Árvore Binária de Busca

Estrutura de Dados — QXD0010

Prof. Atílio Gomes Luiz
gomes.atilio@ufc.br

Universidade Federal do Ceará

1º semestre/2020

Introdução

Tempo para Inserção, Remoção e Busca

Usando Listas Duplamente Ligadas:

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

Tempo para Inserção, Remoção e Busca

Usando Listas Duplamente Ligadas:

- 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 Ligadas:

- 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 Ligadas:

- 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 Ligadas:

- 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 Ligadas:

- 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 $O(\lg n)$ se a árvore for balanceada

Árvore Binária de Busca

Uma Árvore Binária de Busca (ABB) é uma árvore binária em que cada nó contém um elemento de um conjunto ordenável

Árvore Binária de Busca

Uma Árvore Binária de Busca (ABB) é uma árvore binária em que cada nó contém um elemento de um conjunto ordenável

Cada nó r, com subárvores esquerda T_e e direita T_d satisfaz a seguinte propriedade:

Árvore Binária de Busca

Uma Árvore Binária de Busca (ABB) é uma árvore binária em que cada nó contém um elemento de um conjunto ordenável

Cada nó r, com subárvores esquerda T_e e direita T_d satisfaz a seguinte propriedade:

1. $e < r$ para todo elemento $e \in T_e$

Árvore Binária de Busca

Uma Árvore Binária de Busca (ABB) é uma árvore binária em que cada nó contém um elemento de um conjunto ordenável

Cada nó r, com subárvores esquerda T_e e direita T_d satisfaz a seguinte propriedade:

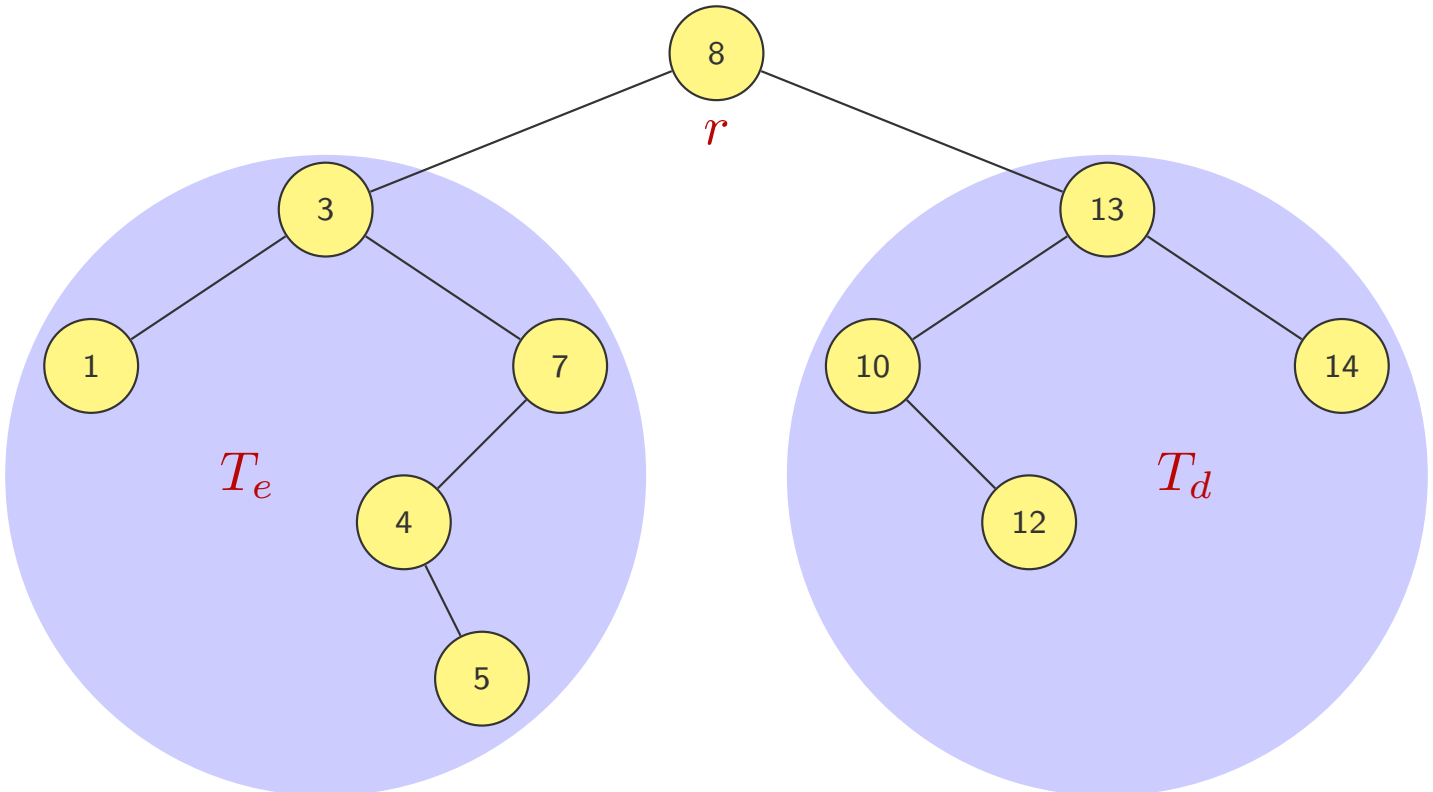
1. $e < r$ para todo elemento $e \in T_e$
2. $d > r$ para todo elemento $d \in T_d$

Árvore Binária de Busca

Uma Árvore Binária de Busca (ABB) é uma árvore binária em que cada nó contém um elemento de um conjunto ordenável

Cada nó r, com subárvores esquerda T_e e direita T_d satisfaz a seguinte propriedade:

1. $e < r$ para todo elemento $e \in T_e$
2. $d > r$ para todo elemento $d \in T_d$



Implementação em C++

ABB.h — TAD Árvore Binária de Busca

Universidade Federal do Ceará
Campus Quixadá

```
1 #ifndef ABB_H
2 #define ABB_H
3
4 struct NoArv;
5
6 int abb_getChave(NoArv *no); // retorna chave do no
7 void abb_preordem(NoArv *raiz); // percurso em preordem
8 NoArv* abb_destruir(NoArv *raiz); // libera memoria
```

ABB.h — TAD Árvore Binária de Busca

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  #ifndef ABB_H
2  #define ABB_H
3
4  struct NoArv;
5
6  int abb_getChave(NoArv *no); // retorna chave do no
7  void abb_preordem(NoArv *raiz); // percurso em preordem
8  NoArv* abb_destruir(NoArv *raiz); // libera memoria
9
10 NoArv* abb_busca(NoArv *raiz, int k); // busca chave k
11 NoArv* abb_inserir(NoArv *raiz, int k); // insere chave k
12 NoArv* abb_remover(NoArv *raiz, int k); // remove chave k
13 void remove_antecessor(NoArv *no); // funcao auxiliar
```

ABB.h — TAD Árvore Binária de Busca

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  #ifndef ABB_H
2  #define ABB_H
3
4  struct NoArv;
5
6  int abb_getChave(NoArv *no); // retorna chave do no
7  void abb_preordem(NoArv *raiz); // percurso em preordem
8  NoArv* abb_destruir(NoArv *raiz); // libera memoria
9
10 NoArv* abb_busca(NoArv *raiz, int k); // busca chave k
11 NoArv* abb_inserir(NoArv *raiz, int k); // insere chave k
12 NoArv* abb_remover(NoArv *raiz, int k); // remove chave k
13 void remove_antecessor(NoArv *no); // funcao auxiliar
14
15 NoArv* abb_minimo(NoArv *raiz); // retorna no minimo
16 NoArv* abb_maximo(NoArv *raiz); // retorna no
```

ABB.h — TAD Árvore Binária de Busca

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  #ifndef ABB_H
2  #define ABB_H
3
4  struct NoArv;
5
6  int abb_getChave(NoArv *no); // retorna chave do no
7  void abb_preordem(NoArv *raiz); // percurso em preordem
8  NoArv* abb_destruir(NoArv *raiz); // libera memoria
9
10 NoArv* abb_busca(NoArv *raiz, int k); // busca chave k
11 NoArv* abb_inserir(NoArv *raiz, int k); // insere chave k
12 NoArv* abb_remover(NoArv *raiz, int k); // remove chave k
13 void remove_antecessor(NoArv *no); // funcao auxiliar
14
15 NoArv* abb_minimo(NoArv *raiz); // retorna no minimo
16 NoArv* abb_maximo(NoArv *raiz); // retorna no
17
18 NoArv* abb_sucessor(NoArv* x, NoArv *raiz);
19 NoArv* abb_antecessor(NoArv* x, NoArv *raiz);
20 NoArv* ancestral_sucessor(NoArv *x, NoArv* raiz);
21 NoArv* ancestral_antecessor(NoArv *x, NoArv* raiz);
22
23 #endif
```

ABB.cpp — Arquivo de Implementação

Primeiras linhas do arquivo de implementação:

```
1  #include <iostream>
2  #include <climits>
3  #include "ABB.h"
4  using namespace std;
5
6  // O no da arvore eh definido como um struct
7  struct NoArv {
8      int chave;
9      NoArv *esq;
10     NoArv *dir;
11 };
12
13 // Recebe um no da arvore e retorna o valor da sua chave
14 // Se o no for nulo, retorna o valor INT_MIN
15 int abb_getChave(NoArv *no) {
16     if(no != nullptr) return no->chave;
17     else return INT_MIN;
18 }
```

Busca por um valor

A ideia é semelhante àquela da busca binária:

Busca por um valor

A ideia é semelhante àquela da busca binária:

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

Busca por um valor

A ideia é semelhante àquela 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 àquela 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 àquela 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 àquela 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 àquela 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

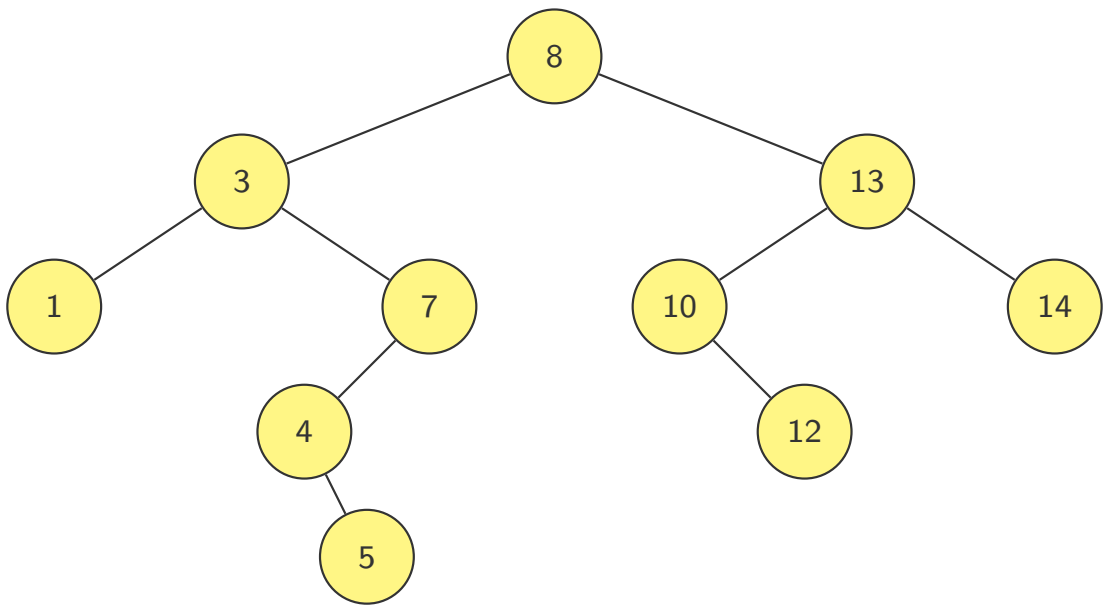
Ex: Buscando por 4

Busca por um valor

A ideia é semelhante àquela 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

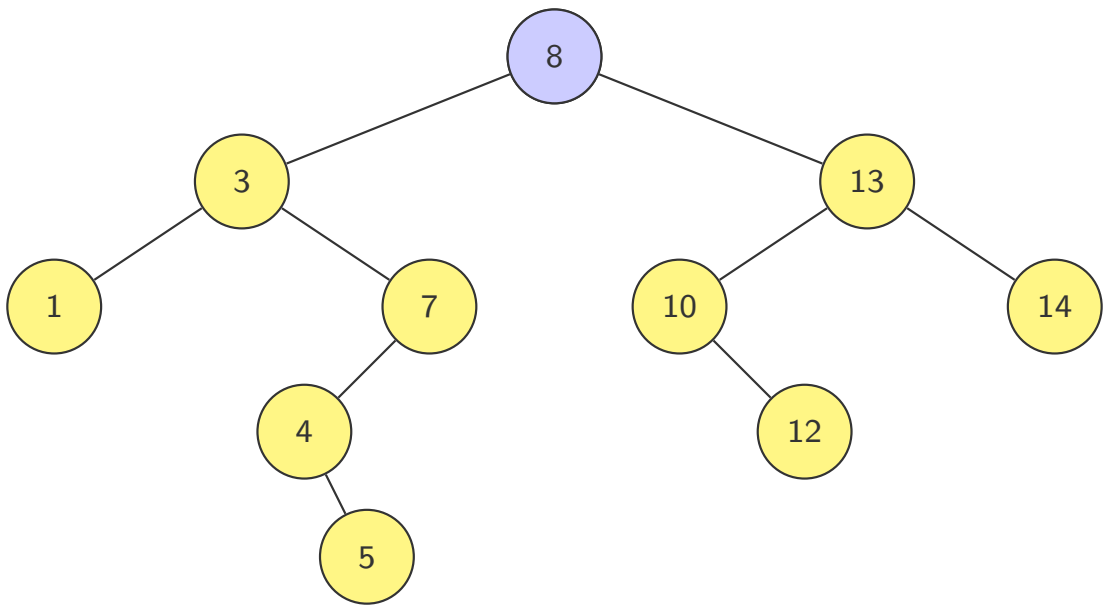


Busca por um valor

A ideia é semelhante àquela 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

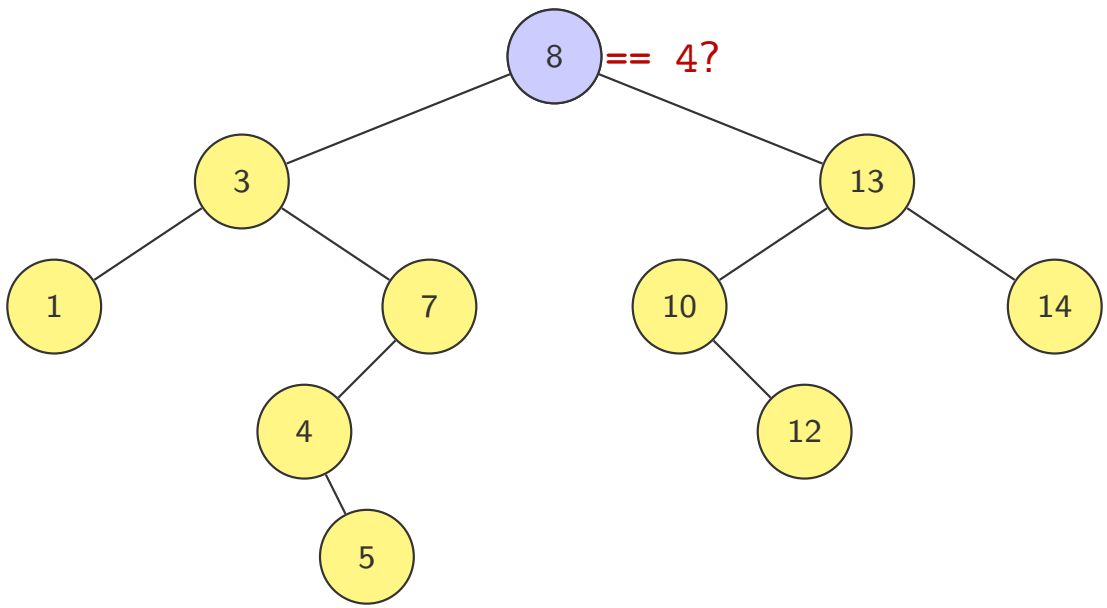


Busca por um valor

A ideia é semelhante àquela 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

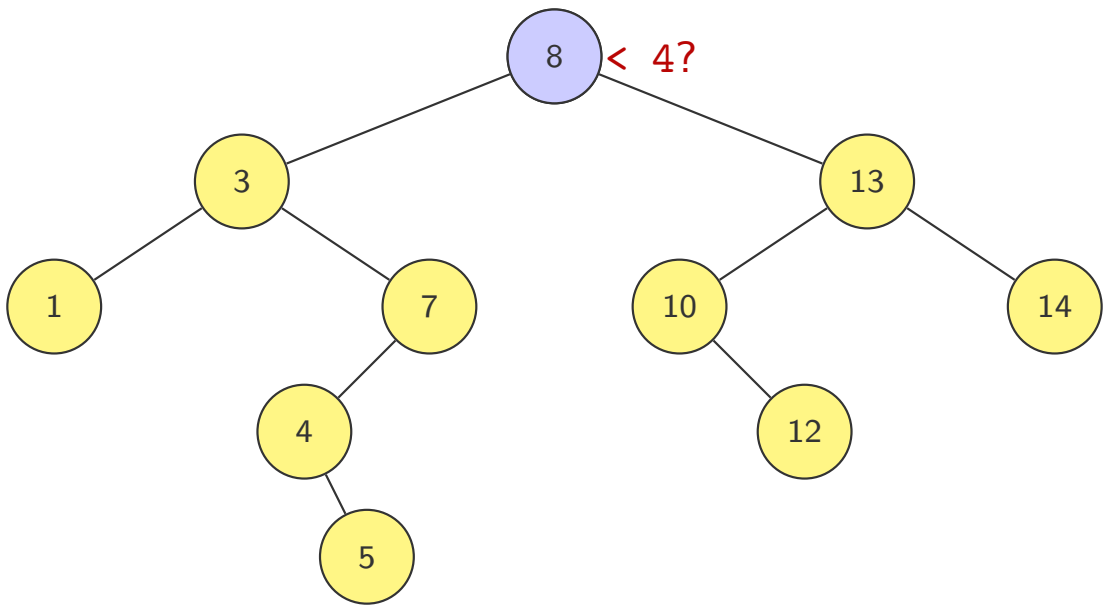


Busca por um valor

A ideia é semelhante àquela 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

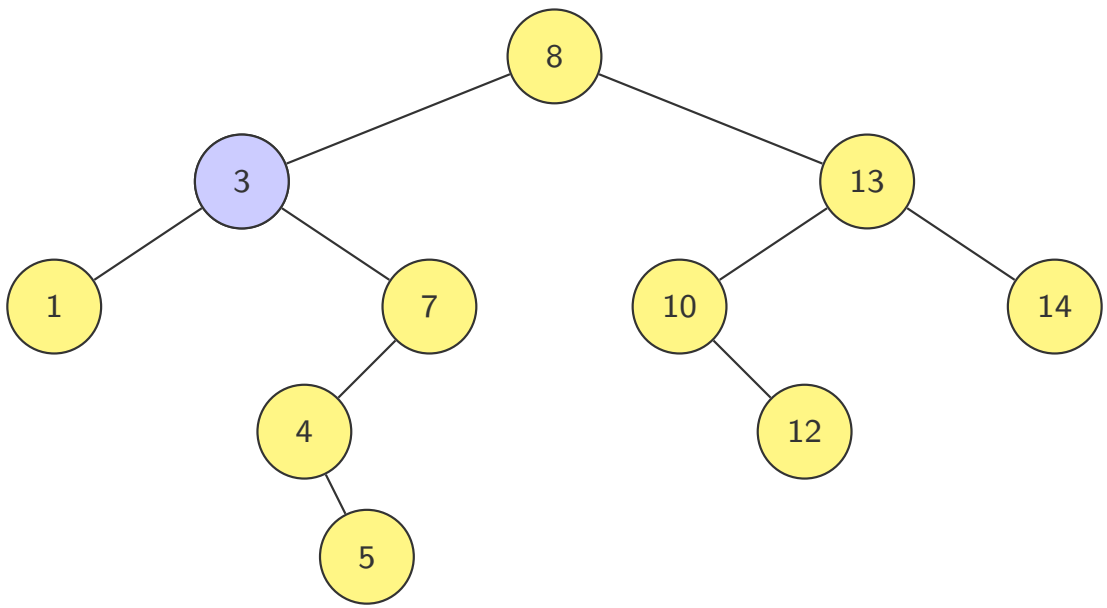


Busca por um valor

A ideia é semelhante àquela 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

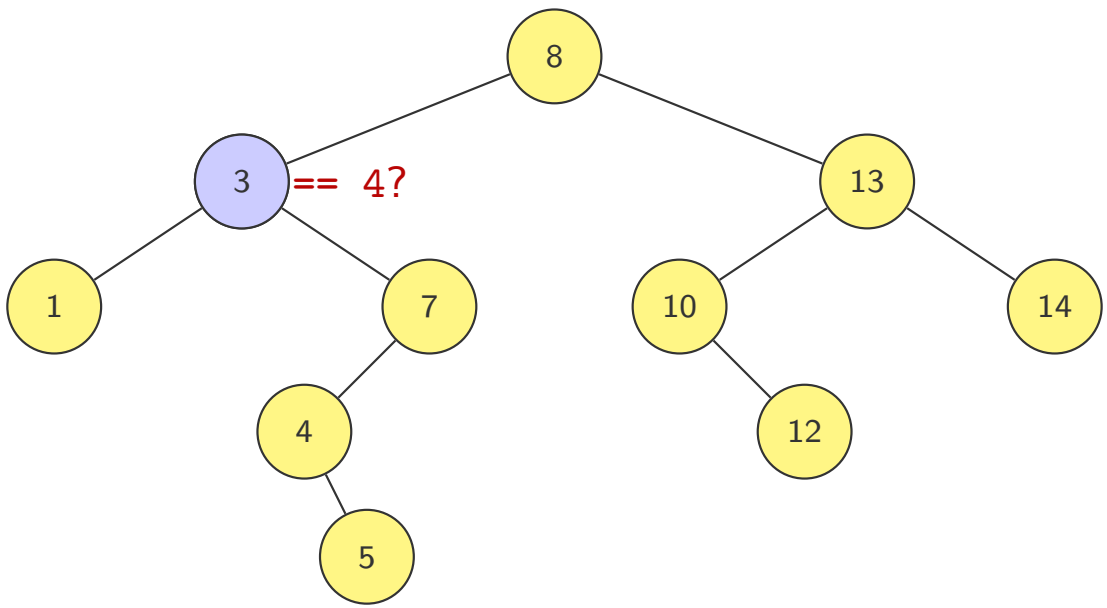


Busca por um valor

A ideia é semelhante àquela 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

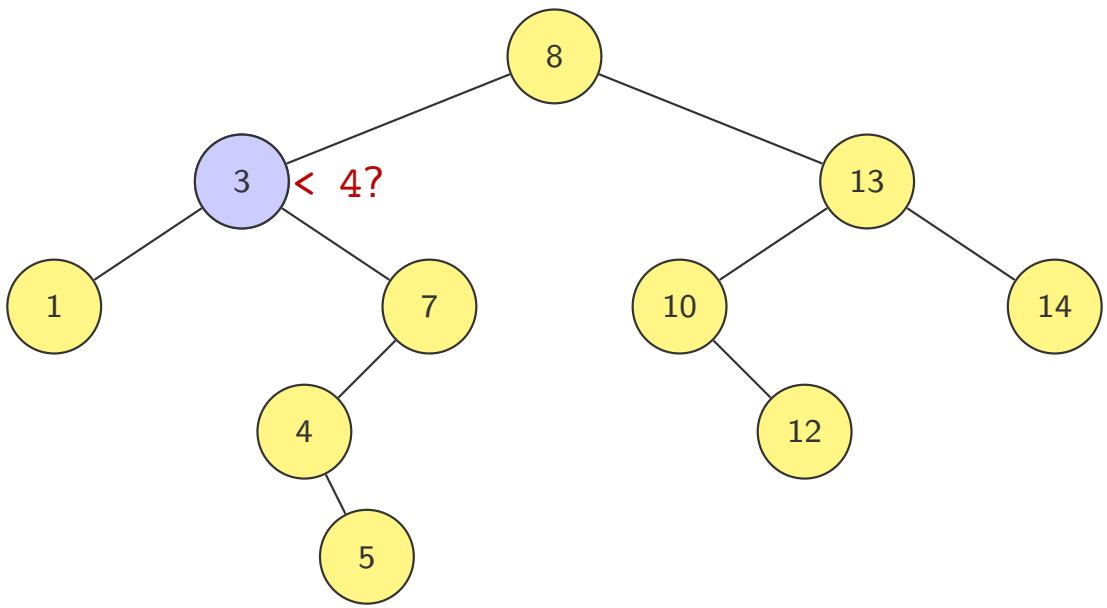


Busca por um valor

A ideia é semelhante àquela 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

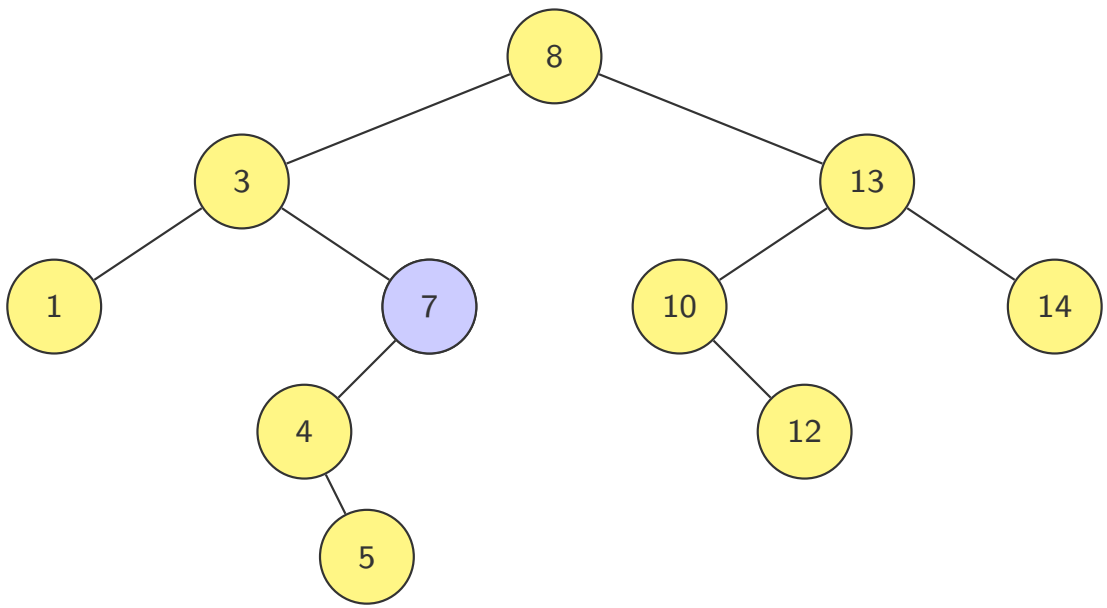


Busca por um valor

A ideia é semelhante àquela 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

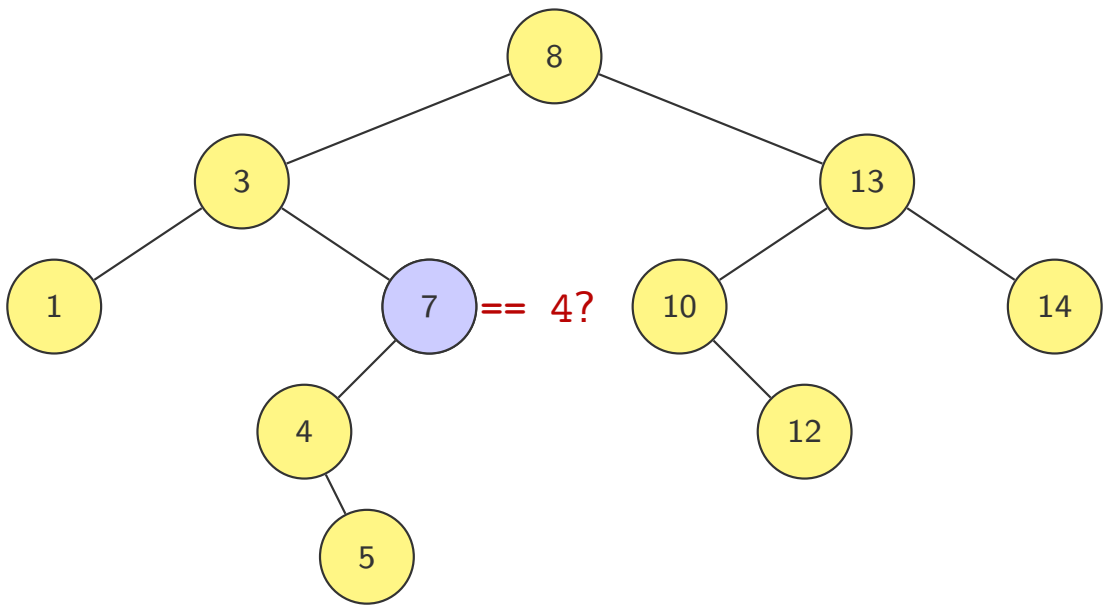


Busca por um valor

A ideia é semelhante àquela 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

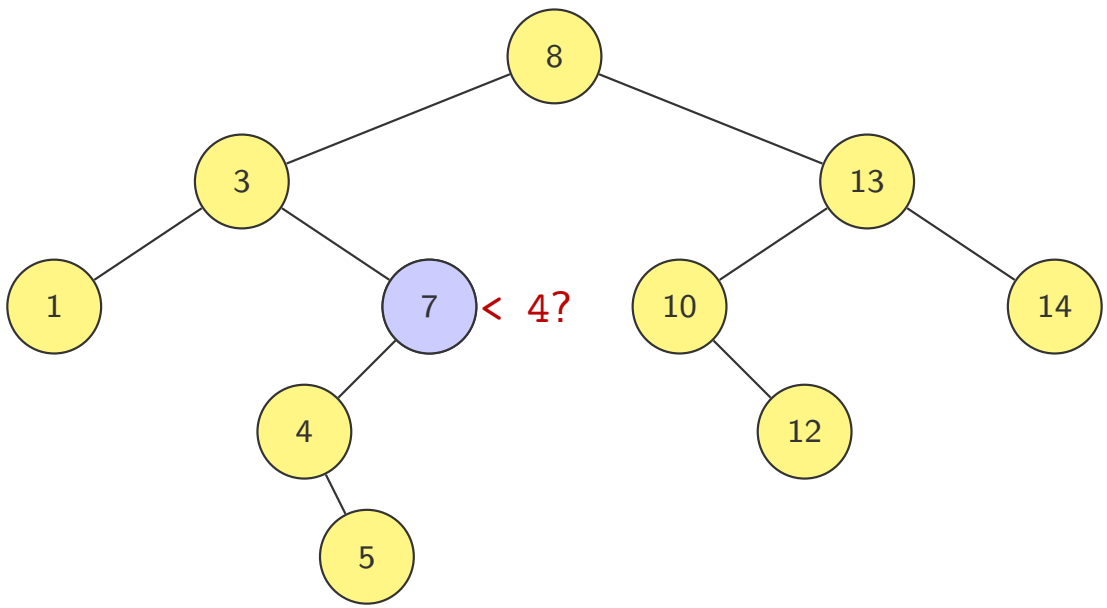


Busca por um valor

A ideia é semelhante àquela 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

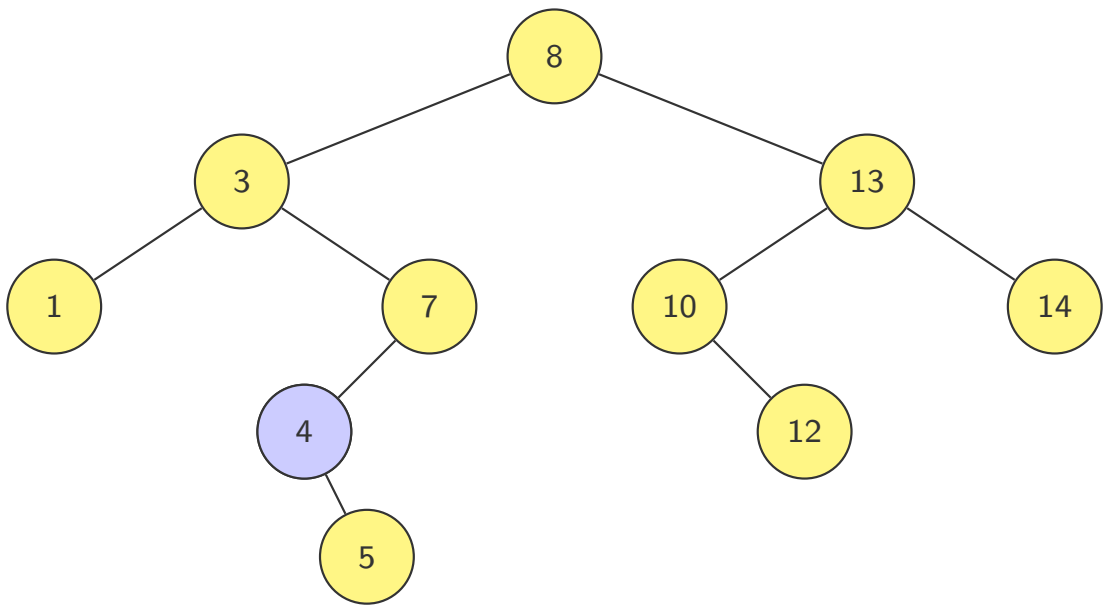


Busca por um valor

A ideia é semelhante àquela 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

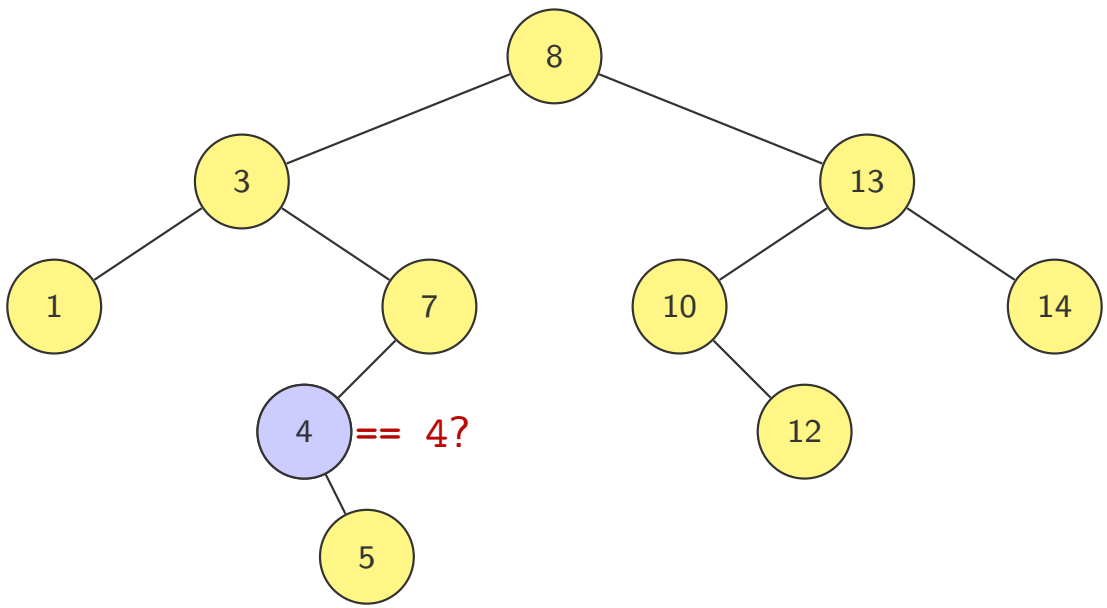


Busca por um valor

A ideia é semelhante àquela 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

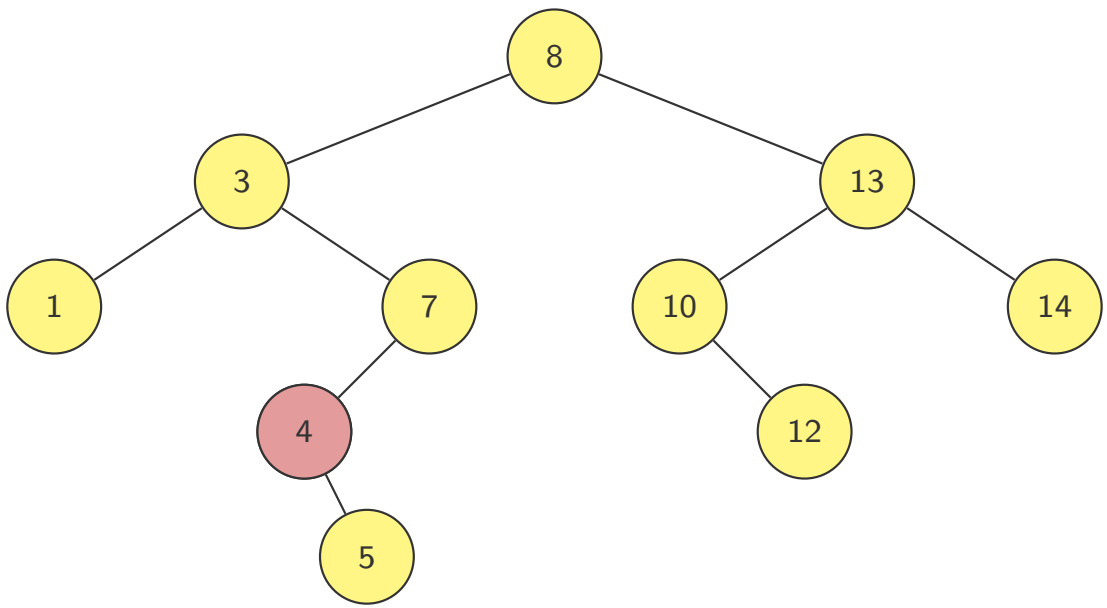


Busca por um valor

A ideia é semelhante àquela 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

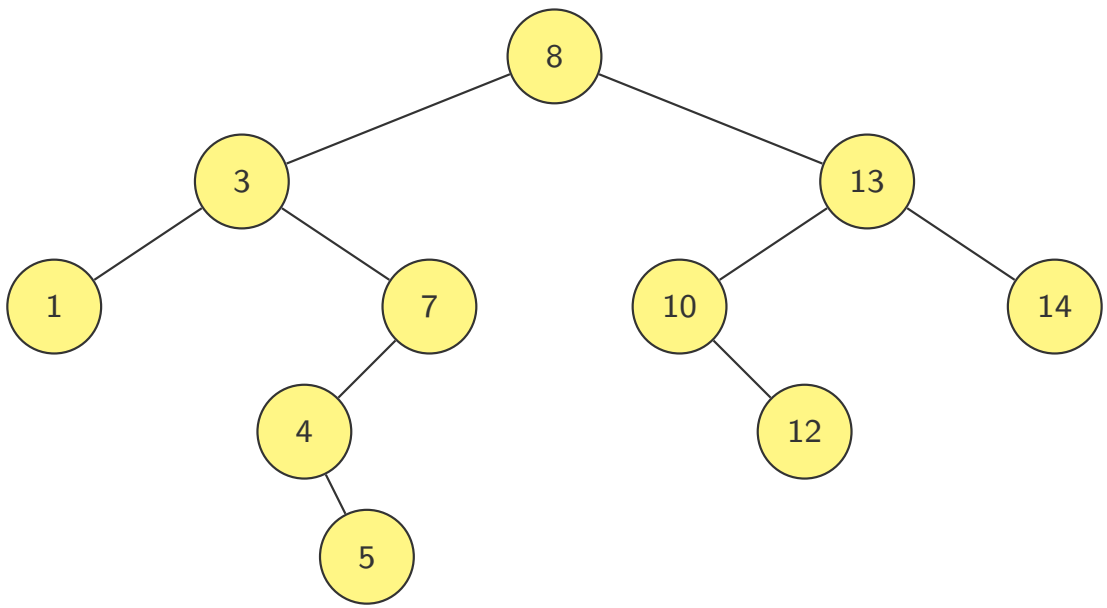


Busca por um valor

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

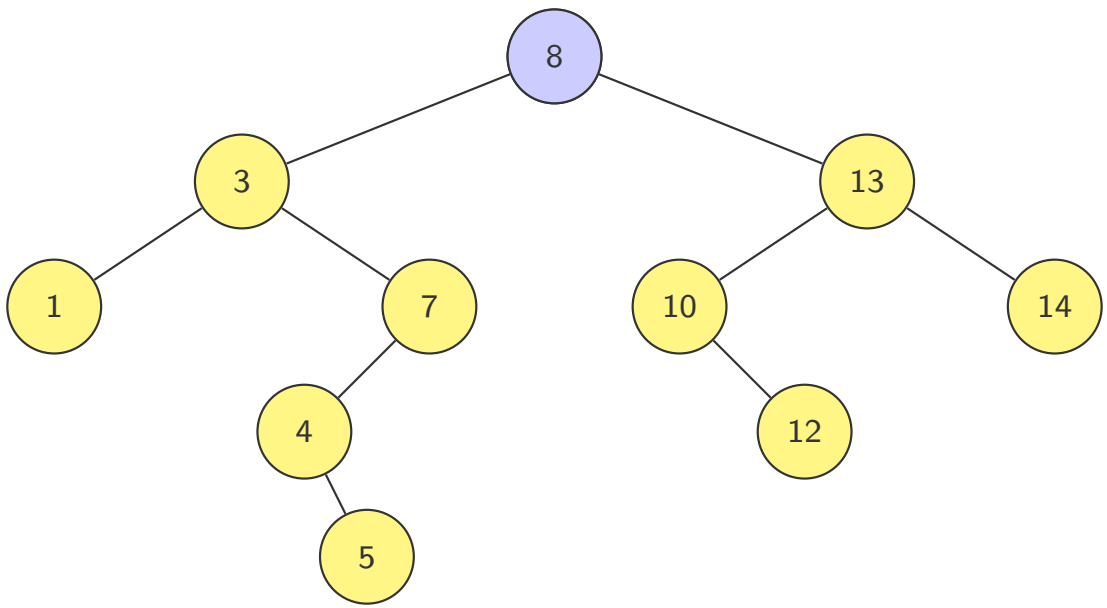


Busca por um valor

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

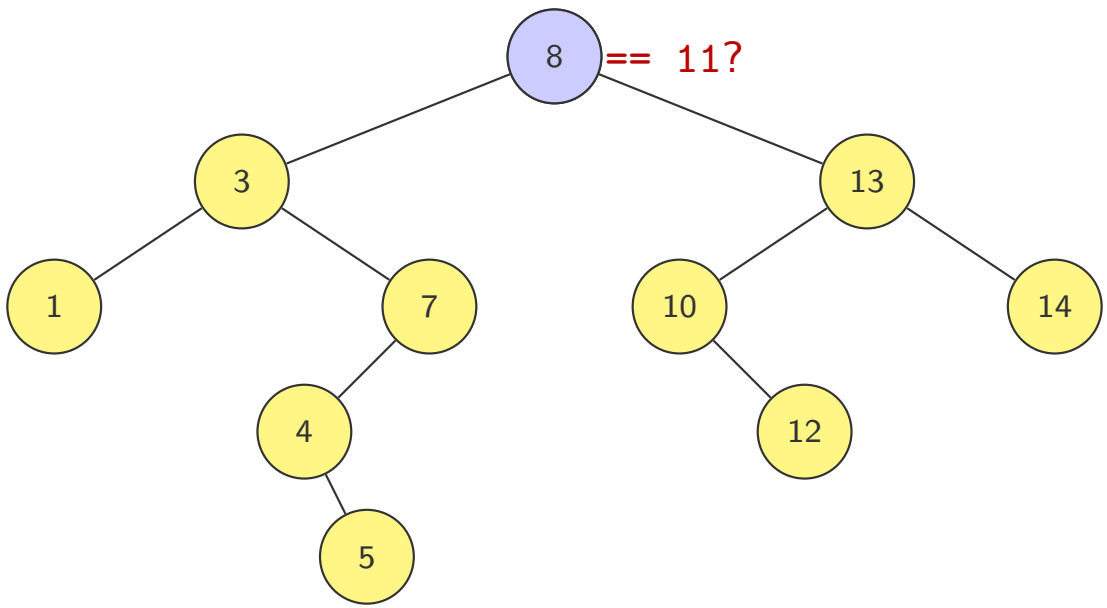


Busca por um valor

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

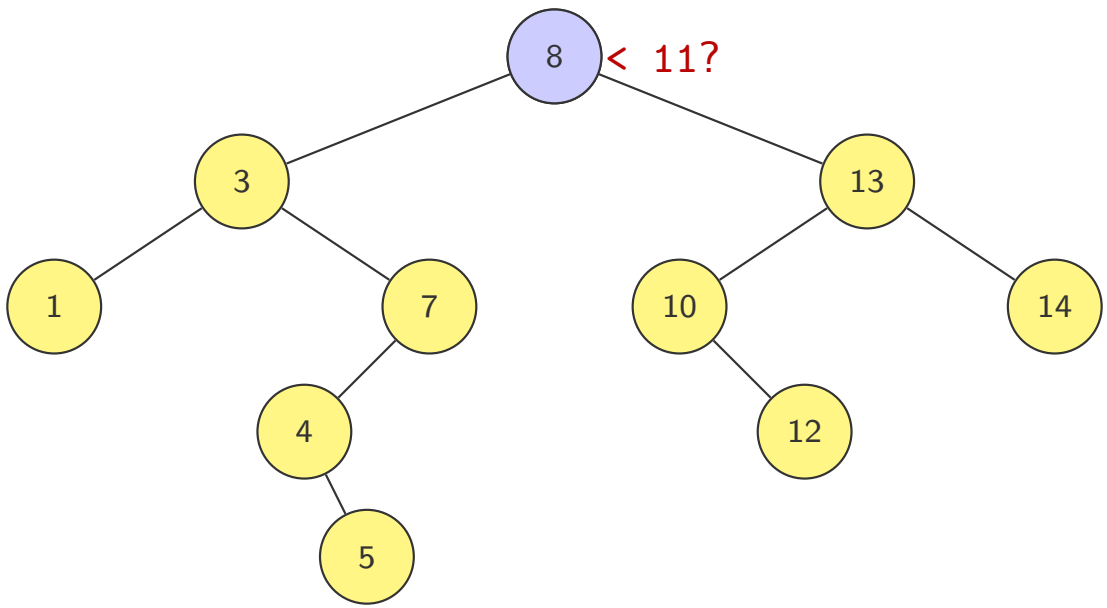


Busca por um valor

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

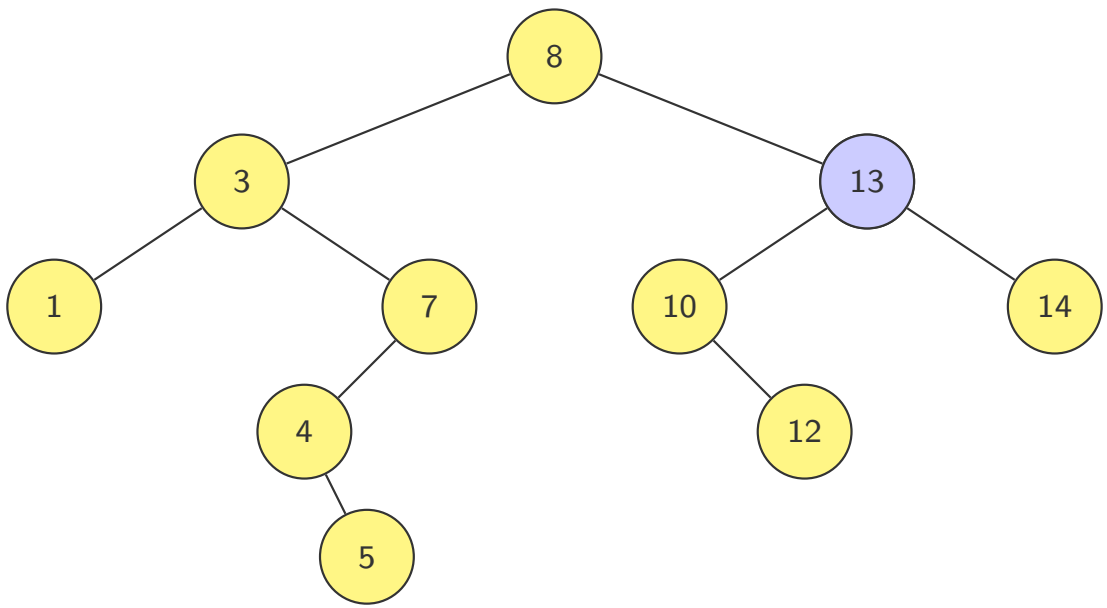


Busca por um valor

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

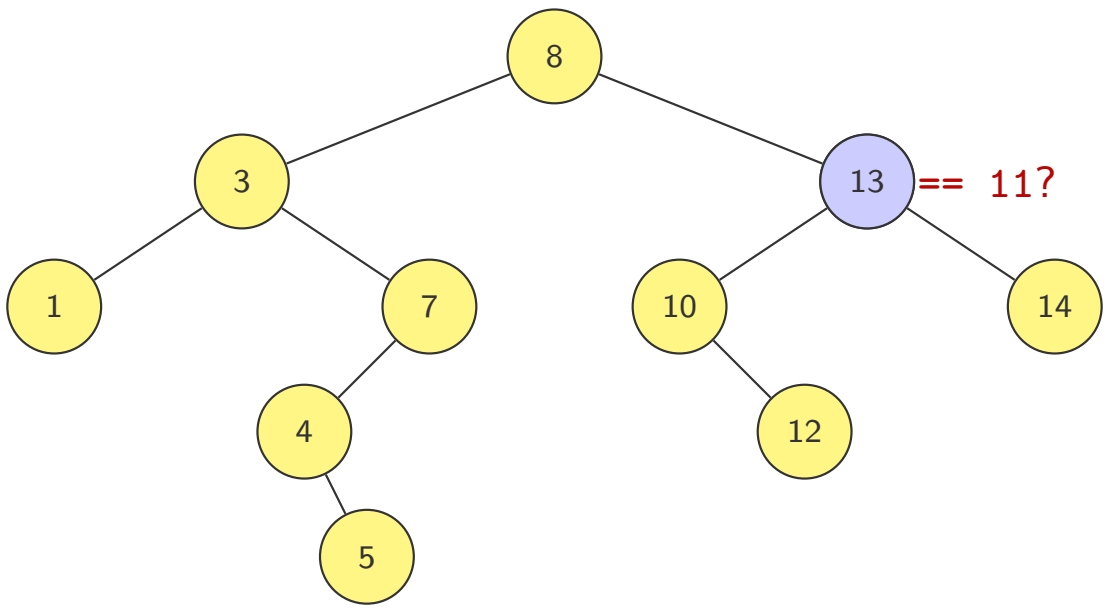


Busca por um valor

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

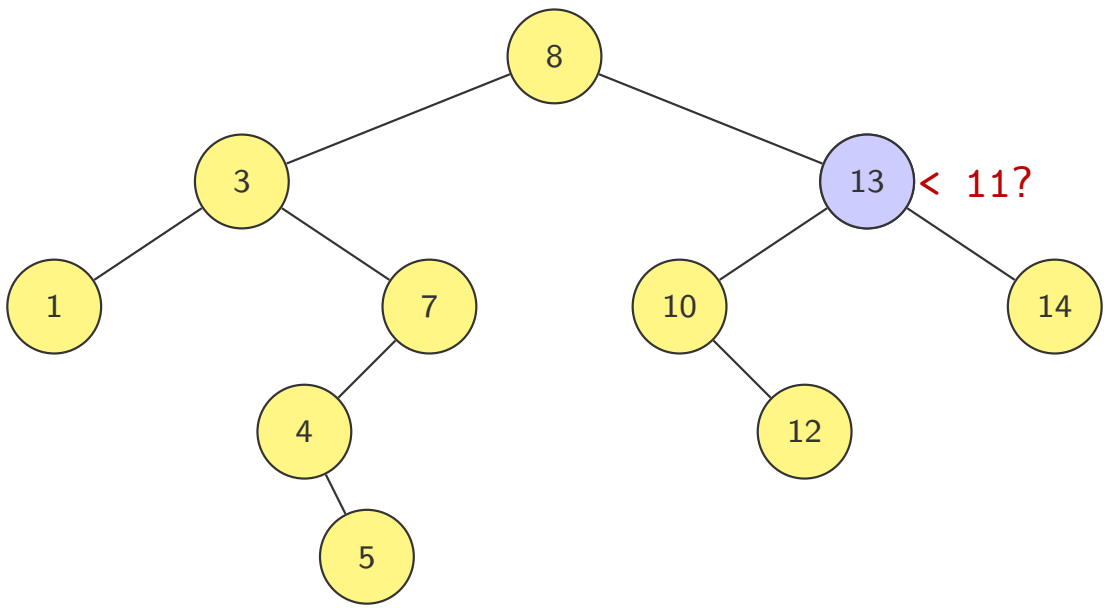


Busca por um valor

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

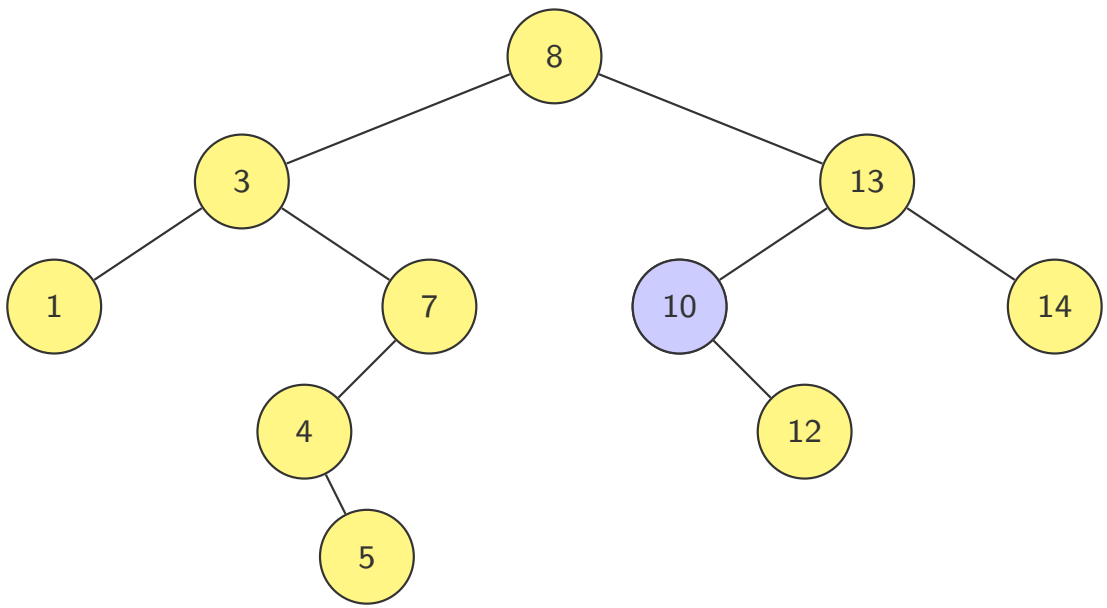


Busca por um valor

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

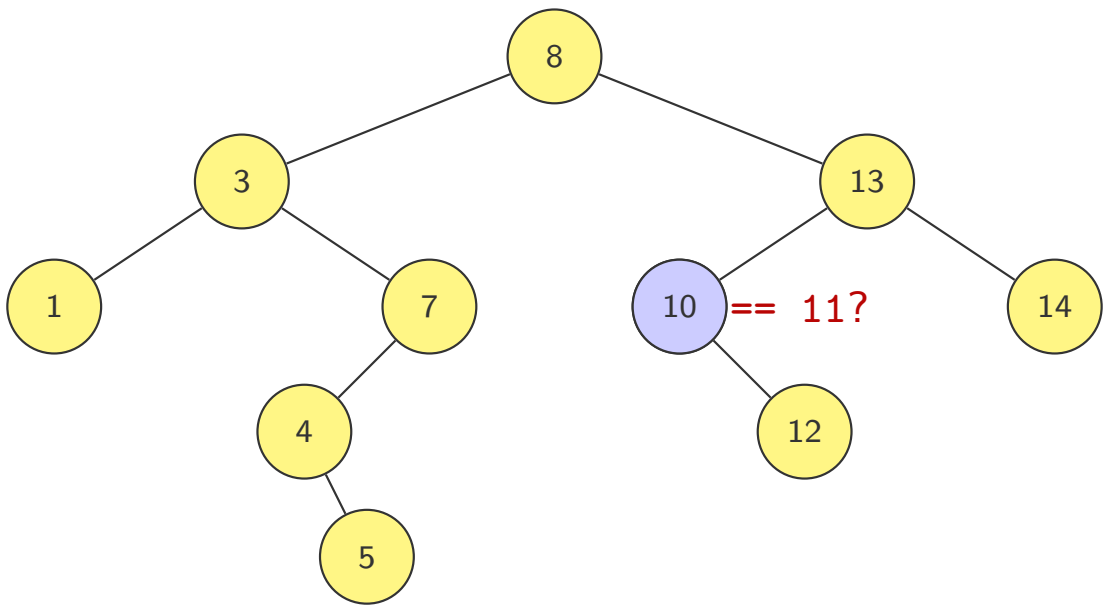


Busca por um valor

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

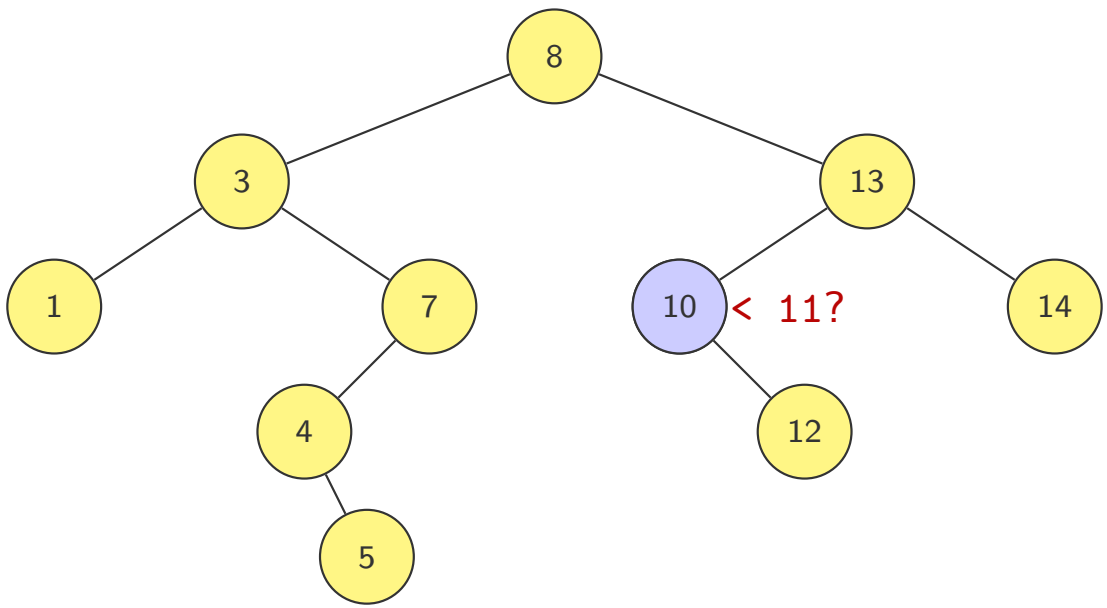


Busca por um valor

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

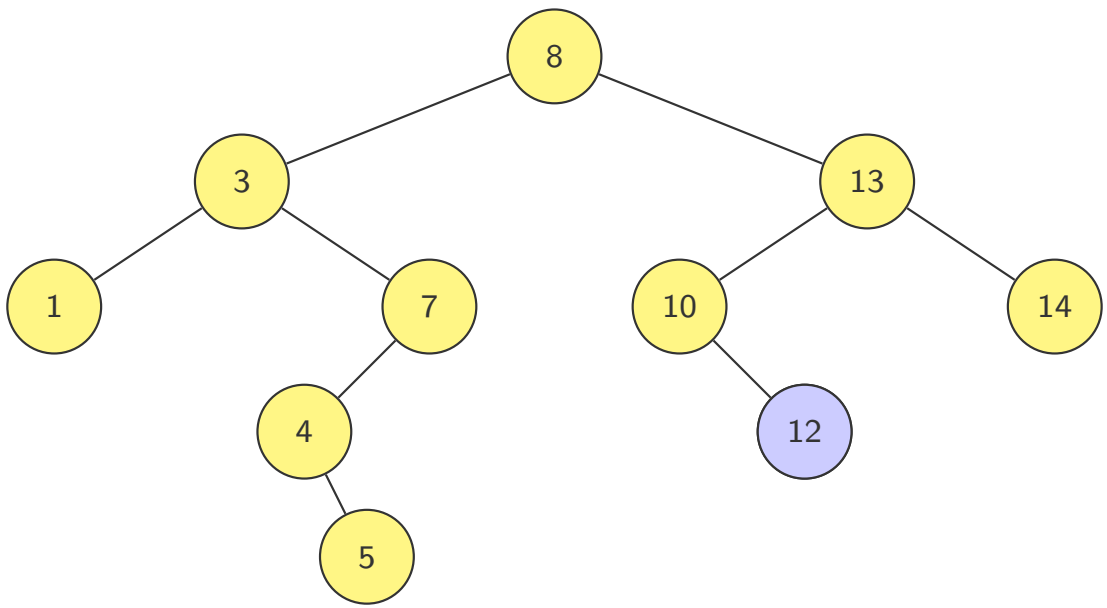


Busca por um valor

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

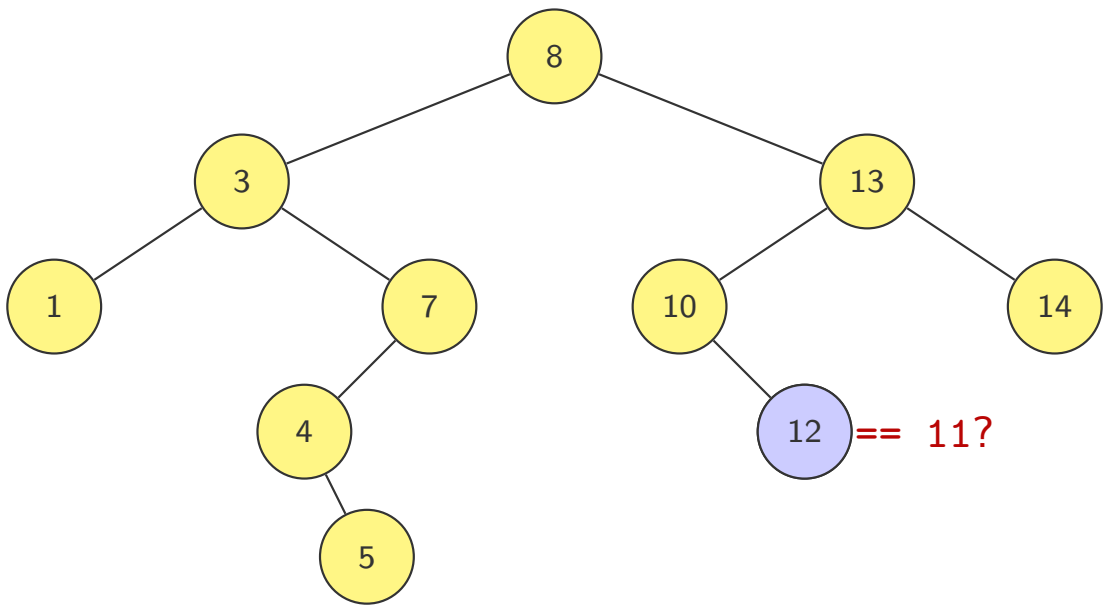


Busca por um valor

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

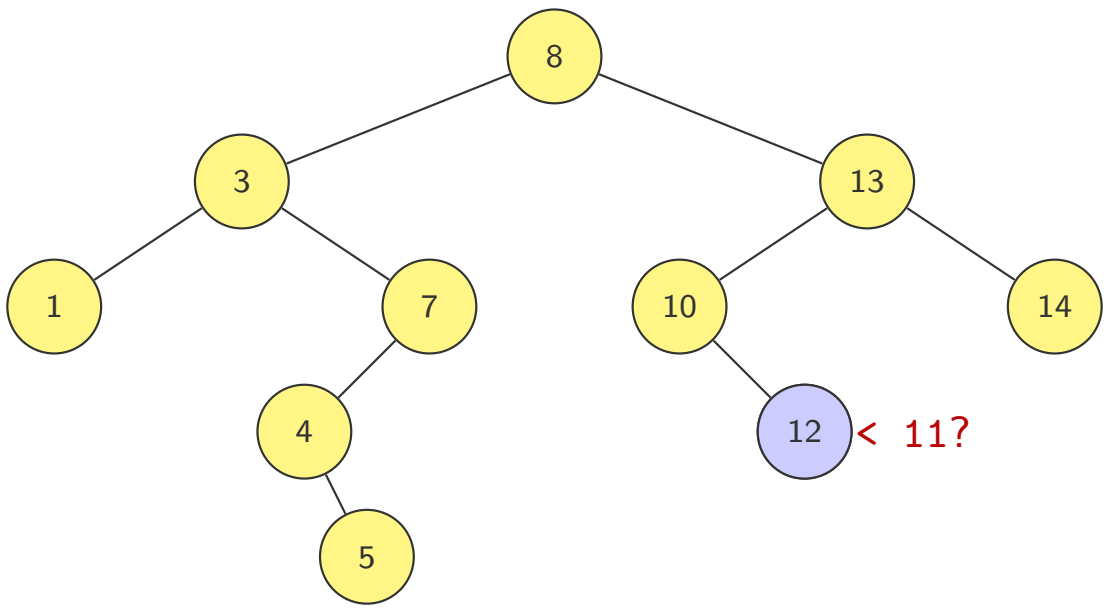


Busca por um valor

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

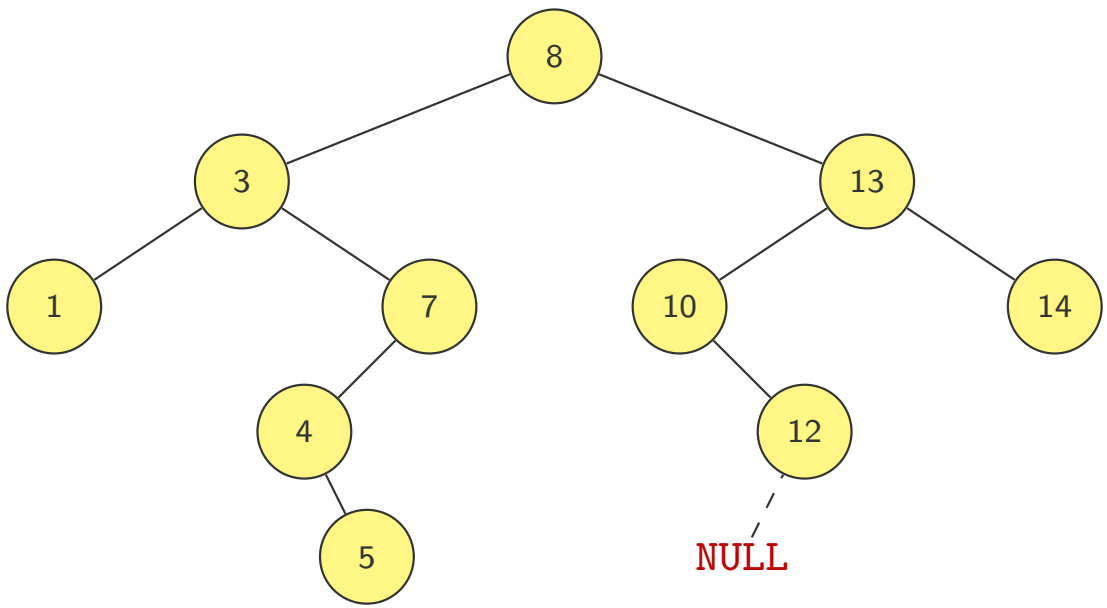


Busca por um valor

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



Busca

Versão recursiva:

```
1 // Recebe a raiz da arvore e uma chave k e retorna
2 // um ponteiro para o no contendo a chave k caso ela
3 // exista na arvore, ou retorna nullptr caso contrario.
4 NoArv* abb_busca(NoArv *raiz, int k) {
```

Busca

Versão recursiva:

```
1  // Recebe a raiz da arvore e uma chave k e retorna
2  // um ponteiro para o no contendo a chave k caso ela
3  // exista na arvore, ou retorna nullptr caso contrario.
4  NoArv* abb_busca(NoArv *raiz, int k) {
5      if(raiz == nullptr || raiz->chave == k)
6          return raiz;
7      if(k > raiz->chave)
8          return abb_busca(raiz->dir, k);
9      else
10         return abb_busca(raiz->esq, k);
11 }
```

Busca

Versão recursiva:

```
1  // Recebe a raiz da arvore e uma chave k e retorna
2  // um ponteiro para o no contendo a chave k caso ela
3  // exista na arvore, ou retorna nullptr caso contrario.
4  NoArv* abb_busca(NoArv *raiz, int k) {
5      if(raiz == nullptr || raiz->chave == k)
6          return raiz;
7      if(k > raiz->chave)
8          return abb_busca(raiz->dir, k);
9      else
10         return abb_busca(raiz->esq, k);
11 }
```

Versão iterativa:

```
1  NoArv* buscar_iterativo(NoArv* raiz, int chave) {
```

Busca

Versão recursiva:

```
1  // Recebe a raiz da arvore e uma chave k e retorna
2  // um ponteiro para o no contendo a chave k caso ela
3  // exista na arvore, ou retorna nullptr caso contrario.
4  NoArv* abb_busca(NoArv *raiz, int k) {
5      if(raiz == nullptr || raiz->chave == k)
6          return raiz;
7      if(k > raiz->chave)
8          return abb_busca(raiz->dir, k);
9      else
10         return abb_busca(raiz->esq, k);
11 }
```

Versão iterativa:

```
1  NoArv* buscar_iterativo(NoArv* raiz, int chave) {
2      while (raiz != nullptr && chave != raiz->chave) {
3          if (chave < raiz->chave)
4              raiz = raiz->esq;
5          else
6              raiz = raiz->dir;
7      }
8      return raiz;
9  }
```

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...

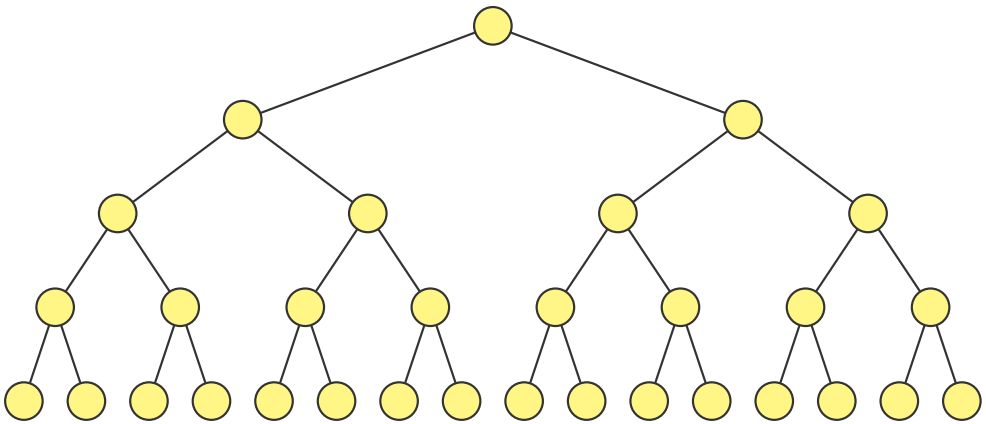
Ex: 31 nós

Eficiência da busca

Qual é o tempo da busca?

- depende da forma da árvore...

Ex: 31 nós



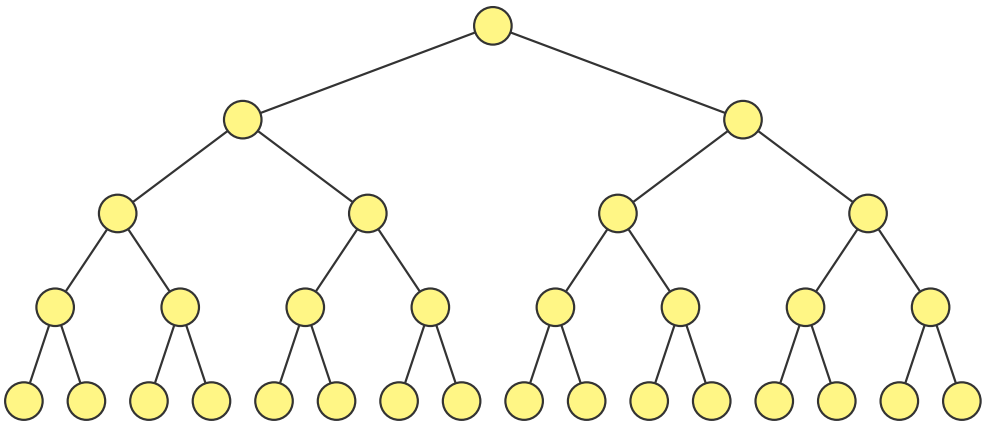
Melhor árvore: $O(\lg n)$

Eficiência da busca

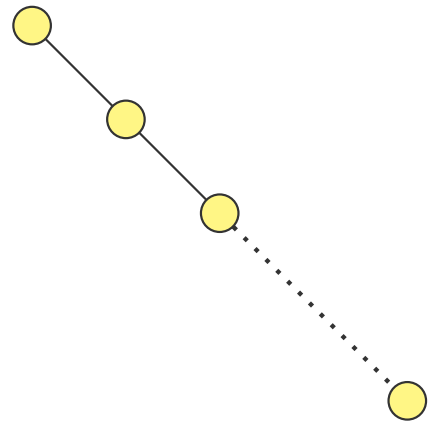
Qual é o tempo da busca?

- depende da forma da árvore...

Ex: 31 nós



Melhor árvore: $O(\lg n)$



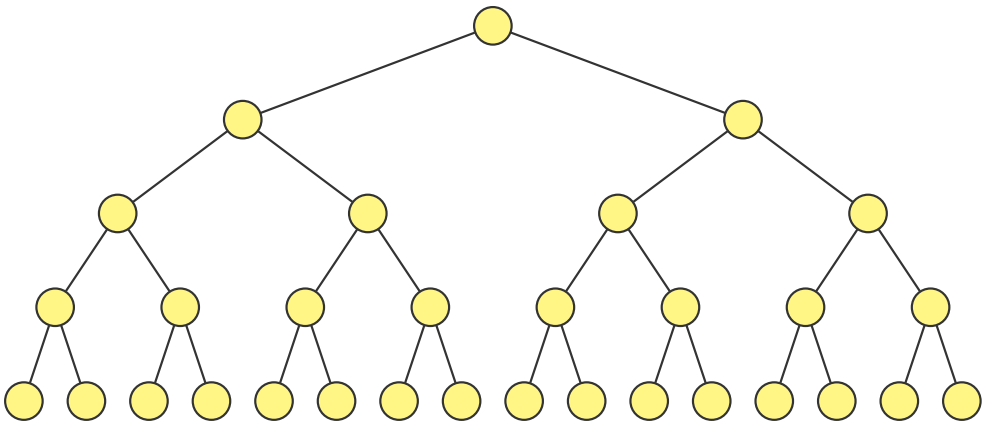
Pior árvore: $O(n)$

Eficiência da busca

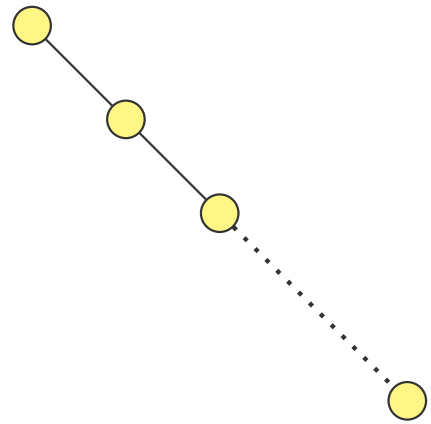
Qual é o tempo da busca?

- depende da forma da árvore...

Ex: 31 nós



Melhor árvore: $O(\lg n)$



Pior árvore: $O(n)$

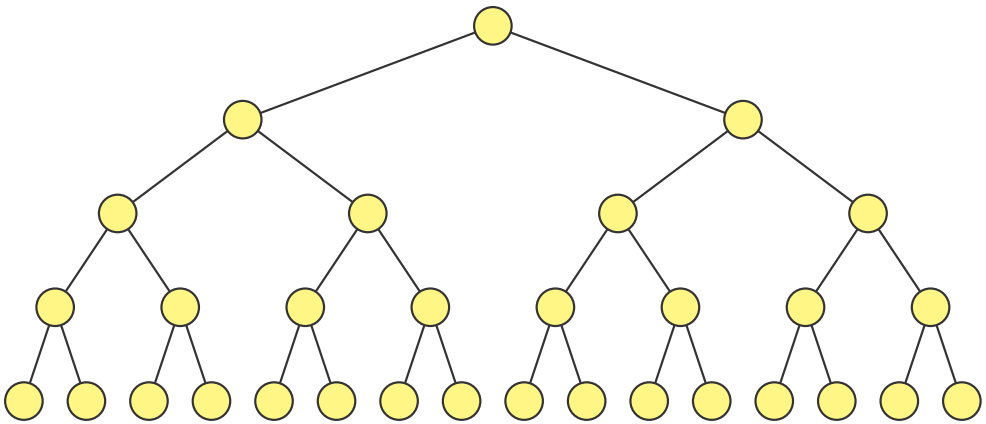
Para ter a pior árvore basta inserir em ordem crescente...

Eficiência da busca

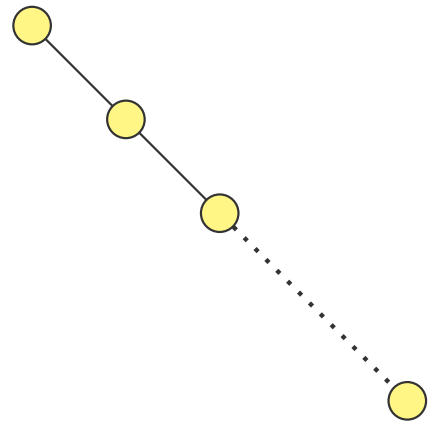
Qual é o tempo da busca?

- 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) $O(\lg n)$

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

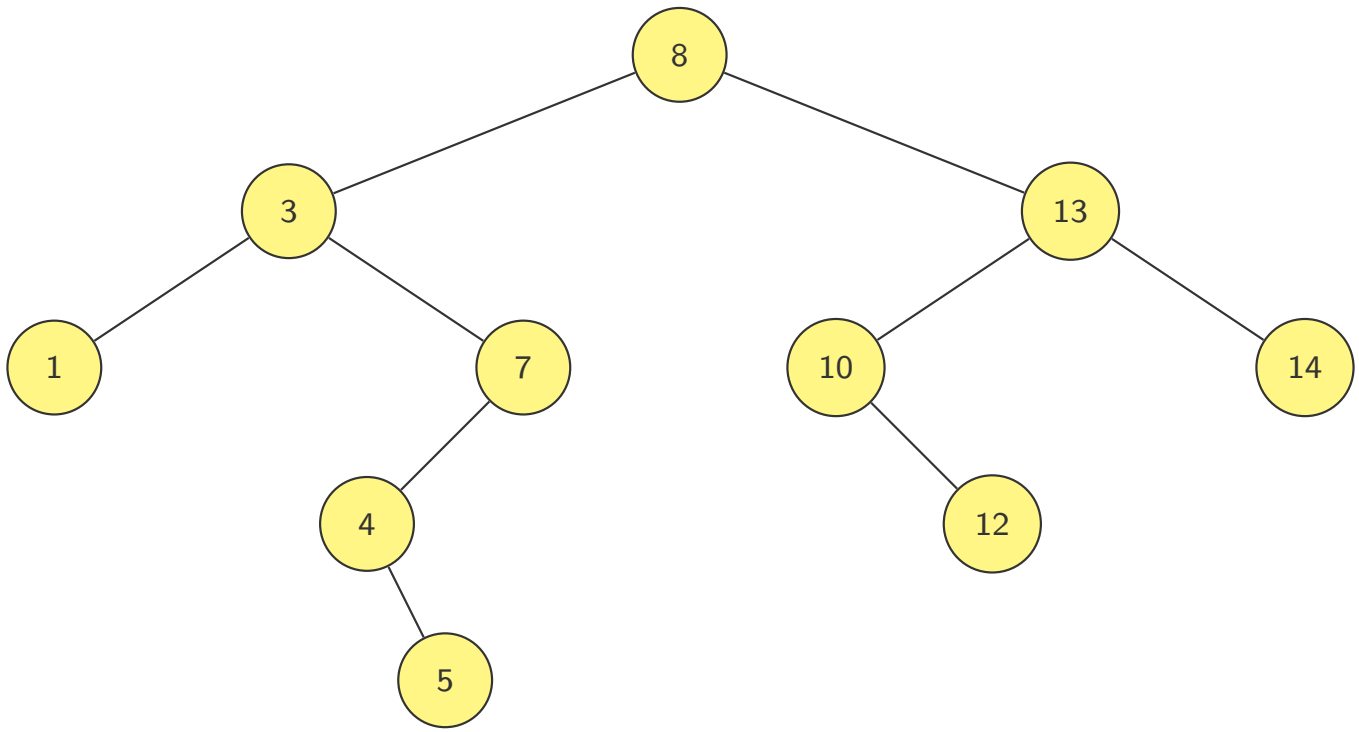
Ex: Inserindo 11

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

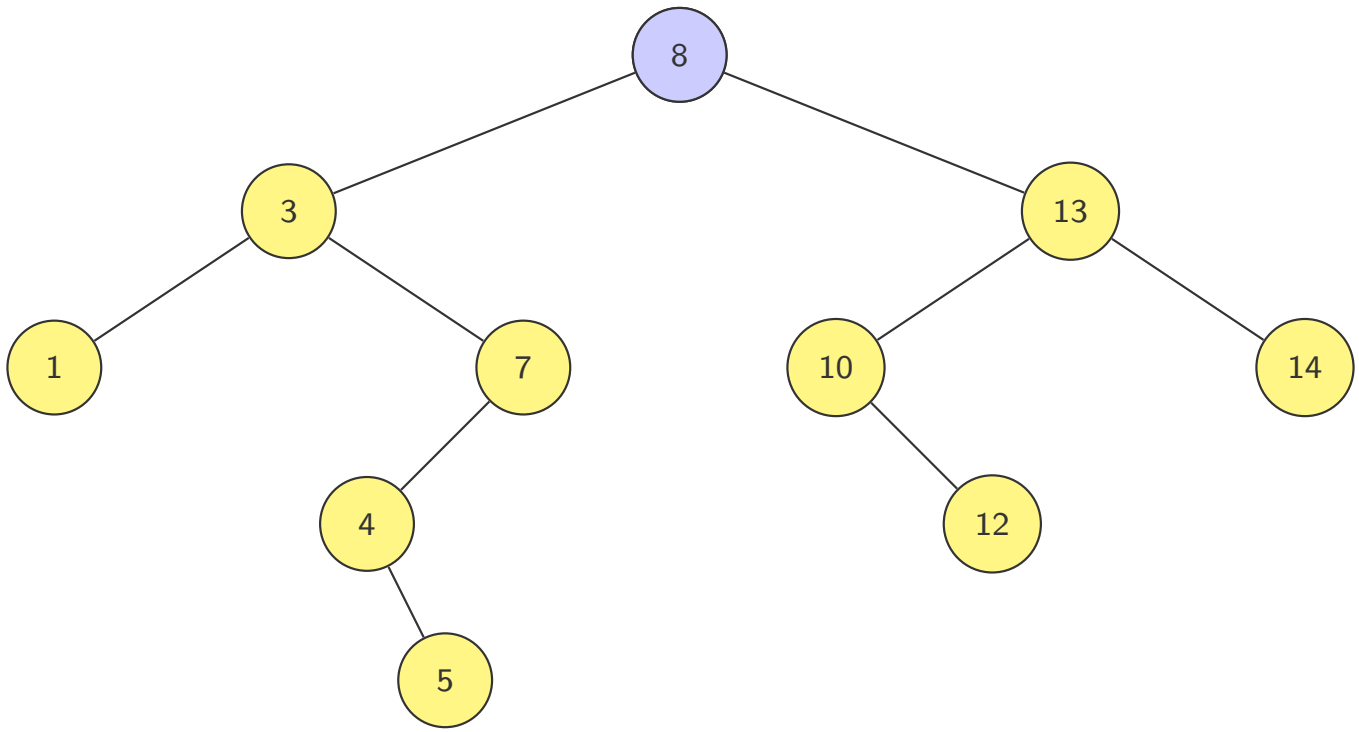


Inserindo um valor

Precisamos determinar onde inserir o valor:

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

Ex: Inserindo 11

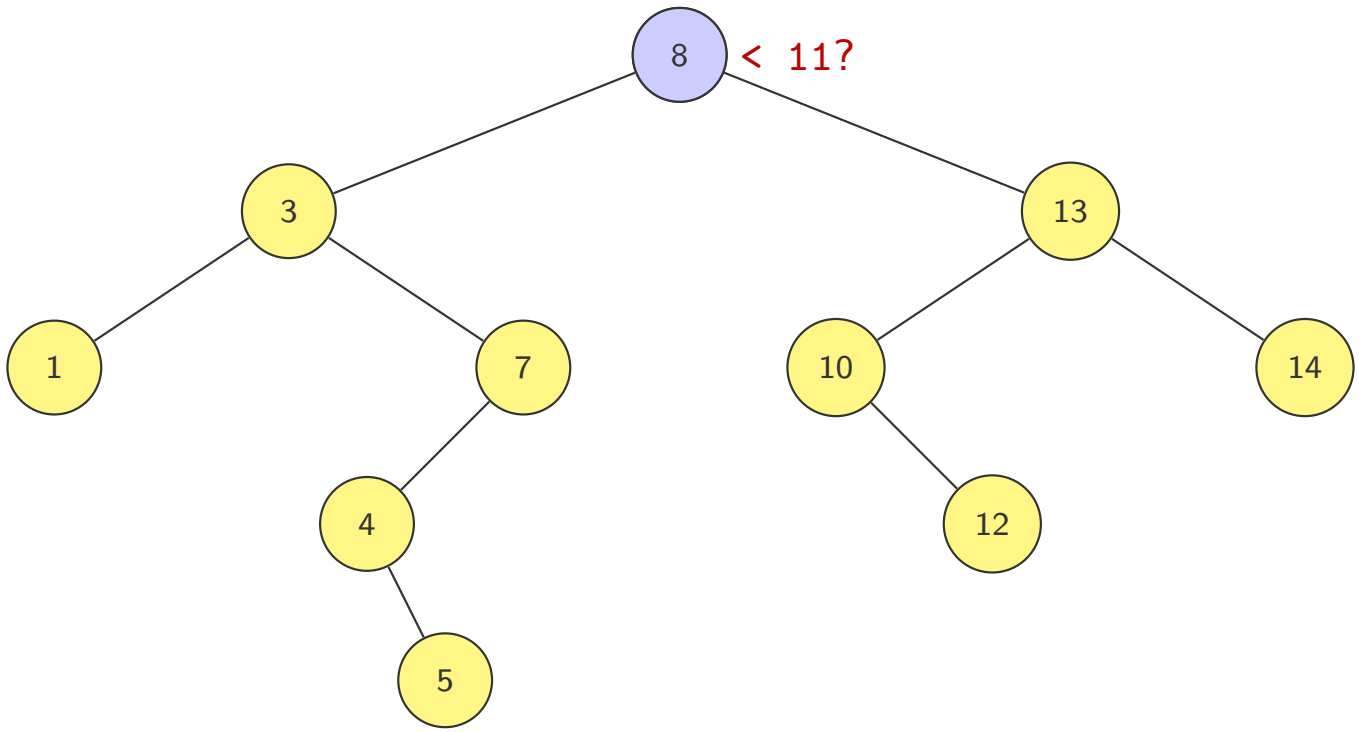


Inserindo um valor

Precisamos determinar onde inserir o valor:

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

Ex: Inserindo 11

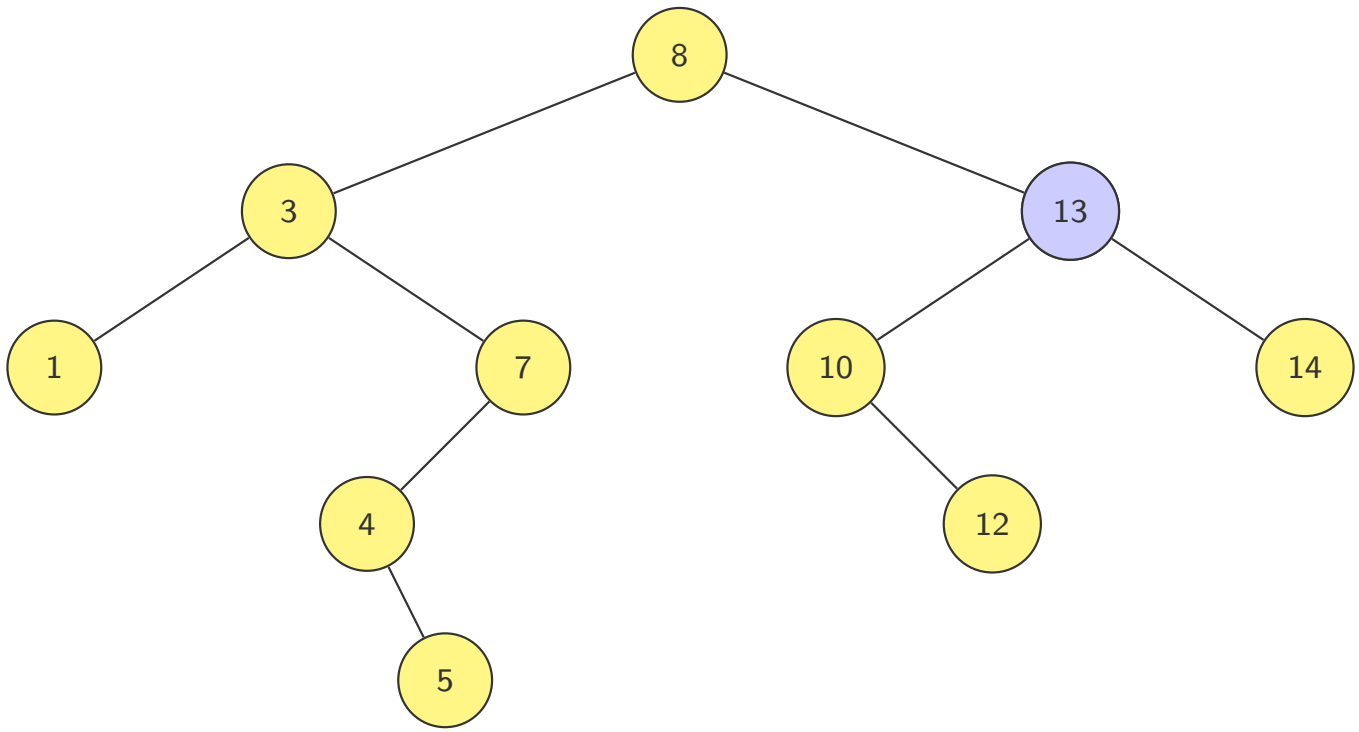


Inserindo um valor

Precisamos determinar onde inserir o valor:

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

Ex: Inserindo 11

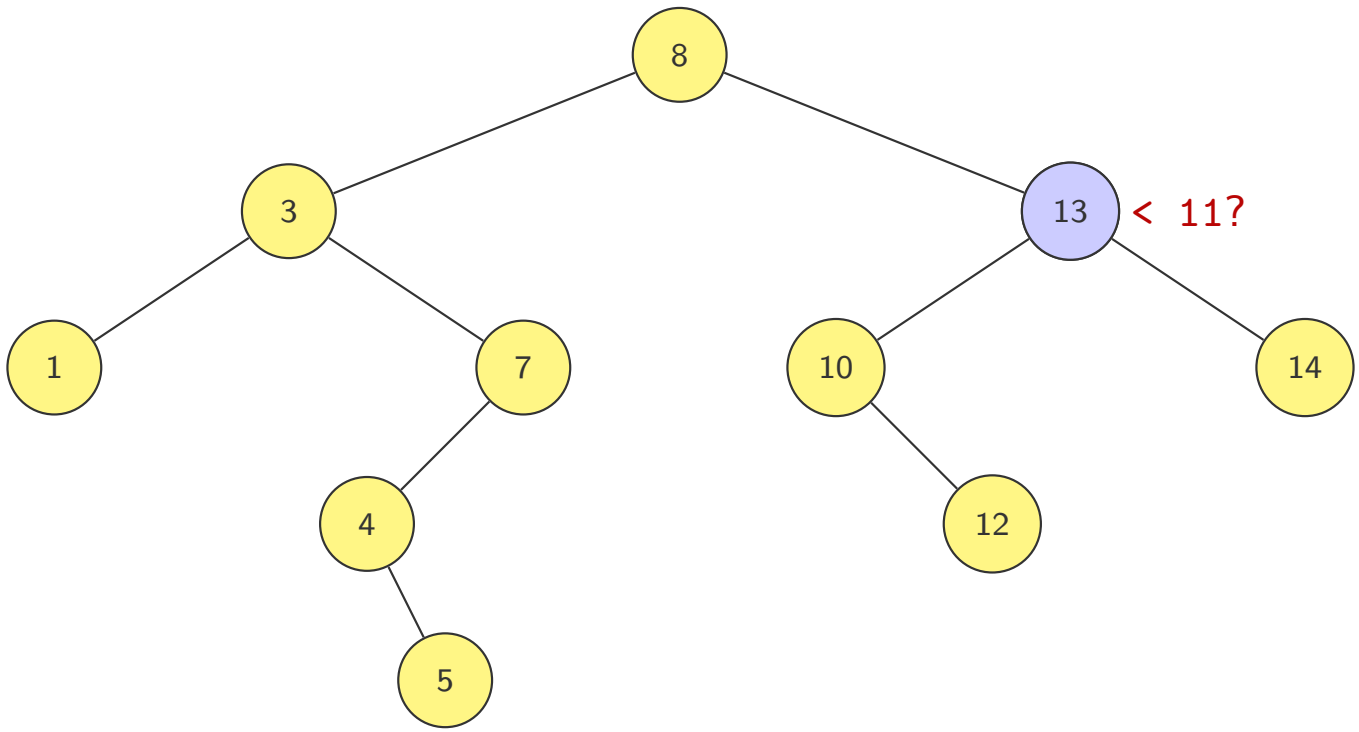


Inserindo um valor

Precisamos determinar onde inserir o valor:

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

Ex: Inserindo 11

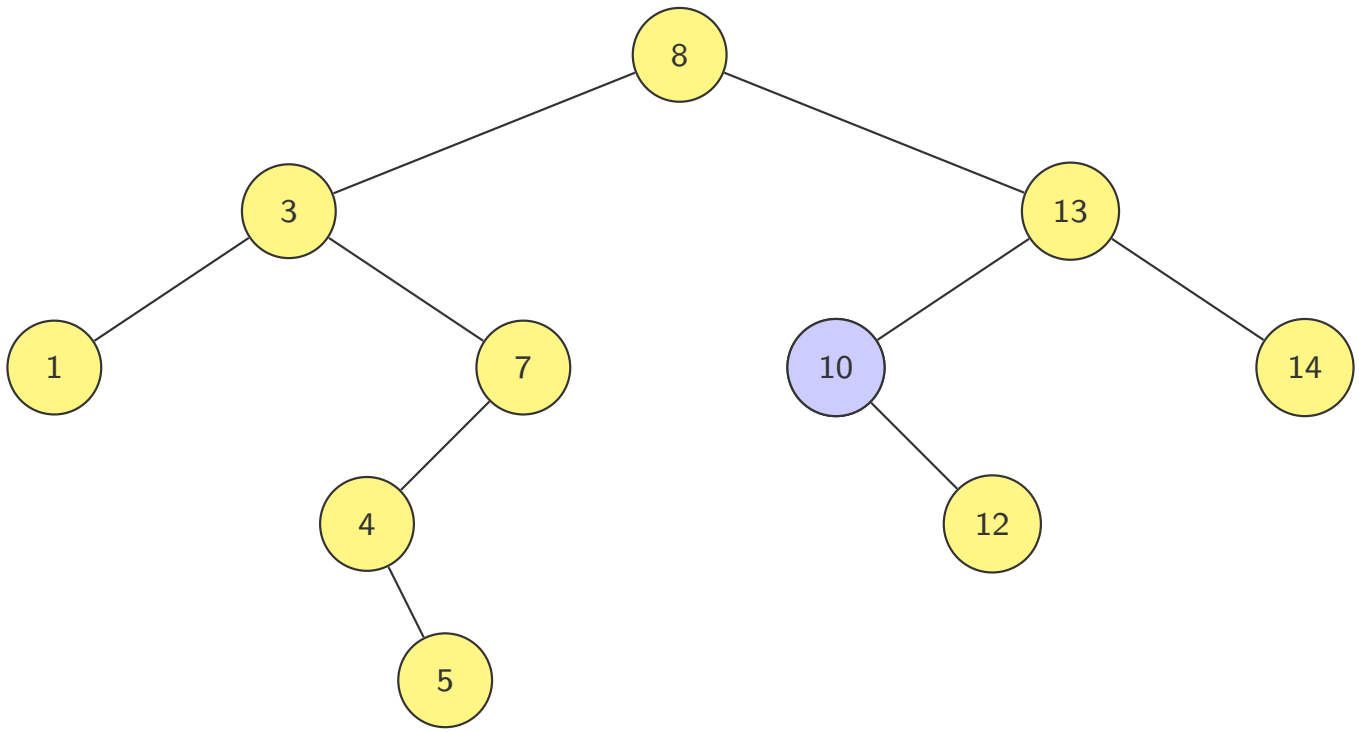


Inserindo um valor

Precisamos determinar onde inserir o valor:

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

Ex: Inserindo 11

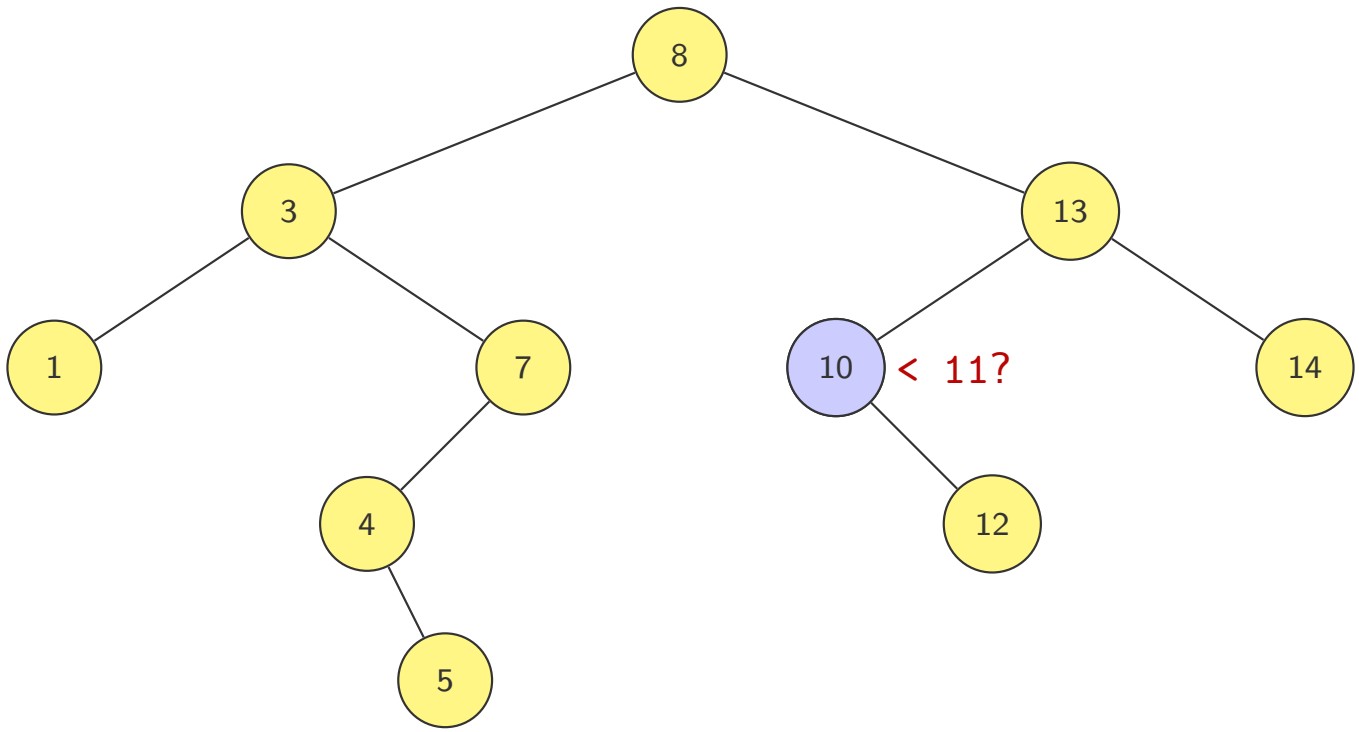


Inserindo um valor

Precisamos determinar onde inserir o valor:

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

Ex: Inserindo 11

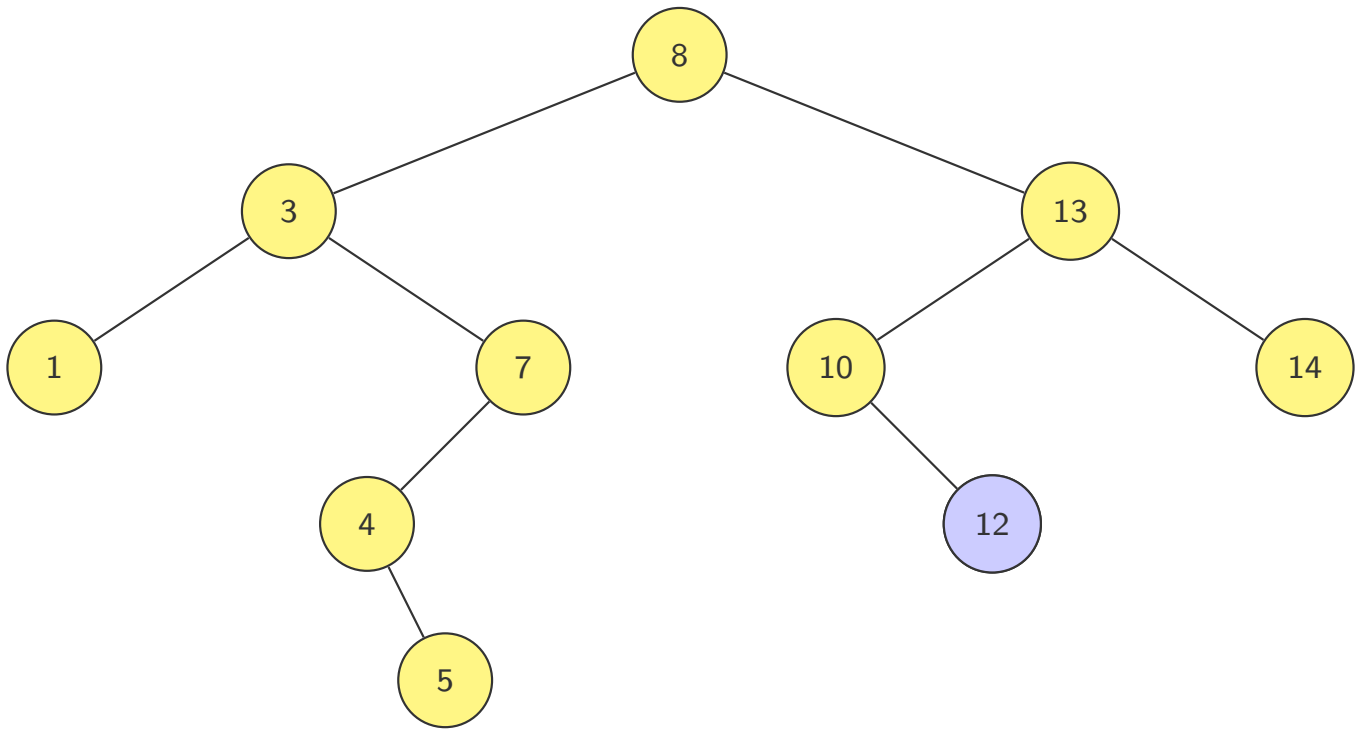


Inserindo um valor

Precisamos determinar onde inserir o valor:

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

Ex: Inserindo 11

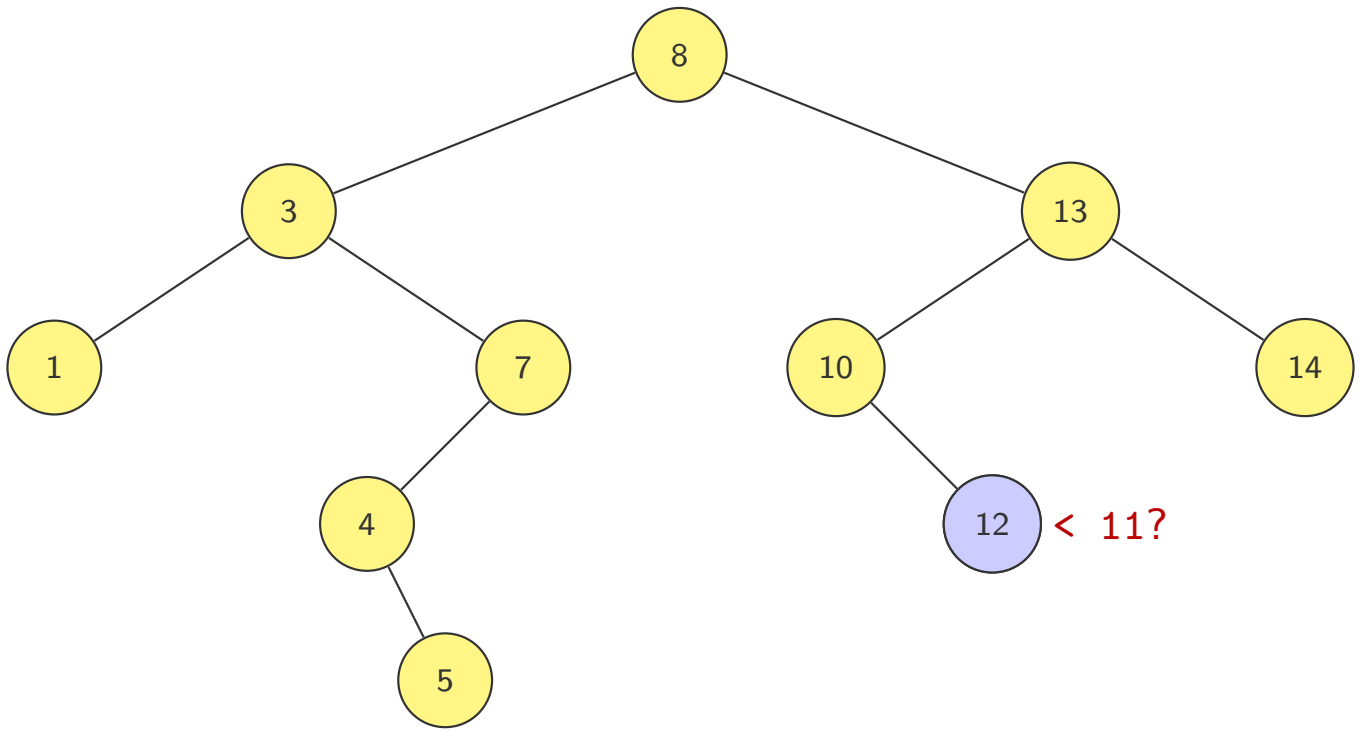


Inserindo um valor

Precisamos determinar onde inserir o valor:

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

Ex: Inserindo 11

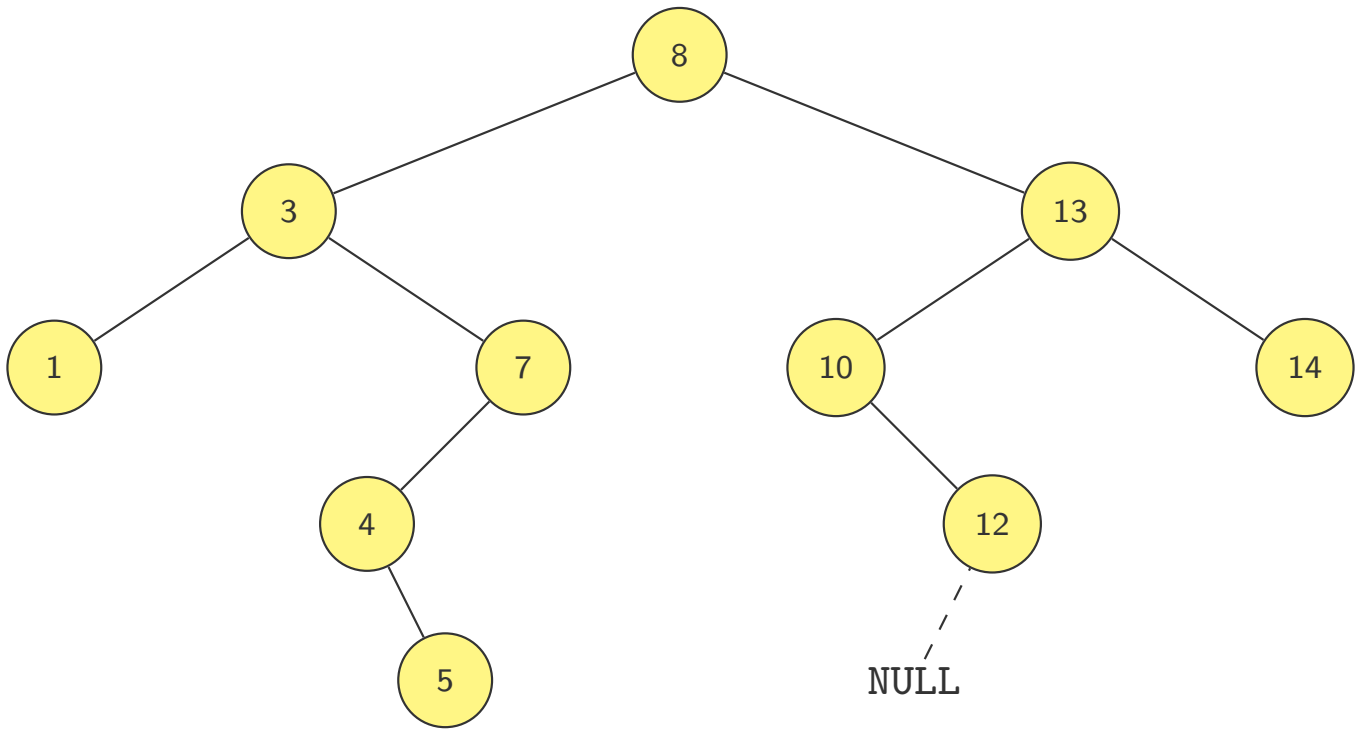


Inserindo um valor

Precisamos determinar onde inserir o valor:

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

Ex: Inserindo 11

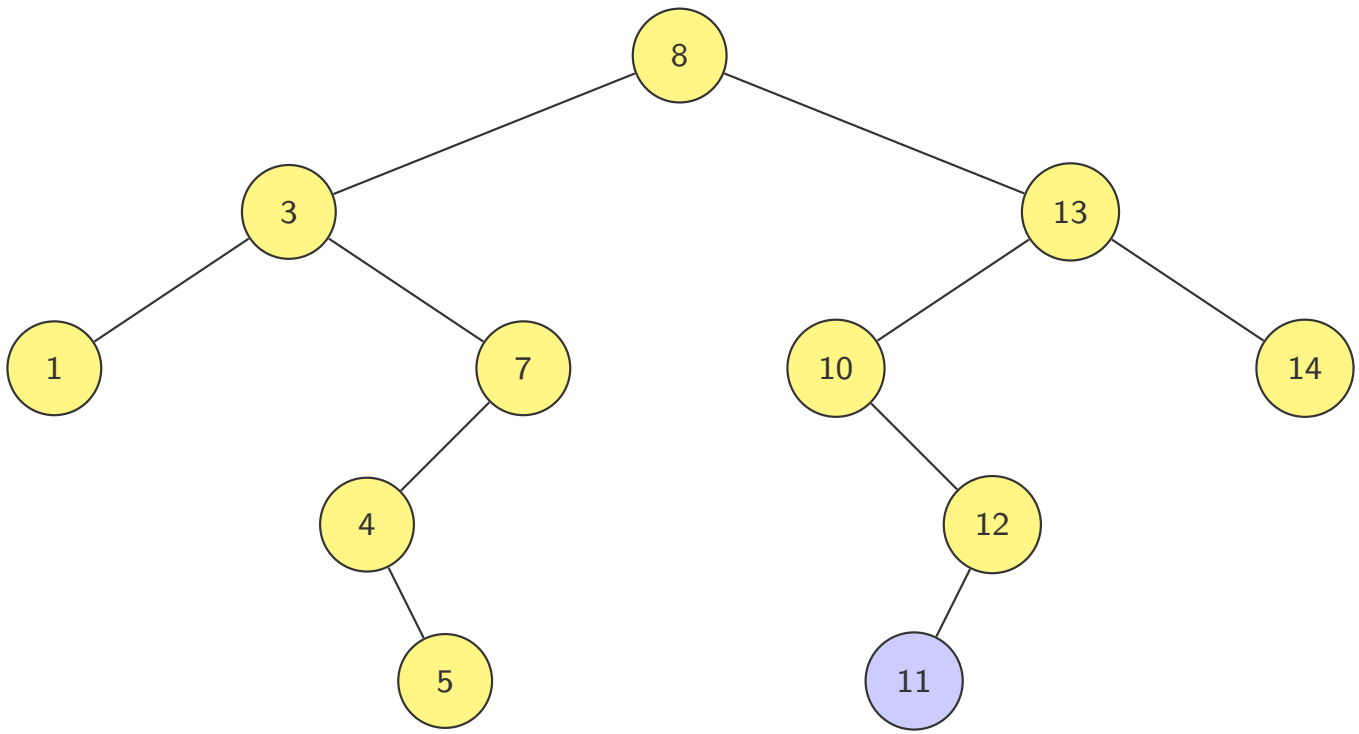


Inserindo um valor

Precisamos determinar onde inserir o valor:

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

Ex: Inserindo 11



Inserção — Implementação

O algoritmo insere na árvore recursivamente

Inserção — Implementação

O algoritmo insere na árvore recursivamente

- devolve um ponteiro para a raiz da “nova” árvore

Inserção — Implementação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

O algoritmo insere na árvore recursivamente

- devolve um ponteiro para a raiz da "nova" árvore

```
1 // Inserir no com chave k na arvore
2 NoArv* abb_inserir(NoArv *raiz, int k) {
```

Inserção — Implementação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

O algoritmo insere na árvore recursivamente

- devolve um ponteiro para a raiz da "nova" árvore

```
1 // Inserir no com chave k na arvore
2 NoArv* abb_inserir(NoArv *raiz, int k) {
3     if(raiz == nullptr) { // Caso base
4         raiz = new NoArv;
5         raiz->chave = k;
6         raiz->esq = raiz->dir = nullptr;
7     }
```

Inserção — Implementação

O algoritmo insere na árvore recursivamente

- devolve um ponteiro para a raiz da "nova" árvore

```
1  // Inserir no com chave k na arvore
2  NoArv* abb_inserir(NoArv *raiz, int k) {
3      if(raiz == nullptr) { // Caso base
4          raiz = new NoArv;
5          raiz->chave = k;
6          raiz->esq = raiz->dir = nullptr;
7      }
8      else if(k > raiz->chave)
9          raiz->dir = abb_inserir(raiz->dir, k);
10     else if(k < raiz->chave)
11         raiz->esq = abb_inserir(raiz->esq, k);
12
13     return raiz;
14 }
```

Inserção — Implementação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

O algoritmo insere na árvore recursivamente

- devolve um ponteiro para a raiz da "nova" árvore

```
1  // Inserir no com chave k na arvore
2  NoArv* abb_inserir(NoArv *raiz, int k) {
3      if(raiz == nullptr) { // Caso base
4          raiz = new NoArv;
5          raiz->chave = k;
6          raiz->esq = raiz->dir = nullptr;
7      }
8      else if(k > raiz->chave)
9          raiz->dir = abb_inserir(raiz->dir, k);
10     else if(k < raiz->chave)
11         raiz->esq = abb_inserir(raiz->esq, k);
12
13     return raiz;
14 }
```

- Exercício: Escreva a versão iterativa da inserção.

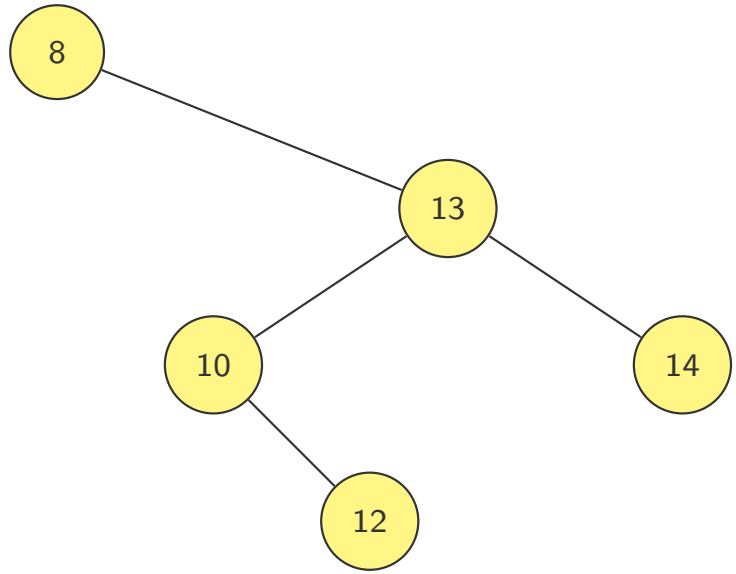
Mínimo da Árvore Binária de Busca

Onde está o nó com a menor chave de uma árvore?

Mínimo da Árvore Binária de Busca

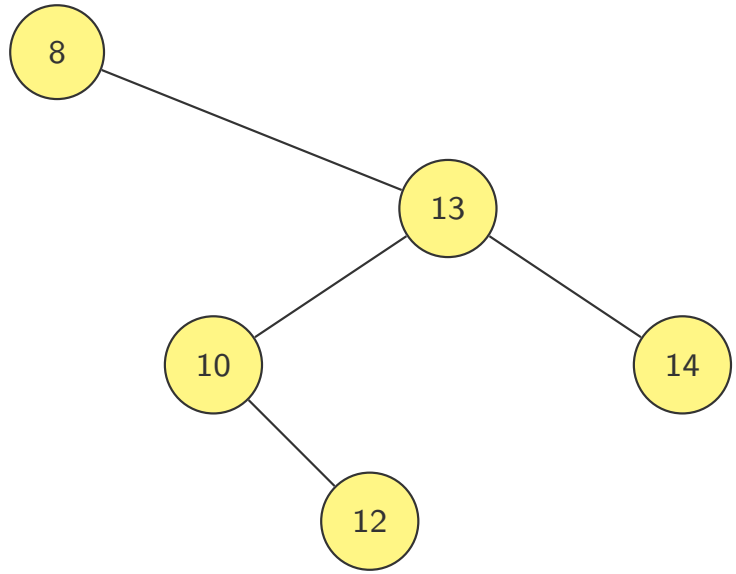
Universidade Federal do Ceará
Campus Quixadá

Onde está o nó com a menor chave de uma árvore?



Mínimo da Árvore Binária de Busca

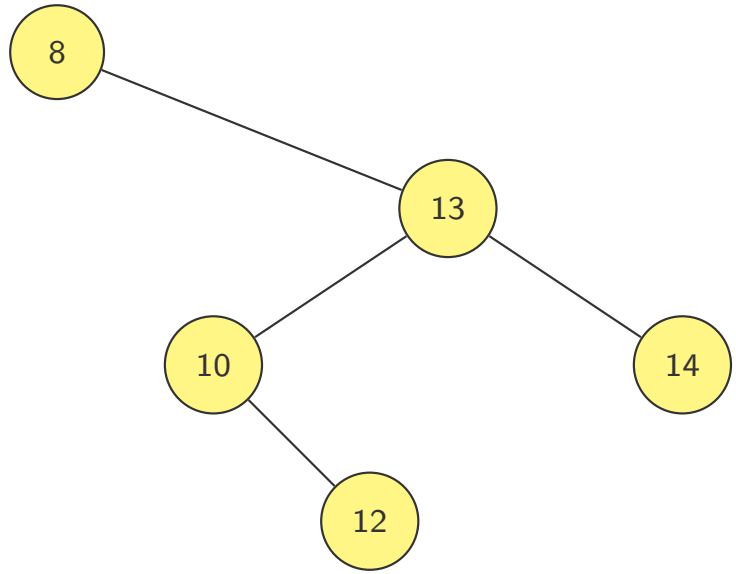
Onde está o nó com a menor chave de uma árvore?



Quem é o mínimo para essa árvore?

Mínimo da Árvore Binária de Busca

Onde está o nó com a menor chave de uma árvore?



Quem é o mínimo para essa árvore?

- É a própria raiz

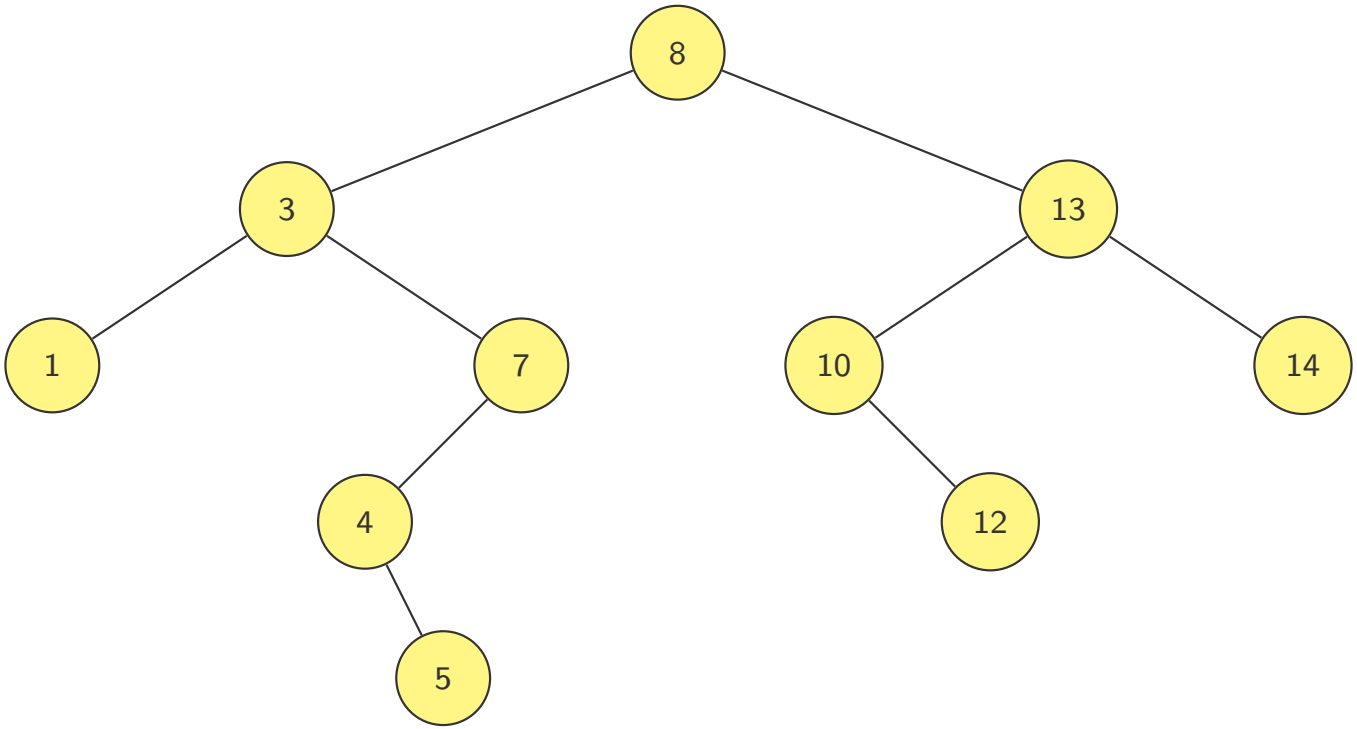
Mínimo da Árvore Binária de Busca

Onde está o nó com a menor chave de uma árvore?

Mínimo da Árvore Binária de Busca

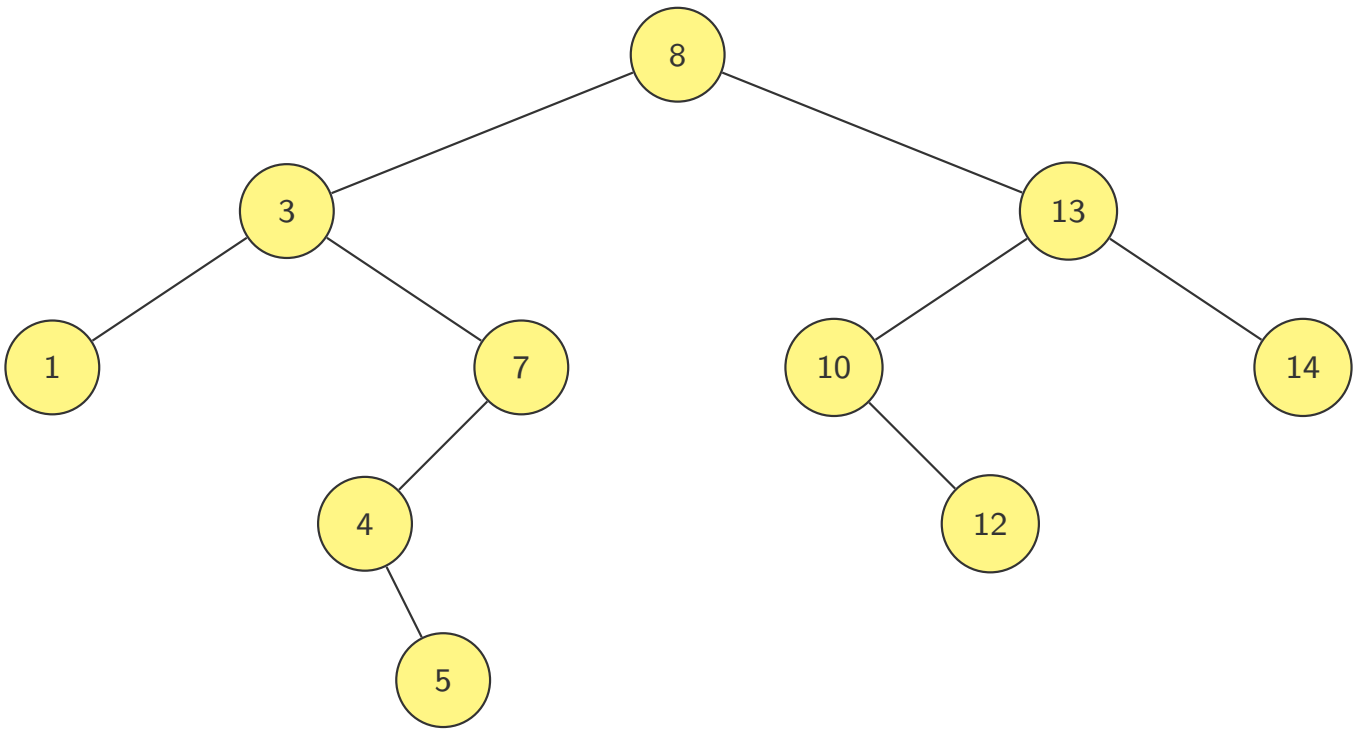
Universidade Federal do Ceará
Campus Quixadá

Onde está o nó com a menor chave de uma árvore?



Mínimo da Árvore Binária de Busca

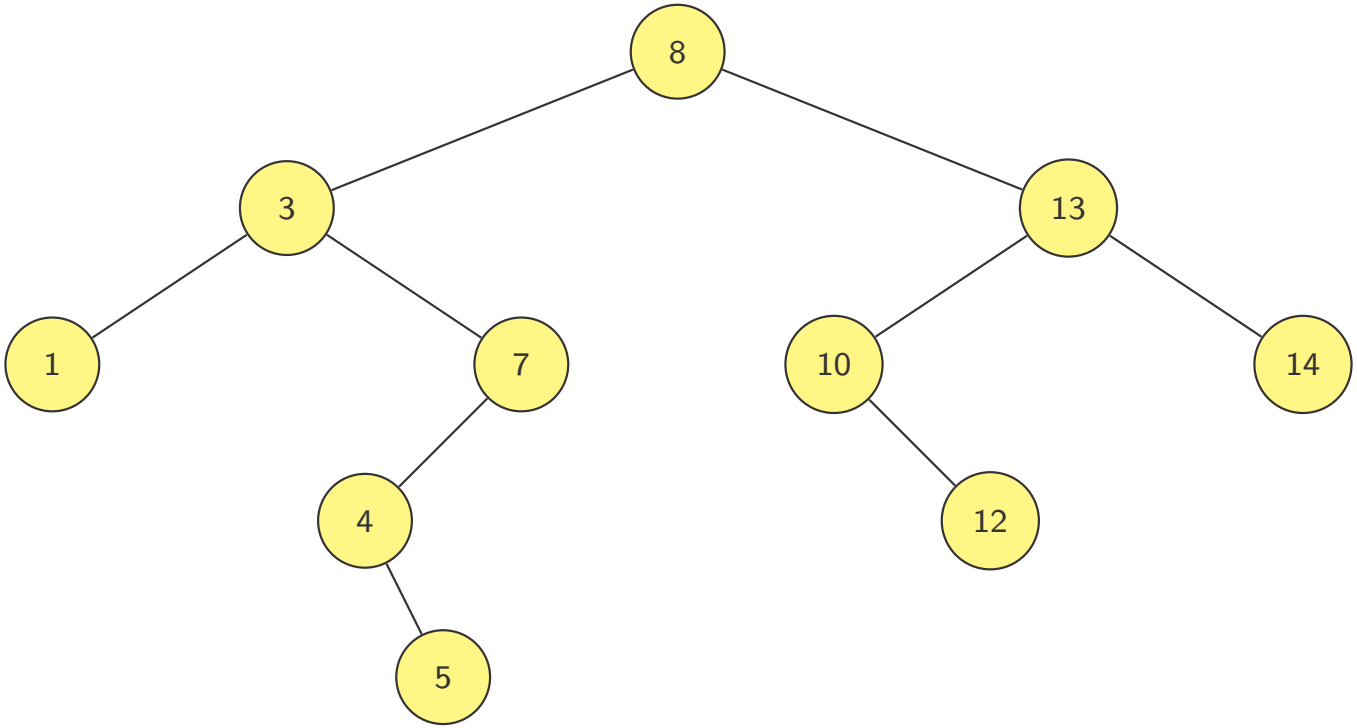
Onde está o nó com a menor chave de uma árvore?



Quem é o mínimo para essa árvore?

Mínimo da Árvore Binária de Busca

Onde está o nó com a menor chave de uma árvore?



Quem é o mínimo para essa árvore?

- É o mínimo da subárvore esquerda

Mínimo - Implementações

Versão recursiva:

Mínimo - Implementações

Universidade Federal do Ceará
Campus Quixadá

Versão recursiva:

```
1 // Retorna ponteiro para o no com menor chave
2 // ou nullptr caso a arvore seja vazia
3 NoArv* abb_minimo(NoArv *raiz){
```

Mínimo - Implementações

Universidade Federal do Ceará
Campus Quixadá

Versão recursiva:

```
1 // Retorna ponteiro para o no com menor chave
2 // ou nullptr caso a arvore seja vazia
3 NoArv* abb_minimo(NoArv *raiz){
```

Mínimo - Implementações

Versão recursiva:

```
1 // Retorna ponteiro para o no com menor chave
2 // ou nullptr caso a arvore seja vazia
3 NoArv* abb_minimo(NoArv *raiz){
4     if(raiz != nullptr && raiz->esq != nullptr)
5         return abb_minimo(raiz->esq);
6     else
7         return raiz;
8 }
```

Mínimo - Implementações

Versão recursiva:

```
1 // Retorna ponteiro para o no com menor chave
2 // ou nullptr caso a arvore seja vazia
3 NoArv* abb_minimo(NoArv *raiz){
4     if(raiz != nullptr && raiz->esq != nullptr)
5         return abb_minimo(raiz->esq);
6     else
7         return raiz;
8 }
```

Versão iterativa:

Mínimo - Implementações

Versão recursiva:

```
1 // Retorna ponteiro para o no com menor chave
2 // ou nullptr caso a arvore seja vazia
3 NoArv* abb_minimo(NoArv *raiz){
4     if(raiz != nullptr && raiz->esq != nullptr)
5         return abb_minimo(raiz->esq);
6     else
7         return raiz;
8 }
```

Versão iterativa:

```
1 // Achar o no com chave minima na arvore (iterativo)
2 NoArv* abb_minimo_iterativo(NoArv* raiz) {
3     while (raiz != nullptr && raiz->esq != nullptr)
4         raiz = raiz->esq;
5     return raiz;
6 }
```

Mínimo - Implementações

Versão recursiva:

```
1 // Retorna ponteiro para o no com menor chave
2 // ou nullptr caso a arvore seja vazia
3 NoArv* abb_minimo(NoArv *raiz){
4     if(raiz != nullptr && raiz->esq != nullptr)
5         return abb_minimo(raiz->esq);
6     else
7         return raiz;
8 }
```

Versão iterativa:

```
1 // Achar o no com chave minima na arvore (iterativo)
2 NoArv* abb_minimo_iterativo(NoArv* raiz) {
3     while (raiz != nullptr && raiz->esq != nullptr)
4         raiz = raiz->esq;
5     return raiz;
6 }
```

Para encontrar o máximo, basta fazer a operação simétrica

Mínimo - Implementações

Versão recursiva:

```
1 // Retorna ponteiro para o no com menor chave
2 // ou nullptr caso a arvore seja vazia
3 NoArv* abb_minimo(NoArv *raiz){
4     if(raiz != nullptr && raiz->esq != nullptr)
5         return abb_minimo(raiz->esq);
6     else
7         return raiz;
8 }
```

Versão iterativa:

```
1 // Achar o no com chave minima na arvore (iterativo)
2 NoArv* abb_minimo_iterativo(NoArv* raiz) {
3     while (raiz != nullptr && raiz->esq != nullptr)
4         raiz = raiz->esq;
5     return raiz;
6 }
```

Para encontrar o máximo, basta fazer a operação simétrica

- Se a subárvore direita existir, é o seu máximo

Mínimo - Implementações

Versão recursiva:

```
1 // Retorna ponteiro para o no com menor chave
2 // ou nullptr caso a arvore seja vazia
3 NoArv* abb_minimo(NoArv *raiz){
4     if(raiz != nullptr && raiz->esq != nullptr)
5         return abb_minimo(raiz->esq);
6     else
7         return raiz;
8 }
```

Versão iterativa:

```
1 // Achar o no com chave minima na arvore (iterativo)
2 NoArv* abb_minimo_iterativo(NoArv* raiz) {
3     while (raiz != nullptr && raiz->esq != nullptr)
4         raiz = raiz->esq;
5     return raiz;
6 }
```

Para encontrar o máximo, basta fazer a operação simétrica

- Se a subárvore direita existir, é o seu máximo
- Senão, é a própria raiz

Máximo - Implementações

Versão recursiva:

Máximo - Implementações

Versão recursiva:

```
1 // Retorna ponteiro para o no com maior chave
2 // ou nullptr caso a arvore seja vazia
3 NoArv* abb_maximo(NoArv *raiz) {
```

Máximo - Implementações

Versão recursiva:

```
1 // Retorna ponteiro para o no com maior chave
2 // ou nullptr caso a arvore seja vazia
3 NoArv* abb_maximo(NoArv *raiz) {
```

Máximo - Implementações

Versão recursiva:

```
1 // Retorna ponteiro para o no com maior chave
2 // ou nullptr caso a arvore seja vazia
3 NoArv* abb_maximo(NoArv *raiz) {
4     if(raiz != nullptr && raiz->dir != nullptr)
5         return abb_maximo(raiz->dir);
6     else
7         return raiz;
8 }
```

Máximo - Implementações

Versão recursiva:

```
1 // Retorna ponteiro para o no com maior chave
2 // ou nullptr caso a arvore seja vazia
3 NoArv* abb_maximo(NoArv *raiz) {
4     if(raiz != nullptr && raiz->dir != nullptr)
5         return abb_maximo(raiz->dir);
6     else
7         return raiz;
8 }
```

Versão iterativa:

Máximo - Implementações

Versão recursiva:

```
1 // Retorna ponteiro para o no com maior chave
2 // ou nullptr caso a arvore seja vazia
3 NoArv* abb_maximo(NoArv *raiz) {
4     if(raiz != nullptr && raiz->dir != nullptr)
5         return abb_maximo(raiz->dir);
6     else
7         return raiz;
8 }
```

Versão iterativa:

```
1 // Achar o no com chave maxima (iterativo)
2 NoArv* abb_maximo_iterativo (NoArv* raiz) {
3   while (raiz != nullptr && raiz->dir != nullptr)
4     raiz = raiz->dir;
5   return raiz;
6 }
```

Sucessor

Dado um nó da árvore, onde está o seu sucessor?

Sucessor

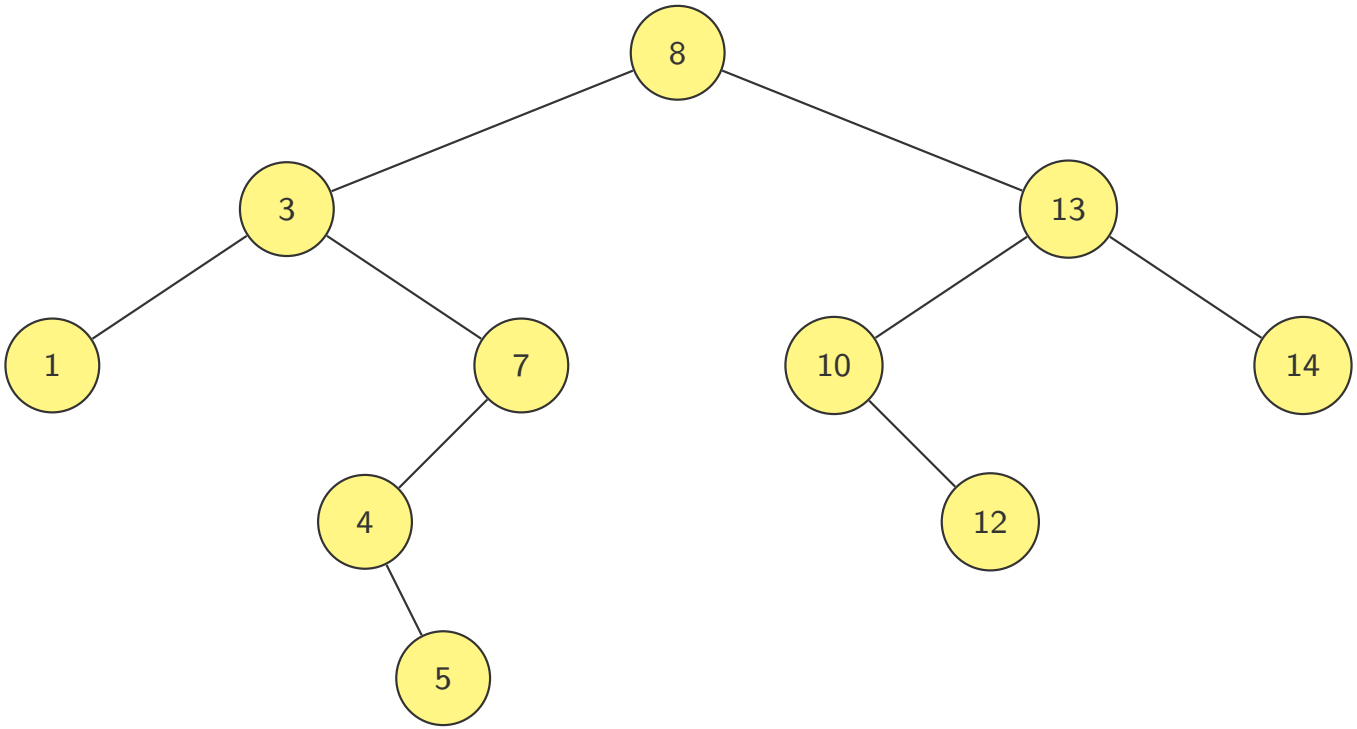
Dado um nó da árvore, onde está o seu sucessor?

- O sucessor é o próximo nó na ordenação

Sucessor

Dado um nó da árvore, onde está o seu sucessor?

- O sucessor é o próximo nó na ordenação

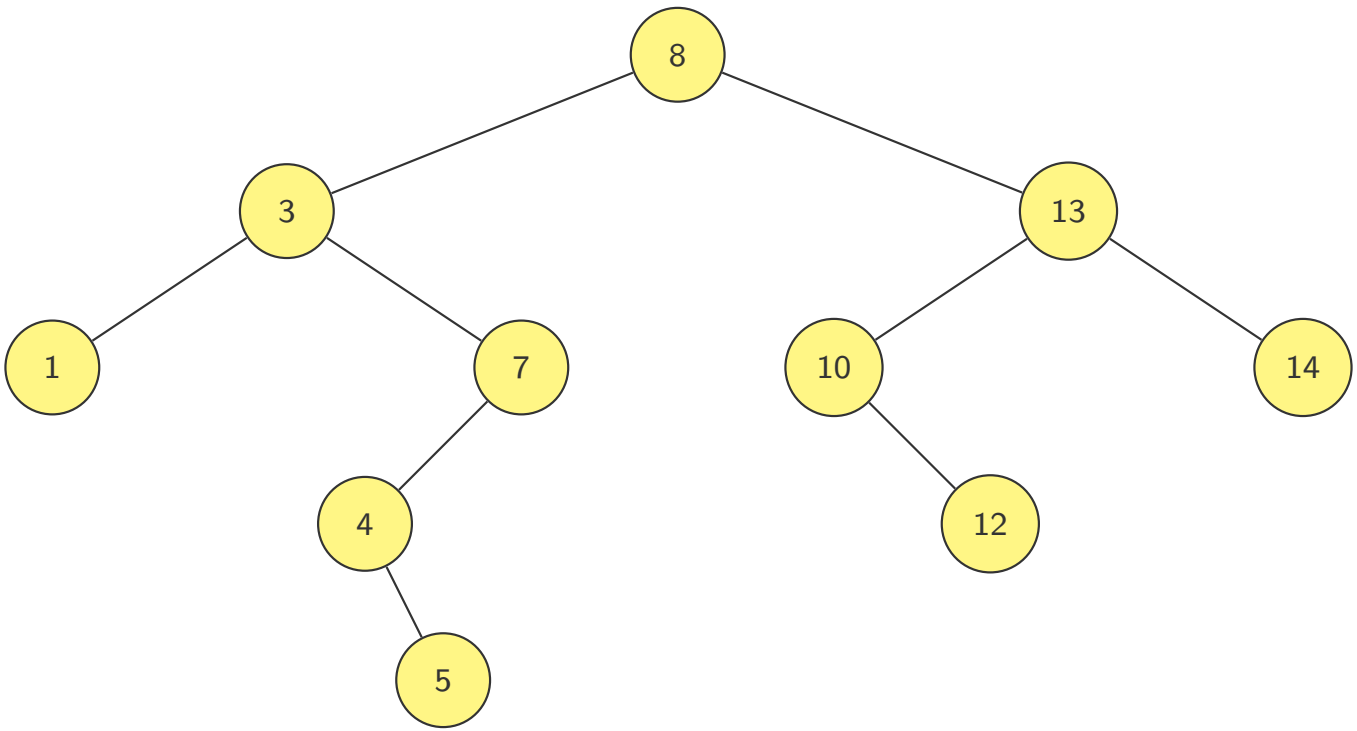


Quem é o sucessor de 3?

Sucessor

Dado um nó da árvore, onde está o seu sucessor?

- O sucessor é o próximo nó na ordenação



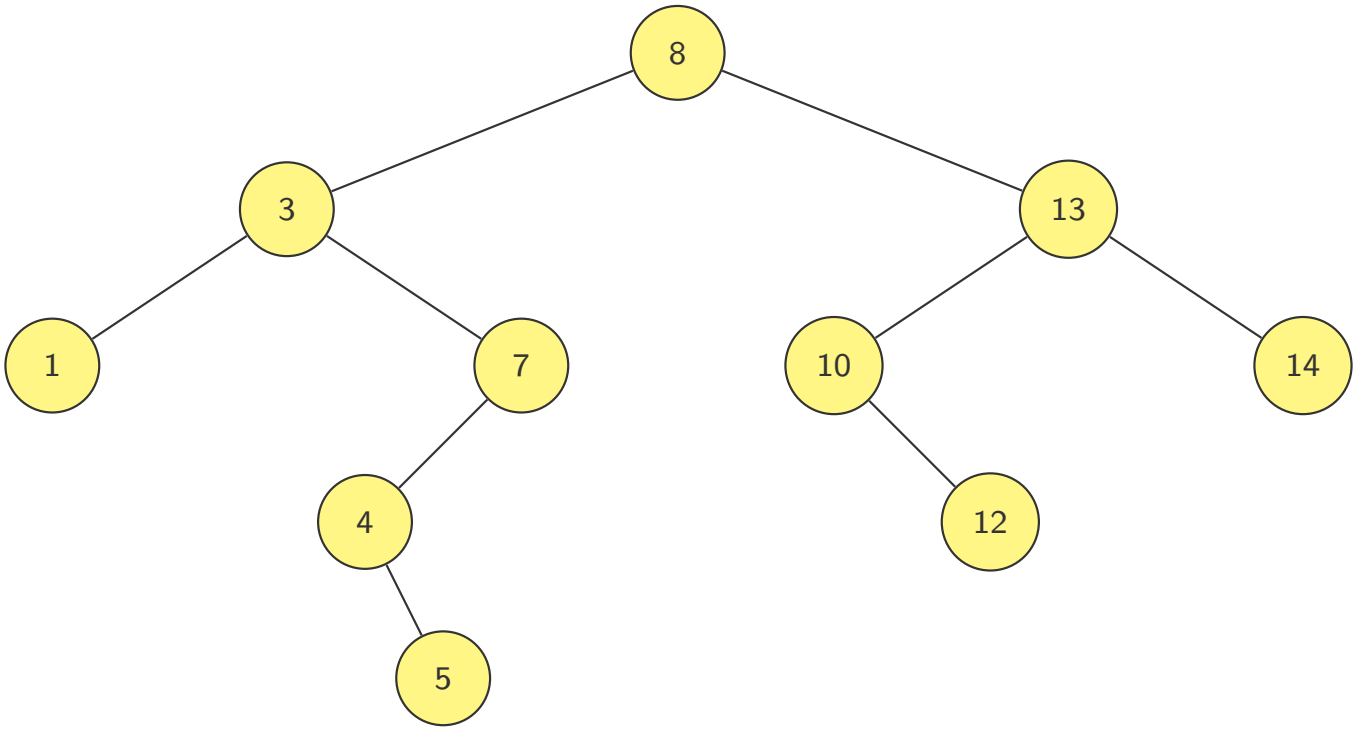
Quem é o sucessor de 3?

- É o mínimo da subárvore direita de 3

Sucessor

Dado um nó da árvore, onde está o seu sucessor?

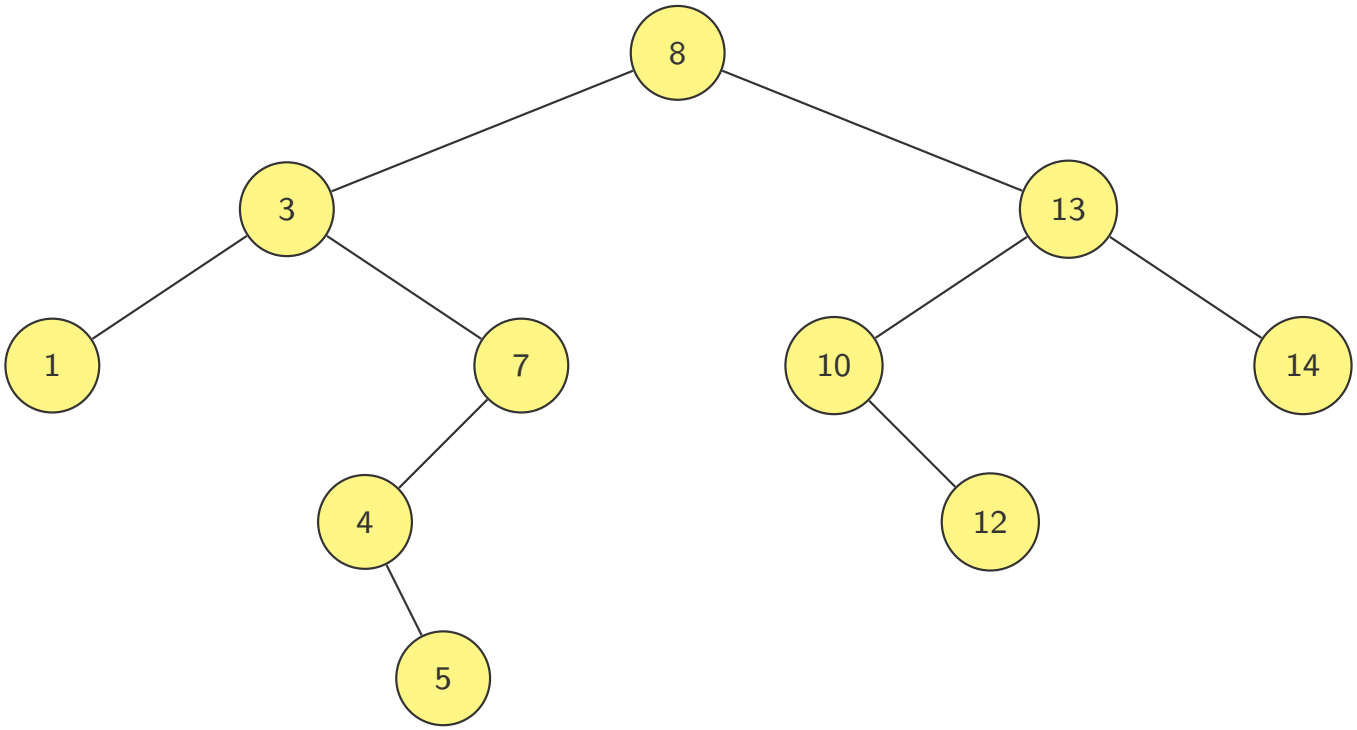
- O sucessor é o próximo nó na ordenação



Sucessor

Dado um nó da árvore, onde está o seu sucessor?

- O sucessor é o próximo nó na ordenação



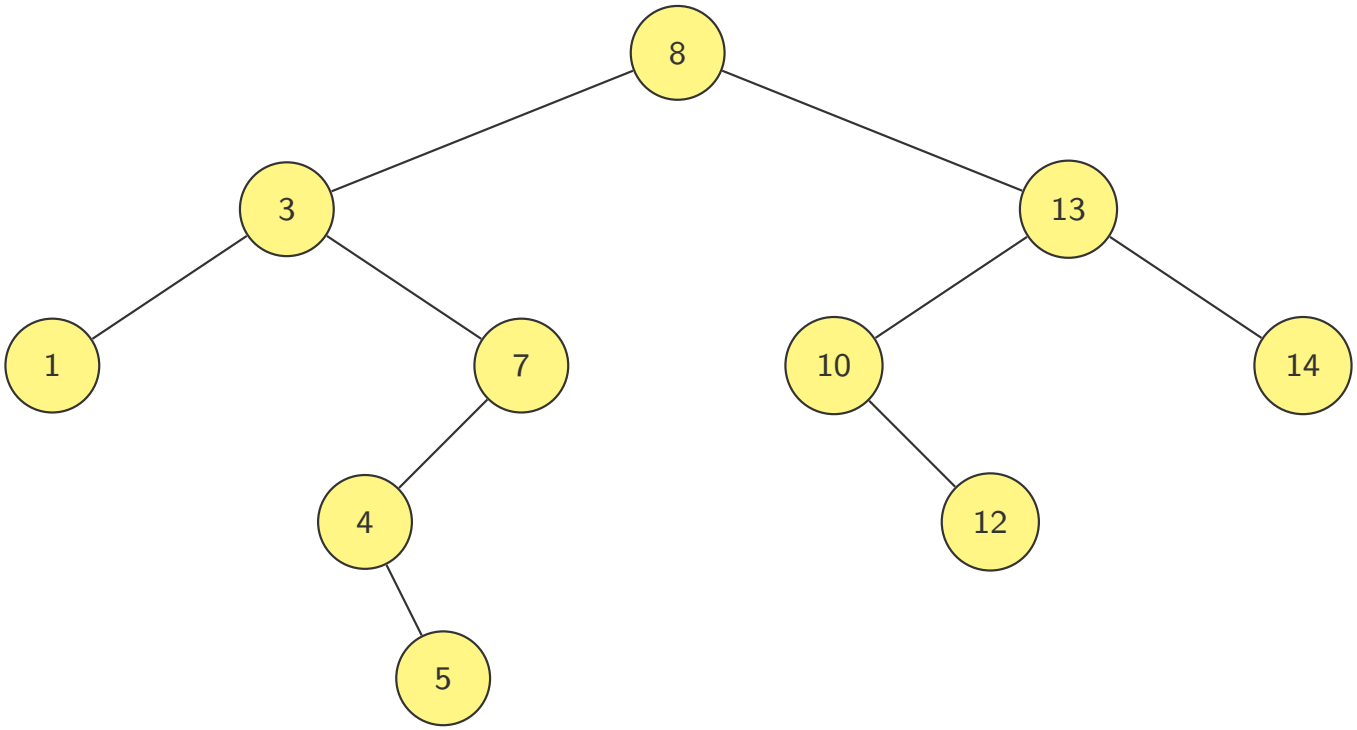
Quem é o sucessor de 7?

Sucessor

Universidade Federal do Ceará
Campus Quixadá

Dado um nó da árvore, onde está o seu sucessor?

- O sucessor é o próximo nó na ordenação



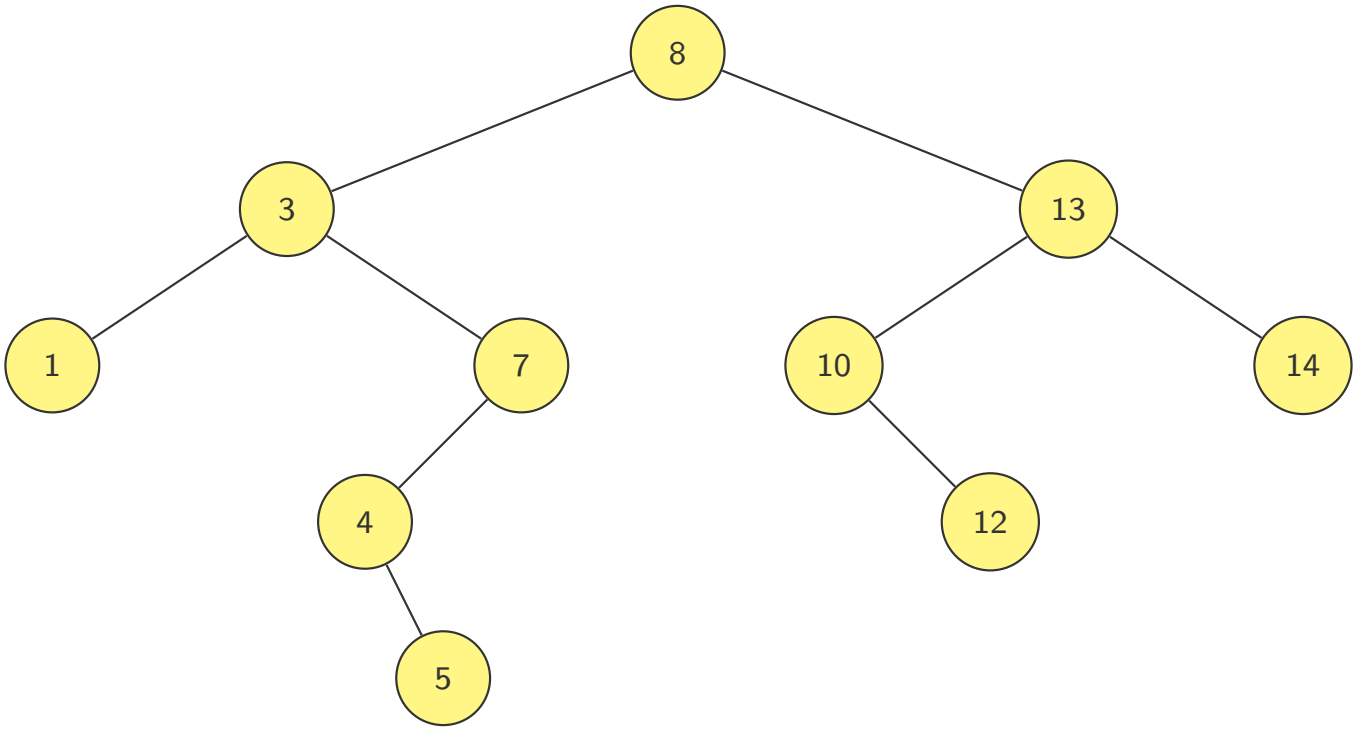
Quem é o sucessor de 7?

- É primeiro ancestral a direita

Sucessor

Dado um nó da árvore, onde está o seu sucessor?

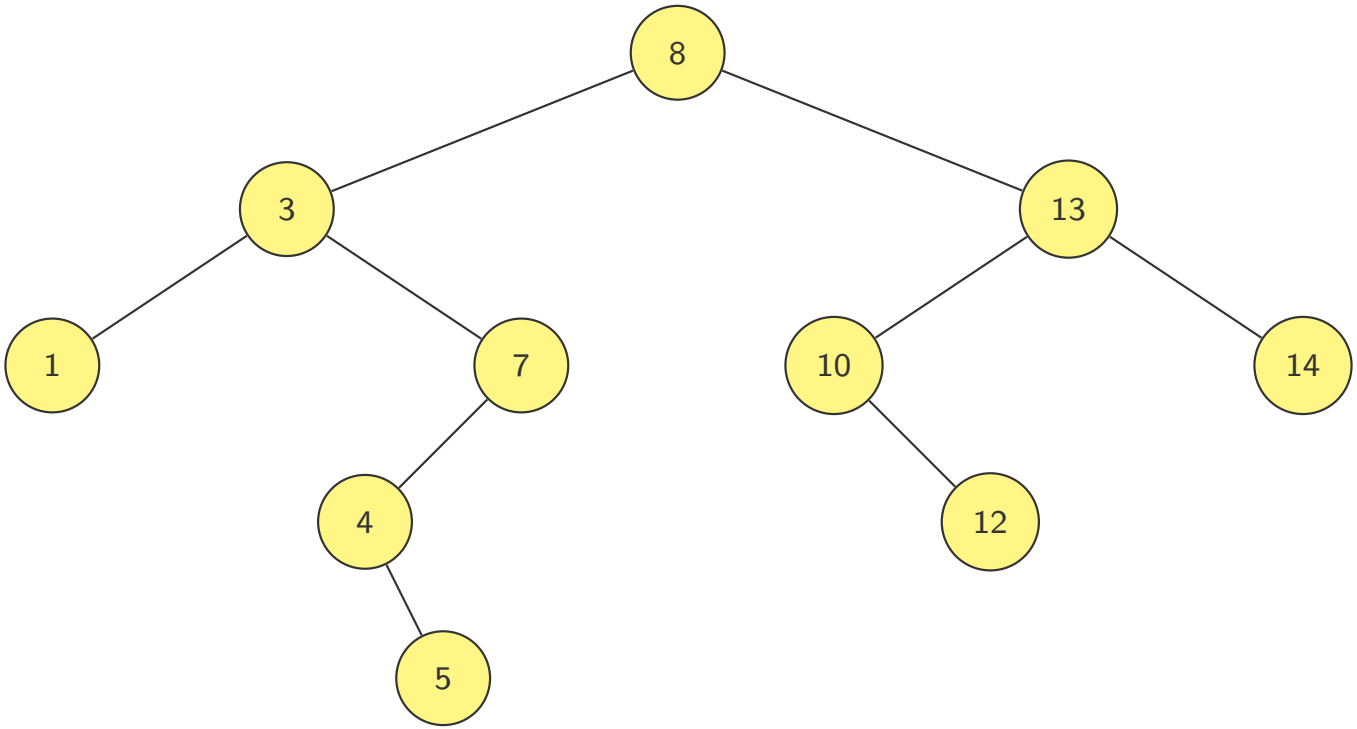
- O sucessor é o próximo nó na ordenação



Sucessor

Dado um nó da árvore, onde está o seu sucessor?

- O sucessor é o próximo nó na ordenação

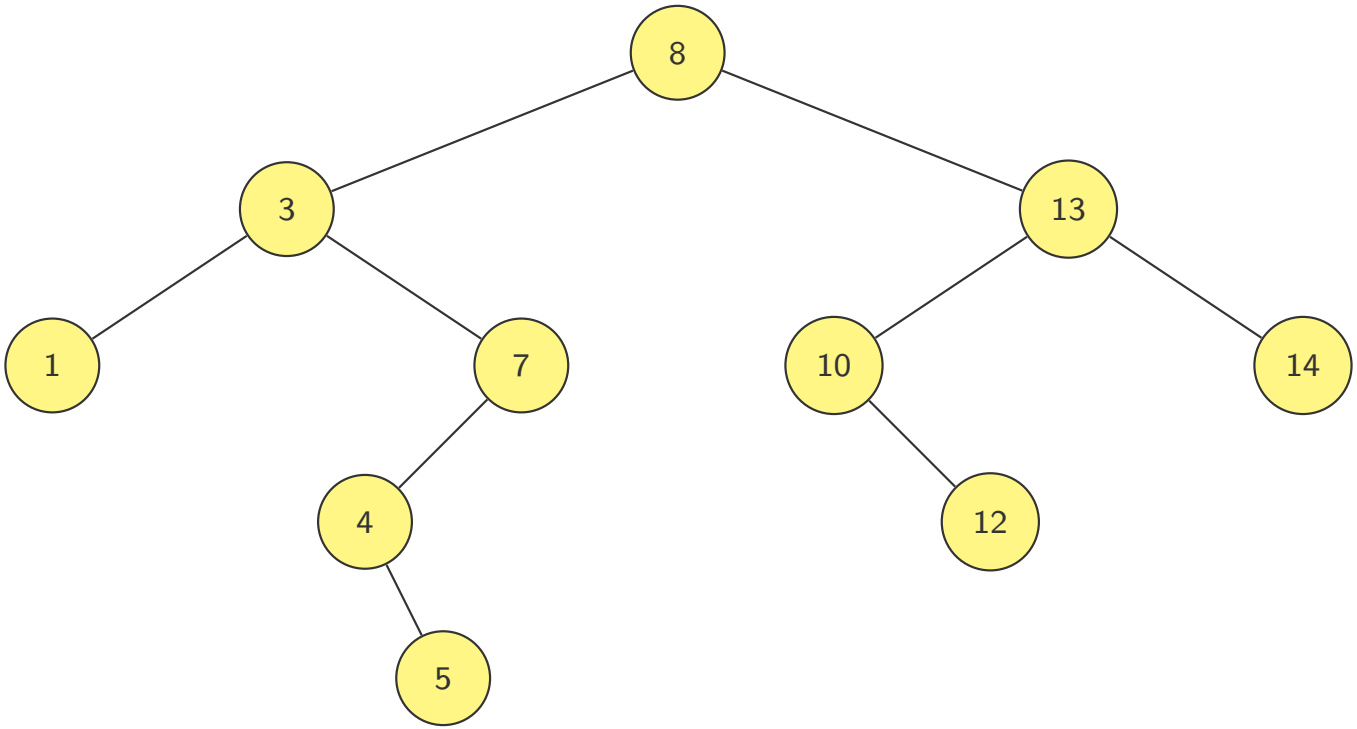


Quem é o sucessor de 14?

Sucessor

Dado um nó da árvore, onde está o seu sucessor?

- O sucessor é o próximo nó na ordenação



Quem é o sucessor de 14?

- não tem sucessor...

Sucessor - Implementação


```
1  // Retorna o ponteiro para o no sucessor do no x
2  // passado como parametro. A funcao tambem recebe
3  // como parametro a raiz da arvore.
4  NoArv* abb_sucessor(NoArv *x, NoArv* raiz) {
5      if(x == nullptr || raiz == nullptr)
6          return nullptr;
7      else if(x->dir != nullptr)
8          return abb_minimo(x->dir);
9      else
10         return ancestral_sucessor(x, raiz);
11 }
```

Sucessor - Implementação


```
1  // Retorna o ponteiro para o no sucessor do no x
2  // passado como parametro. A funcao tambem recebe
3  // como parametro a raiz da arvore.
4  NoArv* abb_sucessor(NoArv *x, NoArv* raiz) {
5      if(x == nullptr || raiz == nullptr)
6          return nullptr;
7      else if(x->dir != nullptr)
8          return abb_minimo(x->dir);
9      else
10         return ancestral_sucessor(x, raiz);
11 }
```

- Exercício para casa: Implementar a função:

  `NoArv *ancestral_sucessor(NoArv *x, NoArv* raiz)`

  Ela recebe o nó `x`, a raiz da árvore e, então, retorna o ancestral de `x` que é também seu sucessor.

Sucessor - Implementação


```
1  // Retorna o ponteiro para o no sucessor do no x
2  // passado como parametro. A funcao tambem recebe
3  // como parametro a raiz da arvore.
4  NoArv* abb_sucessor(NoArv *x, NoArv* raiz) {
5      if(x == nullptr || raiz == nullptr)
6          return nullptr;
7      else if(x->dir != nullptr)
8          return abb_minimo(x->dir);
9      else
10         return ancestral_sucessor(x, raiz);
11 }
```

- Exercício para casa: Implementar a função:

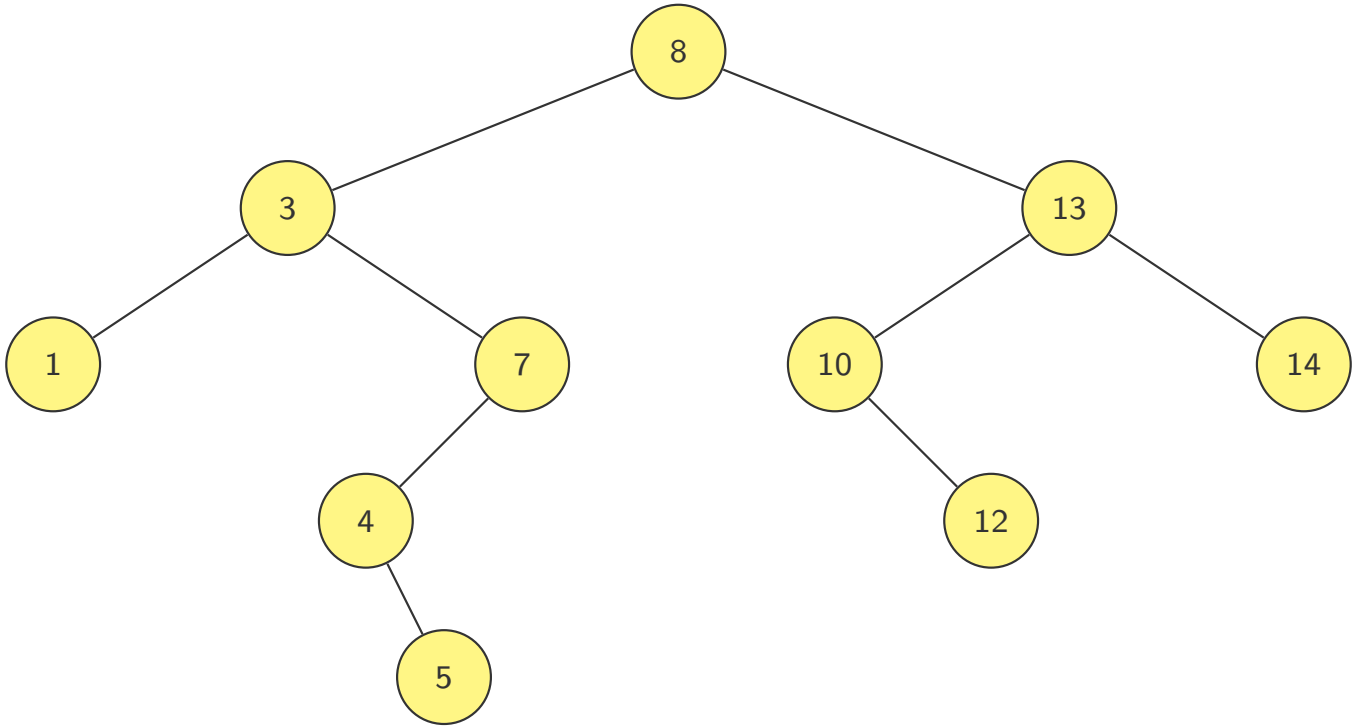
  `NoArv *ancestral_sucessor(NoArv *x, NoArv* raiz)`

  Ela recebe o nó `x`, a raiz da árvore e, então, retorna o ancestral de `x` que é também seu sucessor.

- A implementação da função `abb_antecessor` é simétrica a do sucessor. Implemente-a também.

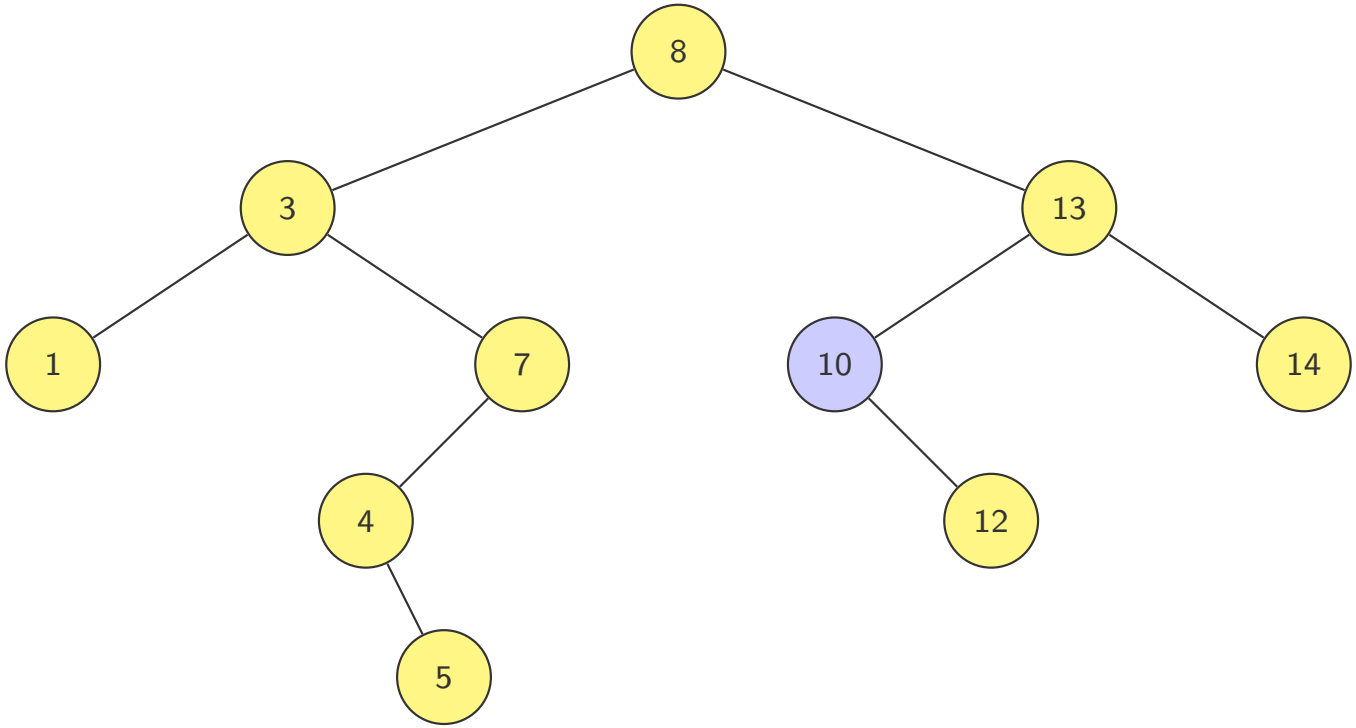
Remoção

Ex: removendo 10



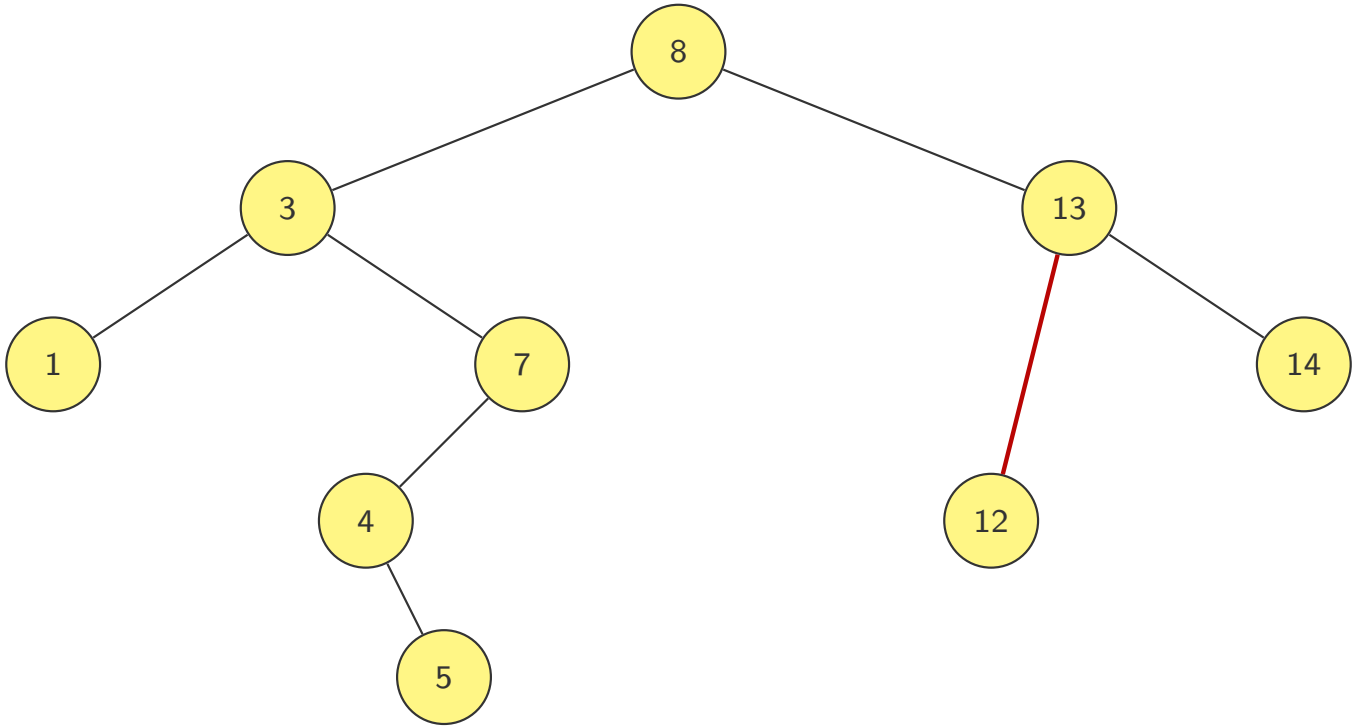
Remoção

Ex: removendo 10



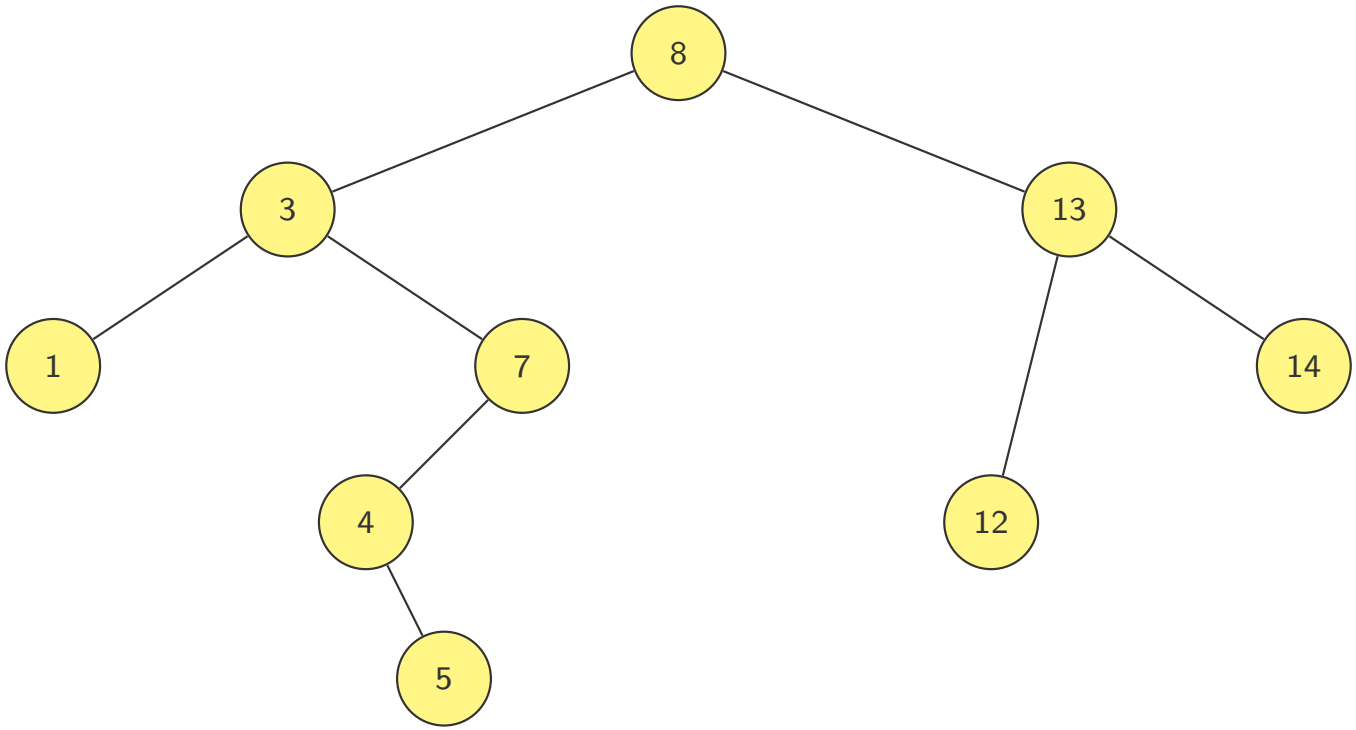
Remoção

Ex: removendo 10



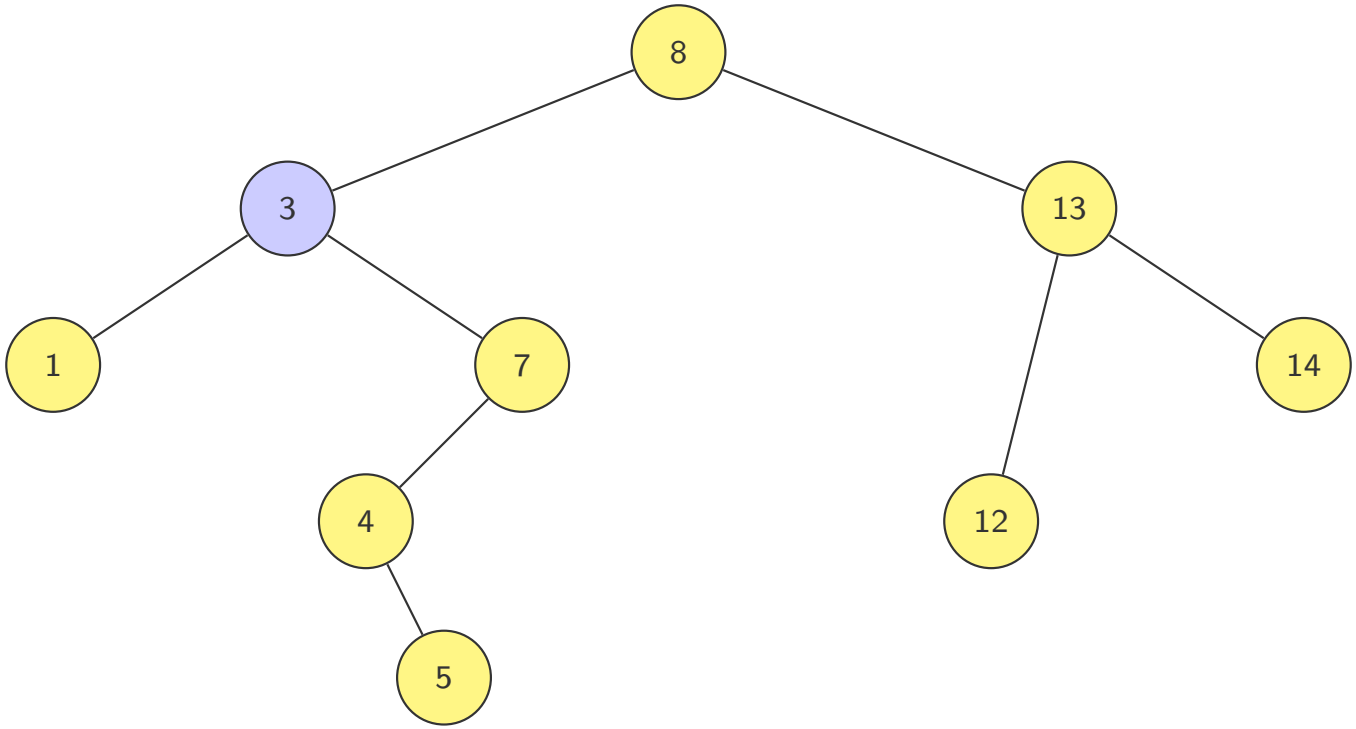
Remoção

Ex: removendo 3



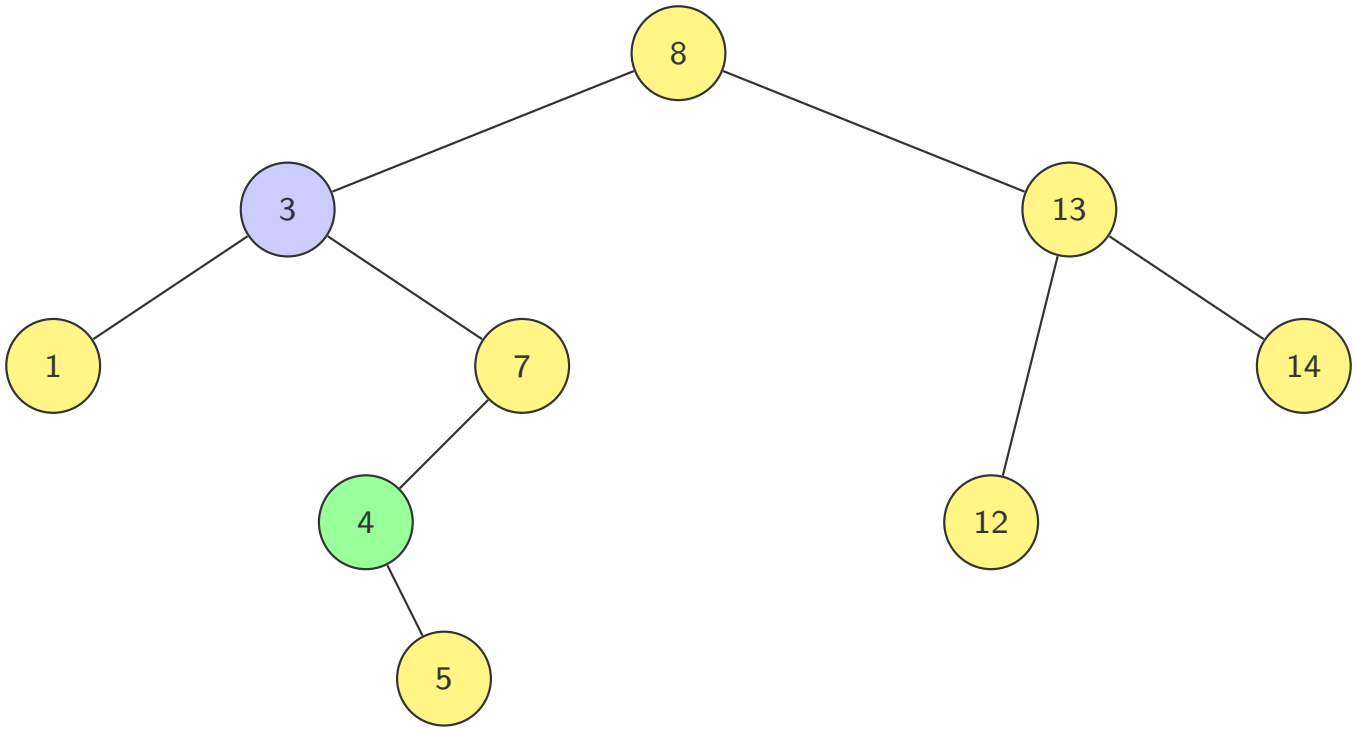
Remoção

Ex: removendo 3



Remoção

Ex: removendo 3

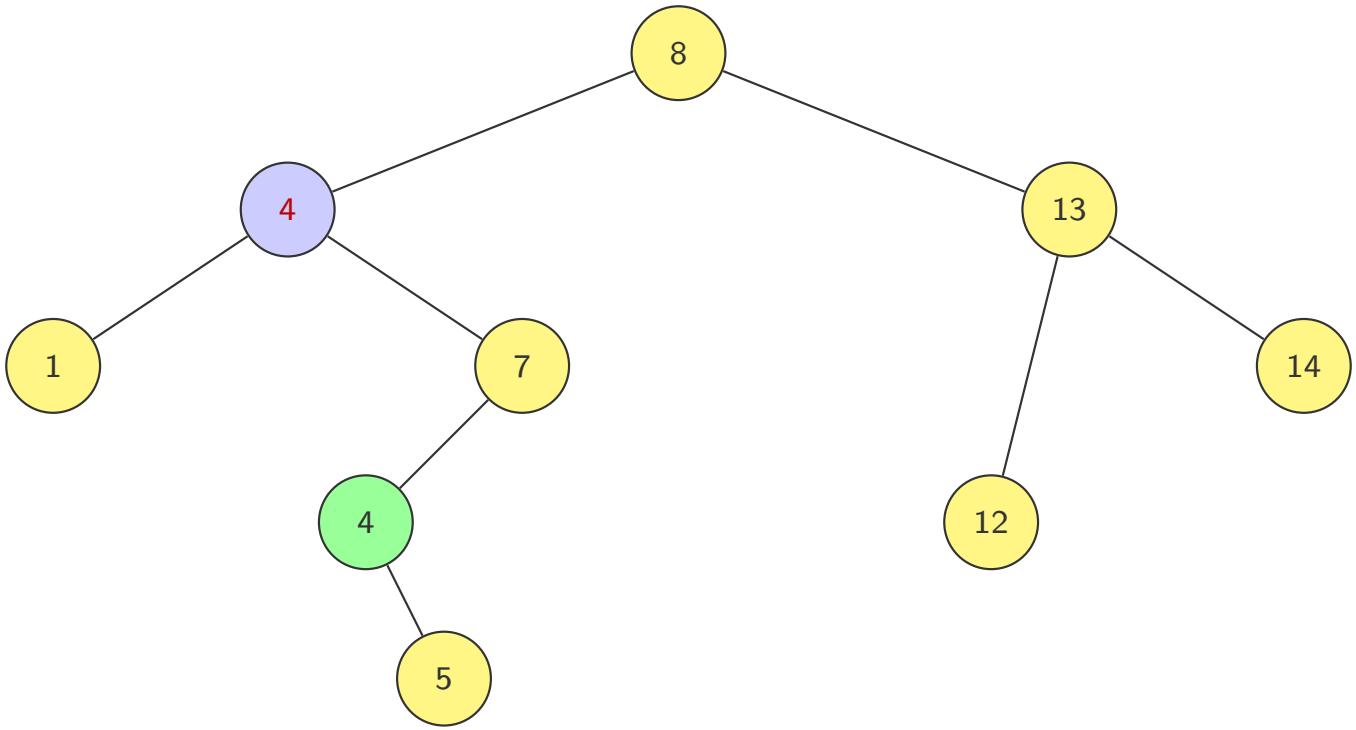


Podemos colocar o sucessor(ou antecessor) de 3 em seu lugar

- Isso mantém a propriedade da árvore binária de busca

Remoção

Ex: removendo 3

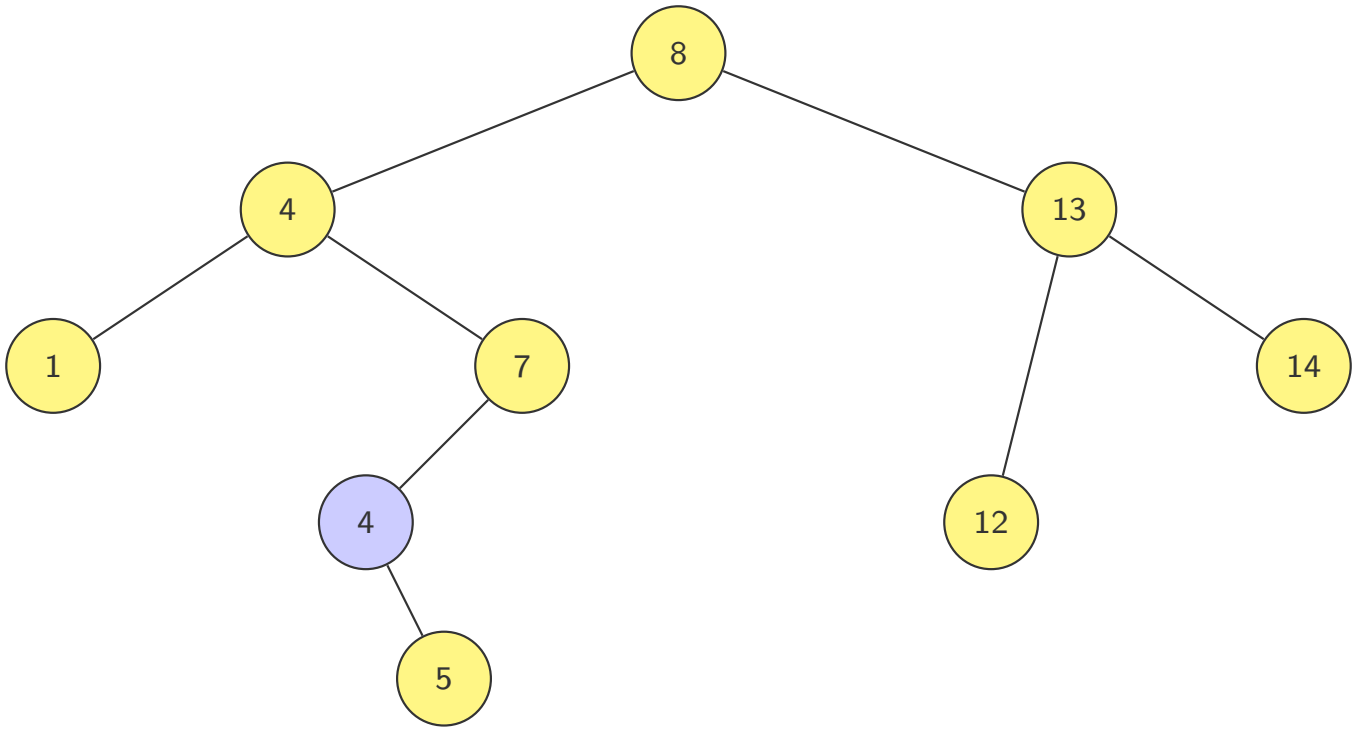


Podemos colocar o sucessor(ou antecessor) de 3 em seu lugar

- Isso mantém a propriedade da árvore binária de busca

Remoção

Ex: removendo 3



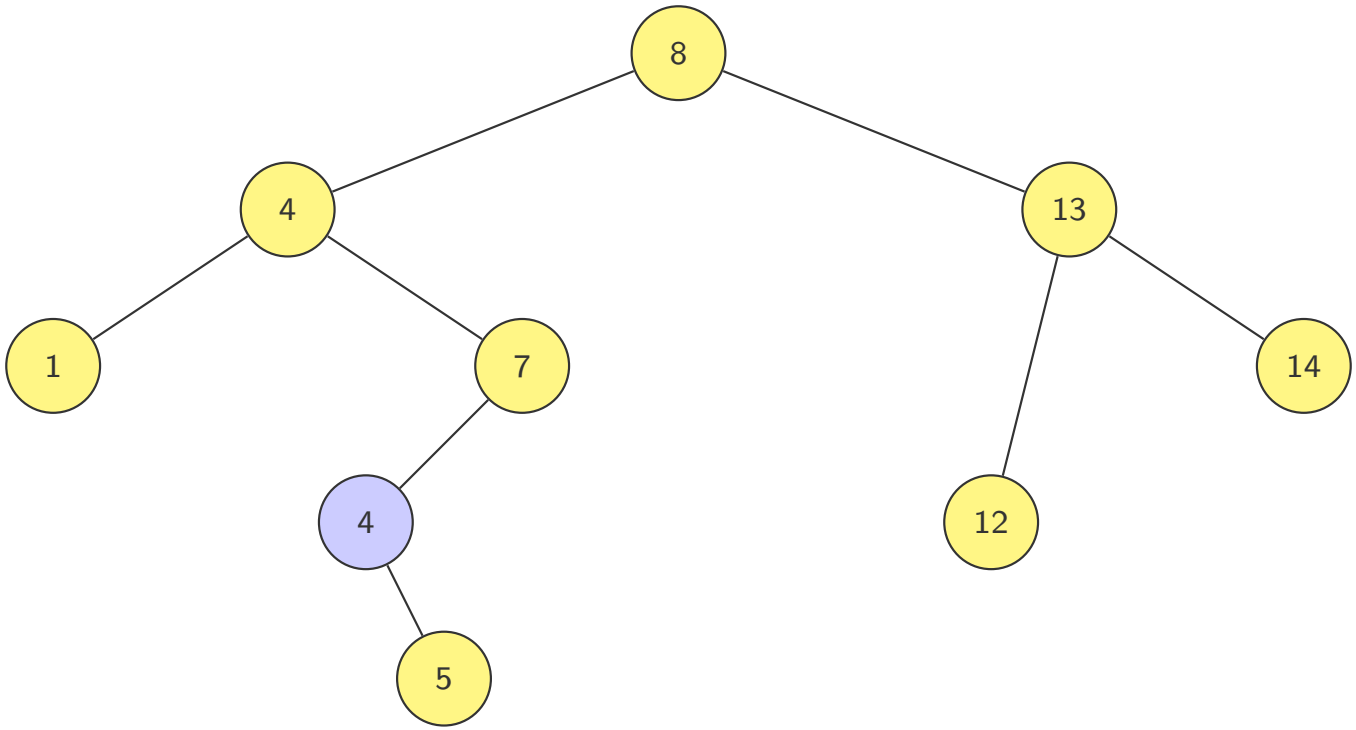
Podemos colocar o sucessor(ou antecessor) de 3 em seu lugar

- Isso mantém a propriedade da árvore binária de busca

E agora removemos o sucessor

Remoção

Ex: removendo 3



Podemos colocar o sucessor(ou antecessor) de 3 em seu lugar

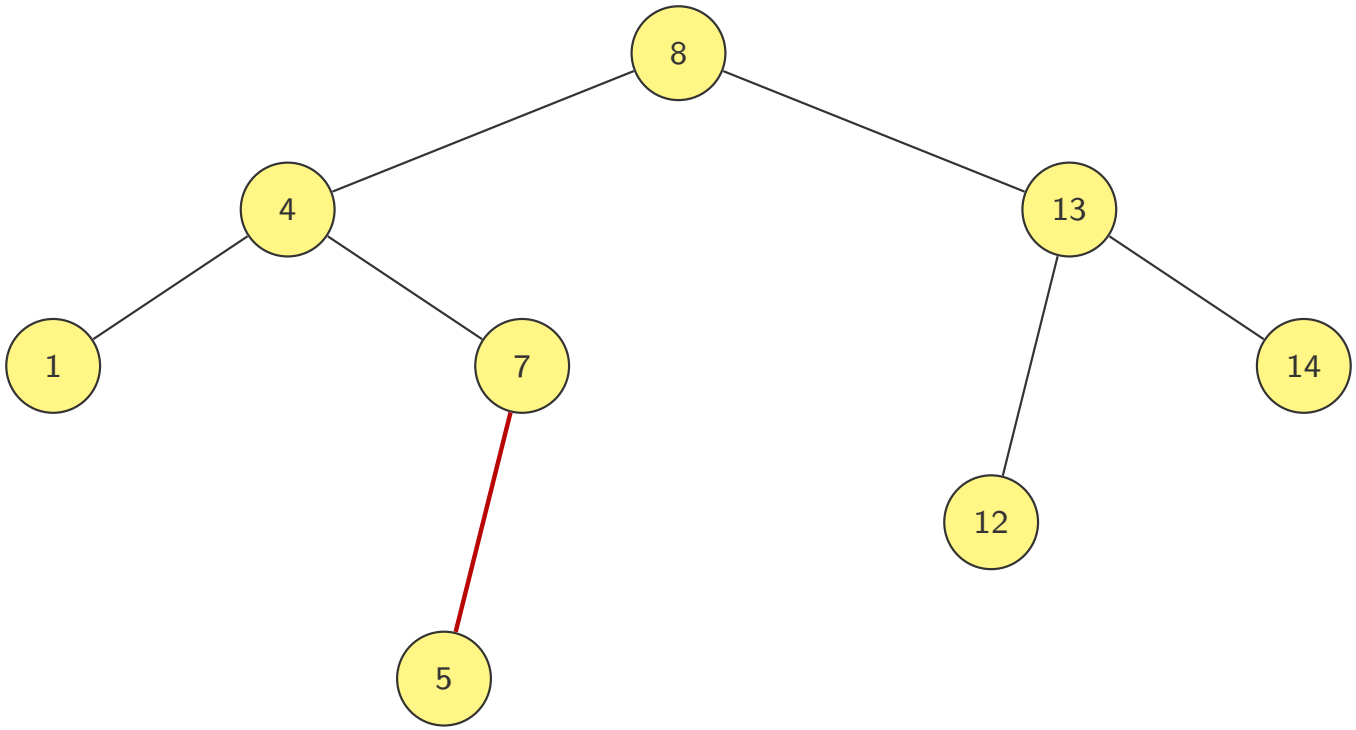
- Isso mantém a propriedade da árvore binária de busca

E agora removemos o sucessor

- O sucessor nunca tem filho esquerdo!

Remoção

Ex: removendo 3



Podemos colocar o sucessor(ou antecessor) de 3 em seu lugar

- Isso mantém a propriedade da árvore binária de busca

E agora removemos o sucessor

- O sucessor nunca tem filho esquerdo!

Remoção - Implementação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 NoArv* abb_remover(NoArv *raiz, int k) {
```

Remoção - Implementação


```
1 NoArv* abb_remover(NoArv *raiz, int k) {
2     if(raiz == nullptr) return nullptr; // arvore vazia
```

Remoção - Implementação


```
1 NoArv* abb_remover(NoArv *raiz, int k) {
2     if(raiz == nullptr) return nullptr; // arvore vazia
3     if(k > raiz->chave) // chave esta a direita
4         raiz->dir = abb_remover(raiz->dir, k);
```

Remoção - Implementação


```
1 NoArv* abb_remover(NoArv *raiz, int k) {
2     if(raiz == nullptr) return nullptr; // arvore vazia
3     if(k > raiz->chave) // chave esta a direita
4         raiz->dir = abb_remover(raiz->dir, k);
5     else if(k < raiz->chave) // chave esta a esquerda
6         raiz->esq = abb_remover(raiz->esq, k);
```

Remoção - Implementação


```
1  NoArv* abb_remover(NoArv *raiz, int k) {
2      if(raiz == nullptr) return nullptr; // arvore vazia
3      if(k > raiz->chave) // chave esta a direita
4          raiz->dir = abb_remover(raiz->dir, k);
5      else if(k < raiz->chave) // chave esta a esquerda
6          raiz->esq = abb_remover(raiz->esq, k);
7      /* achamos a chave */
8      else if(raiz->esq == nullptr && raiz->dir == nullptr) {
9          delete raiz;
10         raiz = nullptr;
11     }
```

Remoção - Implementação


```
1  NoArv* abb_remover(NoArv *raiz, int k) {
2      if(raiz == nullptr) return nullptr; // arvore vazia
3      if(k > raiz->chave) // chave esta a direita
4          raiz->dir = abb_remover(raiz->dir, k);
5      else if(k < raiz->chave) // chave esta a esquerda
6          raiz->esq = abb_remover(raiz->esq, k);
7      /* achamos a chave */
8      else if(raiz->esq == nullptr && raiz->dir == nullptr) {
9          delete raiz;
10         raiz = nullptr;
11     }
12     else if(raiz->esq == nullptr){ // apenas filho direito
13         NoArv *aux = raiz;
14         raiz = raiz->dir;
15         delete aux;
16     }
```

Remoção - Implementação


```
1  NoArv* abb_remover(NoArv *raiz, int k) {
2      if(raiz == nullptr) return nullptr; // arvore vazia
3      if(k > raiz->chave) // chave esta a direita
4          raiz->dir = abb_remover(raiz->dir, k);
5      else if(k < raiz->chave) // chave esta a esquerda
6          raiz->esq = abb_remover(raiz->esq, k);
7      /* achamos a chave */
8      else if(raiz->esq == nullptr && raiz->dir == nullptr) {
9          delete raiz;
10         raiz = nullptr;
11     }
12     else if(raiz->esq == nullptr){ // apenas filho direito
13         NoArv *aux = raiz;
14         raiz = raiz->dir;
15         delete aux;
16     }
17     else if(raiz->dir == nullptr){ // apenas filho esquerdo
18         NoArv *aux = raiz;
19         raiz = raiz->esq;
20         delete aux;
21     }
```

Remoção - Implementação


```
1  NoArv* abb_remover(NoArv *raiz, int k) {
2      if(raiz == nullptr) return nullptr; // arvore vazia
3      if(k > raiz->chave) // chave esta a direita
4          raiz->dir = abb_remover(raiz->dir, k);
5      else if(k < raiz->chave) // chave esta a esquerda
6          raiz->esq = abb_remover(raiz->esq, k);
7      /* achamos a chave */
8      else if(raiz->esq == nullptr && raiz->dir == nullptr) {
9          delete raiz;
10         raiz = nullptr;
11     }
12     else if(raiz->esq == nullptr){ // apenas filho direito
13         NoArv *aux = raiz;
14         raiz = raiz->dir;
15         delete aux;
16     }
17     else if(raiz->dir == nullptr){ // apenas filho esquerdo
18         NoArv *aux = raiz;
19         raiz = raiz->esq;
20         delete aux;
21     }
22     else { remove_antecessor(raiz); } // tem os dois filhos
23     return raiz;
24 }
```

Remoção - Implementação (continuação)


```
1  void remove_antecessor(NoArv *no) {
2      NoArv *t = no->esq; // t sera o maximo da subarvore
       esquerda
3      NoArv *pai = no; // pai sera o pai de t
4
5      while(t->dir != nullptr) {
6          pai = t;
7          t = t->dir;
8      }
9      no->chave = t->chave;
10
11     if(pai->dir == t)
12         pai->dir = t->esq; // O maximo nao tem filho direito
13     else  // O filho esquerdo de raiz nao tem filho direito
14         pai->esq = t->esq;
15     delete t;
16 }
```

main.cpp — Um exemplo de programa cliente

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  #include <iostream>
2  #include "ABB.h"
3  using namespace std;
4
5  int main() {
6      NoArv *arv = NULL;
7
8      arv = abb_inserir(arv, 4);
9      arv = abb_inserir(arv, 2);
10     arv = abb_inserir(arv, 1);
11     arv = abb_inserir(arv, 3);
12     arv = abb_inserir(arv, 6);
13     arv = abb_inserir(arv, 5);
14     arv = abb_inserir(arv, 7);
15
16     abb_preordem(arv);
17
18     arv = abb_destruir(arv);
19
20     return 0;
21 }
```

Exercícios

Exercício

- Escreva uma função que decida se uma dada árvore binária é ou não de busca.
- Conclua a implementação da função `abb_sucessor` vista nesta aula.
- Suponha que todo nó da ABB tenha agora um ponteiro para nó pai. Reimplemente as operações vistas nessa aula considerando este novo ponteiro.
- Escreva uma função que transforme um vetor crescente em uma árvore binária de busca balanceada.
- Escreva uma função que transforme uma árvore binária de busca em um vetor crescente.

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM