MC-202 — Unidade 15
Árvores Binárias de Busca

Rafael C. S. Schouery
rafael@ic.unicamp.br

Universidade Estadual de Campinas

1º semestre/2017

Relembrando

Vimos EDs que permitem inserir, remover e buscar

Relembrando

Vimos EDs que permitem inserir, remover e buscar

Estrutura de Dados	Inserção	Remoção	Busca
Vetor	$\mathrm{O}(1)$	$\mathrm{O}(1)$	$\mathrm{O}(n)$
Lista duplamente ligada	$\mathrm{O}(1)$	$\mathrm{O}(1)$	$\mathrm{O}(n)$
Vetor ordenado	$\mathrm{O}(n)$	$\mathrm{O}(n)$	$\mathrm{O}(\lg n)$

Relembrando

Vimos EDs que permitem inserir, remover e buscar

Estrutura de Dados	Inserção	Remoção	Busca
Vetor	$\mathrm{O}(1)$	$\mathrm{O}(1)$	$\mathrm{O}(n)$
Lista duplamente ligada	$\mathrm{O}(1)$	$\mathrm{O}(1)$	$\mathrm{O}(n)$
Vetor ordenado	$\mathrm{O}(n)$	$\mathrm{O}(n)$	$\mathrm{O}(\lg n)$

Veremos árvores binárias de busca

Relembrando

Vimos EDs que permitem inserir, remover e buscar

Estrutura de Dados	Inserção	Remoção	Busca
Vetor	$\mathrm{O}(1)$	$\mathrm{O}(1)$	$\mathrm{O}(n)$
Lista duplamente ligada	$\mathrm{O}(1)$	$\mathrm{O}(1)$	$\mathrm{O}(n)$
Vetor ordenado	$\mathrm{O}(n)$	$\mathrm{O}(n)$	$\mathrm{O}(\lg n)$

Veremos árvores binárias de busca

- primeiro uma versão simples, depois versão sofisticada

Relembrando

Vimos EDs que permitem inserir, remover e buscar

Estrutura de Dados	Inserção	Remoção	Busca
Vetor	$\mathrm{O}(1)$	$\mathrm{O}(1)$	$\mathrm{O}(n)$
Lista duplamente ligada	$\mathrm{O}(1)$	$\mathrm{O}(1)$	$\mathrm{O}(n)$
Vetor ordenado	$\mathrm{O}(n)$	$\mathrm{O}(n)$	$\mathrm{O}(\lg n)$

Veremos árvores binárias de busca

- primeiro uma versão simples, depois versão sofisticada
- versão sofisticada: três operações levam $\mathrm{O}(\lg n)$

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. $e \leq r$ 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:

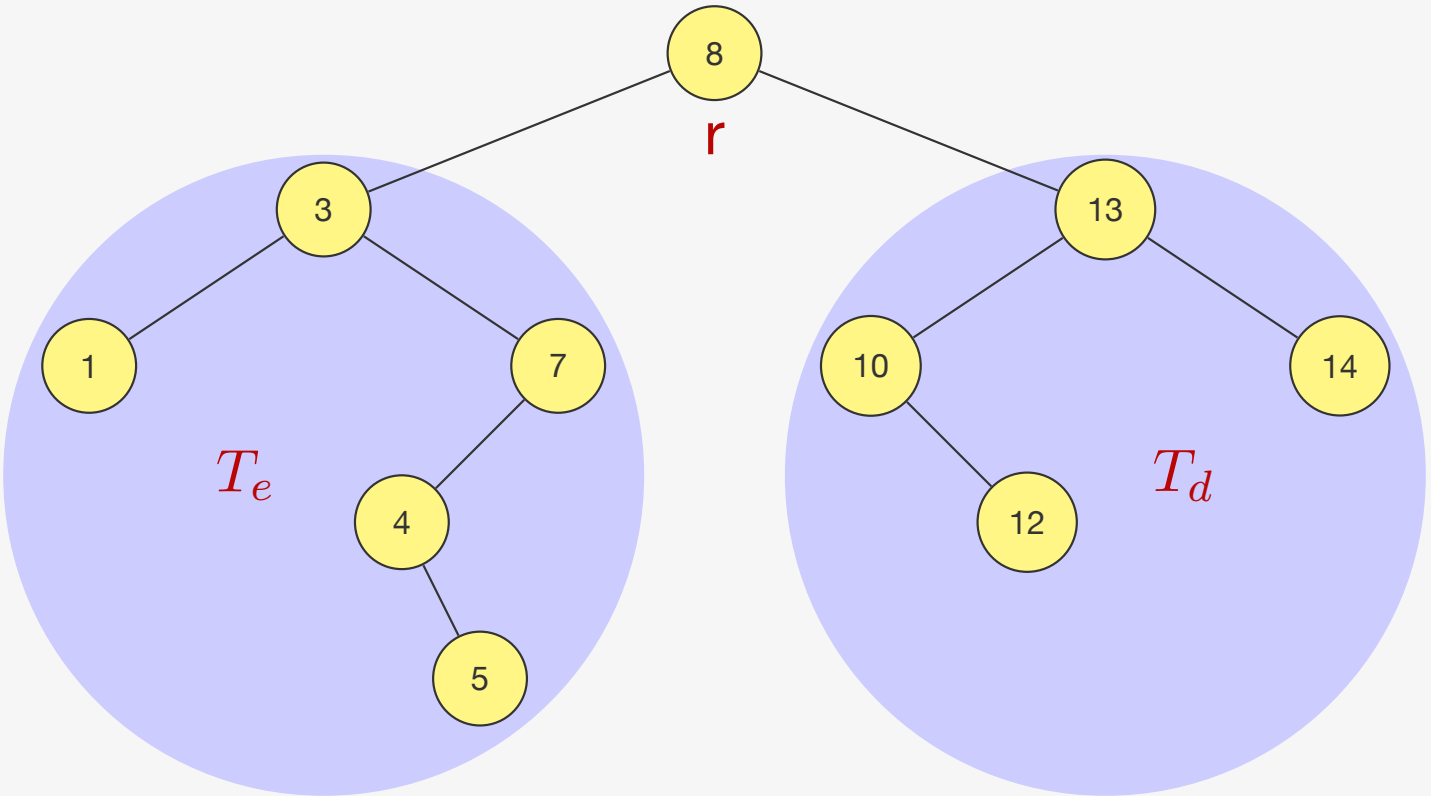
1. $e \leq r$ para todo elemento $e \in T_e$
2. $r \leq d$ para todo elemento $d \in T_d$

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. $e \leq r$ para todo elemento $e \in T_e$
2. $r \leq d$ para todo elemento $d \in T_d$



TAD - Árvores de Busca Binária

```
1  typedef struct No {
2      int chave;
3      struct No *esq, *dir, *pai; /*pai é opcional, usado em
       sucessor e antecessor*/
4  } No;
5
6  void destruir_arvore(No *arvore);
7
8  void inserir(No **arvore, int chave);
9
10 void remover(No **arvore, int chave);
11
12 No * buscar(No *arvore, int chave);
13
14 No * minimo(No *arvore);
15
16 No * maximo(No *arvore);
17
18 No * sucessor(No *x);
19
20 No * antecessor(No *x);
```

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

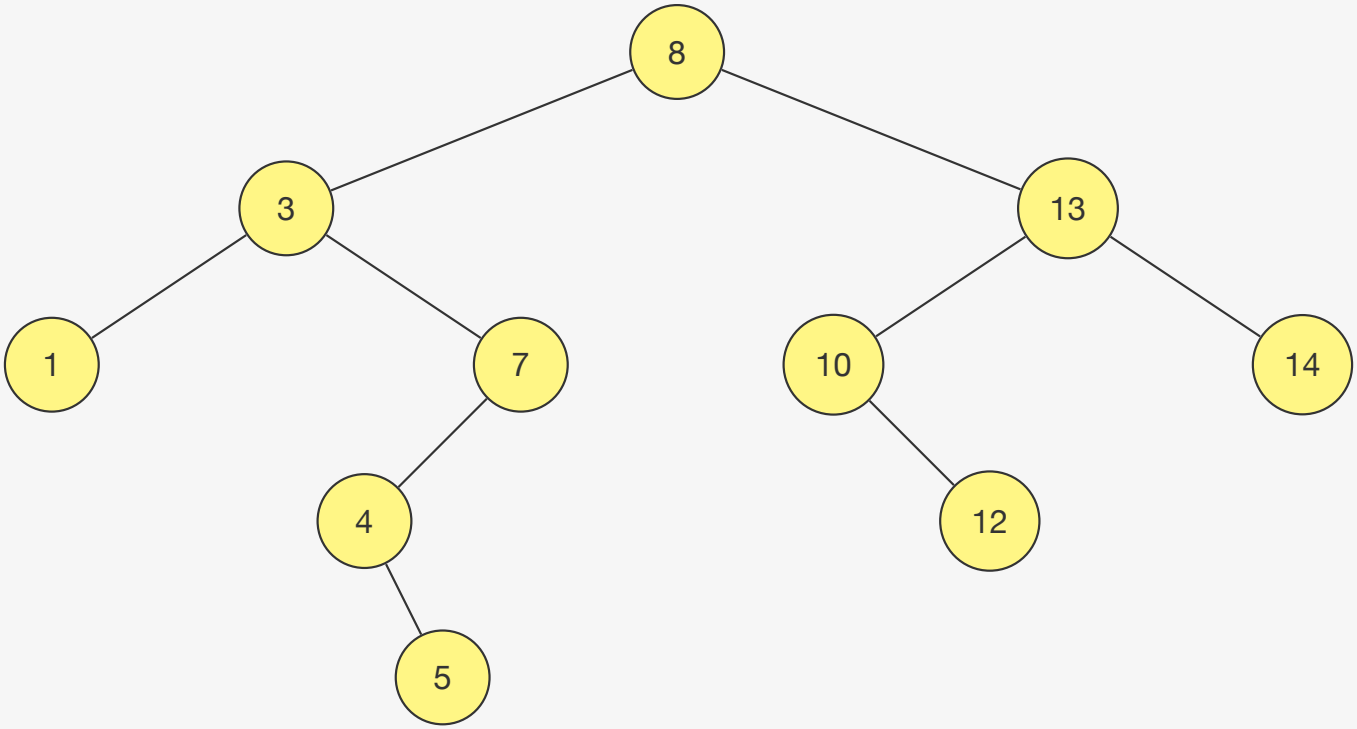
Ex: Buscando por 4

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

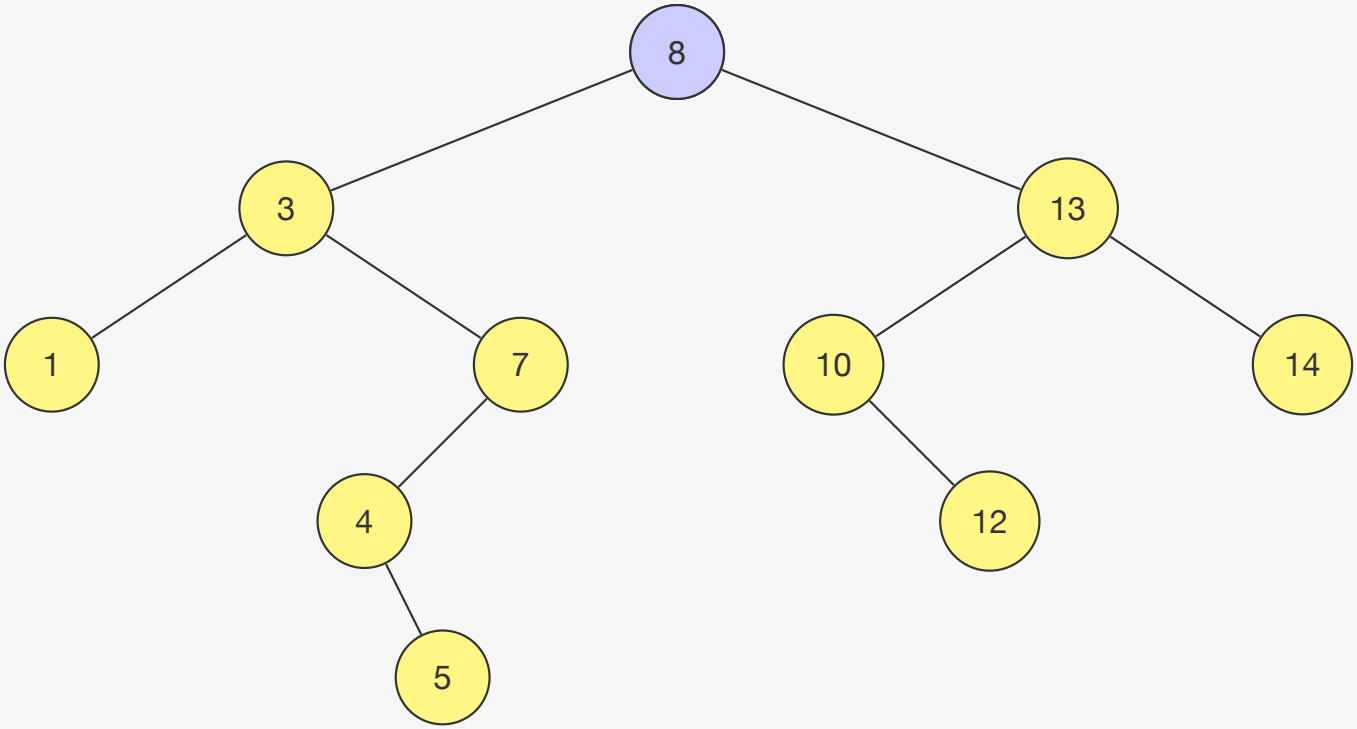


Busca por um valor

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

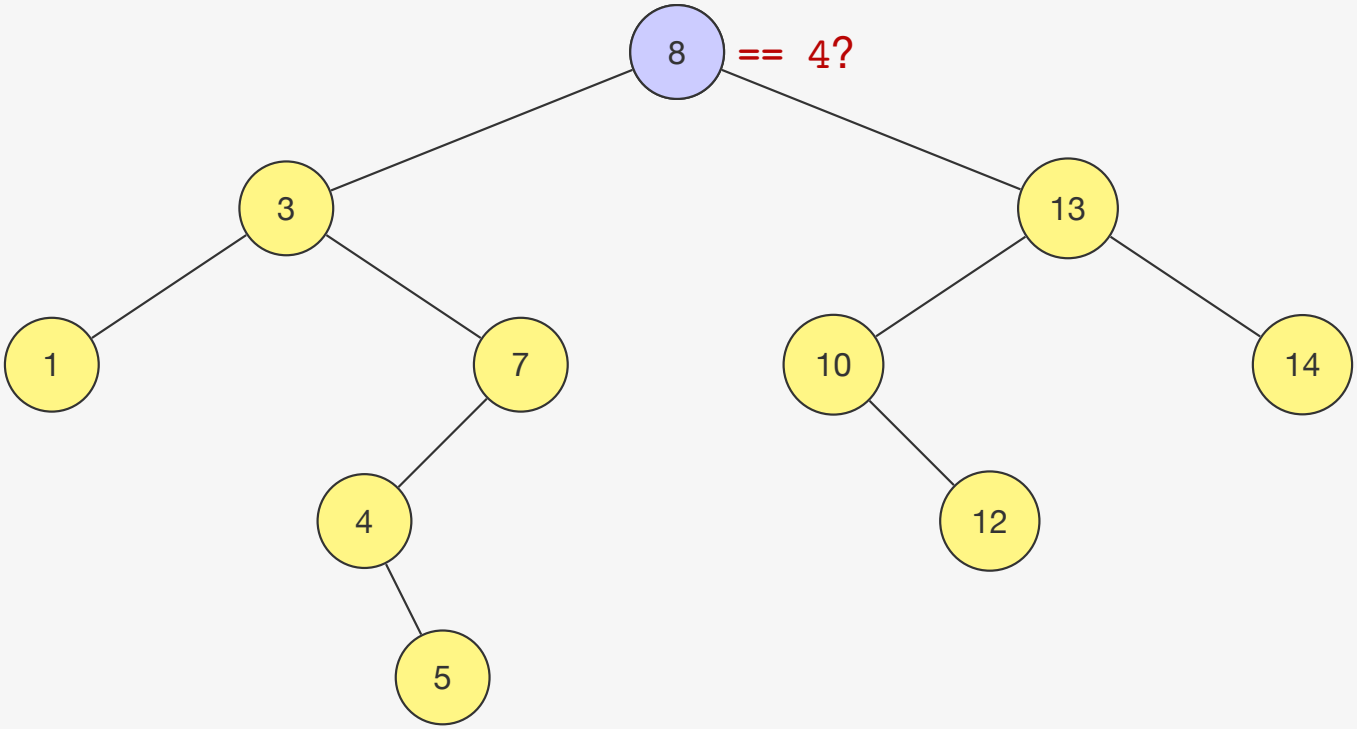


Busca por um valor

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

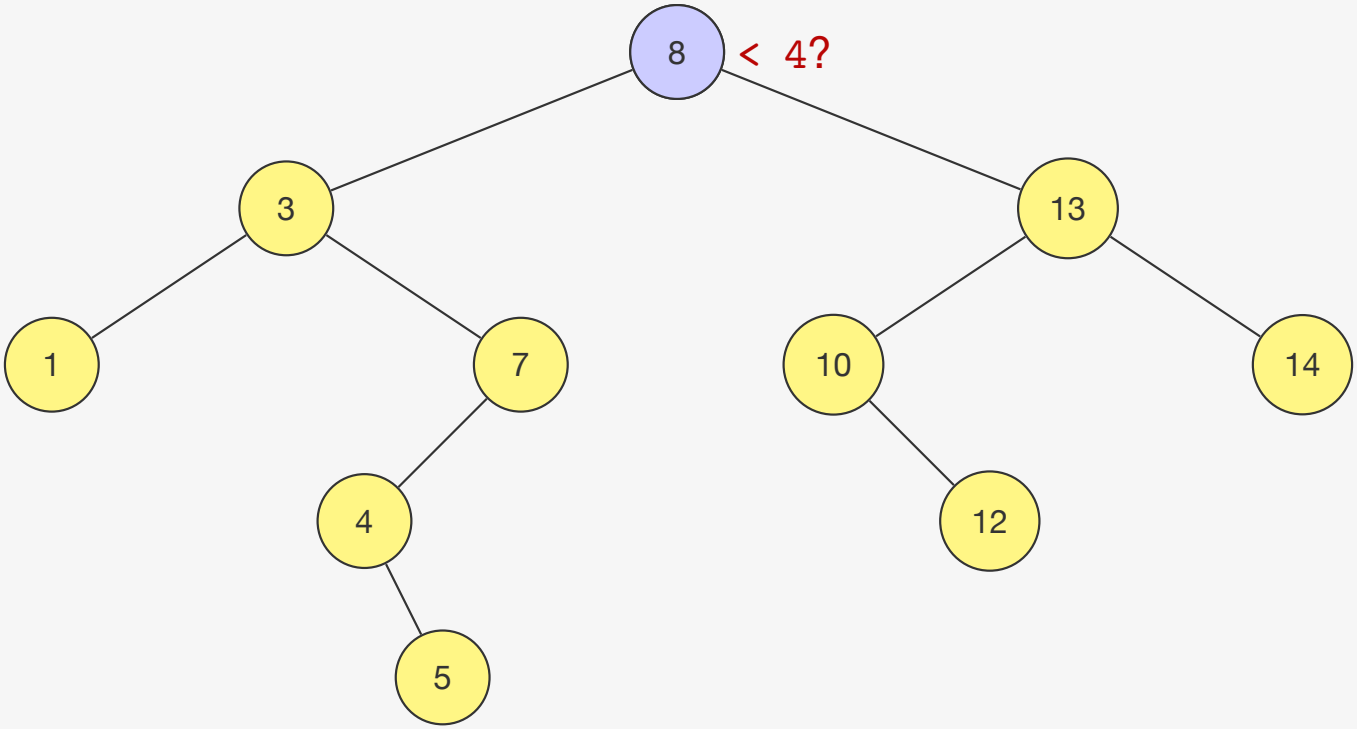


Busca por um valor

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

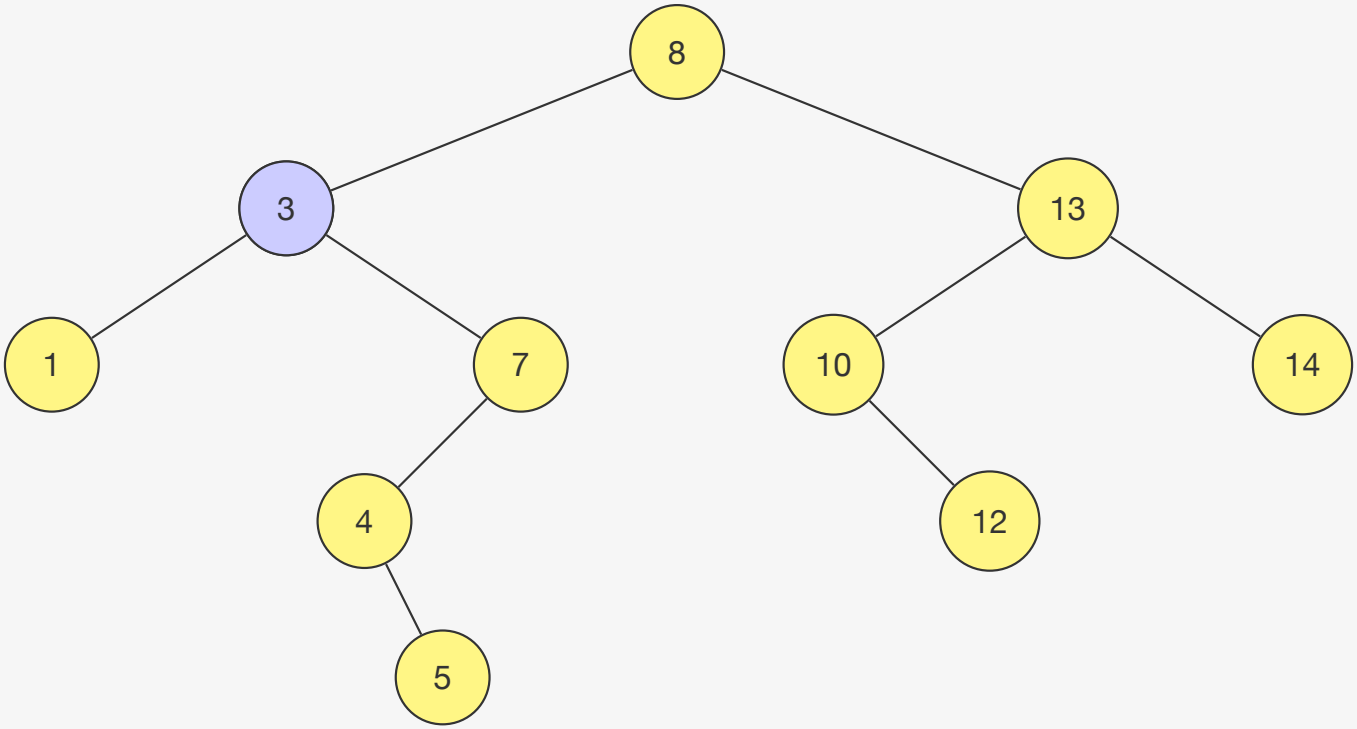


Busca por um valor

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

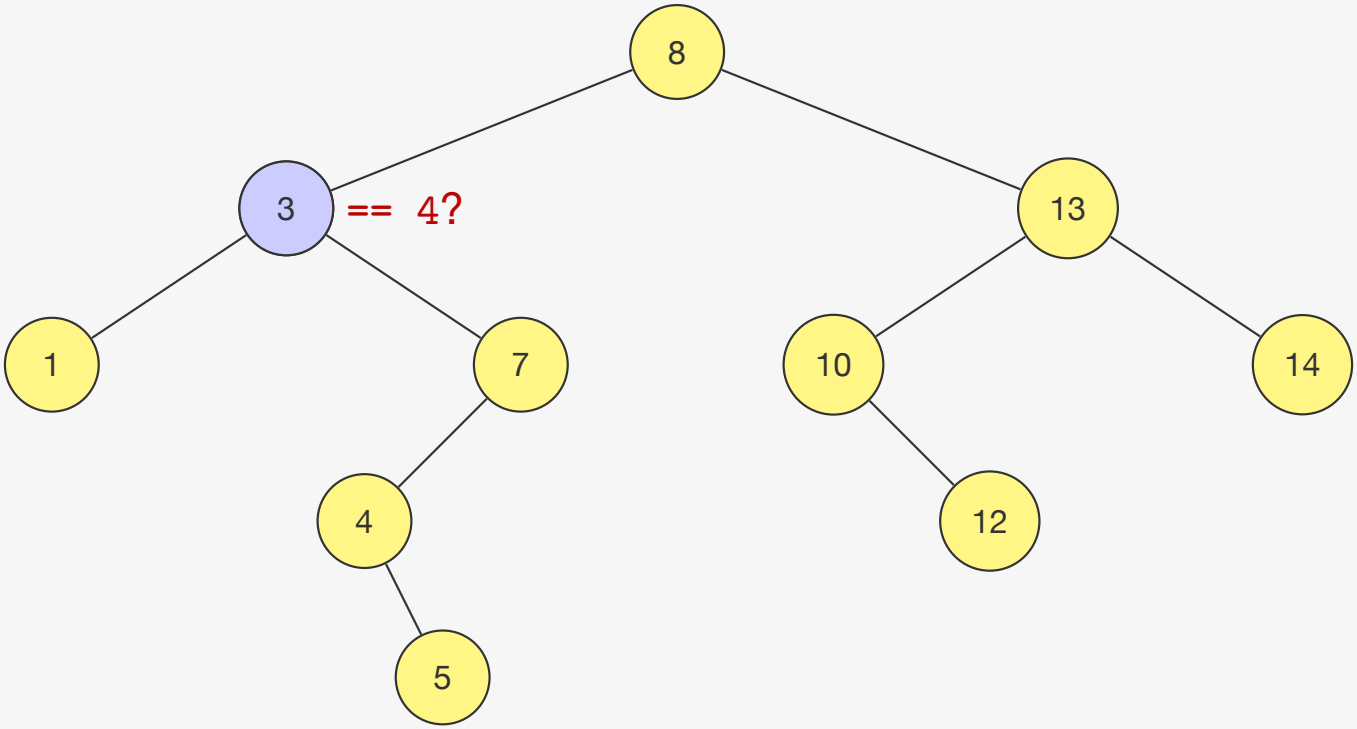


Busca por um valor

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

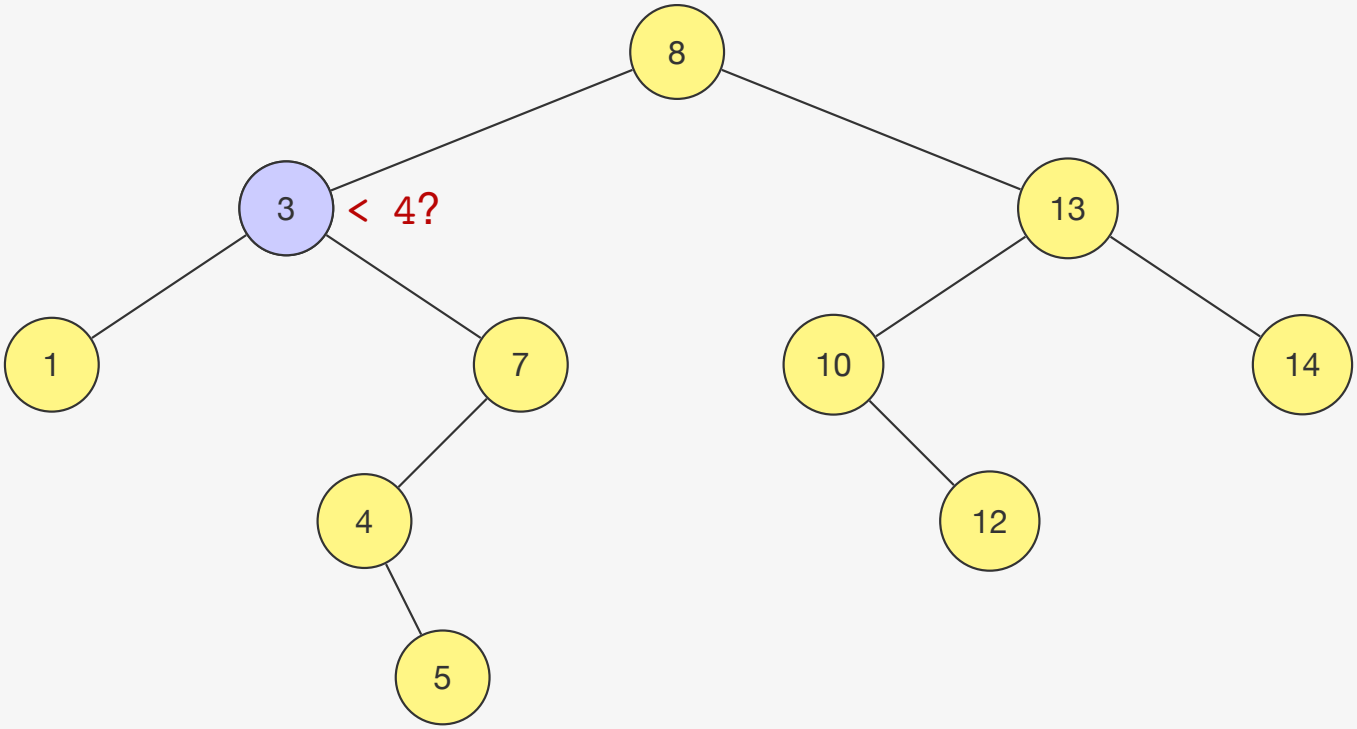


Busca por um valor

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

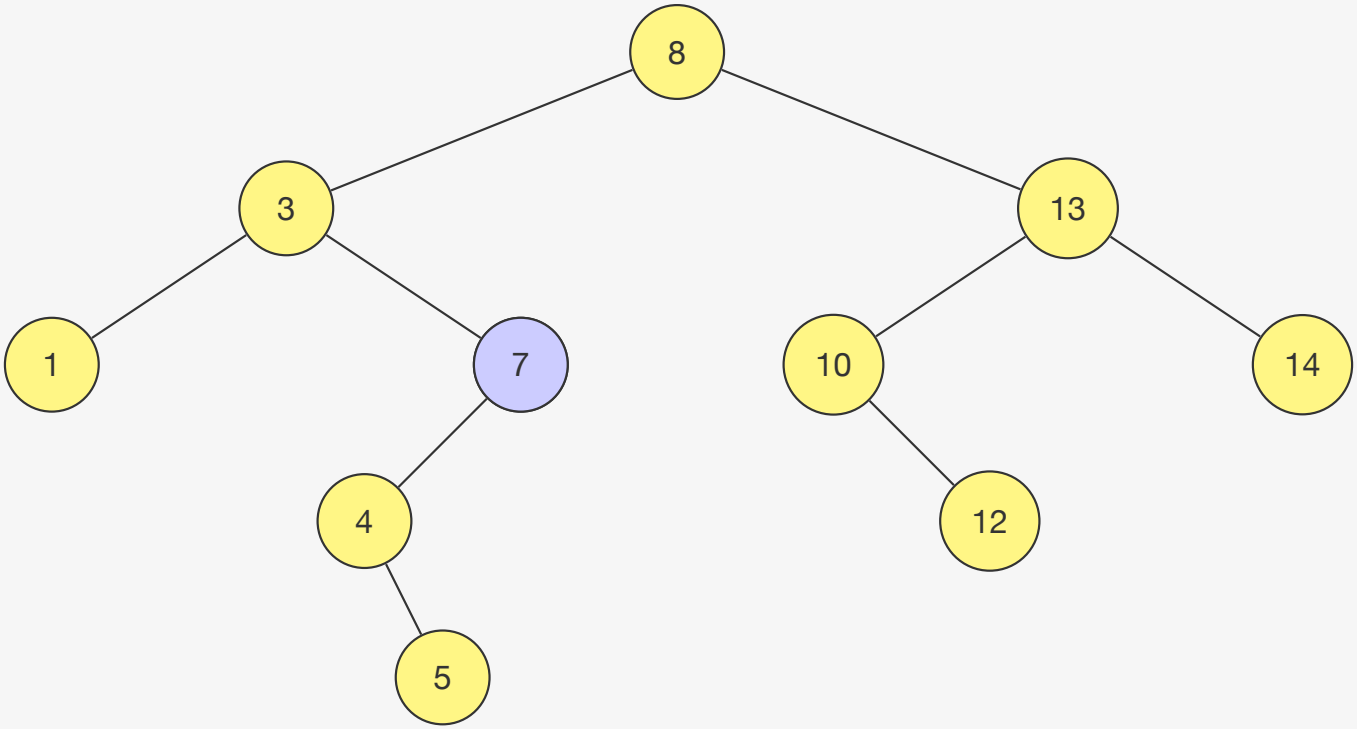


Busca por um valor

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

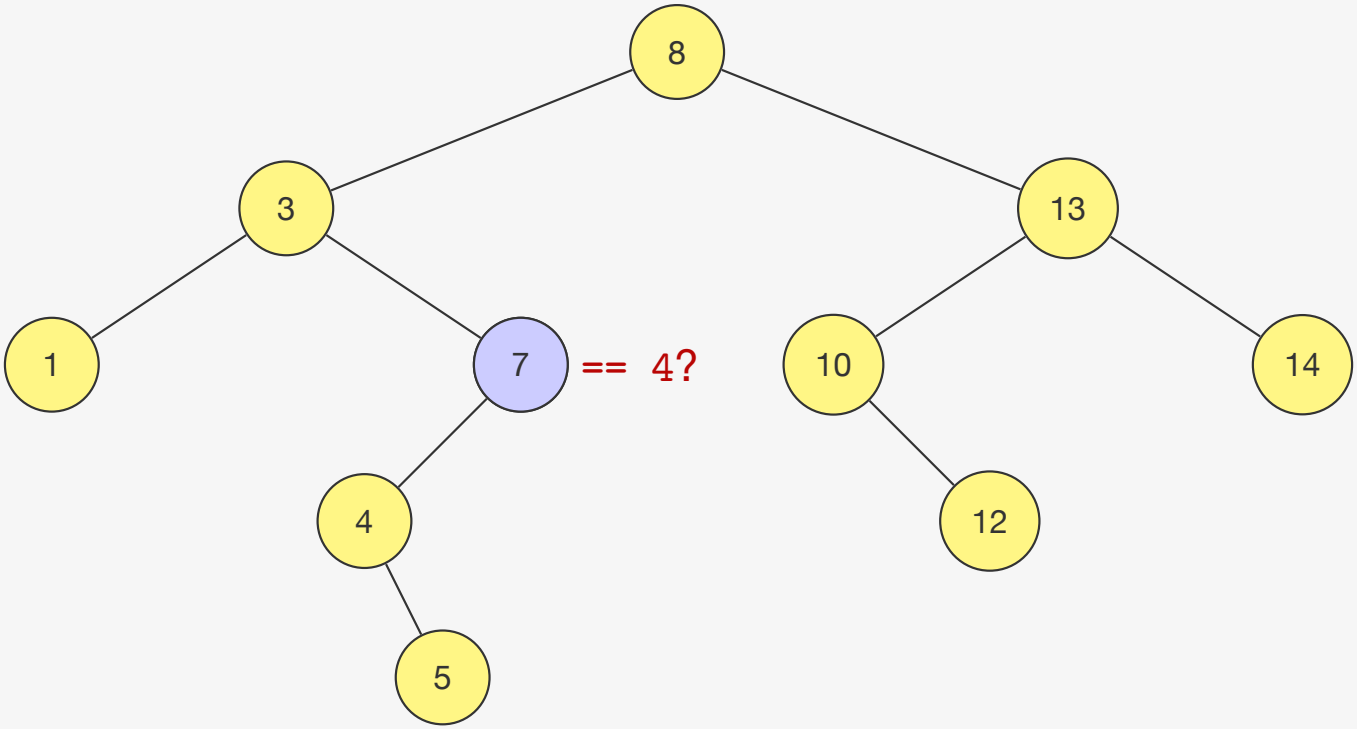


Busca por um valor

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

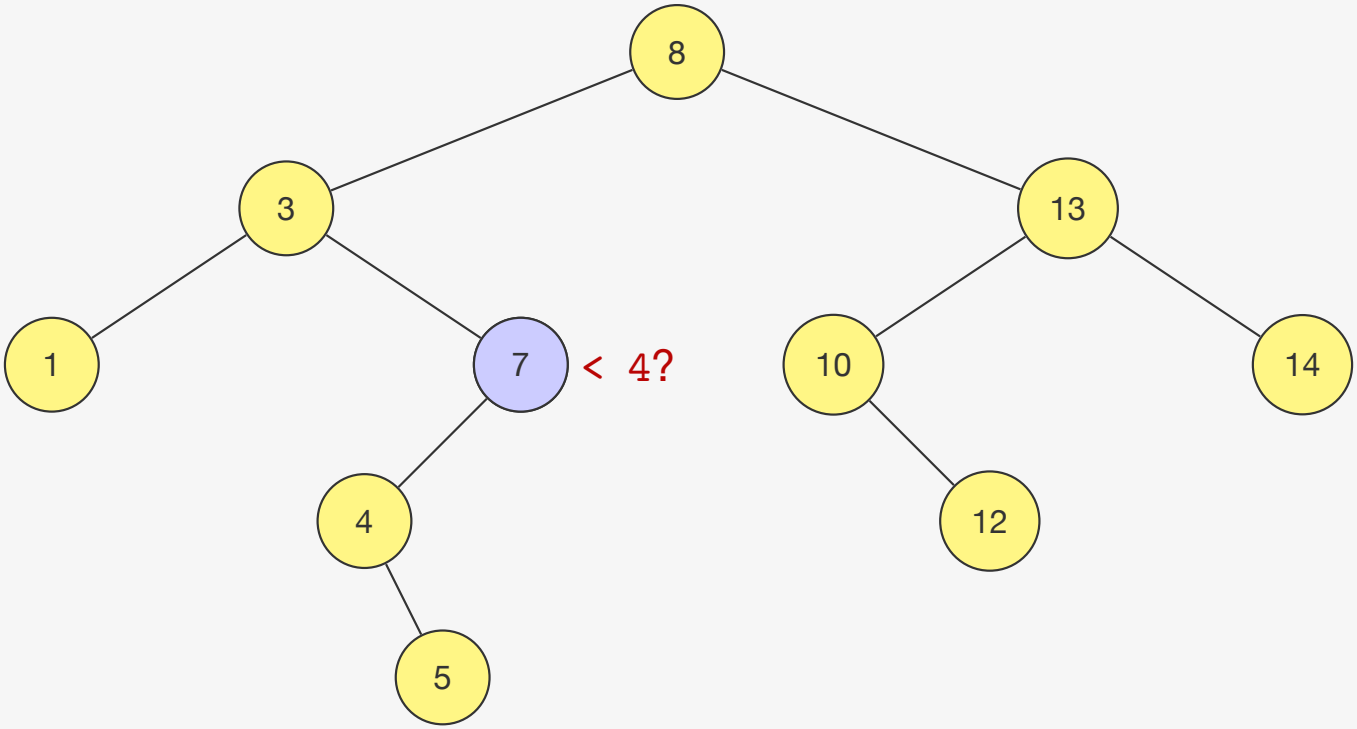


Busca por um valor

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

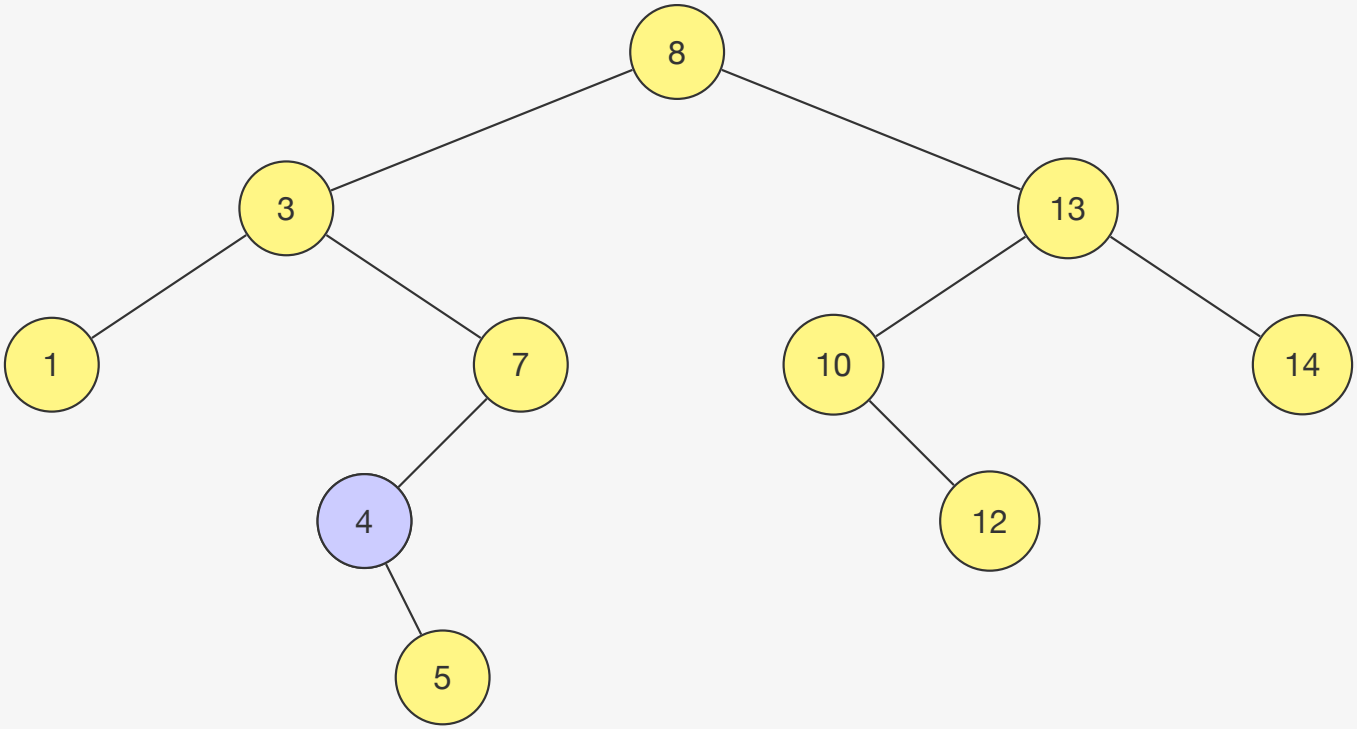


Busca por um valor

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

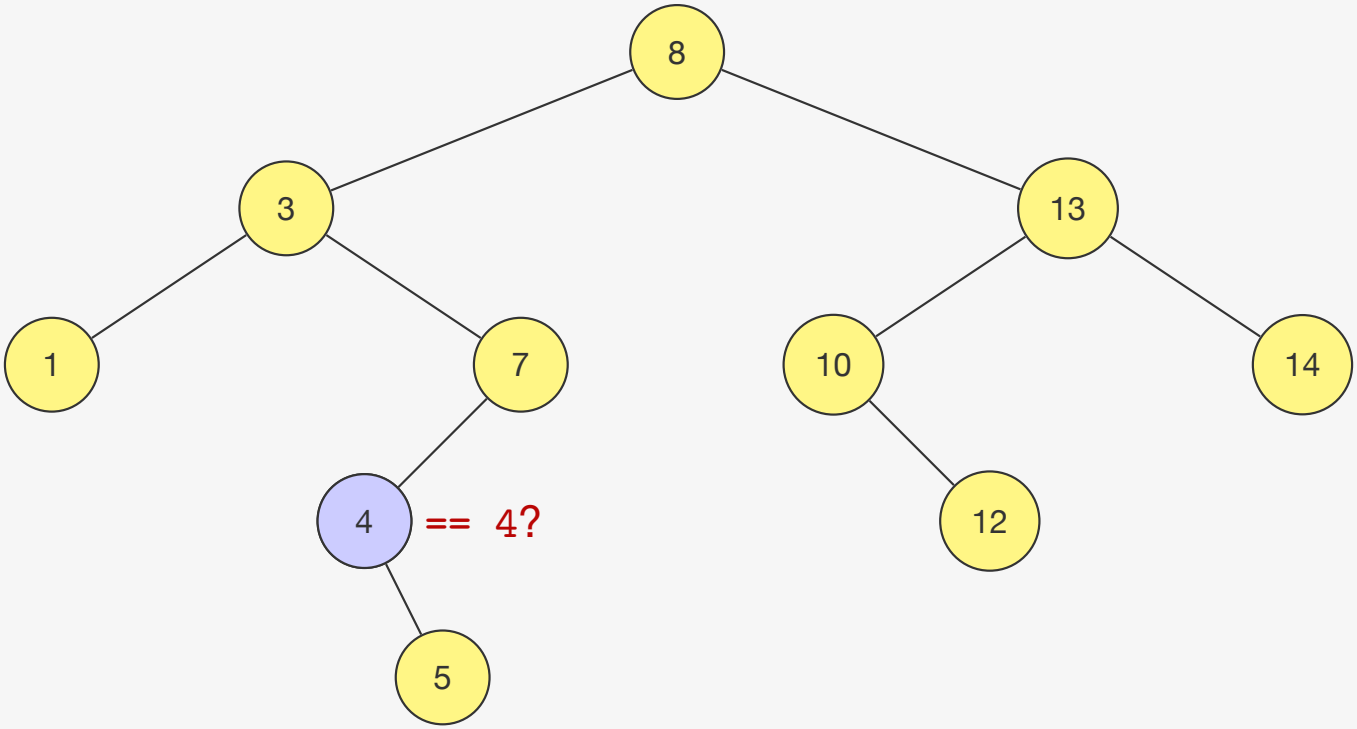


Busca por um valor

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

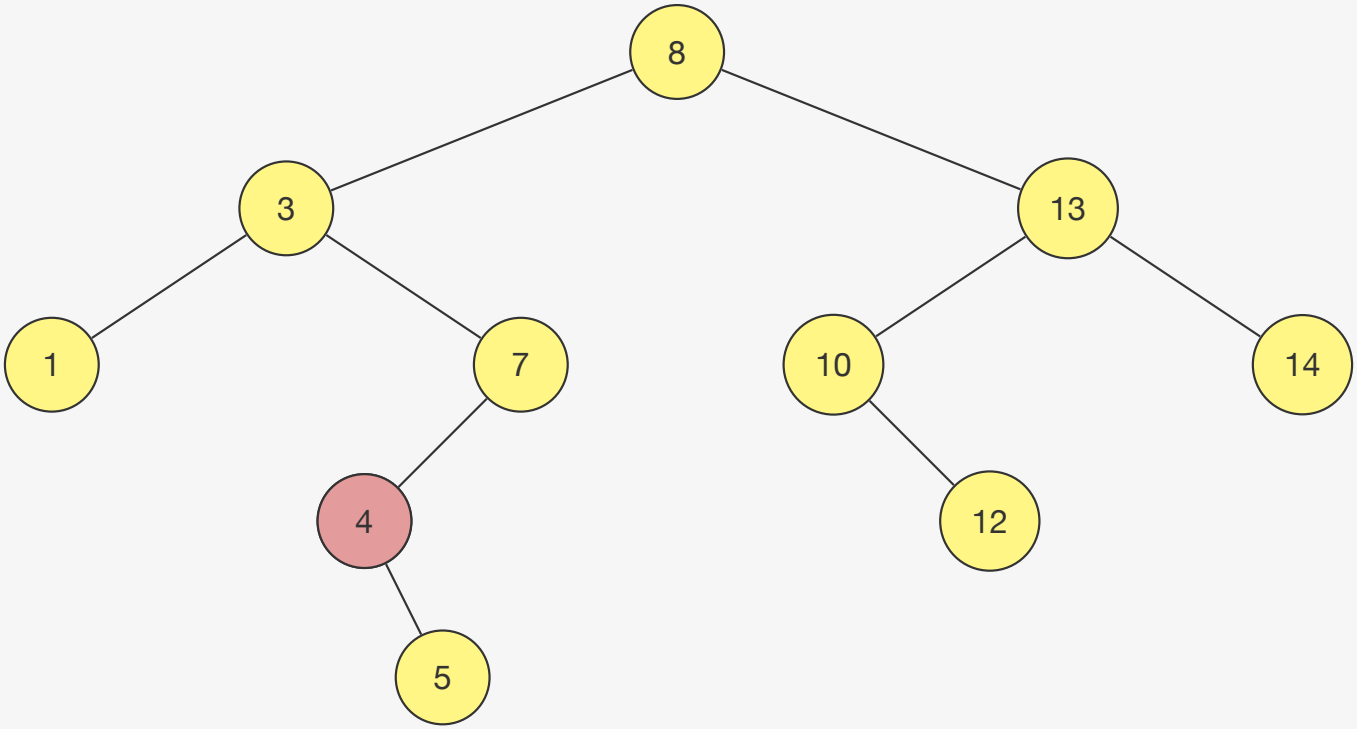


Busca por um valor

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

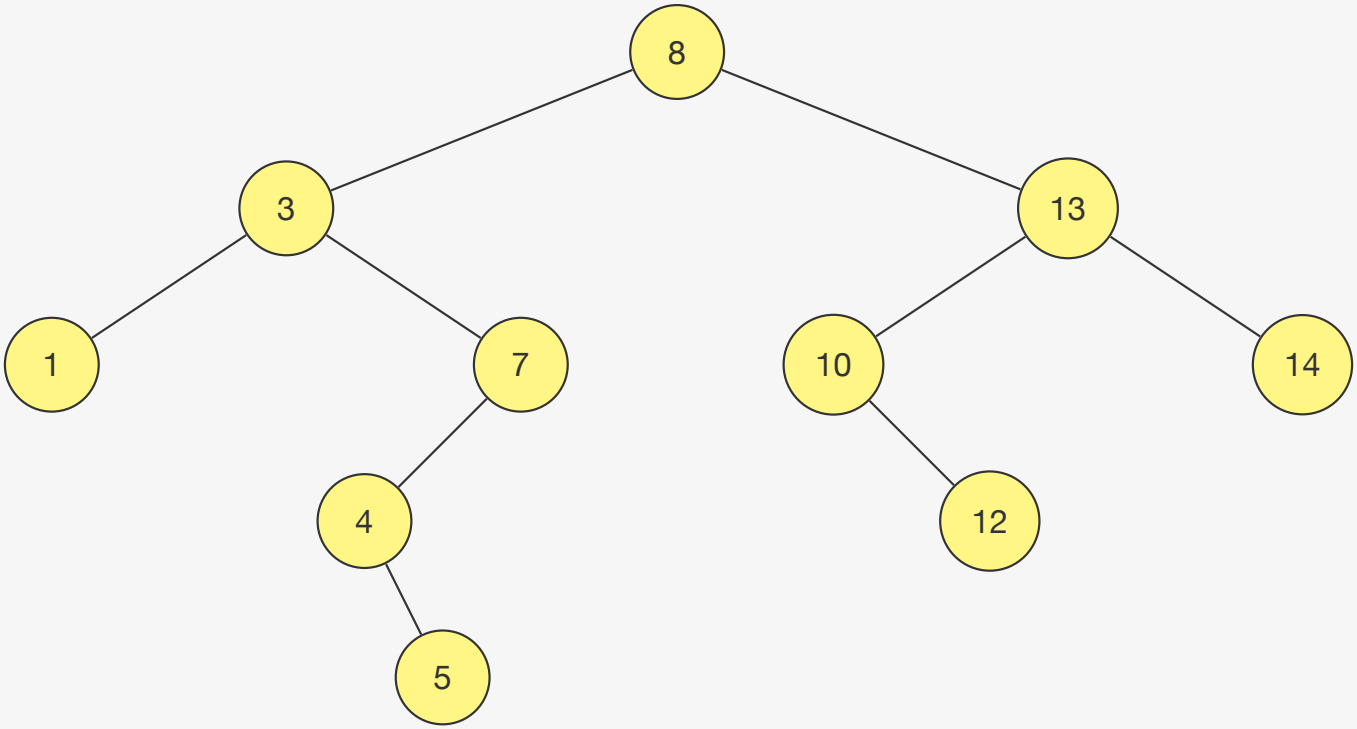


Busca por um valor

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

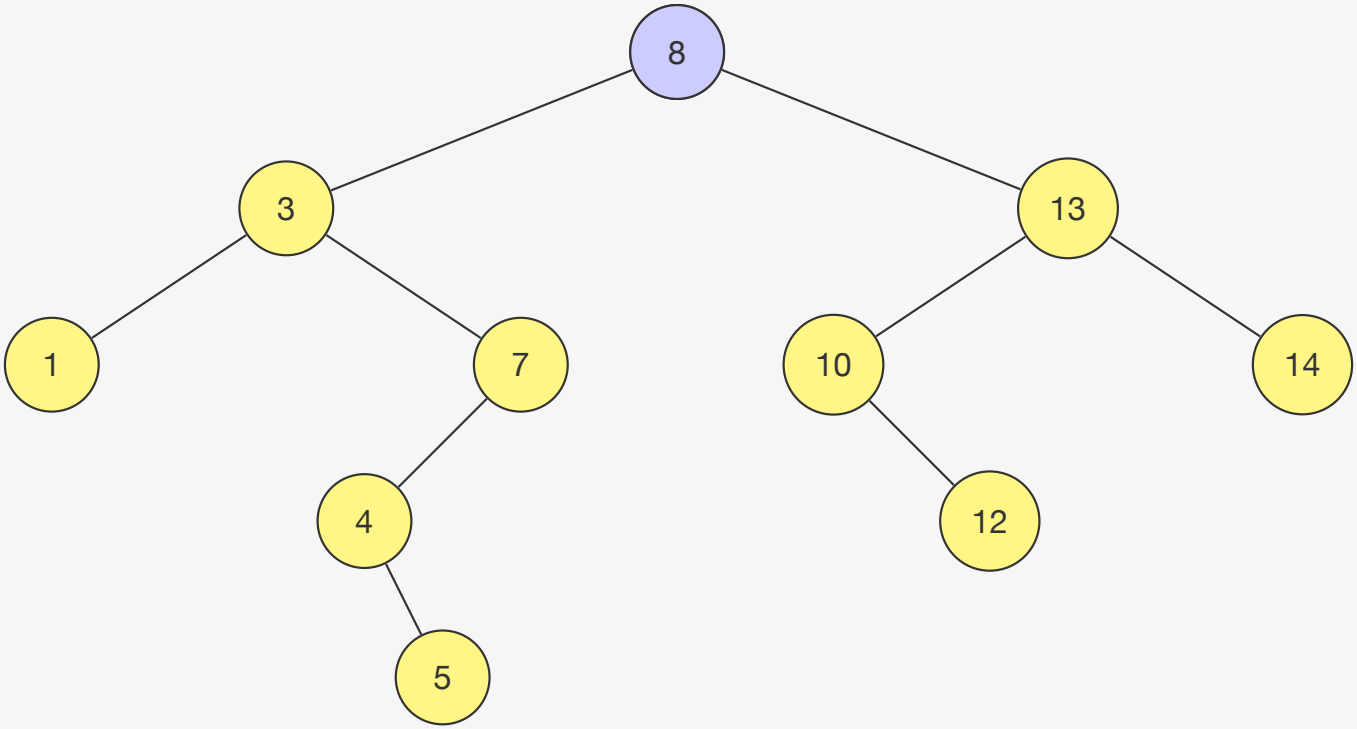


Busca por um valor

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

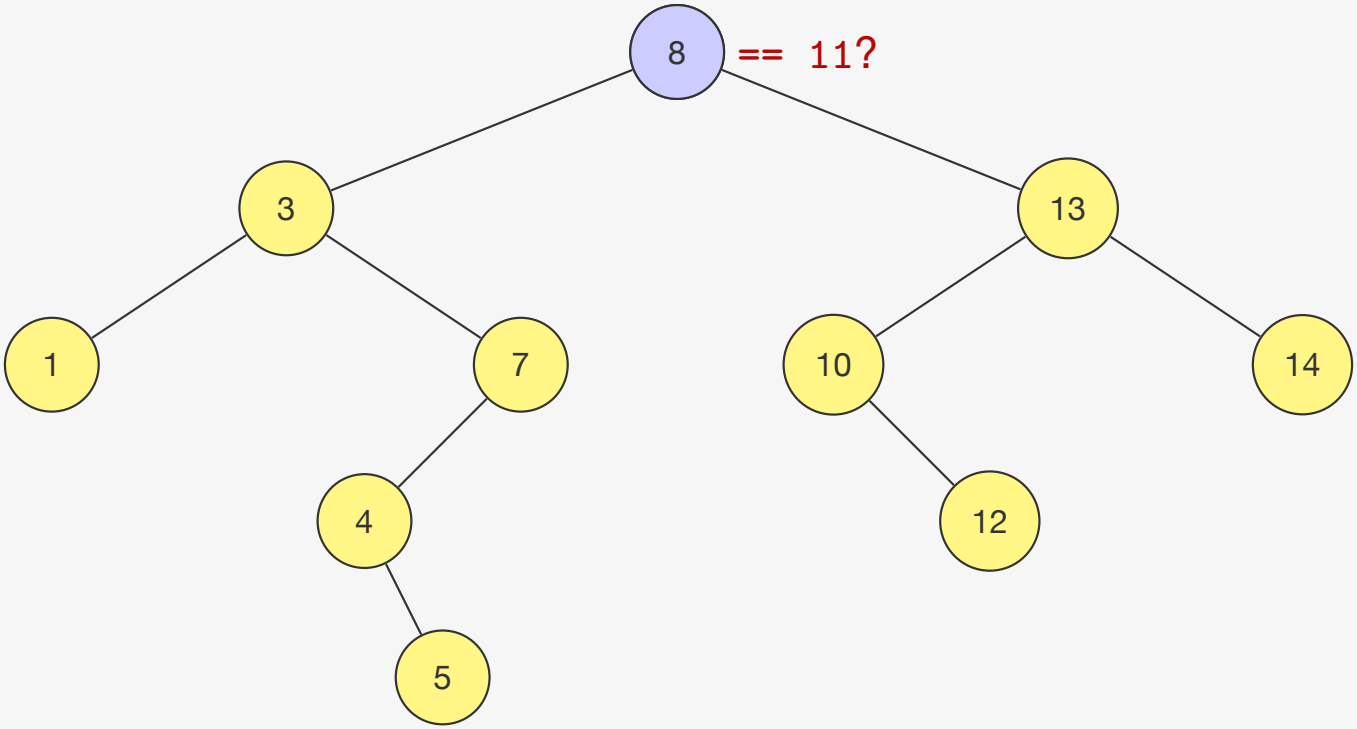


Busca por um valor

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

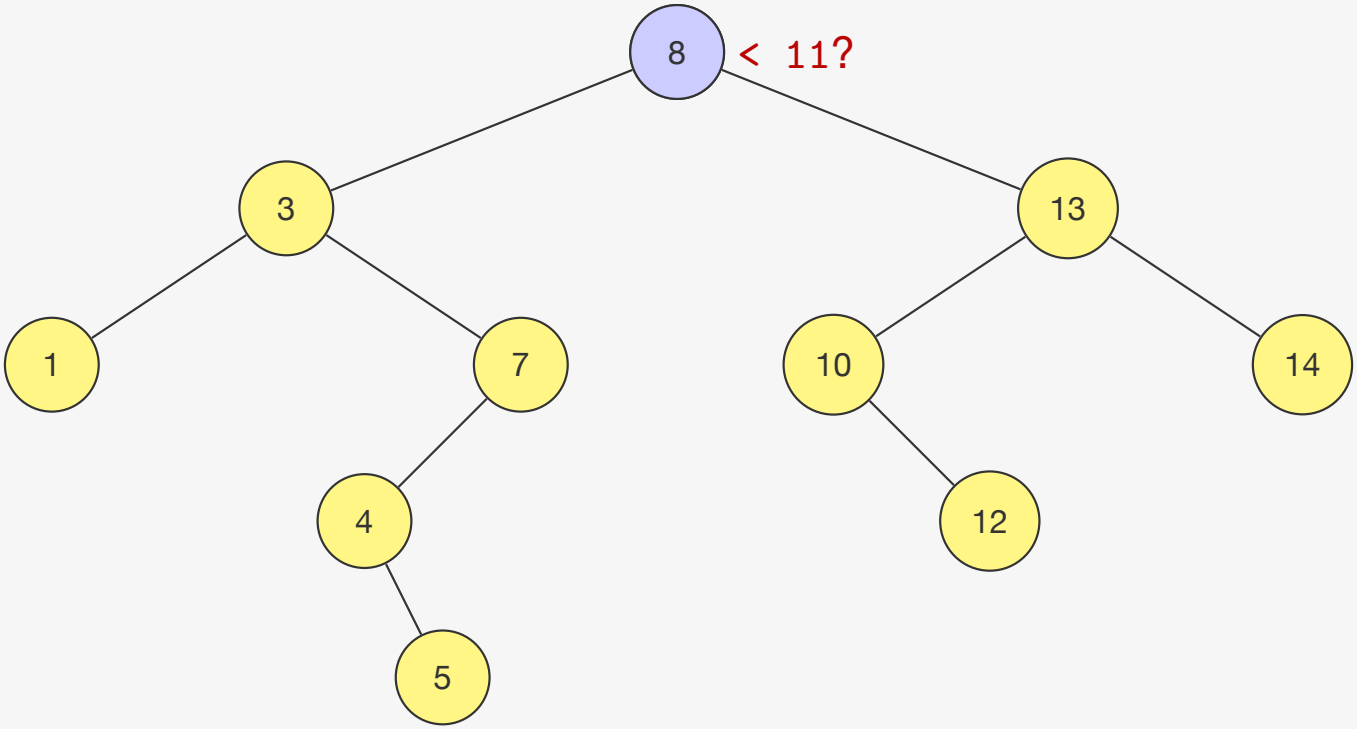


Busca por um valor

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

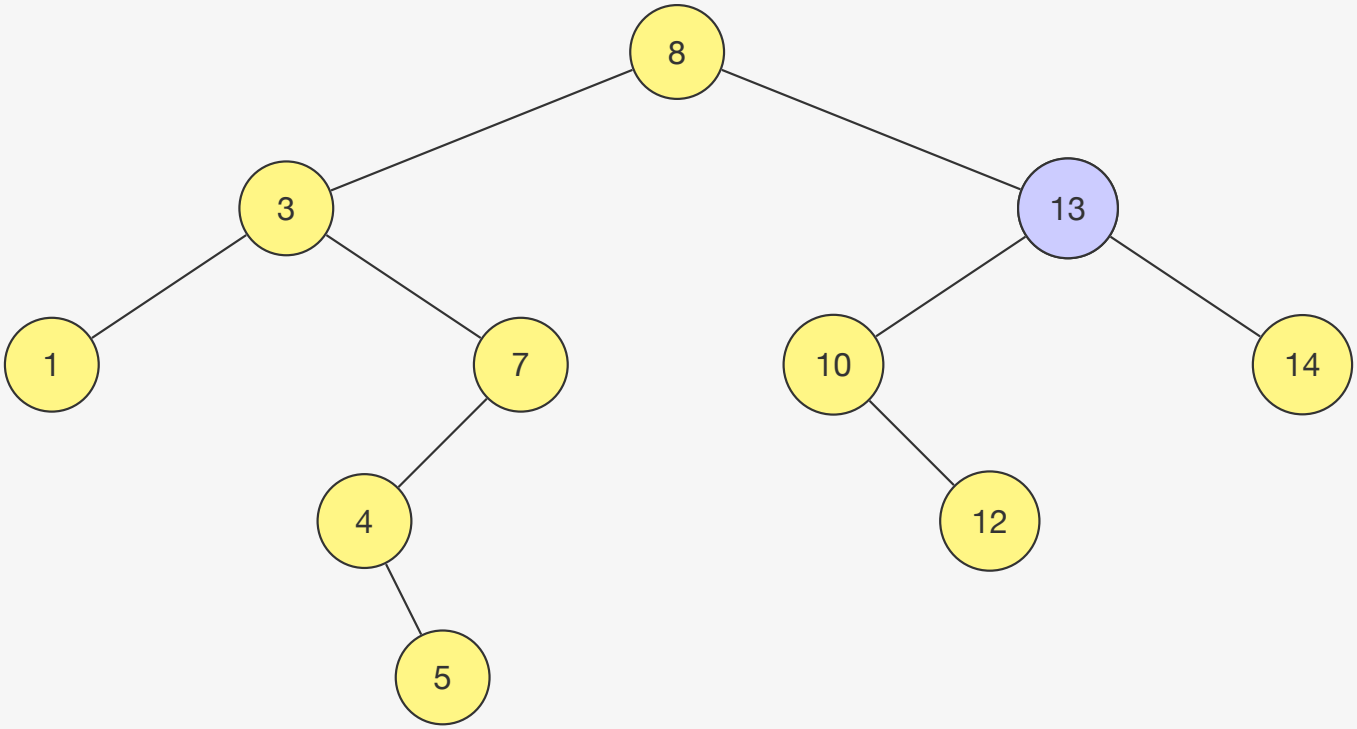


Busca por um valor

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

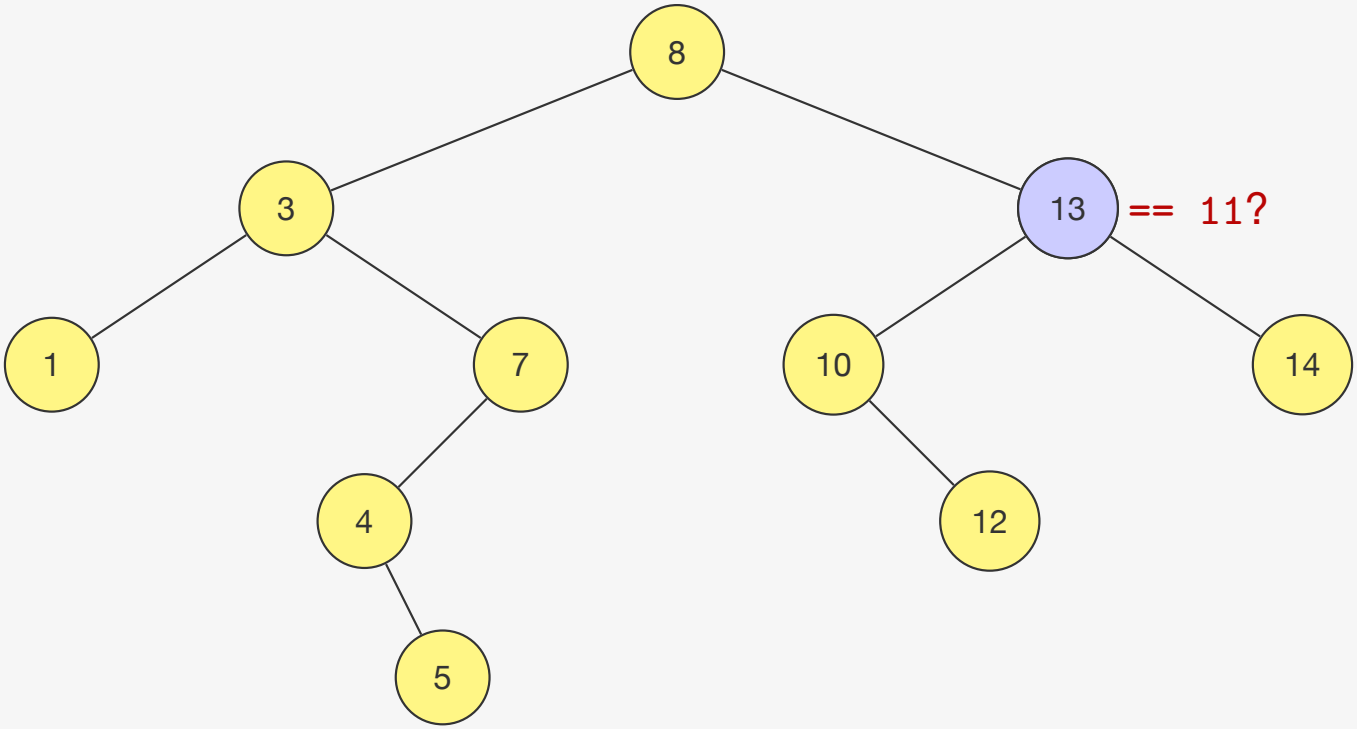


Busca por um valor

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

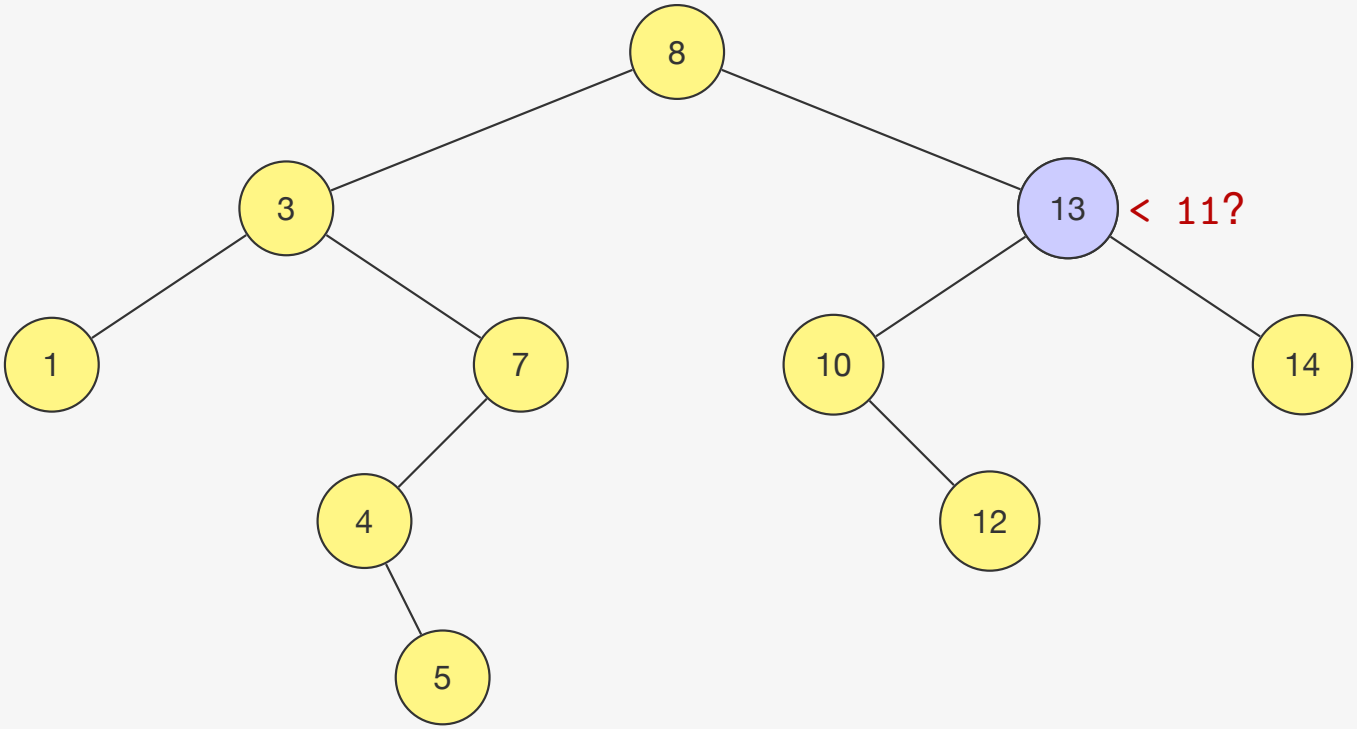


Busca por um valor

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

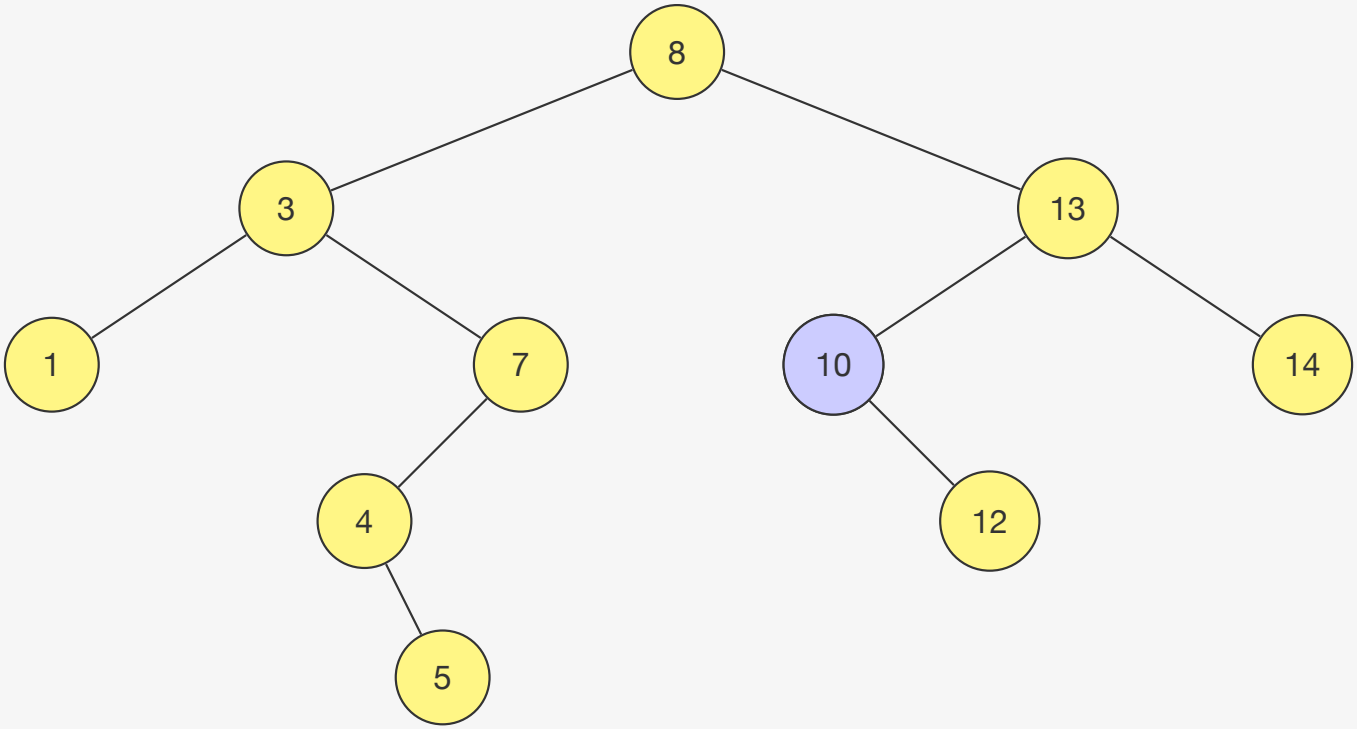


Busca por um valor

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

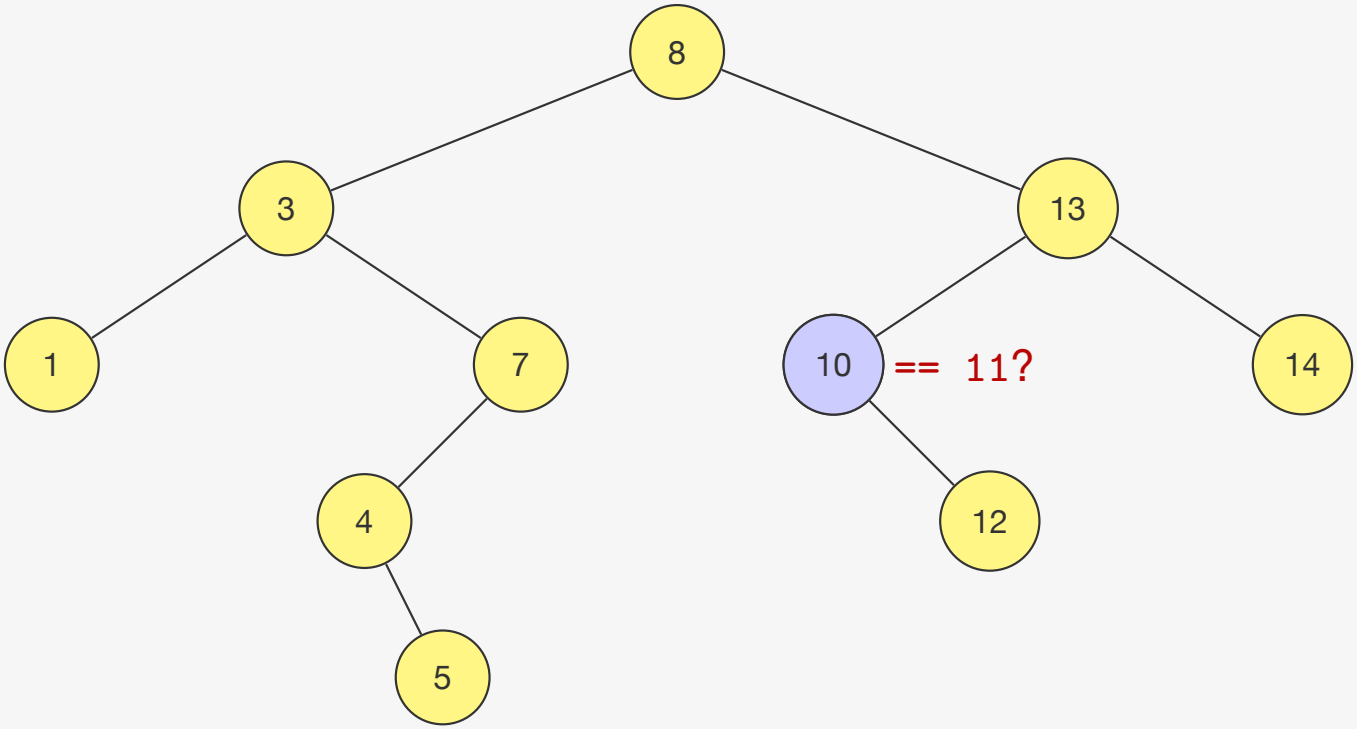


Busca por um valor

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

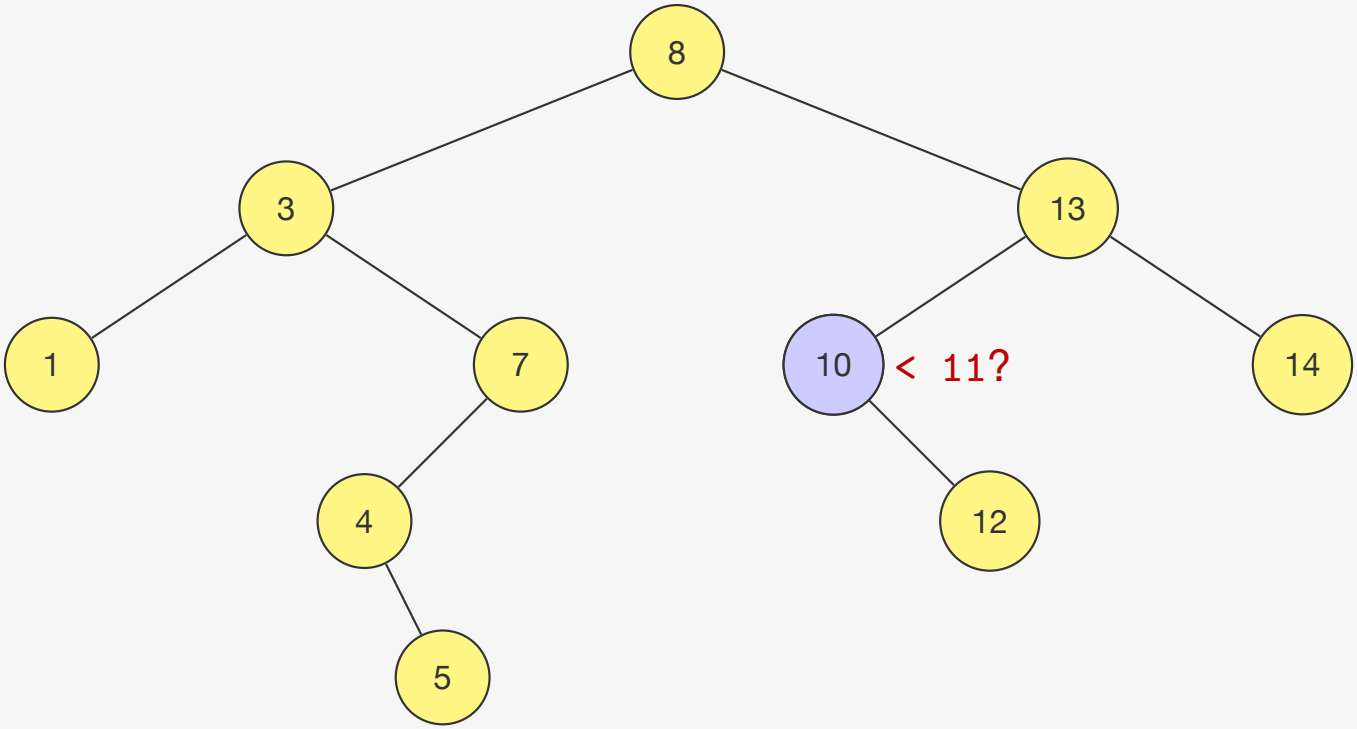


Busca por um valor

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

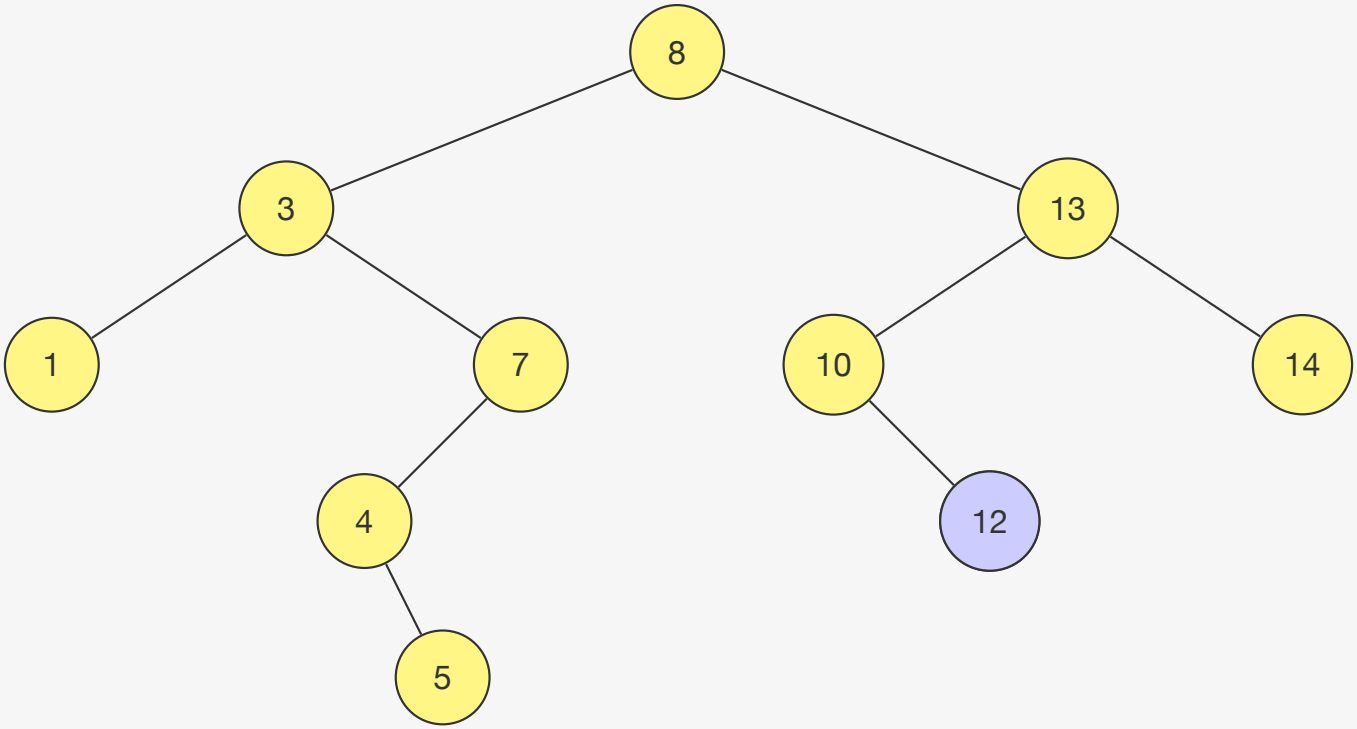


Busca por um valor

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

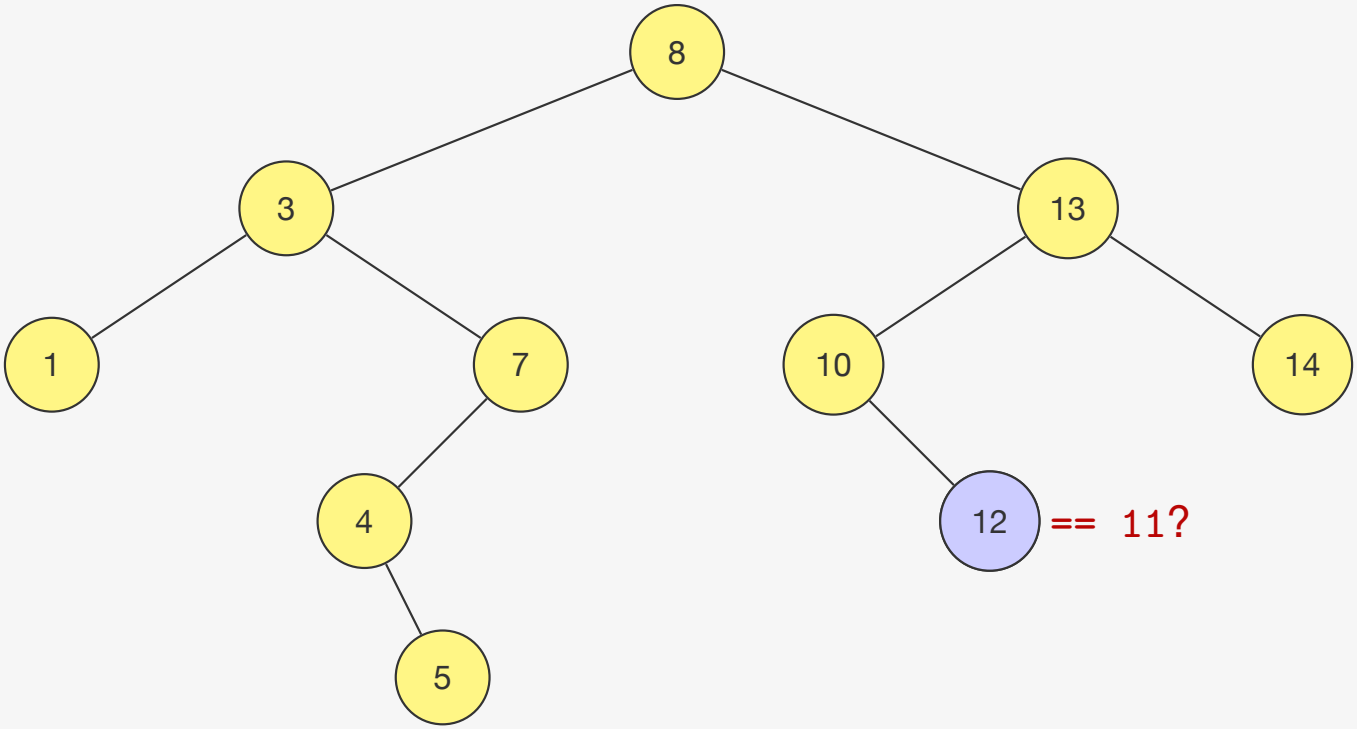


Busca por um valor

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

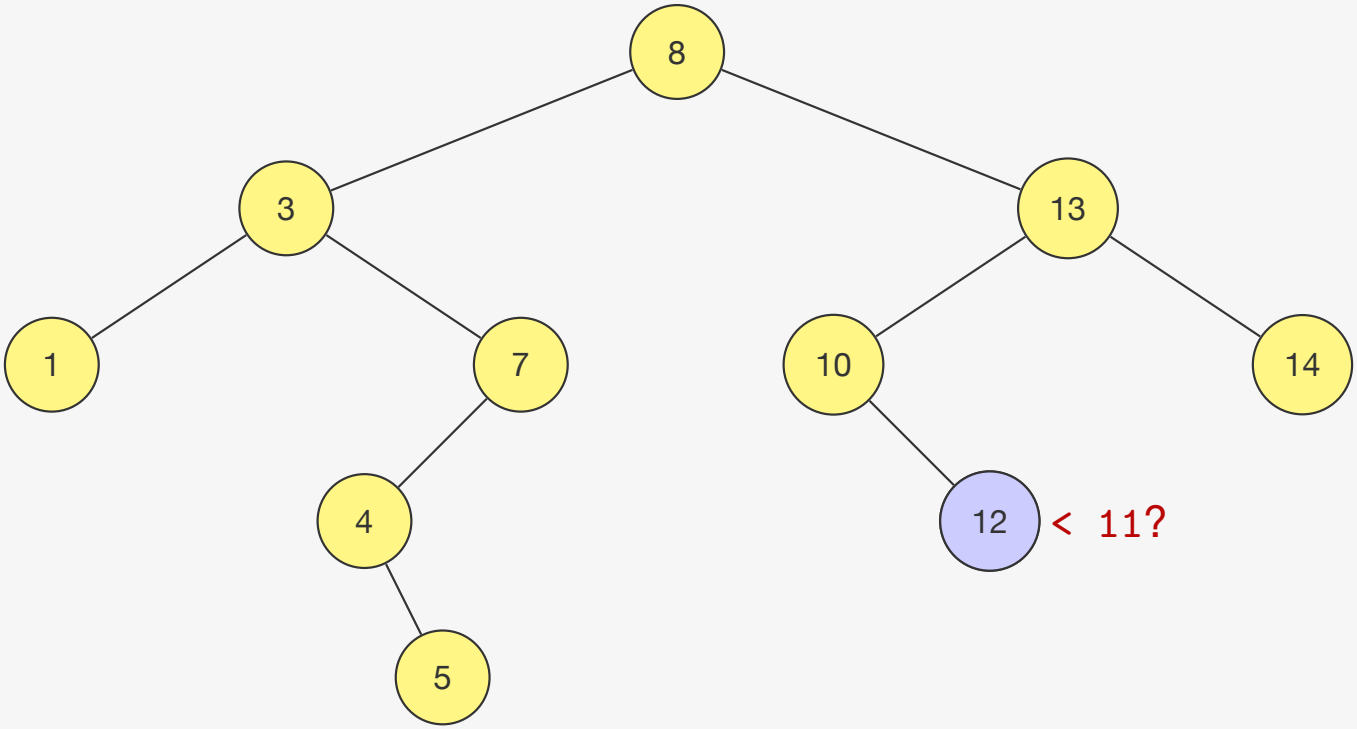


Busca por um valor

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

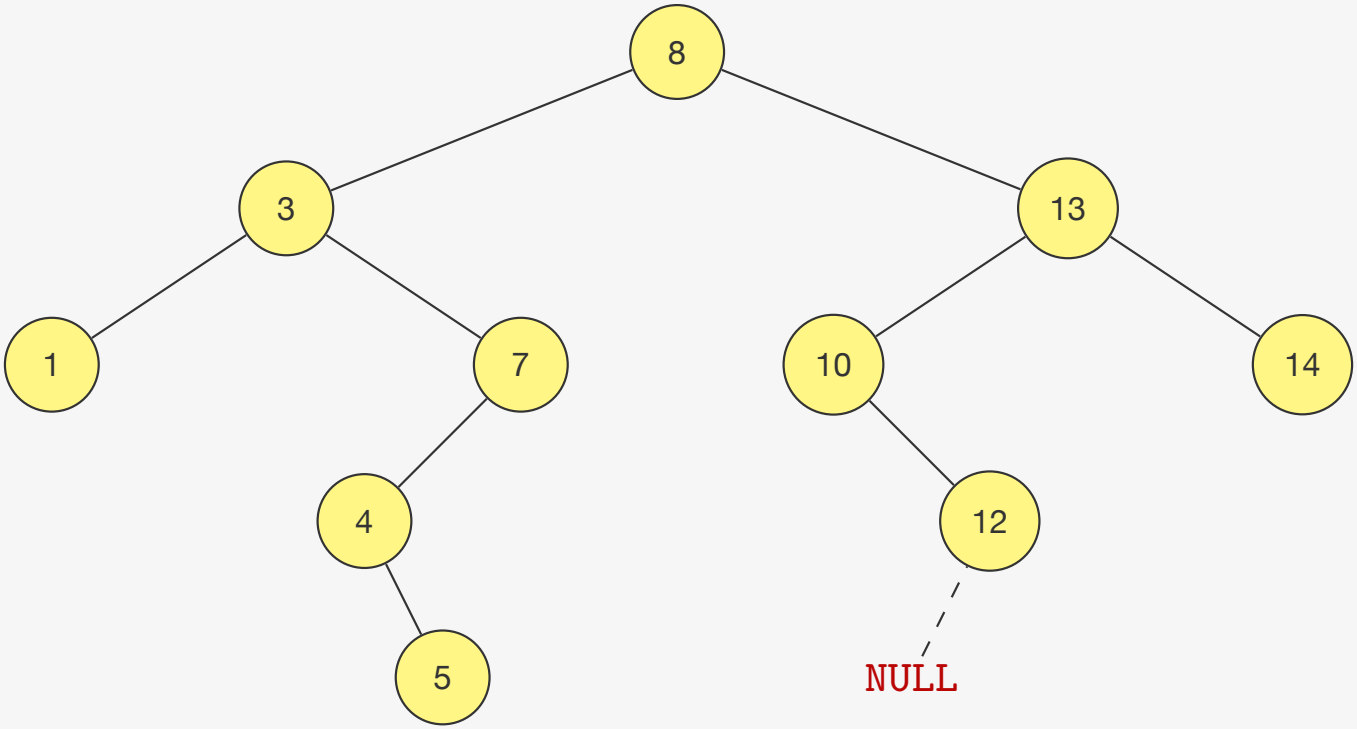


Busca por um valor

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 No* buscar(No *arvore, int chave) {
```

Busca

Versão recursiva:

```
1 No* buscar(No *arvore, int chave) {
2   if (arvore == NULL || chave == arvore->chave)
3     return arvore;
```

Busca

Versão recursiva:

```
1 No* buscar(No *arvore, int chave) {
2   if (arvore == NULL || chave == arvore->chave)
3     return arvore;
4   if (chave < arvore->chave)
5     return buscar(arvore->esq, chave);
```

Busca

Versão recursiva:

```
1 No* buscar(No *arvore, int chave) {
2   if (arvore == NULL || chave == arvore->chave)
3     return arvore;
4   if (chave < arvore->chave)
5     return buscar(arvore->esq, chave);
6   else
7     return buscar(arvore->dir, chave);
8 }
```

Busca

Versão recursiva:

```
1 No* buscar(No *arvore, int chave) {
2   if (arvore == NULL || chave == arvore->chave)
3     return arvore;
4   if (chave < arvore->chave)
5     return buscar(arvore->esq, chave);
6   else
7     return buscar(arvore->dir, chave);
8 }
```

Versão iterativa:

```
1 No* buscar_iterativo(No *arvore, int chave) {
2   while (arvore != NULL && chave != arvore->chave)
3     if (chave < arvore->chave)
4       arvore = arvore->esq;
5     else
6       arvore = arvore->dir;
7   return arvore;
8 }
```

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...

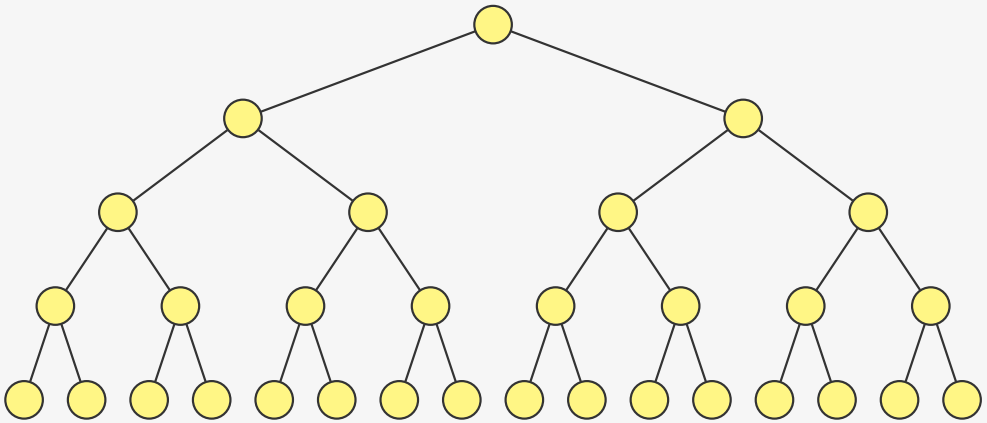
Ex: 31 nós

Eficiência da busca

Qual é o tempo da busca?

- depende da forma da árvore...

Ex: 31 nós



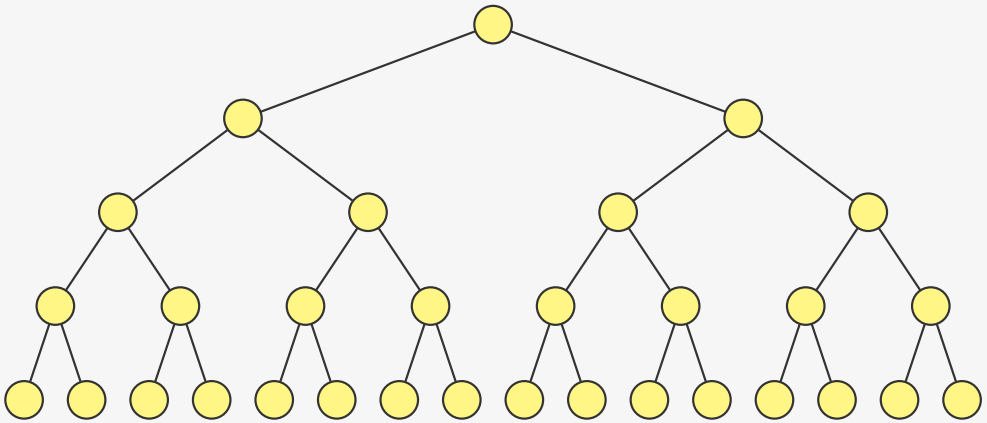
Melhor árvore: $\mathrm{O}(\lg n)$

Eficiência da busca

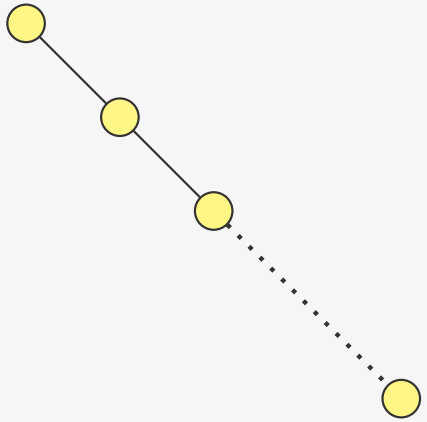
Qual é o tempo da busca?

- depende da forma da árvore...

Ex: 31 nós



Melhor árvore: $\mathrm{O}(\lg n)$



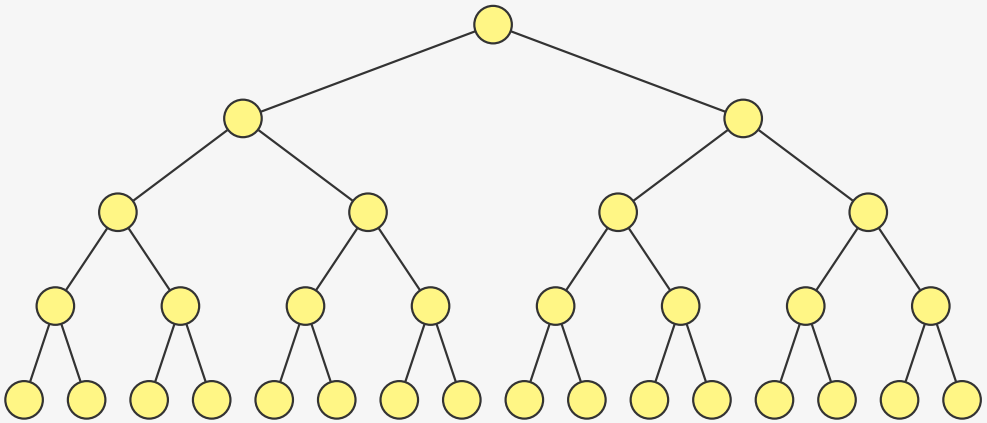
Pior árvore: $\mathrm{O}(n)$

Eficiência da busca

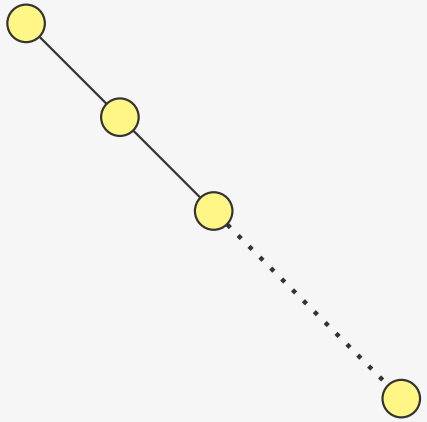
Qual é o tempo da busca?

- depende da forma da árvore...

Ex: 31 nós



Melhor árvore: $\mathrm{O}(\lg n)$



Pior árvore: $\mathrm{O}(n)$

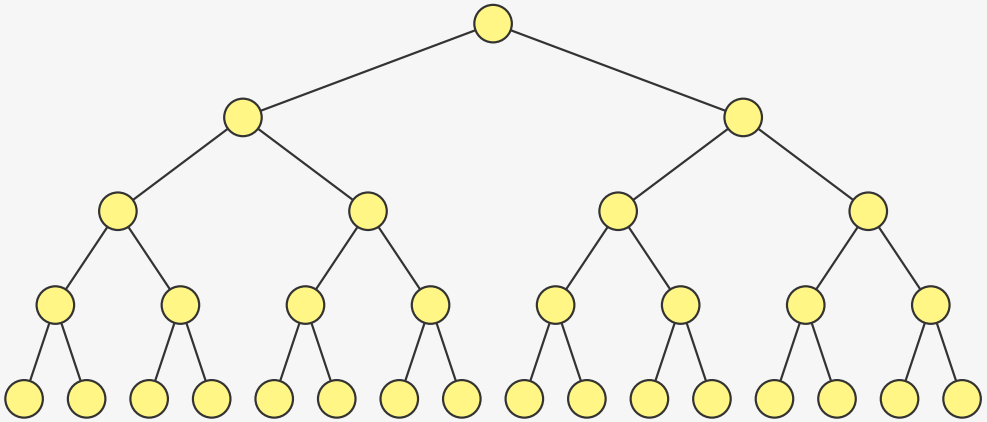
Para ter a pior árvore basta inserir em ordem crescente...

Eficiência da busca

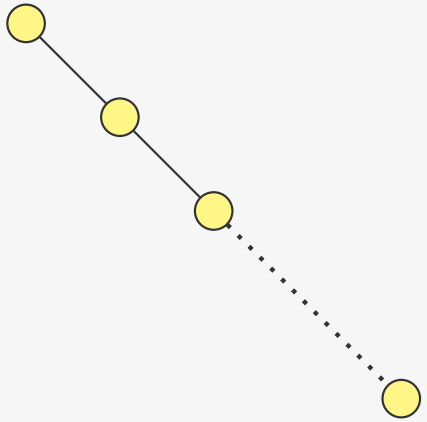
Qual é o tempo da busca?

- depende da forma da árvore...

Ex: 31 nós



Melhor árvore: $\mathrm{O}(\lg n)$



Pior árvore: $\mathrm{O}(n)$

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) $\mathrm{O}(\lg n)$

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

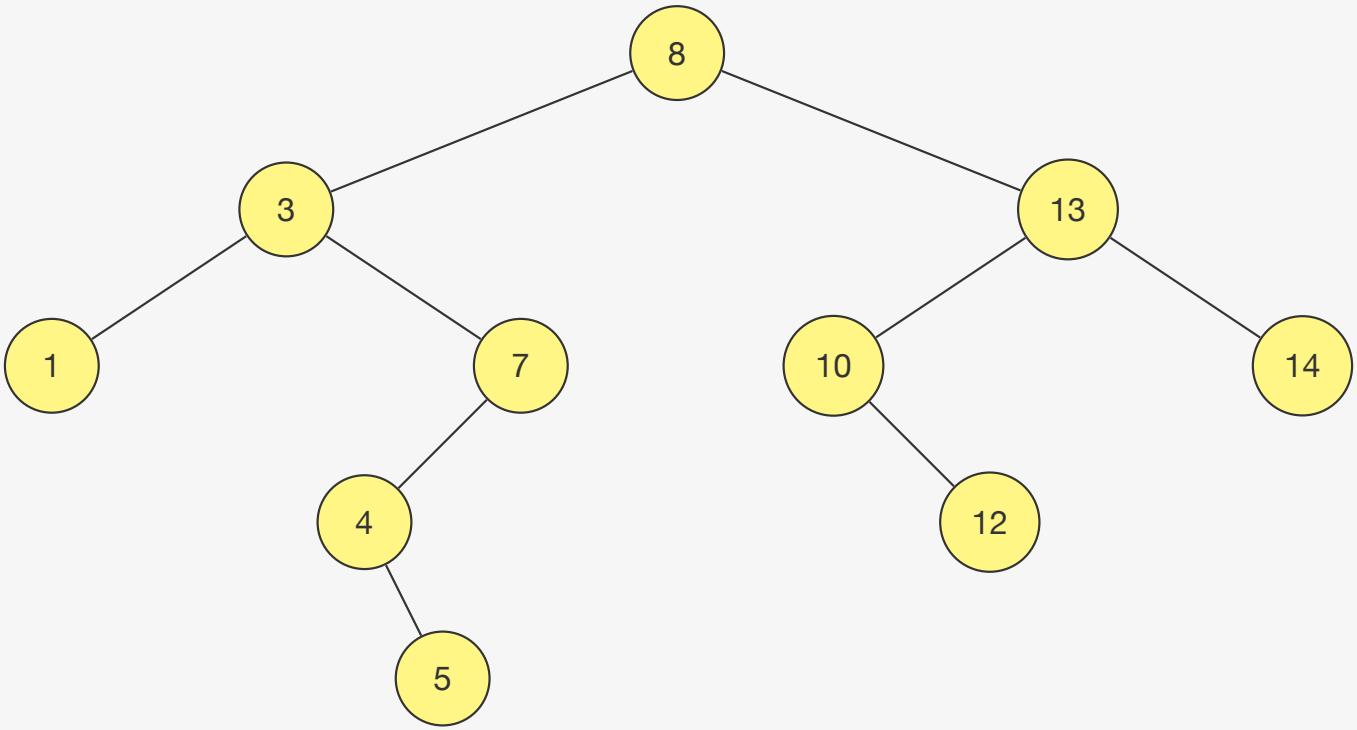
Ex: Inserindo 11

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

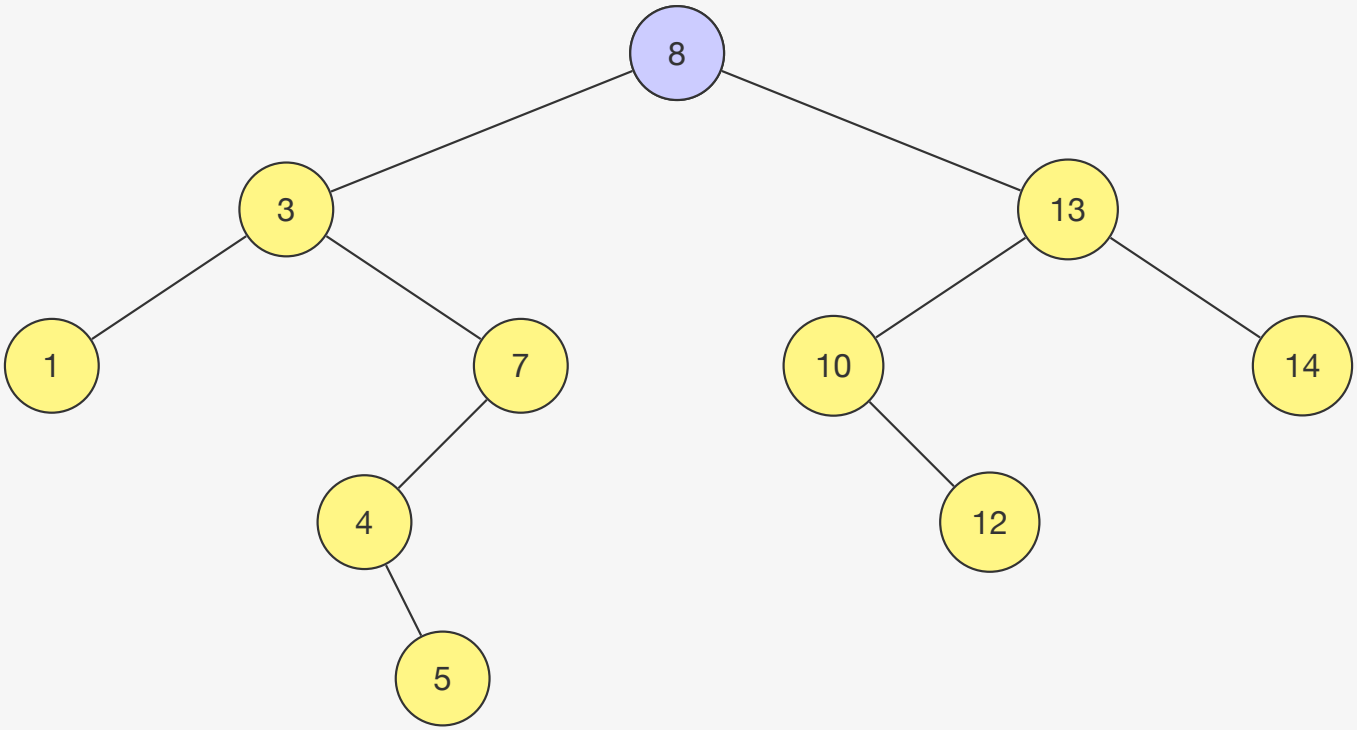


Inserindo um valor

Precisamos determinar onde inserir o valor:

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

Ex: Inserindo 11

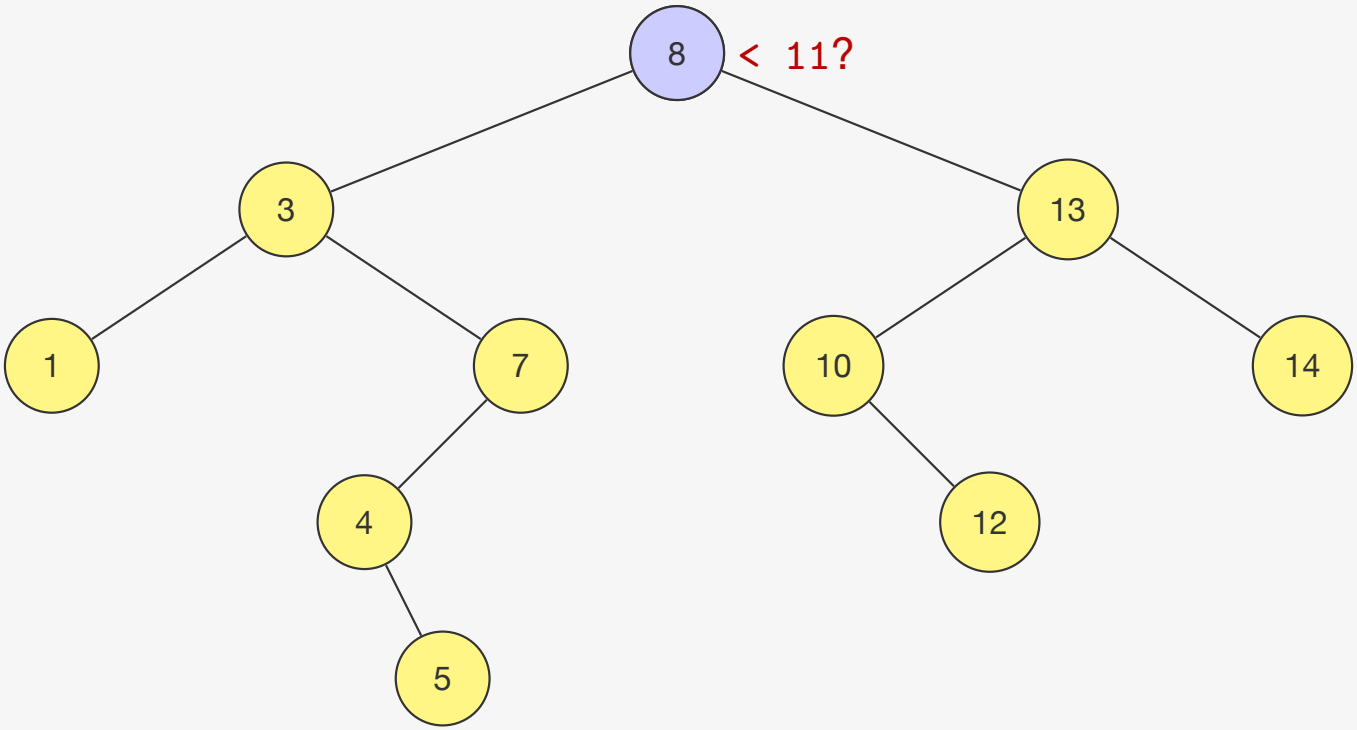


Inserindo um valor

Precisamos determinar onde inserir o valor:

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

Ex: Inserindo 11

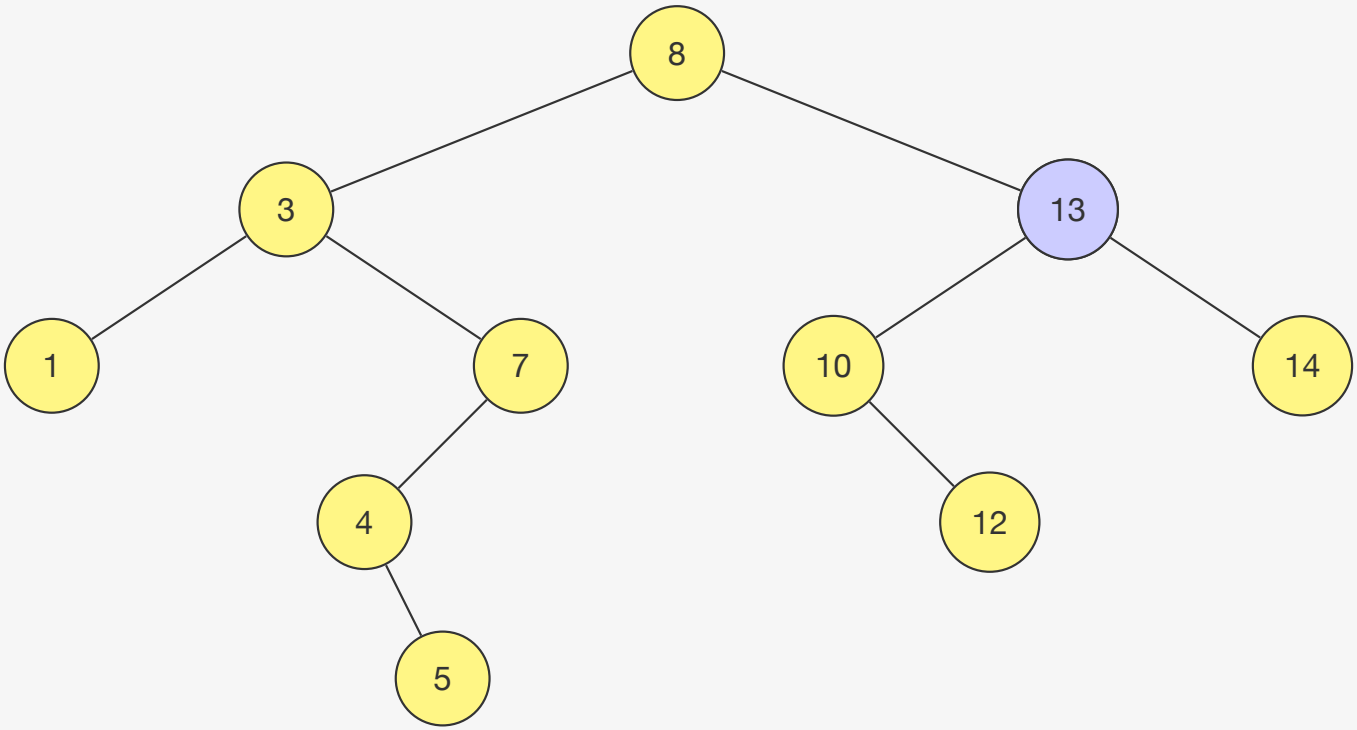


Inserindo um valor

Precisamos determinar onde inserir o valor:

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

Ex: Inserindo 11

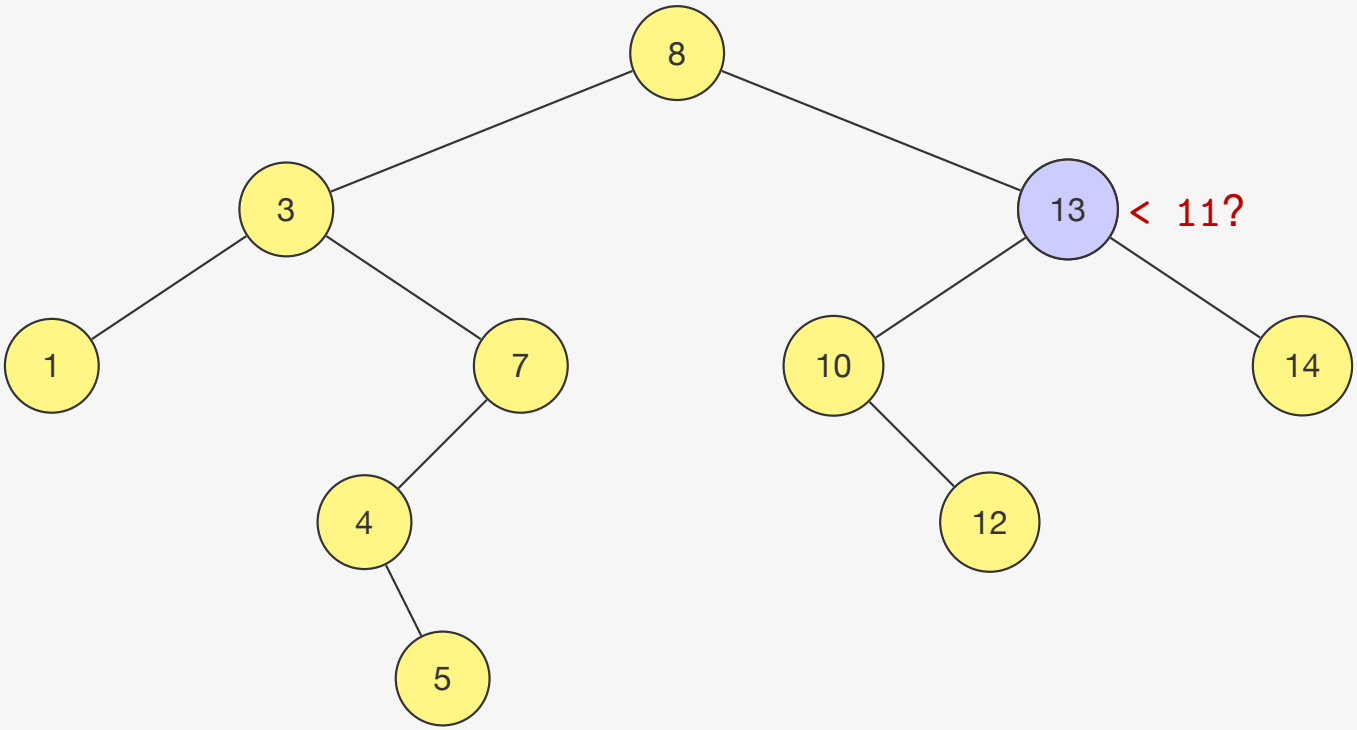


Inserindo um valor

Precisamos determinar onde inserir o valor:

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

Ex: Inserindo 11

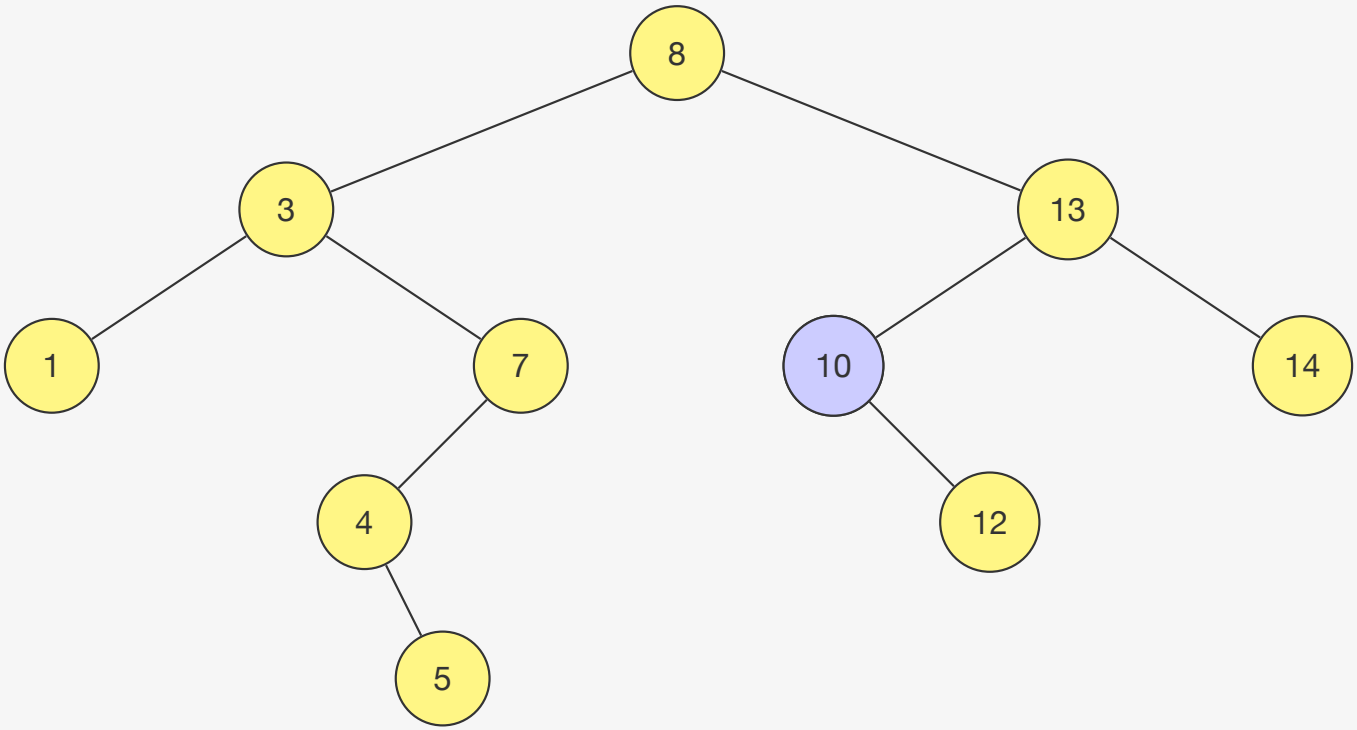


Inserindo um valor

Precisamos determinar onde inserir o valor:

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

Ex: Inserindo 11

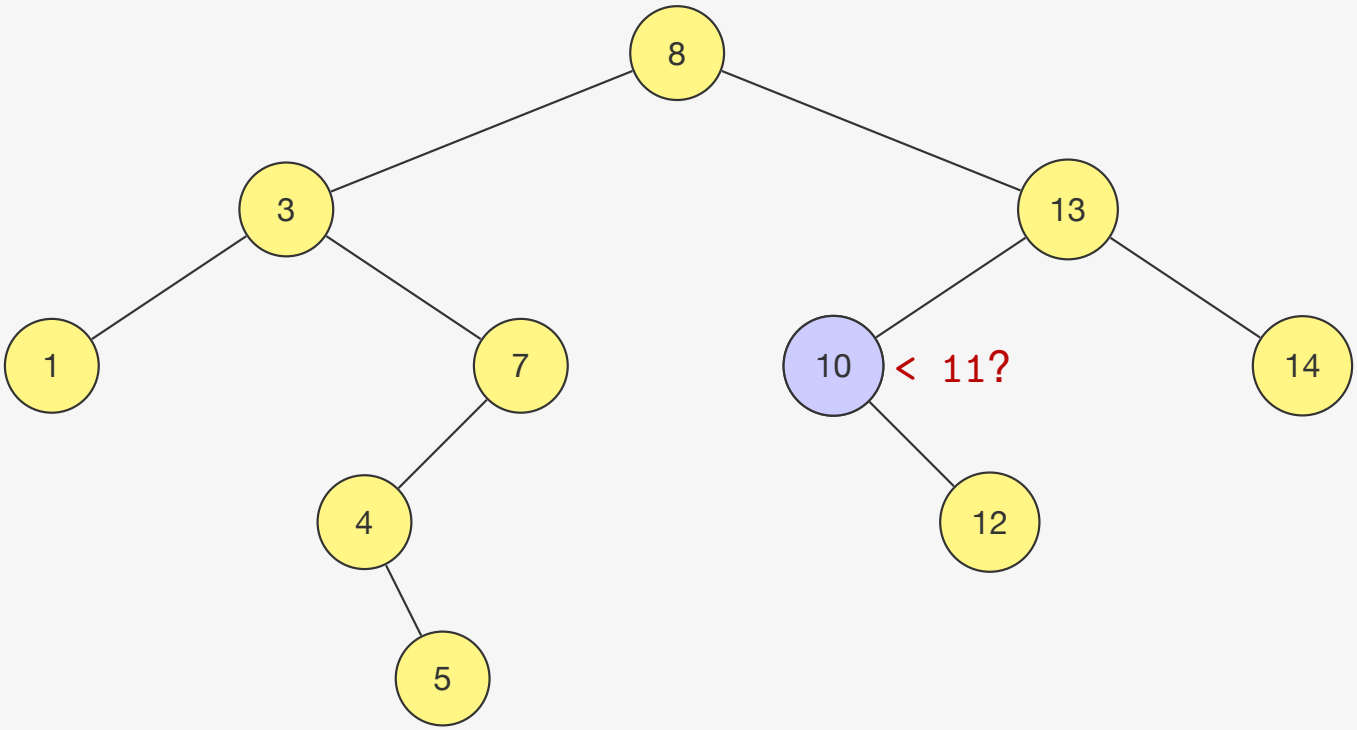


Inserindo um valor

Precisamos determinar onde inserir o valor:

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

Ex: Inserindo 11

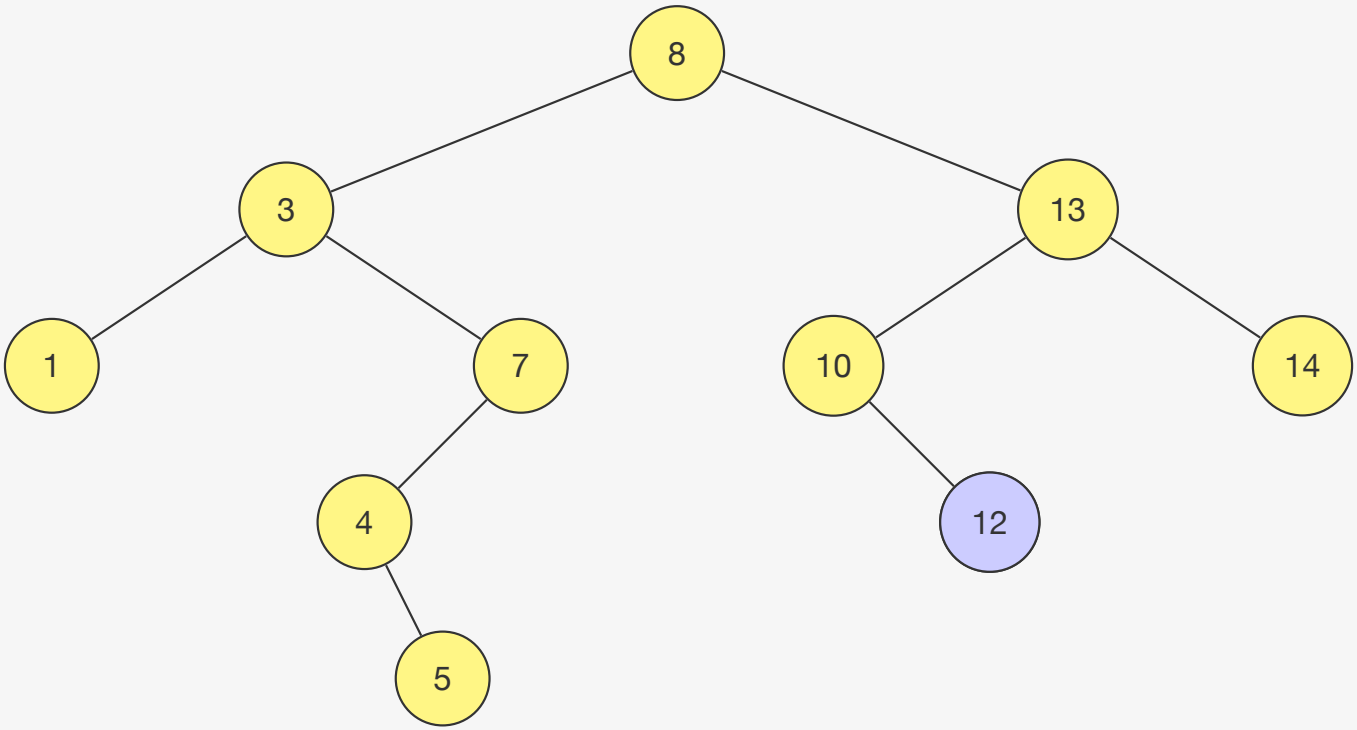


Inserindo um valor

Precisamos determinar onde inserir o valor:

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

Ex: Inserindo 11

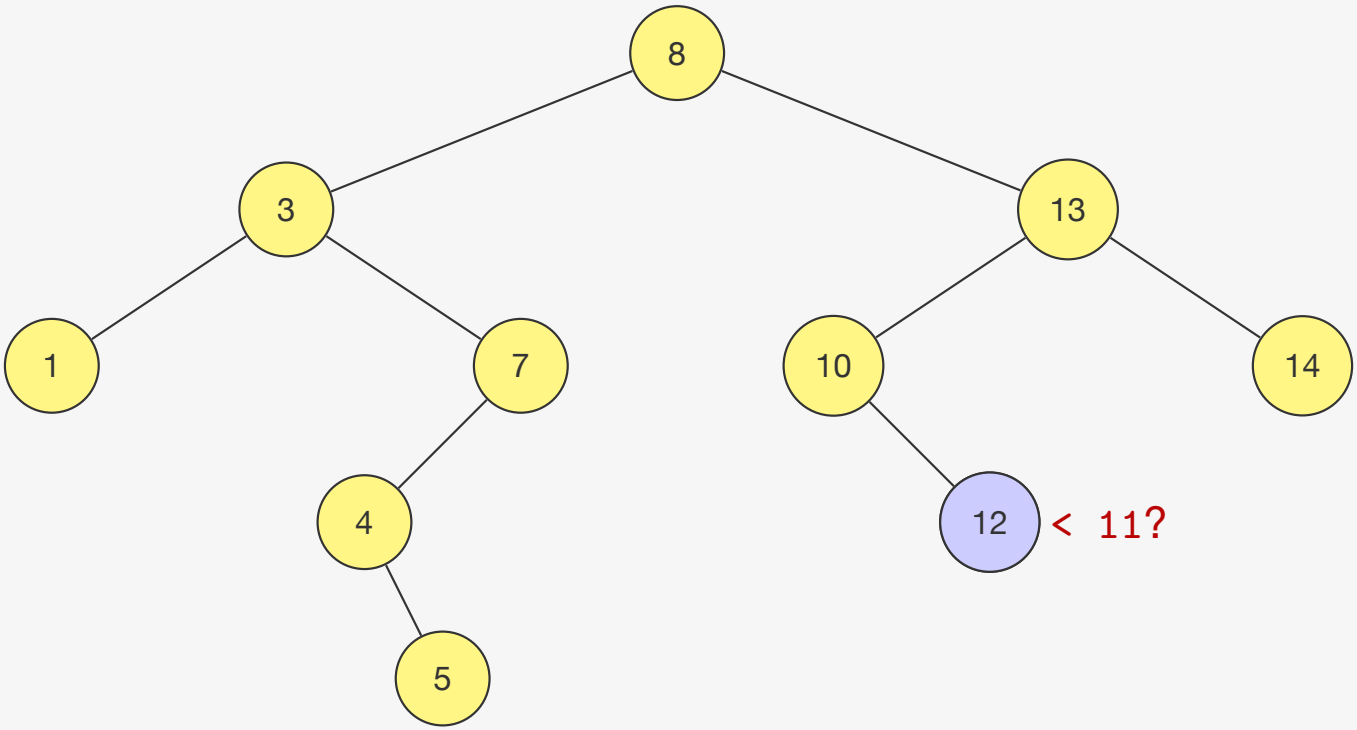


Inserindo um valor

Precisamos determinar onde inserir o valor:

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

Ex: Inserindo 11

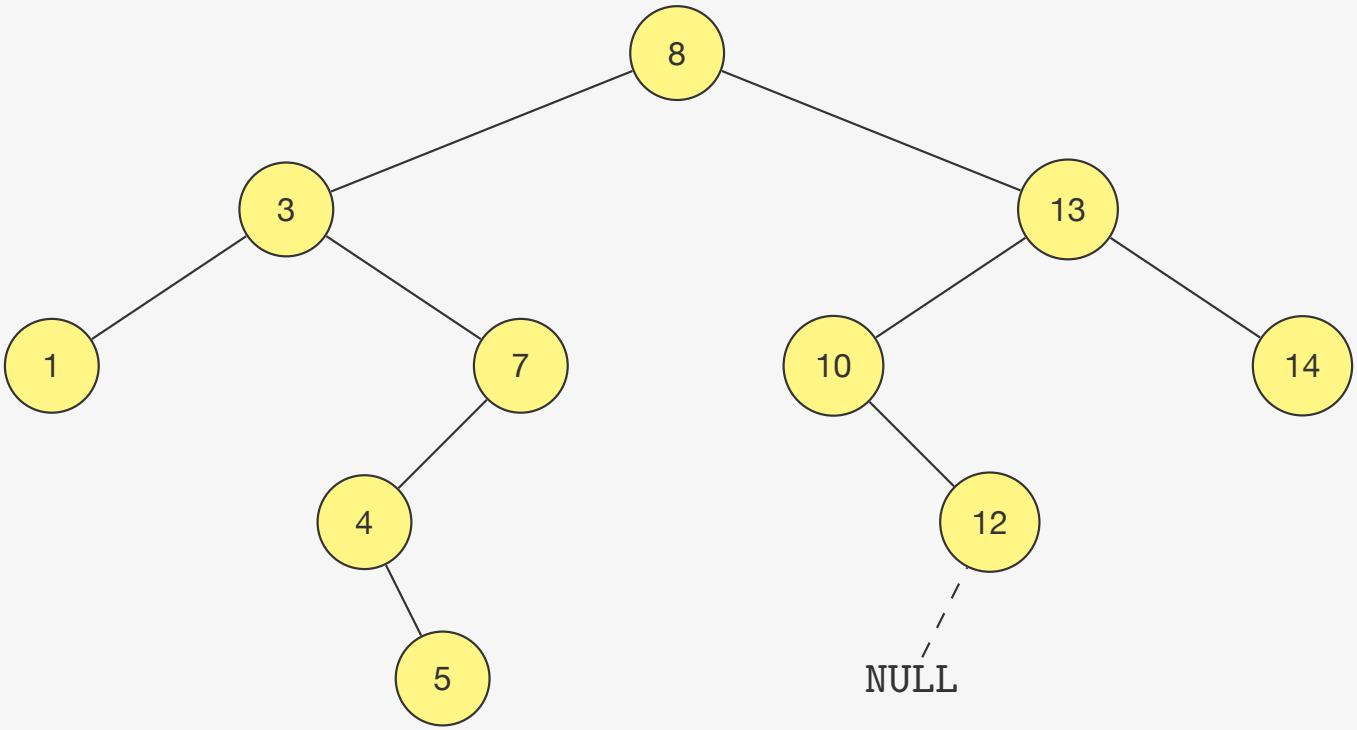


Inserindo um valor

Precisamos determinar onde inserir o valor:

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

Ex: Inserindo 11

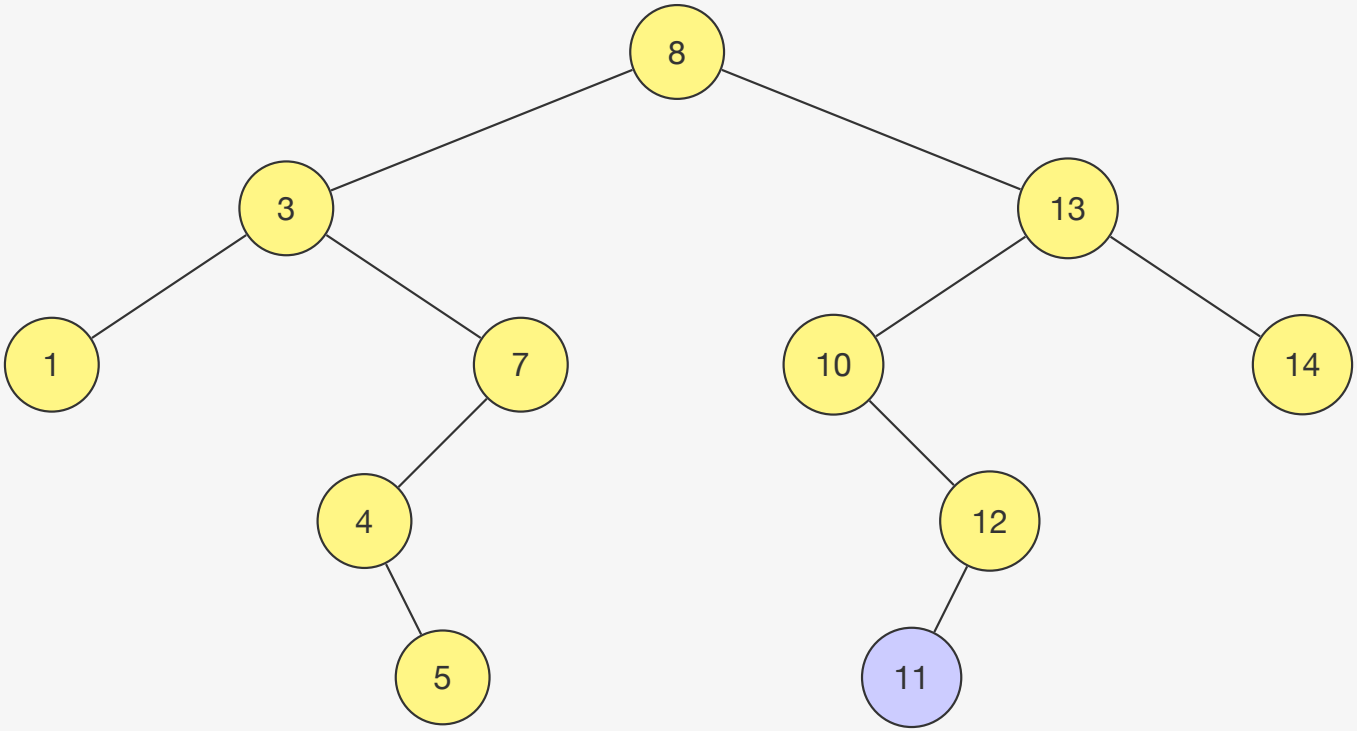


Inserindo um valor

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

O algoritmo insere na árvore recursivamente

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

```
1 No* inserir_rec(No *arvore, int chave) {
```

Inserção - implementação

O algoritmo insere na árvore recursivamente

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

```
1 No* inserir_rec(No *arvore, int chave) {
2   No *novo;
3   if (arvore == NULL) {
4     novo = malloc(sizeof(No));
5     novo->esq = novo->dir = NULL;
6     novo->chave = chave;
7     return novo;
8   }
```

Inserção - implementação

O algoritmo insere na árvore recursivamente

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

```
1  No* inserir_rec(No *arvore, int chave) {
2    No *novo;
3    if (arvore == NULL) {
4      novo = malloc(sizeof(No));
5      novo->esq = novo->dir = NULL;
6      novo->chave = chave;
7      return novo;
8    }
9    if (chave < arvore->chave)
10     arvore->esq = inserir_rec(arvore->esq, chave);
11   else
12     arvore->dir = inserir_rec(arvore->dir, chave);
13   return arvore;
14 }
```

Inserção - implementação

O algoritmo insere na árvore recursivamente

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

```
1  No* inserir_rec(No *arvore, int chave) {
2    No *novo;
3    if (arvore == NULL) {
4      novo = malloc(sizeof(No));
5      novo->esq = novo->dir = NULL;
6      novo->chave = chave;
7      return novo;
8    }
9    if (chave < arvore->chave)
10     arvore->esq = inserir_rec(arvore->esq, chave);
11   else
12     arvore->dir = inserir_rec(arvore->dir, chave);
13   return arvore;
14 }
15
16 void inserir(No **arvore, int chave) {
17   *arvore = inserir_rec(*arvore, chave);
18 }
```

Inserção - implementação

O algoritmo insere na árvore recursivamente

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

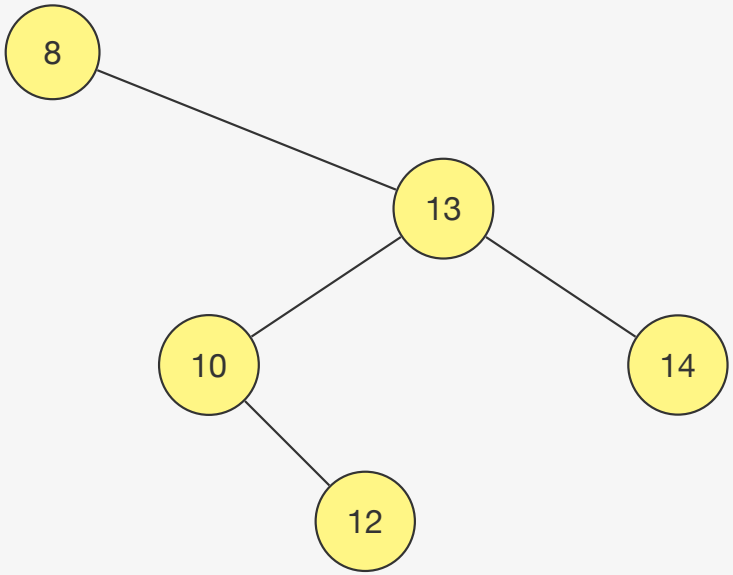
```
1  No* inserir_rec(No *arvore, int chave) {
2    No *novo;
3    if (arvore == NULL) {
4      novo = malloc(sizeof(No));
5      novo->esq = novo->dir = NULL;
6      novo->chave = chave;
7      return novo;
8    }
9    if (chave < arvore->chave)
10     arvore->esq = inserir_rec(arvore->esq, chave);
11   else
12     arvore->dir = inserir_rec(arvore->dir, chave);
13   return arvore;
14 }
15
16 void inserir(No **arvore, int chave) {
17   *arvore = inserir_rec(*arvore, chave);
18 }
```

Mínimo da Árvore

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

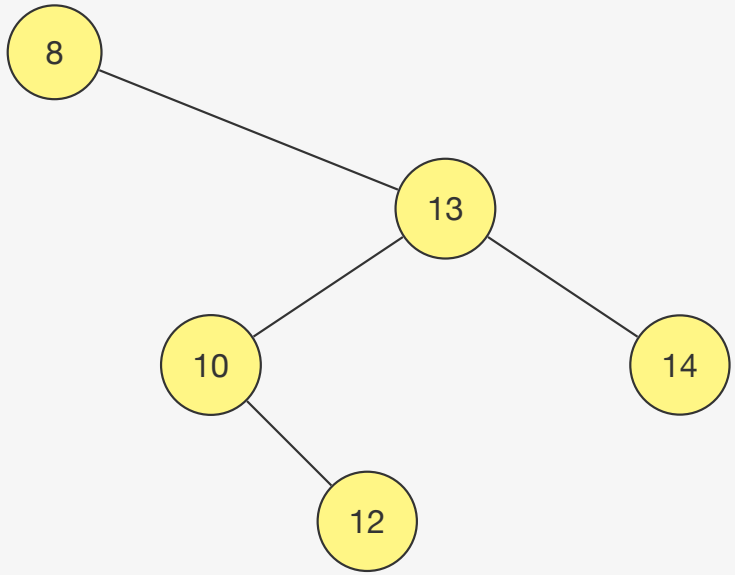
Mínimo da Árvore

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



Mínimo da Árvore

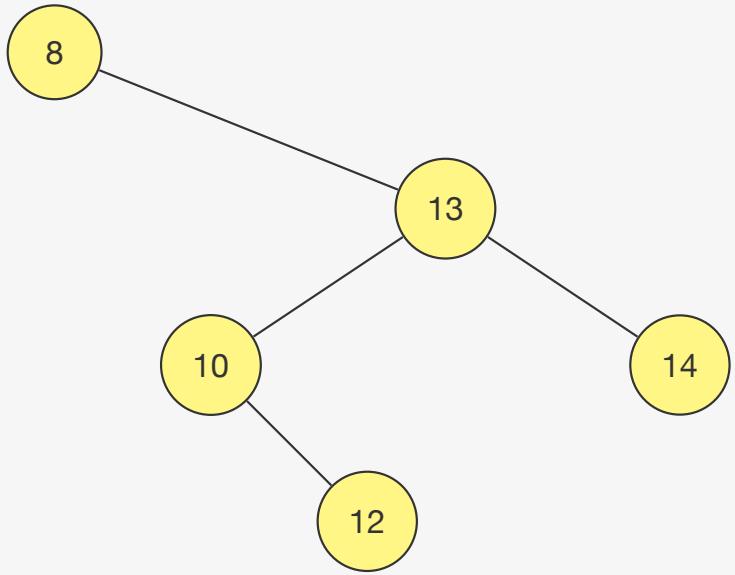
Onde está o nó com a menor chave de uma árvore?



Quem é o mínimo para essa árvore?

Mínimo da Árvore

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



Quem é o mínimo para essa árvore?

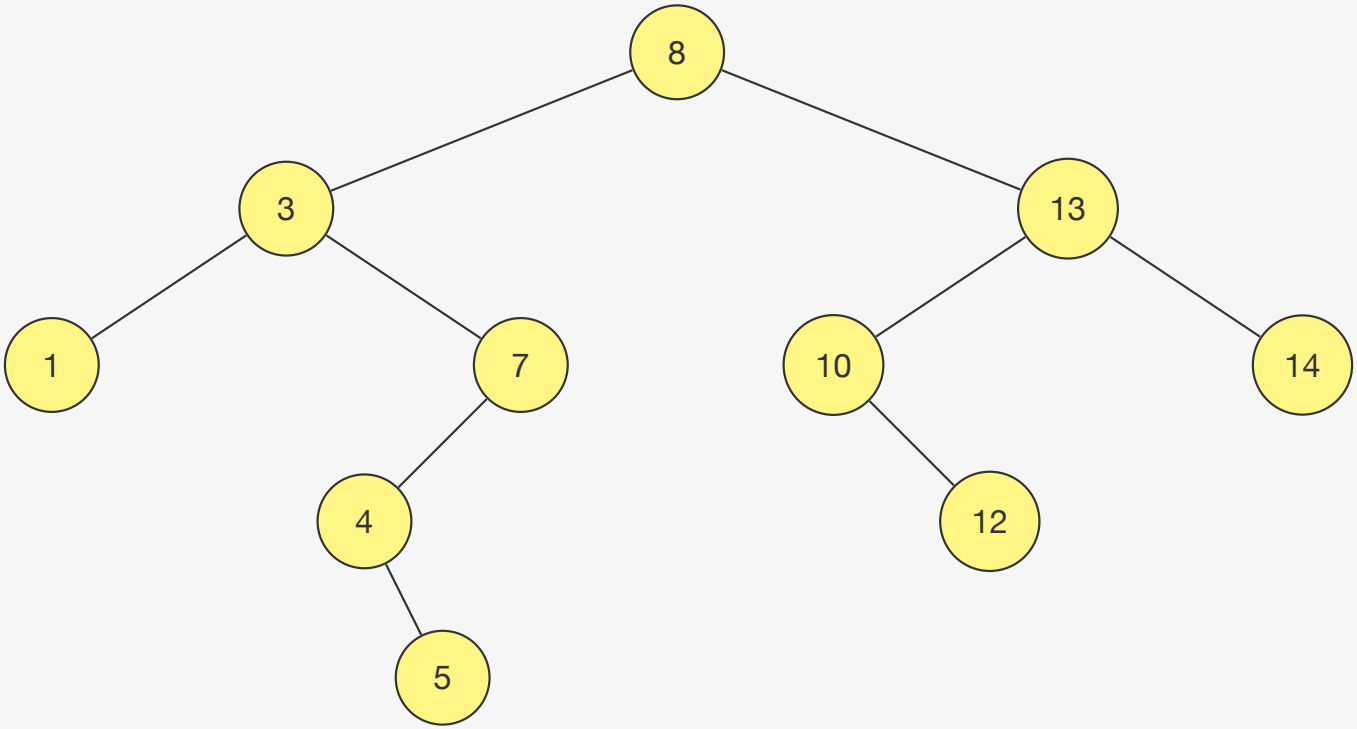
- É a própria raiz

Mínimo da Árvore

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

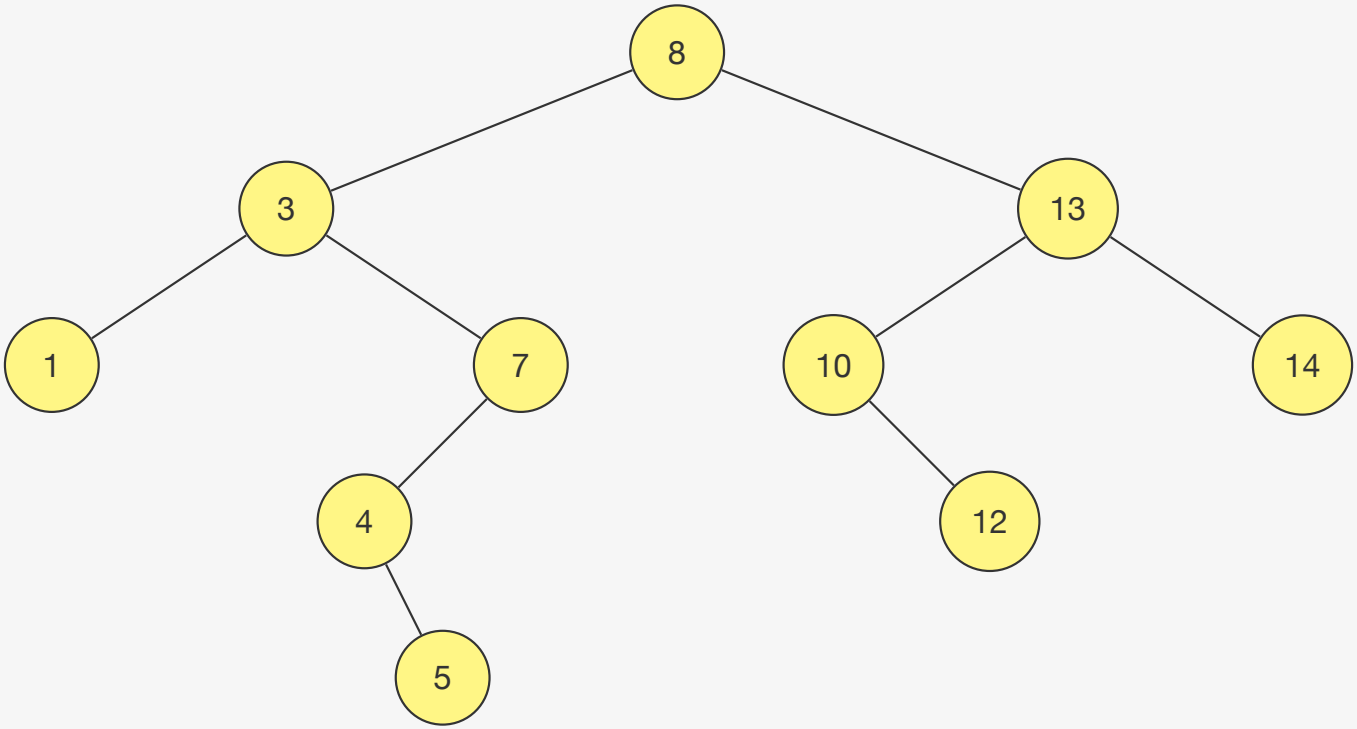
Mínimo da Árvore

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



Mínimo da Árvore

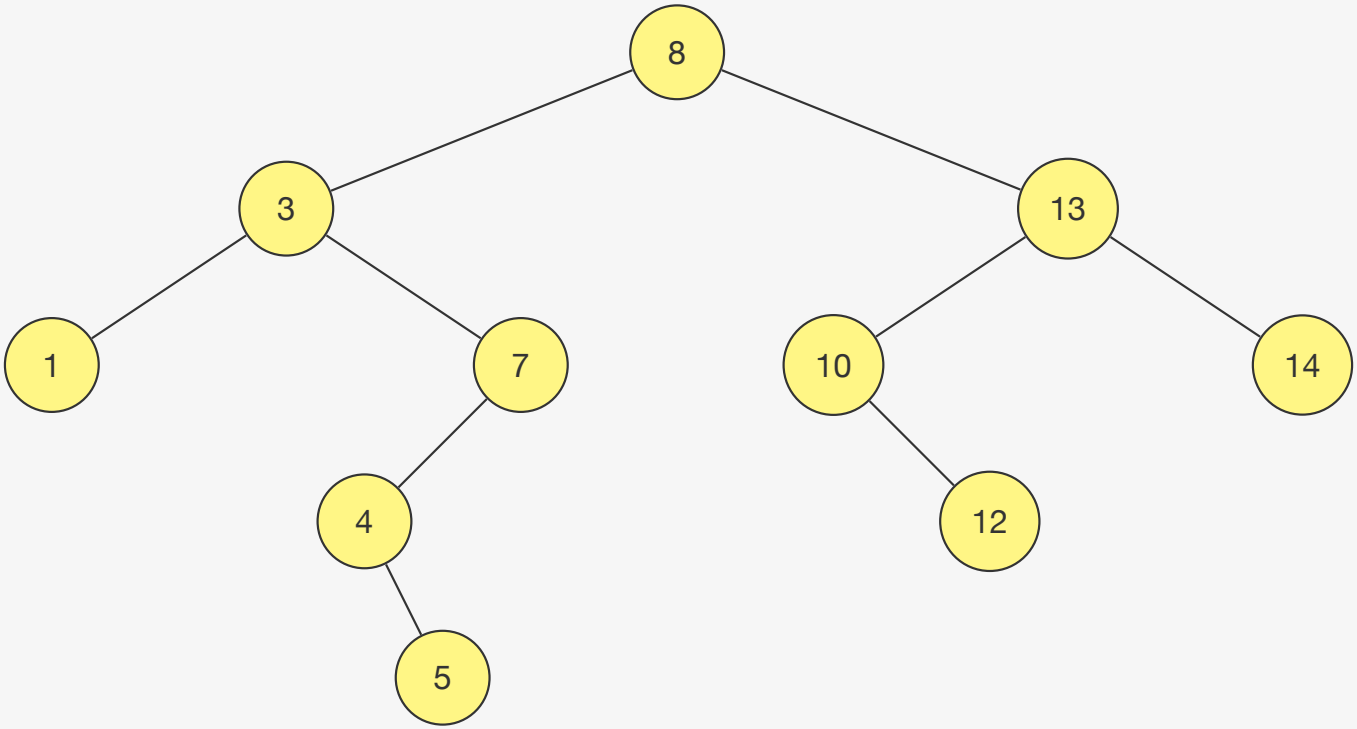
Onde está o nó com a menor chave de uma árvore?



Quem é o mínimo para essa árvore?

Mínimo da Árvore

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:

```
1 No* minimo(No *arvore) {
2   if (arvore == NULL || arvore->esq == NULL)
3     return arvore;
4   return minimo(arvore->esq);
5 }
```

Mínimo - Implementações

Versão recursiva:

```
1 No* minimo(No *arvore) {
2   if (arvore == NULL || arvore->esq == NULL)
3     return arvore;
4   return minimo(arvore->esq);
5 }
```

Versão iterativa:

```
1 No* minimo_iterativo(No *arvore) {
2   while (arvore != NULL && arvore->esq != NULL)
3     arvore = arvore->esq;
4   return arvore;
5 }
```

Mínimo - Implementações

Versão recursiva:

```
1 No* minimo(No *arvore) {
2   if (arvore == NULL || arvore->esq == NULL)
3     return arvore;
4   return minimo(arvore->esq);
5 }
```

Versão iterativa:

```
1 No* minimo_iterativo(No *arvore) {
2   while (arvore != NULL && arvore->esq != NULL)
3     arvore = arvore->esq;
4   return arvore;
5 }
```

Para encontrar o máximo, basta fazer a operação simétrica

Mínimo - Implementações

Versão recursiva:

```
1 No* minimo(No *arvore) {
2   if (arvore == NULL || arvore->esq == NULL)
3     return arvore;
4   return minimo(arvore->esq);
5 }
```

Versão iterativa:

```
1 No* minimo_iterativo(No *arvore) {
2   while (arvore != NULL && arvore->esq != NULL)
3     arvore = arvore->esq;
4   return arvore;
5 }
```

Para encontrar o máximo, basta fazer a operação simétrica

- Se a subárvore direita existir, é o seu máximo

Mínimo - Implementações

Versão recursiva:

```
1 No* minimo(No *arvore) {
2   if (arvore == NULL || arvore->esq == NULL)
3     return arvore;
4   return minimo(arvore->esq);
5 }
```

Versão iterativa:

```
1 No* minimo_iterativo(No *arvore) {
2   while (arvore != NULL && arvore->esq != NULL)
3     arvore = arvore->esq;
4   return arvore;
5 }
```

Para encontrar o máximo, basta fazer a operação simétrica

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

Sucessor

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

Sucessor

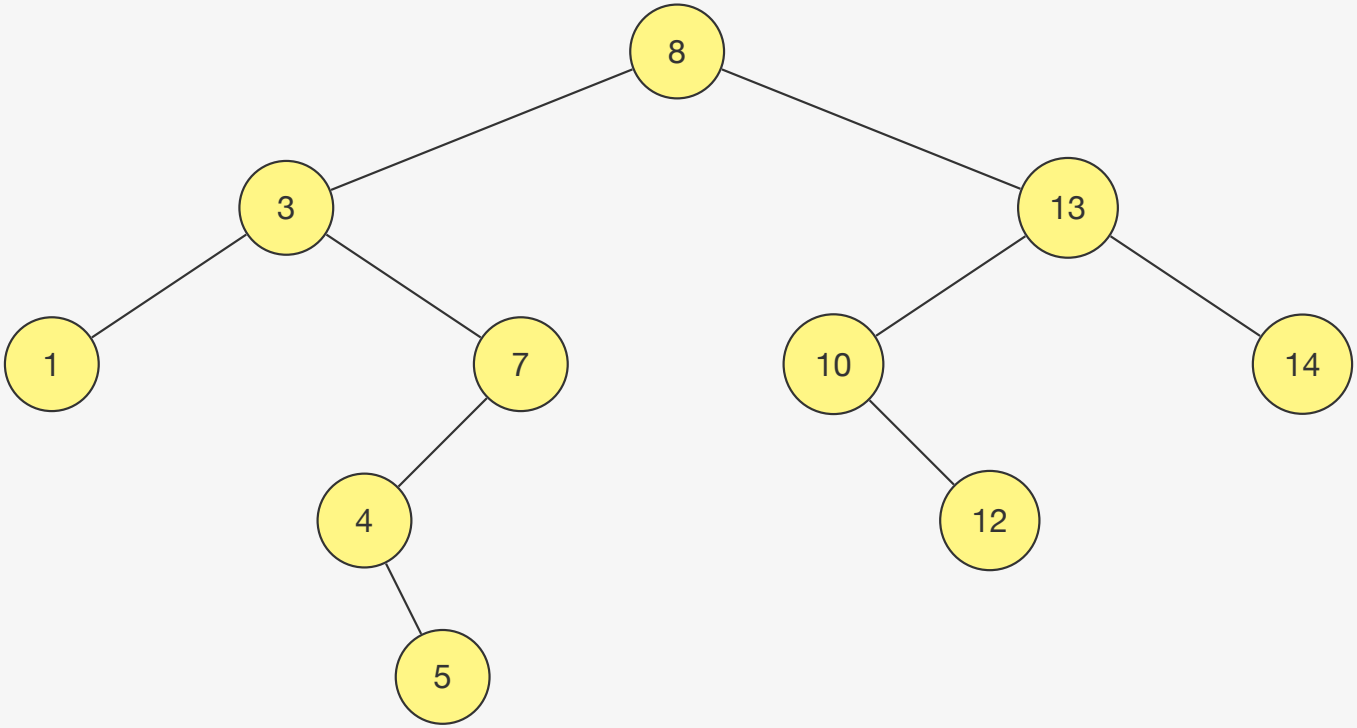
Dado um nó da árvore, onde está o seu sucessor?

- O sucessor é o próximo nó na ordenação

Sucessor

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

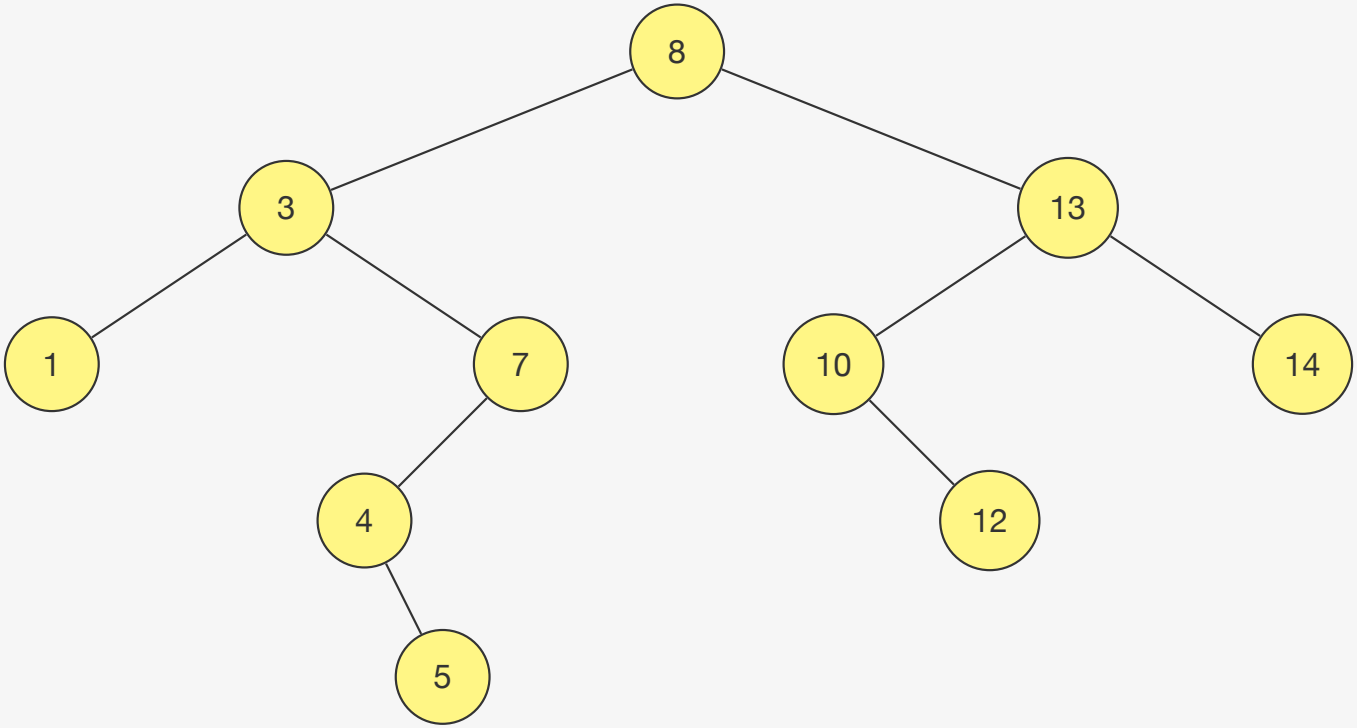
- O sucessor é o próximo nó na ordenação



Sucessor

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

- O sucessor é o próximo nó na ordenação

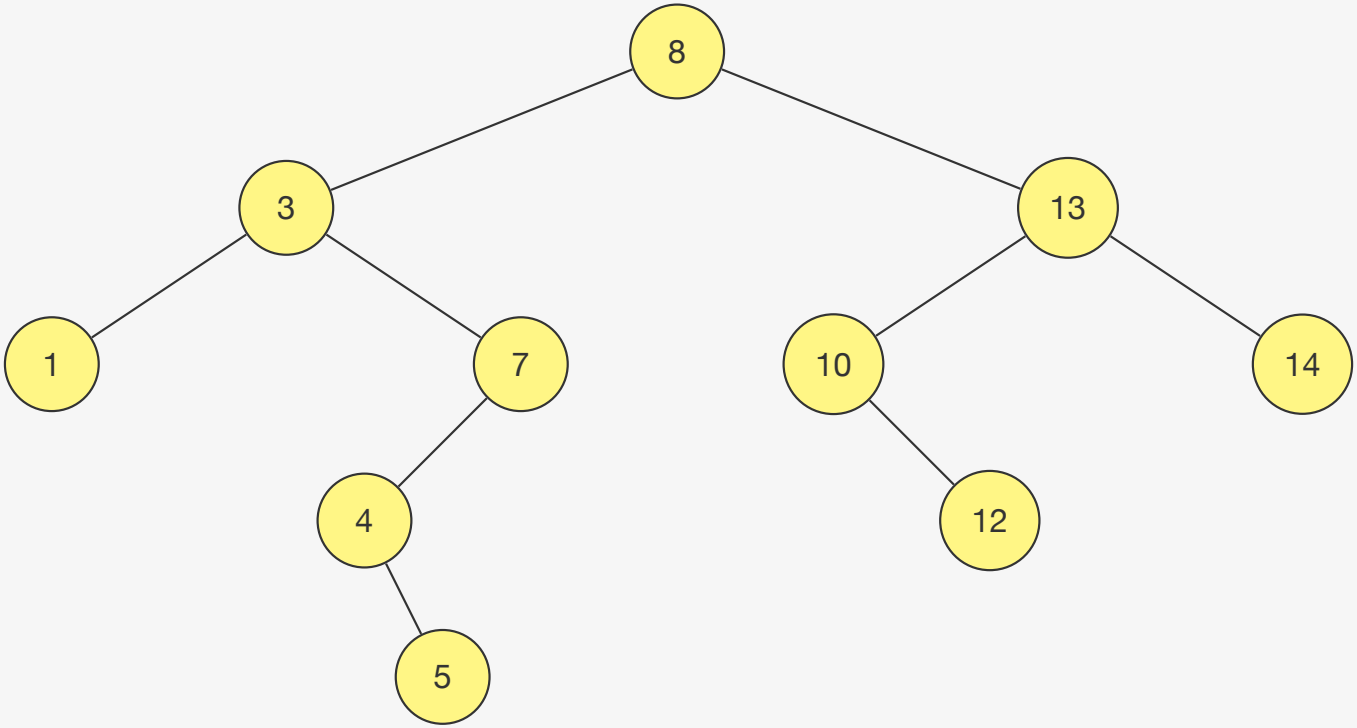


Quem é o sucessor de 3?

Sucessor

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

- O sucessor é o próximo nó na ordenação



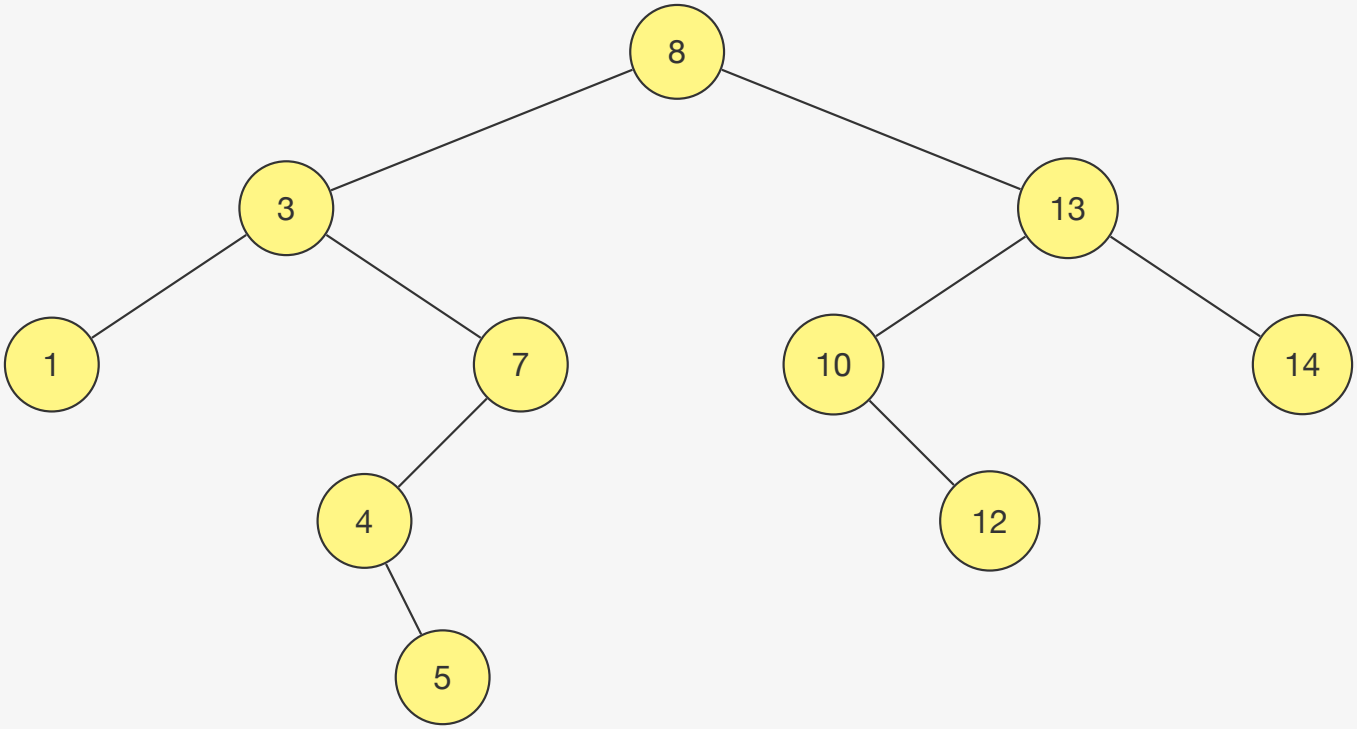
Quem é o sucessor de 3?

- É o mínimo da sua subárvore direita

Sucessor

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

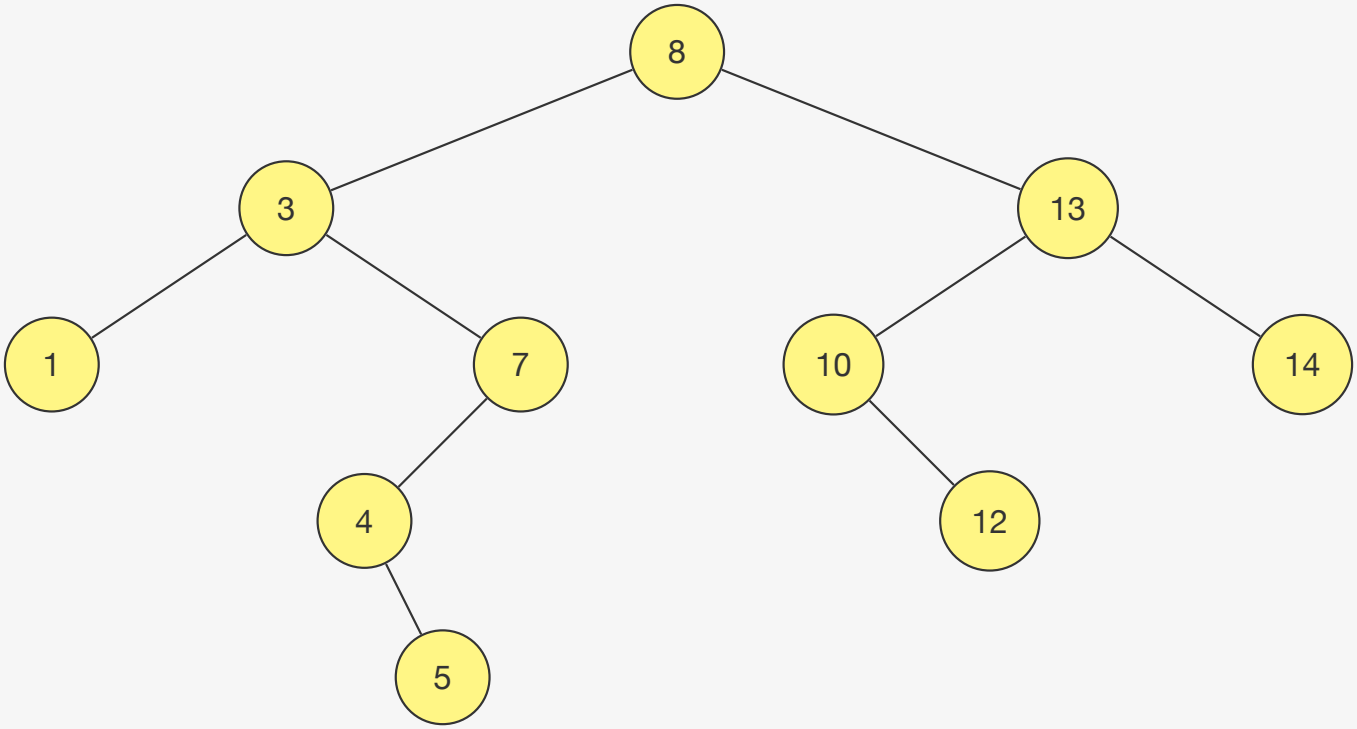
- O sucessor é o próximo nó na ordenação



Sucessor

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

- O sucessor é o próximo nó na ordenação

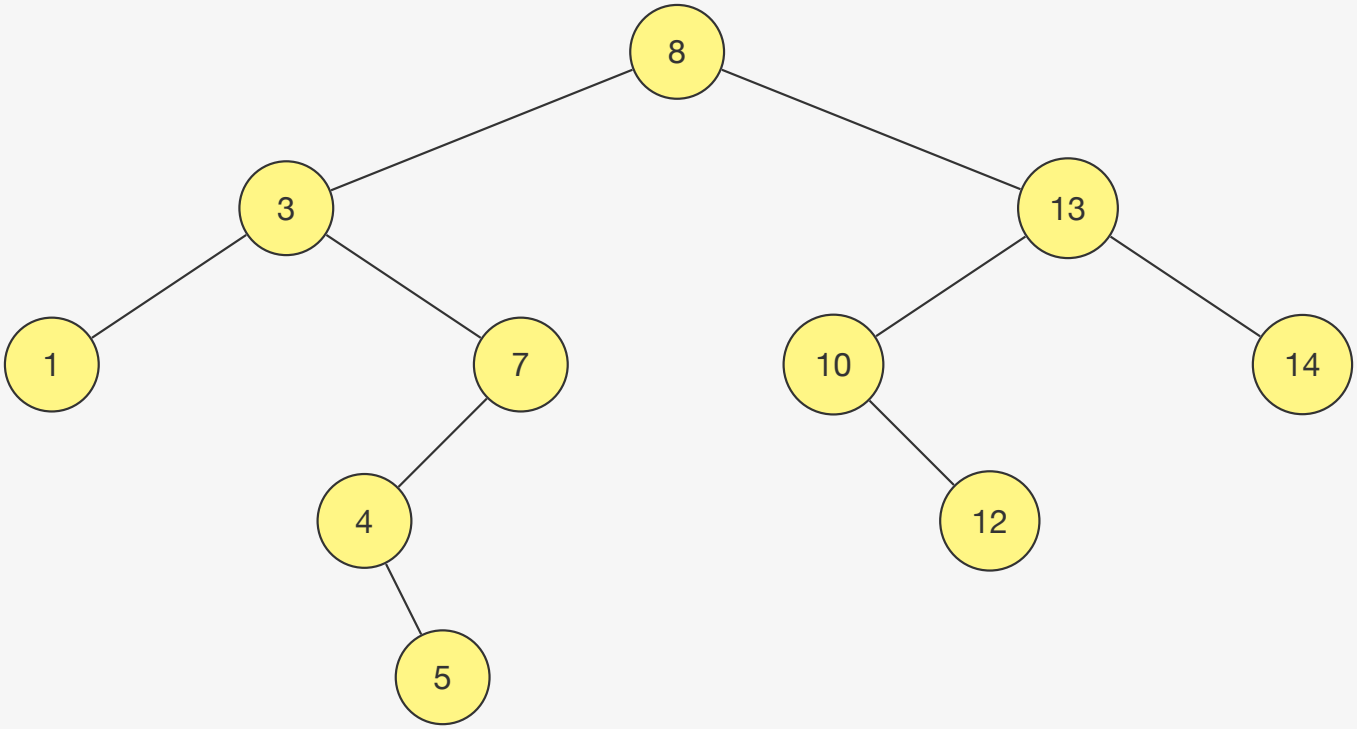


Quem é o sucessor de 7?

Sucessor

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

- O sucessor é o próximo nó na ordenação



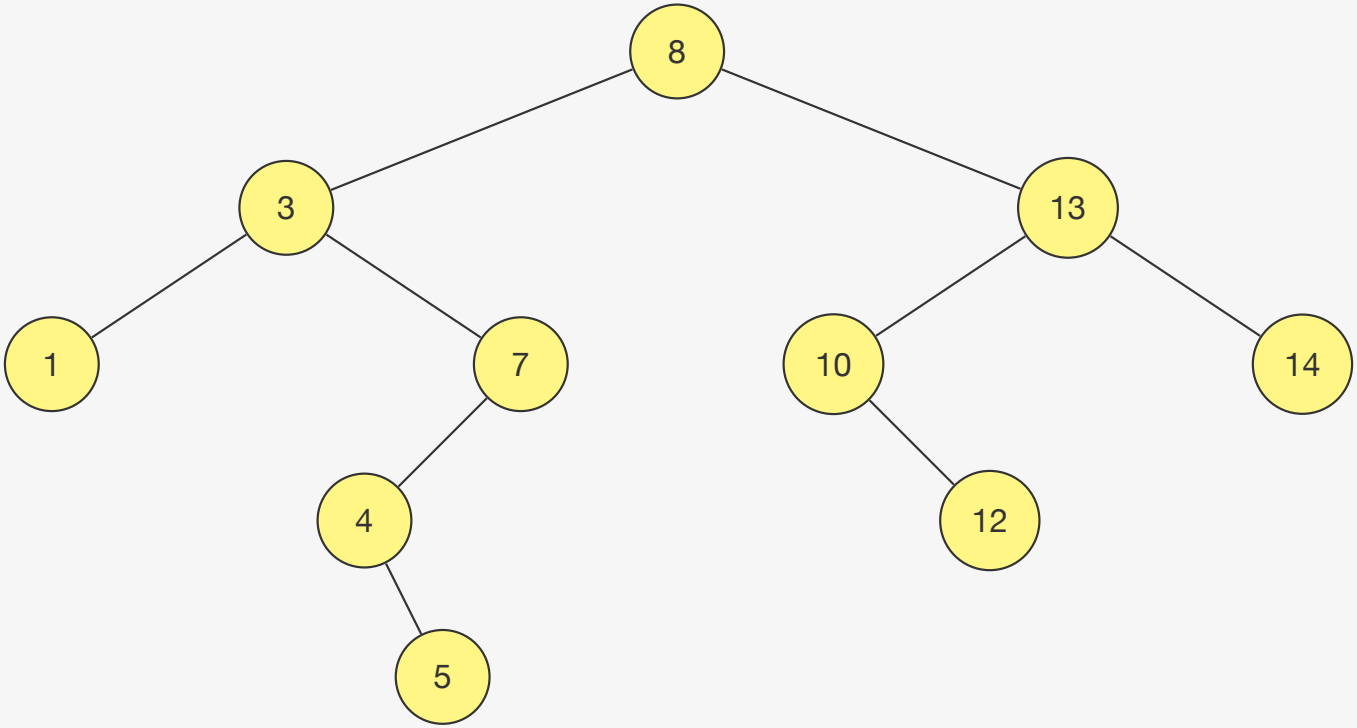
Quem é o sucessor de 7?

- É primeiro ancestral a direita

Sucessor

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

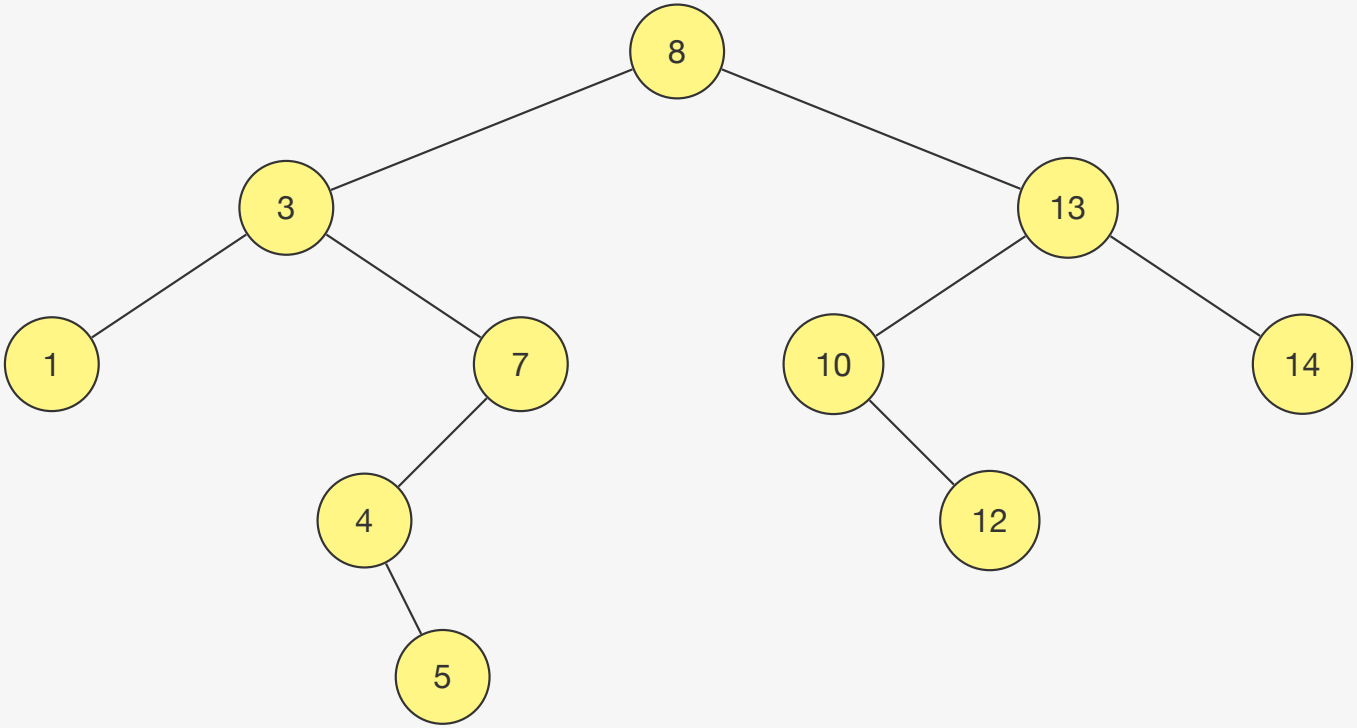
- O sucessor é o próximo nó na ordenação



Sucessor

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

- O sucessor é o próximo nó na ordenação

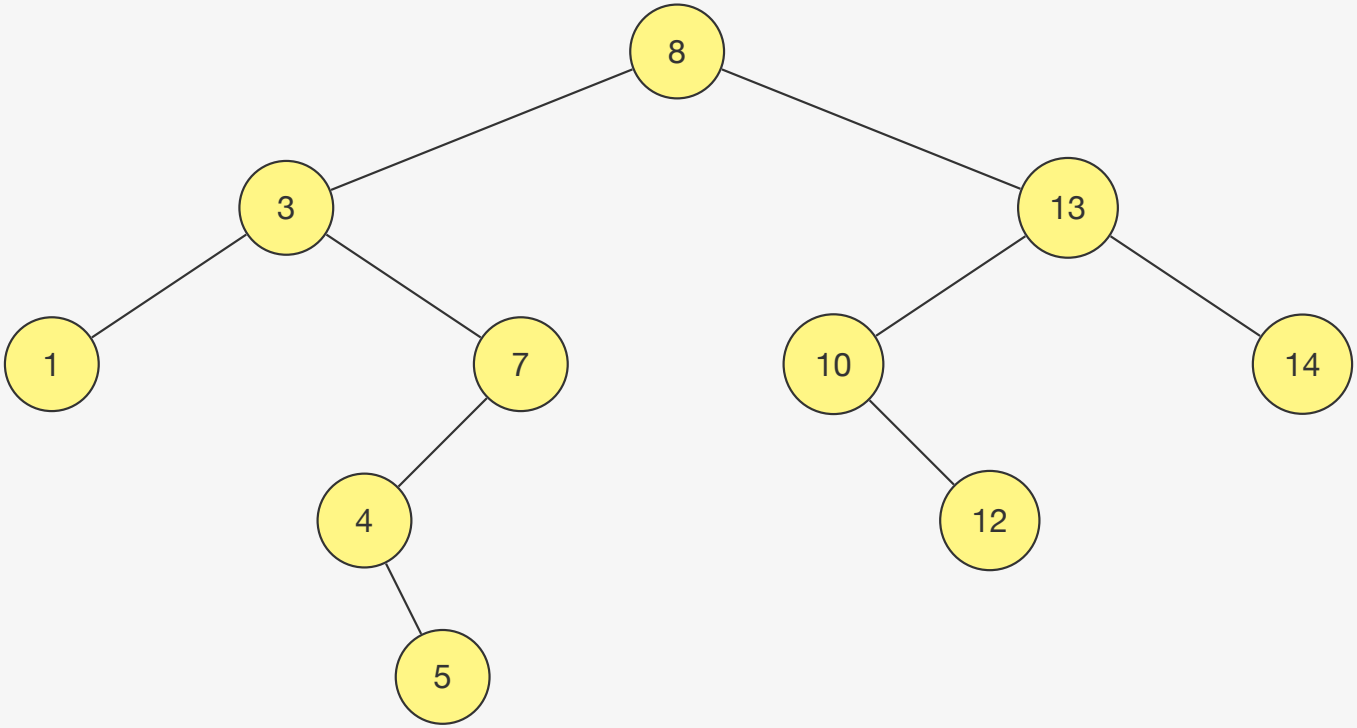


Quem é o sucessor de 14?

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 No* sucessor(No *x) {
2   if (x->dir != NULL)
3     return minimo(x->dir);
4   else
5     return ancestral_a_direita(x);
6 }
```

Sucessor - Implementação

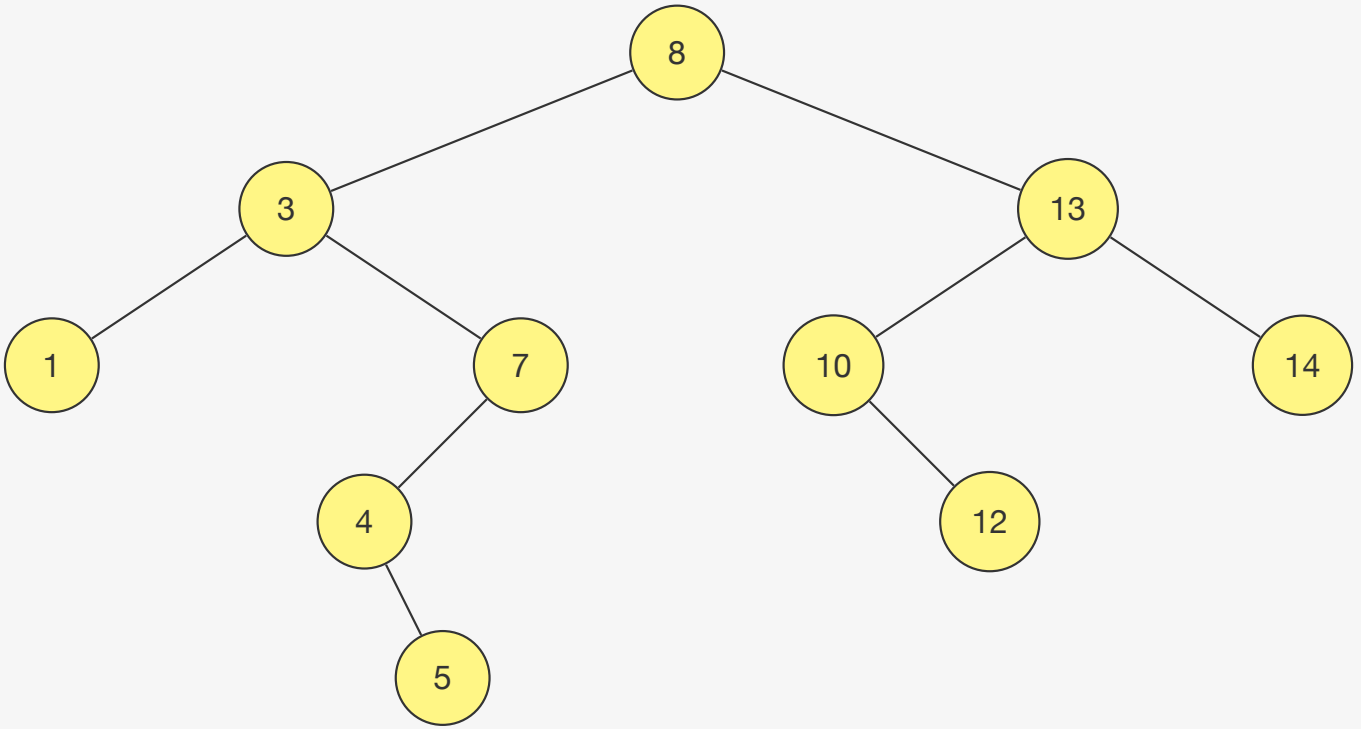
```
1 No* sucessor(No *x) {
2   if (x->dir != NULL)
3     return minimo(x->dir);
4   else
5     return ancestral_a_direita(x);
6 }
```

```
1 No* ancestral_a_direita(No *x) {
2   if (x == NULL)
3     return NULL;
4   if (x->pai == NULL || x->pai->esq == x)
5     return x->pai;
6   else
7     return ancestral_a_direita(x->pai);
8 }
```

A implementação da função `antecessor` é simétrica

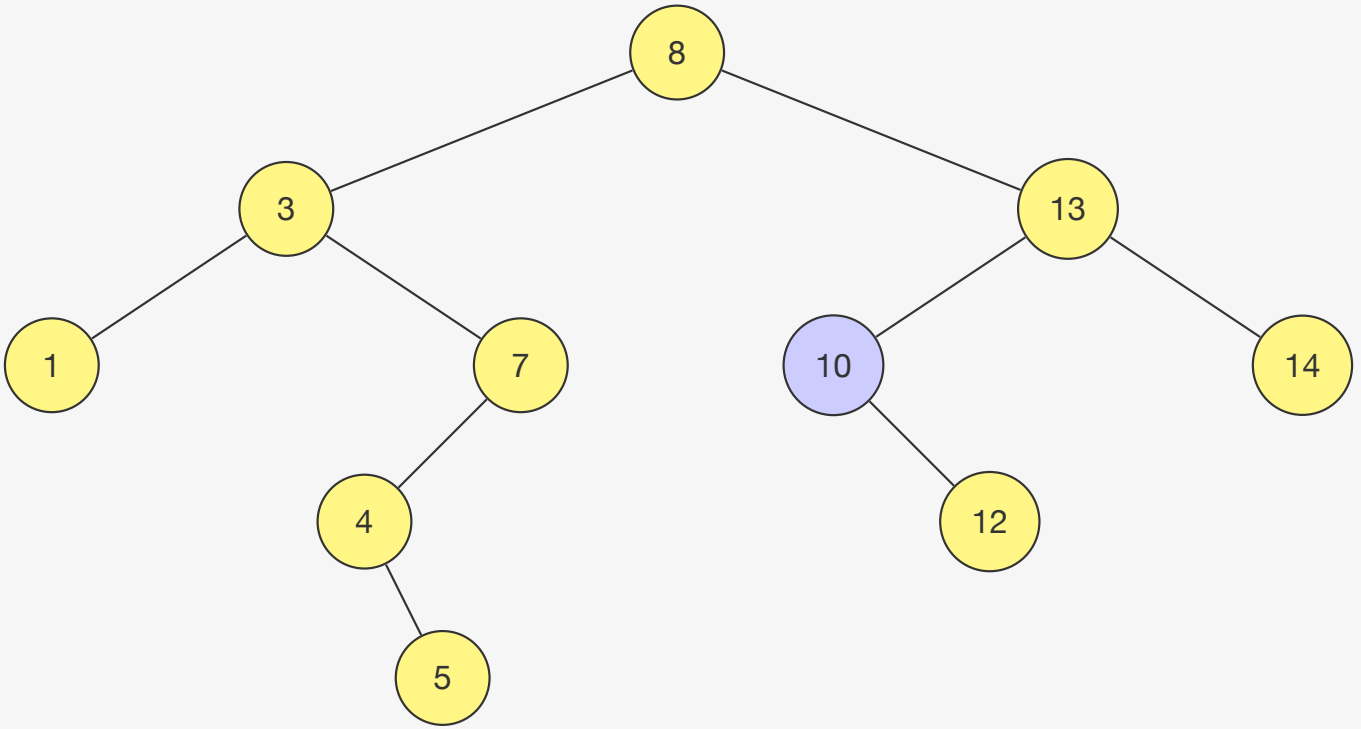
Remoção

Ex: removendo 10



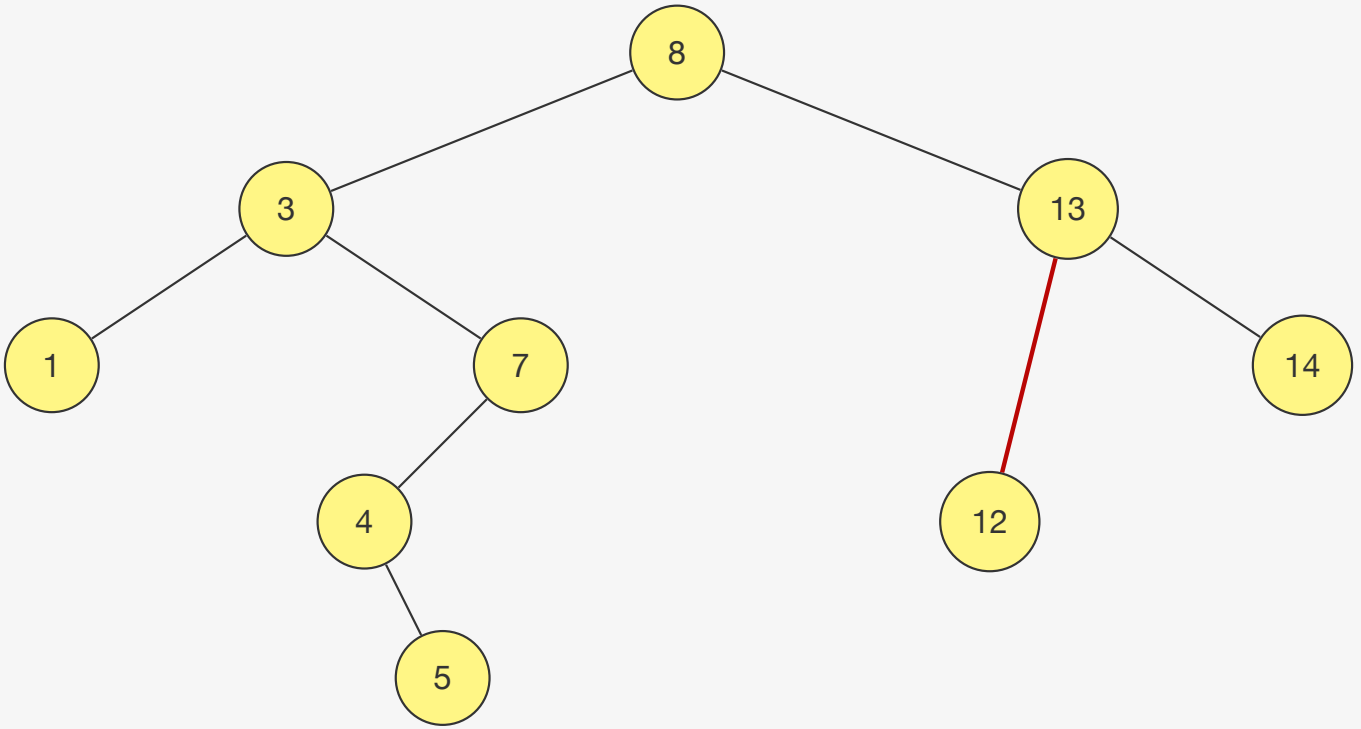
Remoção

Ex: removendo 10



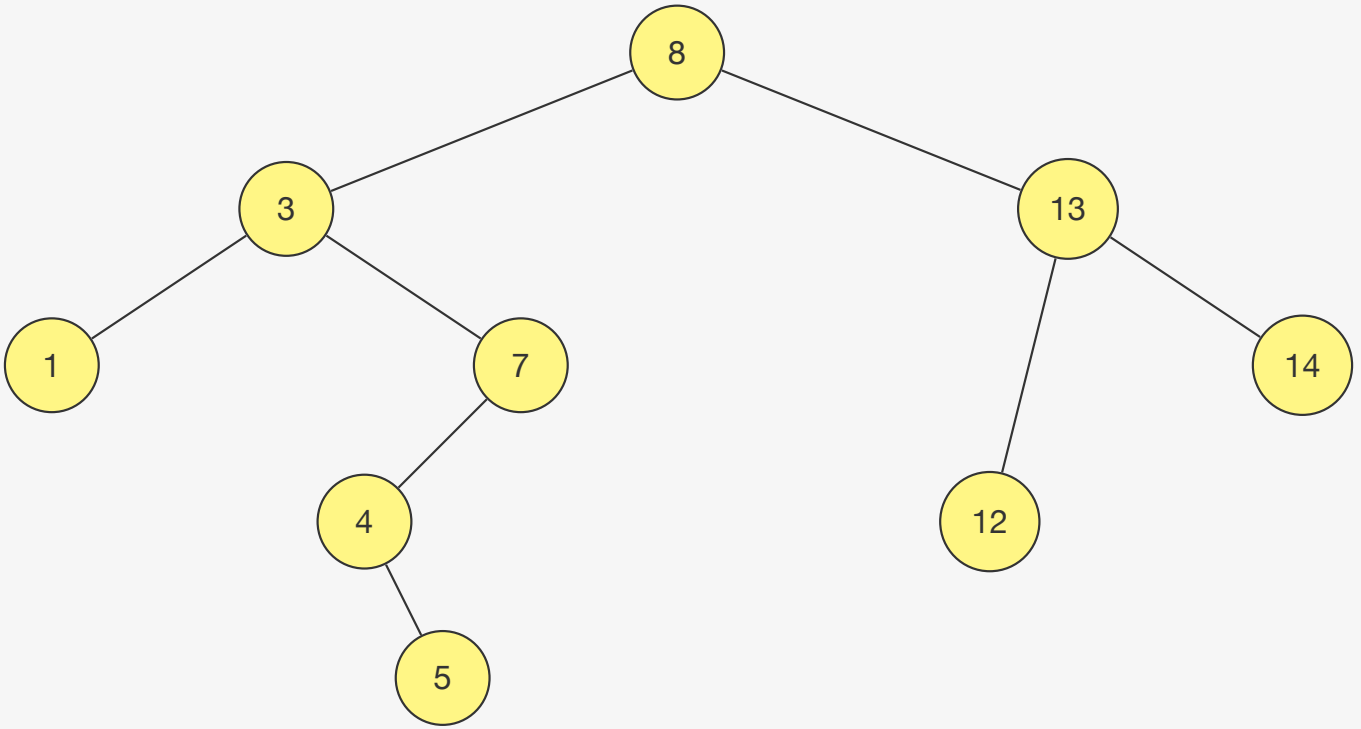
Remoção

Ex: removendo 10



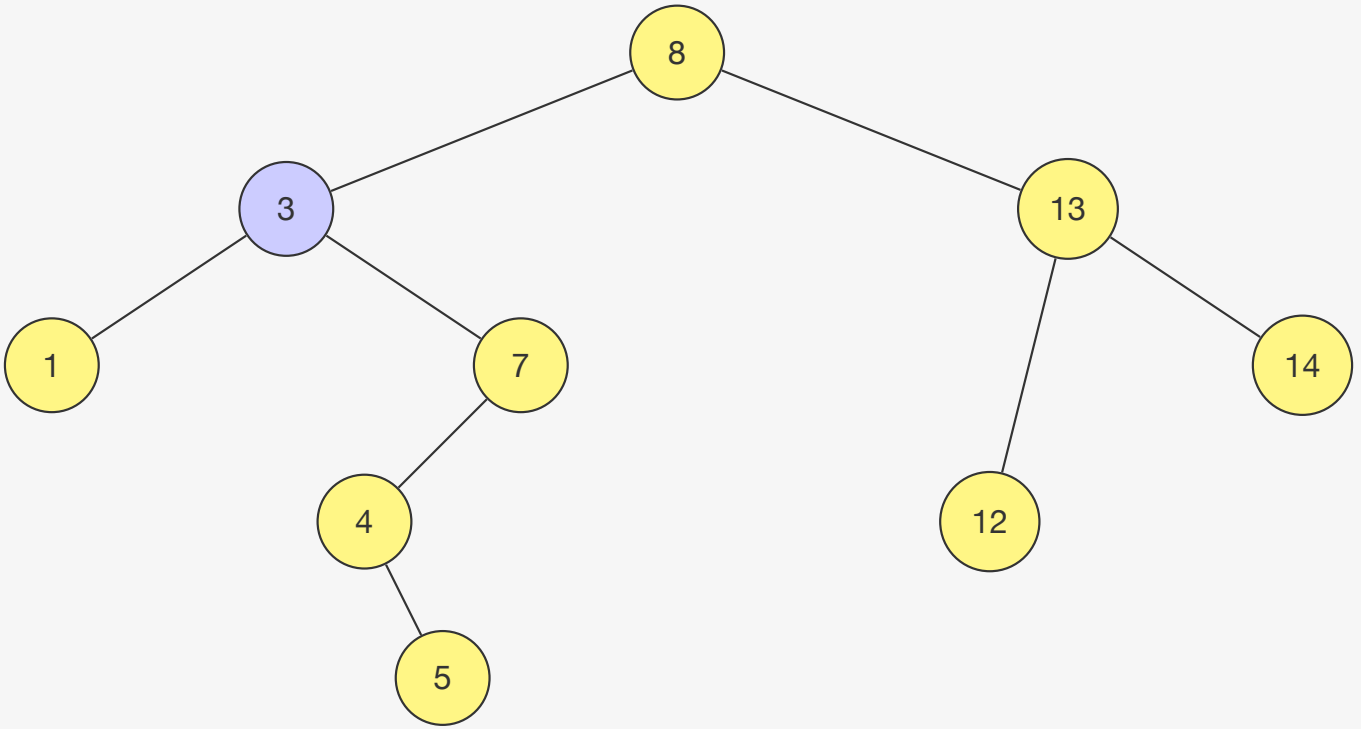
Remoção

Ex: removendo 3



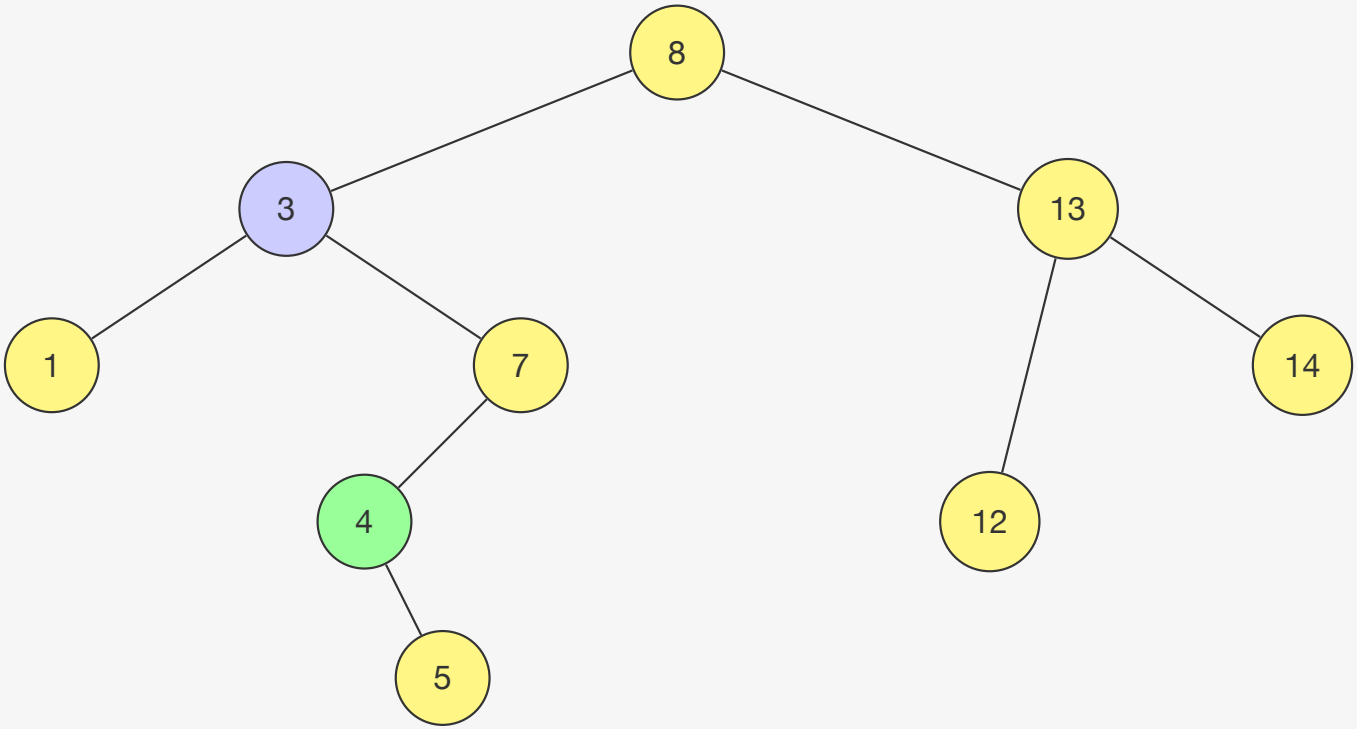
Remoção

Ex: removendo 3



Remoção

Ex: removendo 3

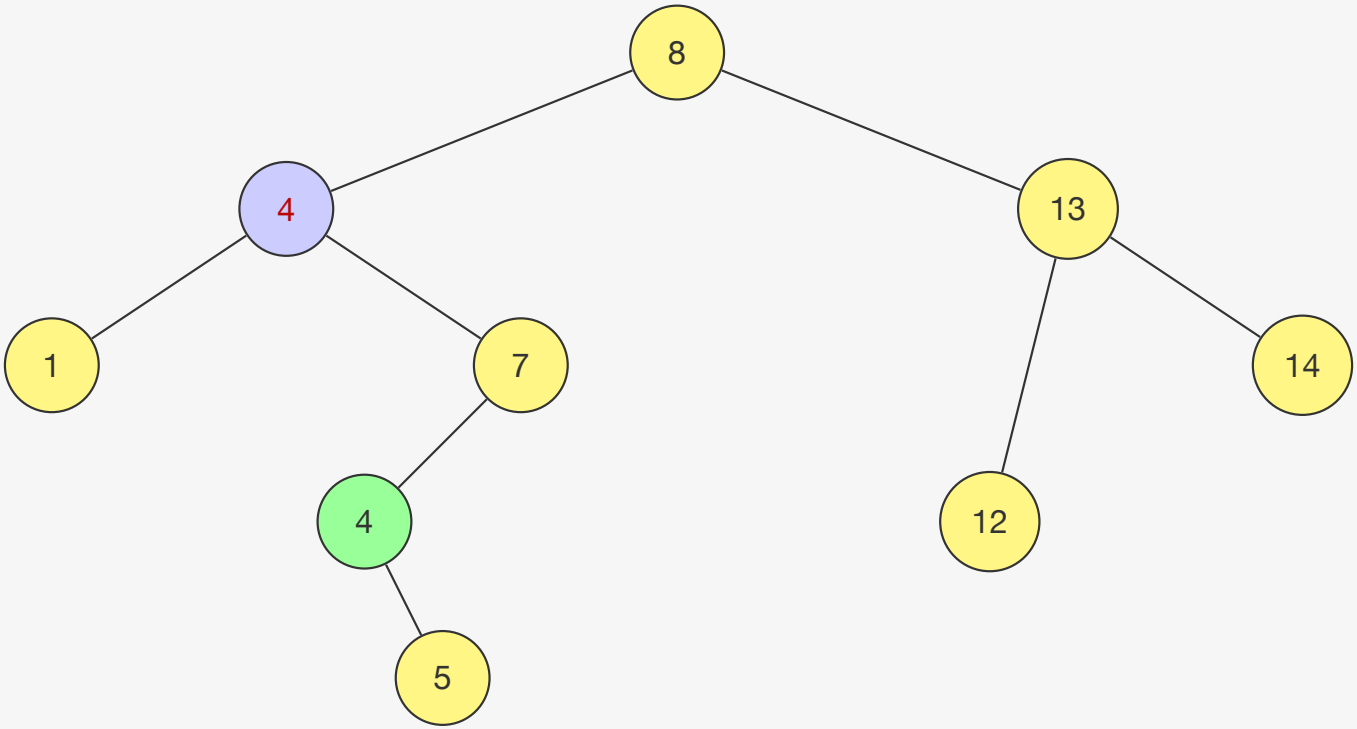


Podemos colocar o sucessor de 3 em seu lugar

- Isso mantém a propriedade da árvore binária de busca

Remoção

Ex: removendo 3

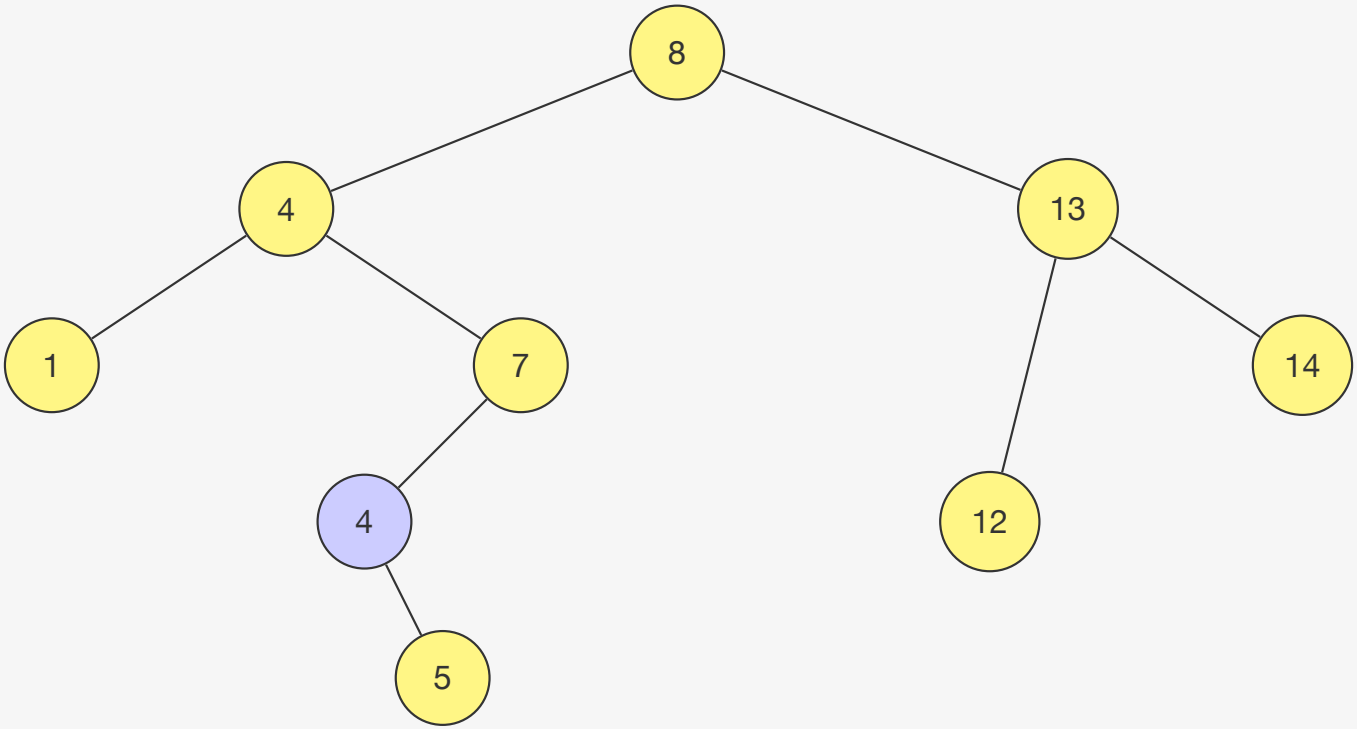


Podemos colocar o sucessor de 3 em seu lugar

- Isso mantém a propriedade da árvore binária de busca

Remoção

Ex: removendo 3



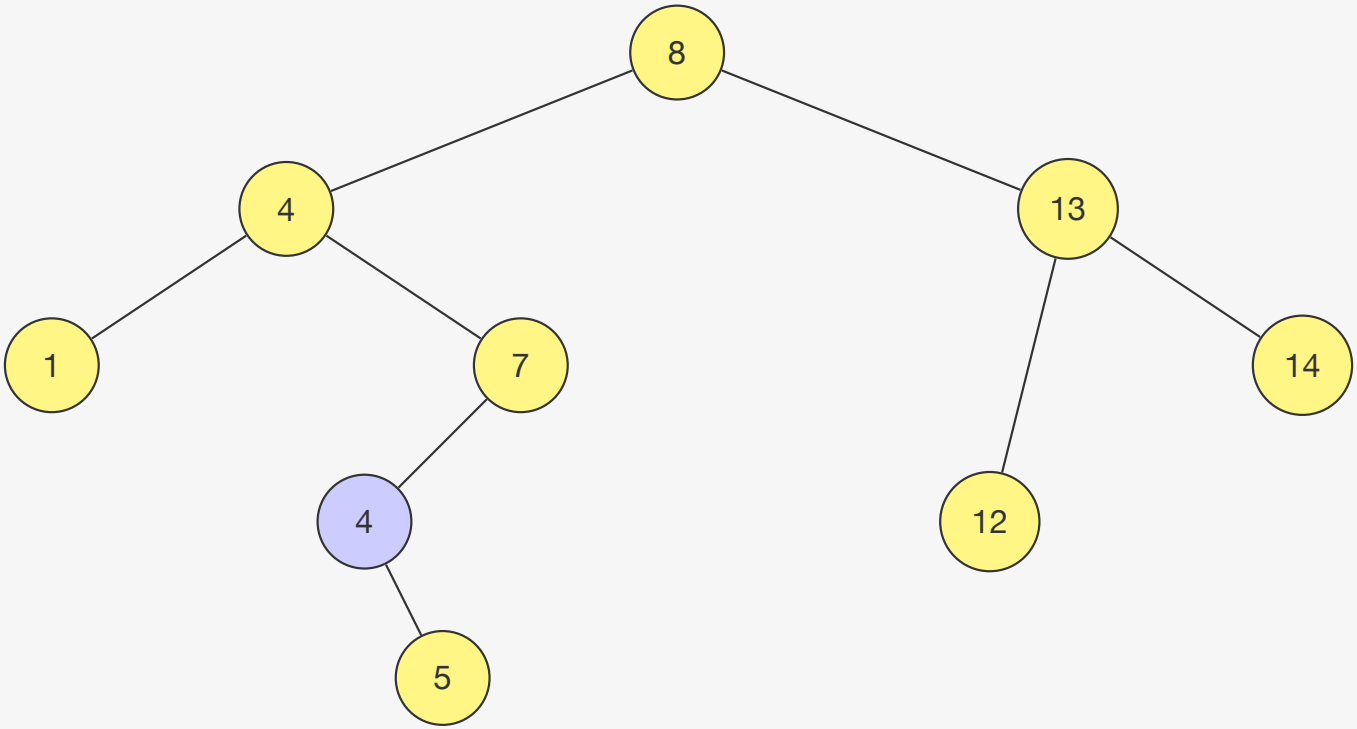
Podemos colocar o sucessor de 3 em seu lugar

- Isso mantém a propriedade da árvore binária de busca

E agora removemos o sucessor

Remoção

Ex: removendo 3



Podemos colocar o sucessor de 3 em seu lugar

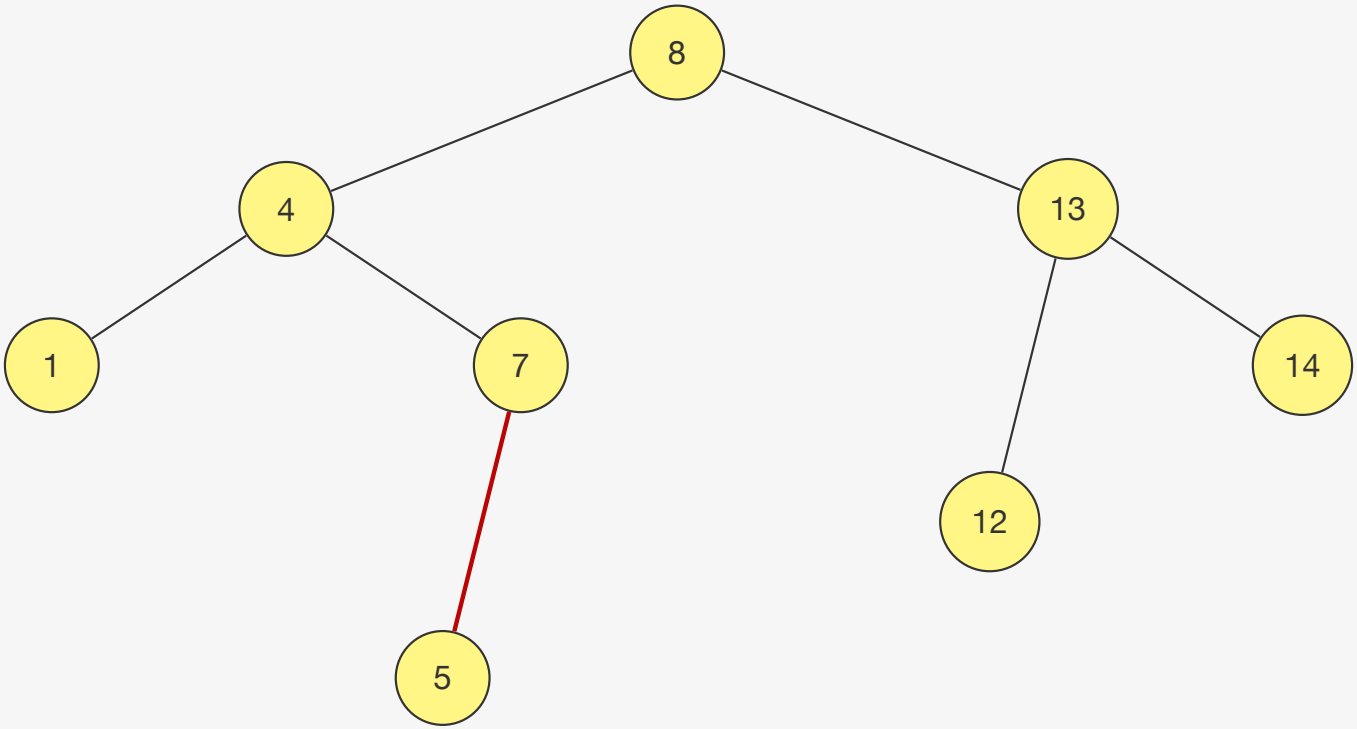
- Isso mantém a propriedade da árvore binária de busca

E agora removemos o sucessor

- O sucessor nunca tem filho esquerdo!

Remoção

Ex: removendo 3



Podemos colocar o sucessor 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

Versão sem ponteiro para `pai` e que não libera o nó

```
1 No* remover_rec(No *arvore, int chave) {
```

Remoção - Implementação

Versão sem ponteiro para `pai` e que não libera o nó

```
1 No* remover_rec(No *arvore, int chave) {
2   if (arvore == NULL)
3     return NULL;
```

Remoção - Implementação

Versão sem ponteiro para `pai` e que não libera o nó

```
1 No* remover_rec(No *arvore, int chave) {
2   if (arvore == NULL)
3     return NULL;
4   if (chave < arvore->chave) {
5     arvore->esq = remover_rec(arvore->esq, chave);
6     return arvore;
7   }
```

Remoção - Implementação

Versão sem ponteiro para `pai` e que não libera o nó

```
1  No* remover_rec(No *arvore, int chave) {
2    if (arvore == NULL)
3      return NULL;
4    if (chave < arvore->chave) {
5      arvore->esq = remover_rec(arvore->esq, chave);
6      return arvore;
7    }
8    else if (chave > arvore->chave) {
9      arvore->dir = remover_rec(arvore->dir, chave);
10     return arvore;
11   }
```

Remoção - Implementação

Versão sem ponteiro para `pai` e que não libera o nó

```
1  No* remover_rec(No *arvore, int chave) {
2    if (arvore == NULL)
3      return NULL;
4    if (chave < arvore->chave) {
5      arvore->esq = remover_rec(arvore->esq, chave);
6      return arvore;
7    }
8    else if (chave > arvore->chave) {
9      arvore->dir = remover_rec(arvore->dir, chave);
10     return arvore;
11   }
12   else if (arvore->esq == NULL)
13     return arvore->dir;
```

Remoção - Implementação

Versão sem ponteiro para `pai` e que não libera o nó

```
1  No* remover_rec(No *arvore, int chave) {
2    if (arvore == NULL)
3      return NULL;
4    if (chave < arvore->chave) {
5      arvore->esq = remover_rec(arvore->esq, chave);
6      return arvore;
7    }
8    else if (chave > arvore->chave) {
9      arvore->dir = remover_rec(arvore->dir, chave);
10     return arvore;
11   }
12   else if (arvore->esq == NULL)
13     return arvore->dir;
14   else if (arvore->dir == NULL)
15     return arvore->esq;
```

Remoção - Implementação

Versão sem ponteiro para `pai` e que não libera o nó

```
1  No* remover_rec(No *arvore, int chave) {
2    if (arvore == NULL)
3      return NULL;
4    if (chave < arvore->chave) {
5      arvore->esq = remover_rec(arvore->esq, chave);
6      return arvore;
7    }
8    else if (chave > arvore->chave) {
9      arvore->dir = remover_rec(arvore->dir, chave);
10     return arvore;
11   }
12   else if (arvore->esq == NULL)
13     return arvore->dir;
14   else if (arvore->dir == NULL)
15     return arvore->esq;
16   else {
17     remover_sucessor(arvore);
18     return arvore;
19   }
20 }
```

Remoção - Implementação

```
1 void remover_sucessor(No *arvore) {
```

Remoção - Implementação

```
1 void remover_sucessor(No *arvore) {
2   No *pai = arvore, *t = arvore->dir;
3   while (t->esq != NULL) {
4     pai = t;
5     t = t->esq;
6   }
```

Remoção - Implementação

```
1 void remover_sucessor(No *arvore) {
2   No *pai = arvore, *t = arvore->dir;
3   while (t->esq != NULL) {
4     pai = t;
5     t = t->esq;
6   }
7   if (pai->esq == t)
8     pai->esq = t->dir;
```

Remoção - Implementação

```
1  void remover_sucessor(No *arvore) {
2    No *pai = arvore, *t = arvore->dir;
3    while (t->esq != NULL) {
4      pai = t;
5      t = t->esq;
6    }
7    if (pai->esq == t)
8      pai->esq = t->dir;
9    else
10     pai->dir = t->dir;
11   arvore->chave = t->chave;
12 }
```

```
1  void remover(No **arvore, int chave) {
2    *arvore = remover_rec(*arvore, chave);
3  }
```

Exercícios

1. Faça uma função que imprime as chaves de uma ABB em ordem crescente
2. Escreva uma função recursiva que calcula o número máximos de comparações feitas em uma busca de uma ABB
3. Implemente as funções máximo e antecessor de um nó da ABB
4. Refaça as implementações para considerar uma árvore com ponteiro para o pai
5. Faça uma implementação da função `sucessor` que não usa o ponteiro `pai`
   - Dica: você precisará da raiz da árvore pois não pode subir