Universidade Federal de Ouro Preto
Campus João Monlevade

CSI 488 – Algoritmos e Estruturas de Dados I

TAD – Árvores Binárias de Busca

Prof. Mateus Ferreira Satler

Índice

1. Introdução

- Na computação, assim como na natureza, existem vários tipos diferentes de árvores.
  - Cada uma delas foi desenvolvida pensando diferentes tipos de aplicações.
- Uma **árvore binária de busca** é uma estrutura de dados não linear utilizada para acesso rápido aos dados.
  - São também conhecidas como dicionários binários

1. Introdução

- A árvore de busca é uma estrutura de dados muito eficiente para armazenar informação.

- Particularmente adequada quando existe necessidade de considerar todos ou alguma combinação de:
  - Acesso direto e sequencial eficientes.
  - Facilidade de inserção e retirada de registros.
  - Boa taxa de utilização de memória.

1. Introdução

- Em uma árvore binária de busca cada nó contém um campo chamado **chave** (key), podendo haver outras informações.

- O campo **chave** especifica, em geral, um valor que identifica de **forma única** um determinado registro ou informação.

- Exemplos de chaves:
  - Número de identidade, número CPF, etc.
  - Assim, todos os valores de chave são distintos entre si.

1. Introdução

- Definição de **Árvore Binária de Busca**
  - É uma árvore binária.
    - Cada nó pode ter 0, 1 ou 2 filhos.
  - Cada nó da árvore possui um valor (chave) associado a ele.
    - Não existem valores de chave repetidos.
    - Esse valor determina a posição do nó na árvore.

1. Introdução

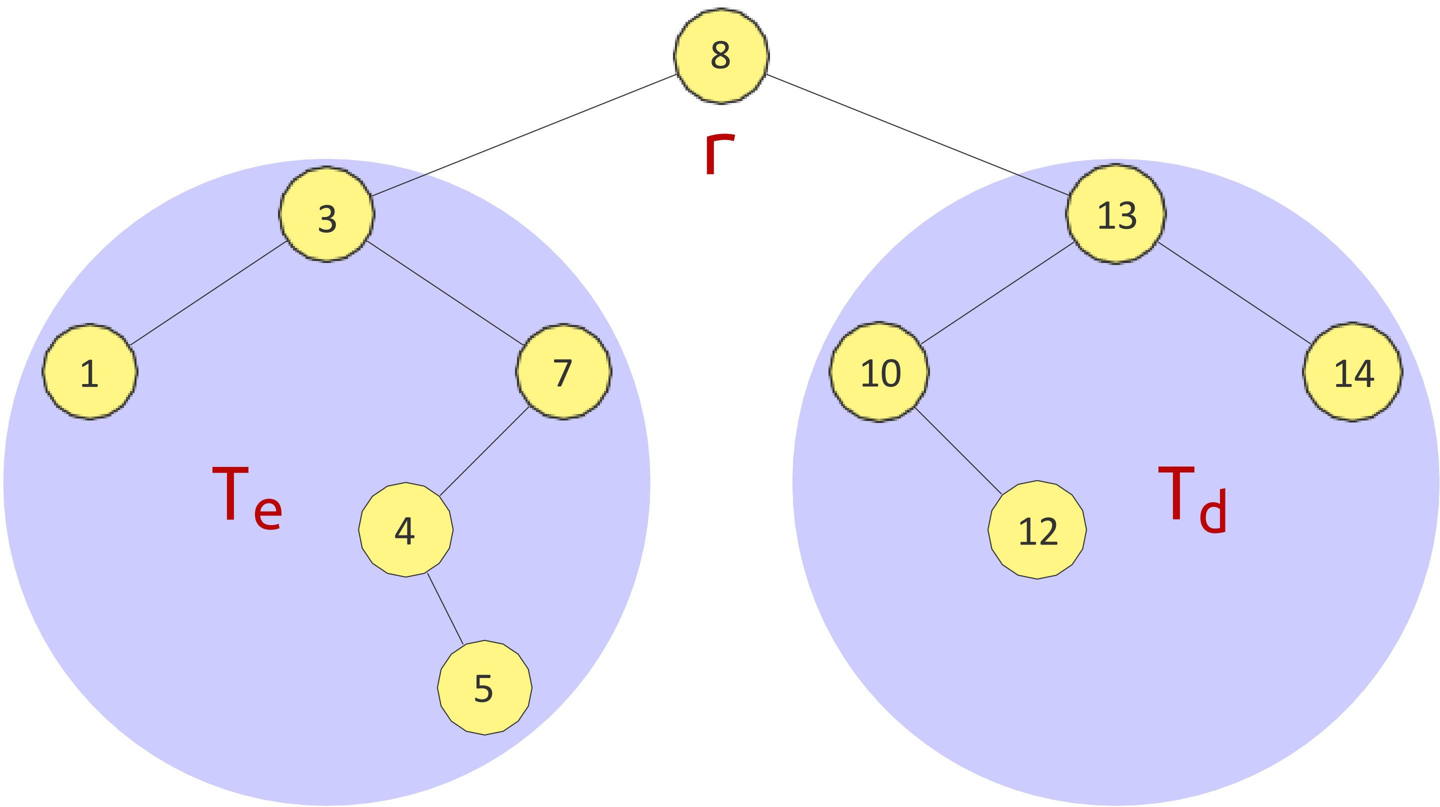
- Regra para posicionamento dos valores na árvore:
  - Para cada nó pai...
    - ... todos os valores da sub-árvore **esquerda** são **menores** do que o nó pai.
    - ... todos os valores da sub-árvore **direita** são **maiores** do que o nó pai.
  - Inserção e remoção devem ser realizadas respeitando essa regra de posicionamento dos nós.

1. Introdução

- Formalmente:
  - Cada nó **r**, com sub-árvores esquerda $\mathbf{T_e}$ e direita $\mathbf{T_d}$ satisfaz a seguinte propriedade:
    - $\mathbf{e < r}$ para todo elemento $\mathbf{e \in T_e}$
    - $\mathbf{d > r}$ para todo elemento $\mathbf{d \in T_d}$

1. Introdução

- Exemplo:



1.2. Implementação

- TAD – Árvore Binária de Busca

```
typedef struct {
  long chave;
  /* outros componentes */
}TRegistro;

typedef struct Tno_Est{
  TRegistro reg;
  struct Tno_Est *pEsq, *pDir;
}TNo;

typedef TNo* TDicionario;
```

2. Busca por um valor

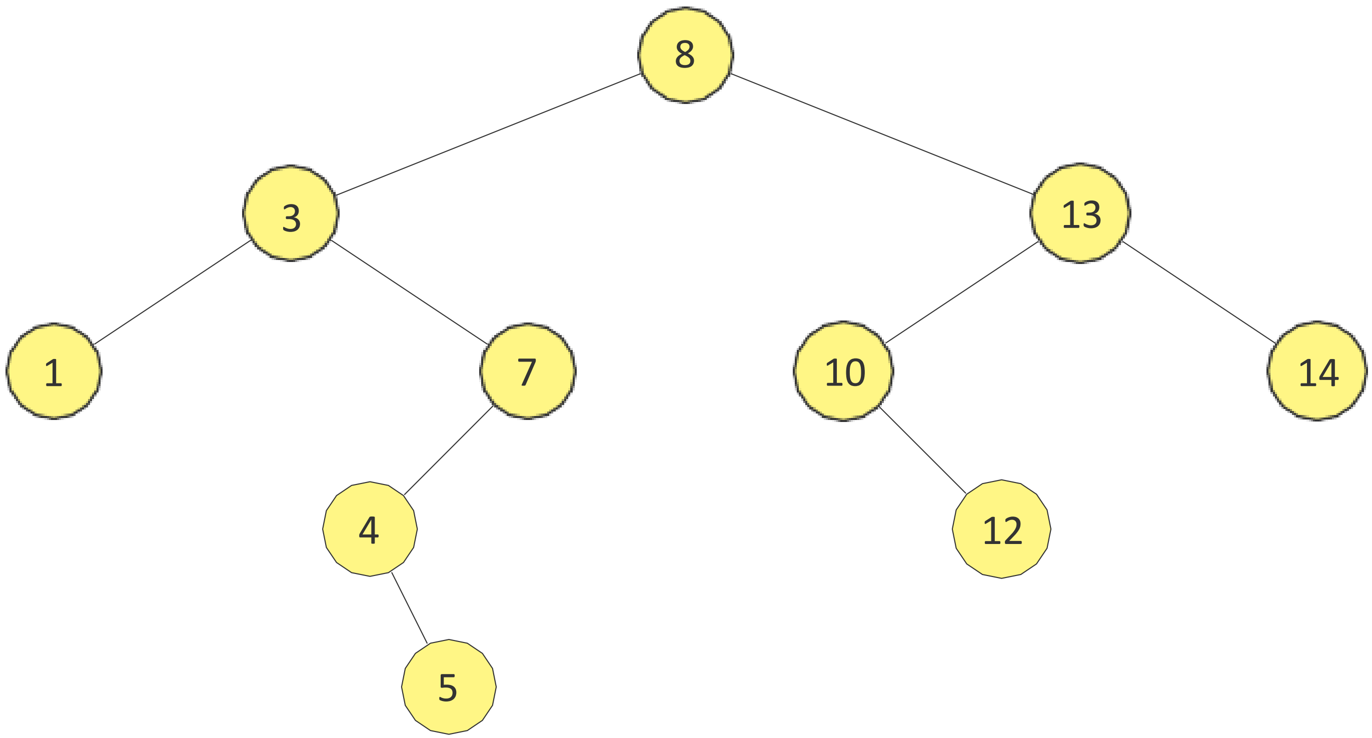
- Para encontrar um registro com uma chave **x**:

1. Compare-a com a chave que está na raiz.
   - Se é **igual**, retorne o registro que está na raiz.
   - Se **x** é **menor**, vá para a sub-árvore esquerda.
   - Se **x** é **maior**, vá para a sub-árvore direita.

