Árvore Binária de Busca

Estrutura de Dados Avançada — QXD0115

Prof. Atílio Gomes Luiz

gomes.atilio@ufc.br

Universidade Federal do Ceará

1º semestre/2024

Introdução

Motivação

Problema da Busca

Seja $S = \{s_1, s_2, \ldots, s_n\}$ um conjunto de chaves satisfazendo $s_1 < s_2 < \ldots < s_n$. Chamamos S um conjunto ordenável.
Seja x um valor dado. O objetivo é verificar se $x \in S$ ou não. Em caso positivo, localizar x em S, isto é, determinar o índice j tal que $x = s_j$.

Tempo para Inserção, Remoção e Busca

Usando Listas Duplamente Encadeadas:

- Podemos inserir e remover em $O(1)$
- Mas buscar demora $O(n)$

Tempo para Inserção, Remoção e Busca

Usando Listas Duplamente Encadeadas:

- Podemos inserir e remover em $O(1)$
- Mas buscar demora $O(n)$

Se usarmos vetores não-ordenados:

- Podemos inserir e remover em $O(1)$

Tempo para Inserção, Remoção e Busca

Usando Listas Duplamente Encadeadas:

- Podemos inserir e remover em $O(1)$
- Mas buscar demora $O(n)$

Se usarmos vetores não-ordenados:

- Podemos inserir e remover em $O(1)$
  - insira no final
  - para remover, troque com o último e remova o último

Tempo para Inserção, Remoção e Busca

Usando Listas Duplamente Encadeadas:

- Podemos inserir e remover em $O(1)$
- Mas buscar demora $O(n)$

Se usarmos vetores não-ordenados:

- Podemos inserir e remover em $O(1)$
  - insira no final
  - para remover, troque com o último e remova o último
- Mas buscar demora $O(n)$

Tempo para Inserção, Remoção e Busca

Usando Listas Duplamente Encadeadas:

- Podemos inserir e remover em $O(1)$
- Mas buscar demora $O(n)$

Se usarmos vetores não-ordenados:

- Podemos inserir e remover em $O(1)$
  - insira no final
  - para remover, troque com o último e remova o último
- Mas buscar demora $O(n)$

Se usarmos vetores ordenados:

- Podemos buscar em $O(\lg n)$ usando Busca binária
- Mas inserir e remover leva $O(n)$

Tempo para Inserção, Remoção e Busca

Usando Listas Duplamente Encadeadas:

- Podemos inserir e remover em $O(1)$
- Mas buscar demora $O(n)$

Se usarmos vetores não-ordenados:

- Podemos inserir e remover em $O(1)$
  - insira no final
  - para remover, troque com o último e remova o último
- Mas buscar demora $O(n)$

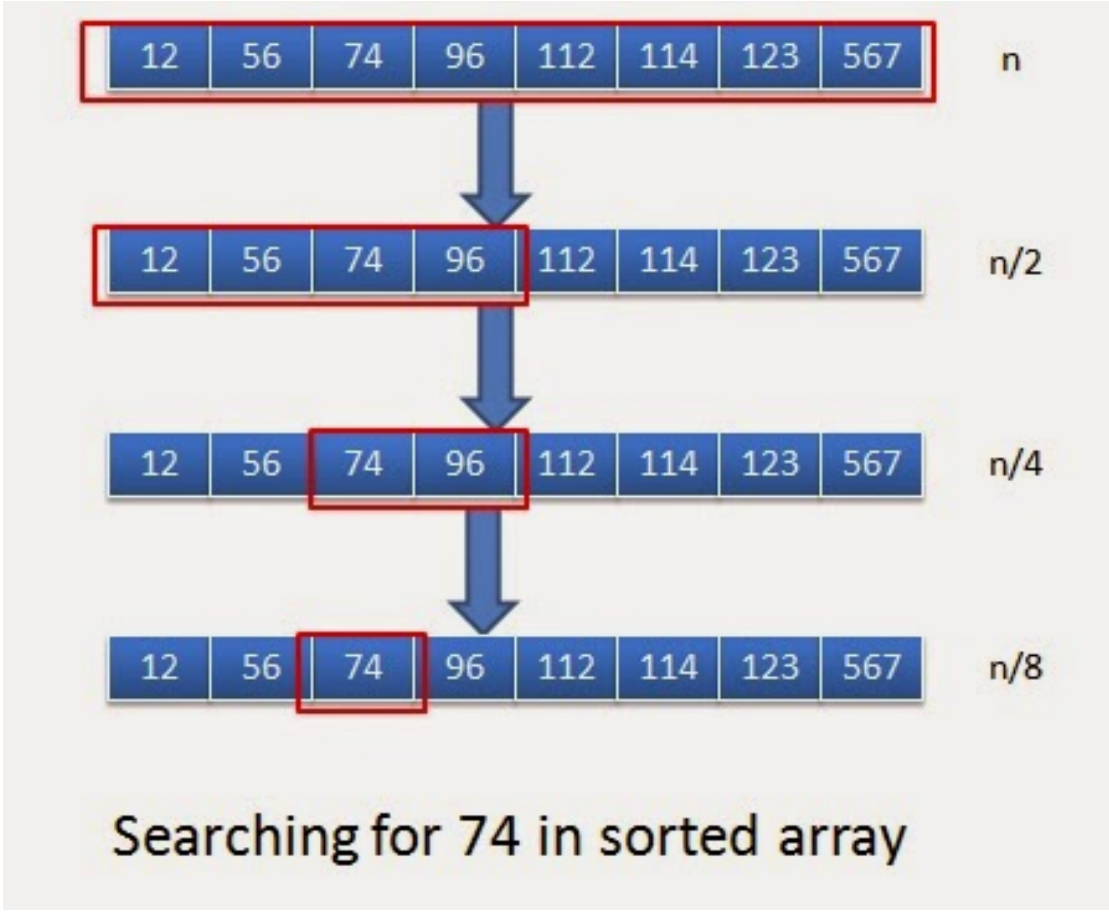
Se usarmos vetores ordenados:

- Podemos buscar em $O(\lg n)$ usando Busca binária
- Mas inserir e remover leva $O(n)$

Veremos árvores binárias de busca

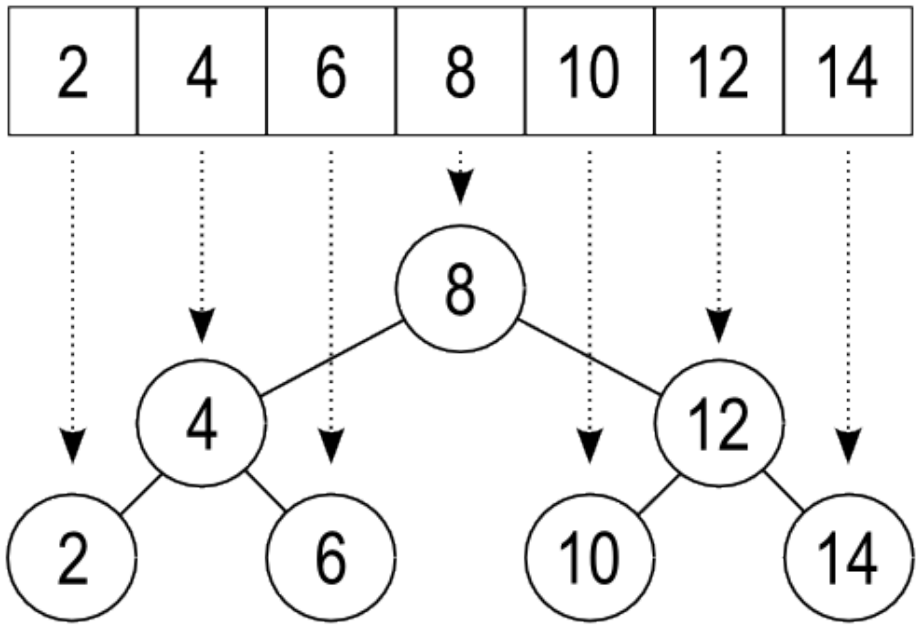
- Inserção, Remoção e Busca levam $O(h)$ onde h é a altura da árvore

Inspiração: Busca binária



Inspiração: Busca binária

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



Árvore Binária de Busca

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Definição

Seja $S = \{s_1, \ldots, s_n\}$ um conjunto ordenável.

Uma Árvore Binária de Busca é uma árvore binária rotulada T com as seguintes características:

Árvore Binária de Busca

Universidade Federal do Ceará
Campus Quixadá

Definição

Seja $S = \{s_1, \ldots, s_n\}$ um conjunto ordenável.

Uma Árvore Binária de Busca é uma árvore binária rotulada T com as seguintes características:

- T possui n nós. Cada nó v de T está rotulado com um elemento $s_j \in S$ e possui como rótulo o valor $r(v) = s_j$.

Árvore Binária de Busca

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Definição

Seja $S = \{s_1, \ldots, s_n\}$ um conjunto ordenável.

Uma Árvore Binária de Busca é uma árvore binária rotulada T com as seguintes características:

- T possui n nós. Cada nó v de T está rotulado com um elemento $s_j \in S$ e possui como rótulo o valor $r(v) = s_j$.
- Seja v um nó de T. Se w pertence à subárvore esquerda de v, então $r(w) < r(v)$.

Árvore Binária de Busca

Definição

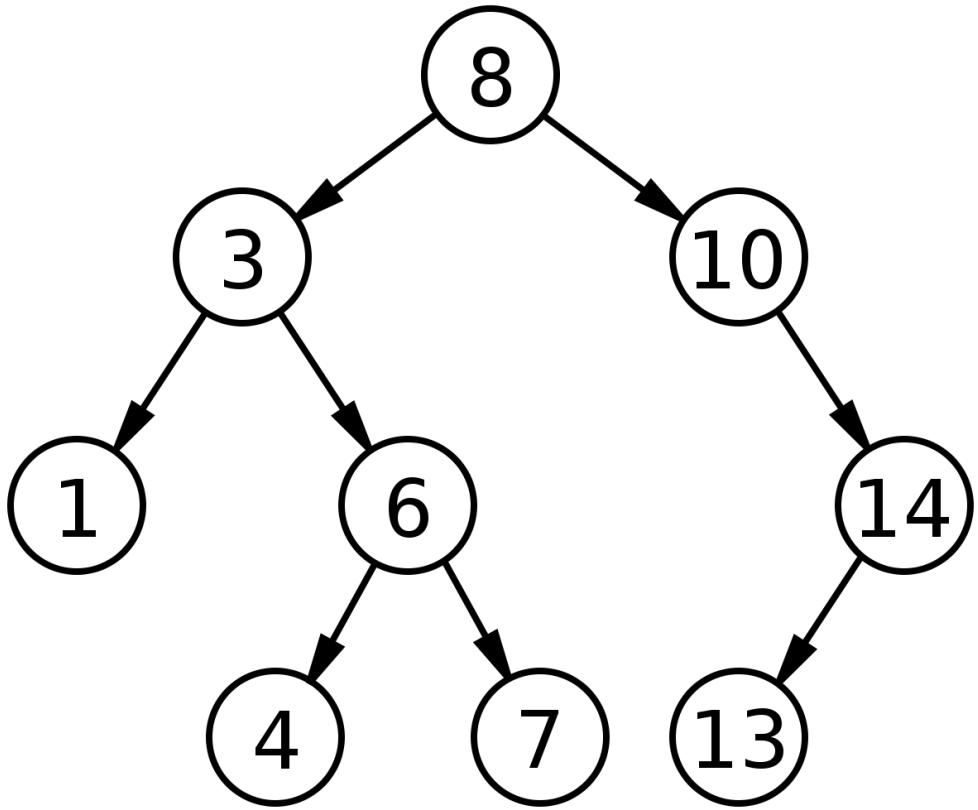
Seja $S = \{s_1, \ldots, s_n\}$ um conjunto ordenável.

Uma Árvore Binária de Busca é uma árvore binária rotulada T com as seguintes características:

- T possui n nós. Cada nó v de T está rotulado com um elemento $s_j \in S$ e possui como rótulo o valor $r(v) = s_j$.
- Seja v um nó de T. Se w pertence à subárvore esquerda de v, então $r(w) < r(v)$.

  Analogamente, se w pertence à subárvore direita de v, então $r(w) > r(v)$.

Exemplo



Árvore Binária de Busca

TAD Árvore Binária de Busca

TAD Árvore Binária de Busca

O tipo abstrato de dado Árvore Binária de Busca, fornece pelo menos as seguintes operações:

- Inserir chave
- Buscar chave
- Remover chave

TAD Árvore Binária de Busca

O tipo abstrato de dado Árvore Binária de Busca, fornece pelo menos as seguintes operações:

- Inserir chave
- Buscar chave
- Remover chave

Funções adicionais:

- buscar o antecessor de uma chave dada
- buscar o sucessor de uma chave dada
- buscar a menor chave
- buscar a maior chave
- retornar uma lista contendo todas as chaves em ordem crescente
- dentre outras ...

Implementação da unidade básica

- O nó da árvore será um **tipo de dado composto** com três campos:

Implementação da unidade básica

- O nó da árvore será um **tipo de dado composto** com três campos:
  - um inteiro (rótulo do nó)

Implementação da unidade básica

- O nó da árvore será um **tipo de dado composto** com três campos:
  - um inteiro (rótulo do nó)
  - um ponteiro para o filho esquerdo do nó

Implementação da unidade básica

- O nó da árvore será um **tipo de dado composto** com três campos:
  - um inteiro (rótulo do nó)
  - um ponteiro para o filho esquerdo do nó
  - um ponteiro para o filho direito do nó.

Implementação da unidade básica

- O nó da árvore será um **tipo de dado composto** com três campos:
  - um inteiro (rótulo do nó)
  - um ponteiro para o filho esquerdo do nó
  - um ponteiro para o filho direito do nó.

Em C++

```
1 struct Node {
2     int key;
3     Node *left;
4     Node *right;
5 };
```

Implementação da unidade básica

- O nó da árvore será um **tipo de dado composto** com três campos:
  - um inteiro (rótulo do nó)
  - um ponteiro para o filho esquerdo do nó
  - um ponteiro para o filho direito do nó.

Em C++

```
1 struct Node {
2     int key;
3     Node *left;
4     Node *right;
5 };
```

- Também é possível adicionar um ponteiro para o nó pai. Neste caso, a implementação mudará para ter que manter este novo campo.

Implementação da unidade básica

- O nó da árvore será um **tipo de dado composto** com três campos:
  - um inteiro (rótulo do nó)
  - um ponteiro para o filho esquerdo do nó
  - um ponteiro para o filho direito do nó.

Em C++

```
1 struct Node {
2     int key;
3     Node *left;
4     Node *right;
5 };
```

- Também é possível adicionar um ponteiro para o nó pai. Neste caso, a implementação mudará para ter que manter este novo campo.

- Usaremos **recursão** na implementação das operações da árvore binária.

Alocação e Liberação de Memória

- Estruturas de dados consomem memória. Em C++, geralmente essa memória é alocada:

Alocação e Liberação de Memória

- Estruturas de dados consomem memória. Em C++, geralmente essa memória é alocada:
  1. de forma direta, usando o operador `new`

Alocação e Liberação de Memória

- Estruturas de dados consomem memória. Em C++, geralmente essa memória é alocada:
  1. de forma direta, usando o operador `new`
  2. de forma indireta, usando o contêiner `std::vector`

Alocação e Liberação de Memória

- Estruturas de dados consomem memória. Em C++, geralmente essa memória é alocada:
  1. de forma direta, usando o operador `new`
  2. de forma indireta, usando o contêiner `std::vector`
  3. usando smart pointers

Alocação e Liberação de Memória

- Estruturas de dados consomem memória. Em C++, geralmente essa memória é alocada:
  1. de forma direta, usando o operador `new`
  2. de forma indireta, usando o contêiner `std::vector`
  3. usando smart pointers

- ⚠ Atenção: Se você alocar memória usando `new`, **você deve garantir que desaloca** a memória, usando o operador `delete`, quando ela não for mais necessária no programa.

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Inserção

Inserindo um valor

Precisamos determinar onde inserir o valor:

Inserindo um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor

Inserindo um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e colocamos ele na posição onde deveria estar

Inserindo um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e colocamos ele na posição onde deveria estar

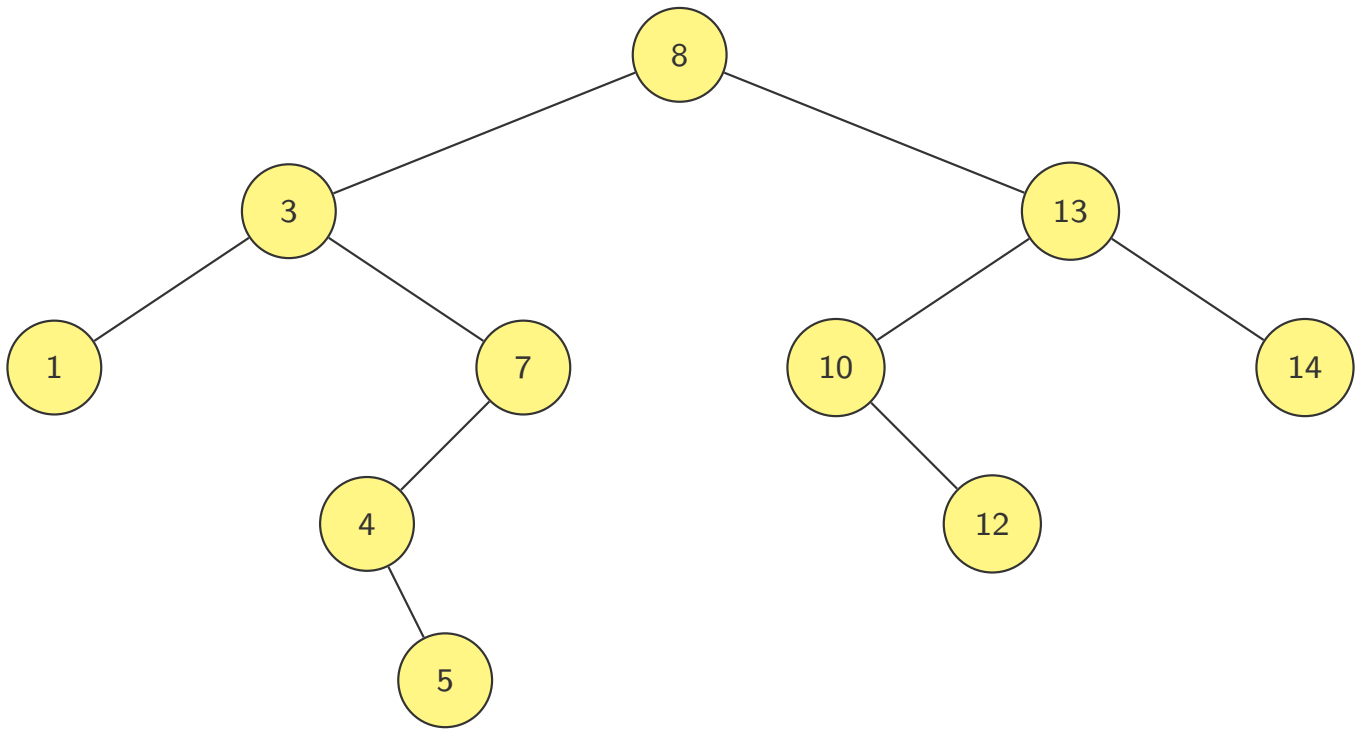
Ex: Inserindo 11

Inserindo um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e colocamos ele na posição onde deveria estar

