árvores balanceadas, o algoritmo é $O(\log N)$

Travessia em árvores binárias de busca

Definição

- A travessia de uma árvore é o processo de visitar cada nó exatamente uma vez
- Visitar significa processar, de algum modo, o nó visitado
- A travessia pode ser interpretada como o processo de linearização de uma árvore
- A definição de travessia não especifica a ordem na qual os nós devem ser visitados
- O número de travessias possíveis de uma árvore é igual o número de permutações de seus nós
- Se a árvore tem n nós, terá $n!$ travessias distintas
- Há, contudo, dois tipos especiais de travessia: travessia por extensão e travessia por profundidade

Travessia por extensão e por profundidade

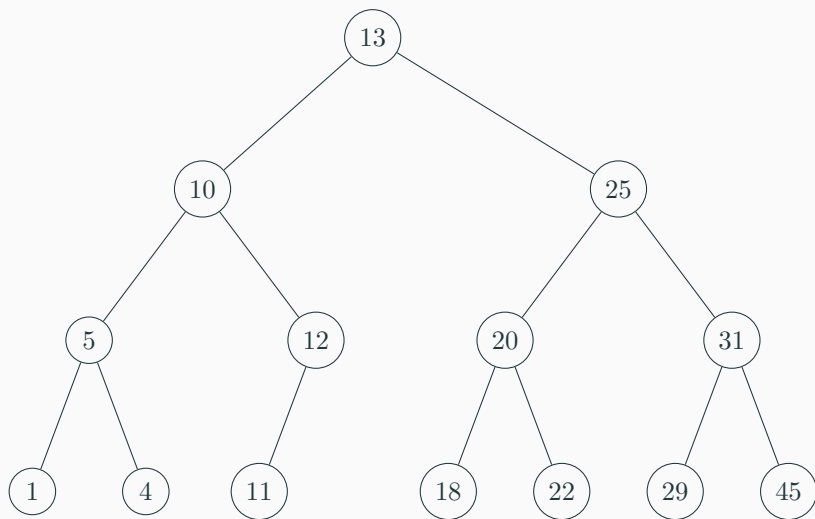
- A travessia por extensão consiste em visitar cada nó começando do nível mais baixo (ou mais alto) e seguindo para baixo (ou para cima) nível a nível, visitando todos os nós daquele nível da esquerda para a direita (ou em sentido oposto)
- Dada a natureza da travessia por extensão, sua implementação pode requerer o auxílio de uma fila
- A travessia por profundidade consiste em ir o mais longe possível à esquerda, retornar até o primeiro cruzamento, tomar à direita e novamente ir o máximo para a esquerda, até que todos os nós tenham sido visitados
- A travessia por profundidade pode ser implementada recursivamente
- Também pode ser implementada iterativamente, com o auxílio de uma pilha

Travessias por profundidade notáveis

- A definição de travessia por profundidade não especifica o momento em que o nó deve ser visitado
- Há 3 tarefas de interesse neste caso:
 1. Visitar o nó (V)
 2. Realizar a travessia da subárvore da esquerda (L)
 3. Realizar a travessia da subárvore da direita (R)
- As 6 possíveis permutações destas tarefas são travessias por profundidade válidas
- As travessias por profundidade mais comuns são:
 1. pré-ordem: VLR
 2. em-ordem: LVR
 3. pós-ordem: LRV

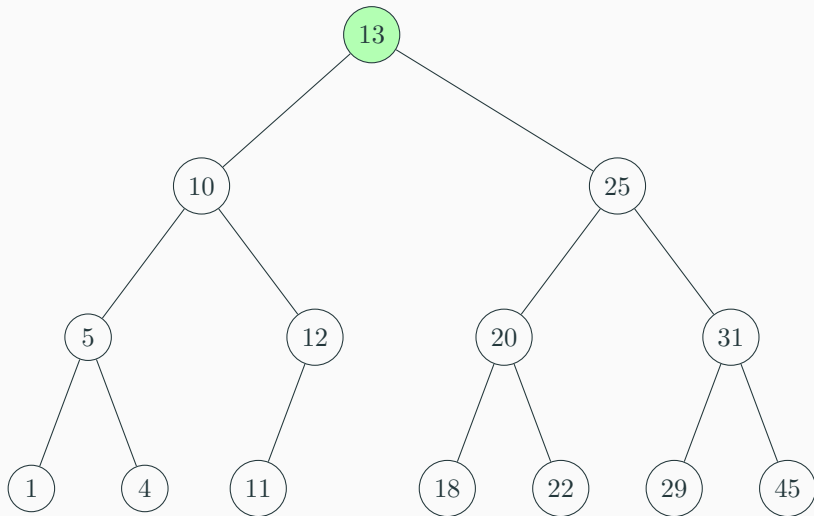
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem:



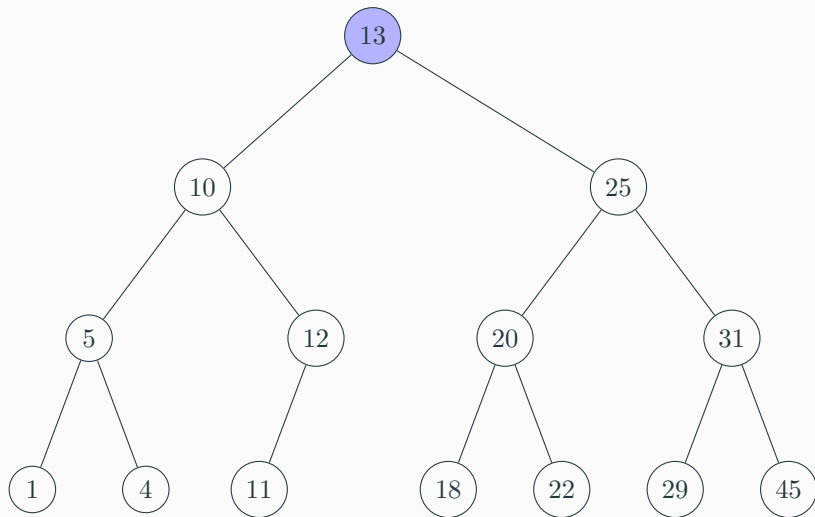
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem:



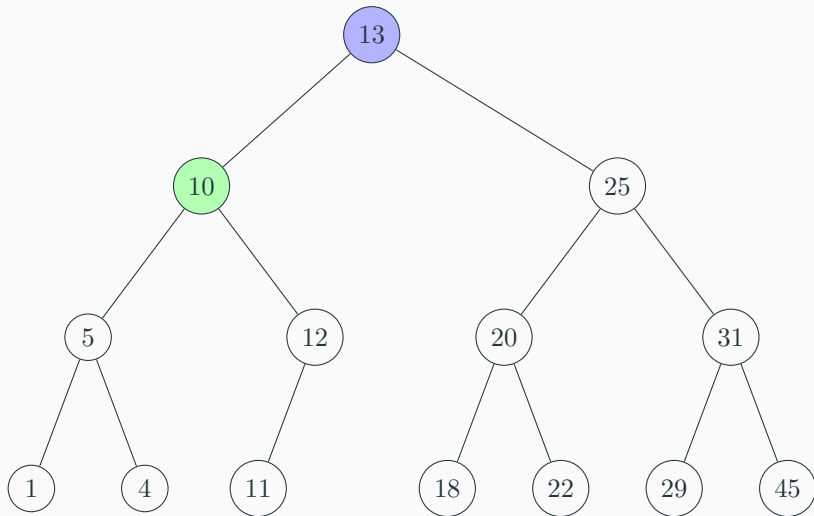
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13



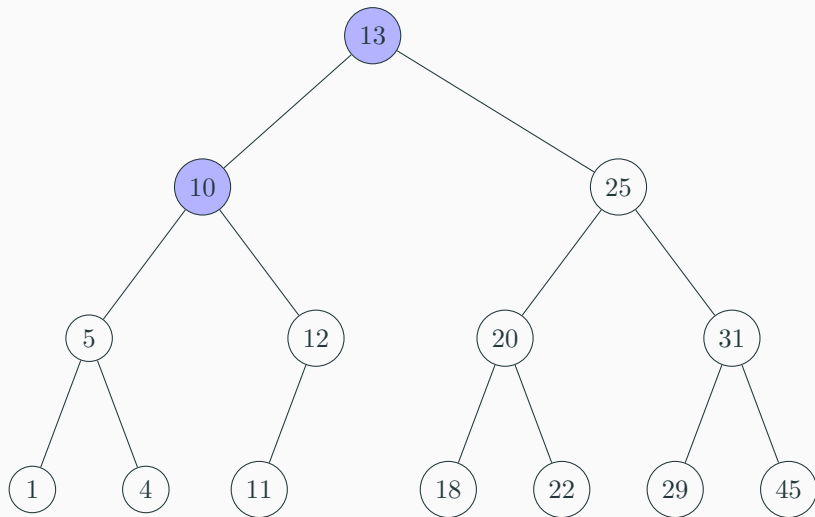
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13



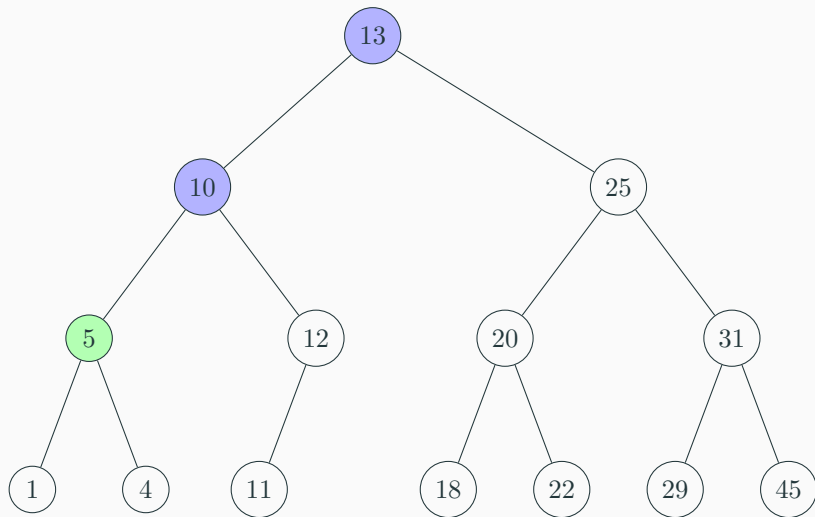
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10



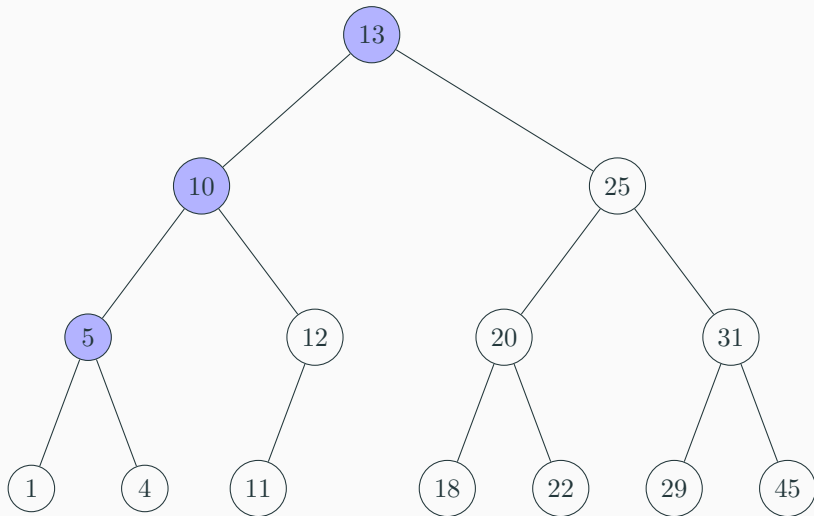
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10



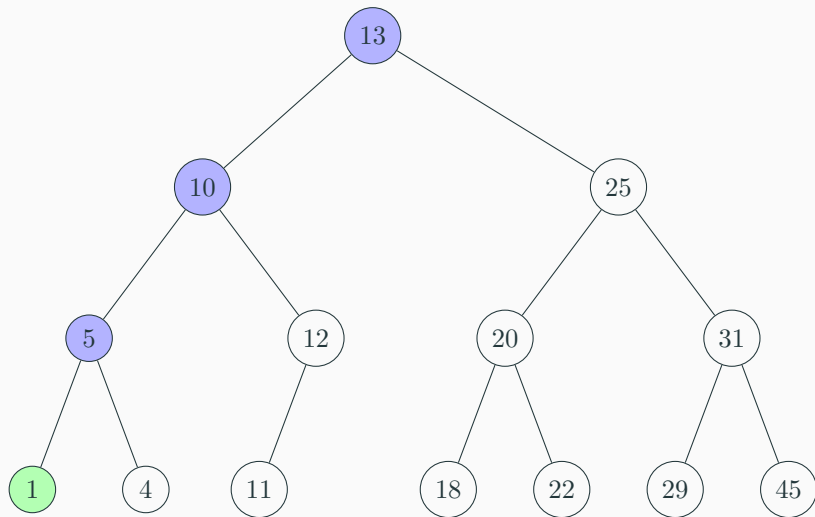
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5



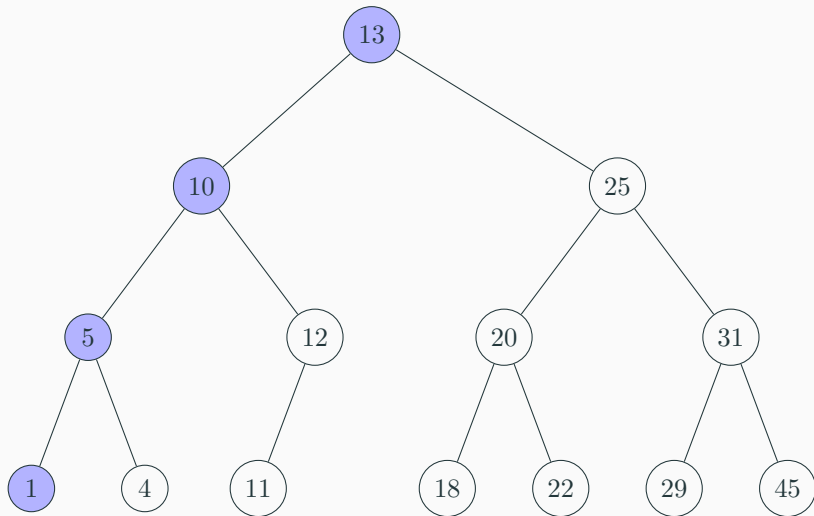
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5



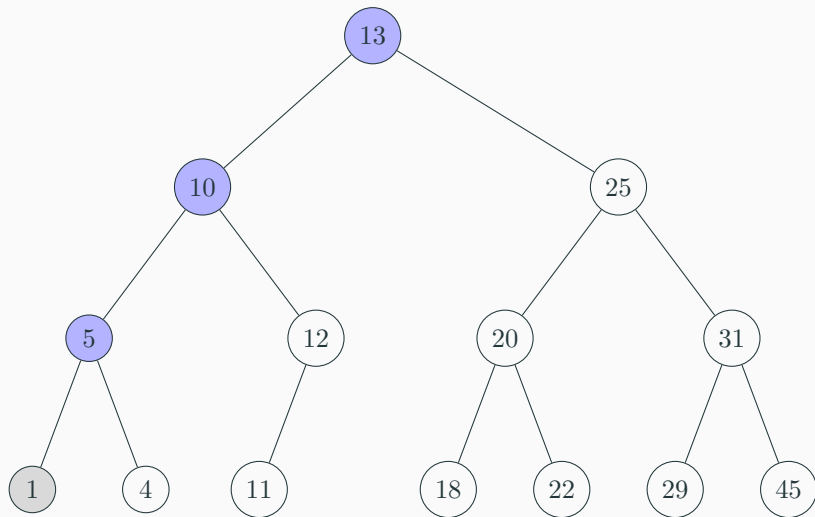
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1



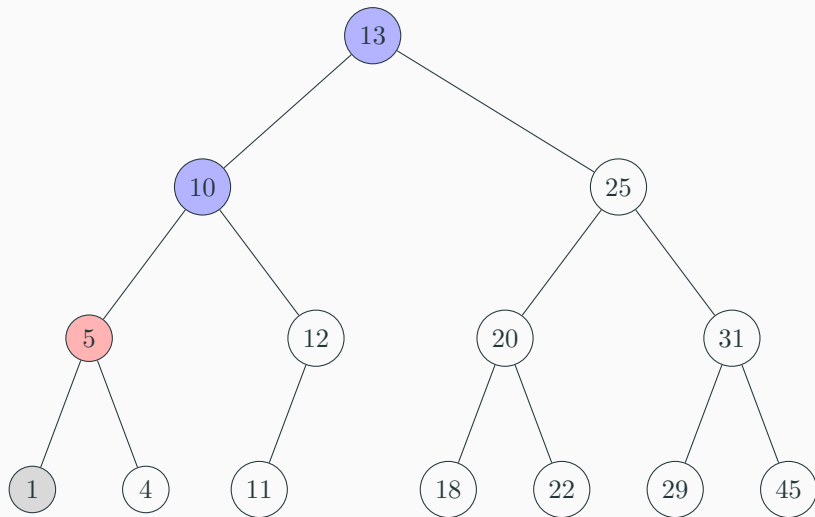
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1



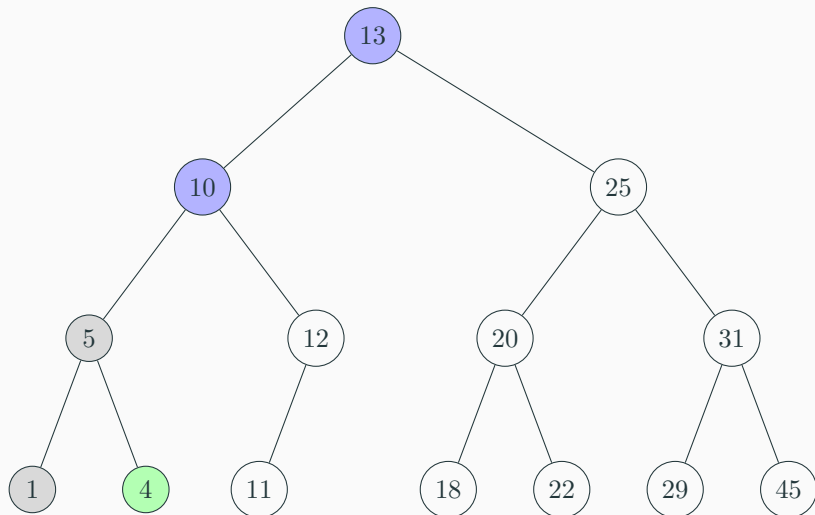
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1



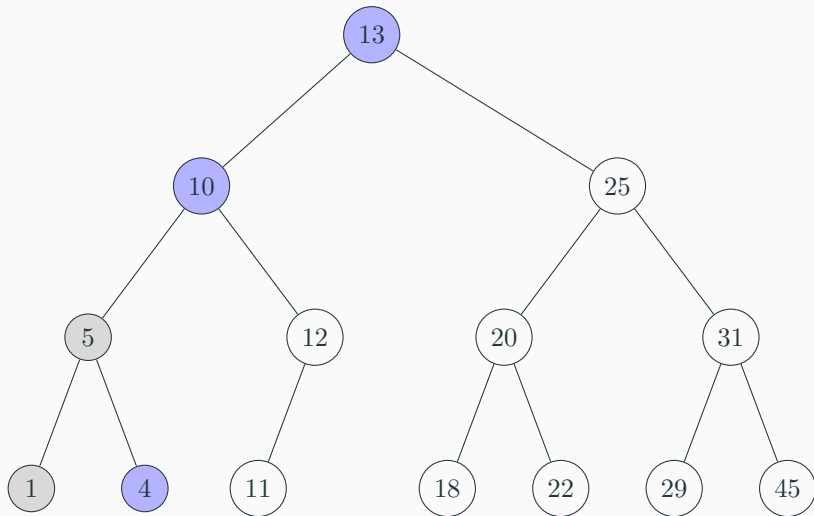
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1



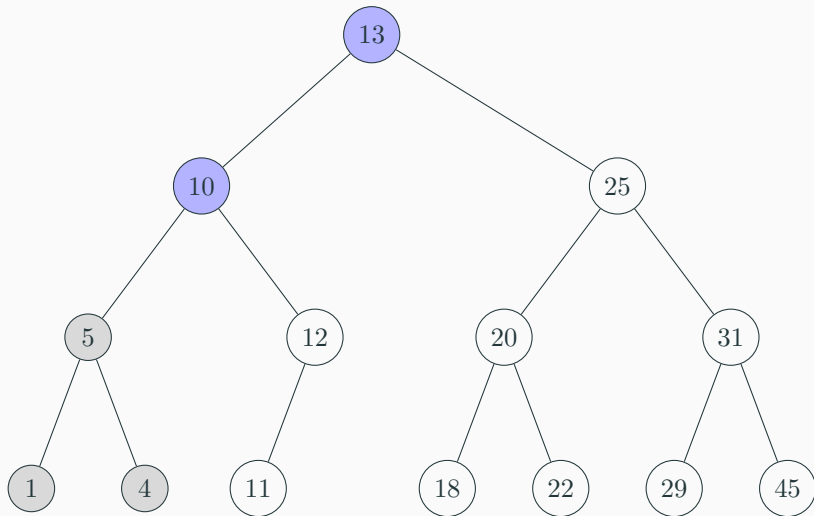
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4



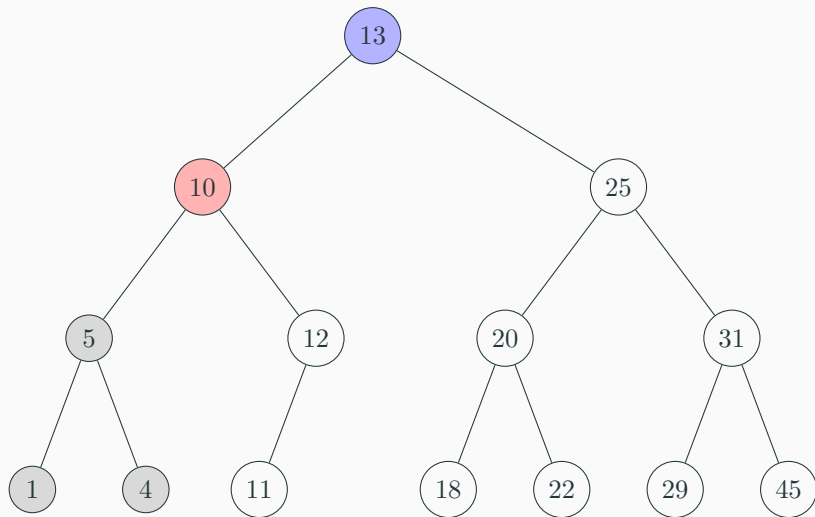
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4



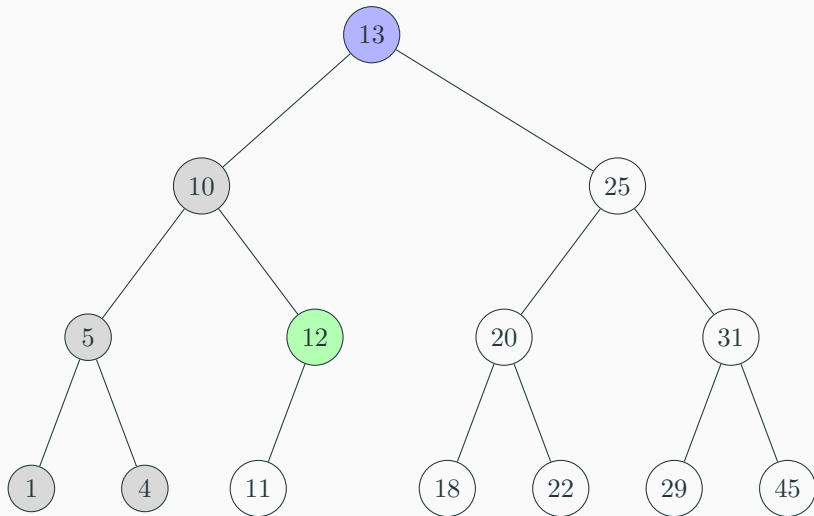
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4



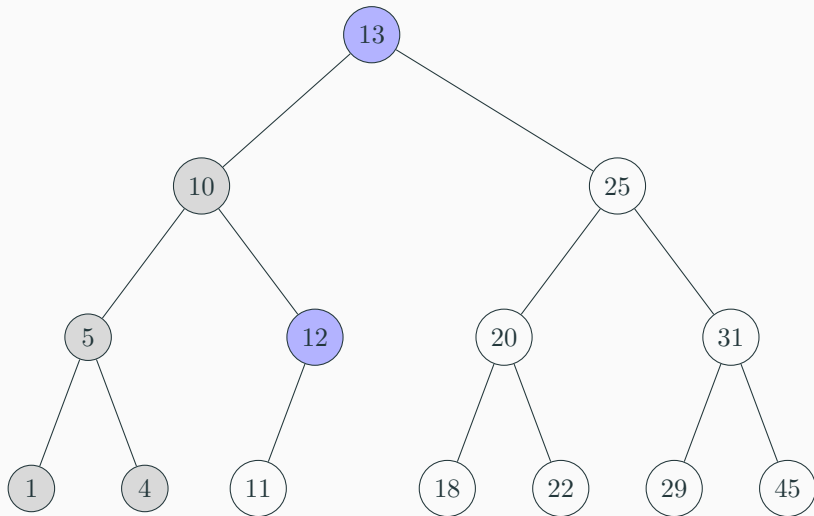
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4



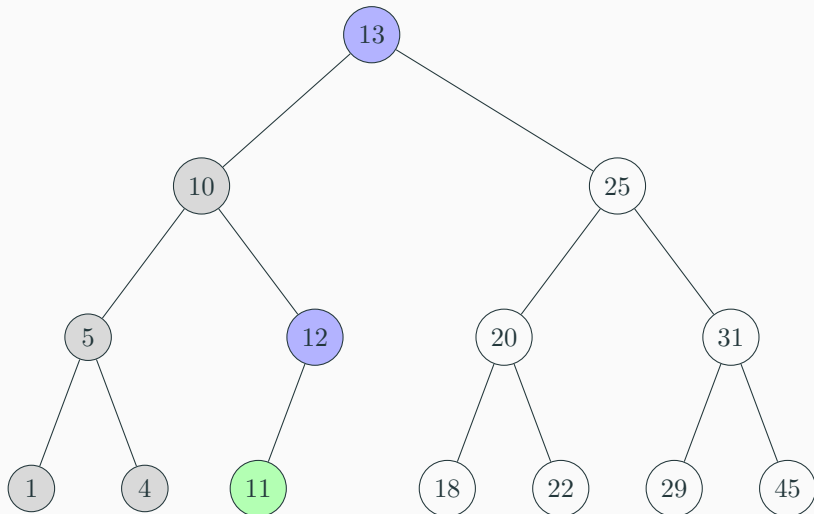
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12



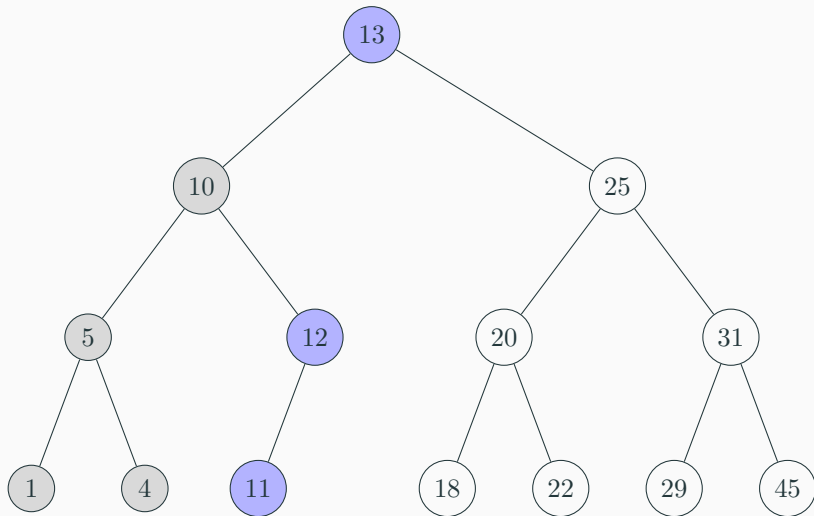
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12



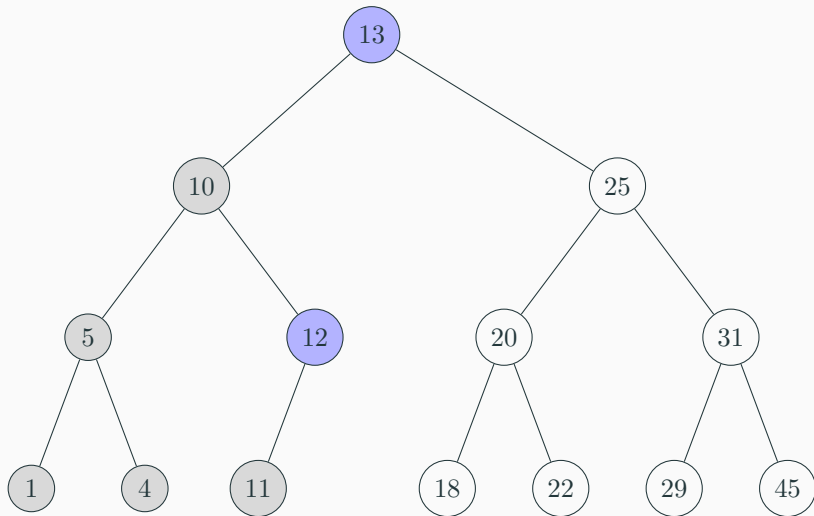
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11



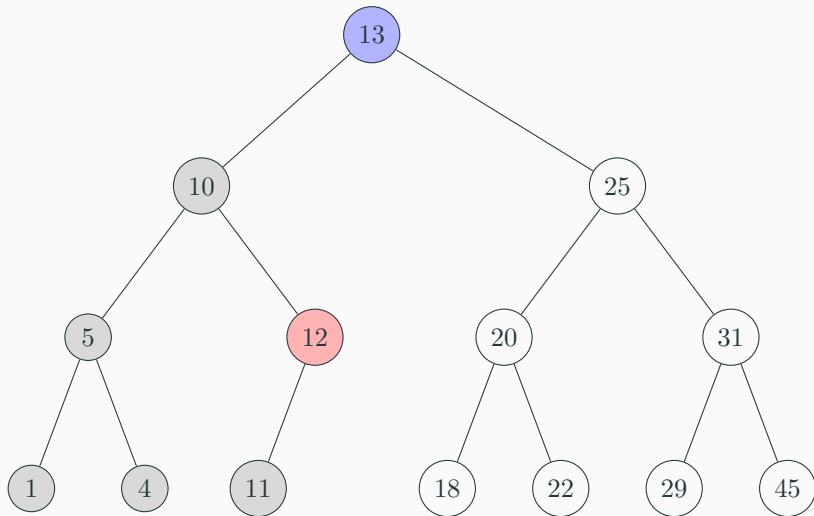
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11



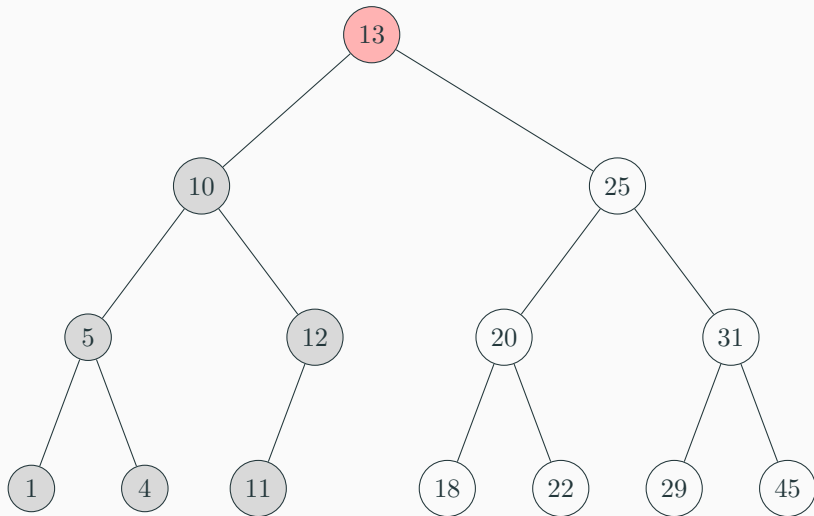
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11



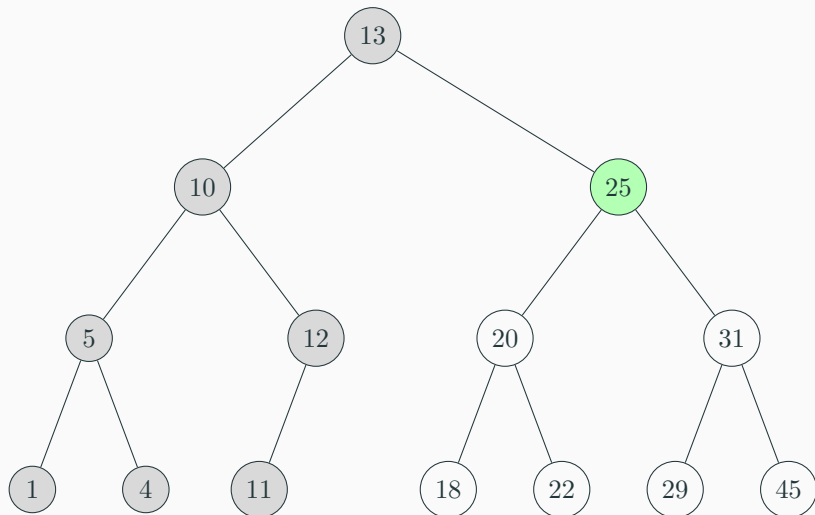
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11



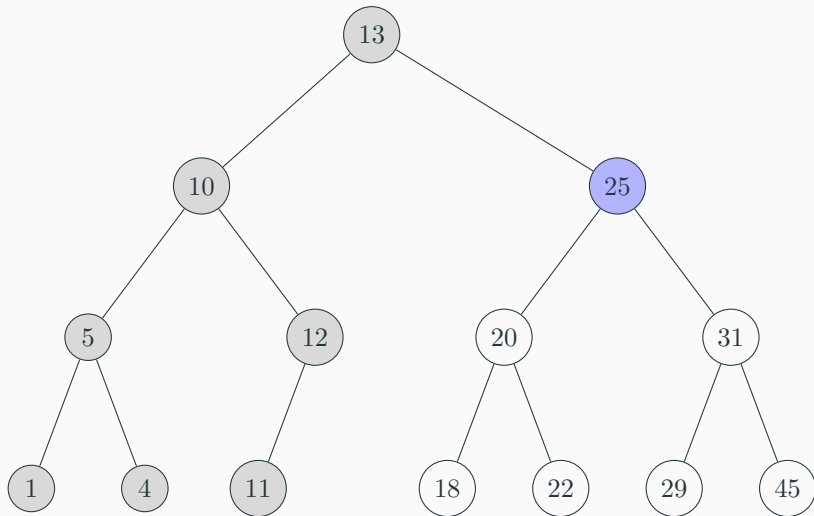
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11



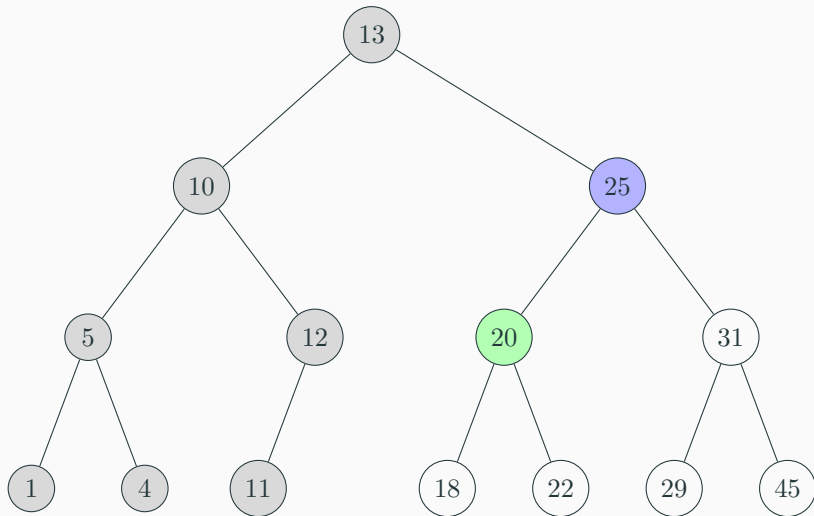
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11, 25



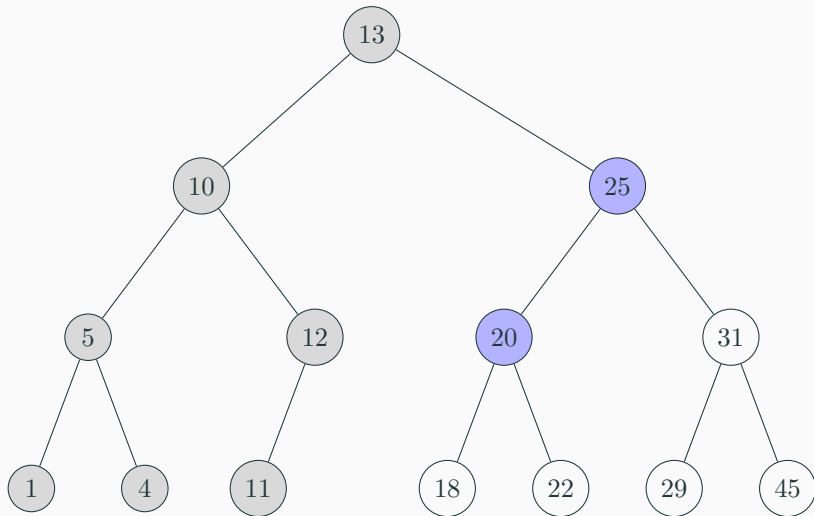
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11, 25



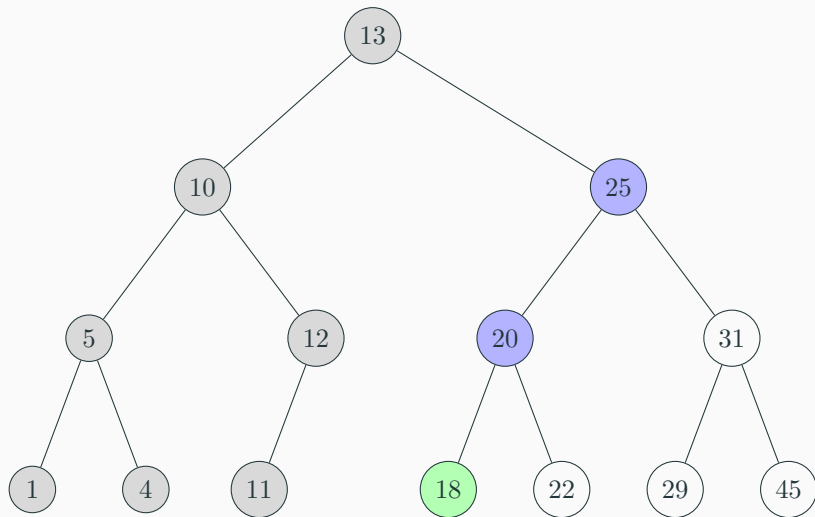
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11, 25, 20



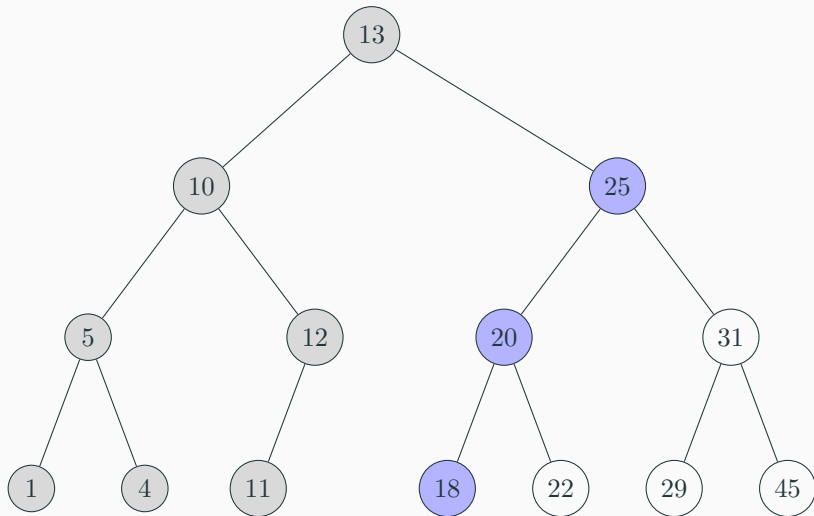
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11, 25, 20



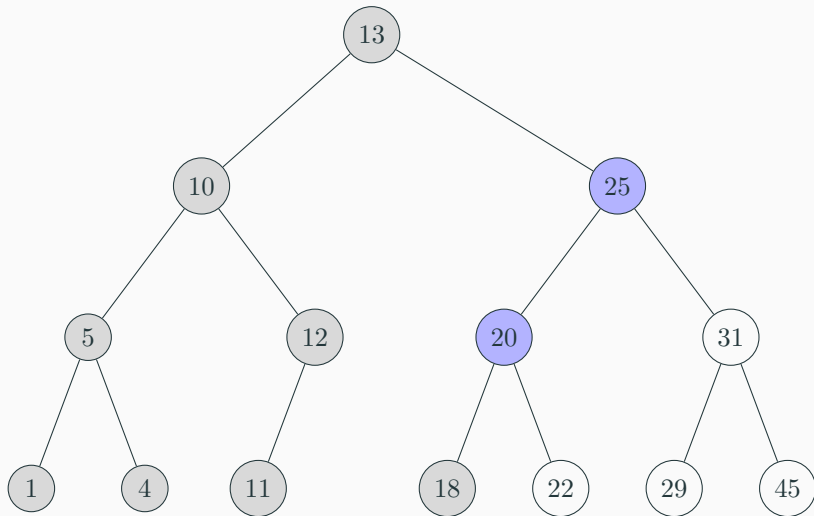
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11, 25, 20, 18, 22, 31, 29, 45



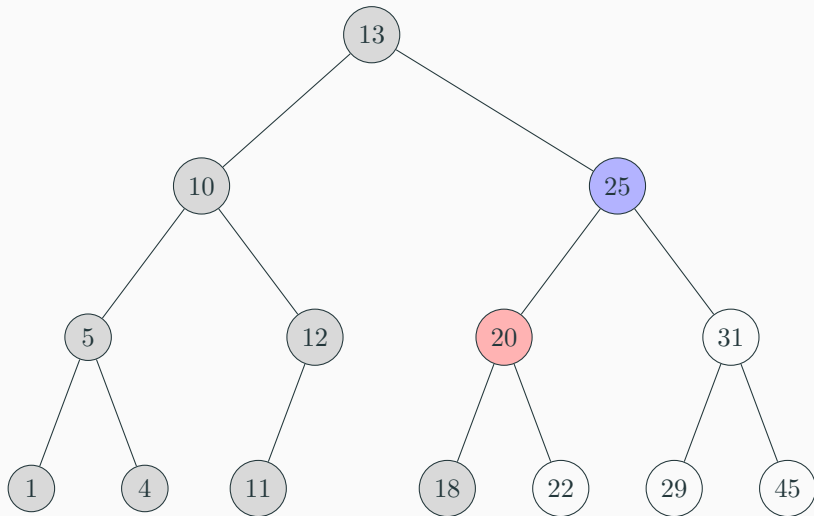
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11, 25, 20, 18, 31, 29, 45



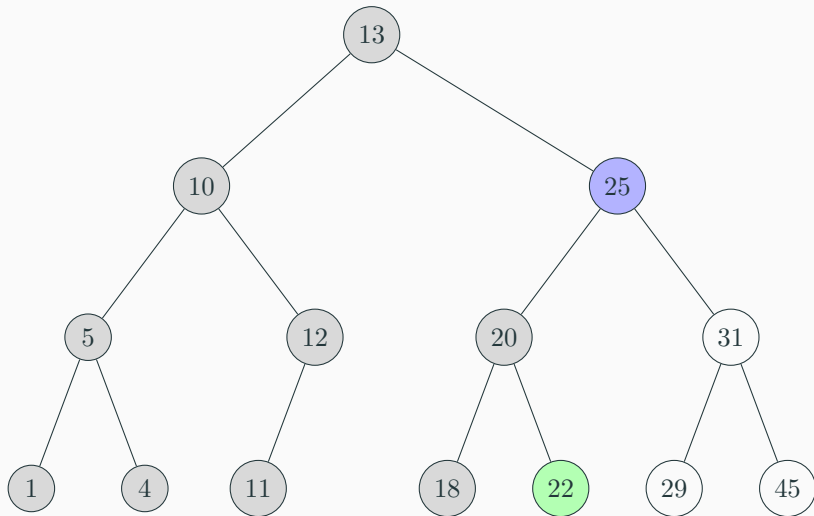
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11, 25, 20, 18, 31, 29, 45



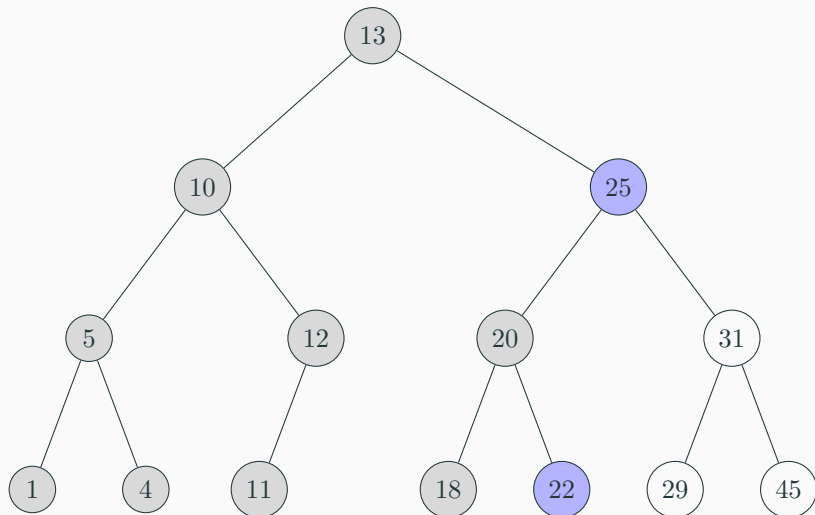
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11, 25, 20, 18



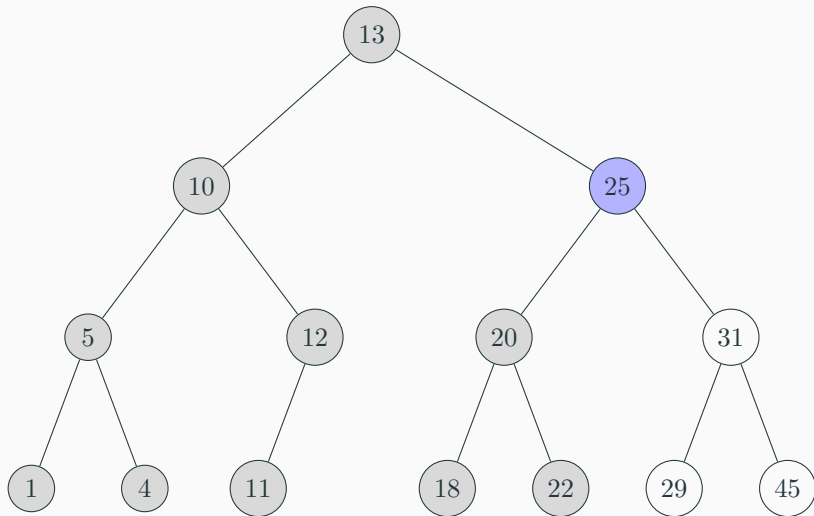
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11, 25, 20, 18, 22



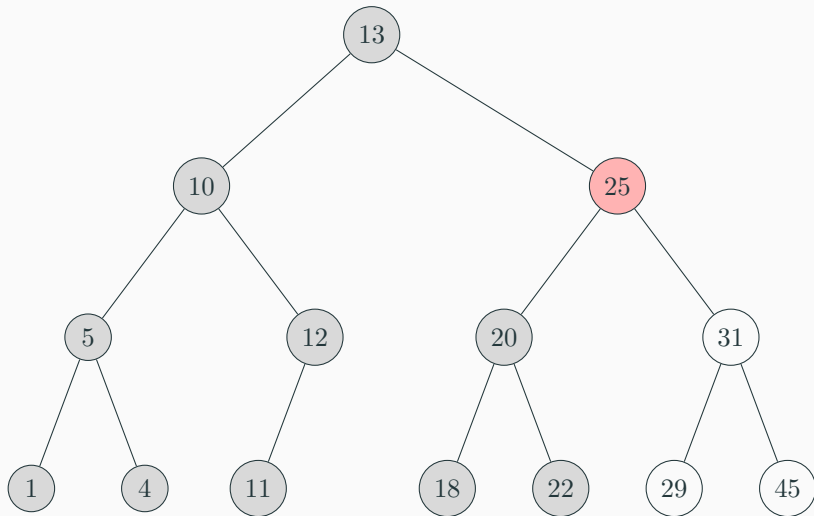
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11, 25, 20, 18, 22, 31, 29, 45



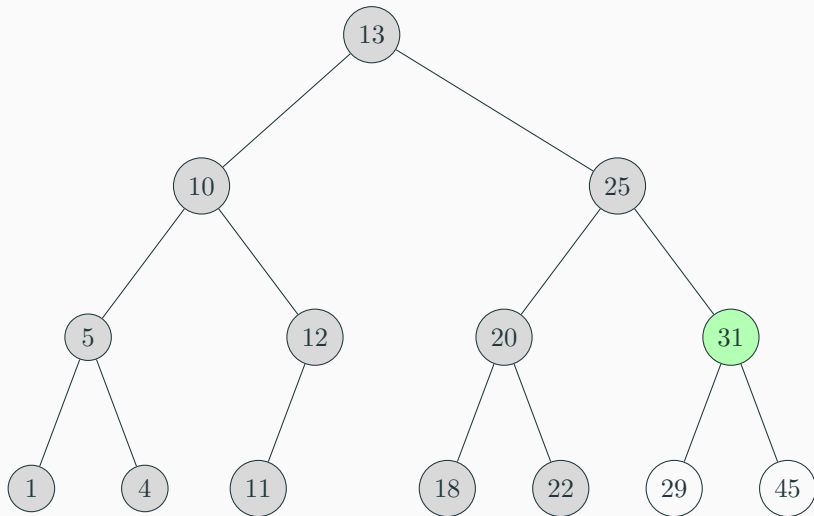
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11, 25, 20, 18, 22, 31, 29, 45



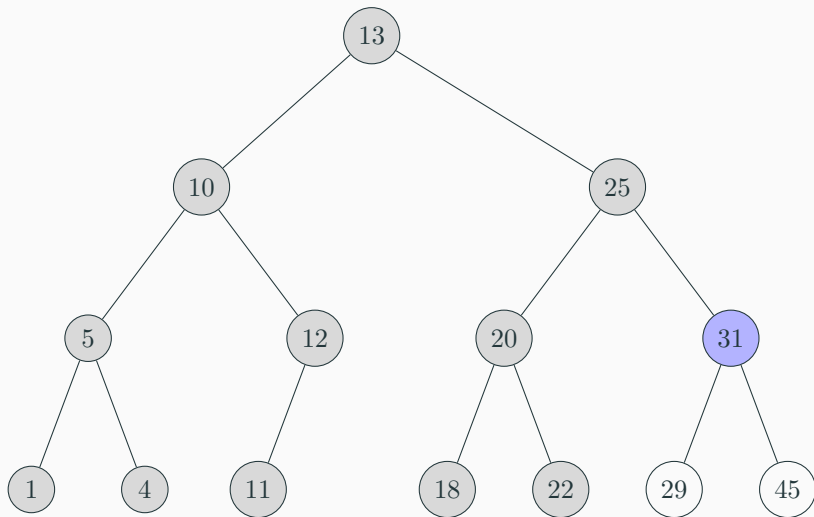
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11, 25, 20, 18, 22



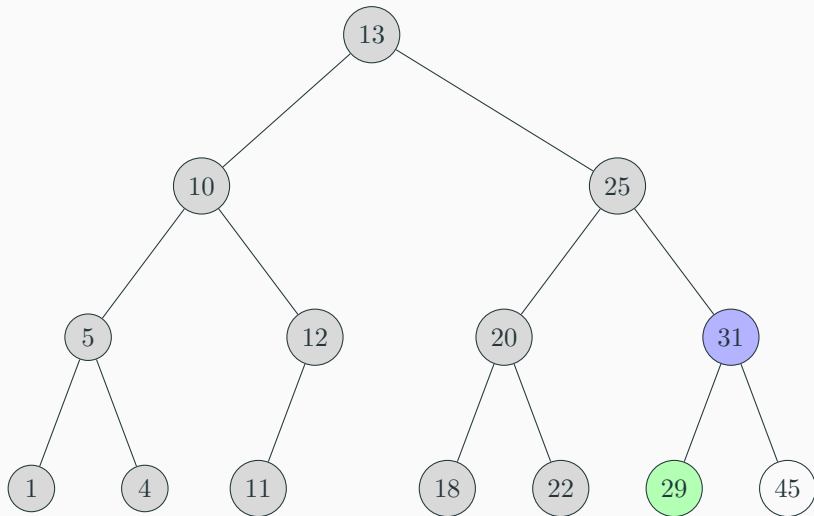
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11, 25, 20, 18, 22, 31



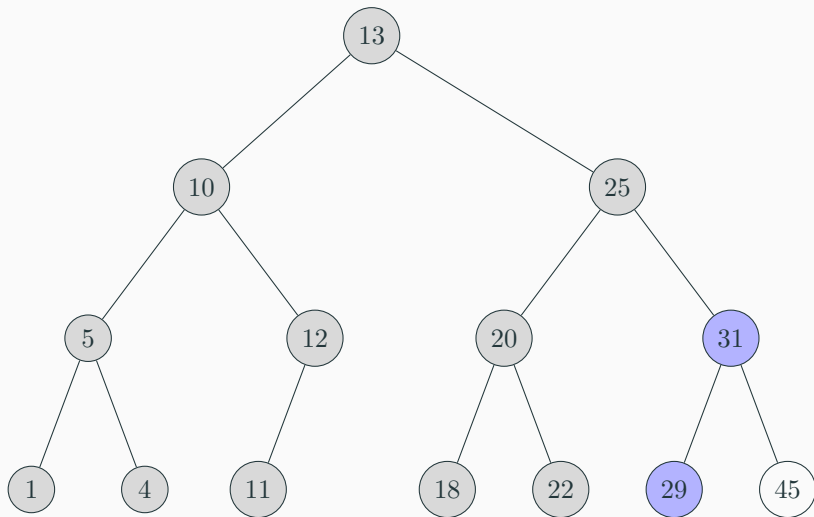
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11, 25, 20, 18, 22, 31



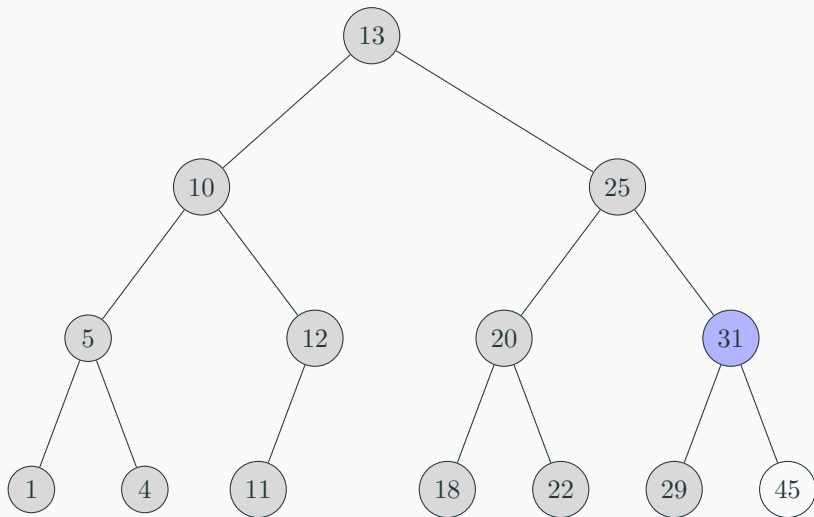
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11, 25, 20, 18, 22, 31, 29



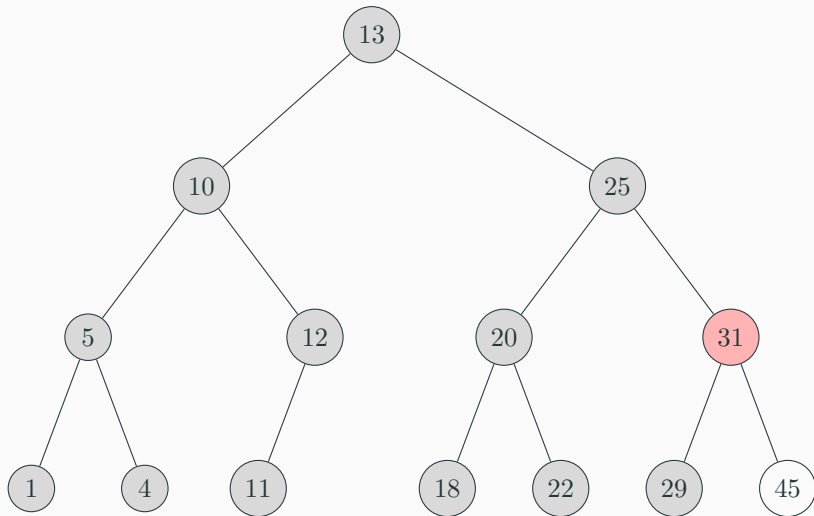
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11, 25, 20, 18, 22, 31, 29



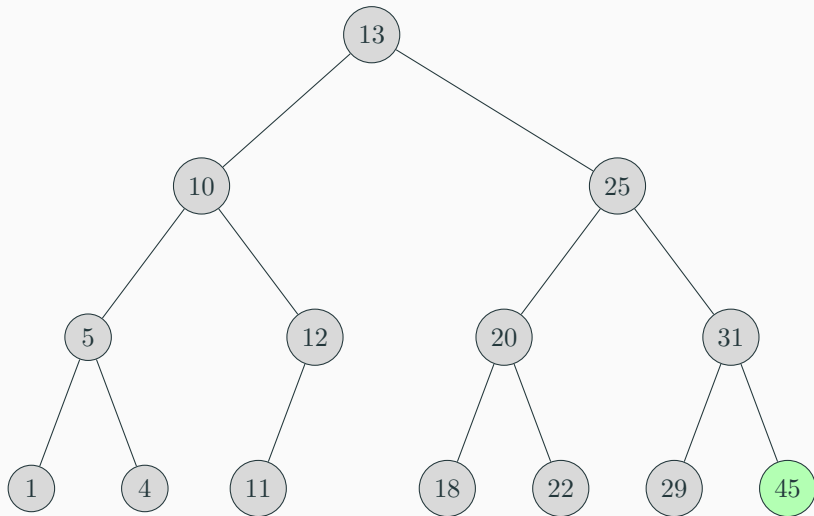
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11, 25, 20, 18, 22, 31, 29



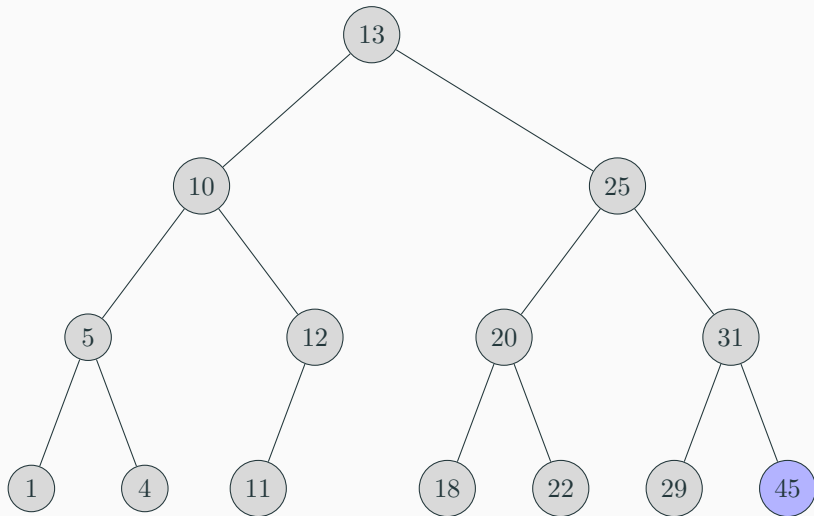
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11, 25, 20, 18, 22, 31, 29



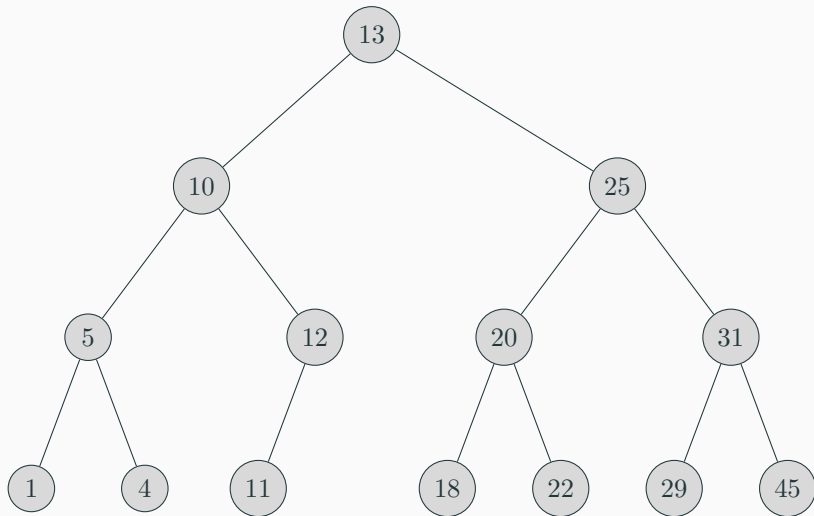
Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11, 25, 20, 18, 22, 31, 29, 45



Exemplo de inserção em árvore binária de busca

Travessia pré-ordem: 13, 10, 5, 1, 4, 12, 11, 25, 20, 18, 22, 31, 29, 45



Implementação das travessias notáveis em C++

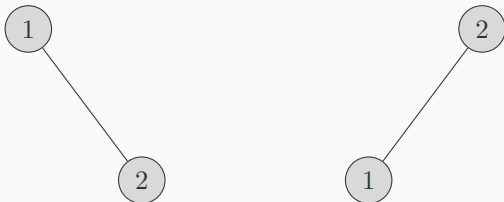
```
1 #include <funcional>
2
3 template<typename T>
4 class BST {
5 private:
6     struct Node {
7         T info;
8         Node *left, *right;
9     };
10
11     Node *root;
12
13     void preorder(Node *node, function<void(Node *)>& visit)
14     {
15         if (node)
16         {
17             visit(node);
18             preorder(node->left);
19             preorder(node->right);
20         }
21     }
```

Implementação das travessias notáveis em C++

```
22
23 void inorder(Node *node, function<void(Node *)>& visit)
24 {
25     if (node)
26     {
27         inorder(node->left);
28         visit(node);
29         inorder(node->right);
30     }
31 }
32
33 void postorder(Node *node, function<void(Node *)>& visit)
34 {
35     if (node)
36     {
37         postorder(node->left);
38         postorder(node->right);
39         visit(node);
40     }
41 }
42 };
```

Reconstrução de árvores binárias a partir de travessias

- Das três travessias notáveis de uma árvore binária de busca, duas permitem a reconstrução da árvore original: a pré-ordem e a pós-ordem
- Veja que a travessia em-ordem não garante a unicidade da árvore: as árvores abaixo tem a mesma travessia em-ordem, e são distintas



Reconstrução de árvores binárias a partir de travessias

- Para uma árvore binária qualquer, um par de travessias, exceto o par pré-ordem/pós-ordem, garante a unicidade de árvore
- Em outras palavras, a travessia em em-ordem, mais um das outras duas travessias, garante a unicidade de árvore
- Isto porque a travessia em-ordem estabelece a ordem relativa dos elementos, a qual fica subentendida numa árvore binária de busca
- Por este motivo a árvore binária de busca pode ser reconstruída usando apenas a pré-ordem ou a pós-ordem
- Para tal, basta inserir os elementos na ordem listada na pré-ordem (ou em ordem reversa, na pós-ordem) e usar a ordenação estabelecida na em-ordem para posicionar os elementos na inserção

Exemplo de construção: pré-ordem e em-ordem

Travessia pré-ordem: A, B, C, D, E, F

Travessia em-ordem: B, C, A, E, D, F

Exemplo de construção: pré-ordem e em-ordem

Travessia pré-ordem: A, B, C, D, E, F

Travessia em-ordem: B, C, A, E, D, F

Exemplo de construção: pré-ordem e em-ordem

Travessia pré-ordem: A, B, C, D, E, F

Travessia em-ordem: B, C, A, E, D, F



Exemplo de construção: pré-ordem e em-ordem

Travessia pré-ordem: A, B, C, D, E, F

Travessia em-ordem: B, C, A, E, D, F



Exemplo de construção: pré-ordem e em-ordem

Travessia pré-ordem: A, B, C, D, E, F

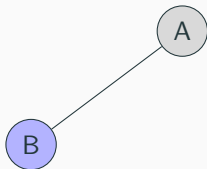
Travessia em-ordem: B, C, A, E, D, F



Exemplo de construção: pré-ordem e em-ordem

Travessia pré-ordem: A, B, C, D, E, F

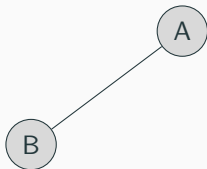
Travessia em-ordem: B, C, A, E, D, F



Exemplo de construção: pré-ordem e em-ordem

Travessia pré-ordem: A, B, C, D, E, F

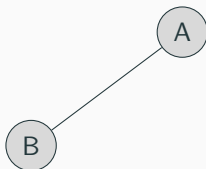
Travessia em-ordem: B, C, A, E, D, F



Exemplo de construção: pré-ordem e em-ordem

Travessia pré-ordem: A, B, C, D, E, F

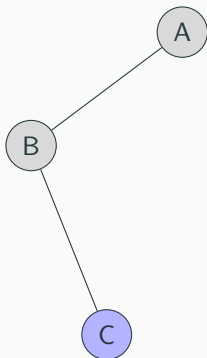
Travessia em-ordem: B, C, A, E, D, F



Exemplo de construção: pré-ordem e em-ordem

Travessia pré-ordem: A, B, C, D, E, F

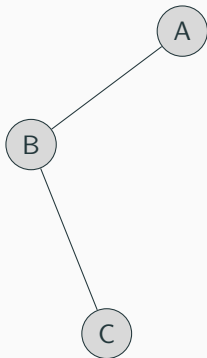
Travessia em-ordem: B, C, A, E, D, F



Exemplo de construção: pré-ordem e em-ordem

Travessia pré-ordem: A, B, C, D, E, F

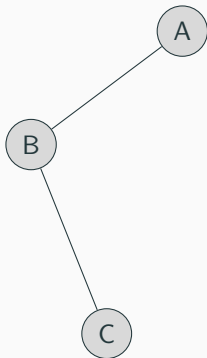
Travessia em-ordem: B, C, A, E, D, F



Exemplo de construção: pré-ordem e em-ordem

Travessia pré-ordem: A, B, C, D, E, F

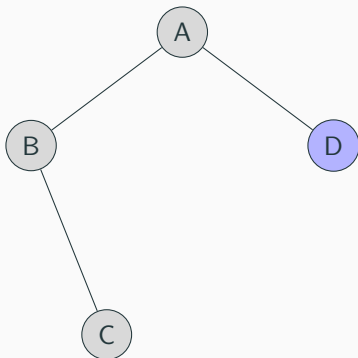
Travessia em-ordem: B, C, A, E, D, F



Exemplo de construção: pré-ordem e em-ordem

Travessia pré-ordem: A, B, C, **D**, E, F

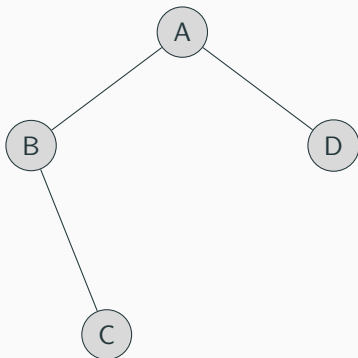
Travessia em-ordem: B, C, **A**, E, **D**, F



Exemplo de construção: pré-ordem e em-ordem

Travessia pré-ordem: A, B, C, D, E, F

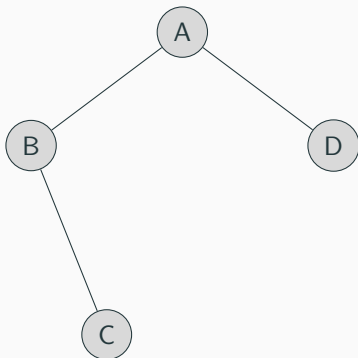
Travessia em-ordem: B, C, A, E, D, F



Exemplo de construção: pré-ordem e em-ordem

Travessia pré-ordem: A, B, C, D, E, F

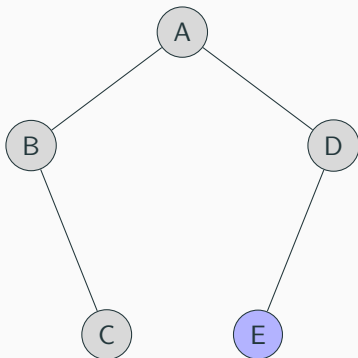
Travessia em-ordem: B, C, A, E, D, F



Exemplo de construção: pré-ordem e em-ordem

Travessia pré-ordem: A, B, C, D, E, F

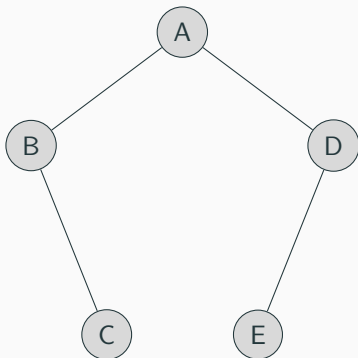
Travessia em-ordem: B, C, A, E, D, F



Exemplo de construção: pré-ordem e em-ordem

Travessia pré-ordem: A, B, C, D, E, F

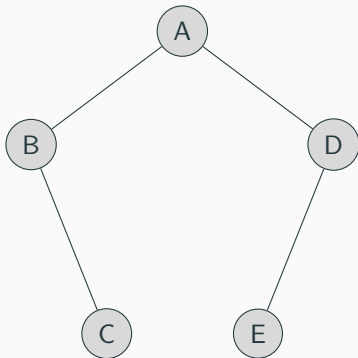
Travessia em-ordem: B, C, A, E, D, F



Exemplo de construção: pré-ordem e em-ordem

Travessia pré-ordem: A, B, C, D, E, **F**

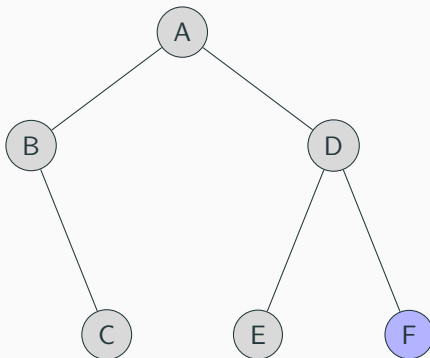
Travessia em-ordem: B, C, **A**, E, D, **F**



Exemplo de construção: pré-ordem e em-ordem

Travessia pré-ordem: A, B, C, D, E, F

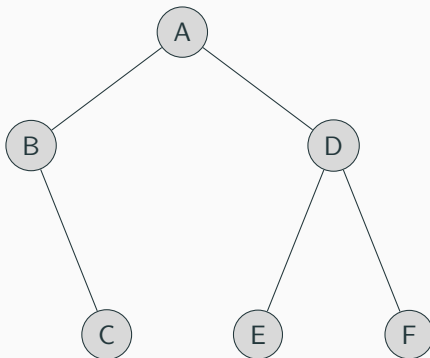
Travessia em-ordem: B, C, A, E, D, F



Exemplo de construção: pré-ordem e em-ordem

Travessia pré-ordem: A, B, C, D, E, F

Travessia em-ordem: B, C, A, E, D, F



1. **DROZDEK**, Adam. *Algoritmos e Estruturas de Dados em C++*, 2002.
2. **KERNIGHAN**, Bryan; **RITCHIE**, Dennis. *The C Programming Language*, 1978.
3. **STROUSTROUP**, Bjarne. *The C++ Programming Language*, 2013.
4. C++ Reference¹.

¹<https://en.cppreference.com/w/>