2. Repita o processo recursivamente, até que a chave procurada seja encontrada ou um nó folha é atingido.

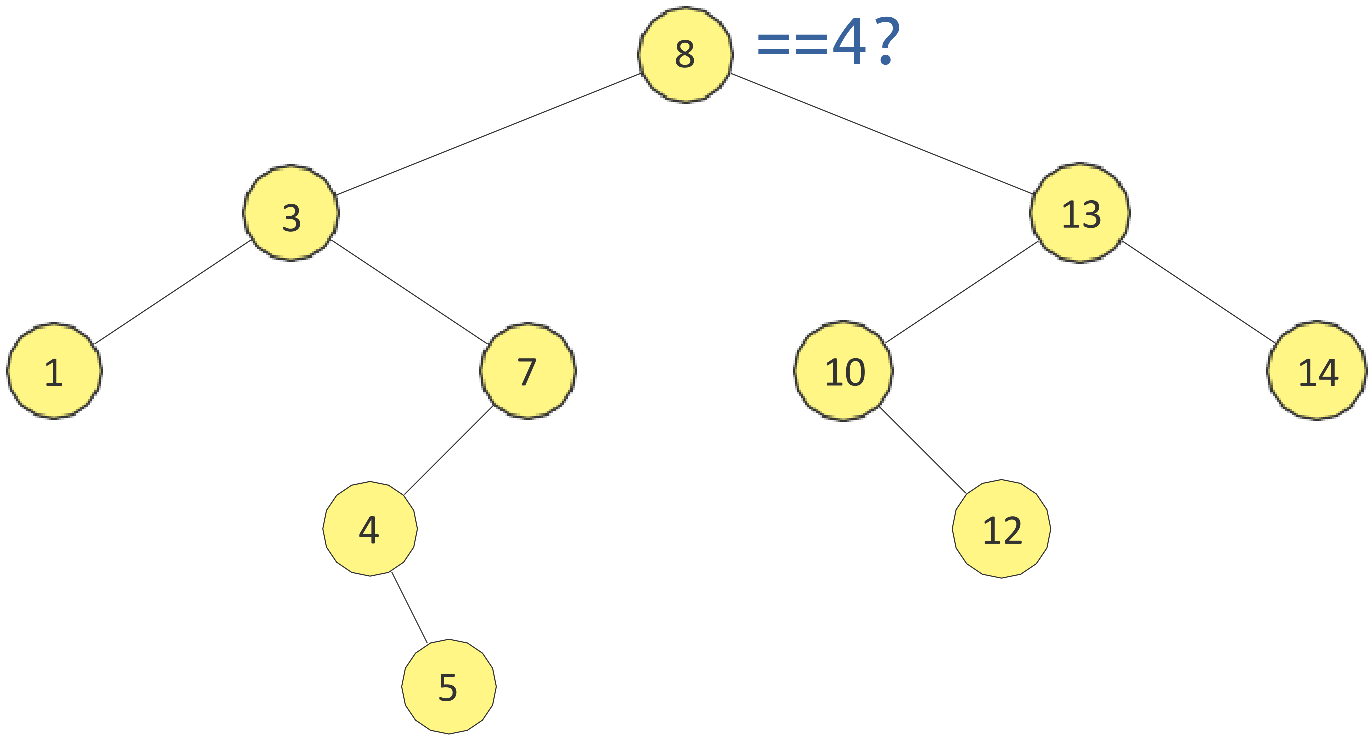
2. Busca por um valor

- Exemplo: busca pelo valor **4**



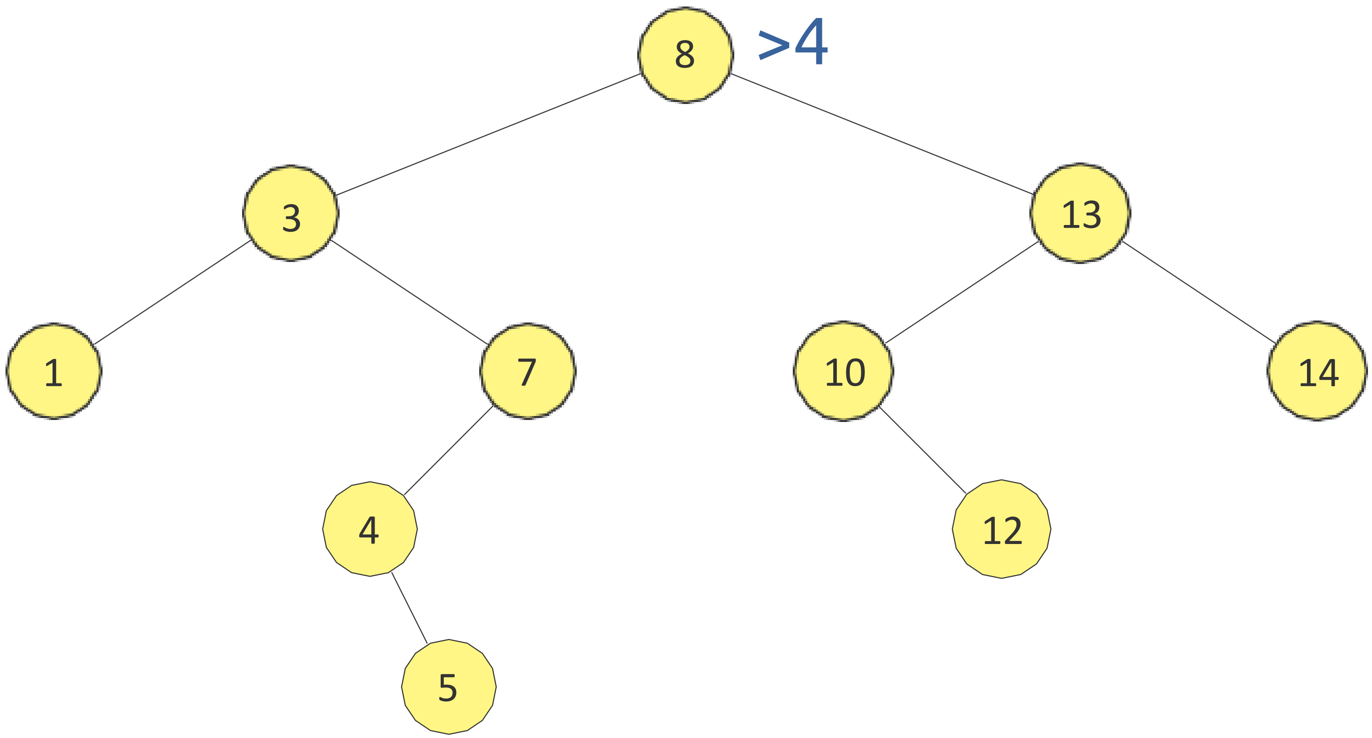
2. Busca por um valor

- Exemplo: busca pelo valor **4**



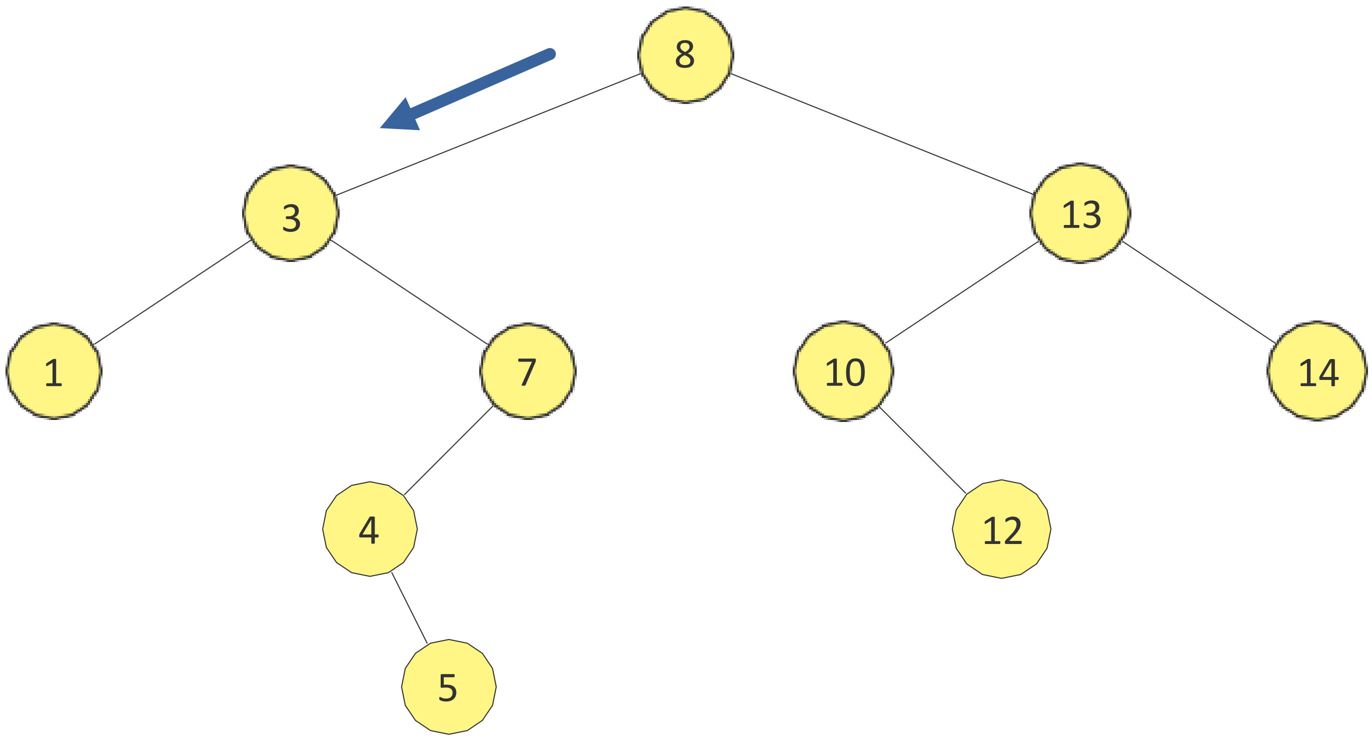
2. Busca por um valor

- Exemplo: busca pelo valor **4**



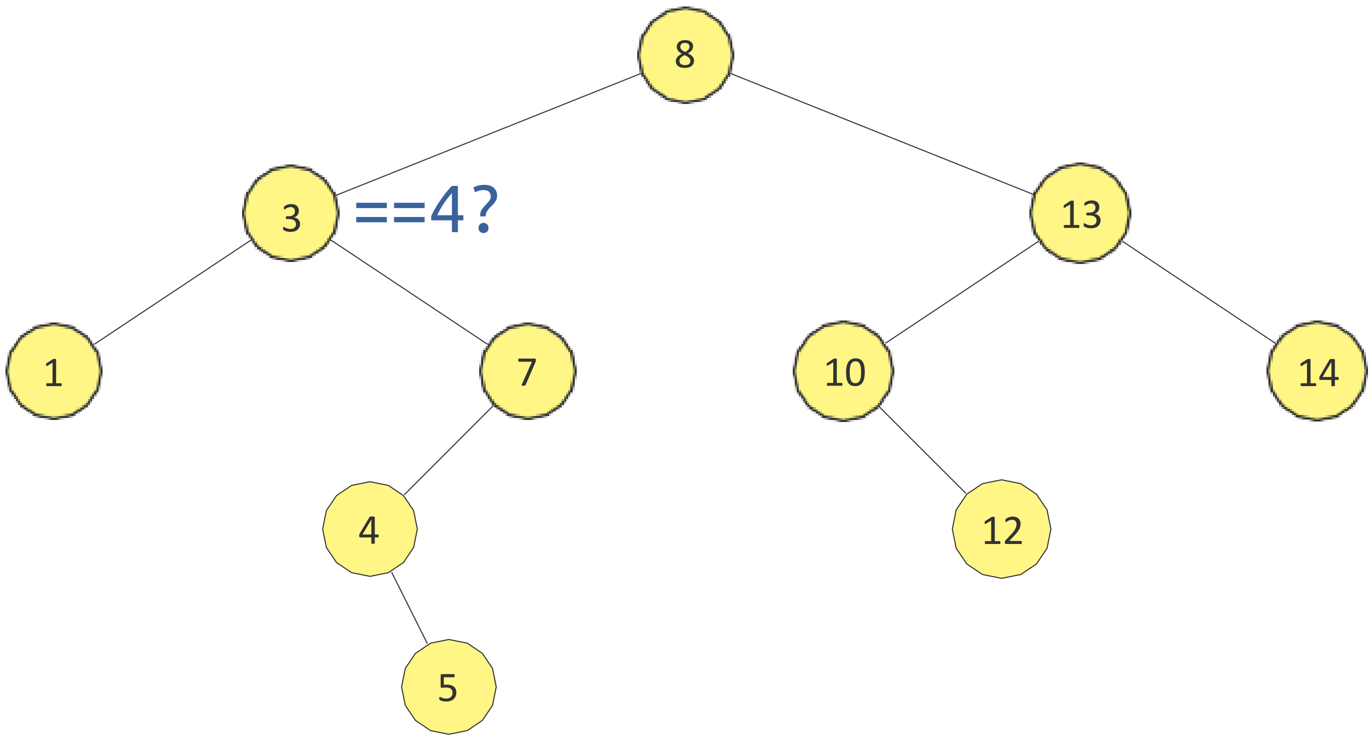
2. Busca por um valor

- Exemplo: busca pelo valor **4**



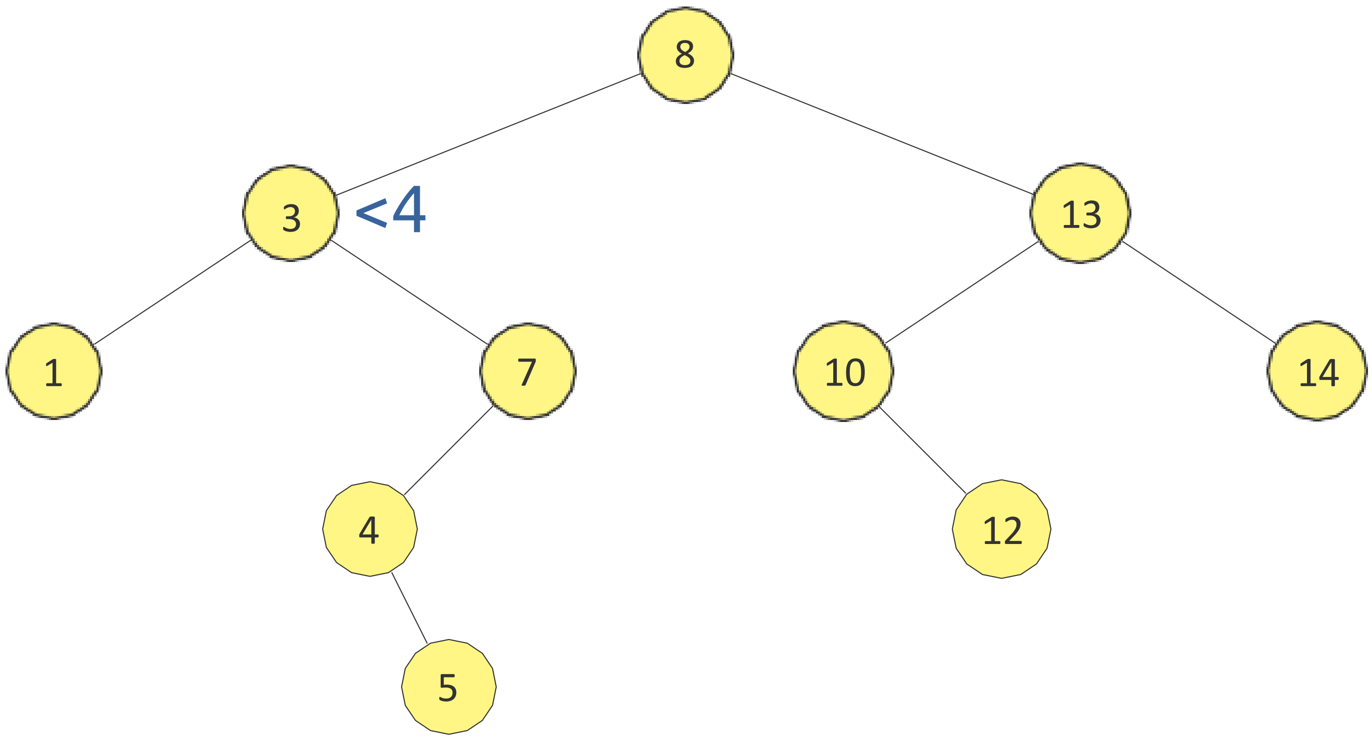
2. Busca por um valor

- Exemplo: busca pelo valor **4**



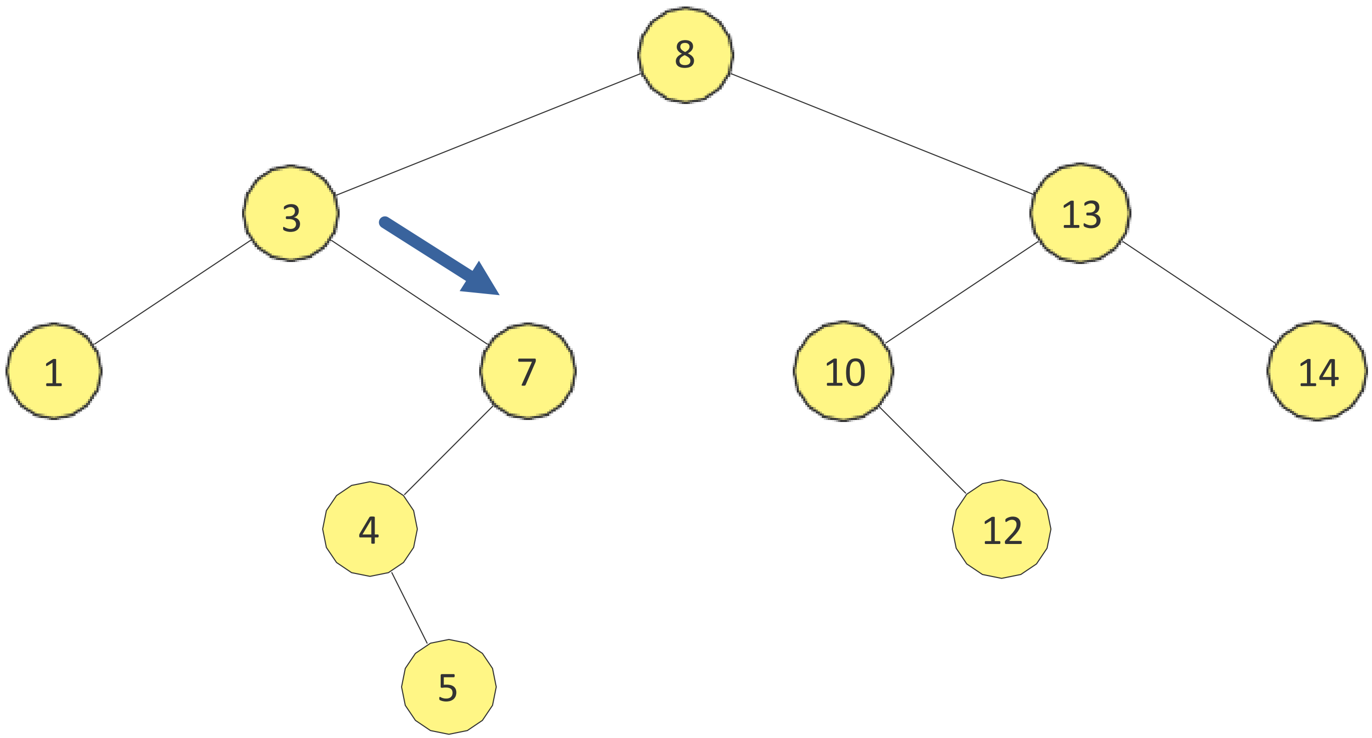
2. Busca por um valor

- Exemplo: busca pelo valor 4



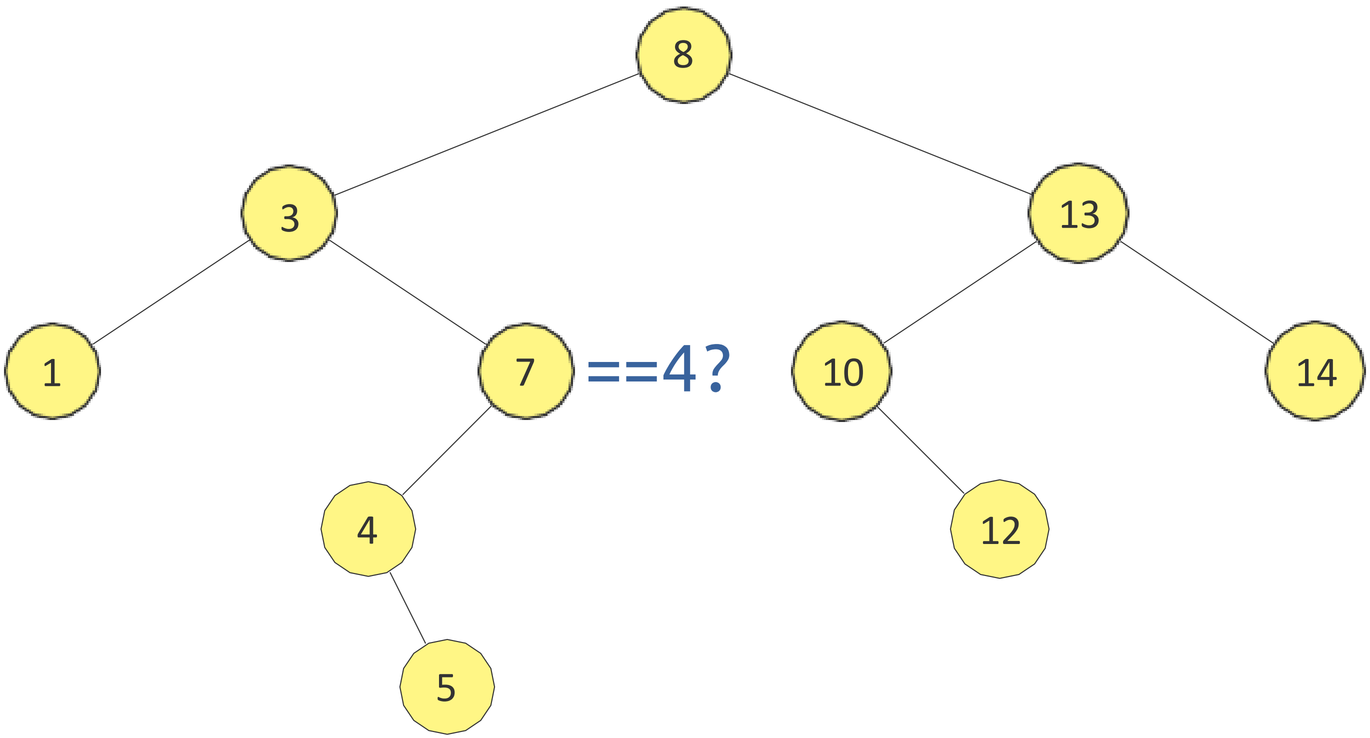
2. Busca por um valor

- Exemplo: busca pelo valor **4**



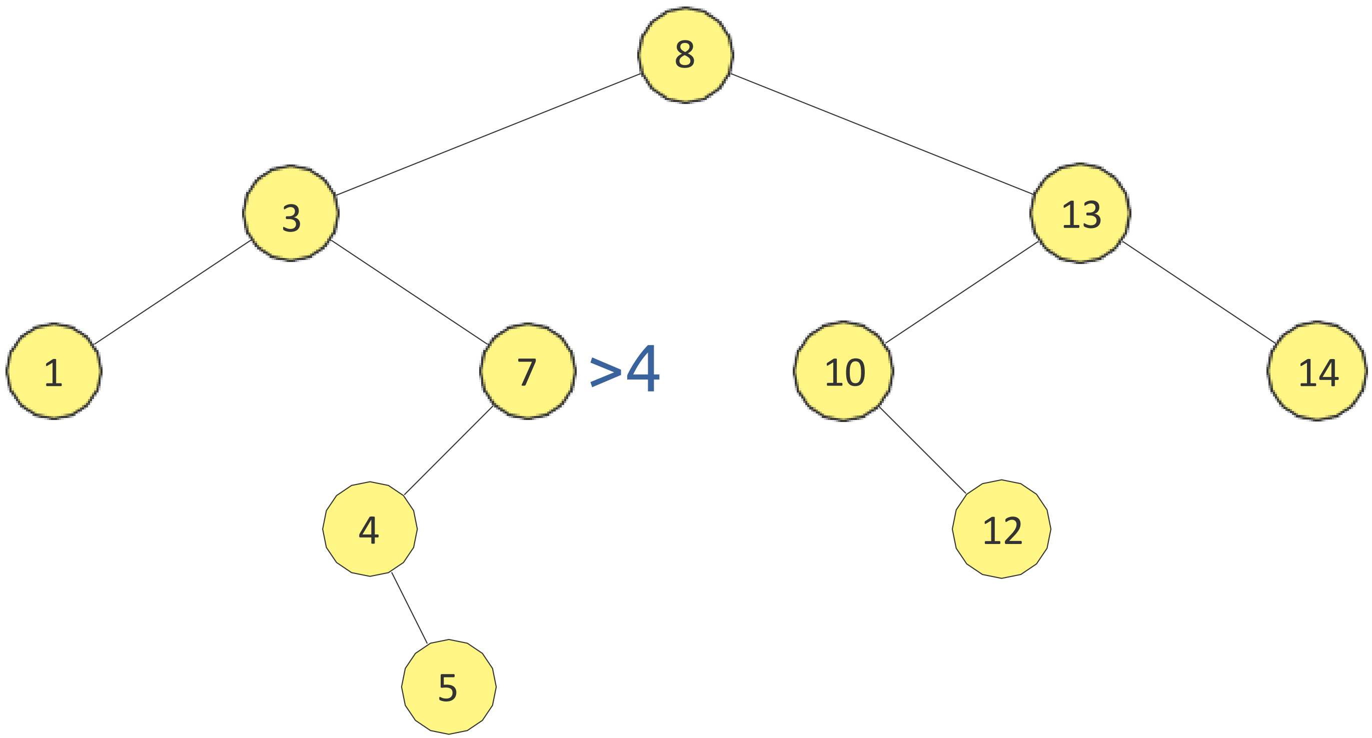
2. Busca por um valor

- Exemplo: busca pelo valor **4**



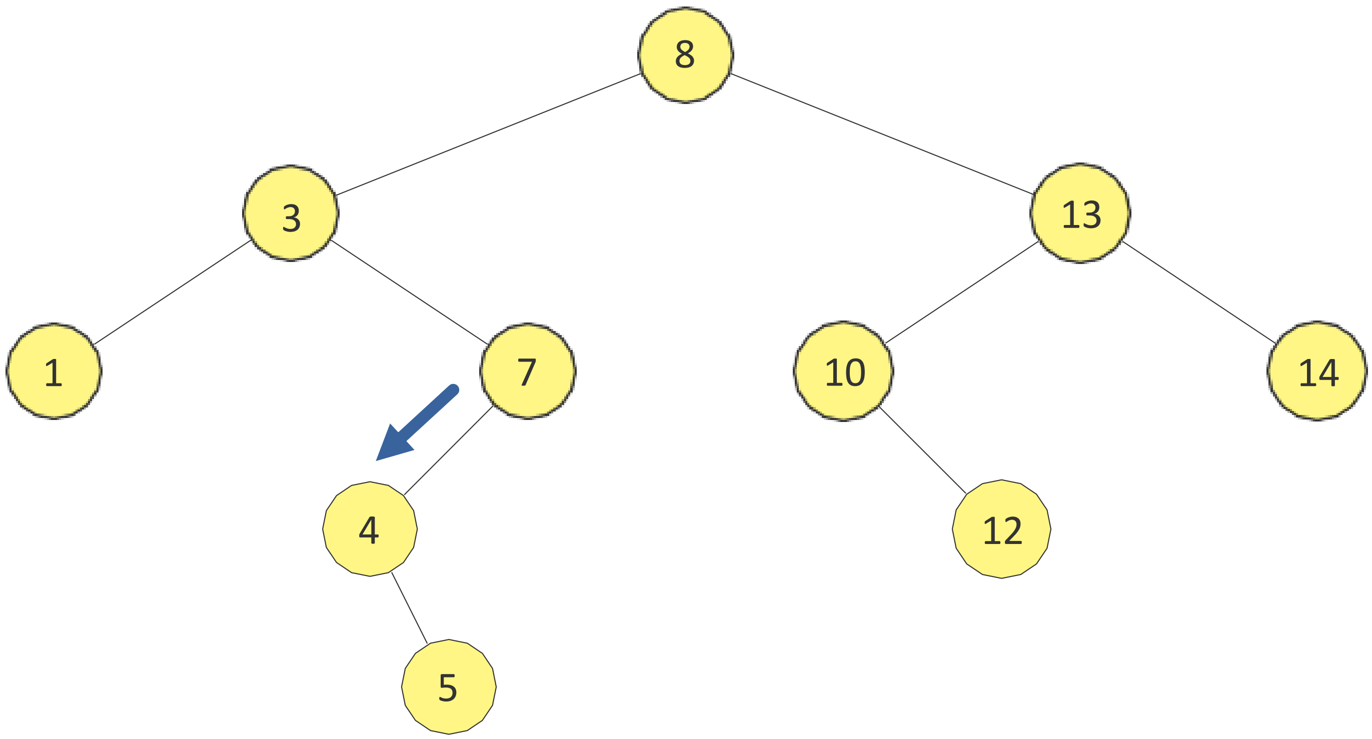
2. Busca por um valor

- Exemplo: busca pelo valor 4



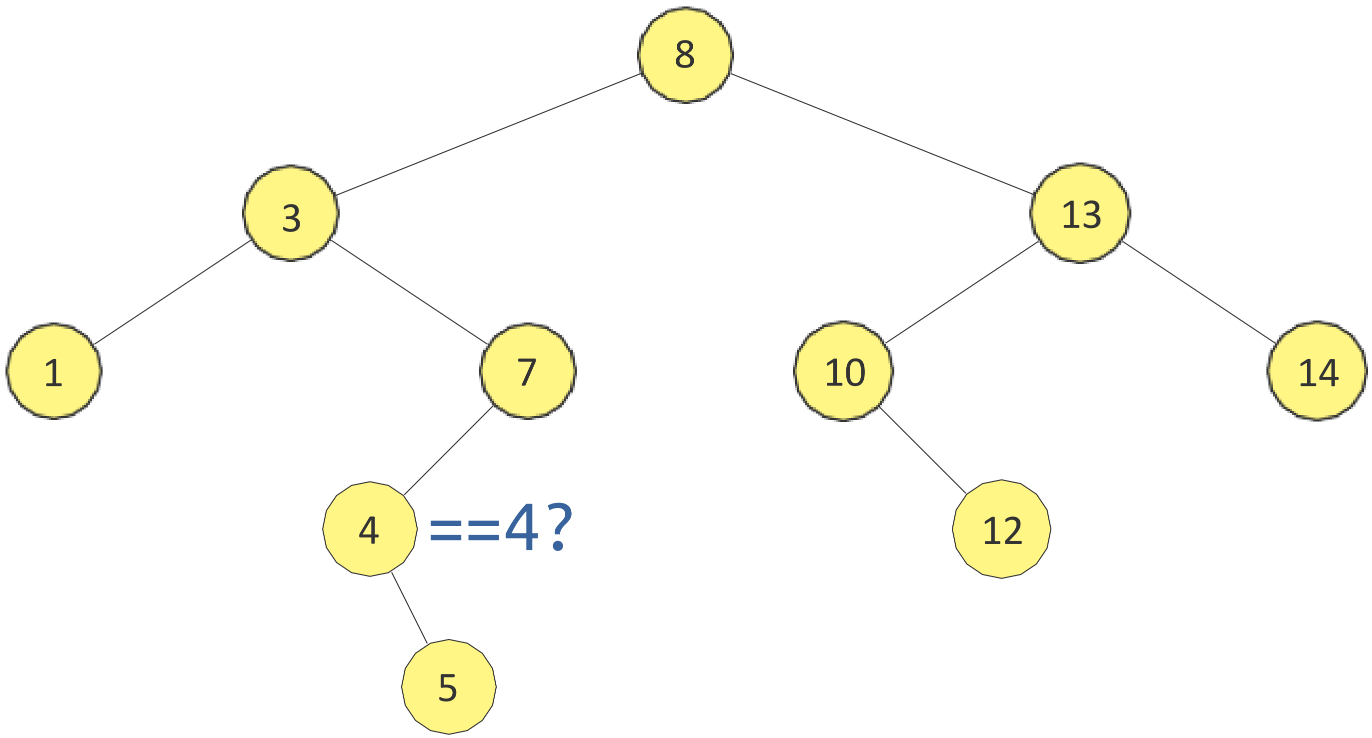
2. Busca por um valor

- Exemplo: busca pelo valor **4**



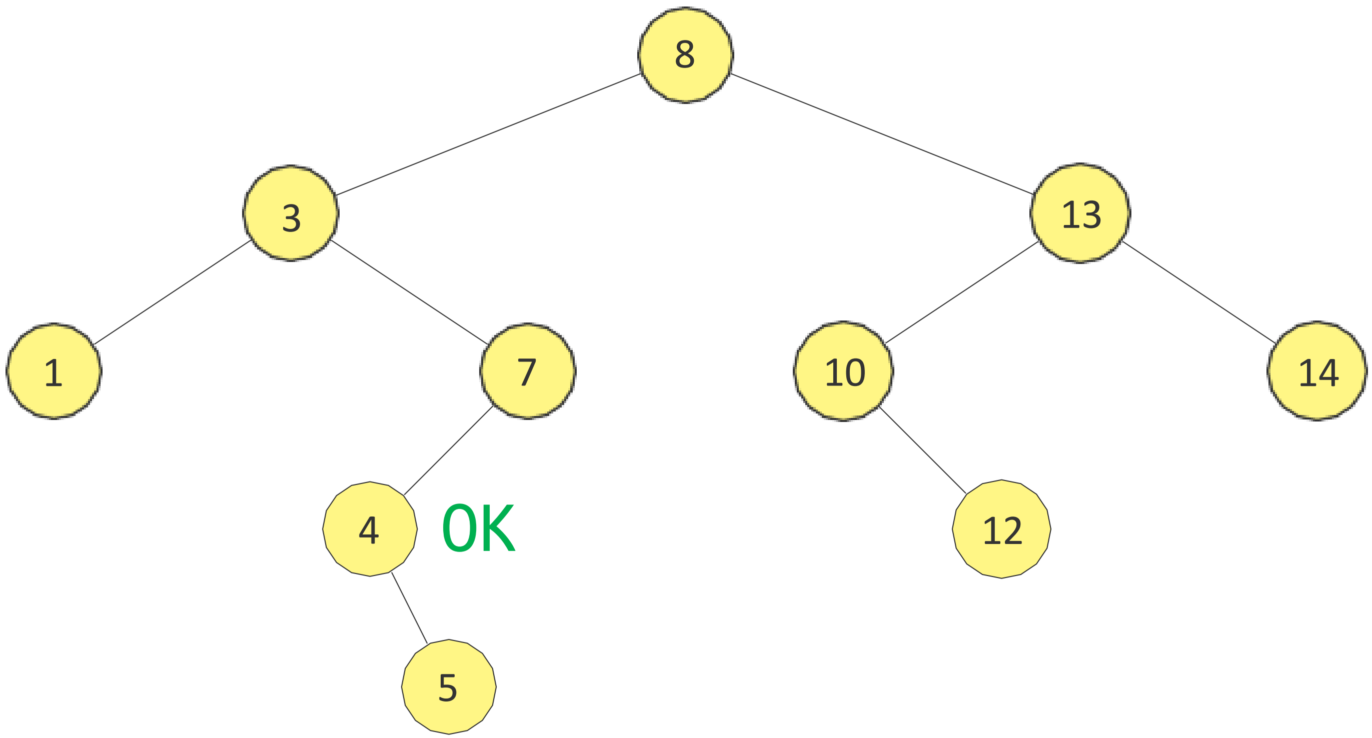
2. Busca por um valor

- Exemplo: busca pelo valor **4**



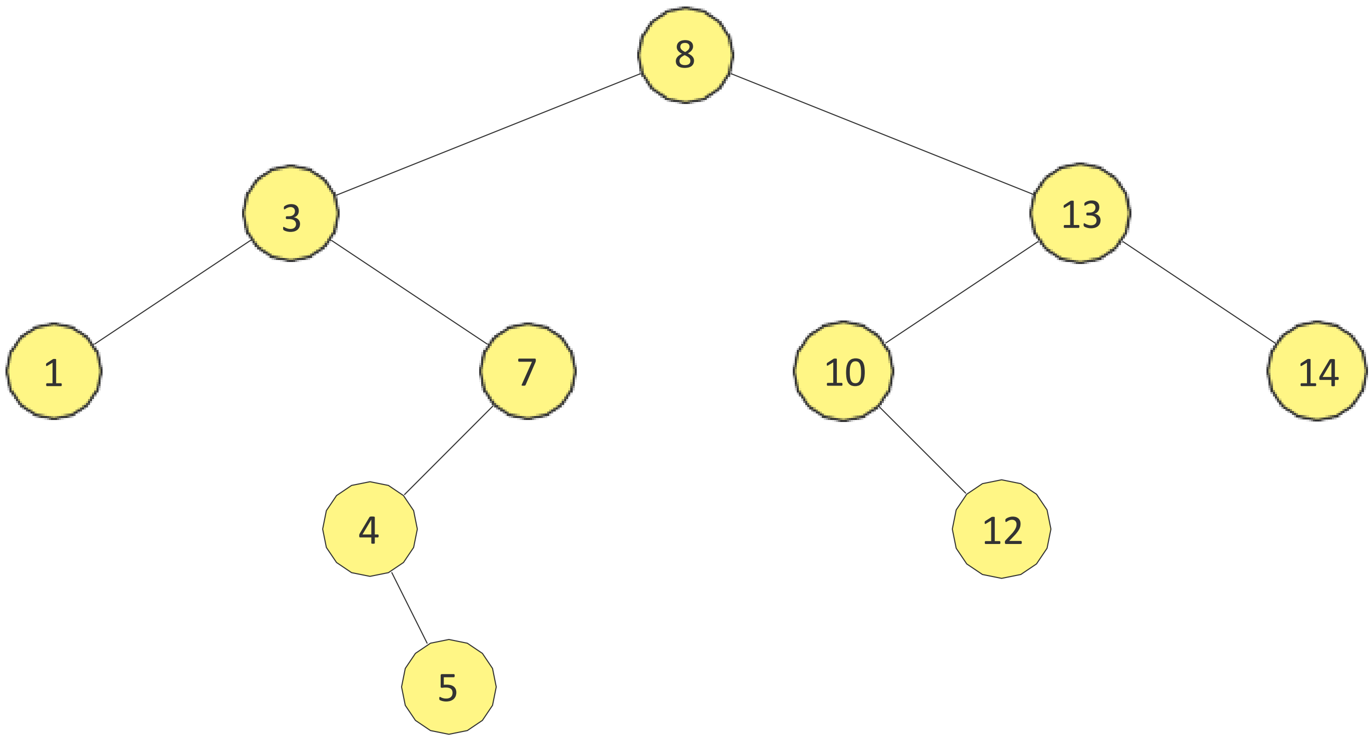
2. Busca por um valor

- Exemplo: busca pelo valor **4**



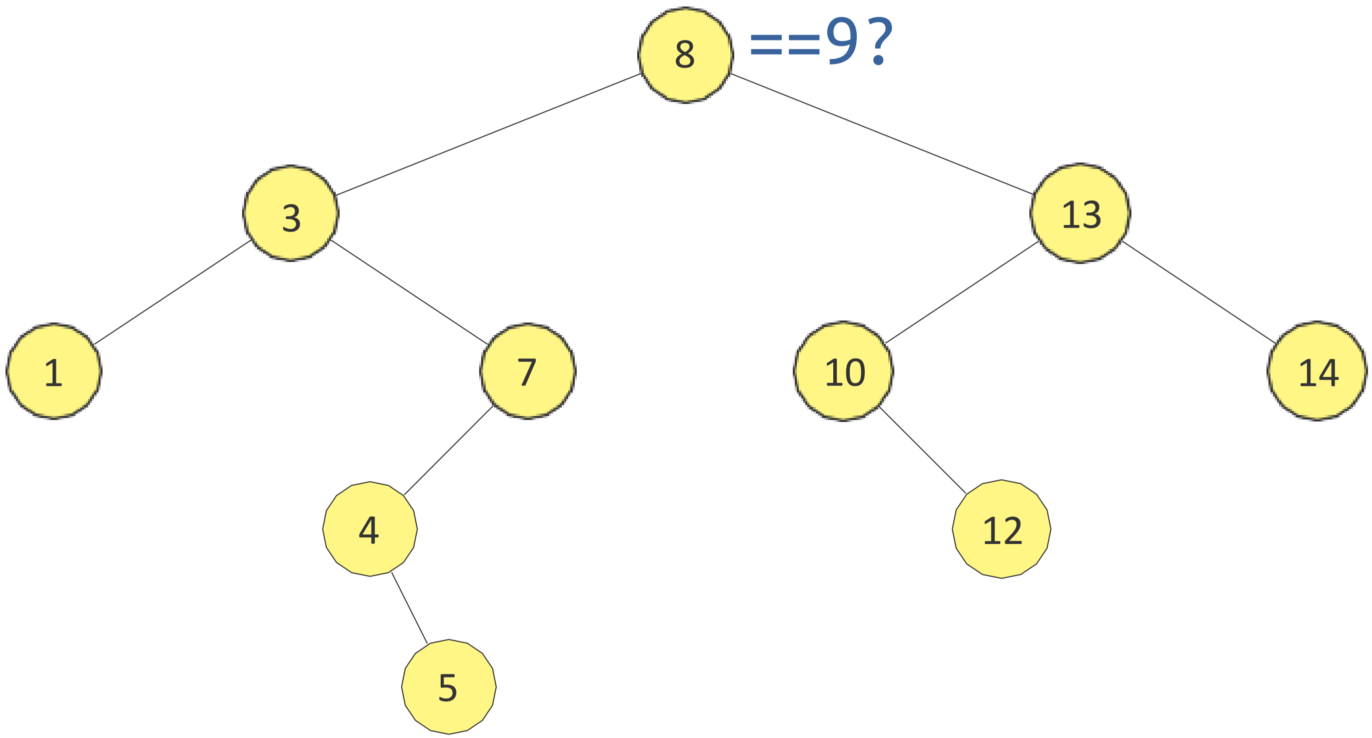
2. Busca por um valor

- Exemplo: busca pelo valor 9



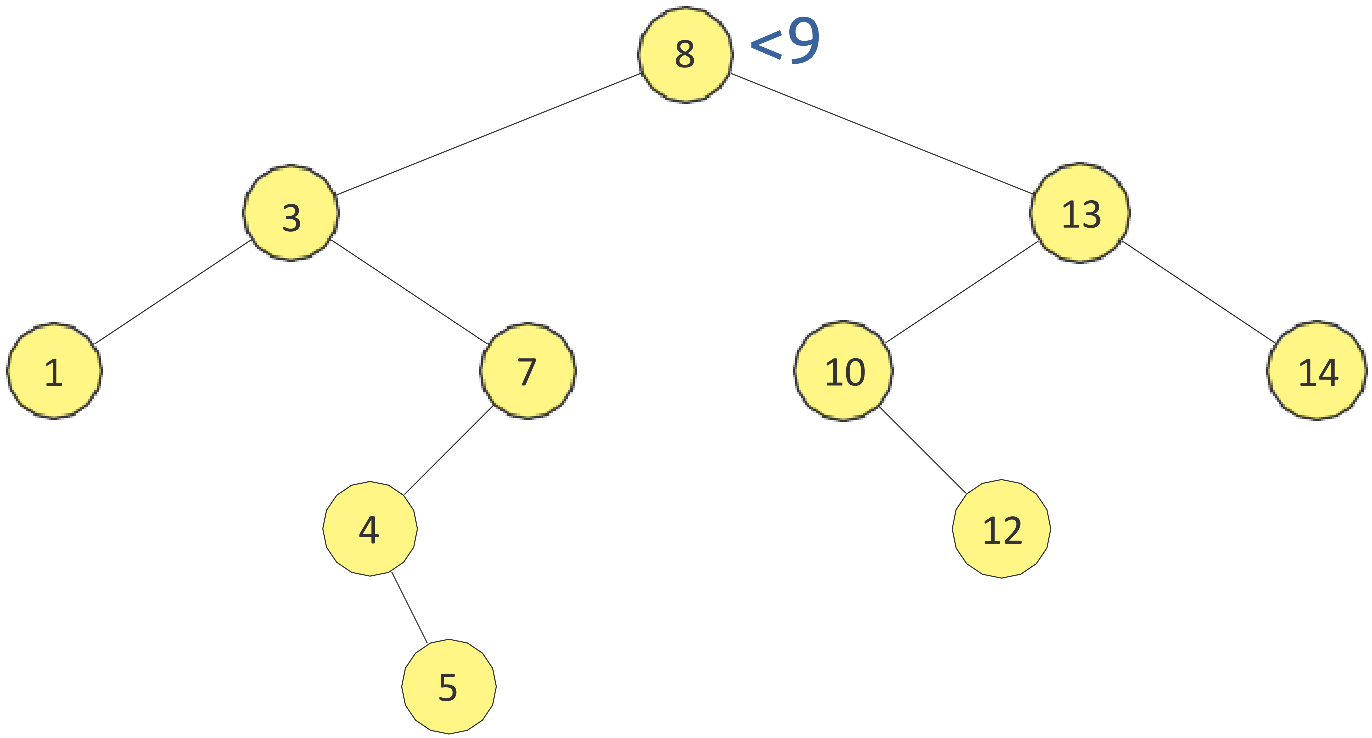
2. Busca por um valor

- Exemplo: busca pelo valor **9**



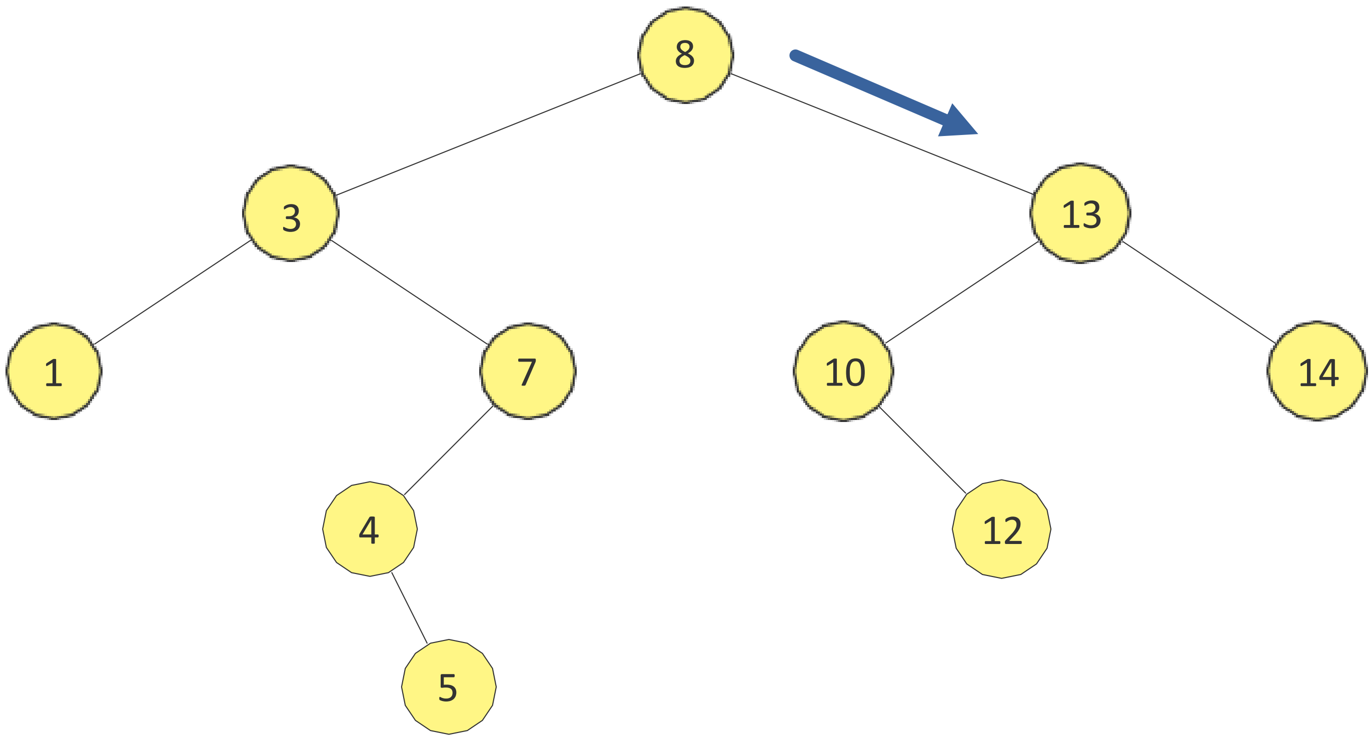
2. Busca por um valor

- Exemplo: busca pelo valor 9



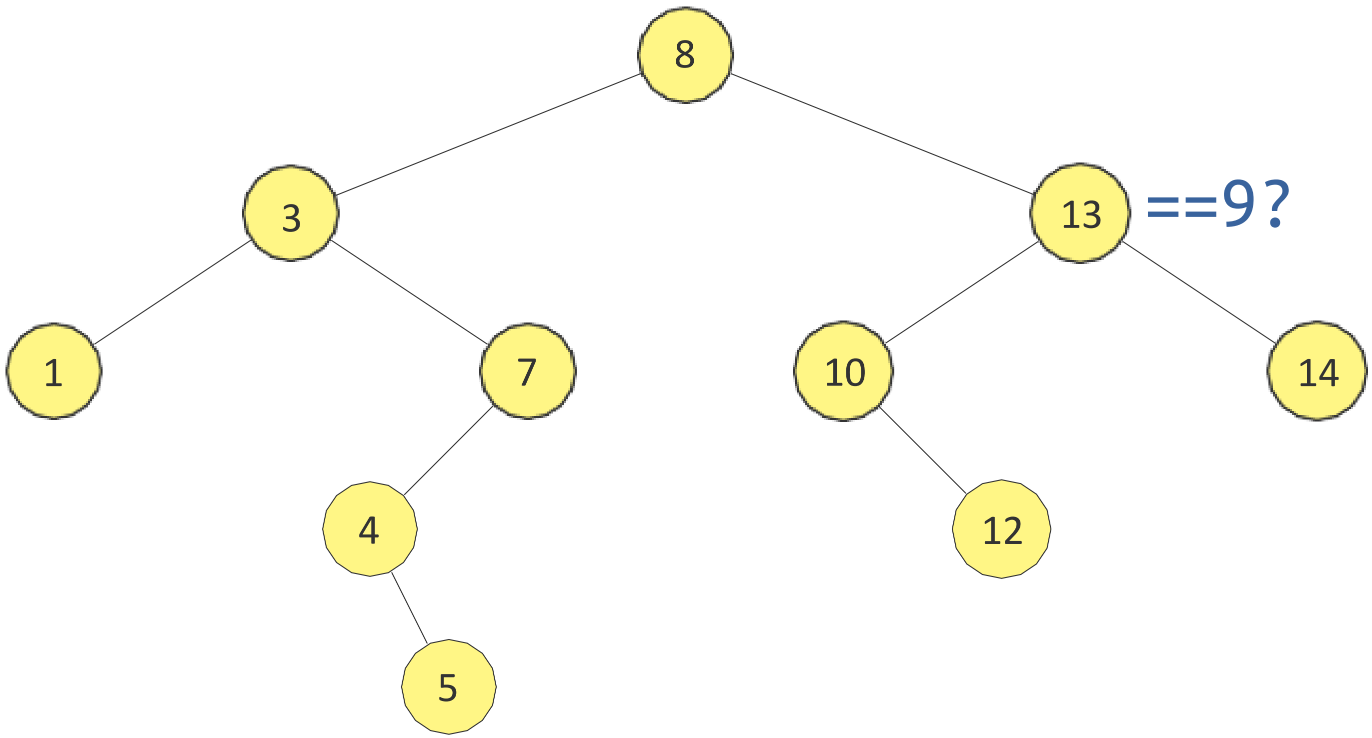
2. Busca por um valor

- Exemplo: busca pelo valor 9



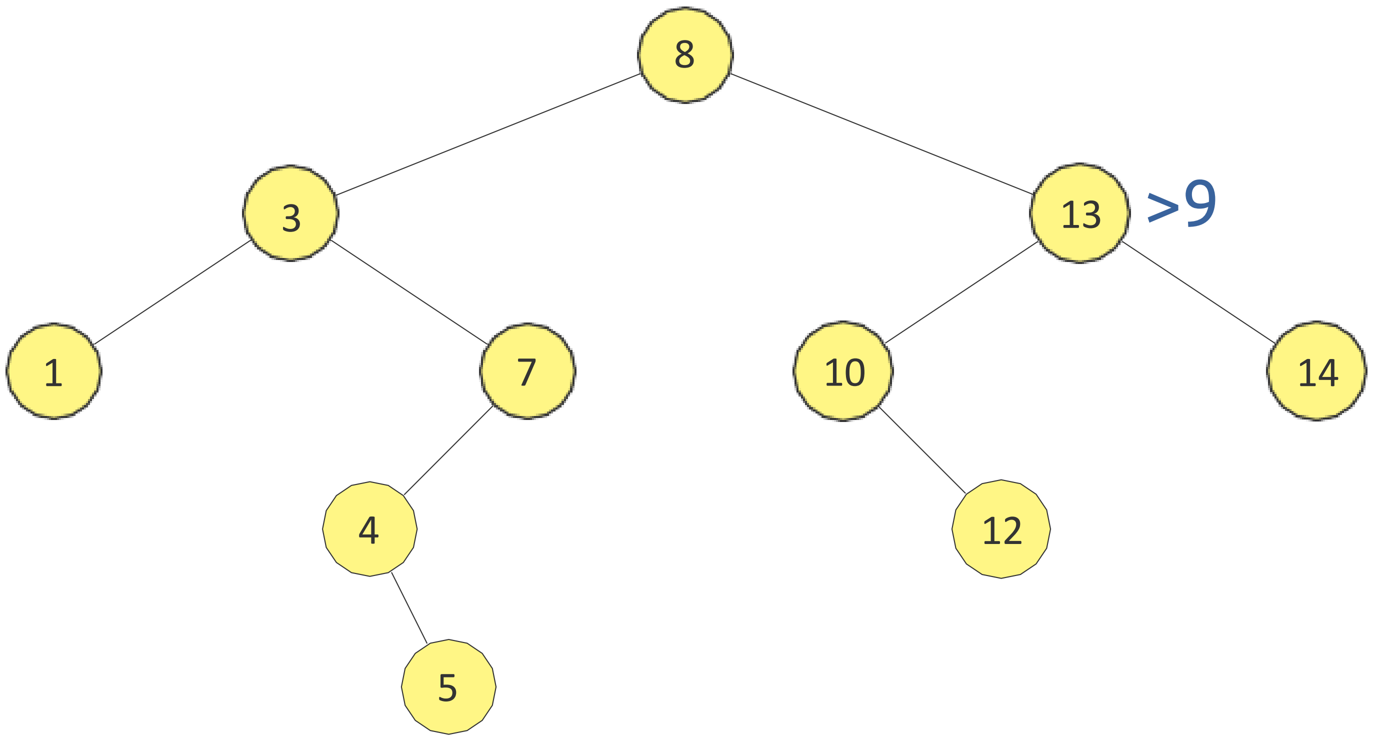
2. Busca por um valor

- Exemplo: busca pelo valor 9



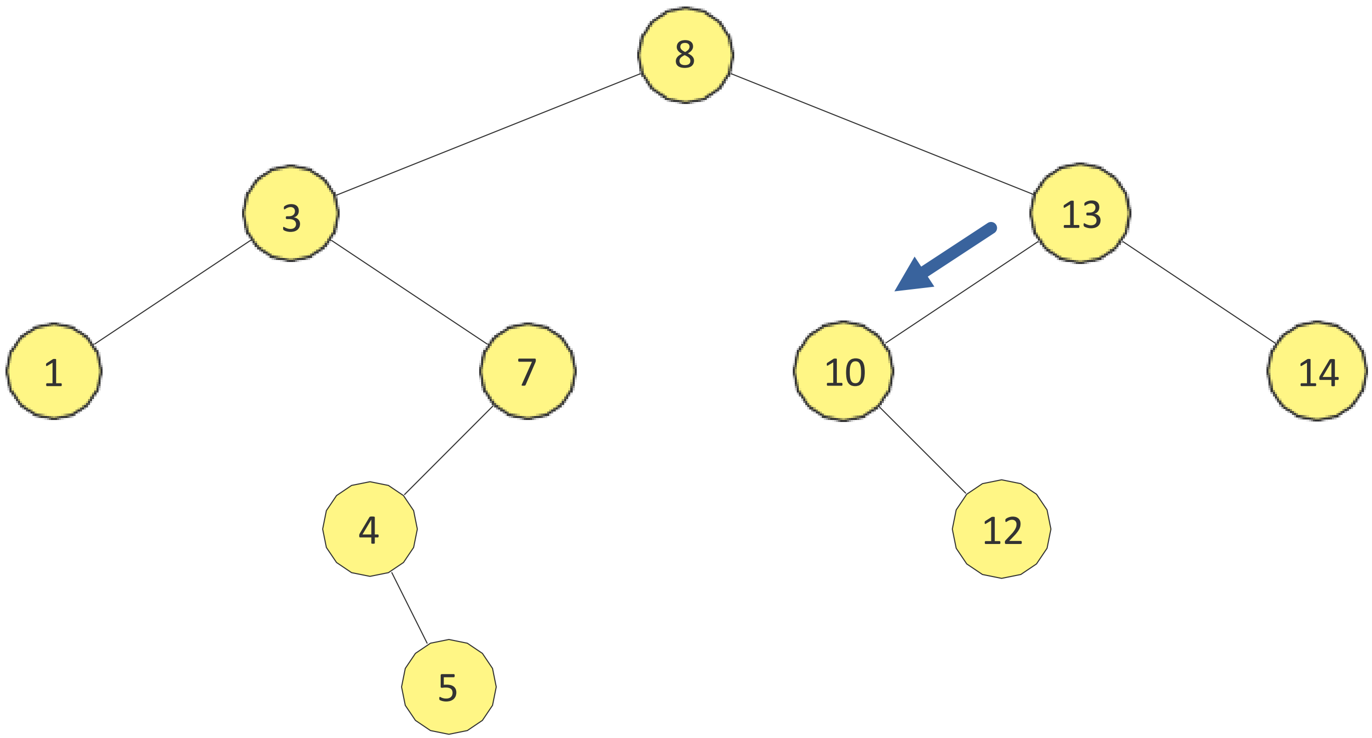
2. Busca por um valor

- Exemplo: busca pelo valor 9



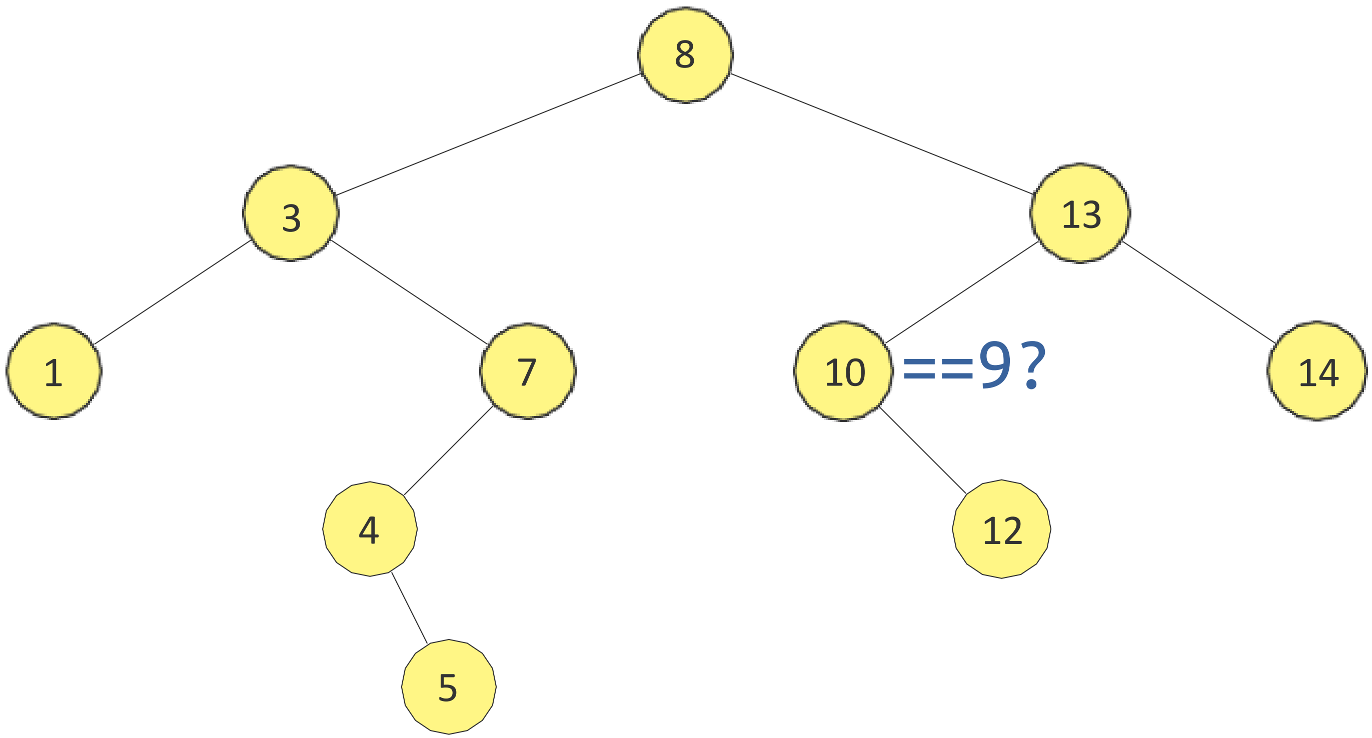
2. Busca por um valor

- Exemplo: busca pelo valor 9



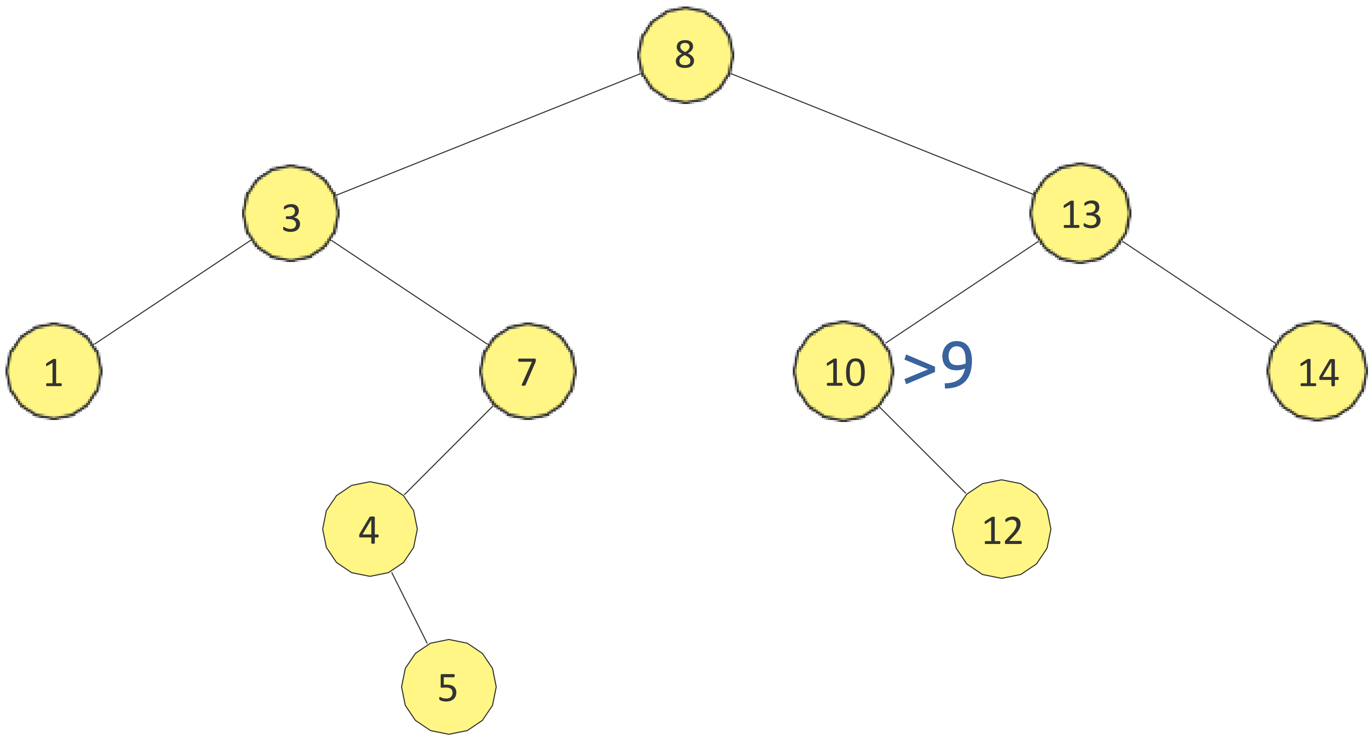
2. Busca por um valor

- Exemplo: busca pelo valor 9



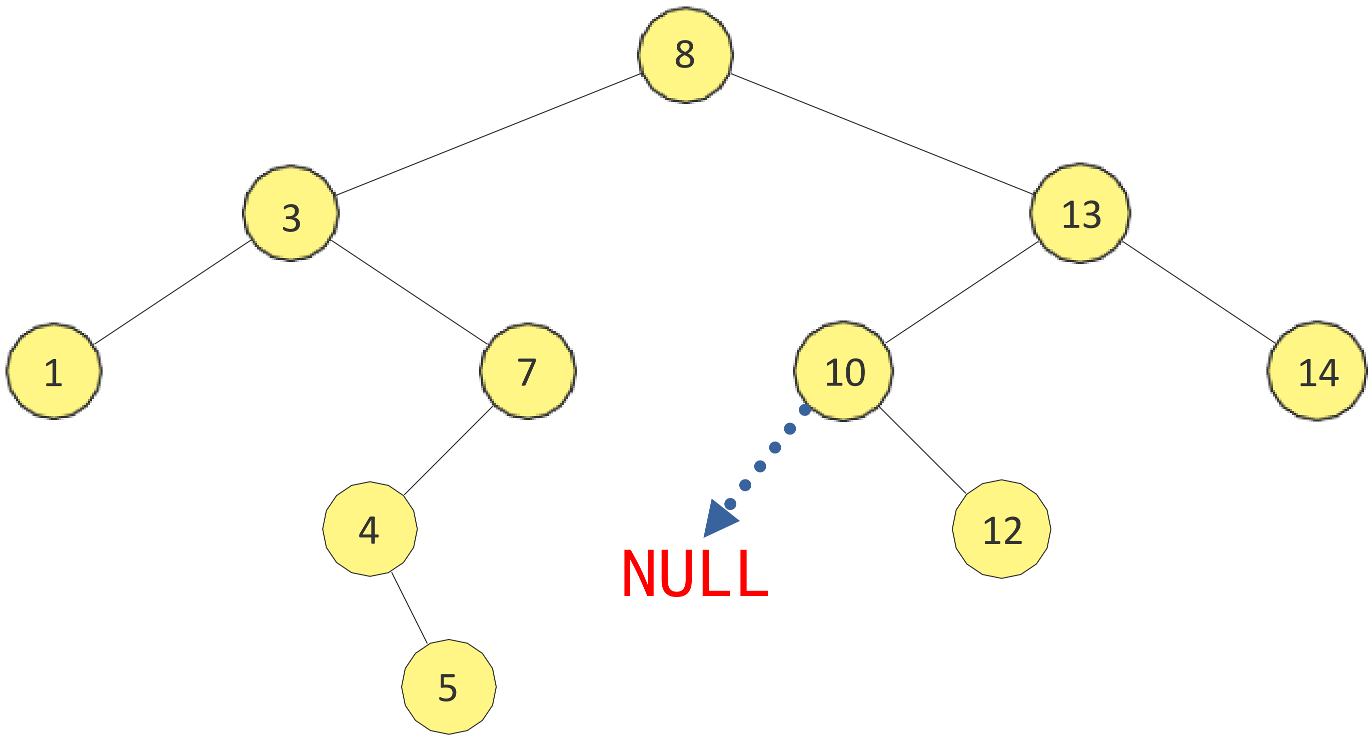
2. Busca por um valor

- Exemplo: busca pelo valor 9



2. Busca por um valor

- Exemplo: busca pelo valor 9



2.1. Implementação – Recursivo

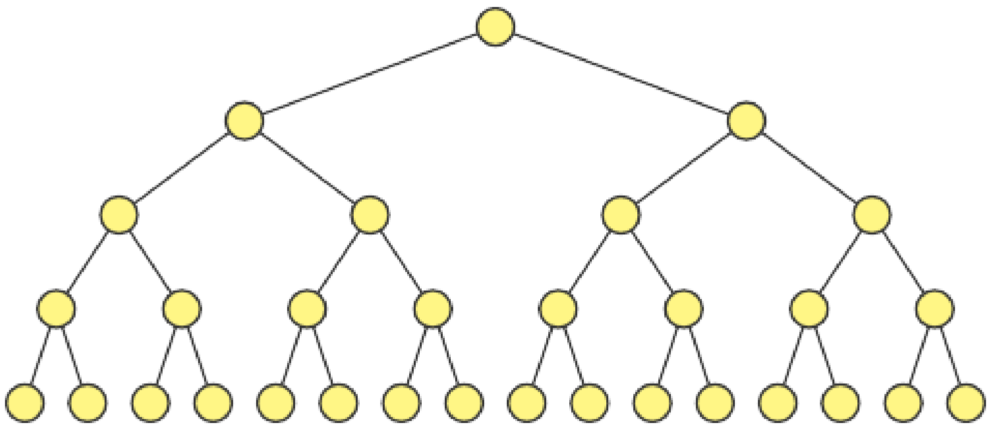
```
int pesquisa (TNo* pRaiz, TRegistro *pX) {
  if (pRaiz == NULL)
    return 0;
  if (pX->chave < pRaiz->reg.chave)
    return pesquisa (pRaiz->pEsq, pX);
  if (pX->chave > pRaiz->reg.chave)
    return pesquisa (pRaiz->pDir, pX);
  /* if (pX->chave == pRaiz->reg.chave) */
  *pX = pRaiz->reg;
  return 1;
}
```

2.1. Implementação – Não Recursivo

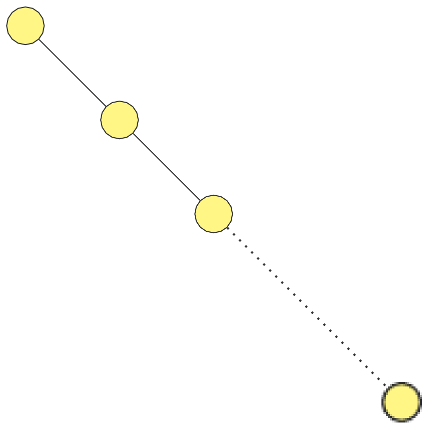
```
int pesquisa (TNo* pRaiz, TRegistro *pX) {
  TNo* pAux = pRaiz;
  while (pAux != NULL) {
    if (pX->chave == pAux->reg.chave) {
      *pX = pAux->Reg;
      return 1; }
    if (pX->chave > pAux->reg.chave)
      pAux = pAux->pDir;
    else /* if (pX->chave < pAux->reg.chave) */
      pAux = pAux->pEsq;
  }
  return 0;
}
```

2.2. Análise

- O tempo de execução dos algoritmos para árvores binárias de busca dependem muito do formato das árvores.



Melhor árvore: **O(lg n)**



Pior árvore: **O(n)**

2.2. Análise

- O número de comparações em uma pesquisa com sucesso:
  - Melhor caso: **O(1)**
  - Pior caso: **O(n)**
  - Caso médio: **O(log n)**

- Para obter o pior caso basta que as chaves sejam inseridas em ordem crescente ou decrescente. Neste caso a árvore resultante é uma lista linear, cujo número médio de comparações é **(n + 1)/2**.

- Para uma árvore de pesquisa aleatória o número esperado de comparações para recuperar um registro qualquer é cerca de **1,39 log n**, apenas 39% pior que a árvore completamente balanceada.

3. Inserção

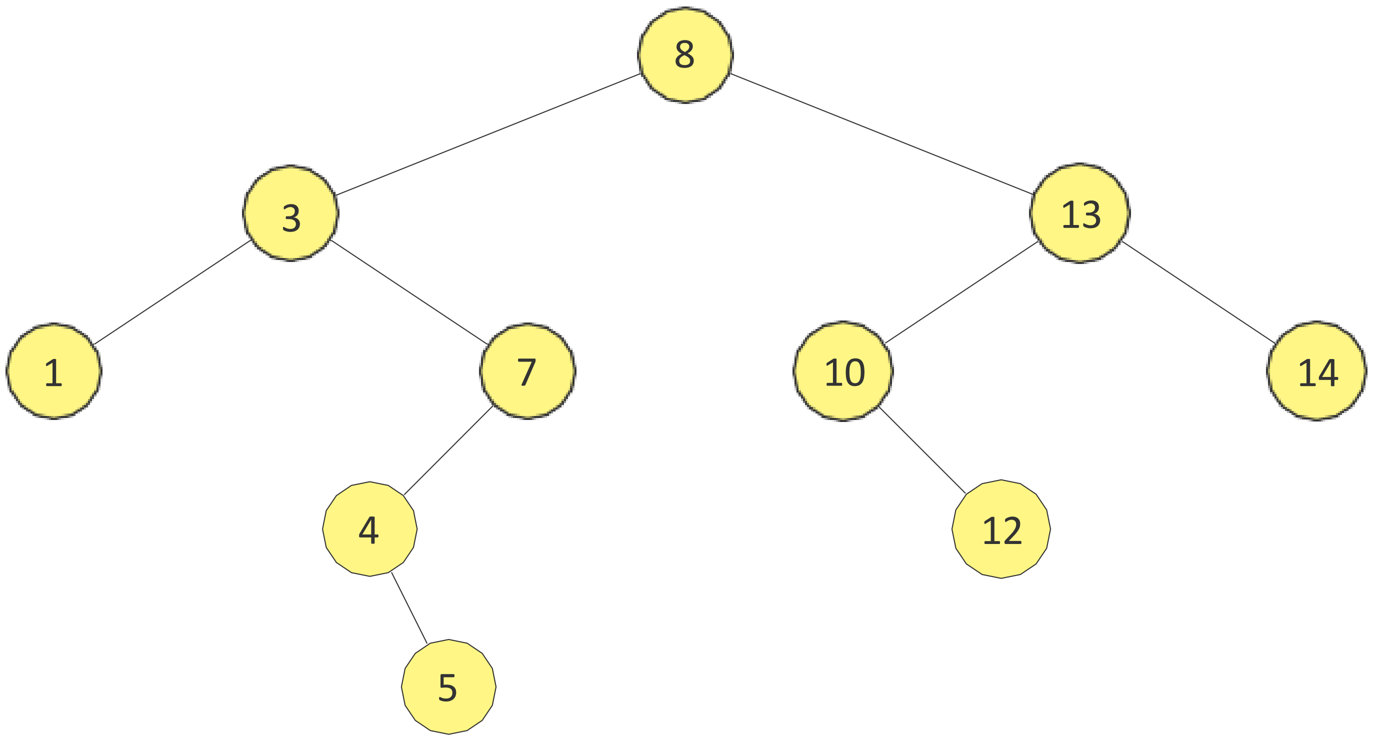
- Para inserir um valor **x** na árvore:
  - Se a raiz é **igual** a **`NULL`**, insira o nó.
  - Se **x** é **menor** do que a raiz: vá para a sub-árvore **esquerda**.
  - Se **x** é **maior** do que a raiz: vá para a sub-árvore **direita.**
  - Aplique o método **recursivamente.**
    - Pode ser feito sem recursão
- Dessa forma, percorremos um conjunto de nós da árvore até chegar ao nó **folha** que irá se tornar o **pai** do novo nó.

3. Inserção

- Em outras palavras:
  - Onde inserir?
    - Atingir um apontador nulo em um processo de busca significa uma pesquisa sem sucesso.
    - O apontador nulo atingido é o ponto de inserção.
  - Como inserir?
    - Criar célula contendo registro.
    - Procurar posição na árvore.
    - Se o registro não estiver na árvore, inserir o registro.

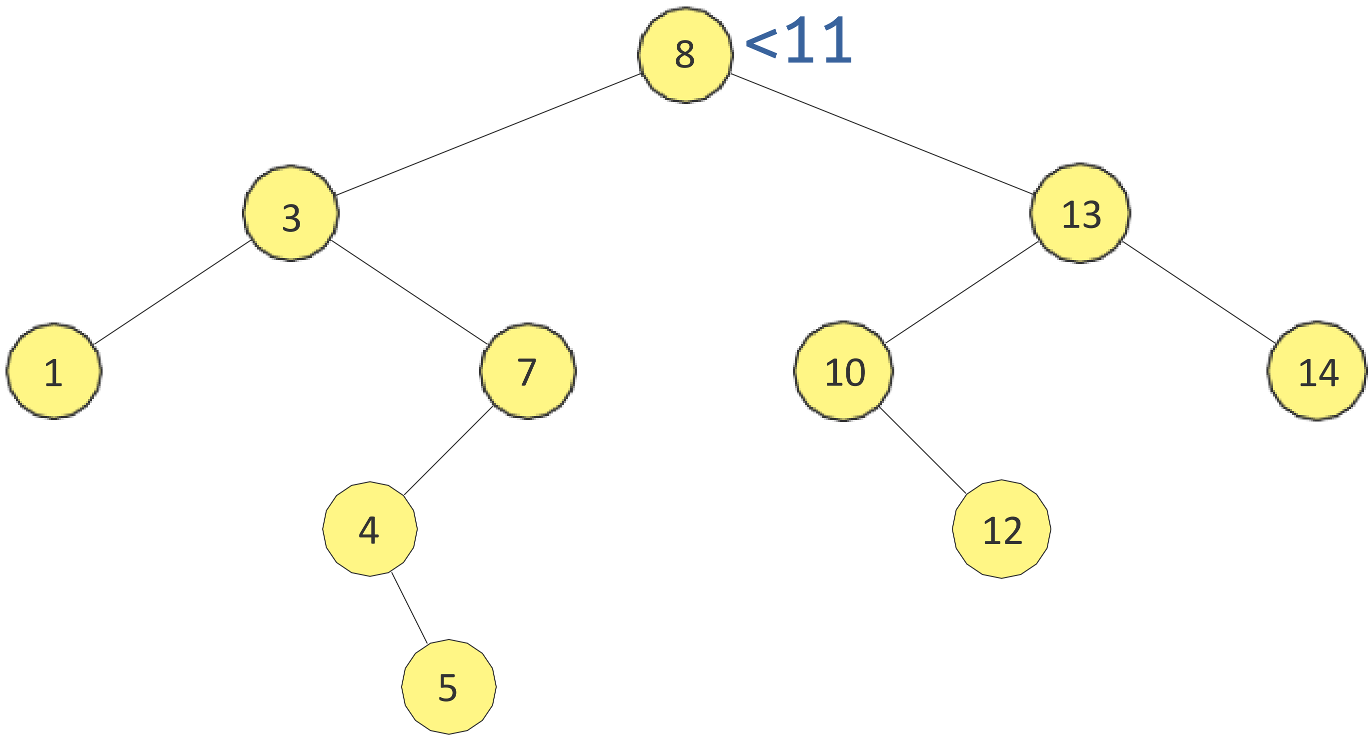
3. Inserção

- Exemplo: inserir o valor 11



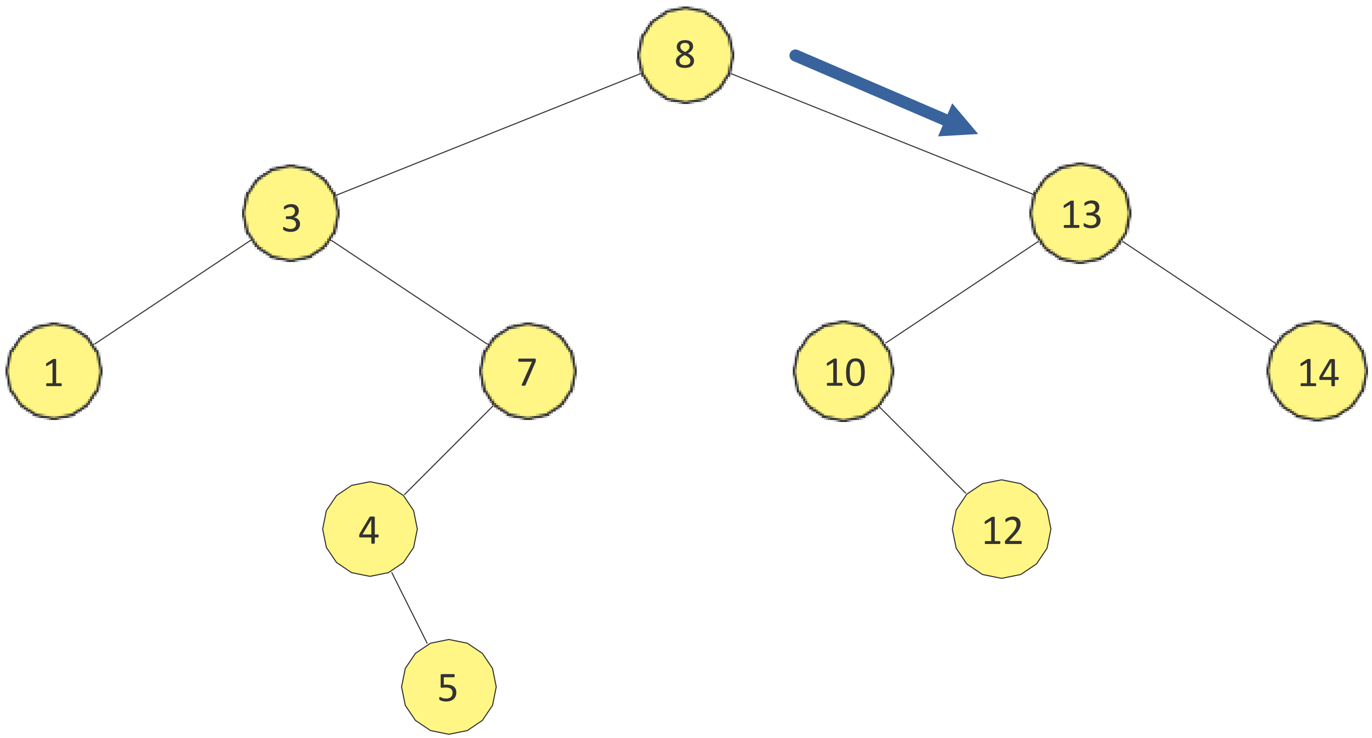
3. Inserção

- Exemplo: inserir o valor 11



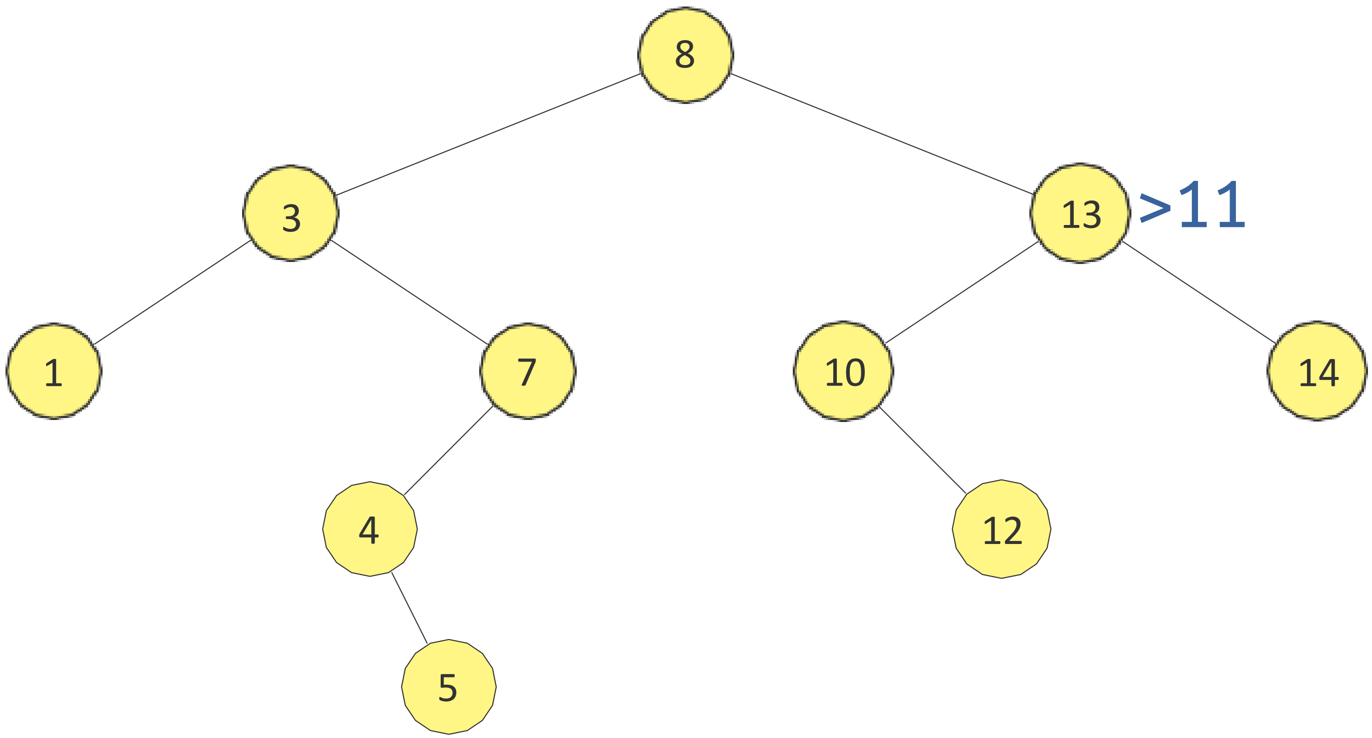
3. Inserção

- Exemplo: inserir o valor 11



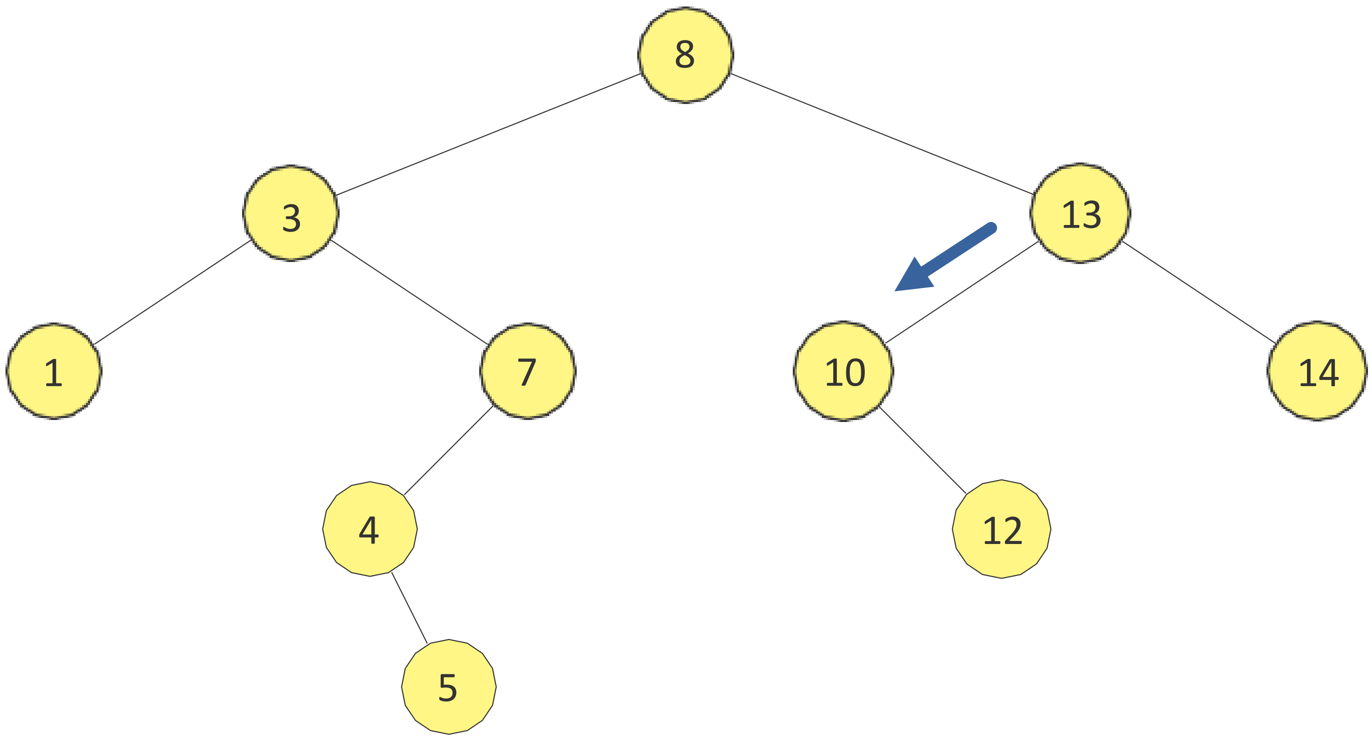
3. Inserção

- Exemplo: inserir o valor 11



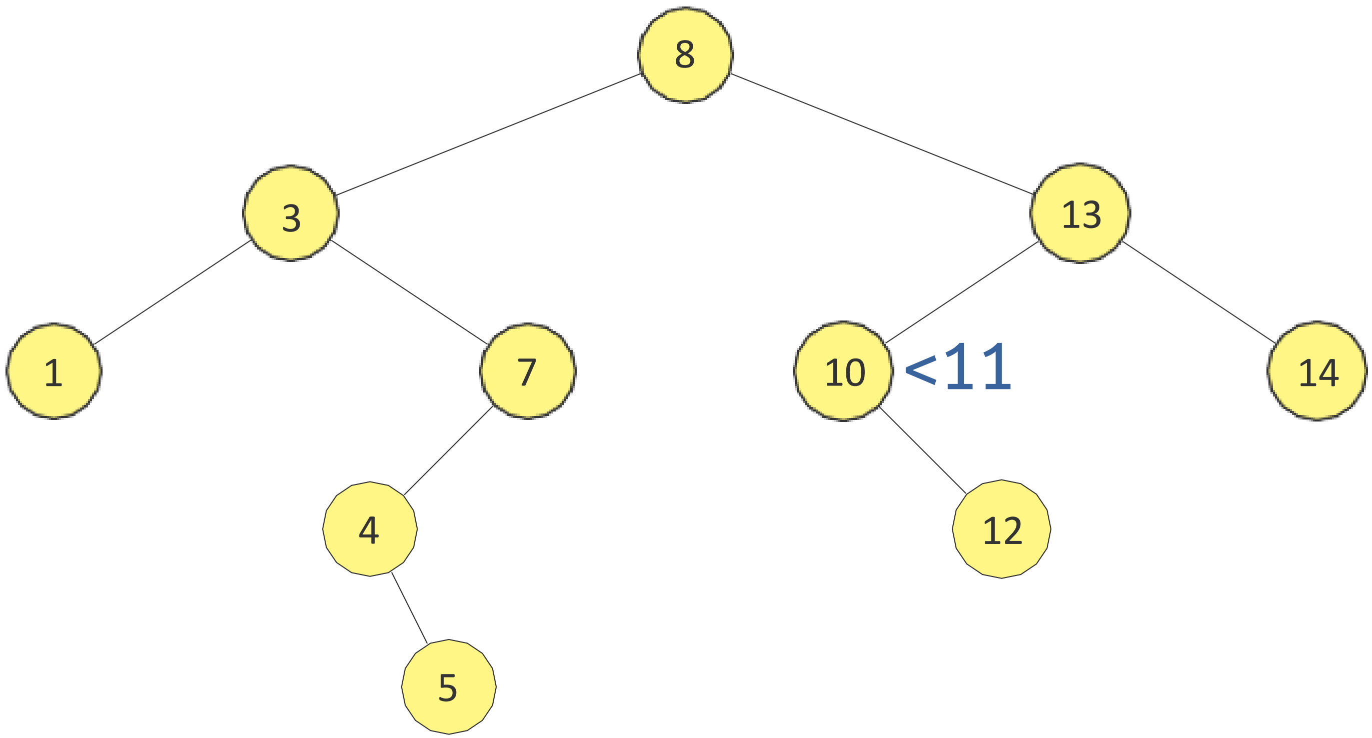
3. Inserção

- Exemplo: inserir o valor 11



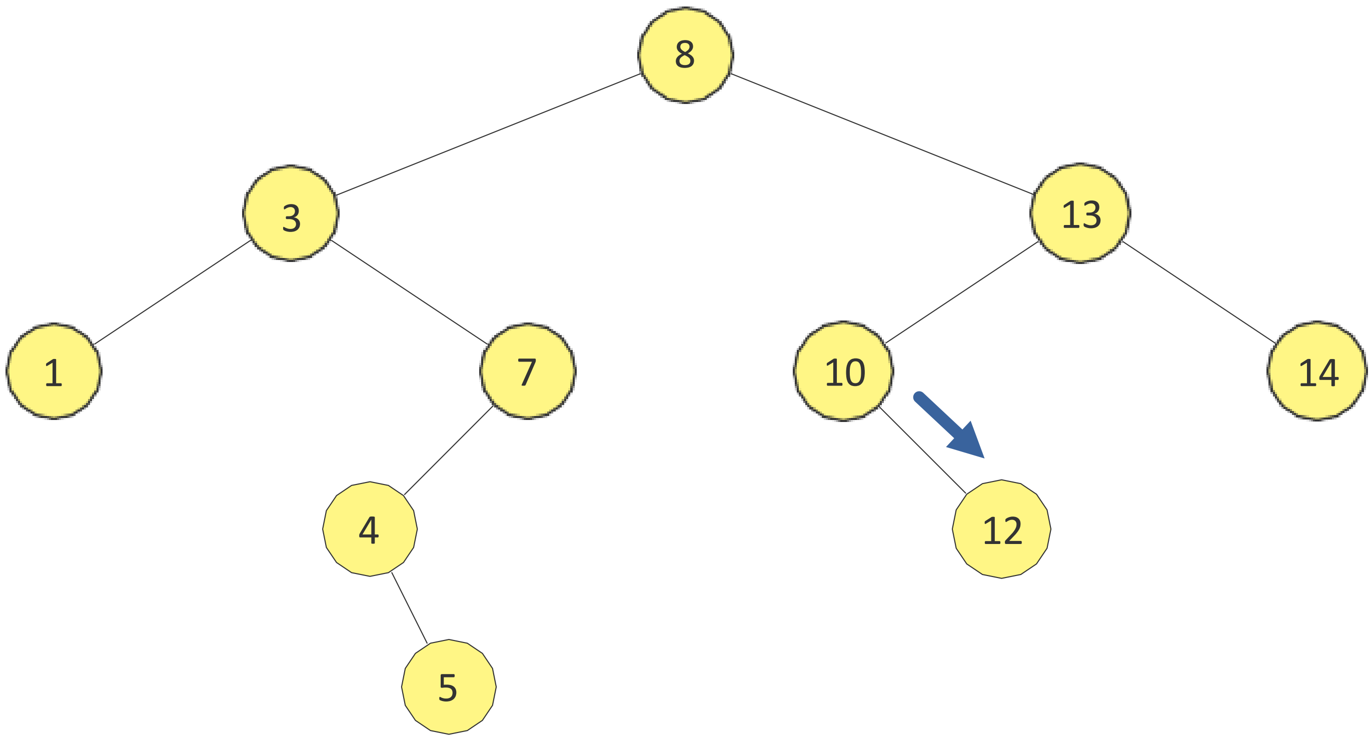
3. Inserção

- Exemplo: inserir o valor 11



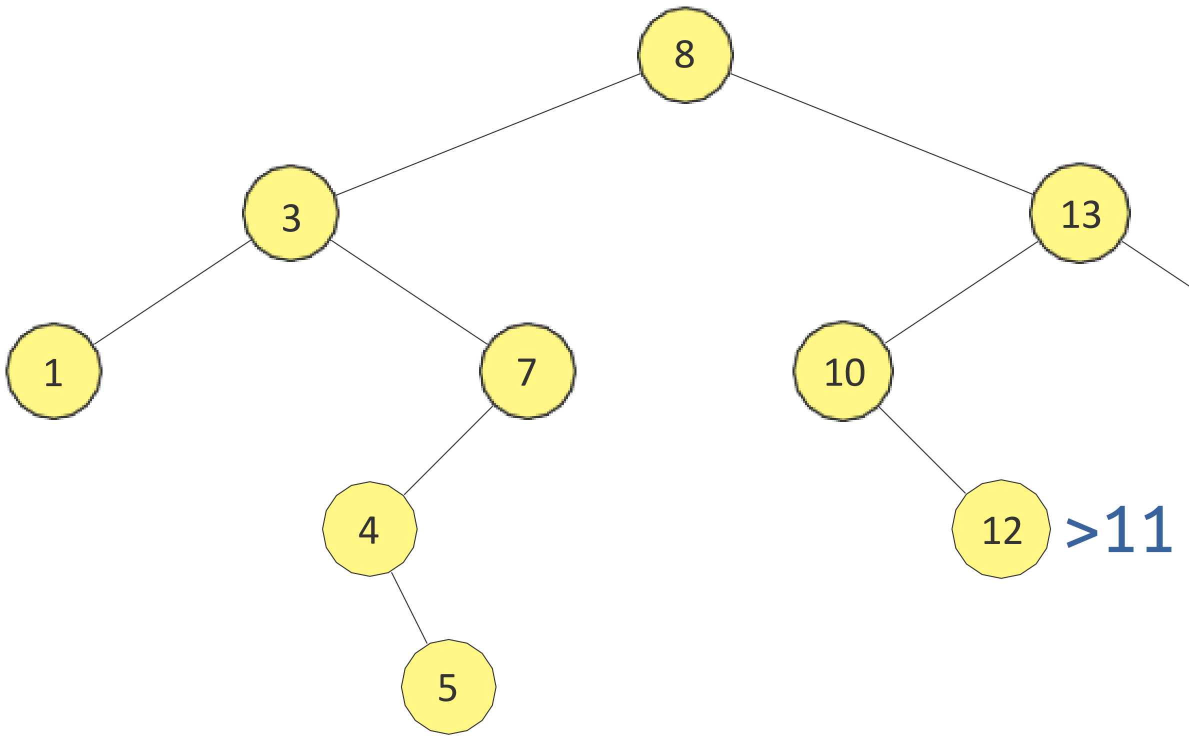
3. Inserção

- Exemplo: inserir o valor 11



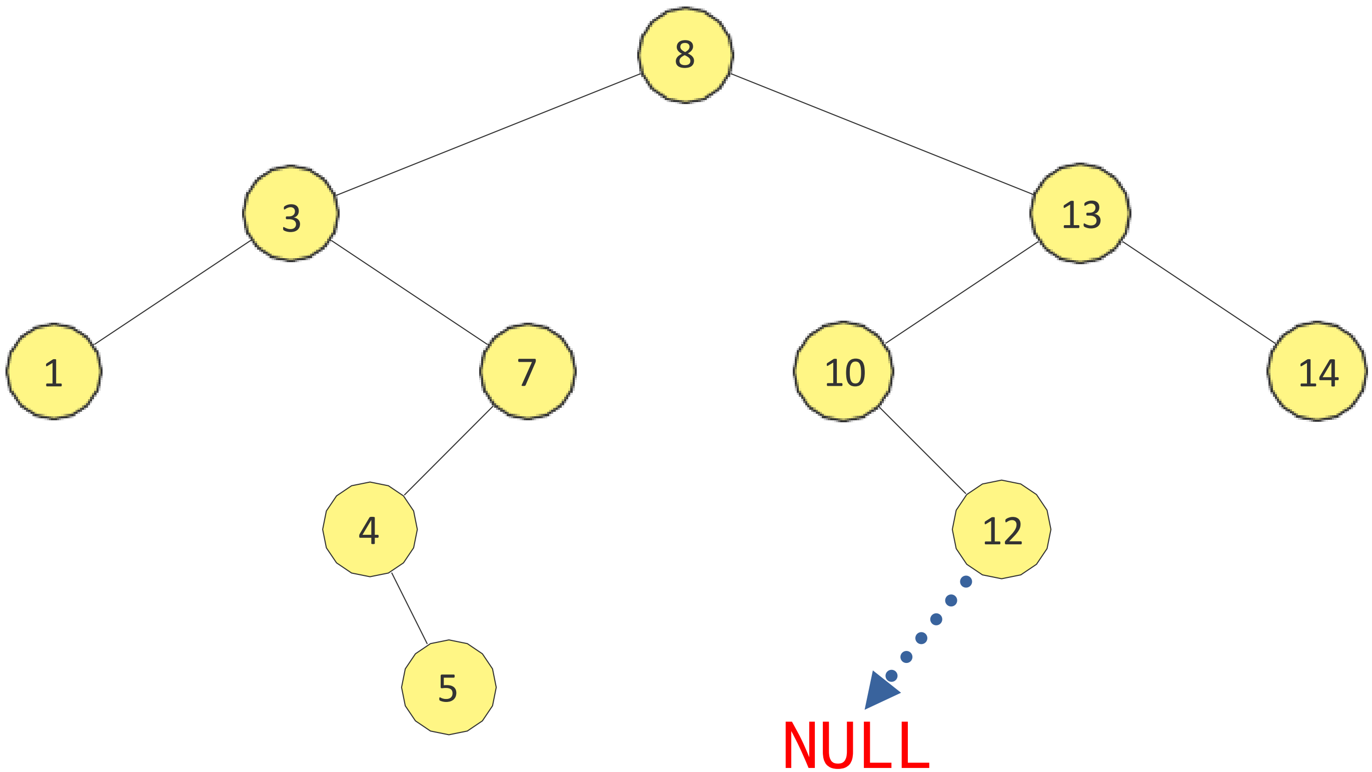
3. Inserção

- Exemplo: inserir o valor 11



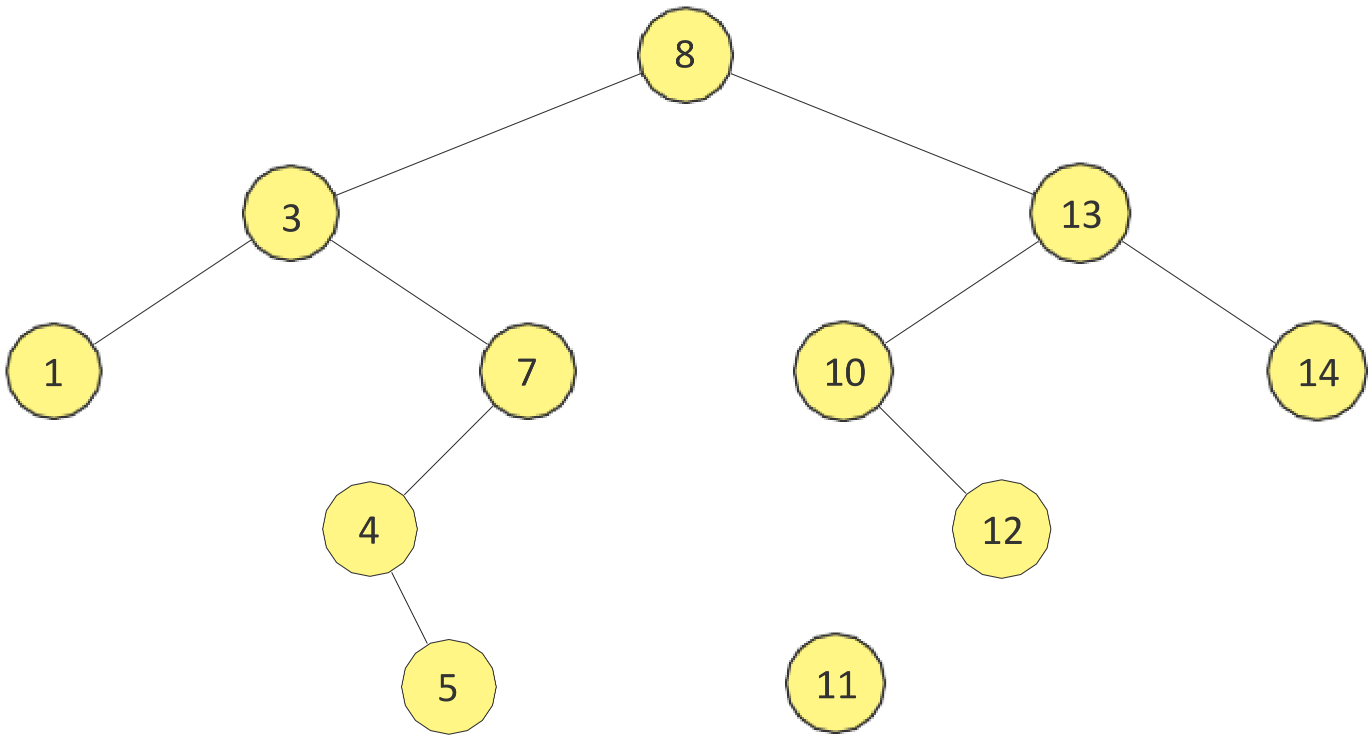
3. Inserção

- Exemplo: inserir o valor 11



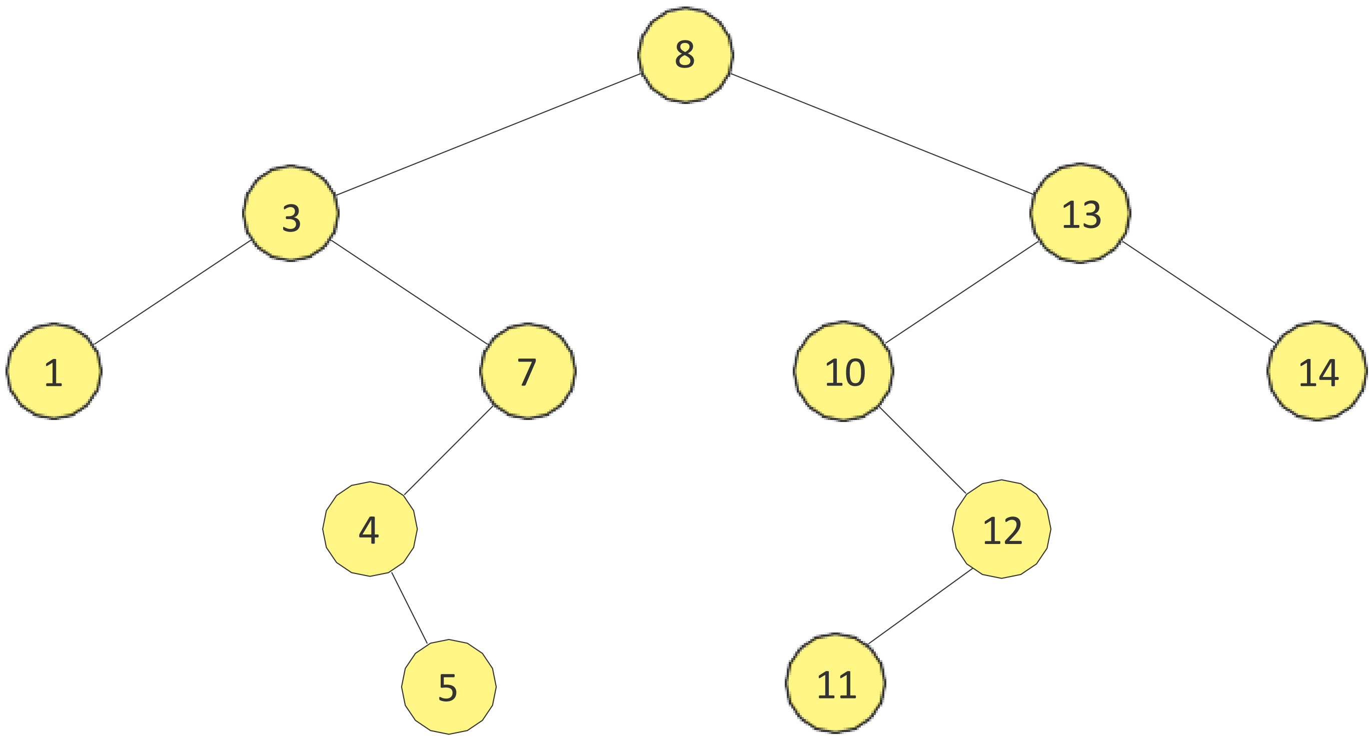
3. Inserção

- Exemplo: inserir o valor 11



3. Inserção

- Exemplo: inserir o valor 11



3.1. Implementação – Recursivo

```
int insere (TNo** ppRaiz, TRegistro x) {

  if (*ppRaiz == NULL) {
    *ppRaiz = cria_no(x);
    return 1; }

  if (x.chave < (*ppRaiz)->reg.chave)
    return insere (&((*ppRaiz)->pEsq), x);

  if (x.chave > (*ppRaiz)->reg.chave)
    return insere (&((*ppRaiz)->pDir), x);

  return 0; /* elemento jah existe */
}
```

3.1. Implementação – Não Recursivo

```
int insere (TNo** ppRaiz, TRegistro x) {
  TNo** ppAux = ppRaiz;

  while (*ppAux != NULL) {
    if (x.chave < (*ppAux)->reg.Chave)
      ppAux = &((*ppAux)->pEsq);
    else if (x.chave > (*ppAux)->reg.chave)
      ppAux = &((*ppAux)->pDir);
    else /* if (x.chave == (*ppAux)->reg.chave) */
      return 0;
  }
  *ppAux = cria_no(x);
  return 1;
}
```

4. Remoção

- Remover um nó de uma árvore binária de busca não é uma tarefa tão simples quanto a inserção.

  - Isso ocorre porque precisamos procurar o nó a ser removido da árvore o qual pode ser um:
    - Nó folha.
    - Nó interno (que pode ser a raiz), com um ou dois filhos.

4. Remoção

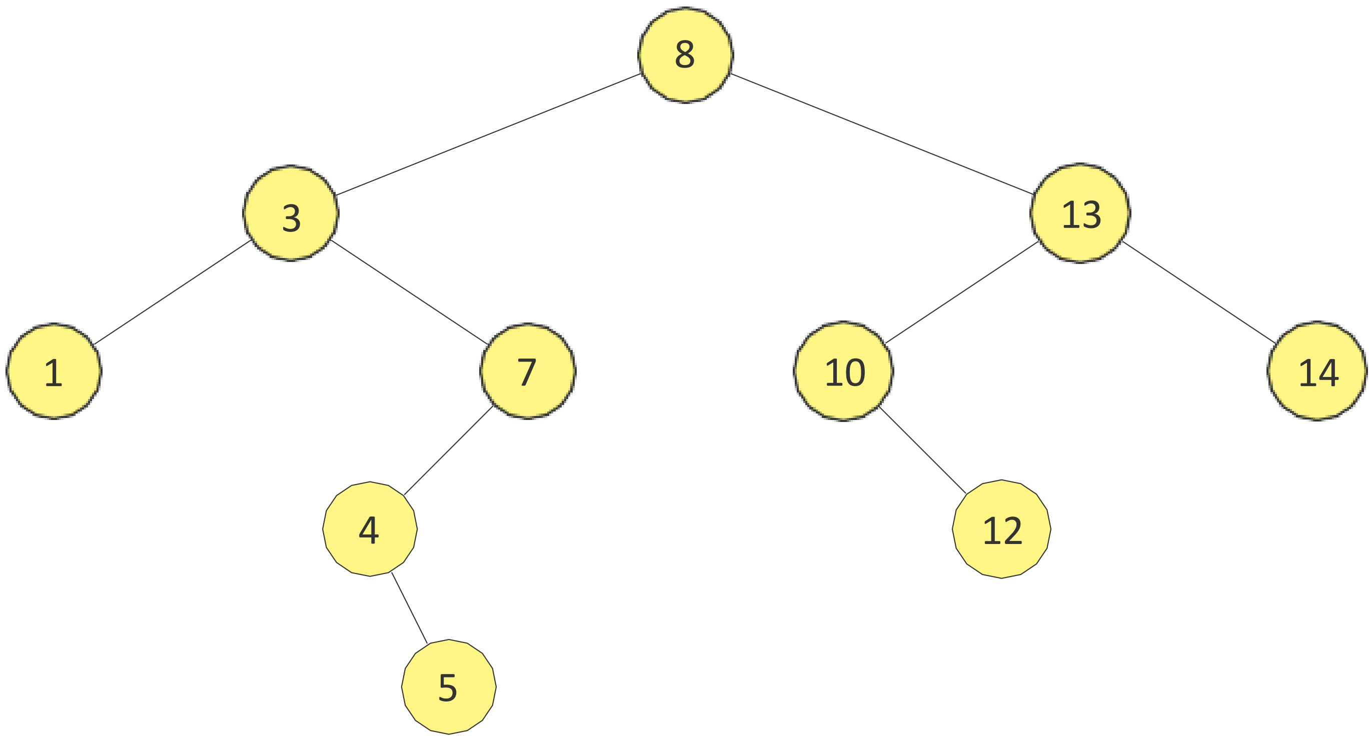
1. Se for um nó folha:
   - Remoção simples: apenas excluir o nó.

2. Se for um nó interno com 1 filho:
   - Remoção também é simples:
     - Excluir o nó.
     - Filho ocupa seu lugar na árvore.

3. Se for um nó interno com 2 filhos:
     - Reorganizar a árvore para que ela continue sendo uma árvore binária de busca.

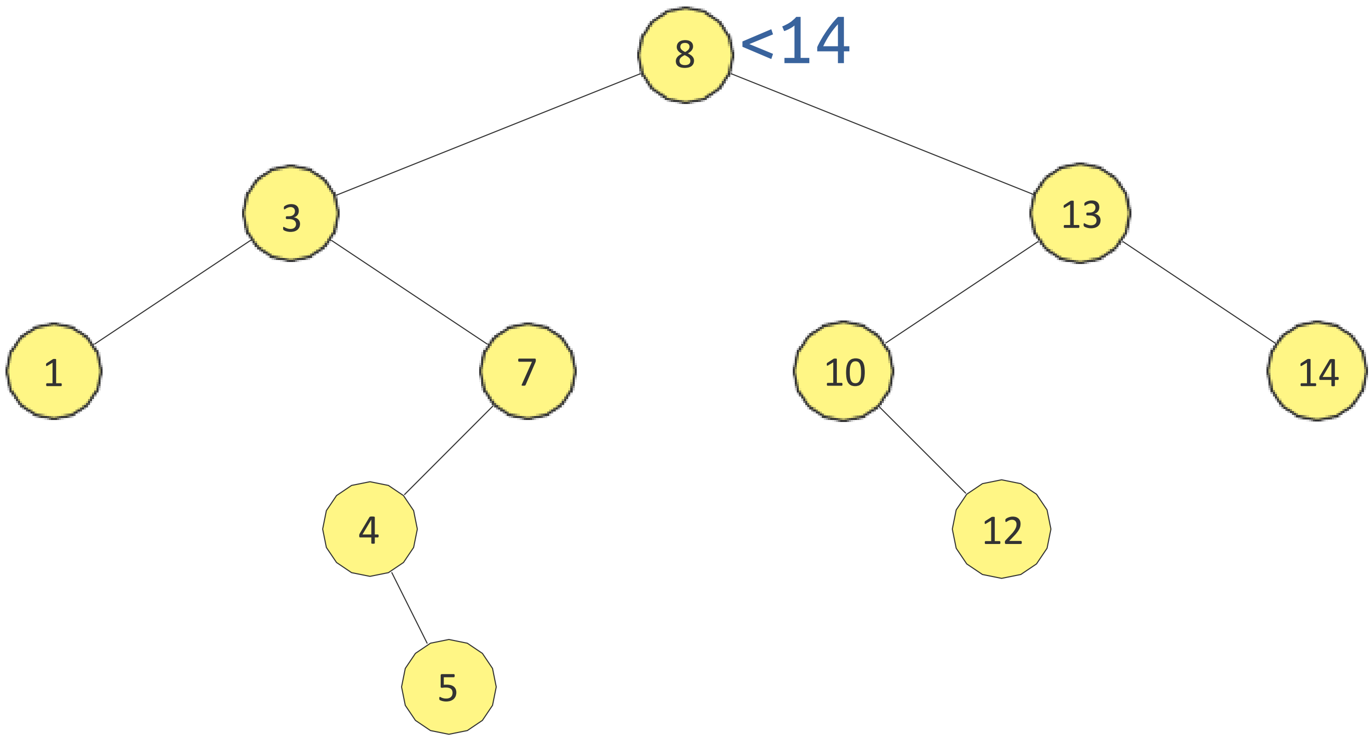
4. Remoção

1. Exemplo: remover o valor **14** (folha)



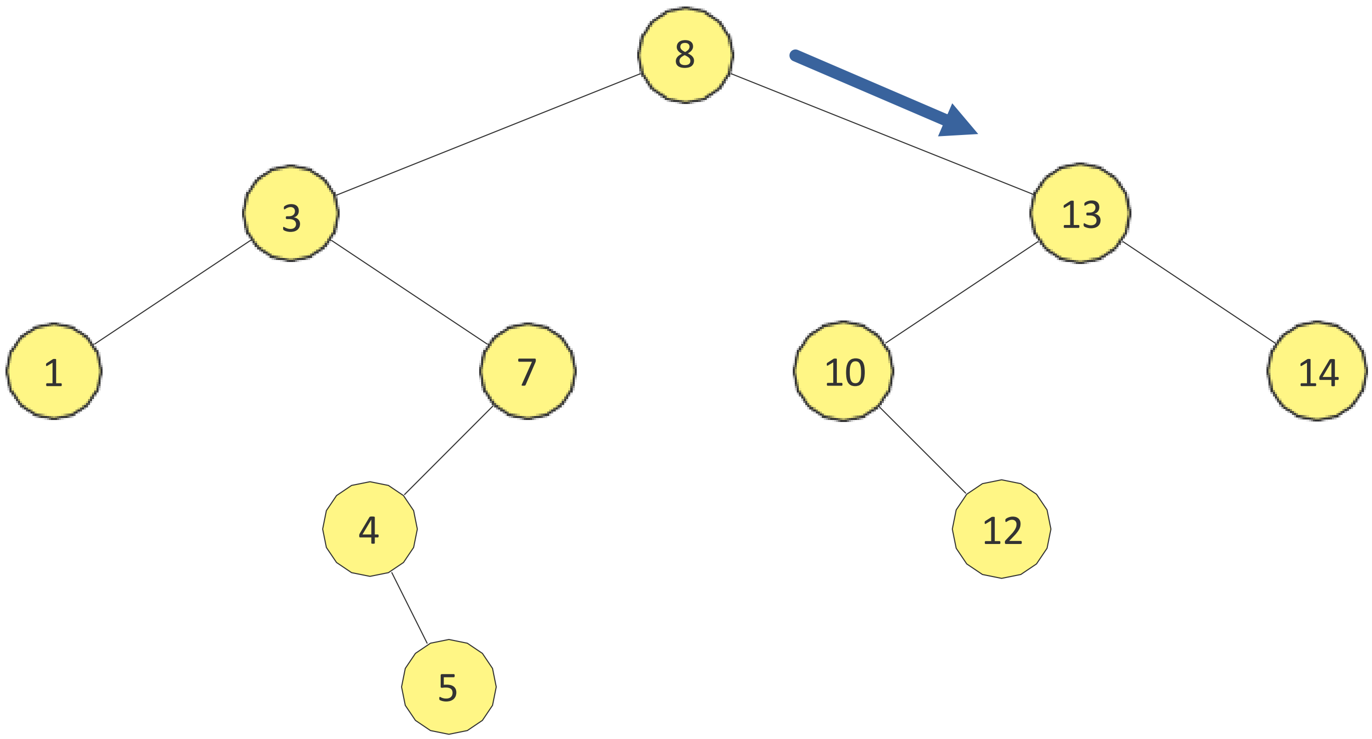
4. Remoção

1. Exemplo: remover o valor **14** (folha)



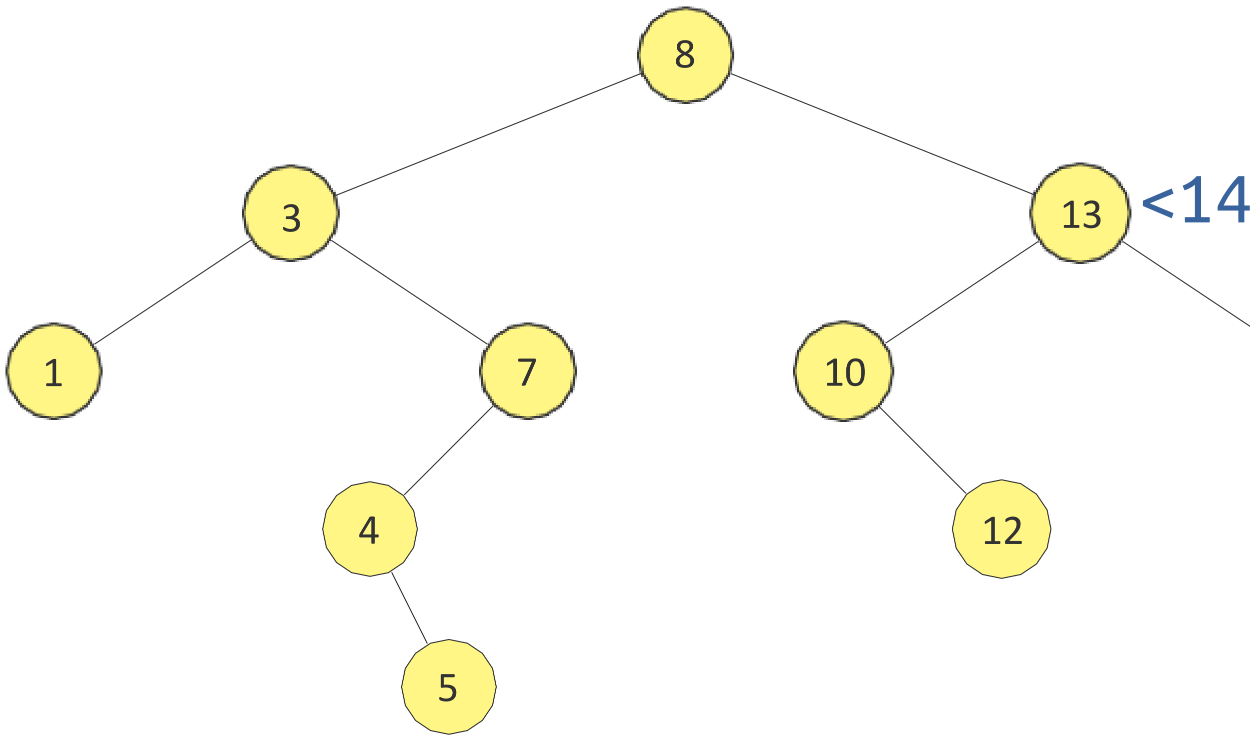
4. Remoção

1. Exemplo: remover o valor **14** (folha)



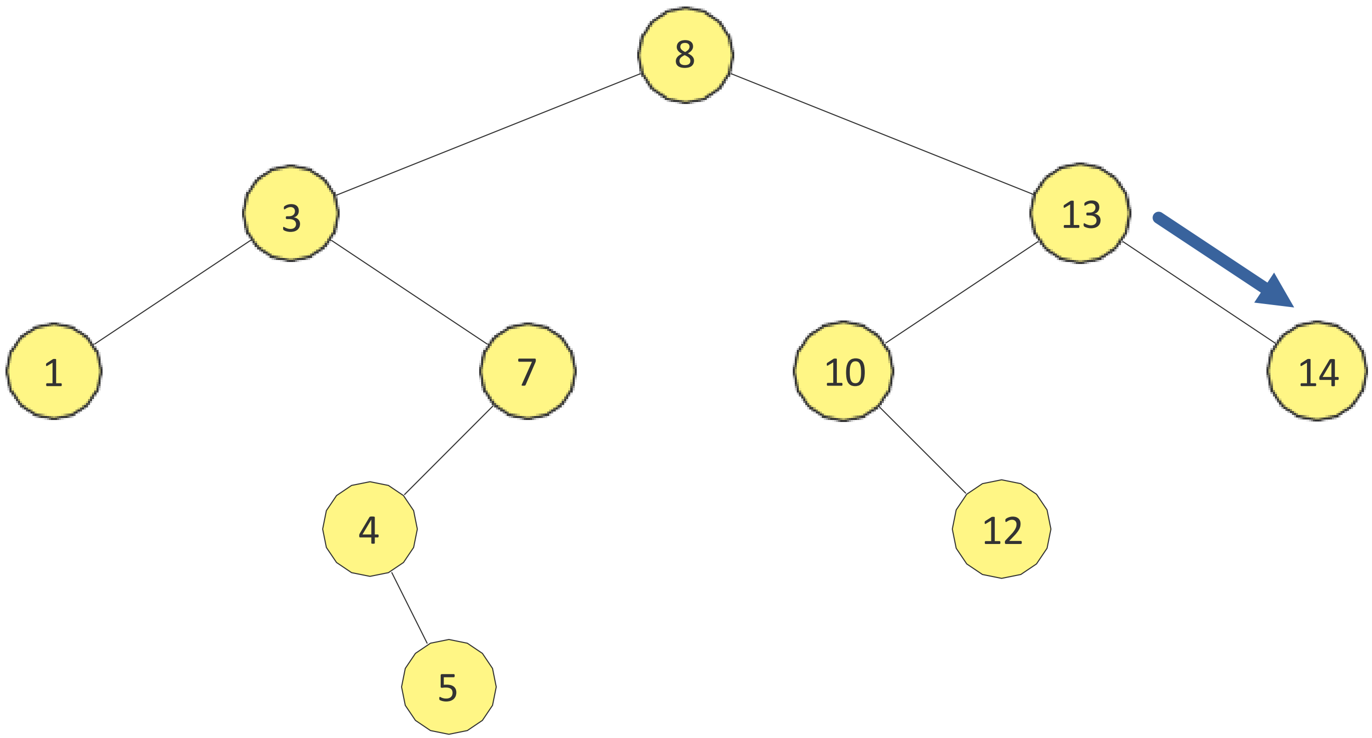
4. Remoção

1. Exemplo: remover o valor **14** (folha)



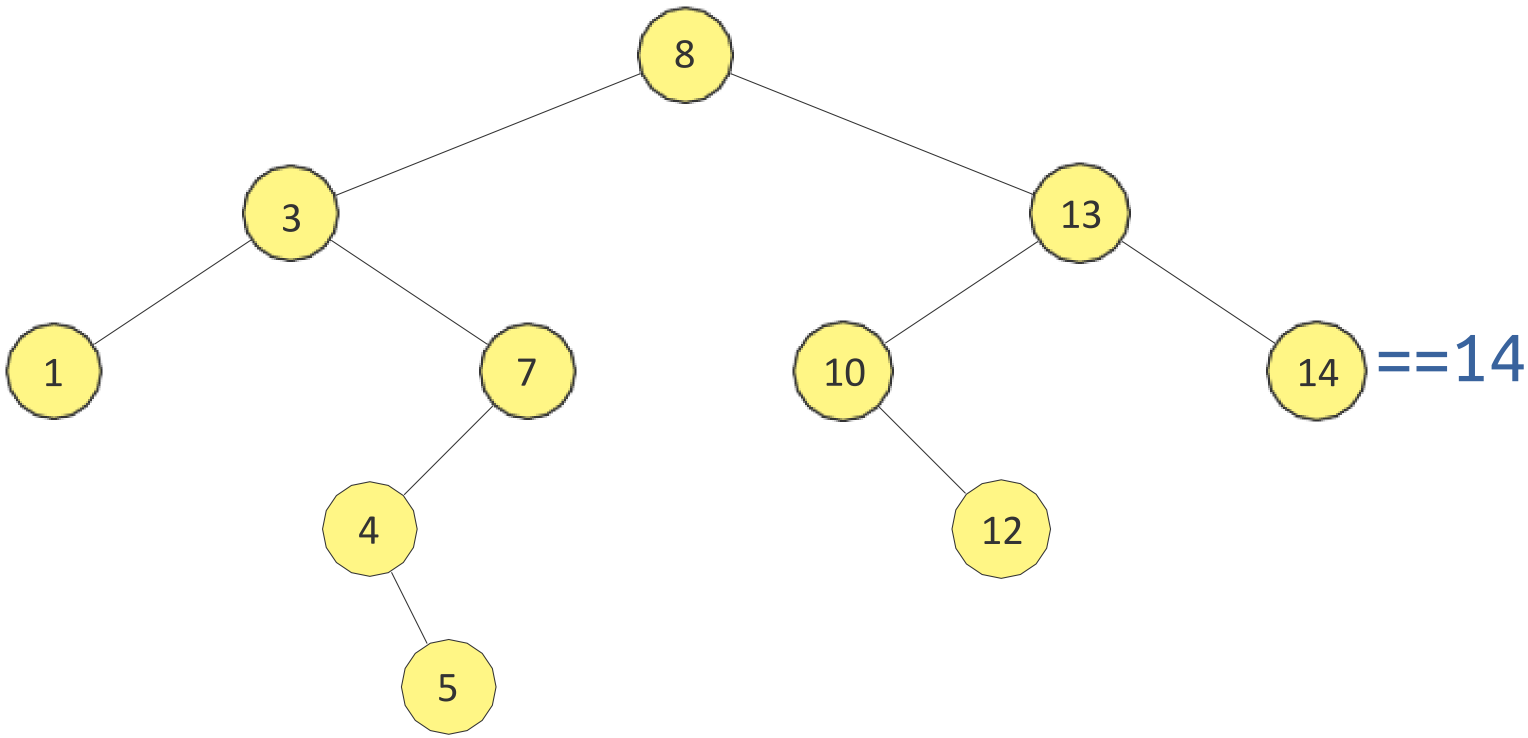
4. Remoção

1. Exemplo: remover o valor 14 (folha)



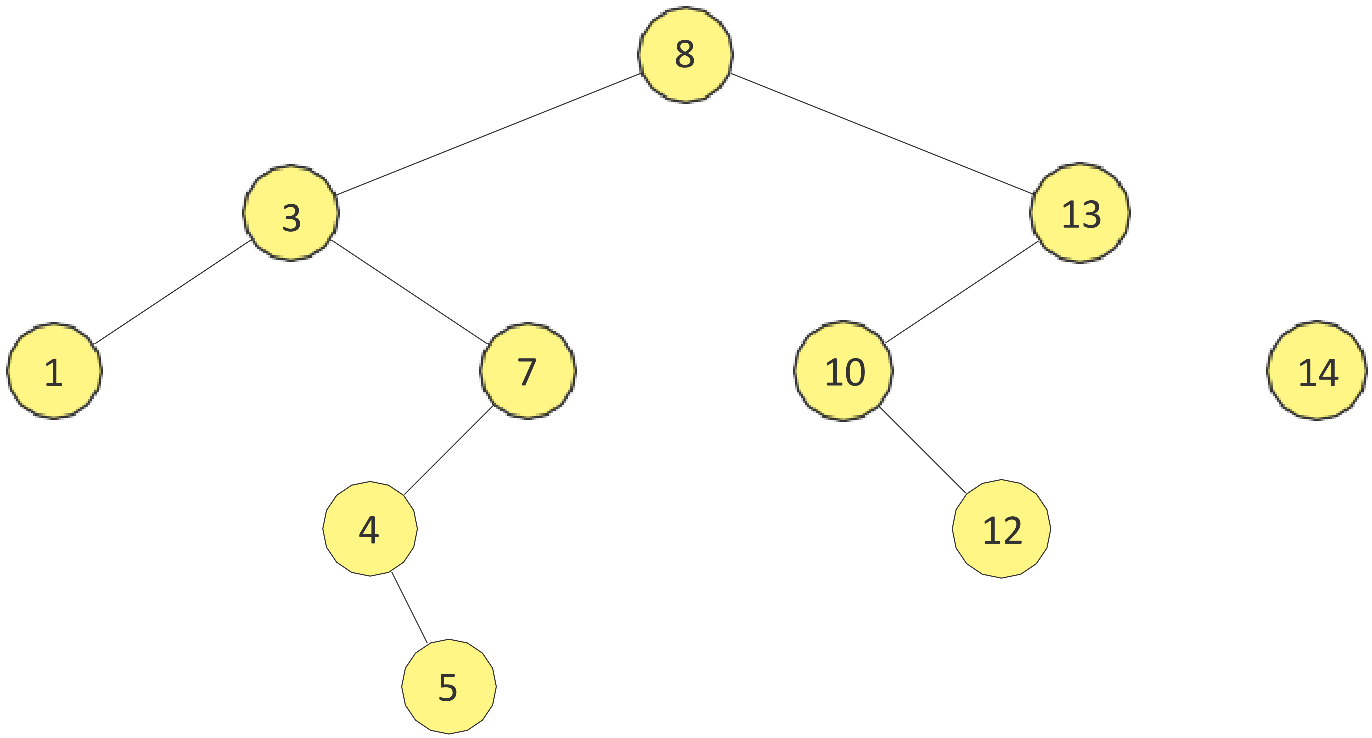
4. Remoção

1. Exemplo: remover o valor 14 (folha)



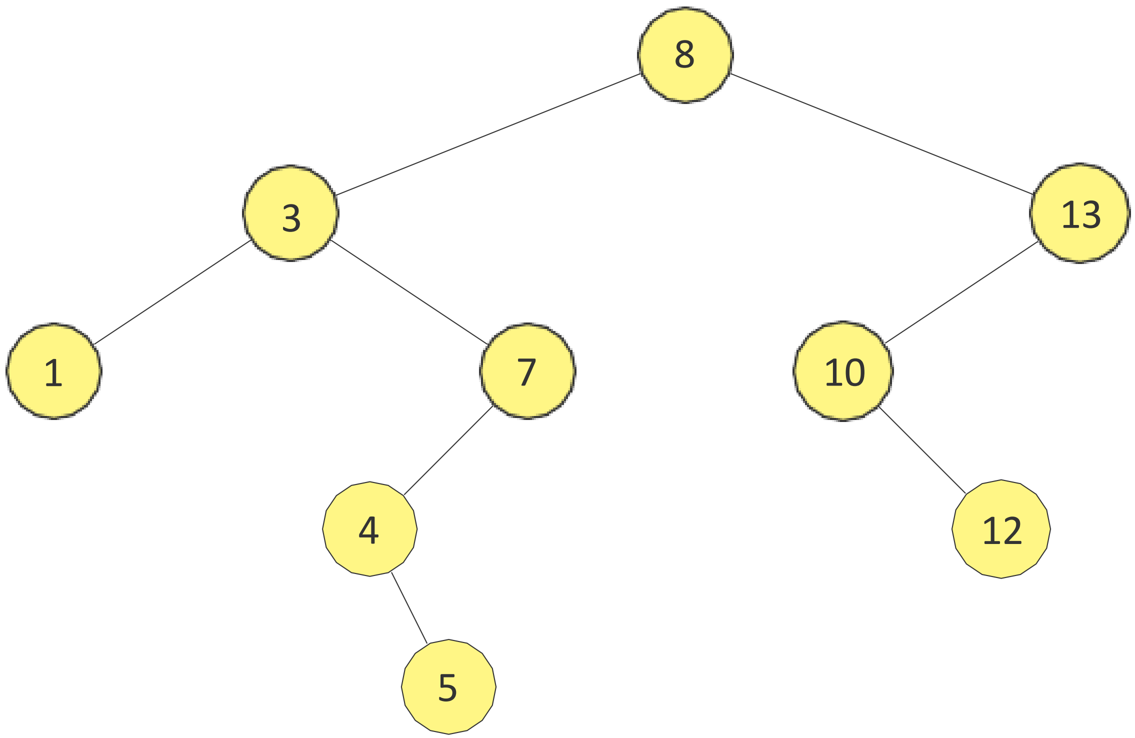
4. Remoção

1. Exemplo: remover o valor **14** (folha)



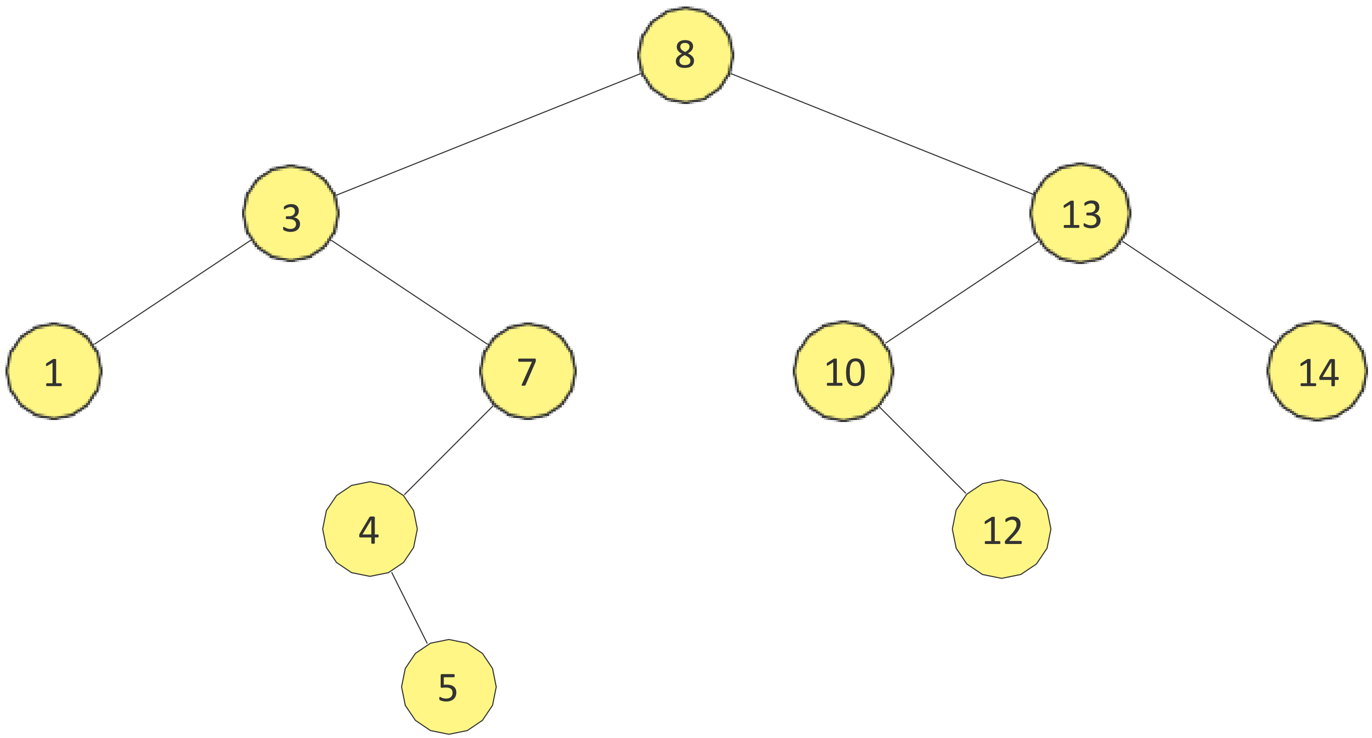
4. Remoção

1. Exemplo: remover o valor **14** (folha)



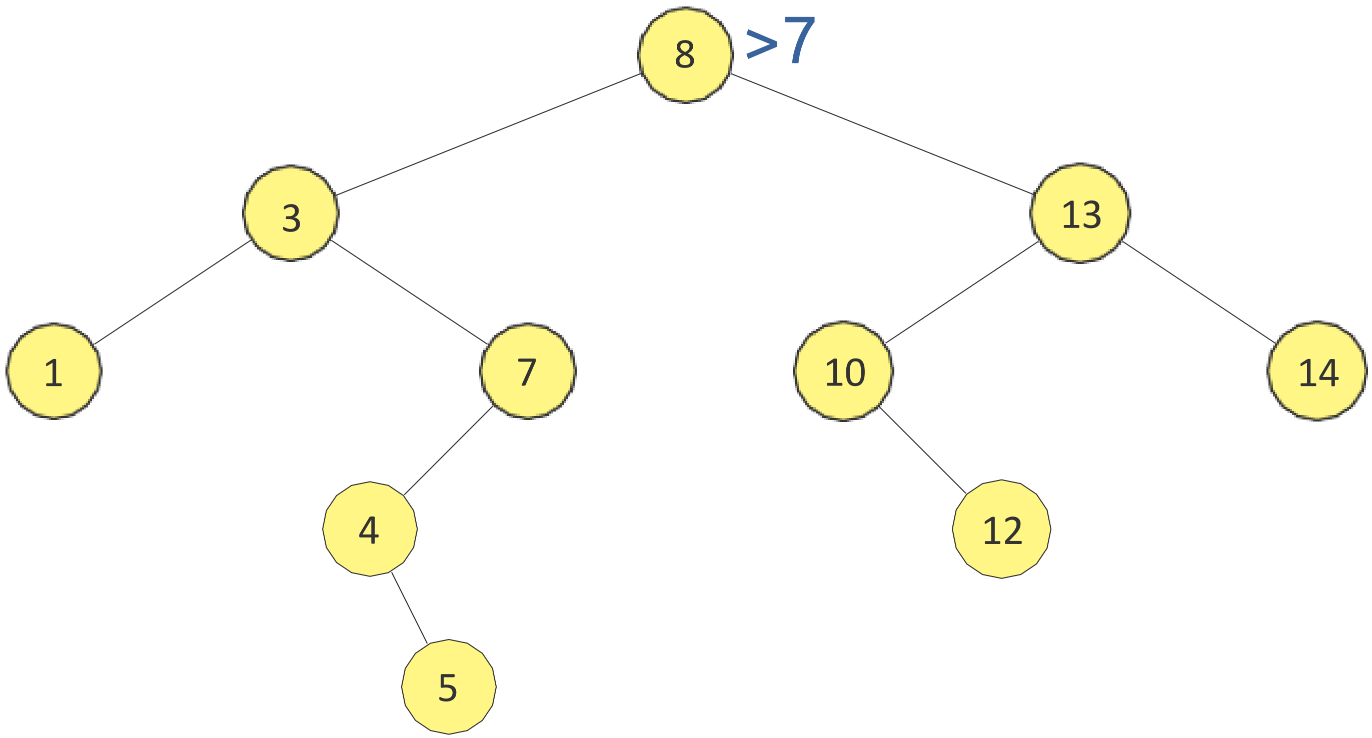
4. Remoção

2. Exemplo: remover o valor **7** (1 filho)



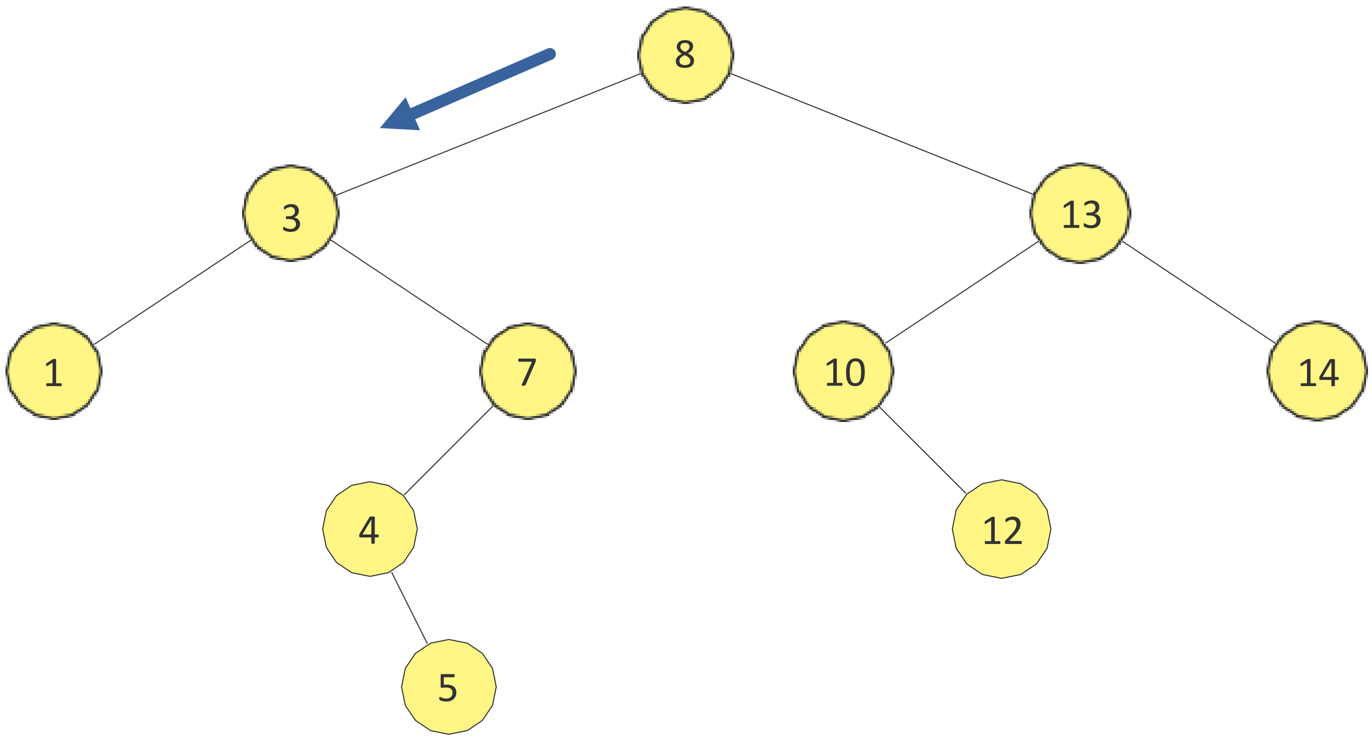
4. Remoção

2. Exemplo: remover o valor **7** (1 filho)



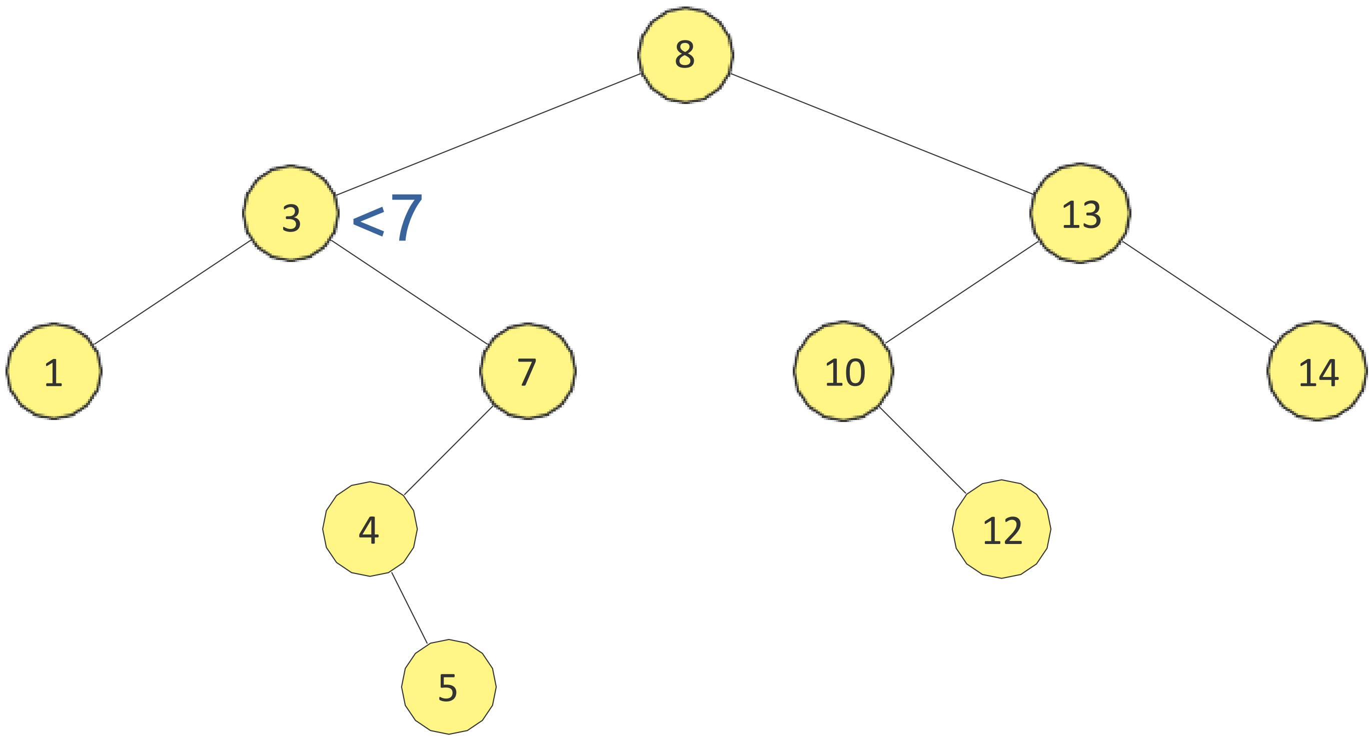
4. Remoção

2. Exemplo: remover o valor **7** (1 filho)



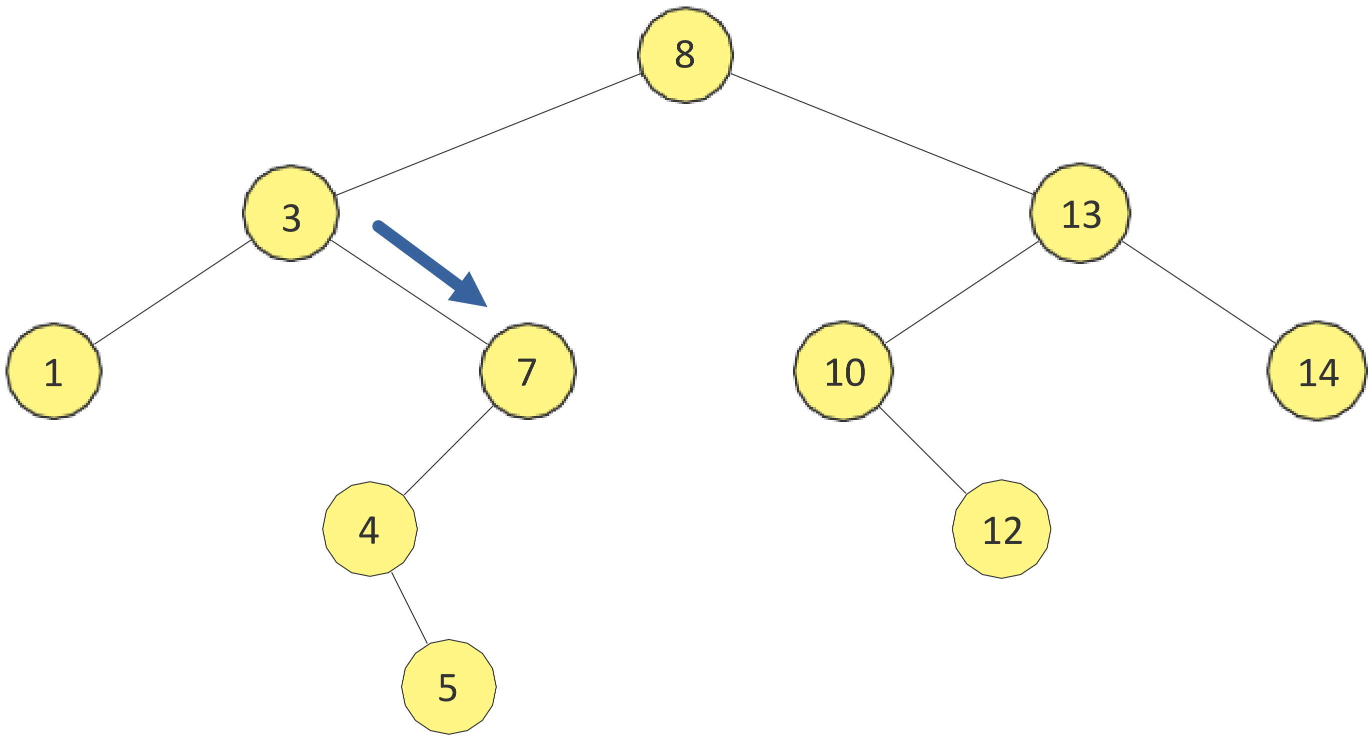
4. Remoção

2. Exemplo: remover o valor **7** (1 filho)



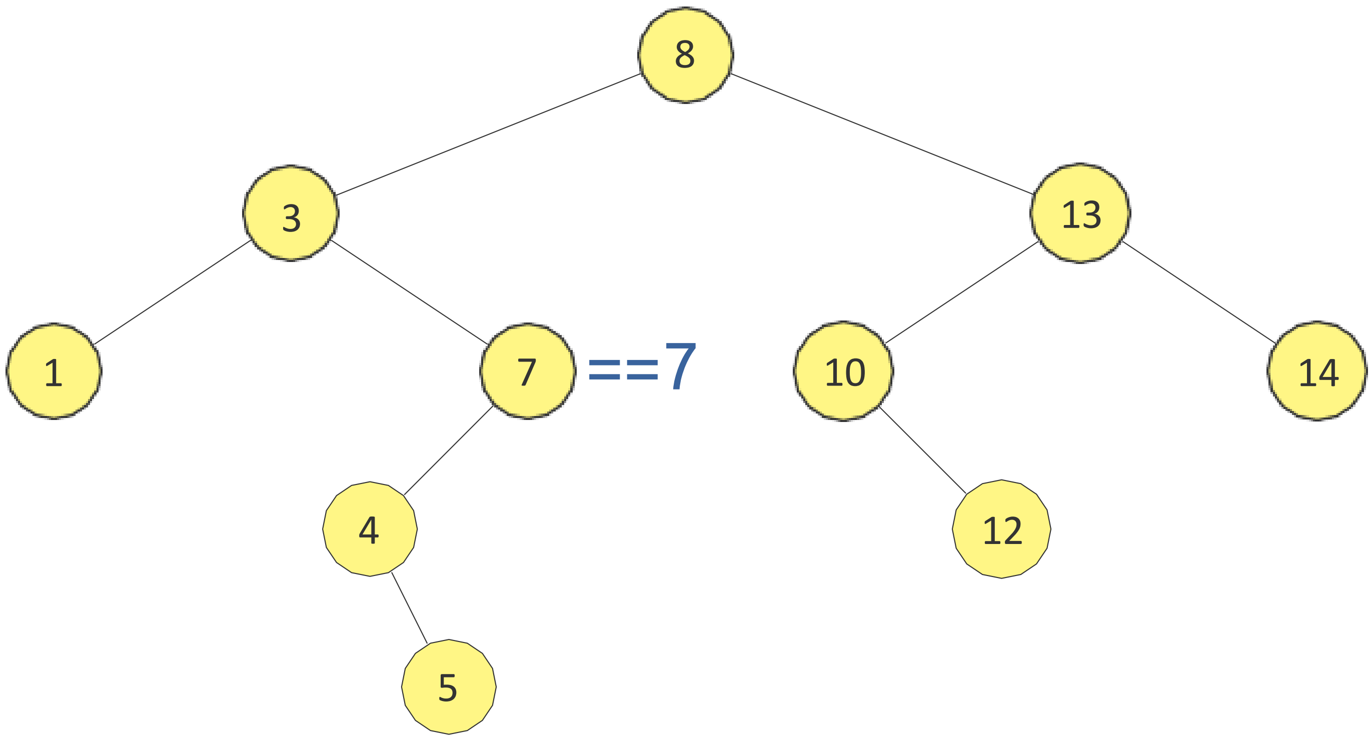
4. Remoção

2. Exemplo: remover o valor **7** (1 filho)



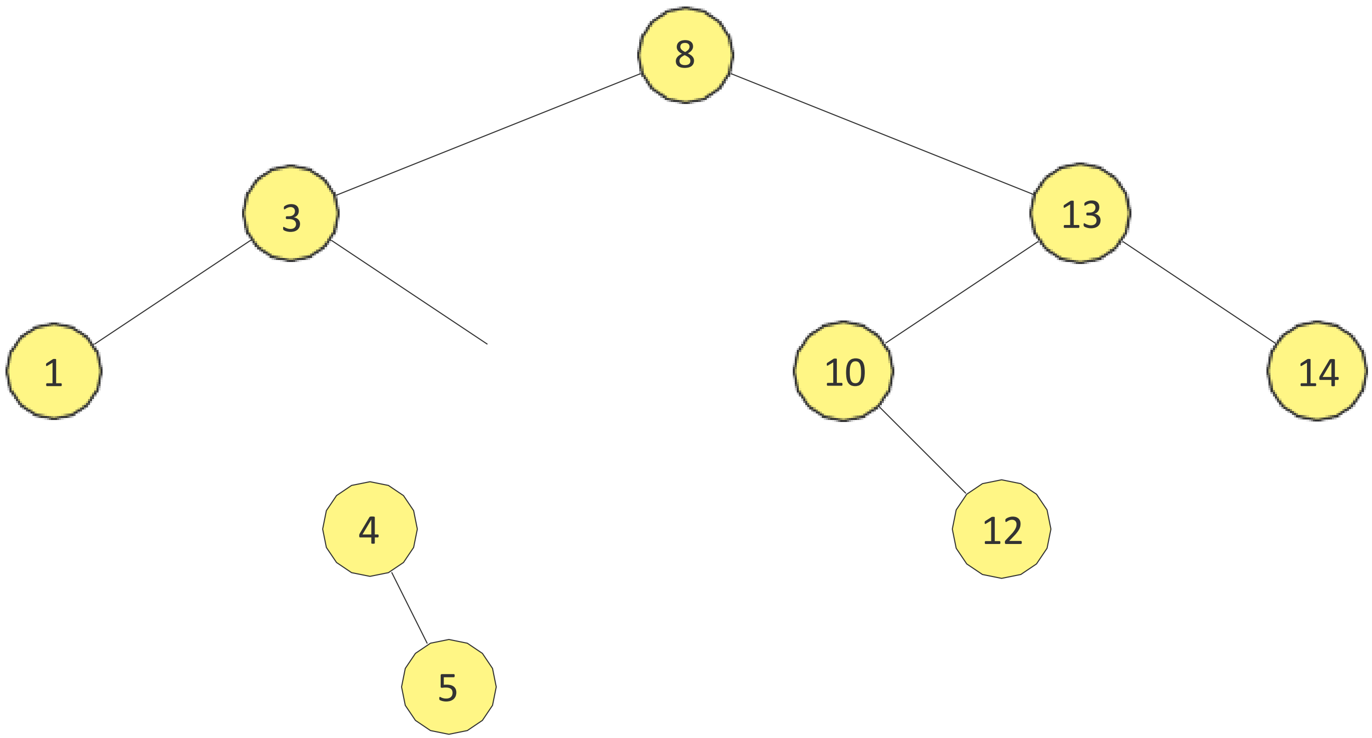
4. Remoção

2. Exemplo: remover o valor **7** (1 filho)



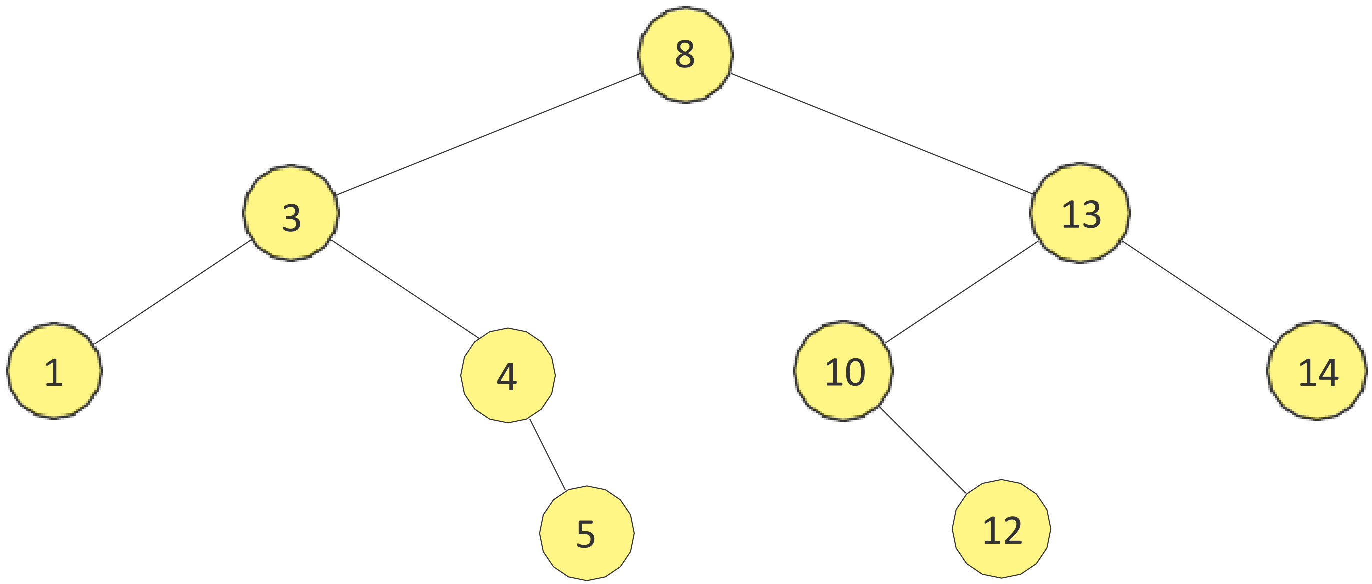
4. Remoção

2. Exemplo: remover o valor **7** (1 filho)



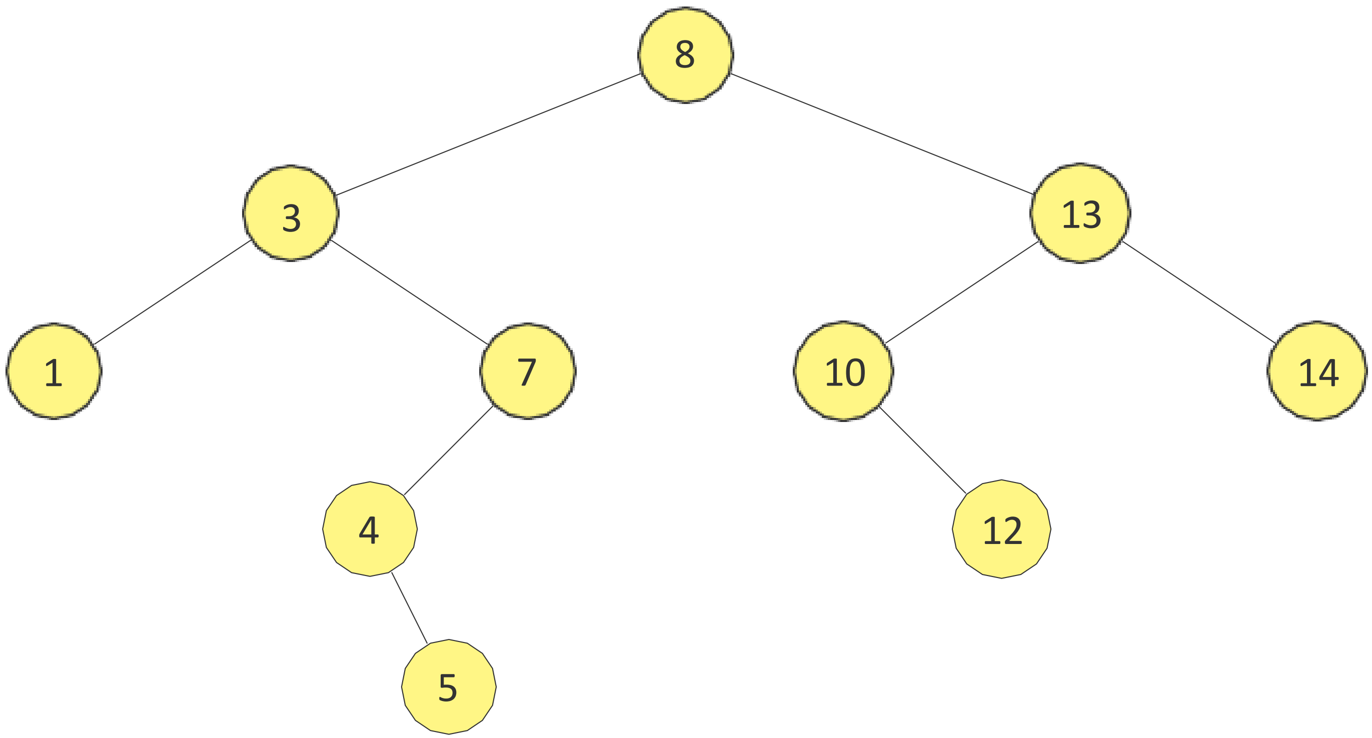
4. Remoção

2. Exemplo: remover o valor **7** (1 filho)



4. Remoção

3. Exemplo: remover o valor 3 (2 filhos)



4. Remoção

3. Exemplo: remover o valor 3 (2 filhos)

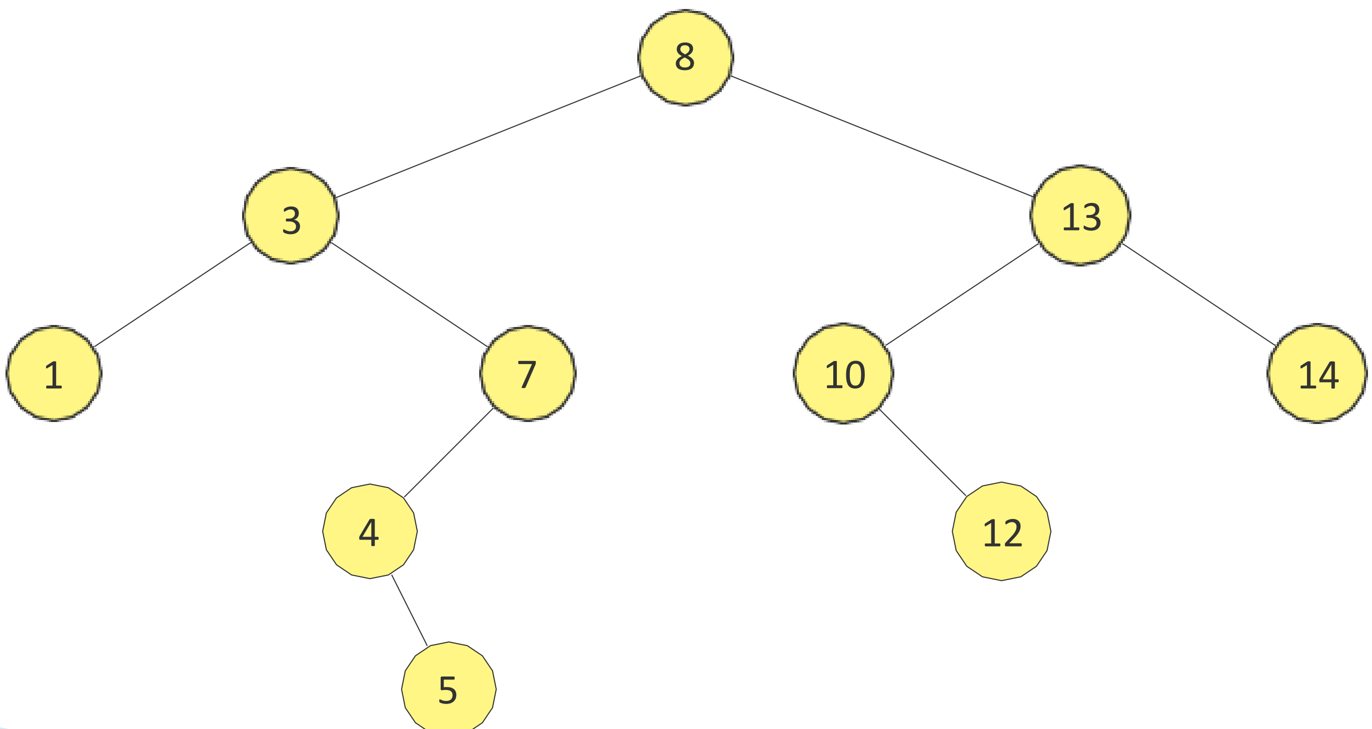
   - No caso do nó conter dois descendentes o registro a ser retirado deve ser primeiro substituído:

     - Pelo seu **antecessor**
       - Registro mais à **direita** na sub-árvore **esquerda.**

     - Ou pelo **sucessor**
       - Registro mais à **esquerda** na sub-árvore **direita.**

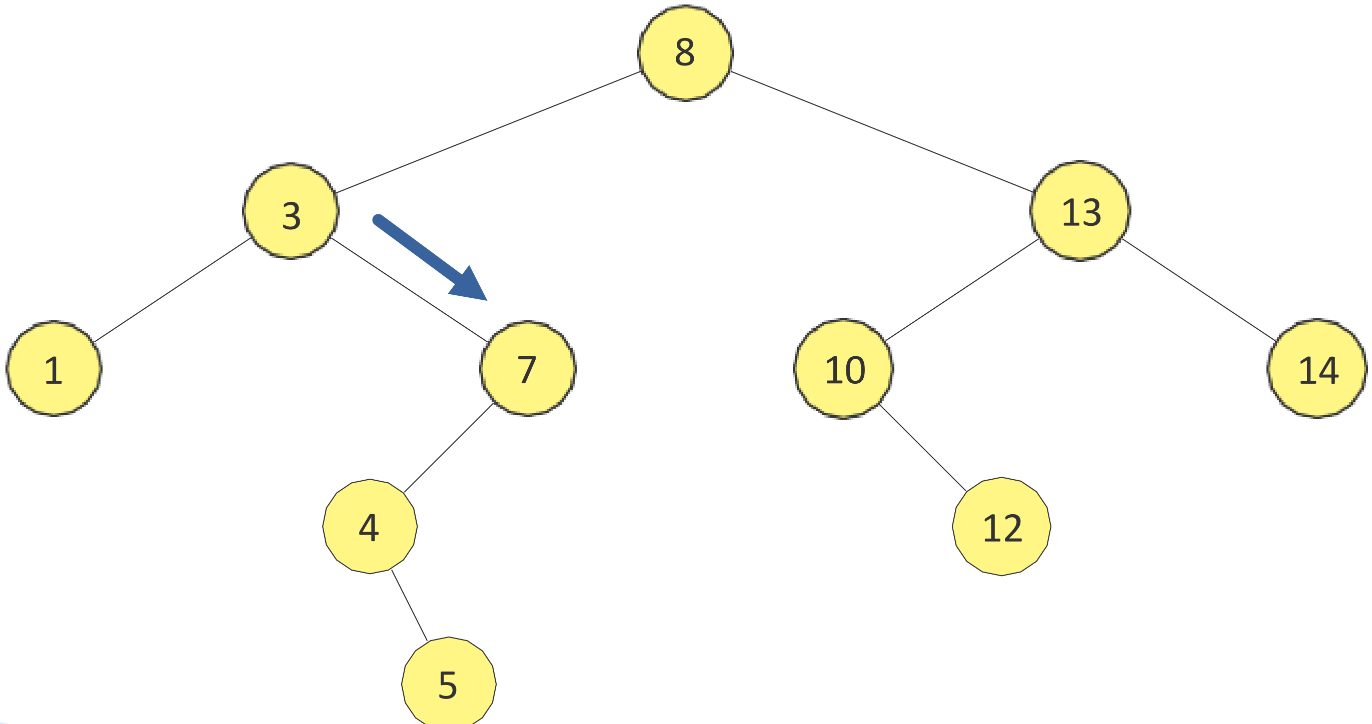
4.1. Sucessor

- Qual o sucessor de 3?
  - É o **mínimo** da sua sub-árvore **direita** de 3, ou seja, 4!
  - O sucessor **nunca** tem filho esquerdo!



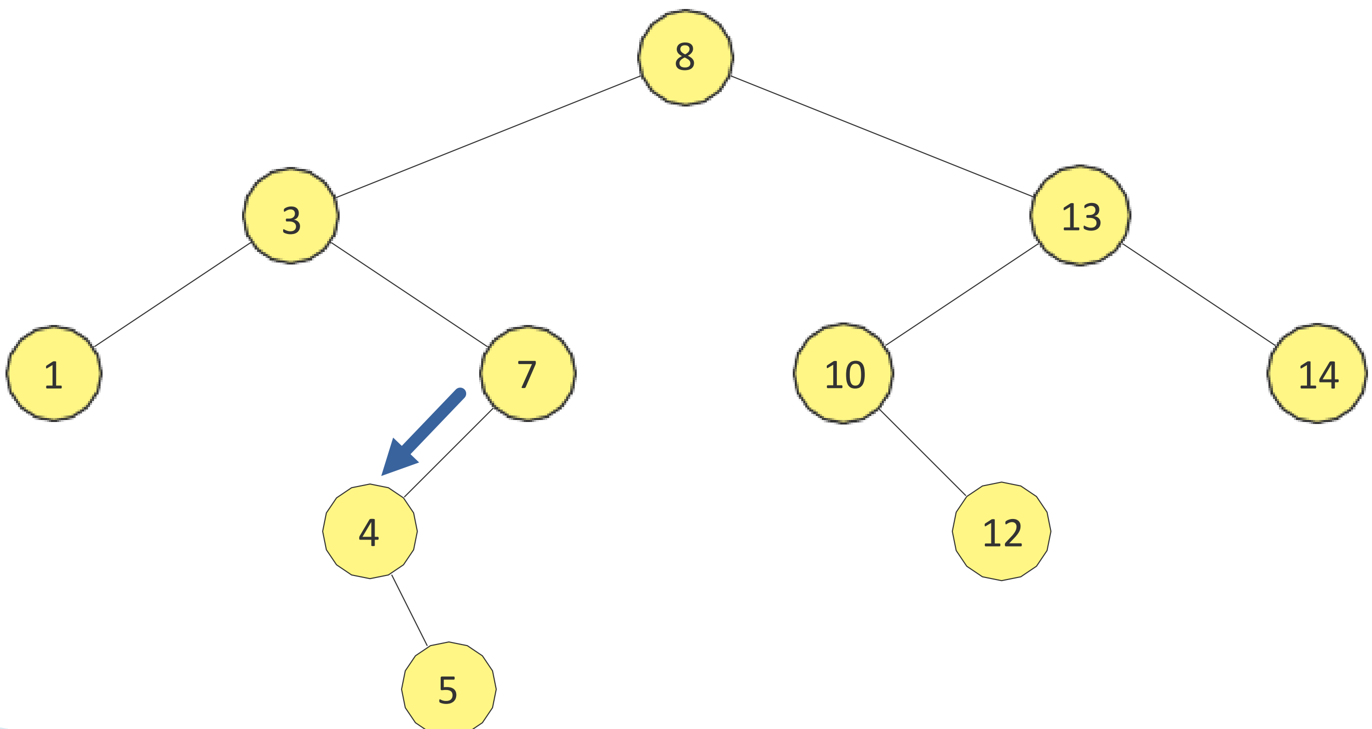
4.1. Sucessor

- Qual o sucessor de 3?
  - É o **mínimo** da sua sub-árvore **direita** de 3, ou seja, 4!
  - O sucessor **nunca** tem filho esquerdo!



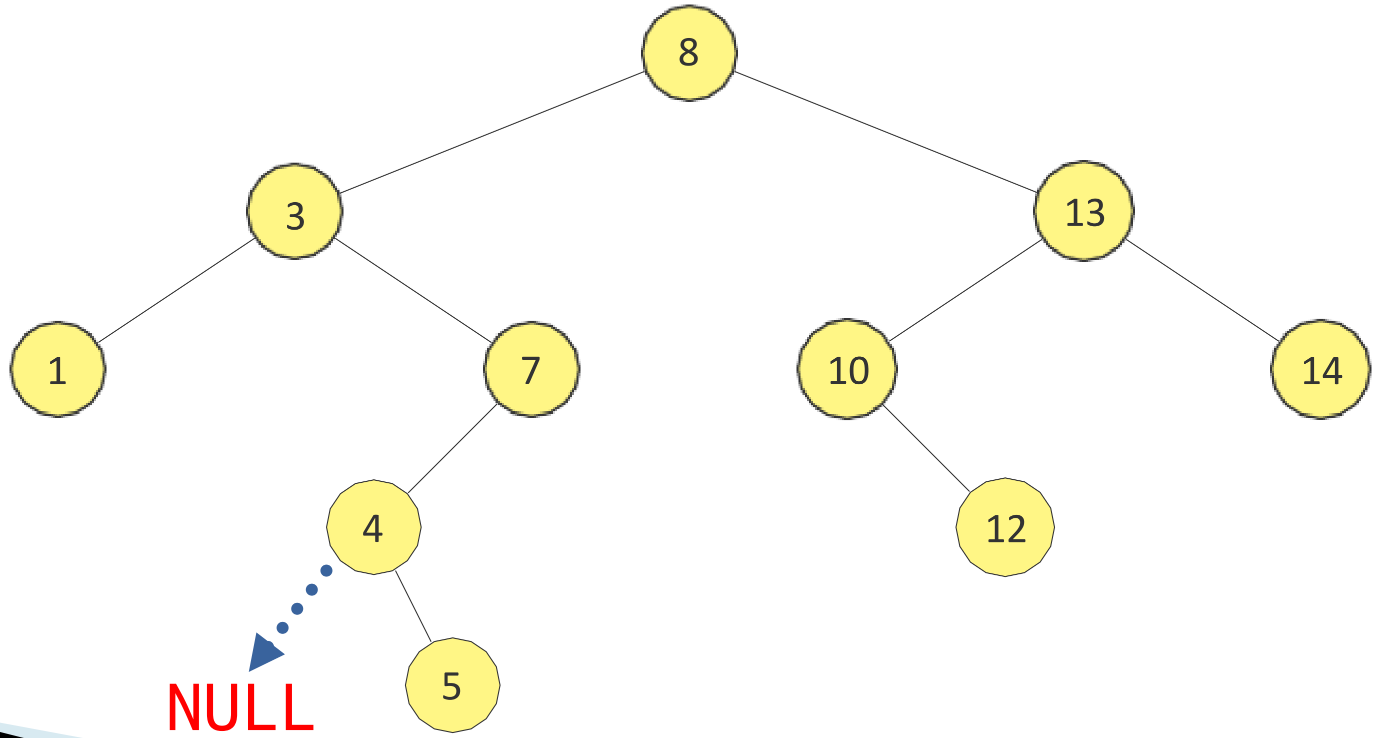
4.1. Sucessor

- Qual o sucessor de 3?
  - É o **mínimo** da sua sub-árvore **direita** de 3, ou seja, 4!
  - O sucessor **nunca** tem filho esquerdo!



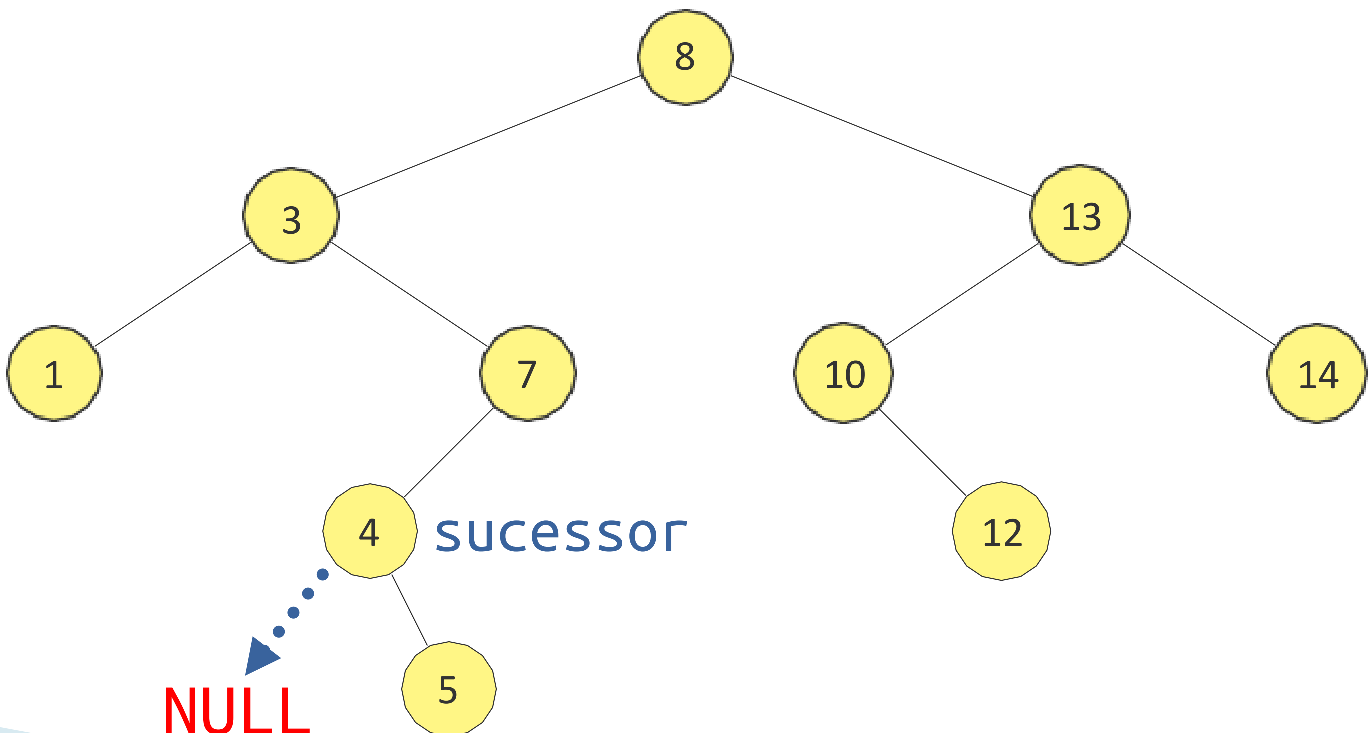
4.1. Sucessor

- Qual o sucessor de 3?
  - É o **mínimo** da sua sub-árvore **direita** de 3, ou seja, 4!
  - O sucessor **nunca** tem filho esquerdo!



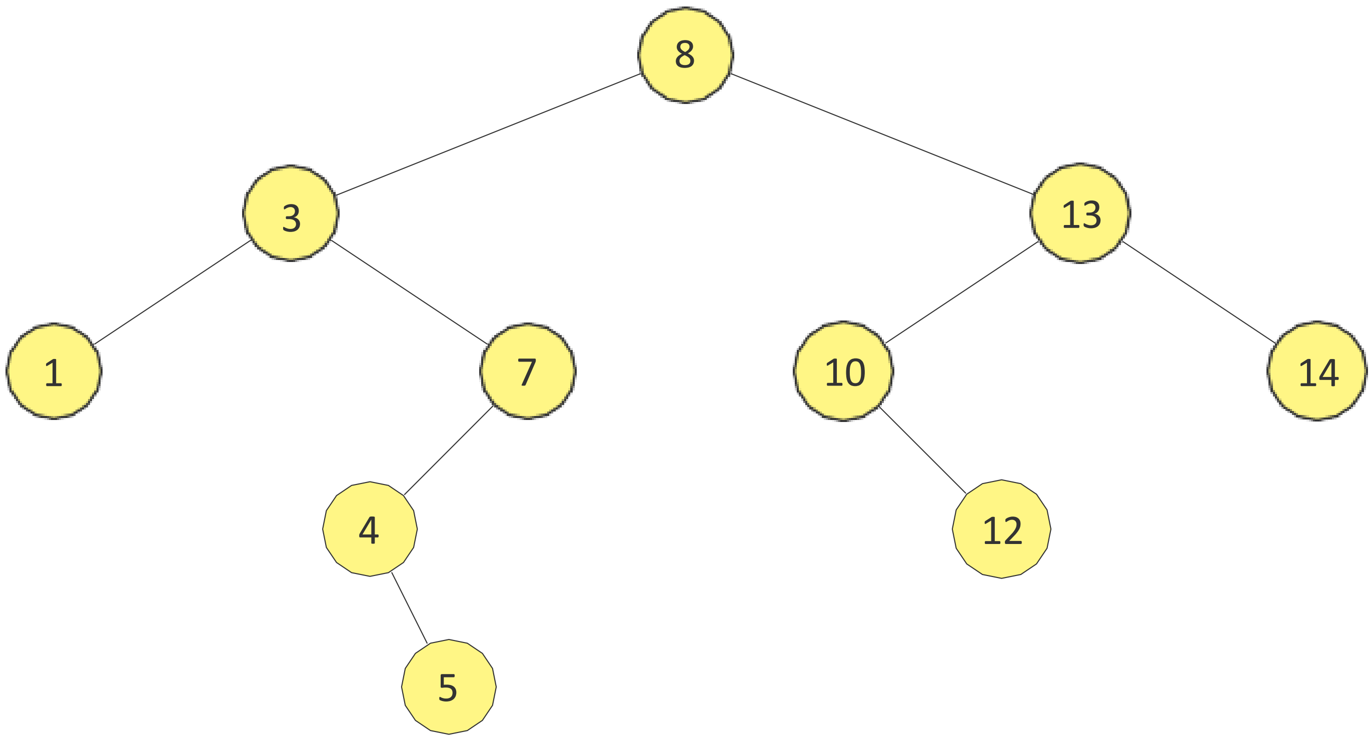
4.1. Sucessor

- Qual o sucessor de 3?
  - É o **mínimo** da sua sub-árvore **direita** de 3, ou seja, 4!
  - O sucessor **nunca** tem filho esquerdo!



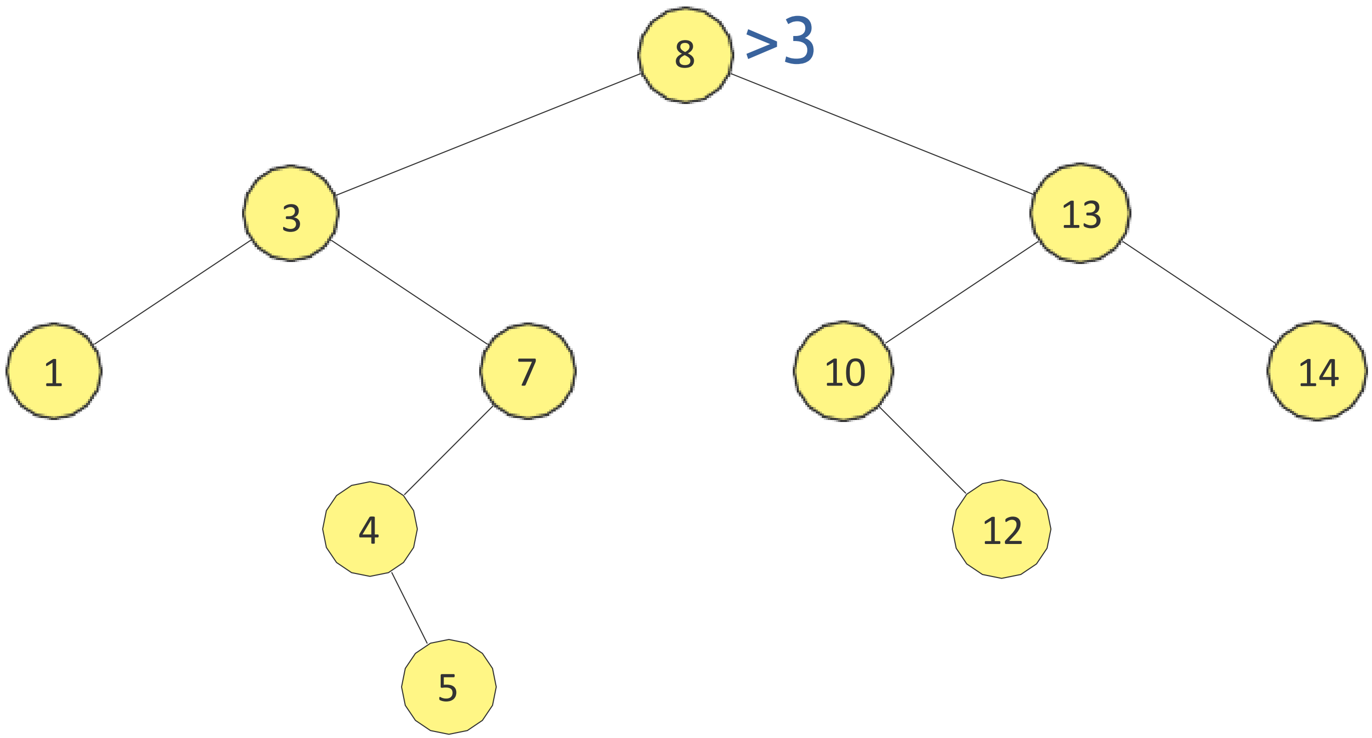
4. Remoção

3. Exemplo: remover o valor **3** (2 filhos)



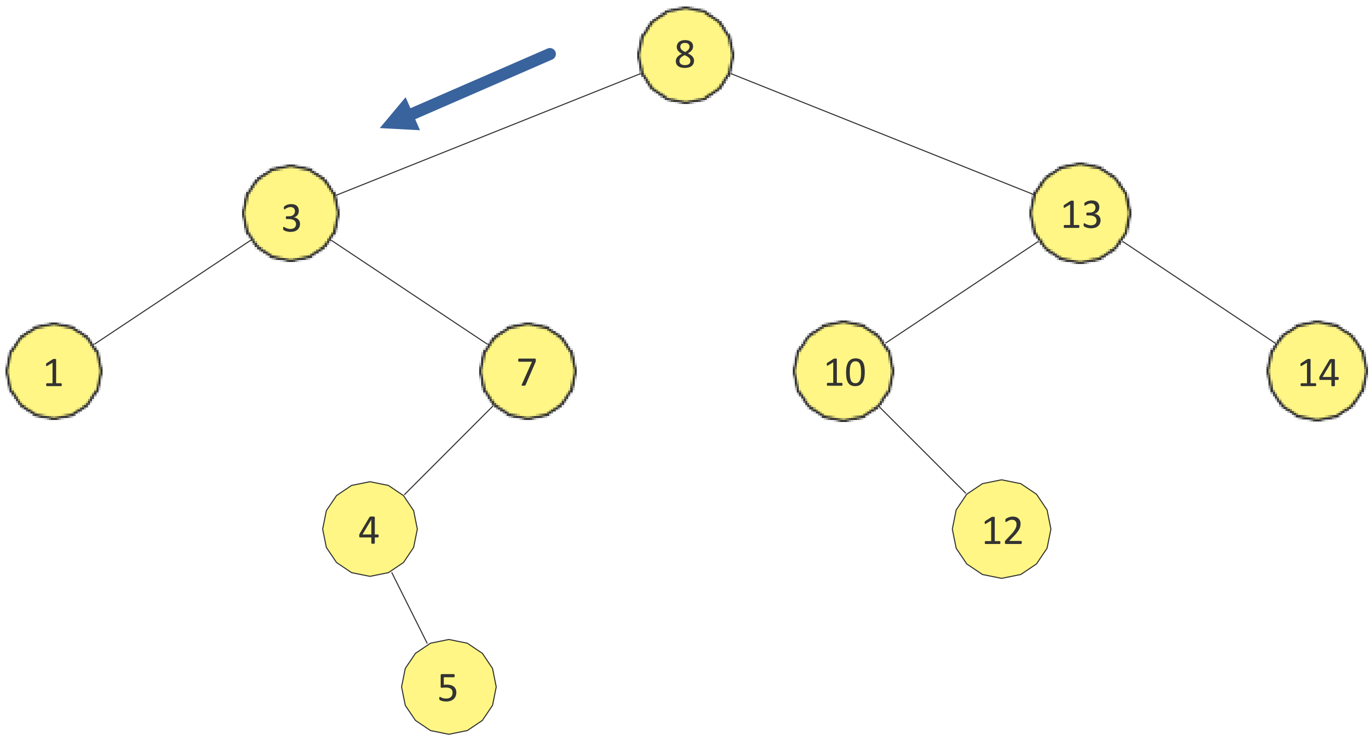
4. Remoção

3. Exemplo: remover o valor 3 (2 filhos)



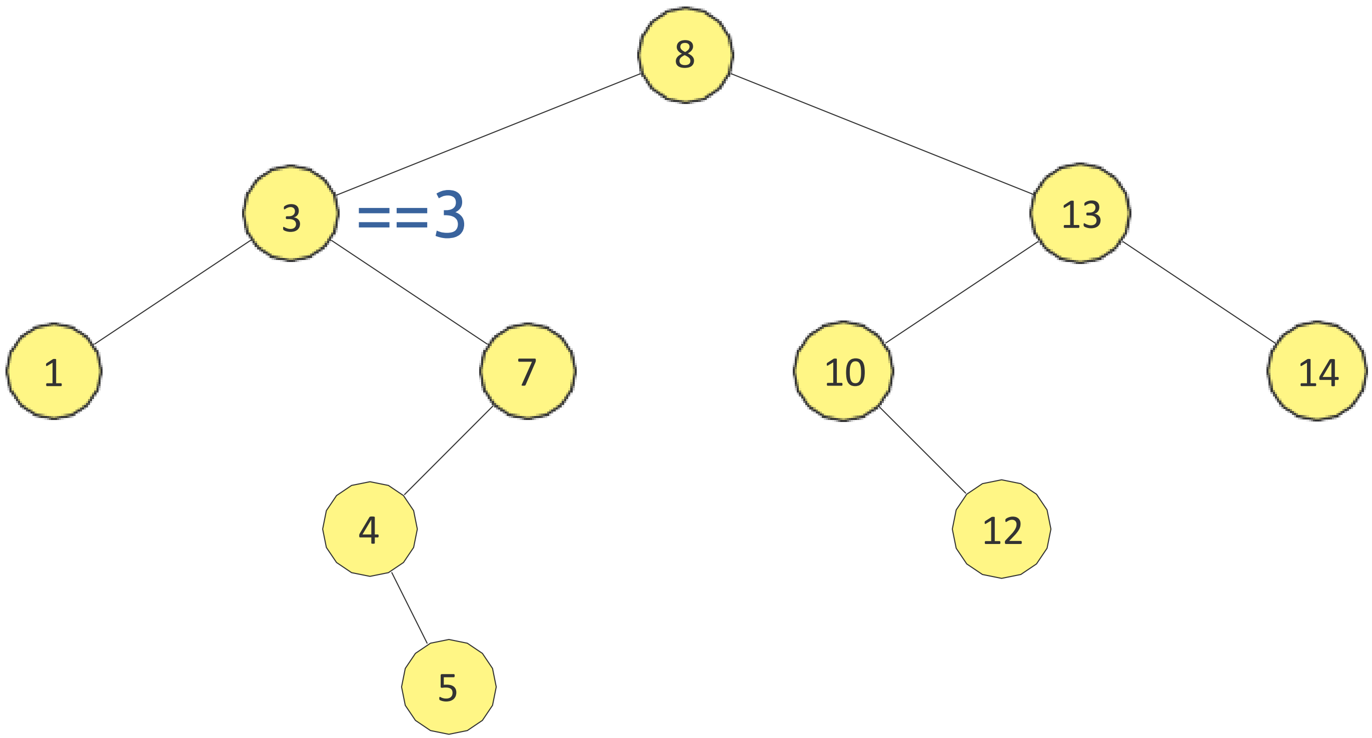
4. Remoção

3. Exemplo: remover o valor 3 (2 filhos)



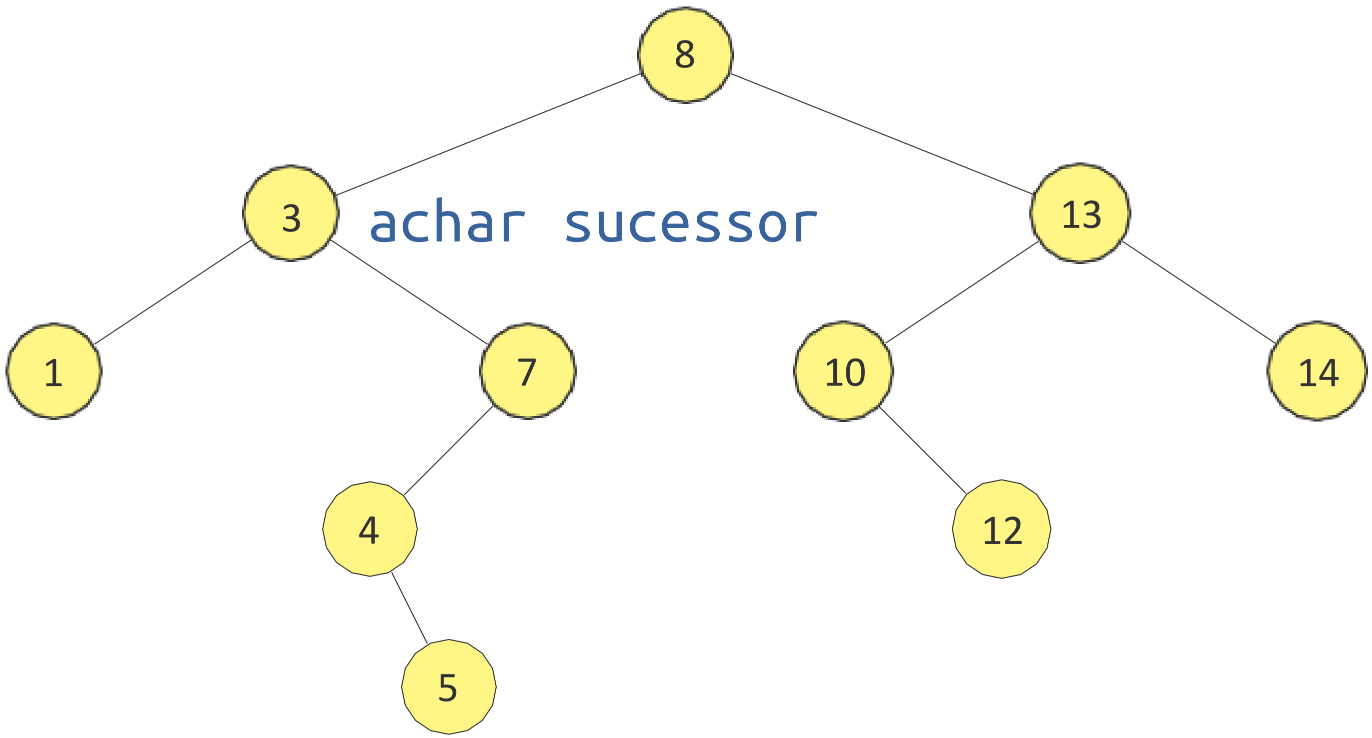
4. Remoção

3. Exemplo: remover o valor 3 (2 filhos)



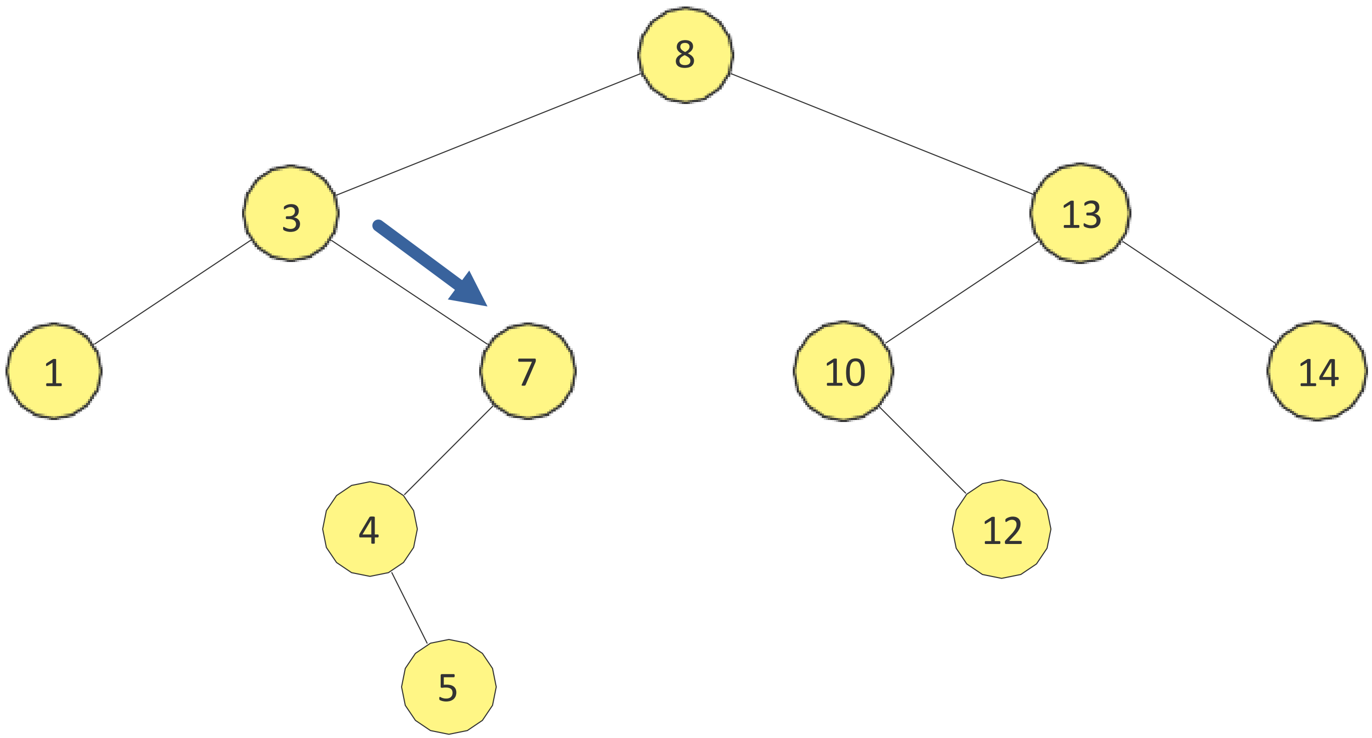
4. Remoção

3. Exemplo: remover o valor **3** (2 filhos)



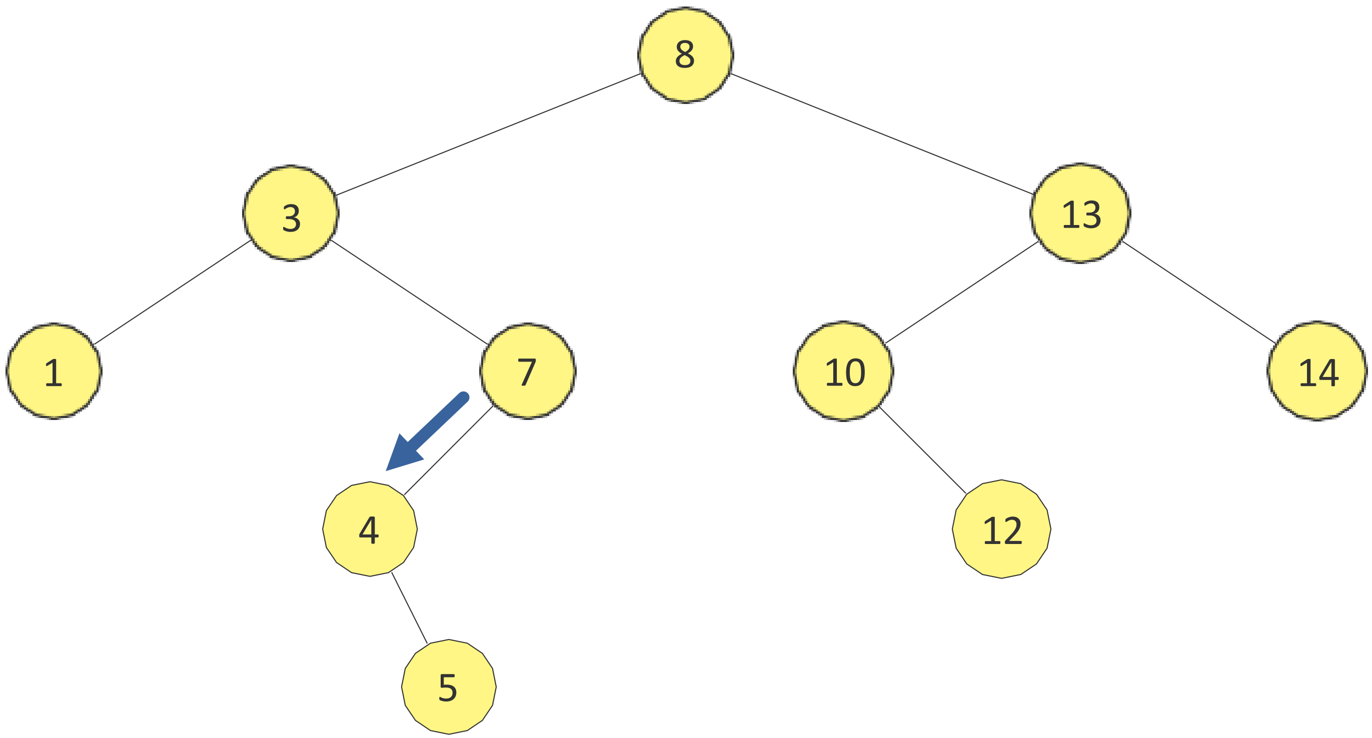
4. Remoção

3. Exemplo: remover o valor 3 (2 filhos)



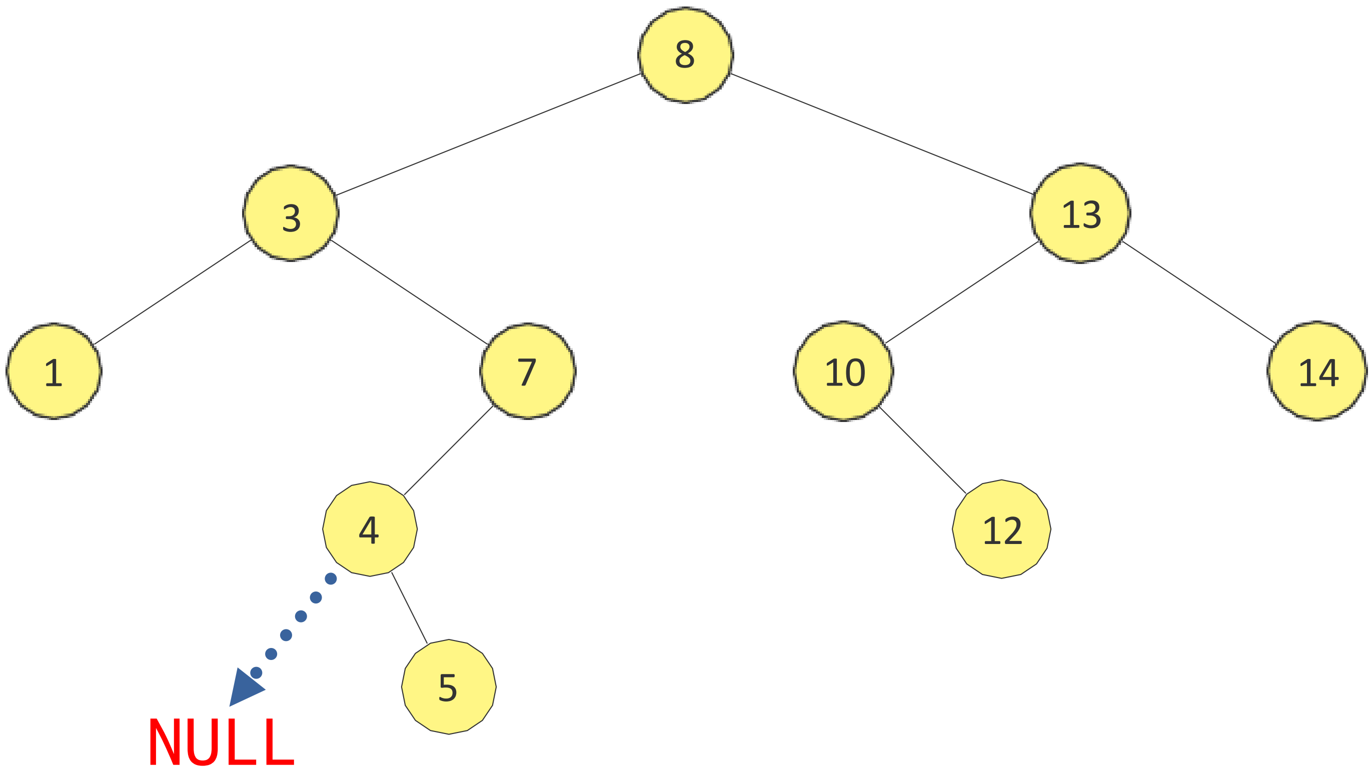
4. Remoção

3. Exemplo: remover o valor **3** (2 filhos)



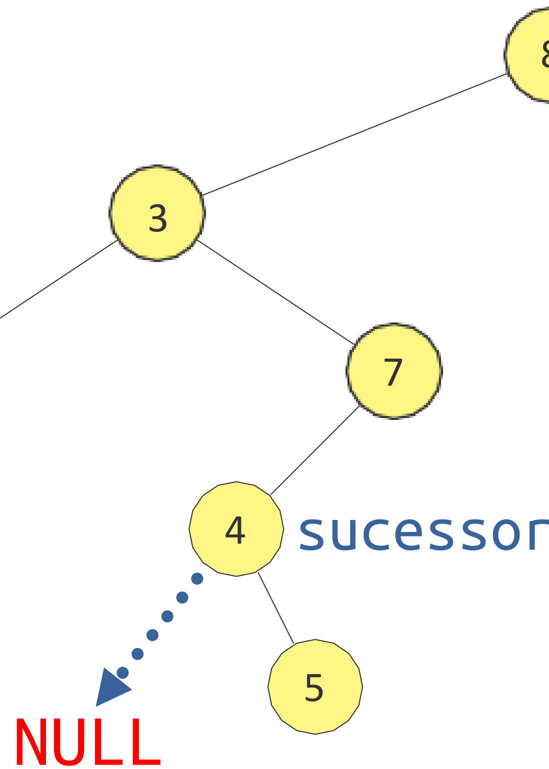
4. Remoção

3. Exemplo: remover o valor **3** (2 filhos)



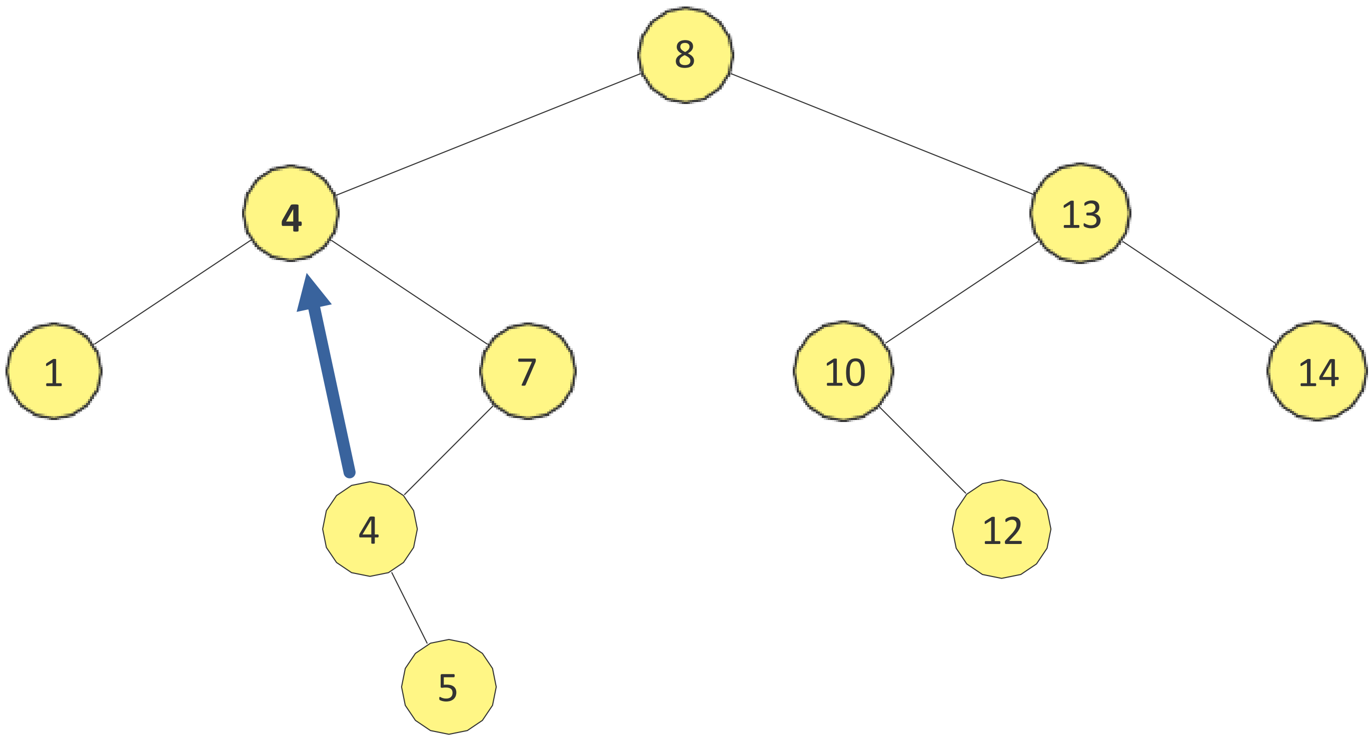
4. Remoção

3. Exemplo: remover o valor 3 (2 filhos)



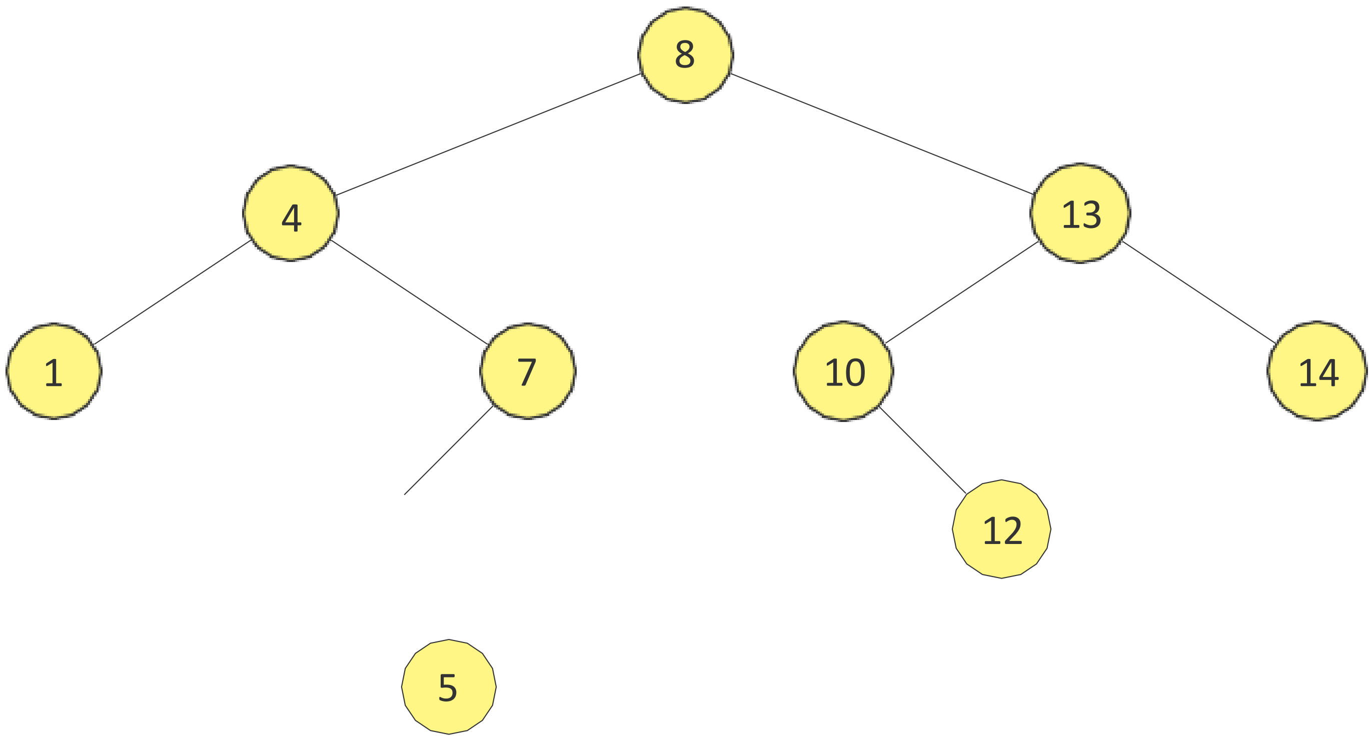
4. Remoção

3. Exemplo: remover o valor 3 (2 filhos)



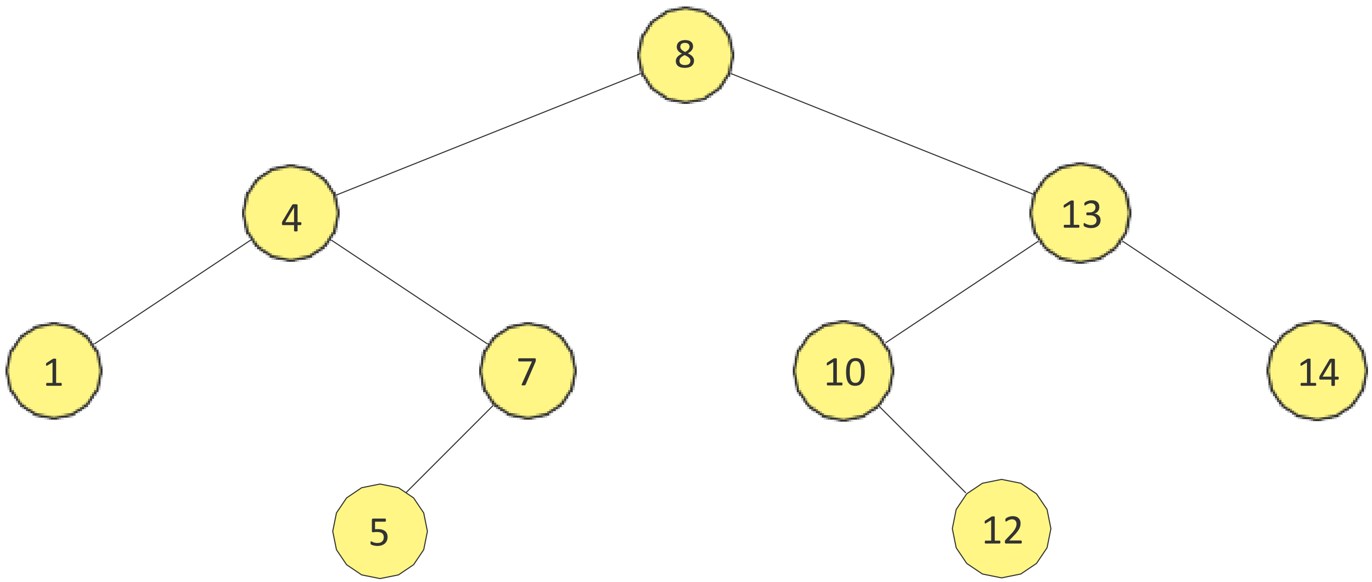
4. Remoção

3. Exemplo: remover o valor **3** (2 filhos)



4. Remoção

3. Exemplo: remover o valor **3** (2 filhos)



4.2. Implementação

```
int retira (TRegistro x, TNo** p) {
  TNo* pAux;

  if (*p == NULL)
    return 0;

  if (x.chave < (*p)->reg.chave)
    return retira (x, &(*p)->pEsq);

  if (x.chave > (*p)->reg.chave)
    return retira (x, &(*p)->pDir);
```

4.2. Implementação

```
  /* Continuação...*/
  /* if (x.chave == (*p)->reg.chave) */
  if ((*p)->pDir == NULL) {
    pAux = *p;
    *p = (*p)->pEsq;
    free (pAux);
    return 1; }

  if ((*p)->pEsq == NULL) {
    pAux = *p;
    *p = (*p)->pDir;
    free (pAux);
    return 1;}

  /* Dois filhos */
  sucessor (*p, &(*p)->pDir);
  return 1;
}
```

4.2. Implementação

```
void sucessor (TNo* q, TNo** r) {
  TNo* pAux;

  if ((*r)->pEsq != NULL) {
    sucessor (q, &(*r)->pEsq);
    return; }

  q->reg = (*r)->reg;
  pAux = *r;
  *r = (*r)->pDir;
  free (pAux);
}
```

5. Caminhamento

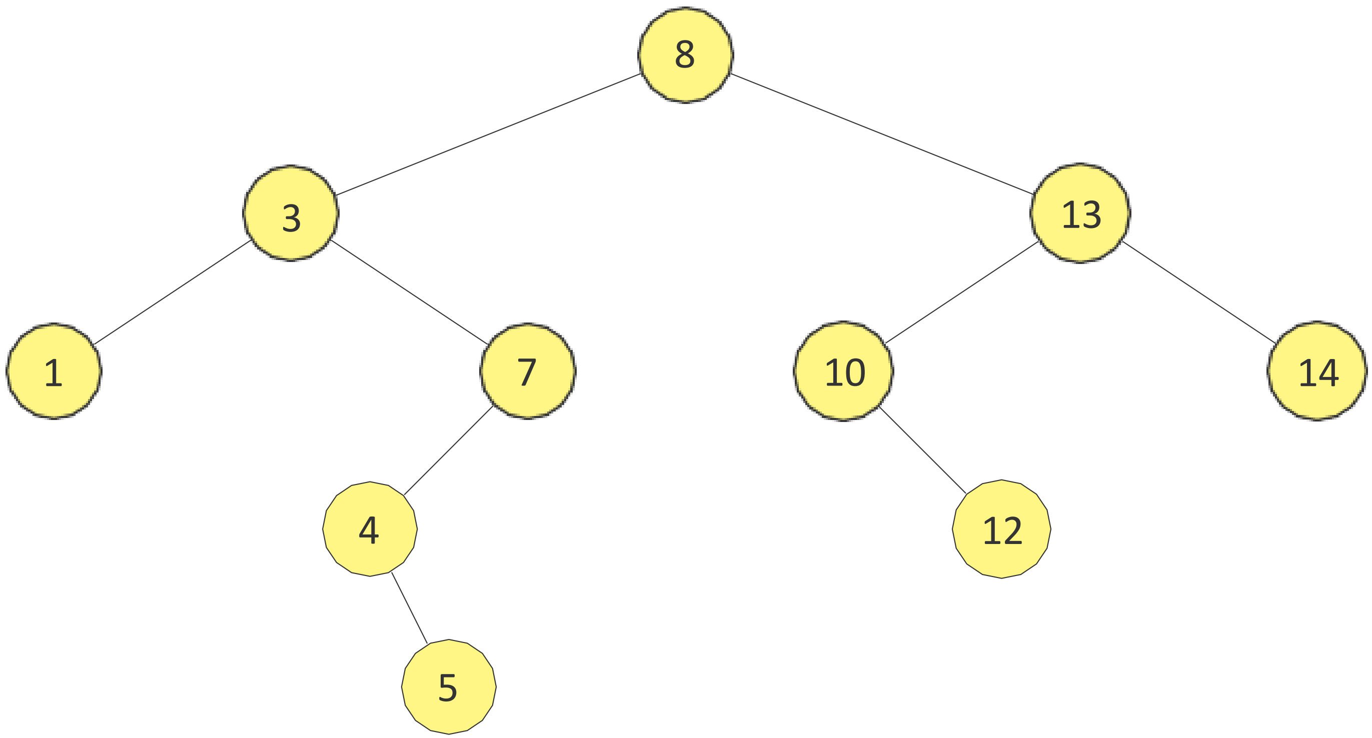
- Após construída a árvore, pode ser necessário percorrer todo os registros que a compõem.

- Existe mais de uma ordem de caminhamento em árvores, mas a mais útil é a chamada ordem de caminhamento **central (InOrder)**.

- Uma característica importante do caminhamento central é que os nós são visitados de forma **ordenada**.

5. Caminhamento

- **InOrder:** 1, 3, 4, 5, 7, 8, 10, 12, 13, 14



5.1. Implementação

```
void central (TNo* pRaiz) {

  if (pRaiz != NULL) {

    central (pRaiz->pEsq);
    printf ("%ld\n", pRaiz->reg.chave);
    central (pRaiz->pDir);
  }
}
```

6. Referências

- Material de aula dos Profs. Luiz Chaimowicz e Raquel O. Prates, da UFMG: https://homepages.dcc.ufmg.br/~glpappa/aeds2/AEDS2.1%20Conceitos%20Basicos%20TAD.pdf

- Horowitz, E. & Sahni, S.; Fundamentos de Estruturas de Dados, Editora Campus, 1984.

- Wirth, N.; Algoritmos e Estruturas de Dados, Prentice/Hall do Brasil, 1989.

- Material de aula do Prof. José Augusto Baranauskas, da USP: https://dcm.ffclrp.usp.br/~augusto/teaching.htm

- Material de aula do Prof. Rafael C. S. Schouery, da Unicamp: https://www.ic.unicamp.br/~rafael/cursos/2s2019/mc202/index.html