Ex: Inserindo 11

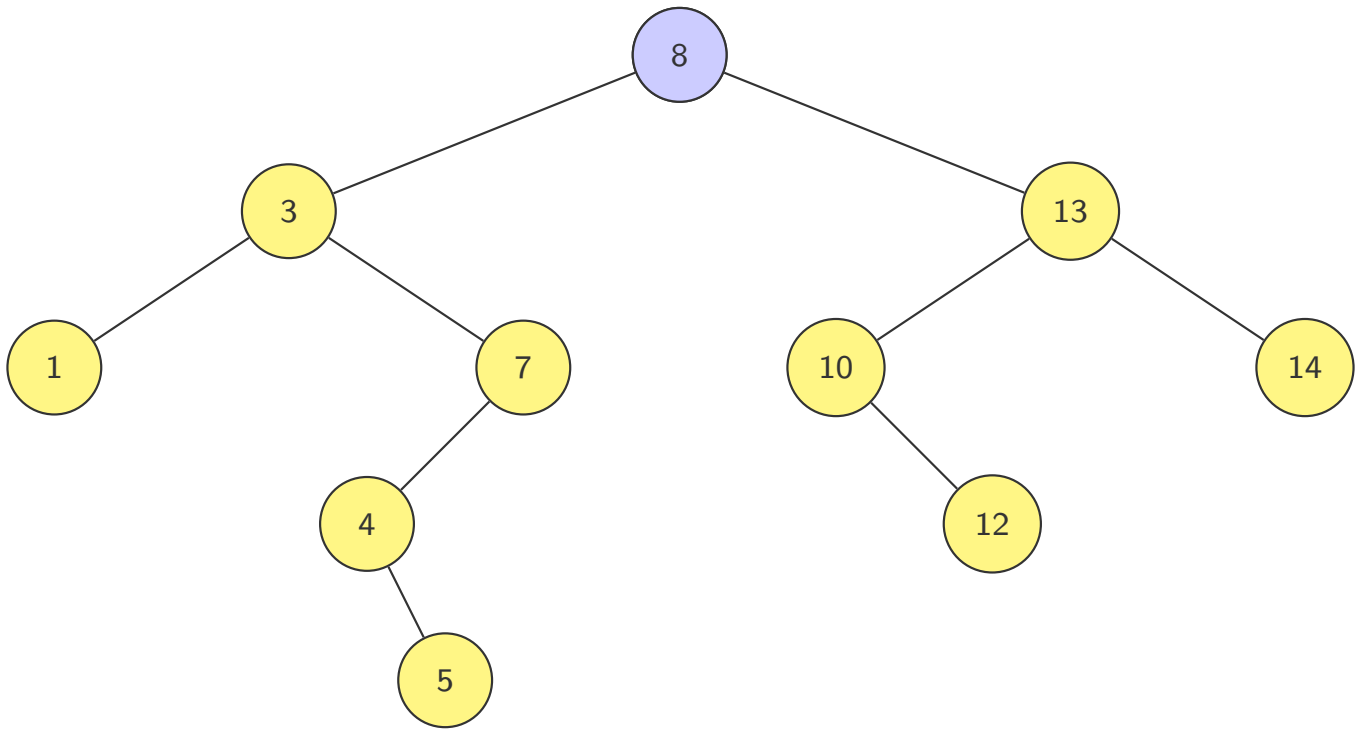


Inserindo um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e colocamos ele na posição onde deveria estar

Ex: Inserindo 11

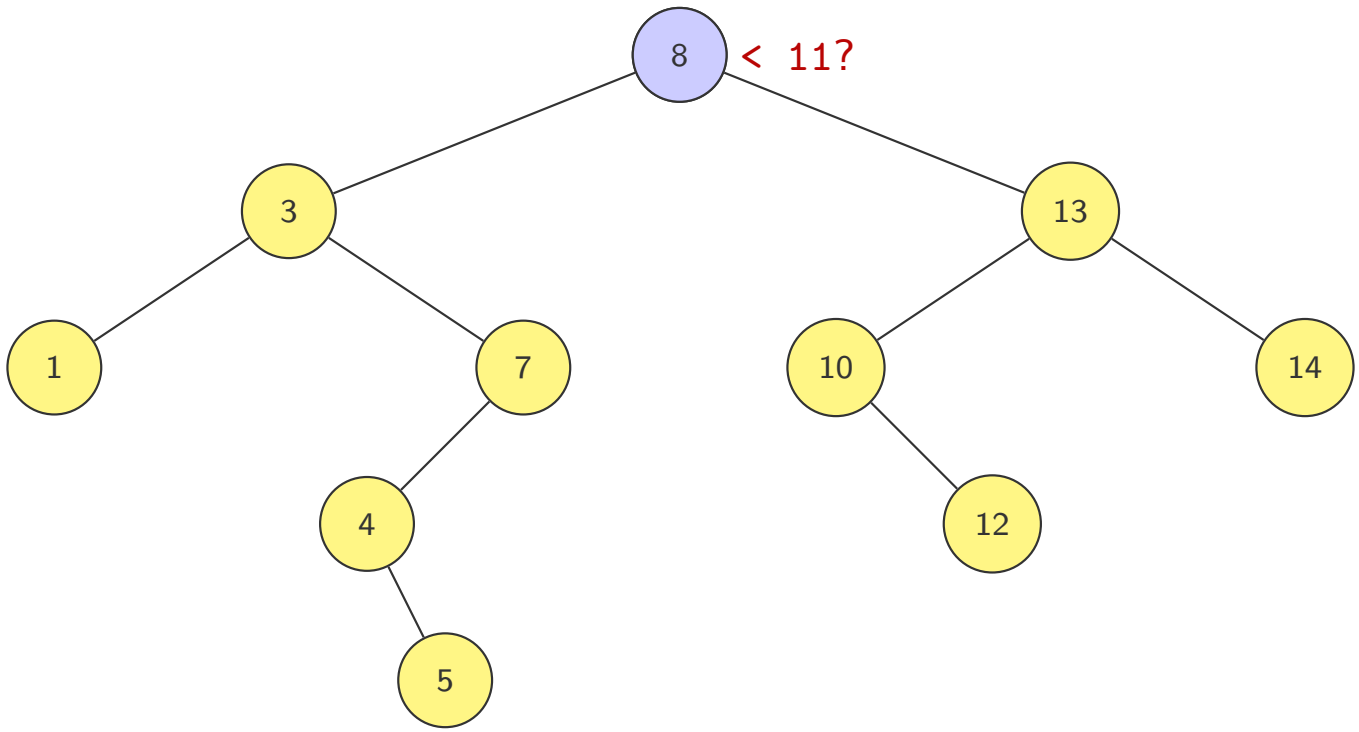


Inserindo um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e colocamos ele na posição onde deveria estar

Ex: Inserindo 11

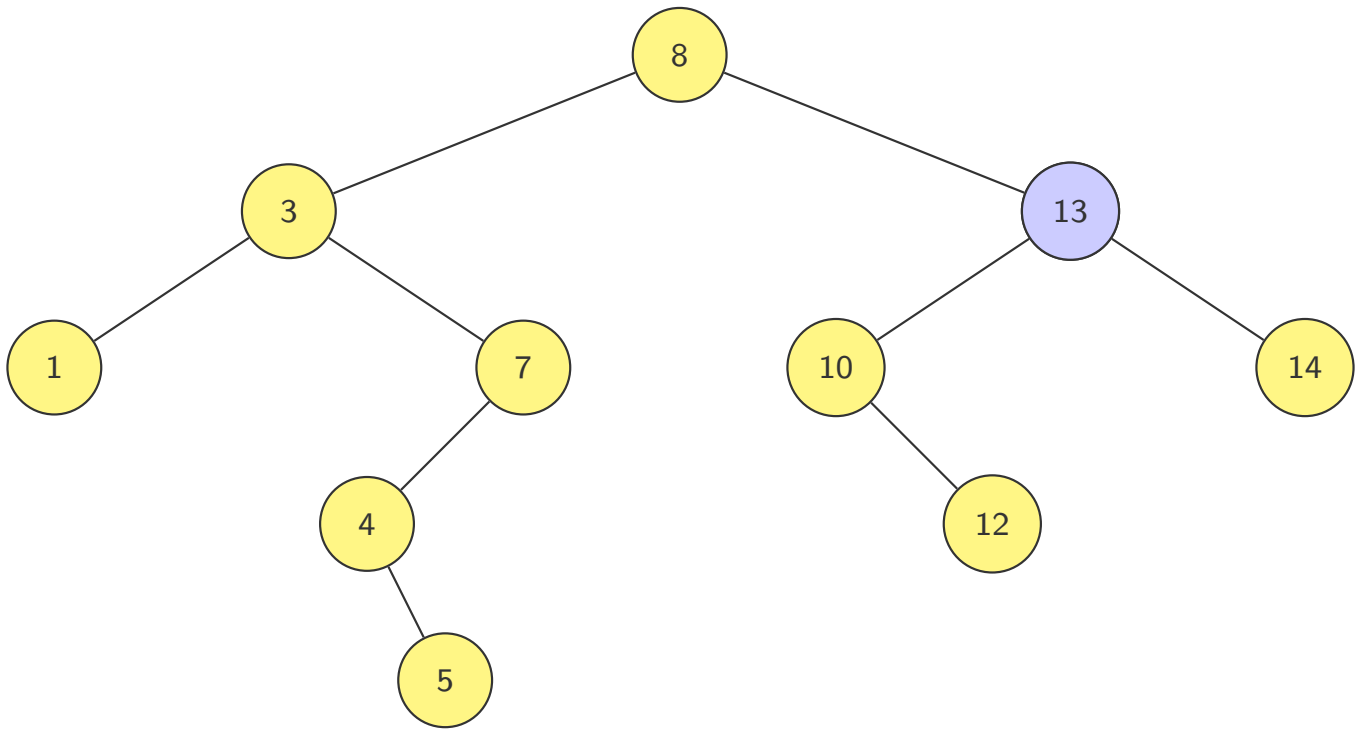


Inserindo um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e colocamos ele na posição onde deveria estar

Ex: Inserindo 11

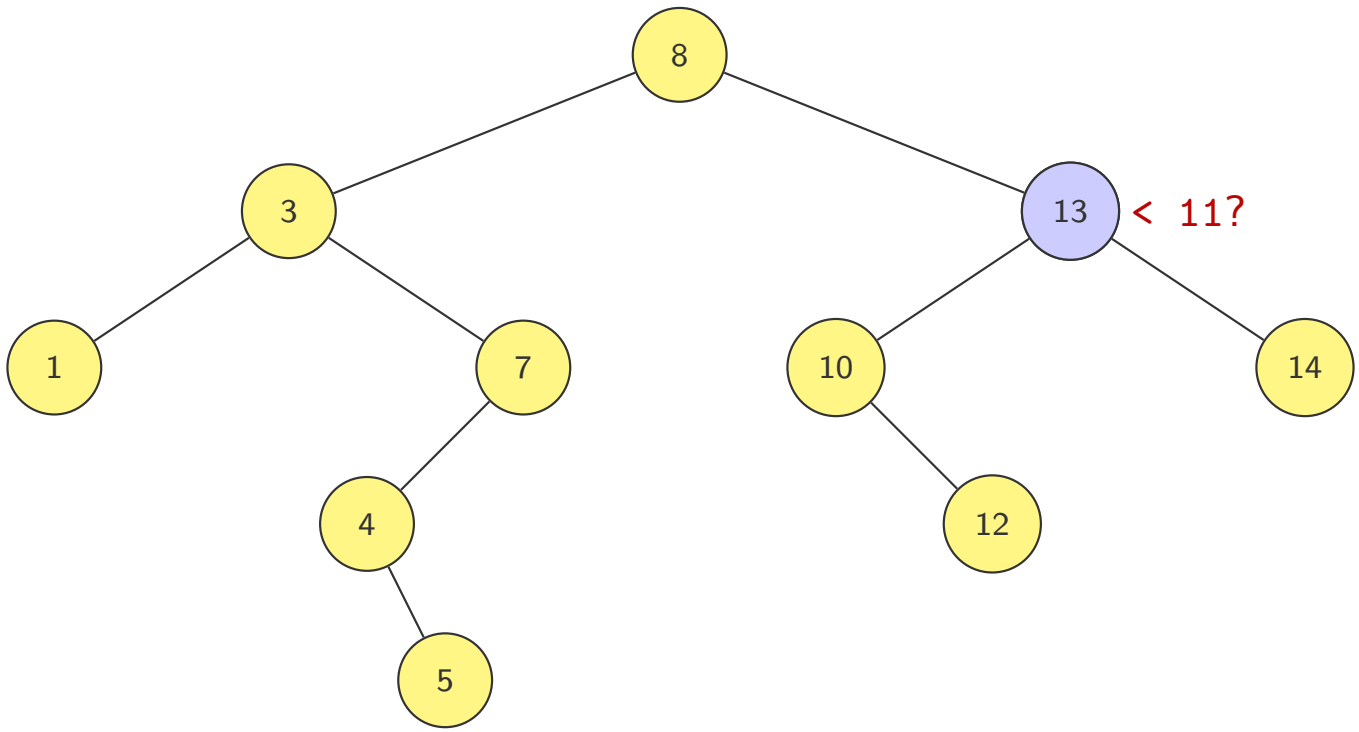


Inserindo um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e colocamos ele na posição onde deveria estar

Ex: Inserindo 11

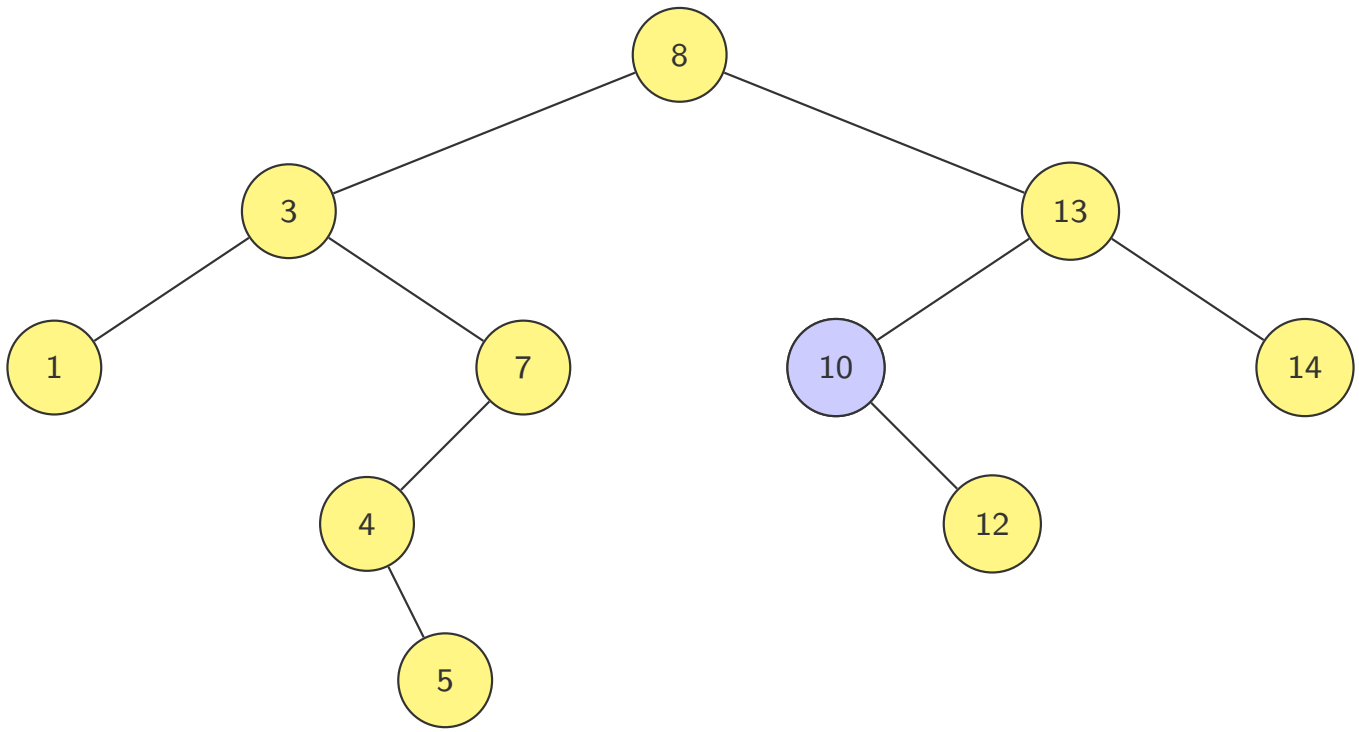


Inserindo um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e colocamos ele na posição onde deveria estar

Ex: Inserindo 11

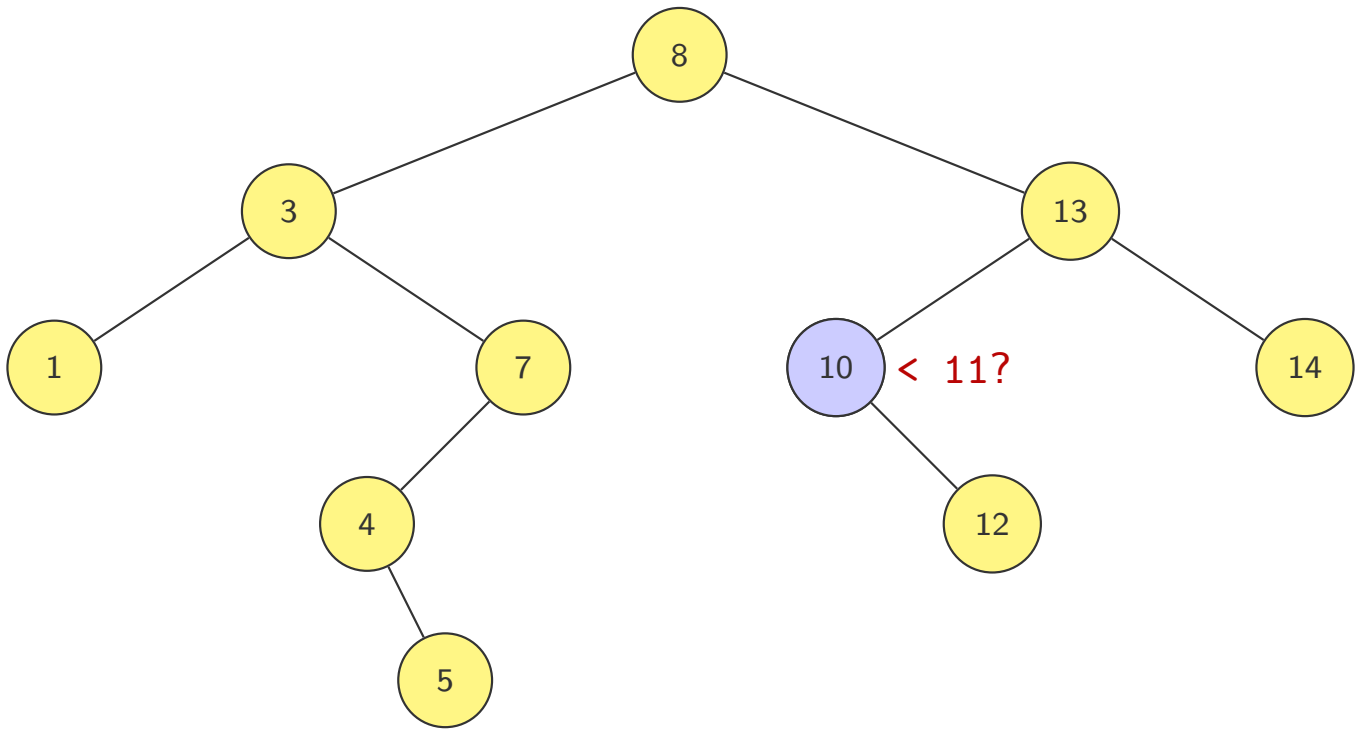


Inserindo um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e colocamos ele na posição onde deveria estar

Ex: Inserindo 11

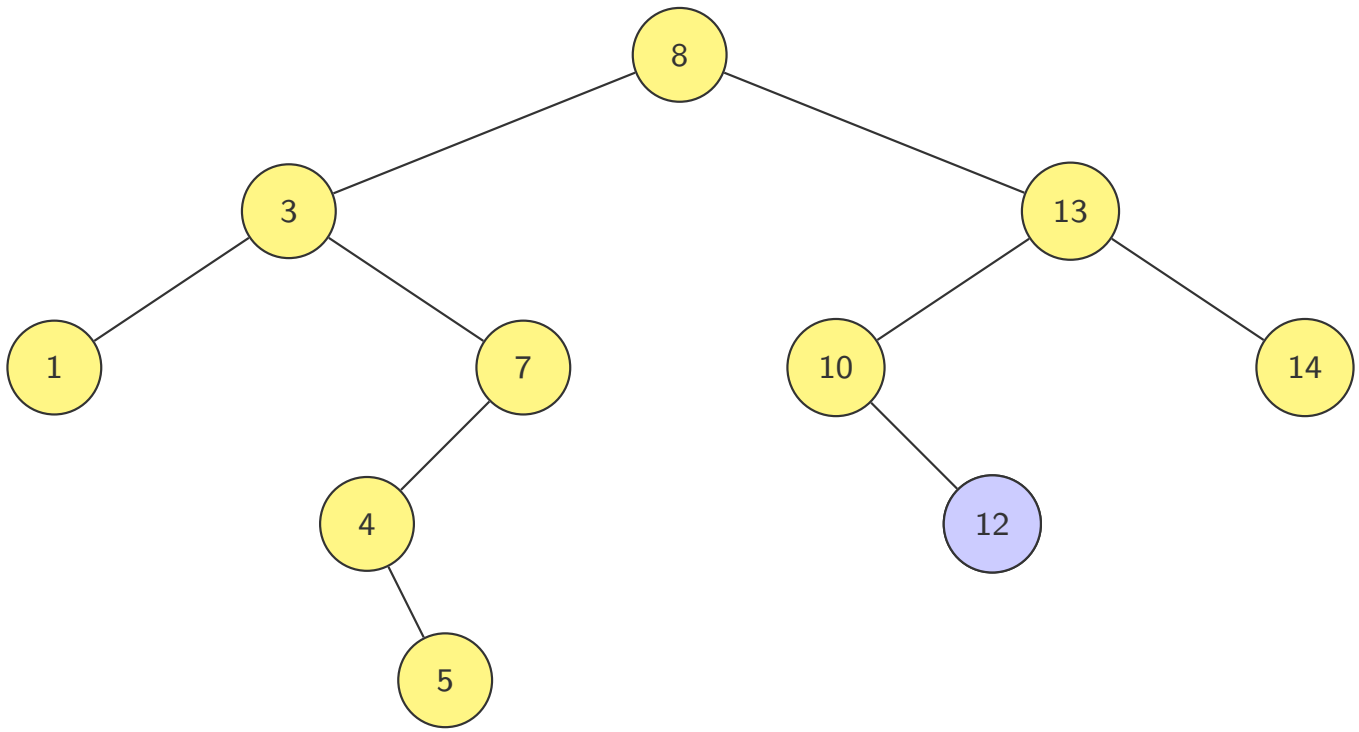


Inserindo um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e colocamos ele na posição onde deveria estar

Ex: Inserindo 11

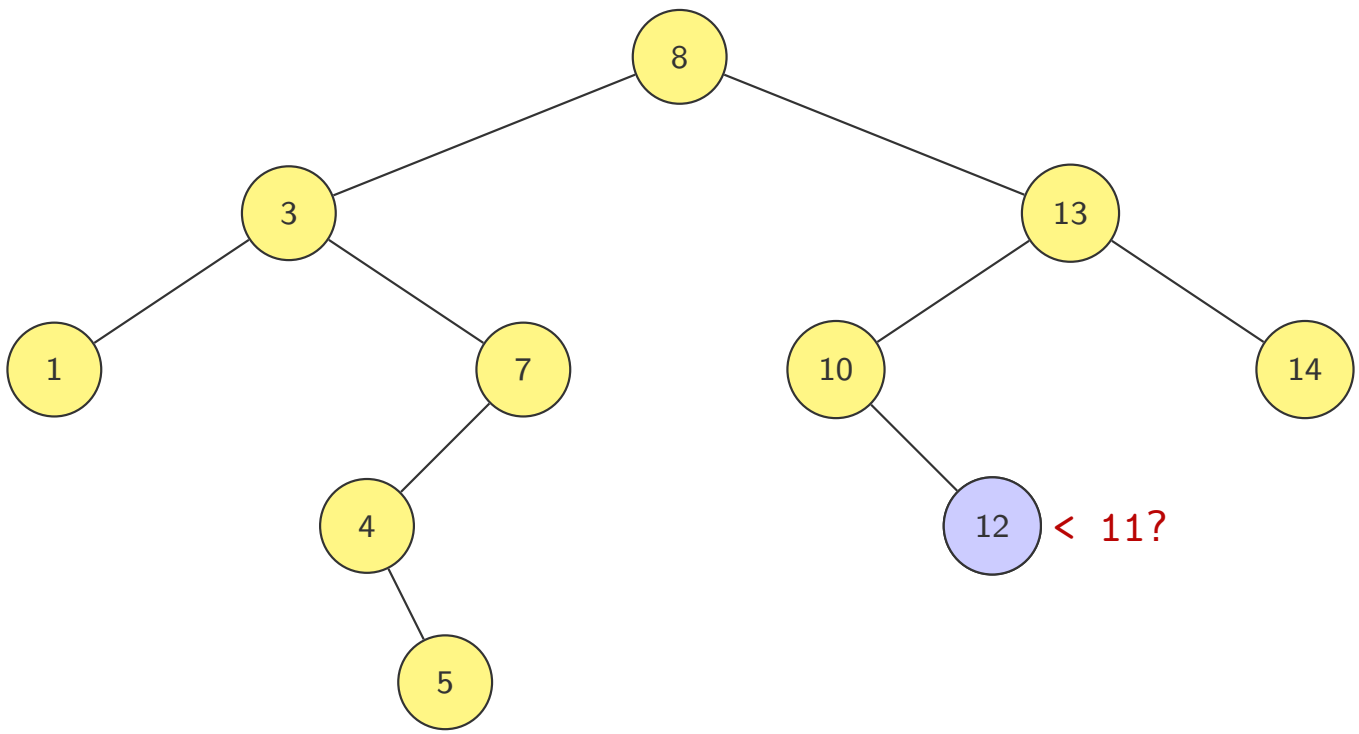


Inserindo um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e colocamos ele na posição onde deveria estar

Ex: Inserindo 11

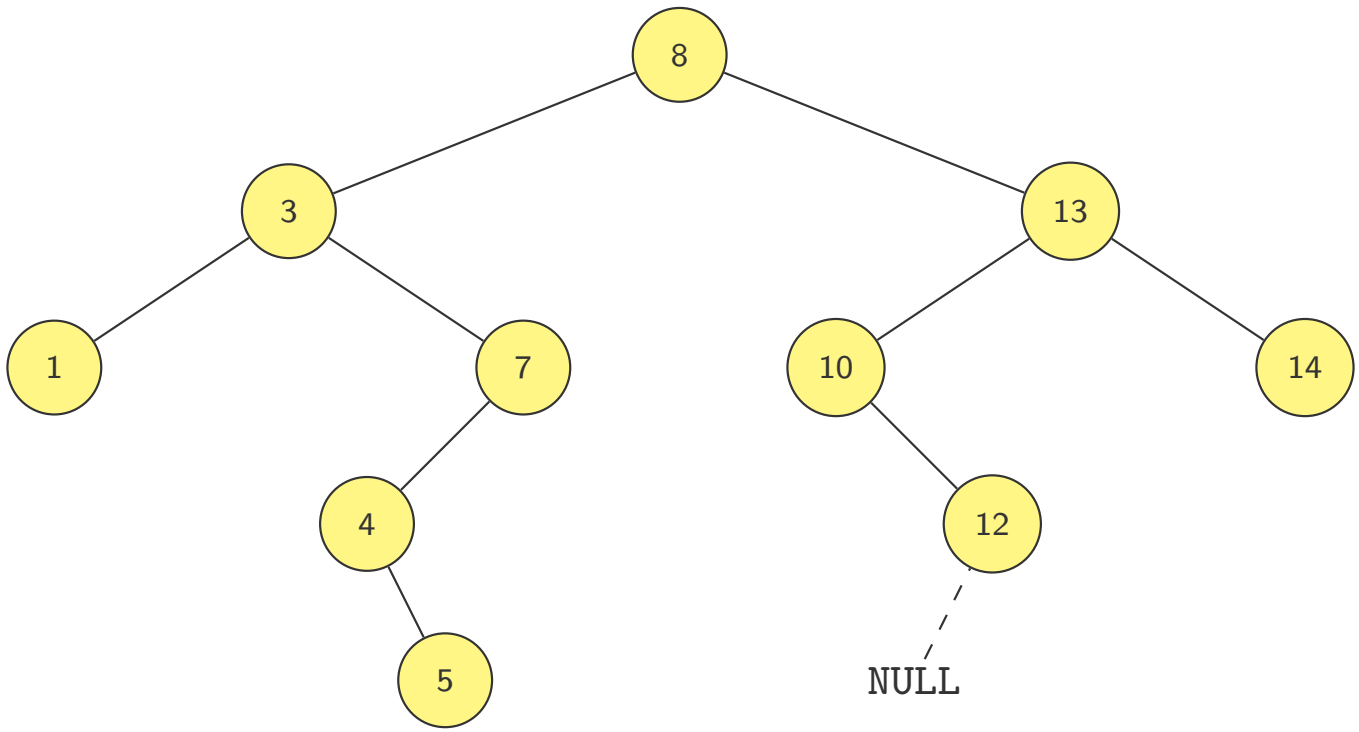


Inserindo um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e colocamos ele na posição onde deveria estar

Ex: Inserindo 11

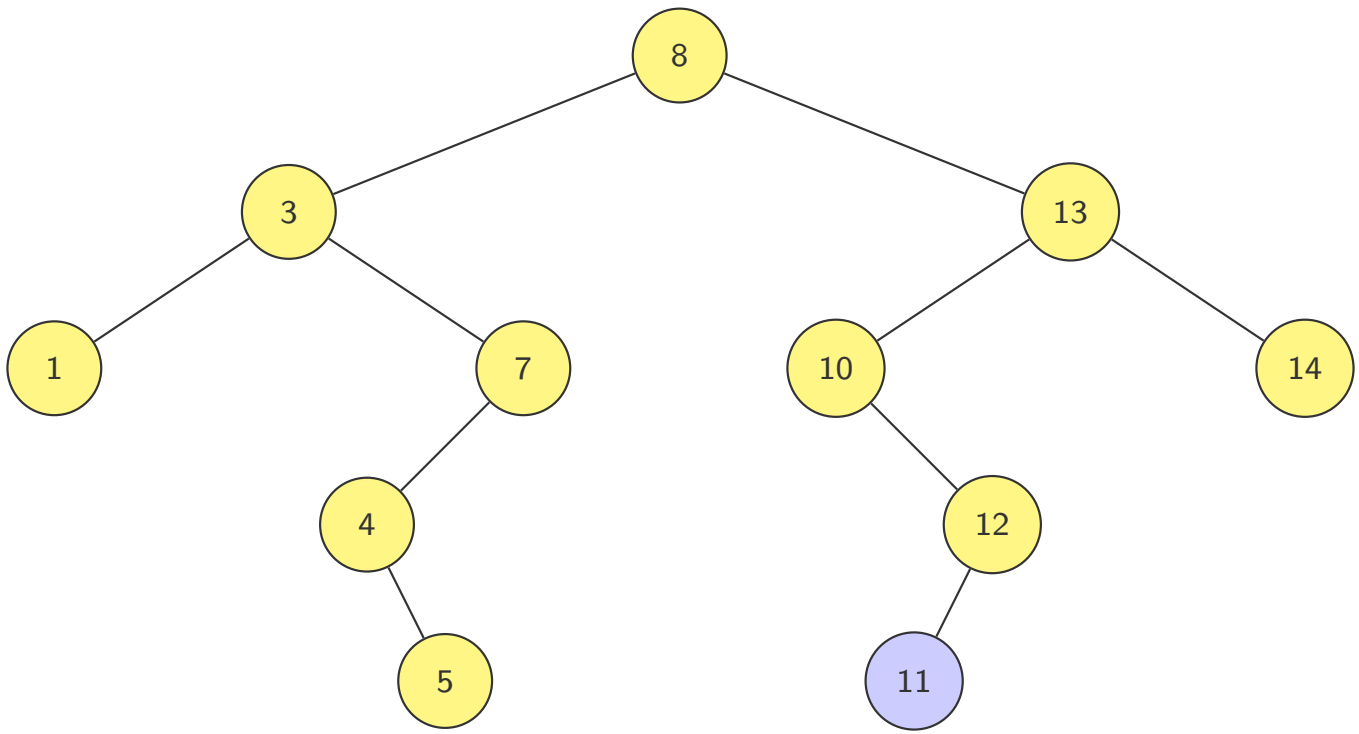


Inserindo um valor

Precisamos determinar onde inserir o valor:

- fazemos uma busca pelo valor
- e colocamos ele na posição onde deveria estar

Ex: Inserindo 11



Pseudocódigo da Inserção

Pseudocódigo da Inserção

Algorithm insere(Node *node, int k)

Require: ponteiro para raiz e chave a ser inserida
Ensure: ponteiro para raiz da árvore resultante

```
1: if node == nulo then
2:     cria novo nó e o retorna
3: end if
4: if k > node→key then
5:     node→right = insere(node→right, k)
6: else if k < node→key then
7:     node→left = busca(node→left, k)
8: end if
9: return node
```

Pseudocódigo da Inserção

Algorithm insere(Node *node, int k)

Require: ponteiro para raiz e chave a ser inserida
Ensure: ponteiro para raiz da árvore resultante

```
1: if node == nulo then
2:     cria novo nó e o retorna
3: end if
4: if k > node→key then
5:     node→right = insere(node→right, k)
6: else if k < node→key then
7:     node→left = busca(node→left, k)
8: end if
9: return node
```

Qual a complexidade deste algoritmo?

Busca

Busca por um valor

A ideia é semelhante a da busca binária:

Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore

Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz

Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda

Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz

Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

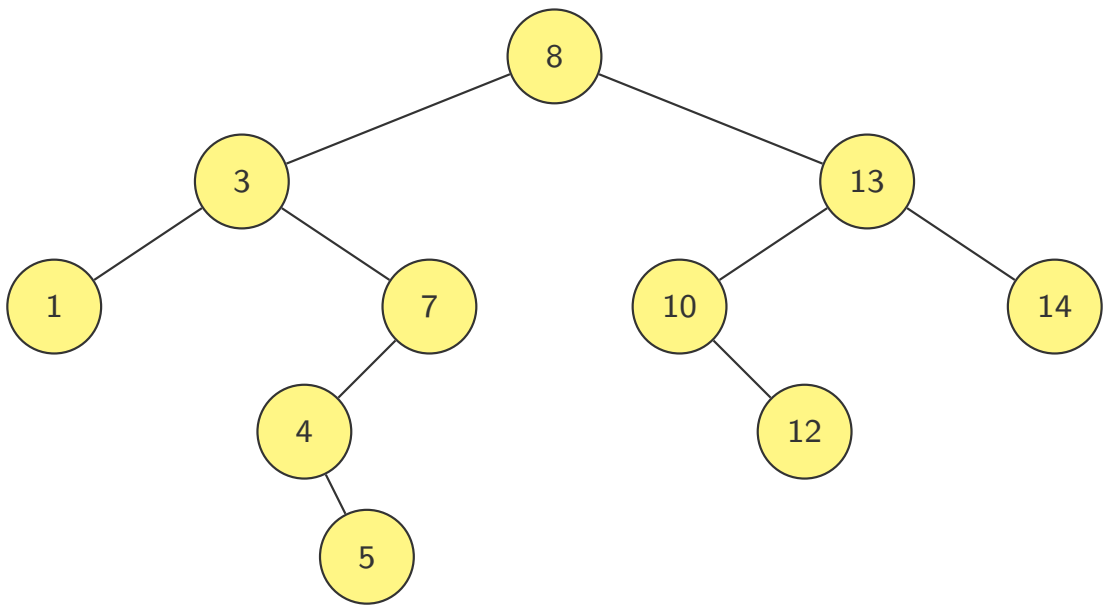
Ex: Buscando por 4

Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 4

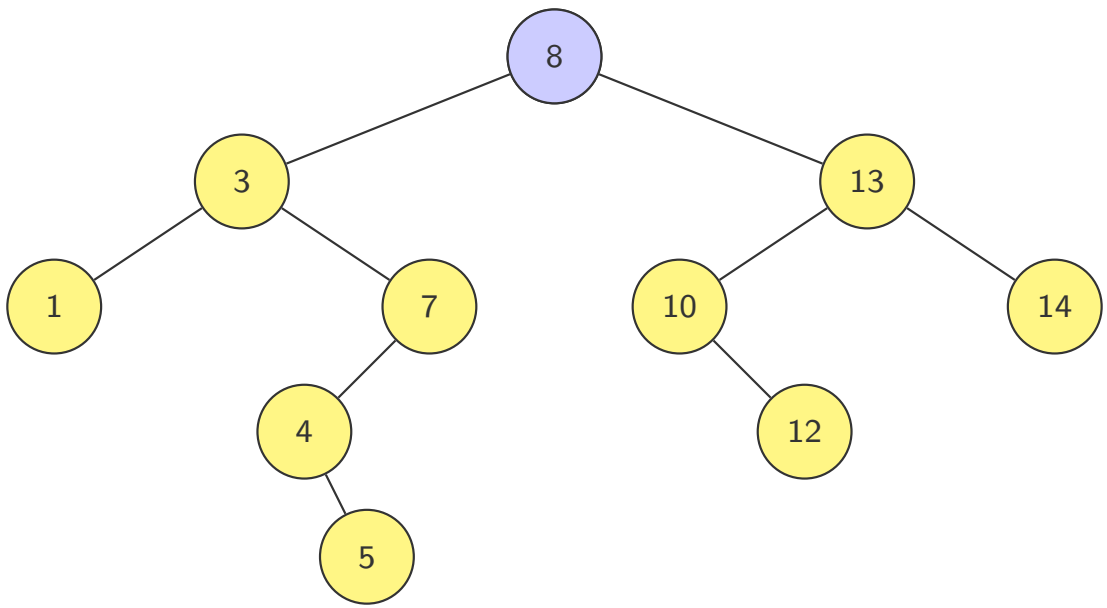


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 4

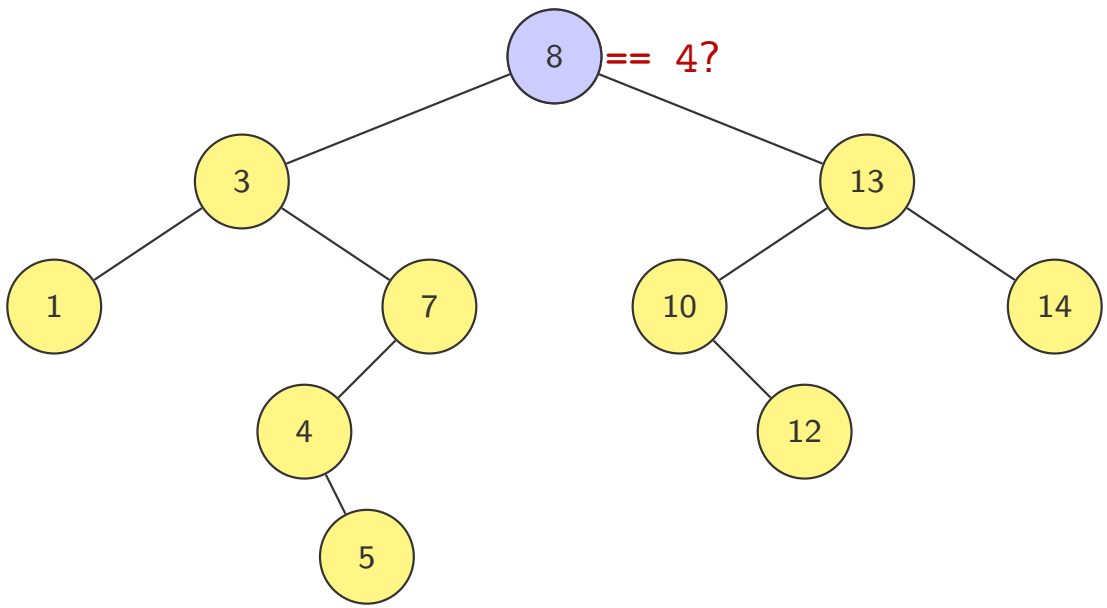


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 4

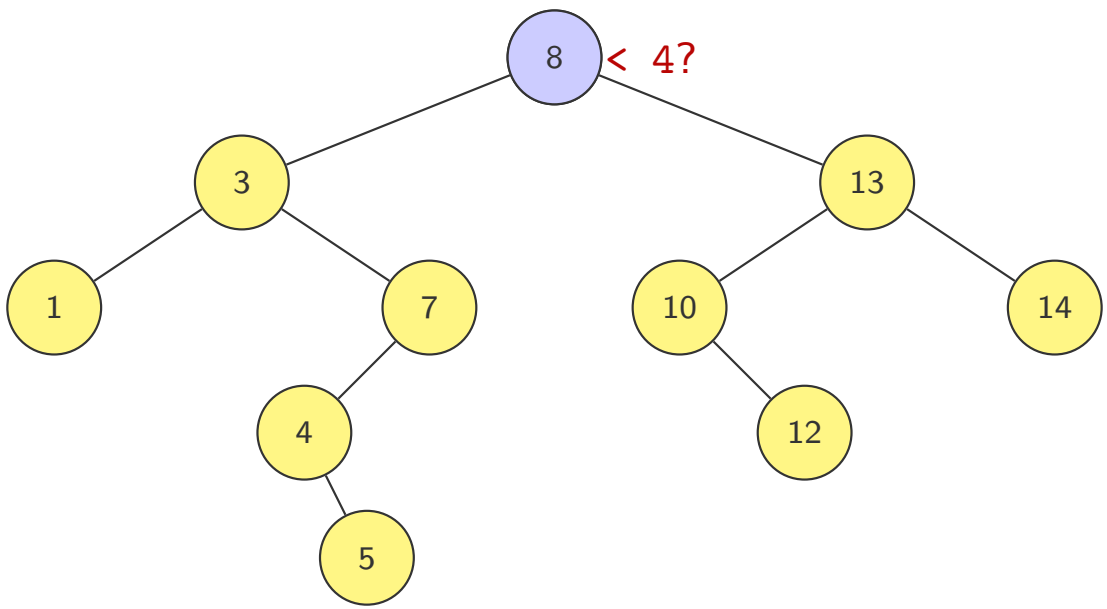


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 4

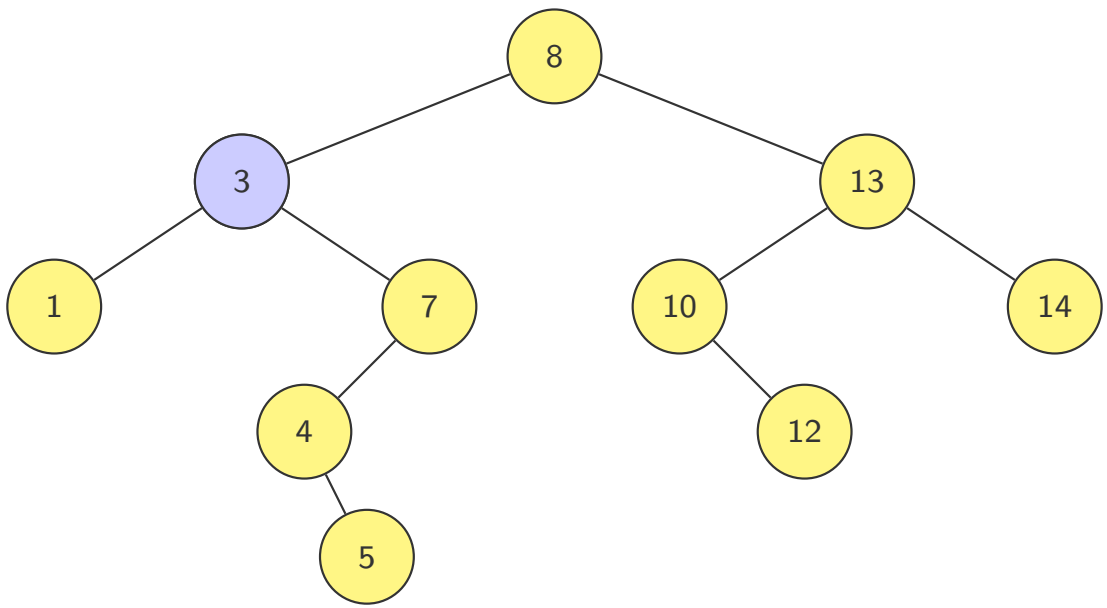


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 4

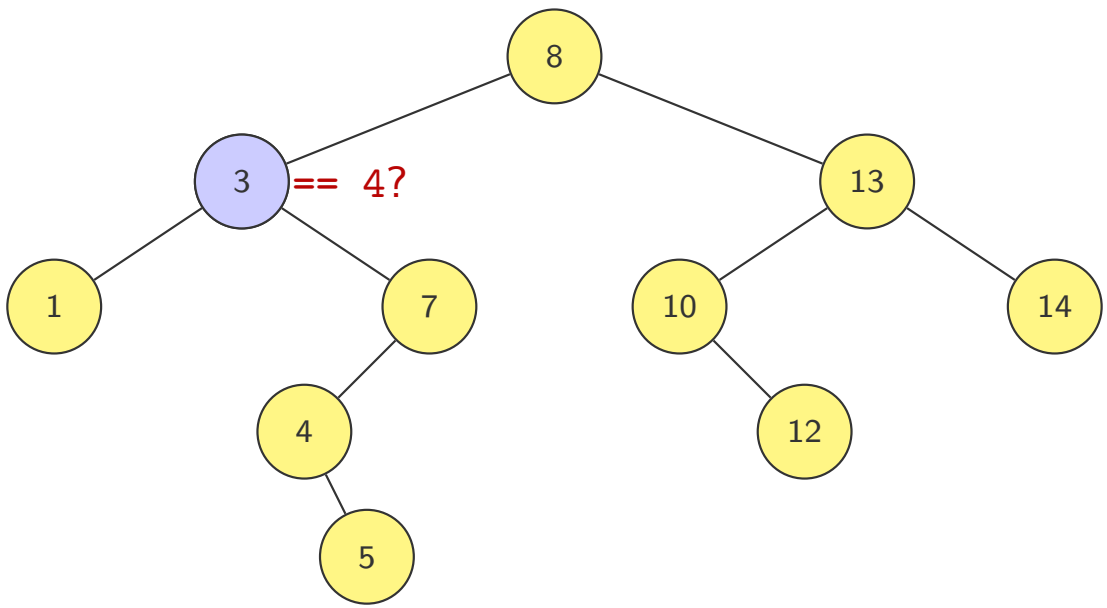


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 4

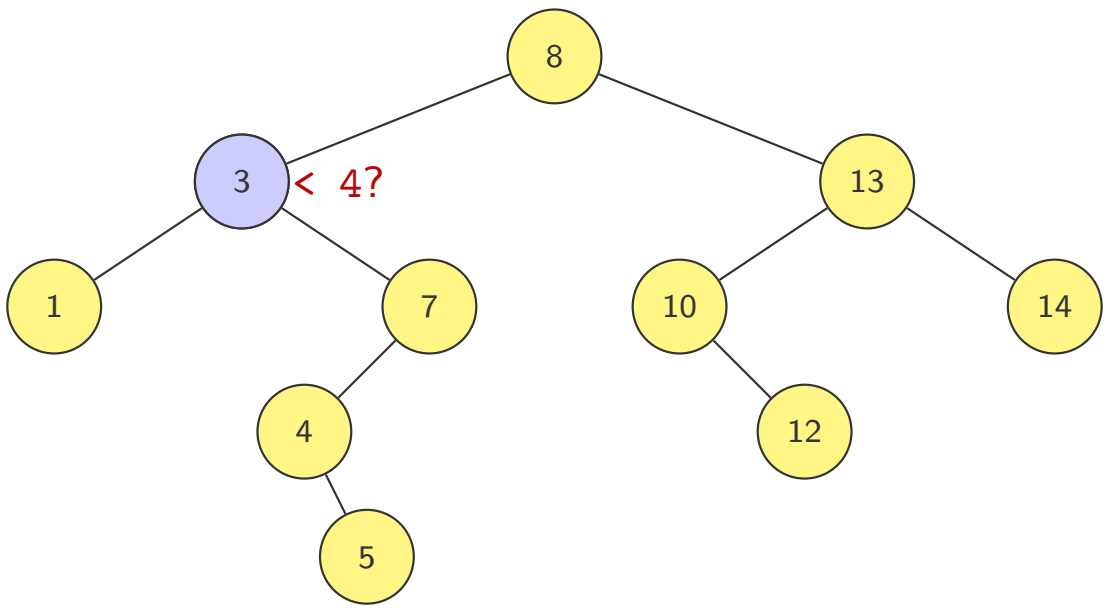


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 4

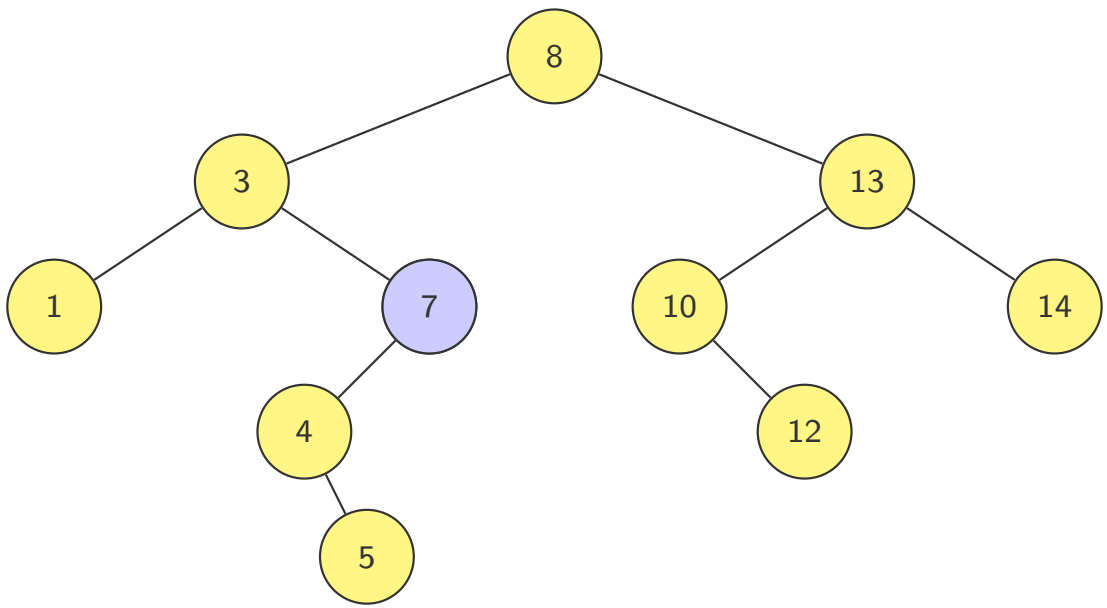


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 4

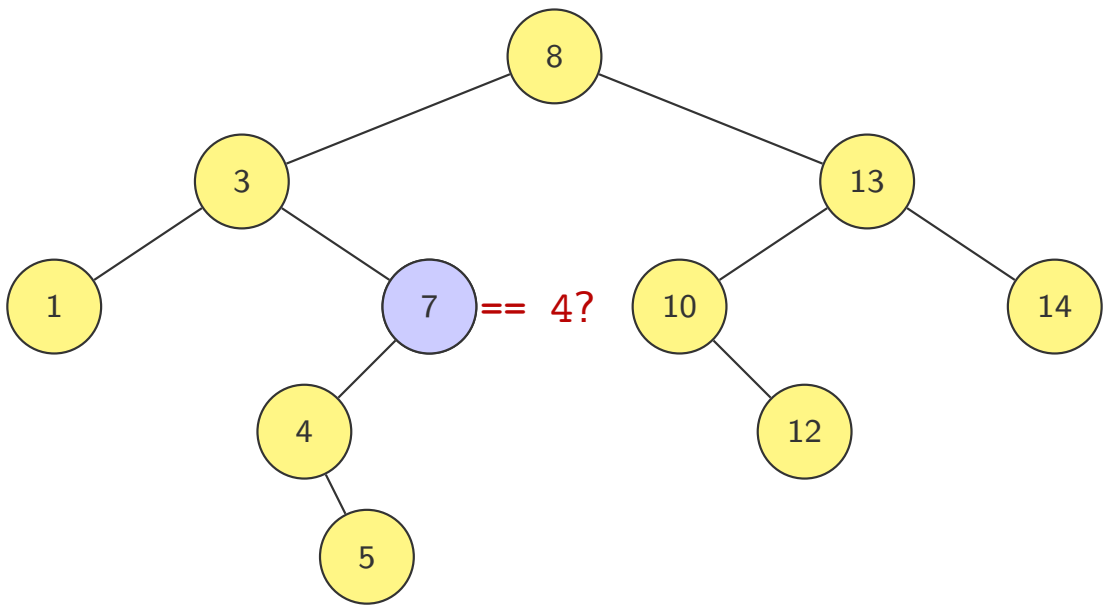


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 4

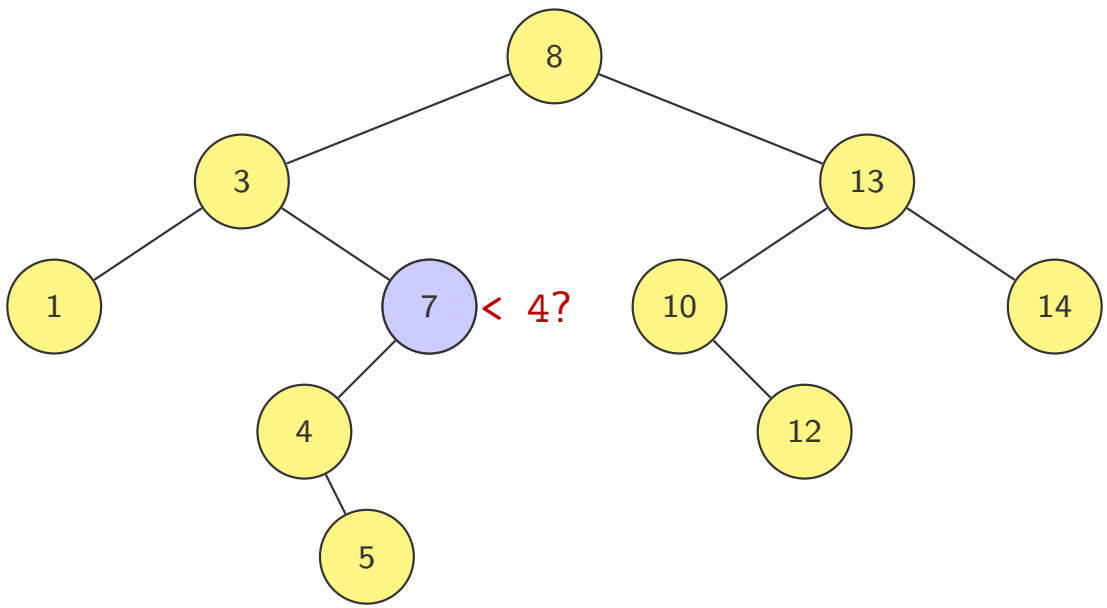


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 4

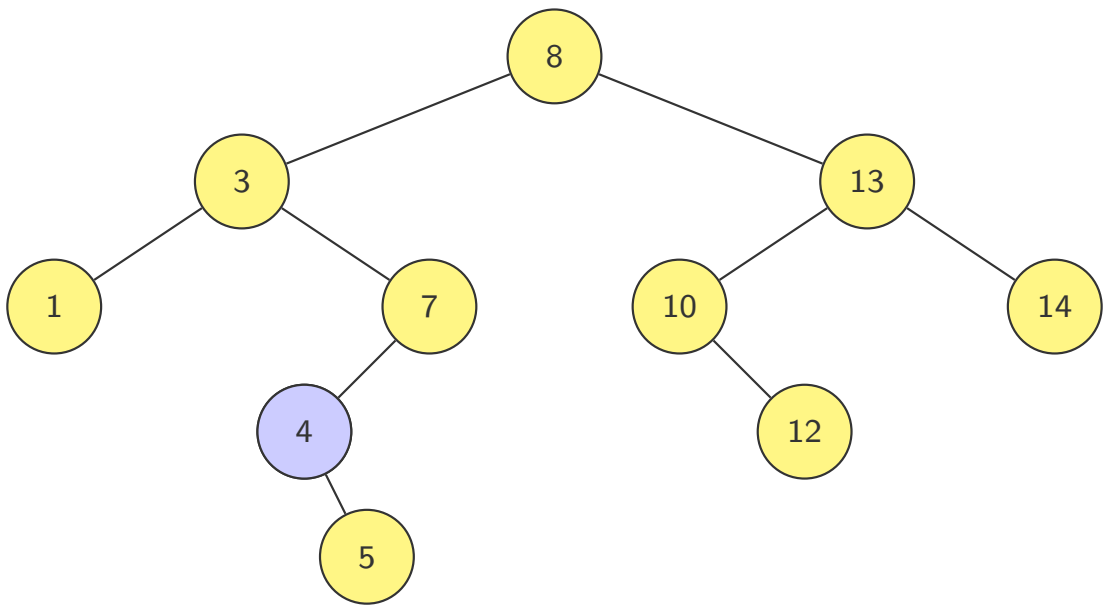


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 4

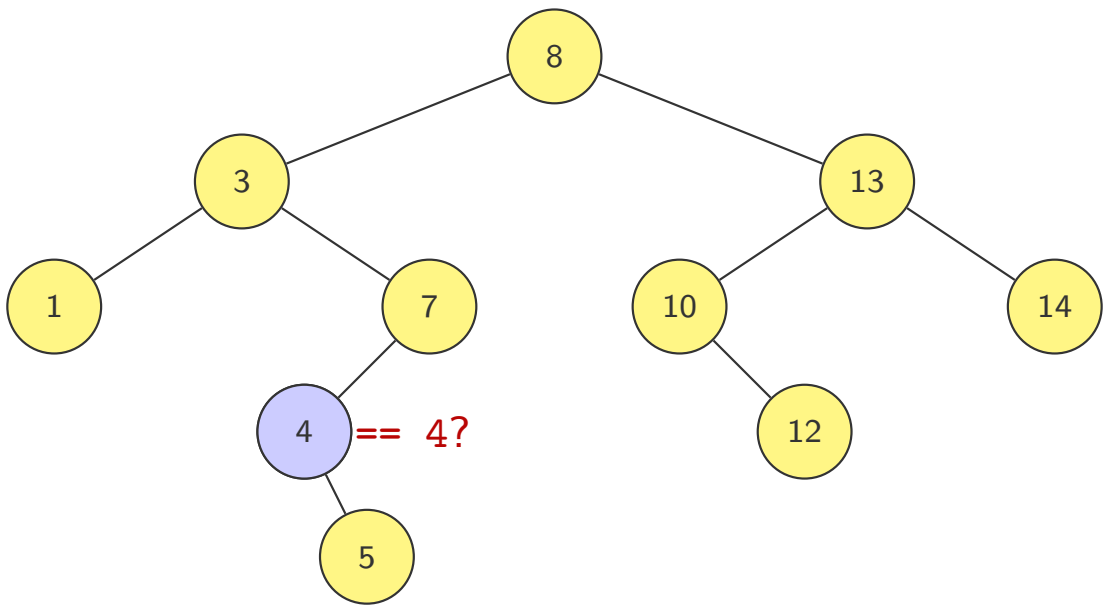


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 4

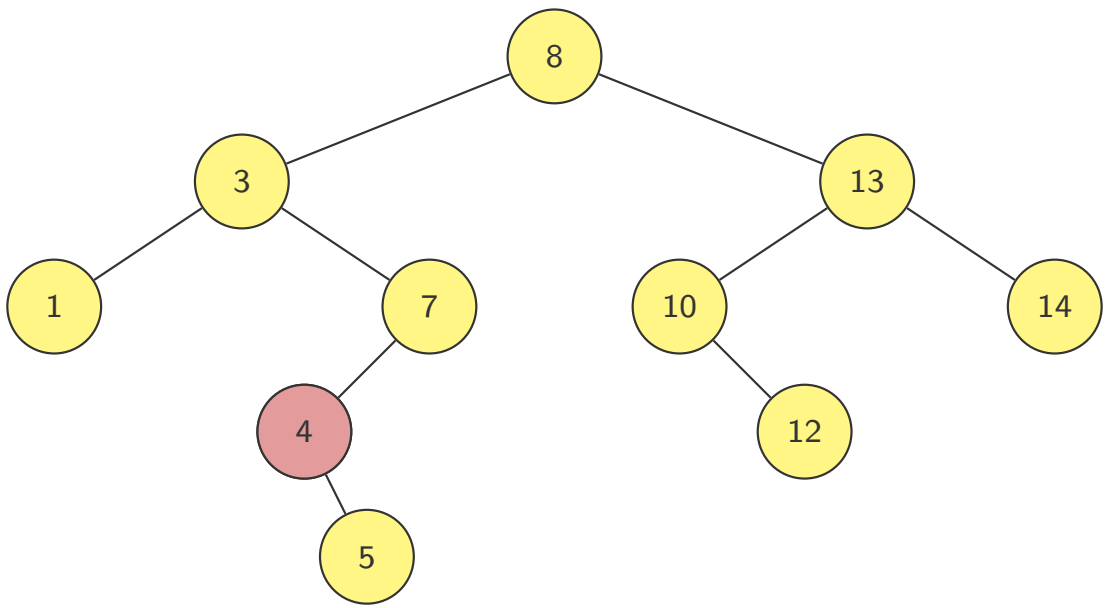


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 4

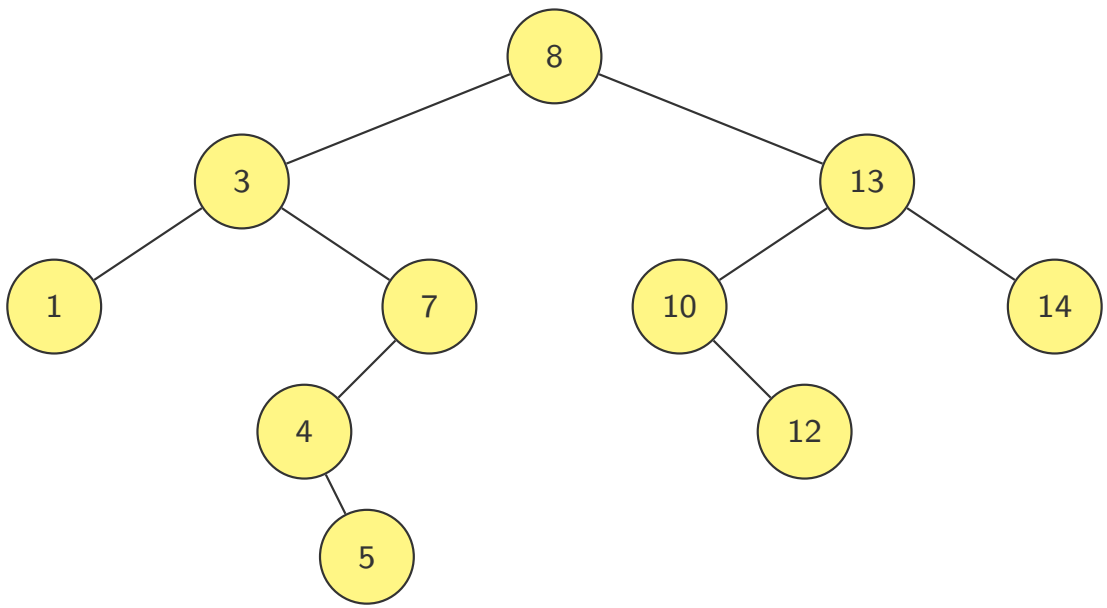


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 11

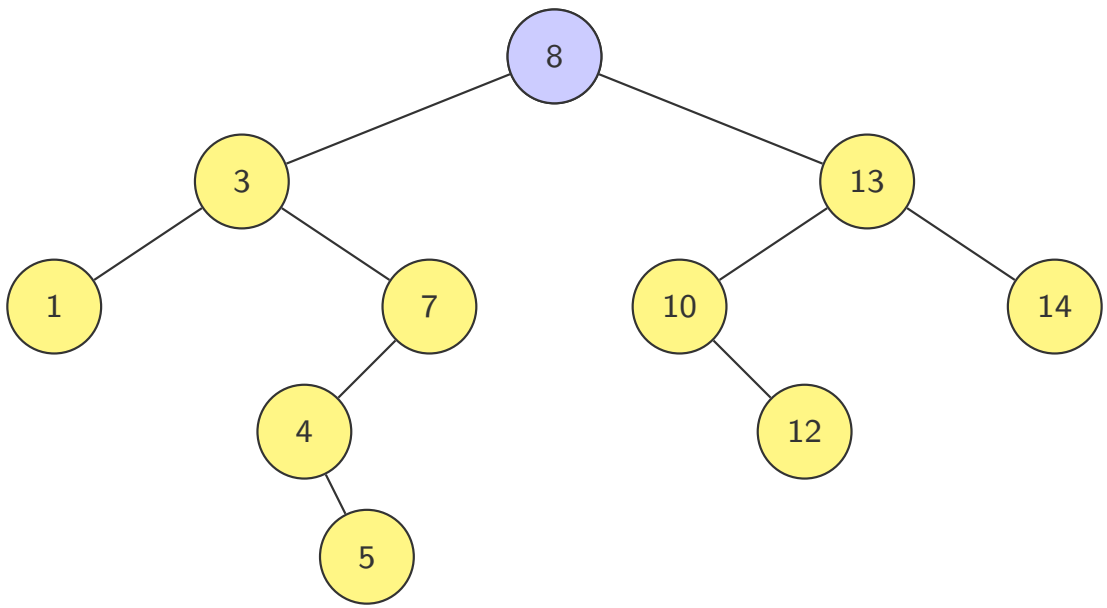


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 11

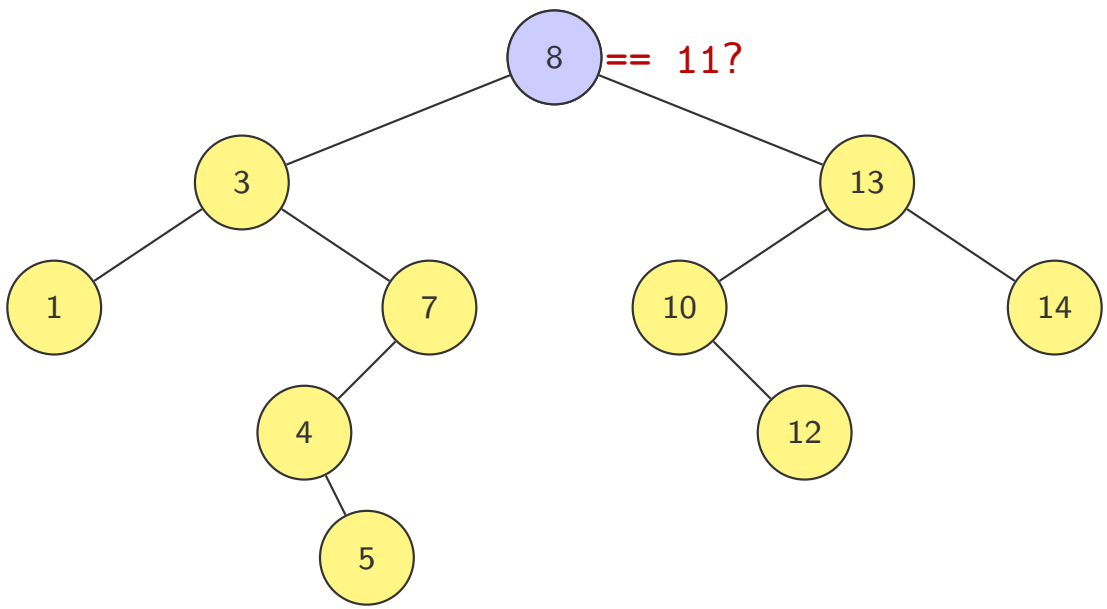


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 11

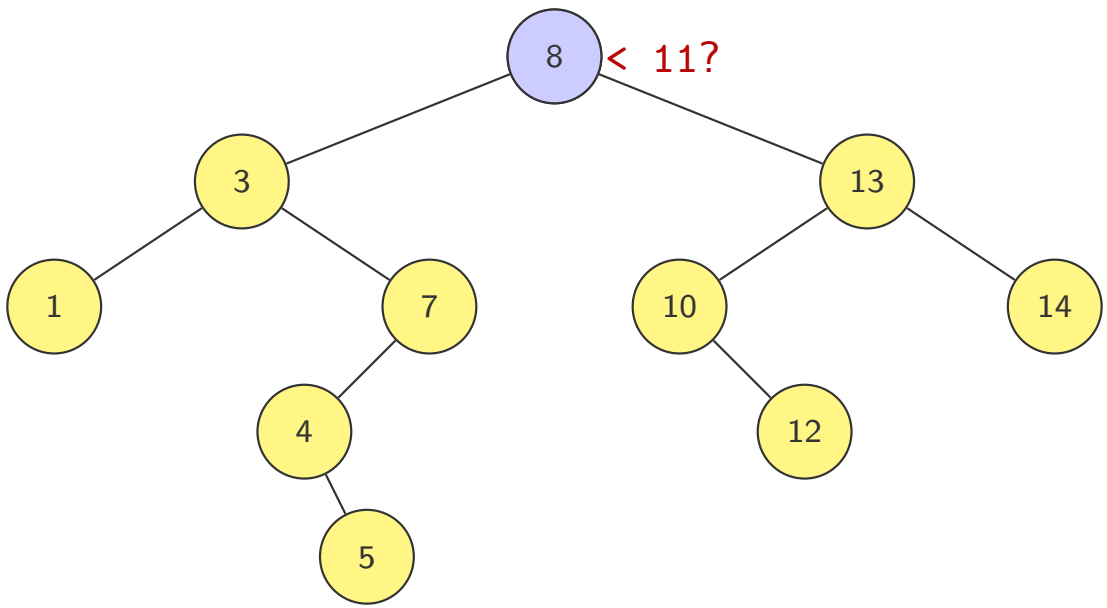


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 11

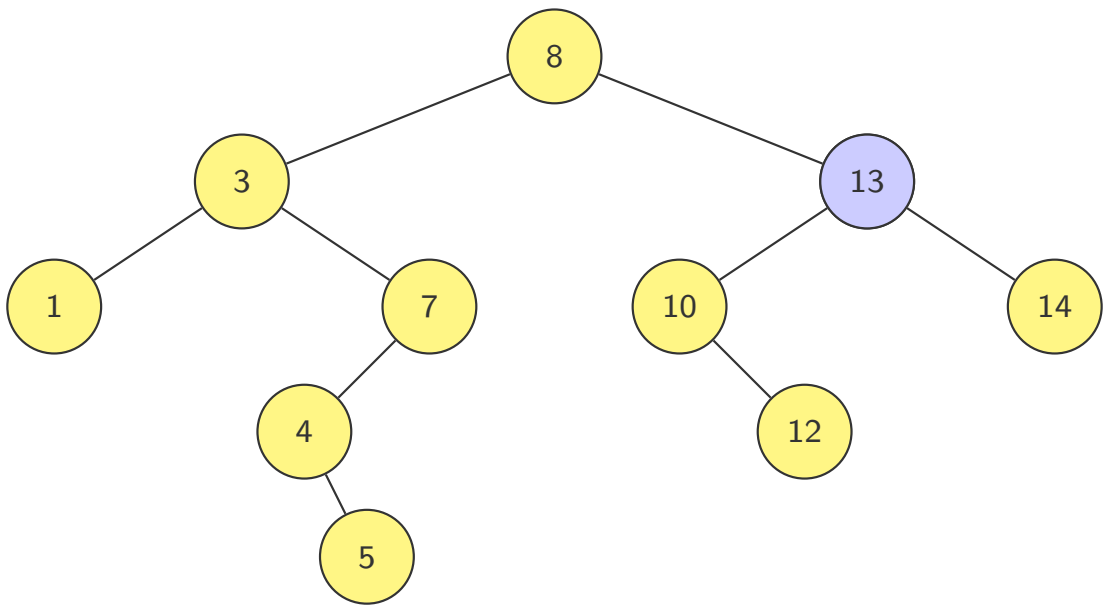


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 11

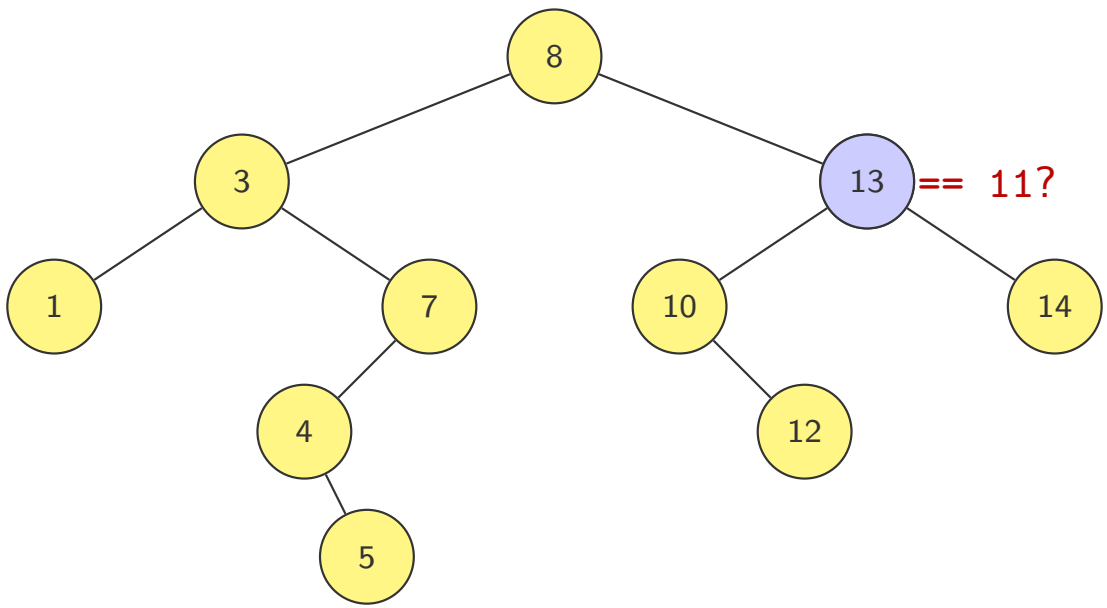


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 11

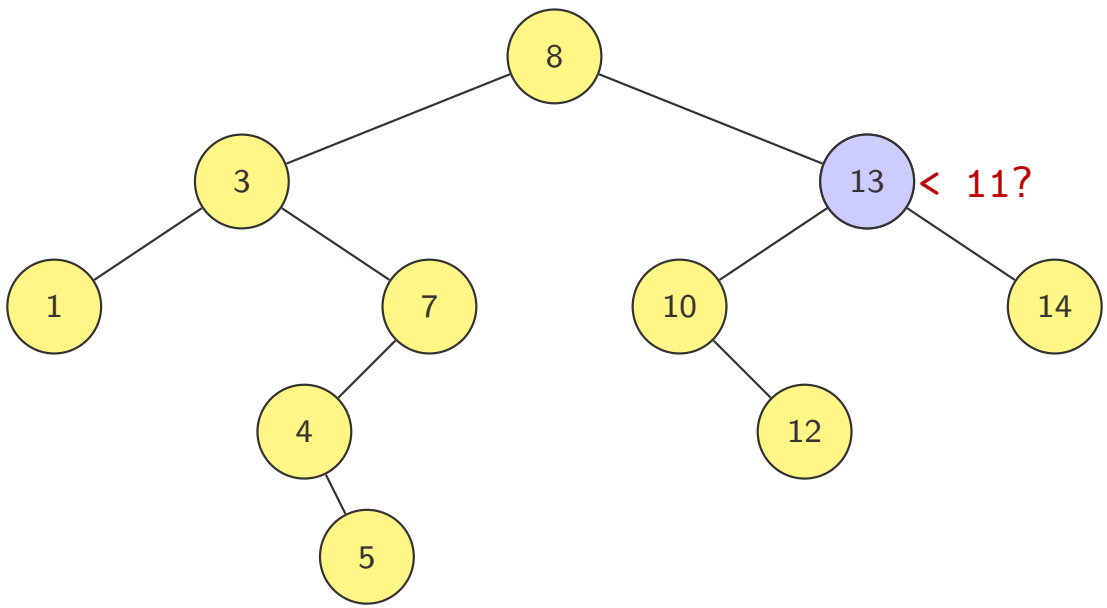


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 11

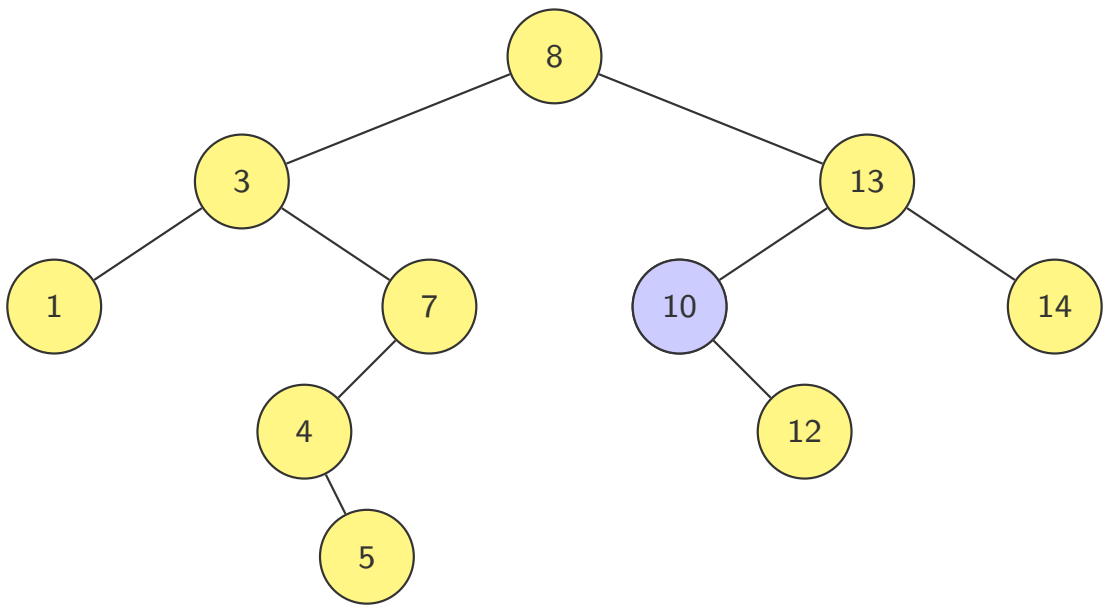


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 11

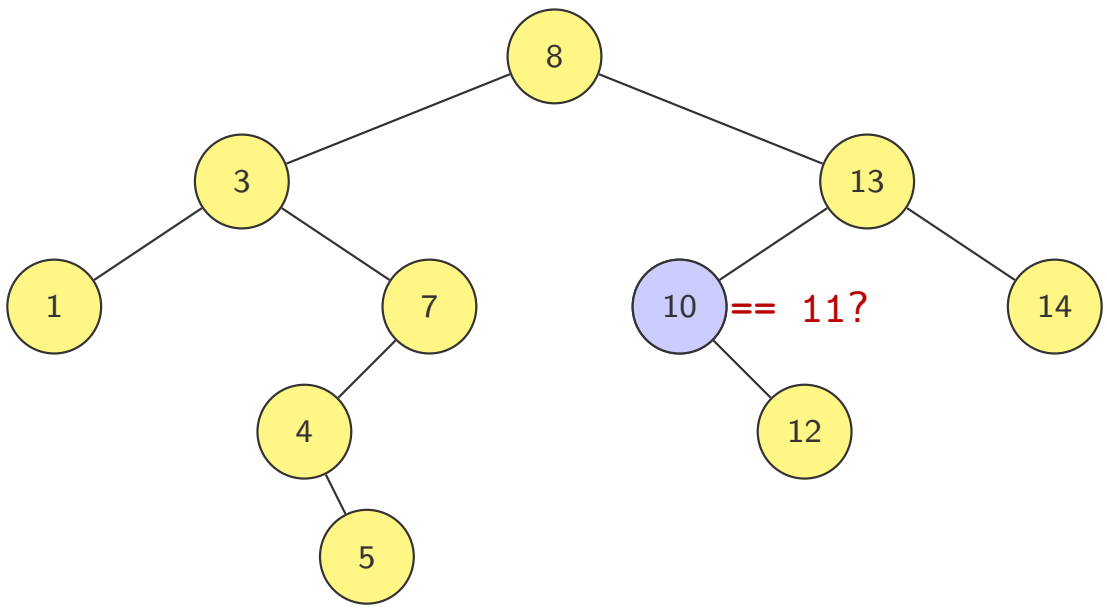


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 11

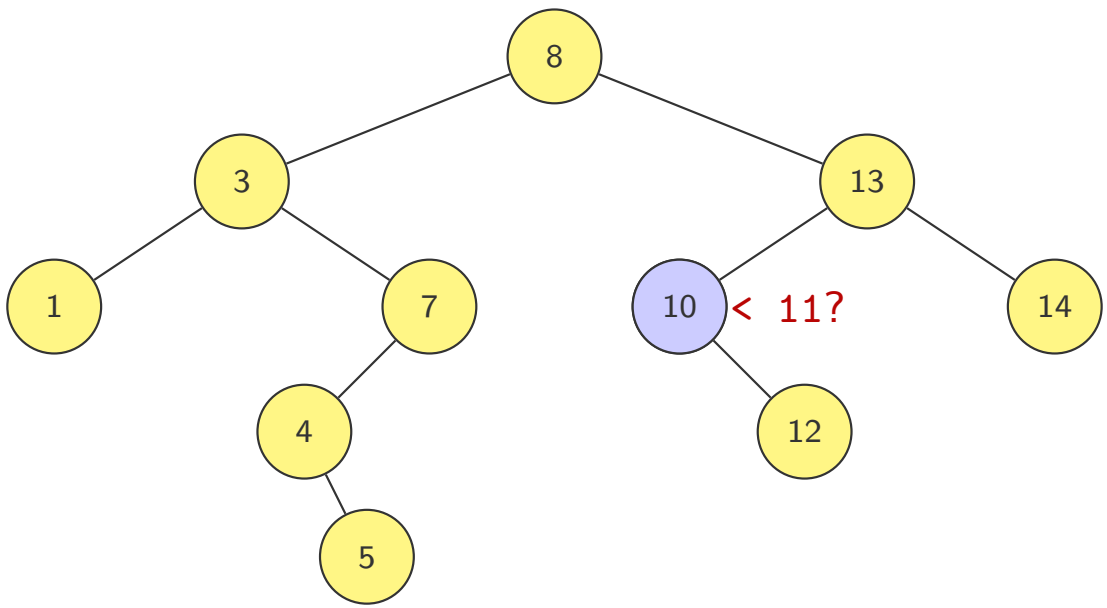


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 11

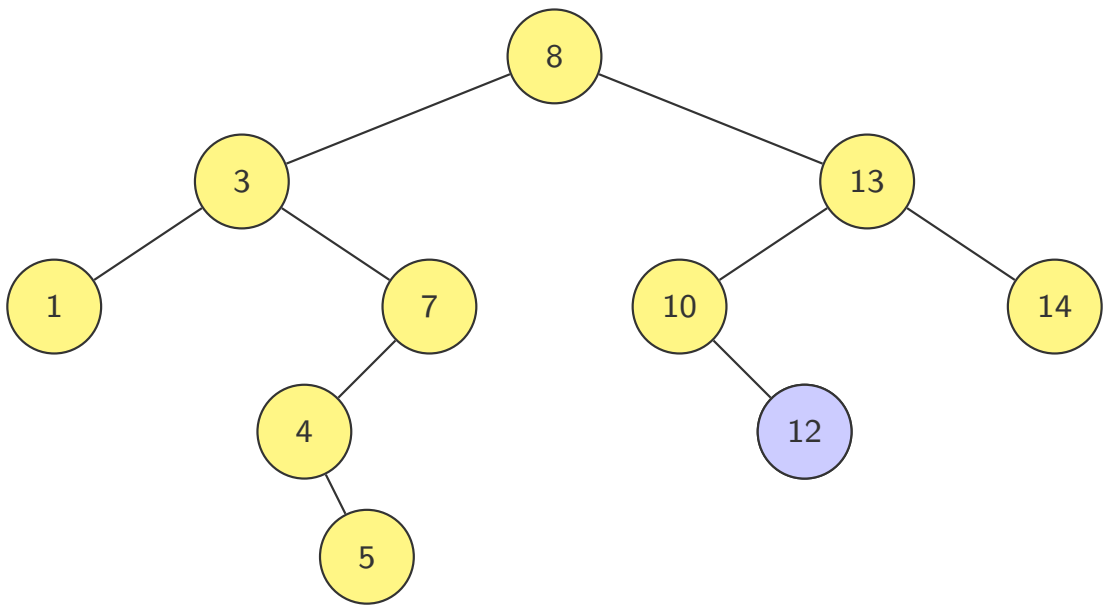


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 11

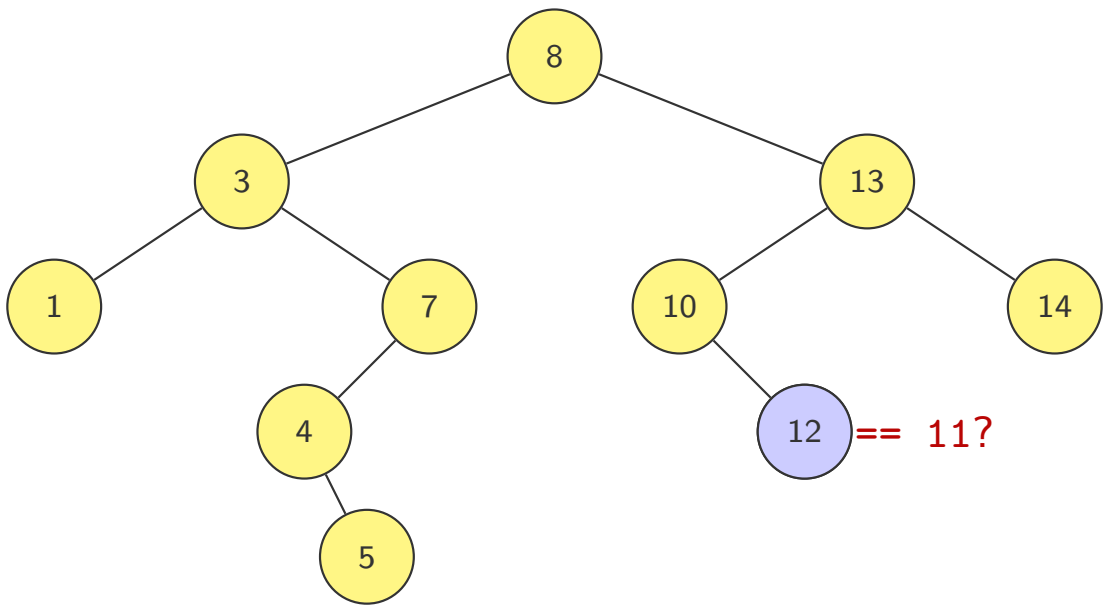


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 11

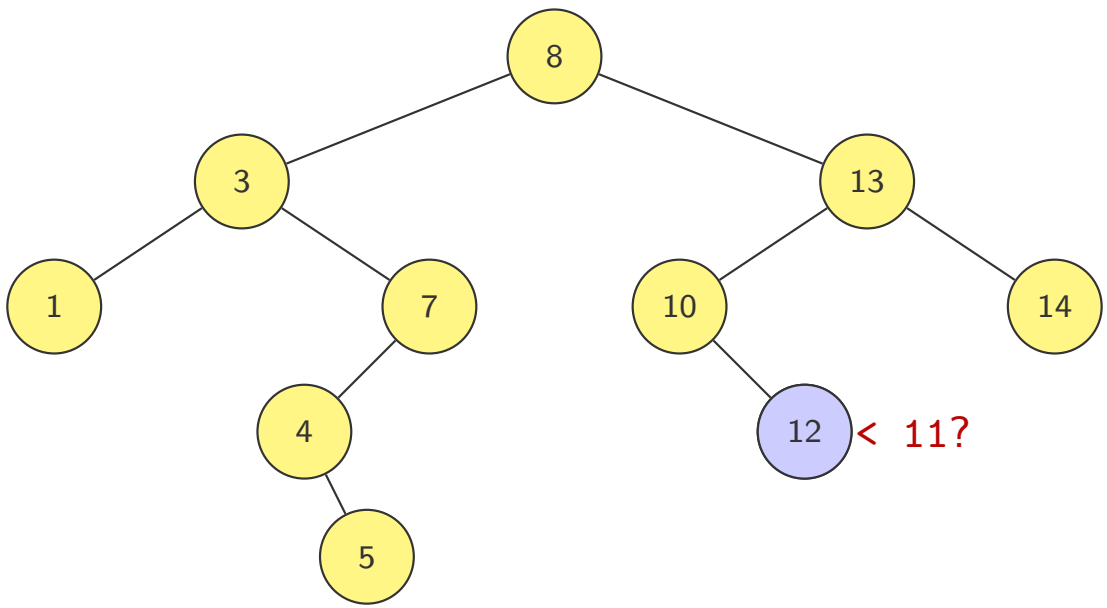


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 11

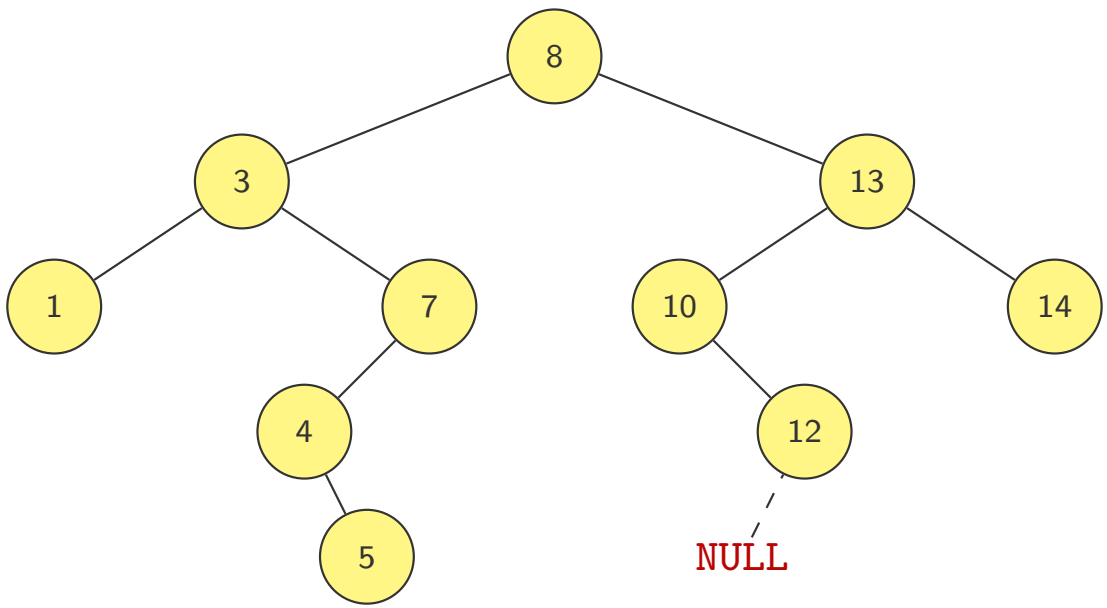


Busca por um valor

A ideia é semelhante a da busca binária:

- Ou o valor a ser buscado está na raiz da árvore
- Ou é menor do que o valor da raiz
  - Se estiver na árvore, está na subárvore esquerda
- Ou é maior do que o valor da raiz
  - Se estiver na árvore, está na subárvore direita

Ex: Buscando por 11



Pseudocódigo da Busca

Pseudocódigo da Busca

Algorithm busca(Node *node, int k)

Require: ponteiro para raiz e chave a ser buscada

Ensure: ponteiro para o nó contendo a chave k; ou nulo caso a chave não seja encontrada

```
1: if node == nullptr or node→key == k then
2:     return node
3: end if
4: if k > node→key then
5:     return busca(node→right, k)
6: else
7:     return busca(node→left, k)
8: end if
```

Eficiência da busca

Qual é o tempo da busca?

Eficiência da busca

Qual é o tempo da busca?

- depende da forma da árvore...

Eficiência da busca

Qual é o tempo da busca?

- depende da forma da árvore...

Ex: 31 nós

Eficiência da busca

Qual é o tempo da busca?

- depende da forma da árvore...

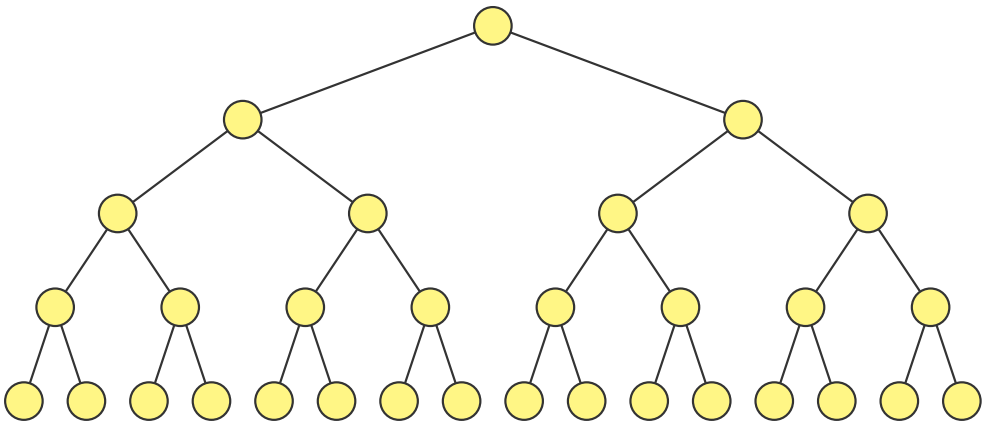
Ex: 31 nós

Eficiência da busca

Qual é o tempo da busca?

- depende da forma da árvore...

Ex: 31 nós



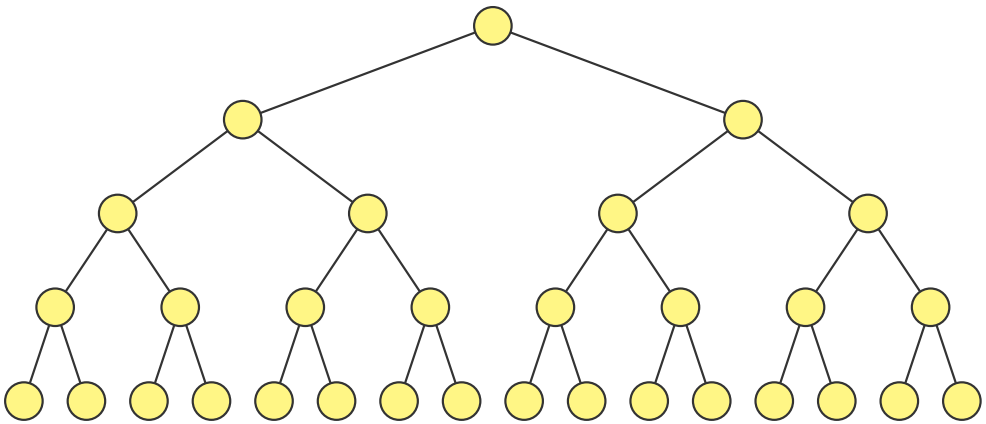
Melhor árvore: $O(\lg n)$

Eficiência da busca

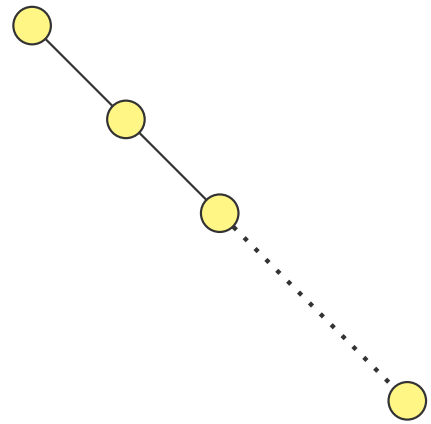
Qual é o tempo da busca?

- depende da forma da árvore...

Ex: 31 nós



Melhor árvore: $O(\lg n)$



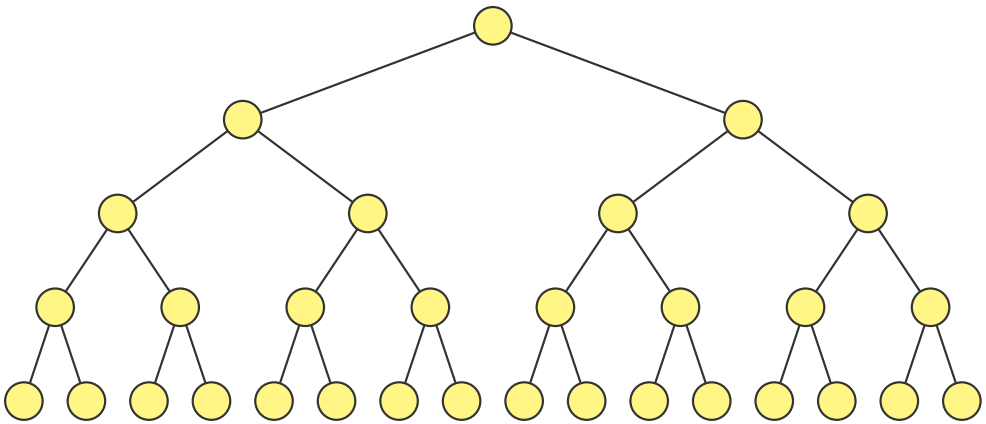
Pior árvore: $O(n)$

Eficiência da busca

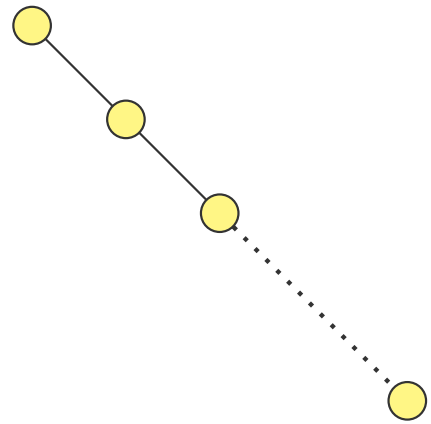
Qual é o tempo da busca?

- depende da forma da árvore...

Ex: 31 nós



Melhor árvore: $O(\lg n)$



Pior árvore: $O(n)$

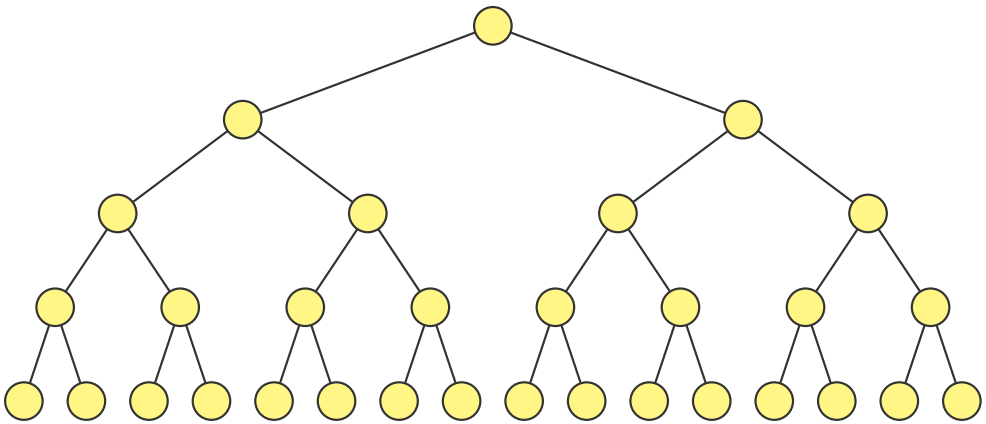
Para ter a pior árvore basta inserir em ordem crescente...

Eficiência da busca

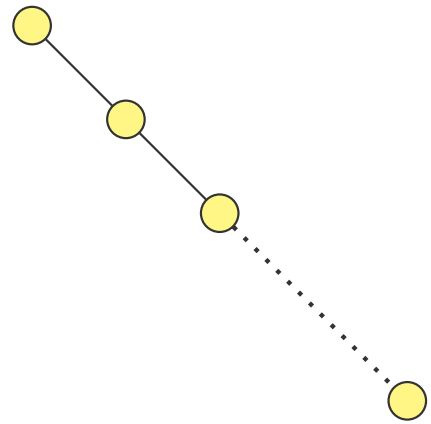
Qual é o tempo da busca?

- depende da forma da árvore...

Ex: 31 nós



Melhor árvore: $O(\lg n)$



Pior árvore: $O(n)$

Para ter a pior árvore basta inserir em ordem crescente...

Caso médio: em uma árvore com n elementos adicionados em ordem aleatória a busca demora (em média) $O(\lg n)$

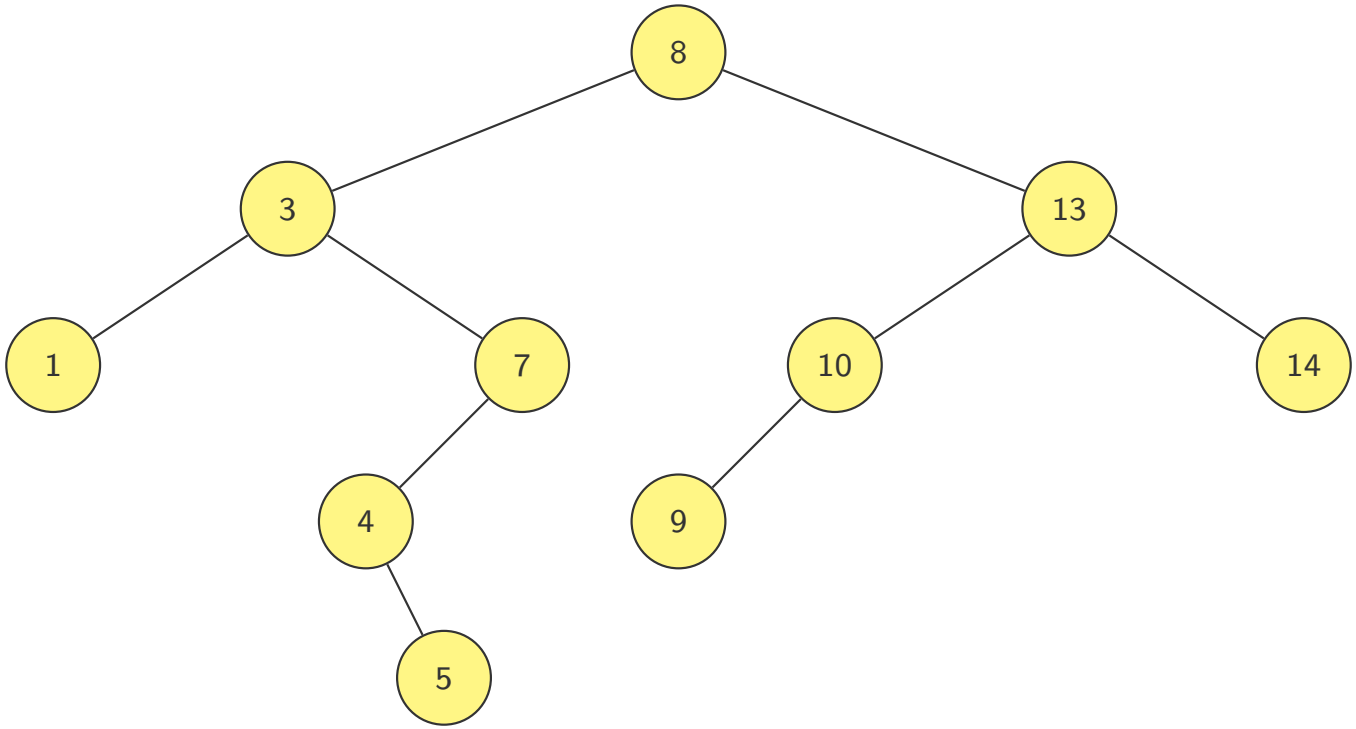
Remoção

Remoção

- Considere o problema de remover um nó de uma árvore binária de busca de tal forma que a árvore resultante continue de busca.
- Primeiro, precisamos fazer uma busca pelo nó a ser removido.
- Uma vez encontrado o nó, quais dificuldades podem surgir que dificultam a simples remoção do nó?

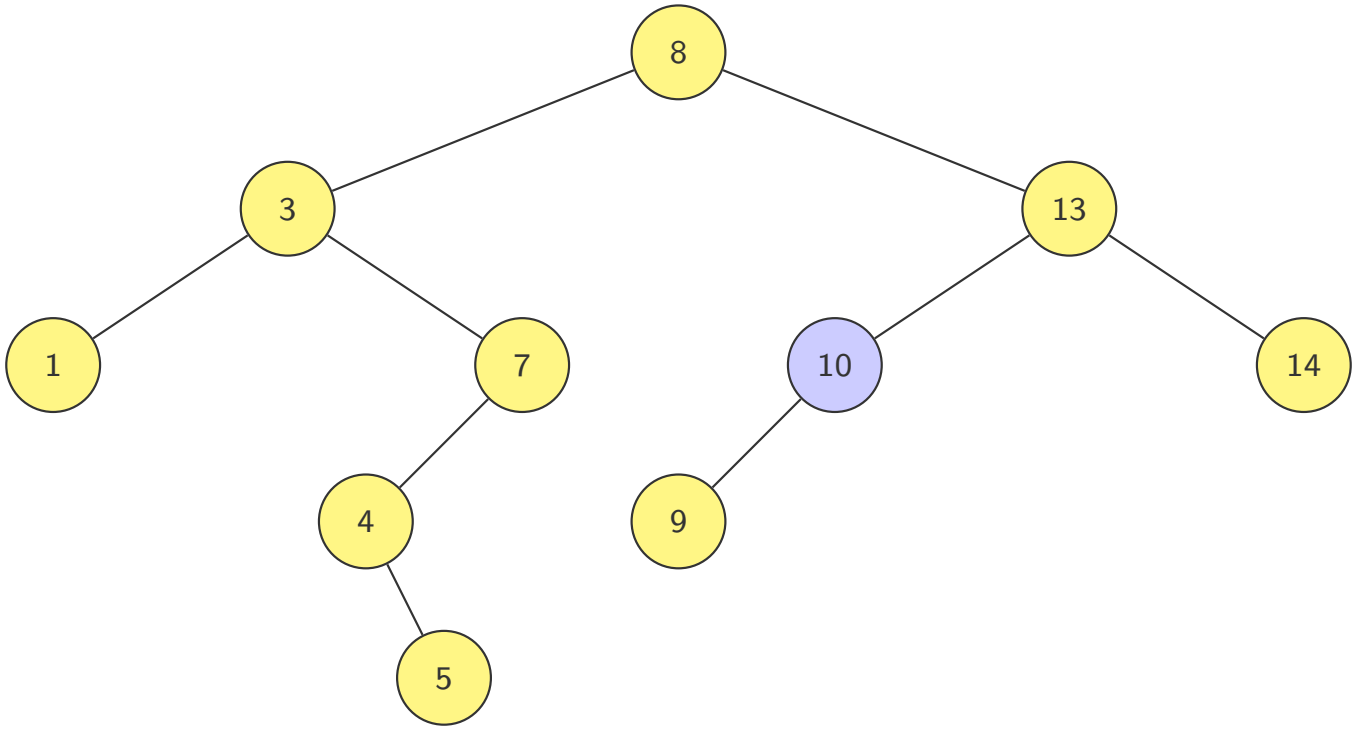
Remoção

Exemplo: queremos remover a chave 10



Remoção

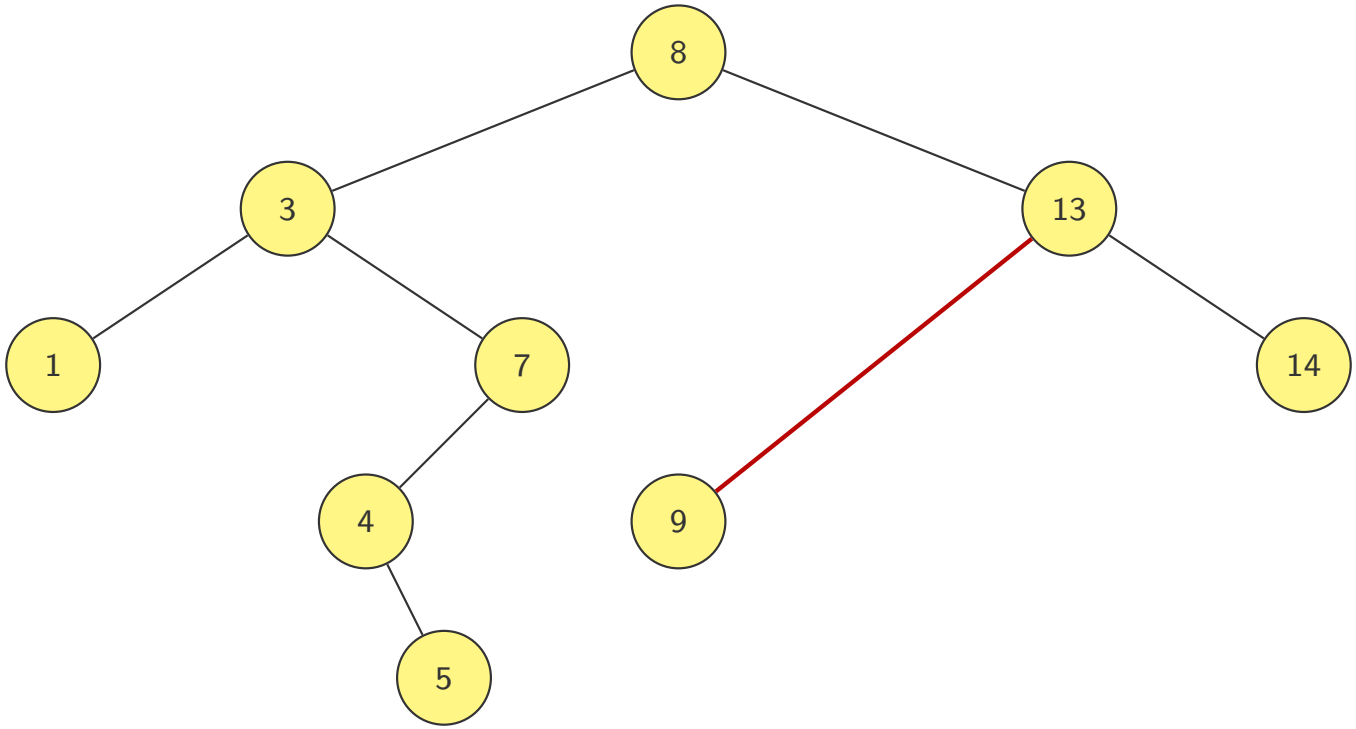
Exemplo: queremos remover a chave 10



- O nó x a ser removido pode ter exatamente um filho.

Remoção

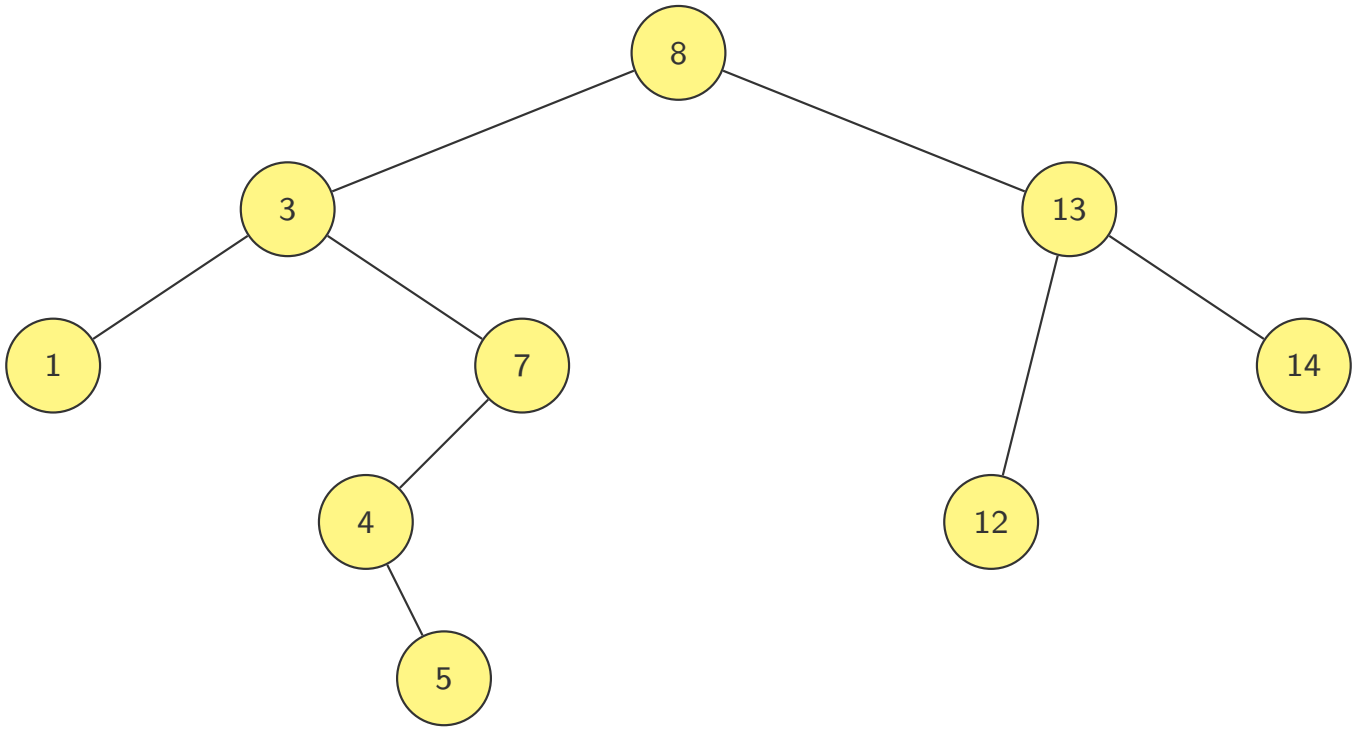
Exemplo: queremos remover a chave 10



- O nó x a ser removido pode ter exatamente um filho.
- Neste caso, fazemos o único filho de x ser filho do seu pai e depois removemos o nó x.

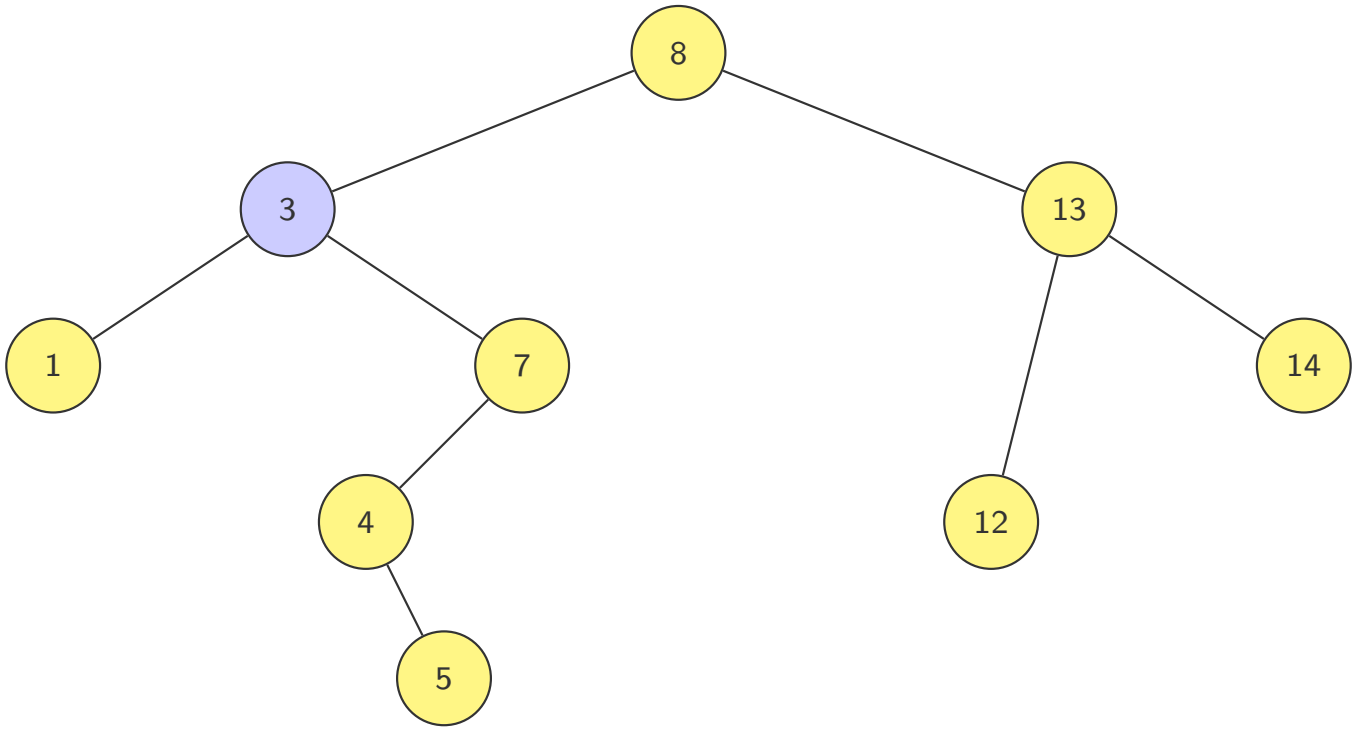
Remoção

Exemplo: removendo a chave 3



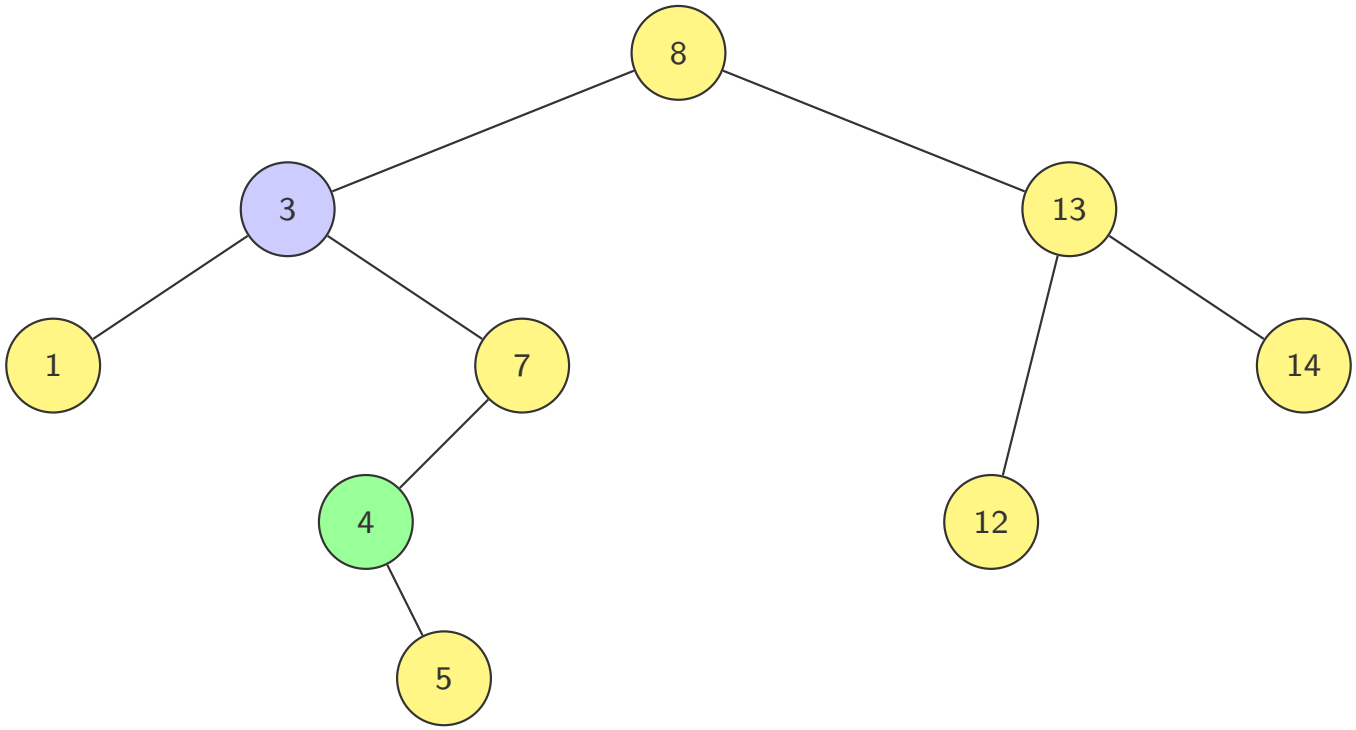
Remoção

Exemplo: removendo a chave 3



Remoção

Exemplo: removendo a chave 3

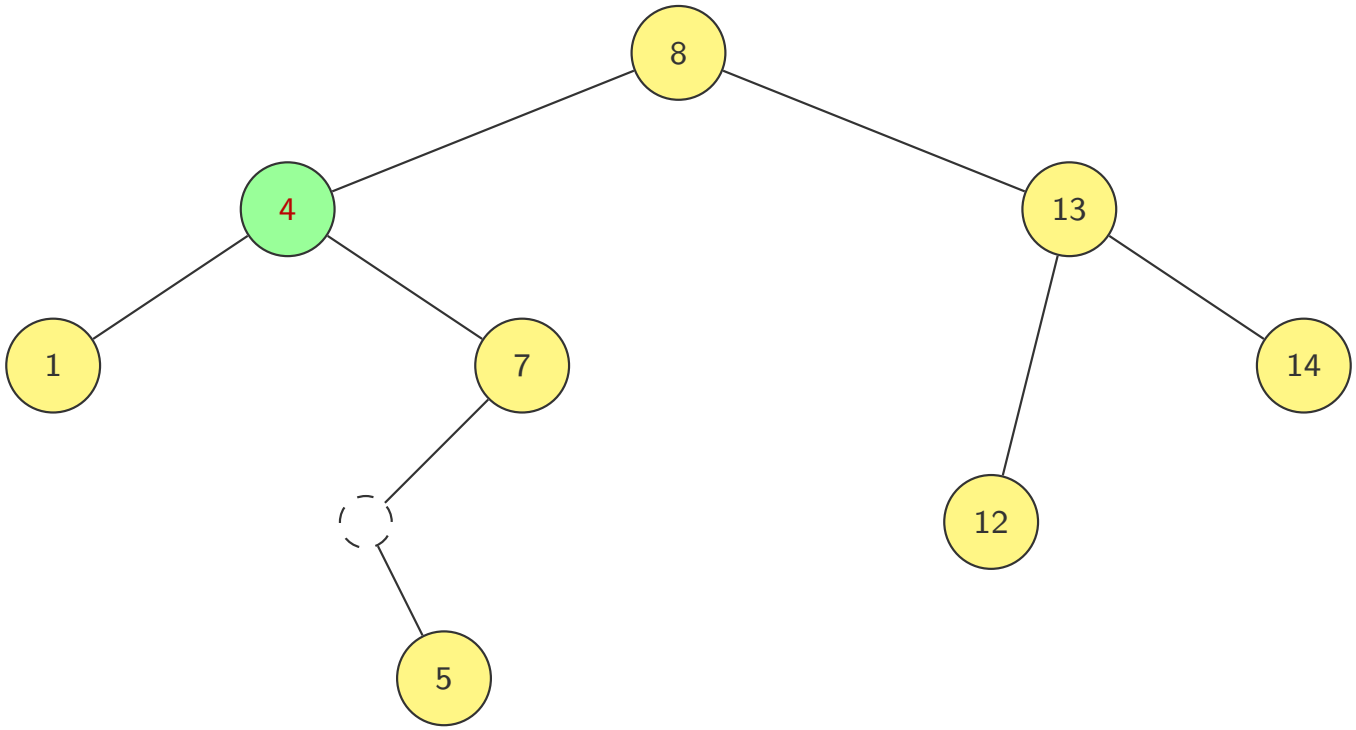


Podemos colocar o sucessor de 3 em seu lugar

- Isso mantém a propriedade da árvore binária de busca

Remoção

Exemplo: removendo a chave 3

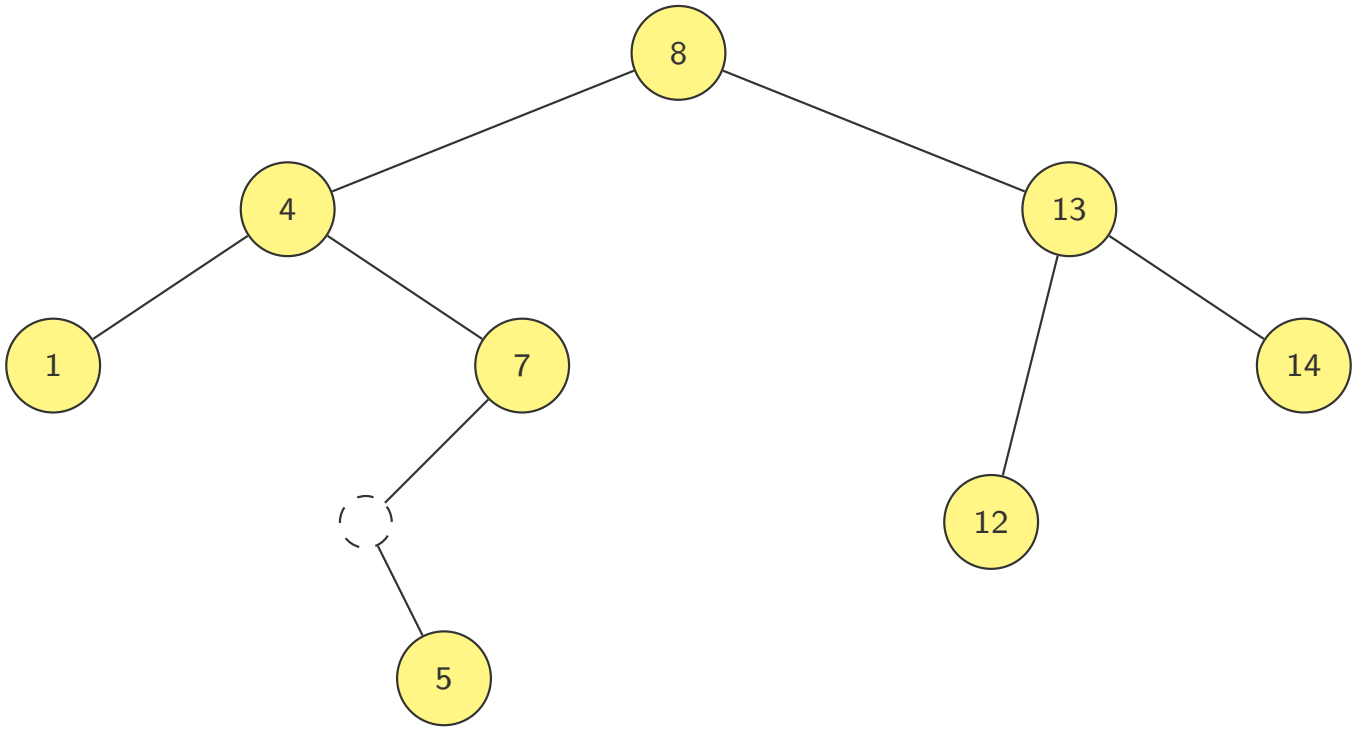


Podemos colocar o sucessor de 3 em seu lugar

- Isso mantém a propriedade da árvore binária de busca

Remoção

Exemplo: removendo a chave 3



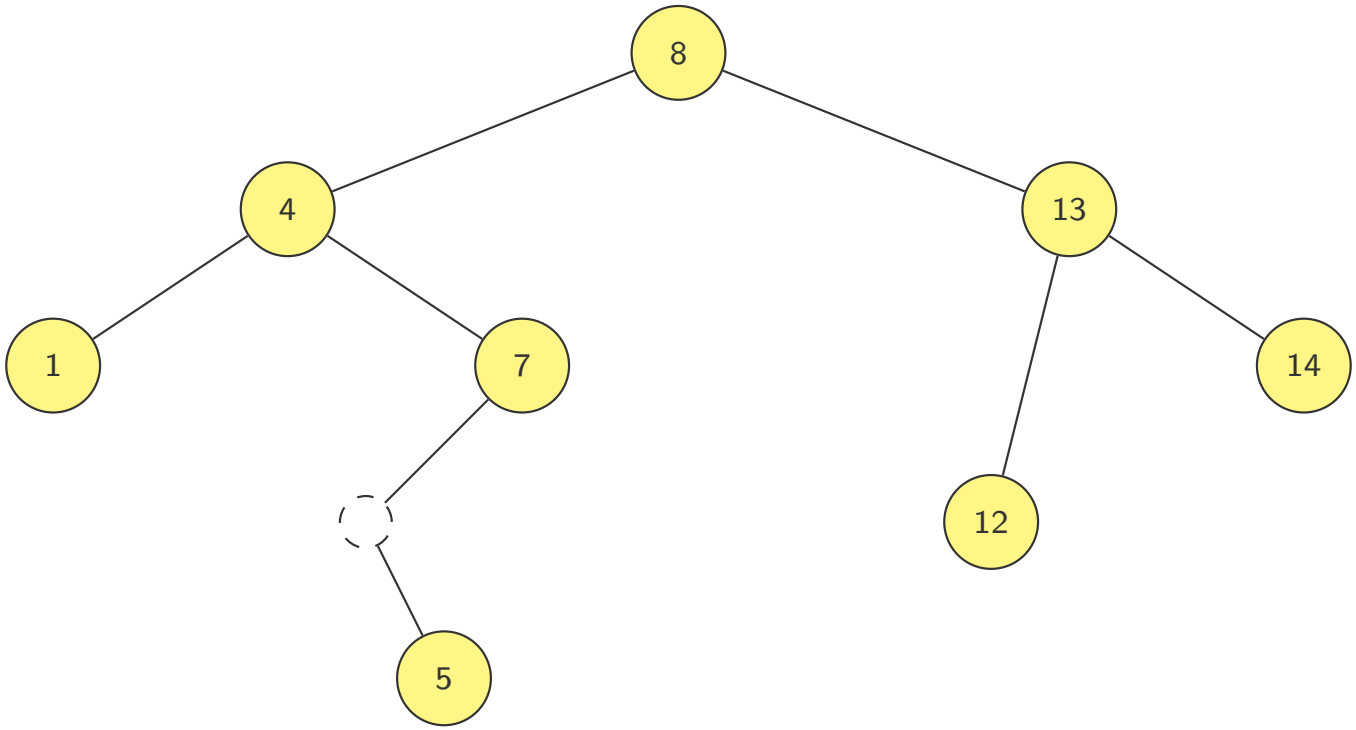
Podemos colocar o sucessor de 3 em seu lugar

- Isso mantém a propriedade da árvore binária de busca

E agora colocamos o filho direito do sucessor no seu lugar

Remoção

Exemplo: removendo a chave 3



Podemos colocar o sucessor de 3 em seu lugar

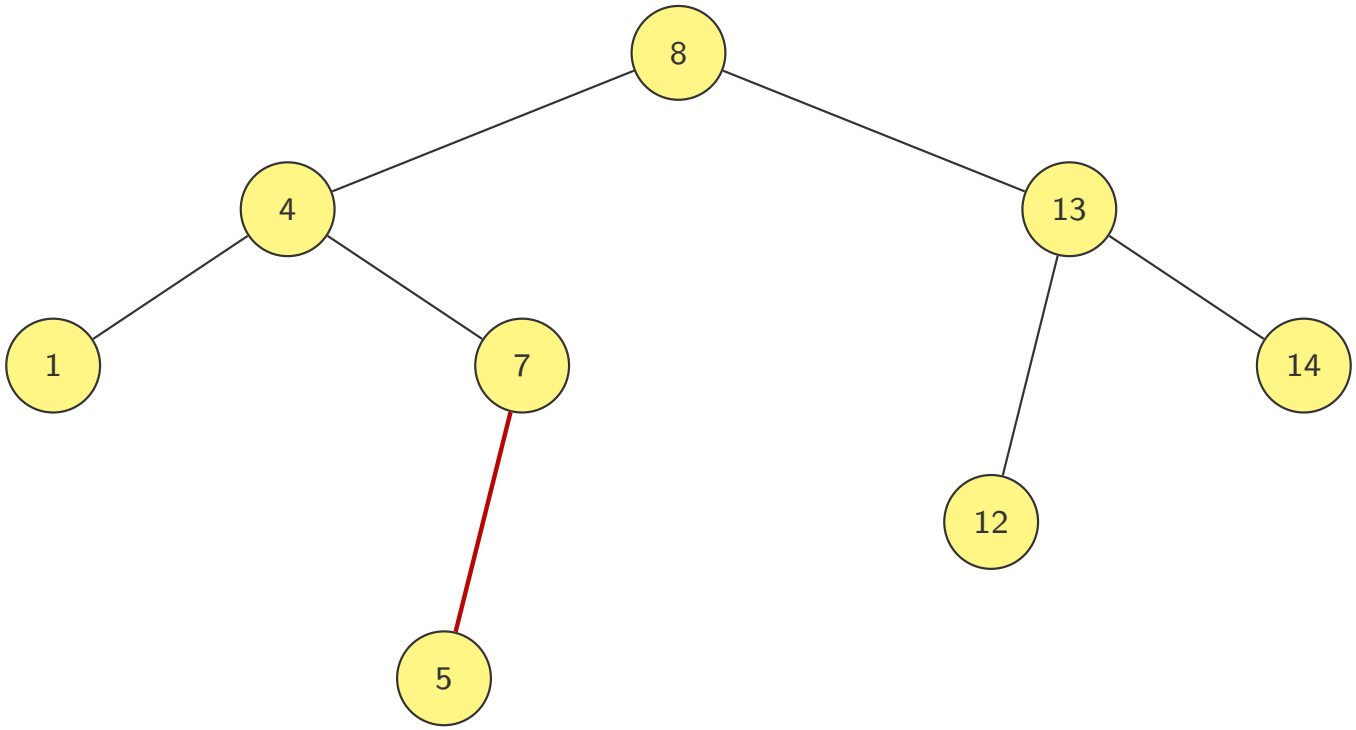
- Isso mantém a propriedade da árvore binária de busca

E agora colocamos o filho direito do sucessor no seu lugar

- O sucessor nunca tem filho esquerdo!

Remoção

Exemplo: removendo a chave 3



Podemos colocar o sucessor de 3 em seu lugar

- Isso mantém a propriedade da árvore binária de busca

E agora colocamos o filho direito do sucessor no seu lugar

- O sucessor nunca tem filho esquerdo!

Remoção

- Note que o nó a ser removido é a raiz de uma árvore.
  Essa raiz pode ou não ter filhos.

- Logo, trato esse problema como a remoção da raiz de uma árvore.

Remoção

- Note que o nó a ser removido é a raiz de uma árvore.
  Essa raiz pode ou não ter filhos.

- Logo, trato esse problema como a remoção da raiz de uma árvore.

Vou seguir um dos seguintes passos para remover a raiz:

1. Se a raiz **não tiver** filho direito, o filho esquerdo dela assume o papel de raiz;
2. Se a raiz **tiver** filho direito, basta fazer com que o nó sucessor à raiz assuma o papel da raiz.

Pseudocódigo da Remoção: Parte I

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  /**
2   * Algoritmo Recursivo
3   * Entrada: Recebe a raiz da arvore e a
4   *          chave k a ser removida
5   * Saida:   Retorna a raiz da nova arvore.
6   */
7  remove(Node *node, int k) {
8      if(node == nulo)
9          return nulo
10     if(k == node->key) // Achou o nodo a ser removido
11         return removeRoot(node); // funcao auxiliar
12     // Ainda nao achamos o nodo, vamos busca-lo
13     if(k < node->key)
14         node->left = remove(k, node->left);
15     else
16         node->right = remove(k, node->right);
17     return node;
18 }
```

Pseudocódigo da Remoção: Parte II


```
1  // Recebe um ponteiro node para a raiz de uma arvore e
2  // remove a raiz, rearranjando a arvore de modo que ela
3  // continue sendo de busca. Devolve o endereco da nova raiz
4  removeRoot(Node *node) {
5      Node *pai, *q;
6      if(node->right == nulo)
7          q = node->left;
8      else {
9          pai = node;
10         q = node->right;
11         while(q->left != nulo) {
12             pai = q;
13             q = q->left;
14         }
15         if(pai != node) {
16             pai->left = q->right;
17             q->right = node->right;
18         }
19         q->left = node->left;
20     }
21     delete node;
22     return q;
23 }
```

Remoção: Complexidade

Qual a complexidade do algoritmo de remoção?

Exercícios

Exercícios

- Conclua a implementação das funções que foram deixadas em aberto nos slides anteriores.

- Suponha que todo nó da BST tenha agora um ponteiro para nó pai. Reimplemente as operações vistas nessa aula considerando este novo ponteiro.

- Escreva uma função que receba como argumento uma BST vazia e um vetor $A[p..q]$ com $q - p + 1$ inteiros em ordem crescente e popule a BST com os inteiros do vetor A de modo que ela seja uma árvore binária de busca completa (altura igual a $\lceil \log_2 (n + 1) \rceil$). Sua função pode ter o seguinte protótipo:
  `void construirBST(BST *t, int A[], int p, int q);`

- Escreva uma função que transforme uma árvore binária de busca em um vetor crescente.

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM