Árvores: árvores binárias

Prof. Marcelo Rosa

Algoritmos e Estrutura de Dados 2 (AE43CP)
Engenharia de Computação
Departamento Acadêmico de Informática (Dainf)
Universidade Tecnológica Federal do Paraná (UTFPR)
Campus Pato Branco

DAINF
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CÂMPUS PATO BRANCO

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Sumário

1 Introdução

2 Árvores Binárias
- Operação de Busca
- Operação de Inserção
- Operação de Remoção
- Propriedades
- Percurso

Sumário

1 Introdução

2 Árvores Binárias
- Operação de Busca
- Operação de Inserção
- Operação de Remoção
- Propriedades
- Percurso

Introdução

- Listas lineares, filas e pilhas não são adequadas para representar dados que devem ser organizados de forma hierárquica
- Árvore é uma estrutura de dados muito eficiente para armazenamento de informação
- Árvores são estruturas de dados não lineares
- Aplicações de árvores

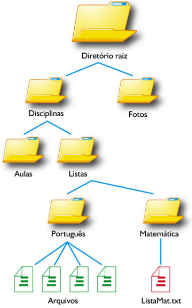


Figure 1: Sistema de arquivos

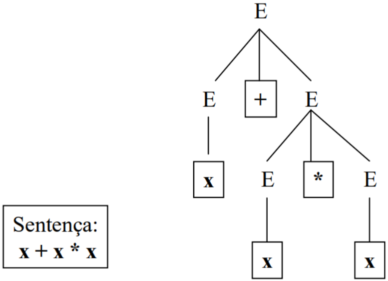


Figure 2: Análise sintática - Compiladores

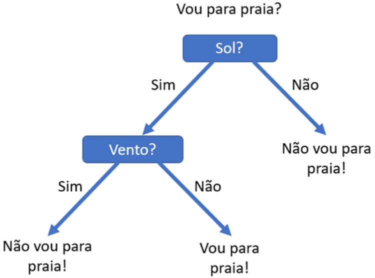


Figure 3: Árvore de decisão - IA

Introdução

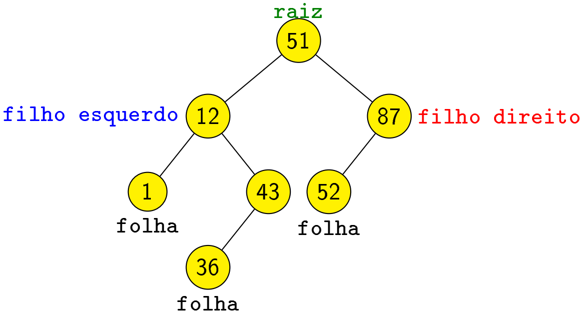
- Exemplos de tipos de árvores
  - Árvore n-ária, com $n > 1$
  - Árvore binária de busca
  - AVL
  - Árvore vermelha-preta (rubro-negra)
  - Árvore B
  - ...

Sumário

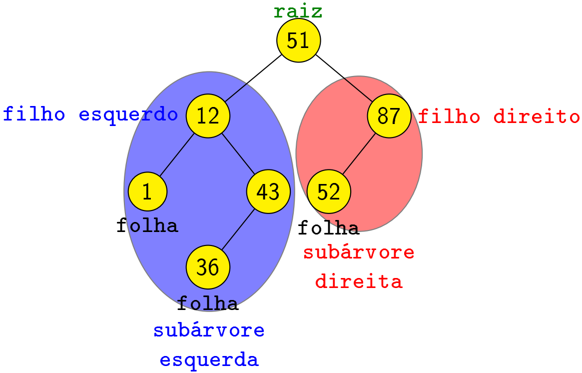
1 Introdução

2 Árvores Binárias
- Operação de Busca
- Operação de Inserção
- Operação de Remoção
- Propriedades
- Percurso

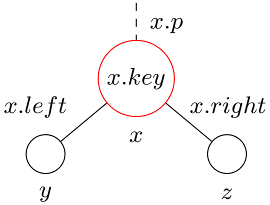
- Uma **árvore binária** é uma estrutura de dados hierárquica em que cada elemento (chamado de **nó**) pode ter, no máximo, **dois filhos**: um à esquerda e outro à direita.
  - O primeiro nó da árvore é chamado de **raiz**.
  - Um nó que não tem filhos é chamado de **folha**.
  - Cada nó pode ter subárvores (esquerda e direita), que também são árvores binárias.



- Uma **árvore binária** é uma estrutura de dados hierárquica em que cada elemento (chamado de **nó**) pode ter, no máximo, **dois filhos**: um à esquerda e outro à direita.
  - O primeiro nó da árvore é chamado de **raiz**.
  - Um nó que não tem filhos é chamado de **folha**.
  - Cada nó pode ter subárvores (esquerda e direita), que também são árvores binárias.



- Uma árvore binária T pode ser representada por uma lista encadeada, onde cada nó x é um objeto com os seguintes atributos



Árvore binária de busca

- **Propriedade da árvore binária de busca**: para todo nó x em T é verdade que
  1. para todo nó y na **subárvore esquerda** T_e de x

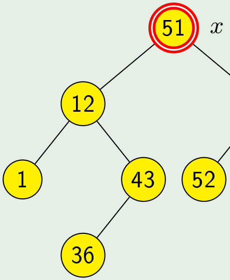
$$y.key \leq x.key$$

Árvore binária de busca

- **Propriedade da árvore binária de busca**: para todo nó x em T é verdade que
  1. para todo nó y na **subárvore esquerda** T_e de x

$$y.key \leq x.key$$

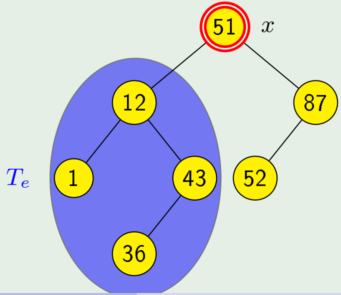
Exemplo



- **Propriedade da árvore binária de busca**: para todo nó x em T é verdade que
  1. para todo nó y na **subárvore esquerda** T_e de x

$$y.key \leq x.key$$

Exemplo



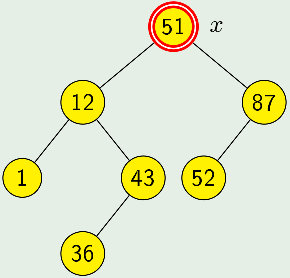
Árvore binária de busca

- **Propriedade da árvore binária de busca**: para todo nó x em T é verdade que
  1. para todo nó y na **subárvore esquerda** T_e de x
  $$y.key \leq x.key$$
  2. para todo nó z na **subárvore direita** T_d de x
  $$z.key \geq x.key$$

Árvore binária de busca

- **Propriedade da árvore binária de busca**: para todo nó x em T é verdade que
  1. para todo nó y na **subárvore esquerda** T_e de x
  $$y.key \leq x.key$$
  2. para todo nó z na **subárvore direita** T_d de x
  $$z.key \geq x.key$$

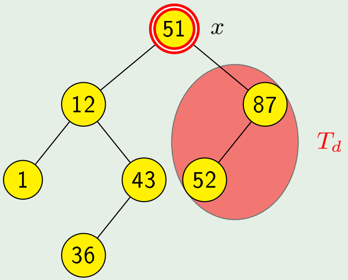
Exemplo



Árvore binária de busca

- **Propriedade da árvore binária de busca**: para todo nó x em T é verdade que
  1. para todo nó y na **subárvore esquerda** T_e de x
  $$y.key \leq x.key$$
  2. para todo nó z na **subárvore direita** T_d de x
  $$z.key \geq x.key$$

Exemplo



Sumário

1 Introdução

2 Árvores Binárias
- Operação de Busca
- Operação de Inserção
- Operação de Remoção
- Propriedades
- Percurso

Operação de busca

- Como verificamos se um nó com uma dada chave de identificação k está em uma árvore binária de busca T?

Operação de busca

- Como verificamos se um nó com uma dada chave de identificação k está em uma árvore binária de busca T?
    1. Comece a busca a partir da raiz, vamos usar x como sendo o nó atual

Operação de busca

- Como verificamos se um nó com uma dada chave de identificação k está em uma árvore binária de busca T?
  1. Comece a busca a partir da raiz, vamos usar x como sendo o nó atual
  2. Se a (sub)árvore estiver vazia ($x = Null$), a busca termina sem encontrar o nó

Operação de busca

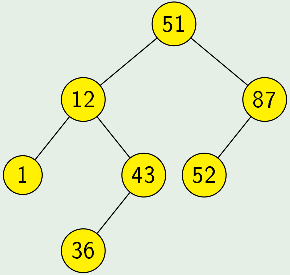
- Como verificamos se um nó com uma dada chave de identificação k está em uma árvore binária de busca T?
  1. Comece a busca a partir da raiz, vamos usar x como sendo o nó atual
  2. Se a (sub)árvore estiver vazia ($x = Null$), a busca termina sem encontrar o nó
  3. Caso contrário, compare k com a chave ($x.key$) do nó atual x
     - Se $x.key$ for igual a k, então retorne x
     - Se $x.key$ for menor, torne o filho da esquerda de x o nó atual e retorne para o passo 2.
     - Se $x.key$ for maior, torne o filho da direita de x o nó atual e retorne para o passo 2.

Operação de busca

- Como verificamos se um nó com uma dada chave de identificação k está em uma árvore binária de busca T?
  1. Comece a busca a partir da raiz, vamos usar x como sendo o nó atual
  2. Se a (sub)árvore estiver vazia ($x = Null$), a busca termina sem encontrar o nó
  3. Caso contrário, compare k com a chave ($x.key$) do nó atual x
     - Se $x.key$ for igual a k, então retorne x
     - Se $x.key$ for menor, torne o filho da esquerda de x o nó atual e retorne para o passo 2.
     - Se $x.key$ for maior, torne o filho da direita de x o nó atual e retorne para o passo 2.

Exemplo

Considere $k = 36$

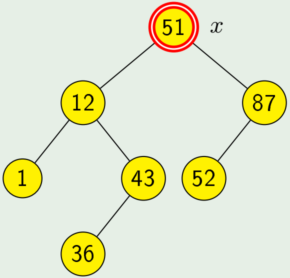


Operação de busca

- Como verificamos se um nó com uma dada chave de identificação k está em uma árvore binária de busca T?
  1. Comece a busca a partir da raiz, vamos usar x como sendo o nó atual
  2. Se a (sub)árvore estiver vazia ($x = Null$), a busca termina sem encontrar o nó
  3. Caso contrário, compare k com a chave ($x.key$) do nó atual x
     - Se $x.key$ for igual a k, então retorne x
     - Se $x.key$ for menor, torne o filho da esquerda de x o nó atual e retorne para o passo 2.
     - Se $x.key$ for maior, torne o filho da direita de x o nó atual e retorne para o passo 2.

Exemplo

Considere $k = 36$

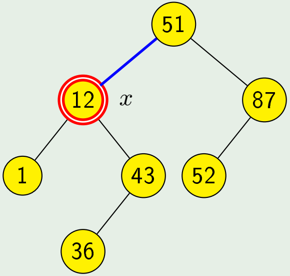


Operação de busca

- Como verificamos se um nó com uma dada chave de identificação k está em uma árvore binária de busca T?
  1. Comece a busca a partir da raiz, vamos usar x como sendo o nó atual
  2. Se a (sub)árvore estiver vazia ($x = Null$), a busca termina sem encontrar o nó
  3. Caso contrário, compare k com a chave ($x.key$) do nó atual x
     - Se $x.key$ for igual a k, então retorne x
     - Se $x.key$ for menor, torne o filho da esquerda de x o nó atual e retorne para o passo 2.
     - Se $x.key$ for maior, torne o filho da direita de x o nó atual e retorne para o passo 2.

Exemplo

Considere $k = 36$

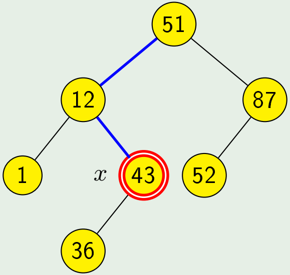


Operação de busca

- Como verificamos se um nó com uma dada chave de identificação k está em uma árvore binária de busca T?
  1. Comece a busca a partir da raiz, vamos usar x como sendo o nó atual
  2. Se a (sub)árvore estiver vazia ($x = Null$), a busca termina sem encontrar o nó
  3. Caso contrário, compare k com a chave ($x.key$) do nó atual x
     - Se $x.key$ for igual a k, então retorne x
     - Se $x.key$ for menor, torne o filho da esquerda de x o nó atual e retorne para o passo 2.
     - Se $x.key$ for maior, torne o filho da direita de x o nó atual e retorne para o passo 2.

Exemplo

Considere $k = 36$

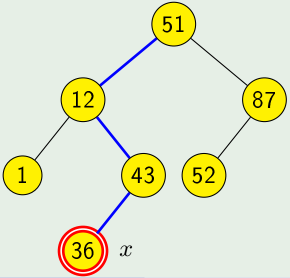


Operação de busca

- Como verificamos se um nó com uma dada chave de identificação k está em uma árvore binária de busca T?
  1. Comece a busca a partir da raiz, vamos usar x como sendo o nó atual
  2. Se a (sub)árvore estiver vazia ($x = Null$), a busca termina sem encontrar o nó
  3. Caso contrário, compare k com a chave ($x.key$) do nó atual x
     - Se $x.key$ for igual a k, então retorne x
     - Se $x.key$ for menor, torne o filho da esquerda de x o nó atual e retorne para o passo 2.
     - Se $x.key$ for maior, torne o filho da direita de x o nó atual e retorne para o passo 2.

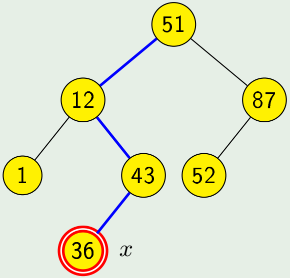
Exemplo

Considere $k = 36$



Exemplo: $k = 36$

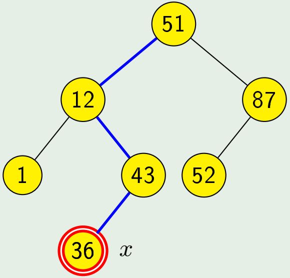
- **Exemplo:** considere $k = 36$



Complexidade da operação de busca

Exemplo: $k = 36$

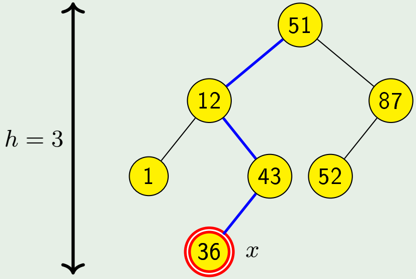
- **Exemplo:** considere $k = 36$



- A **altura** de um dado nó z em uma árvore binária T é definida como sendo o **número de arestas (ou arcos) do maior caminho** de z até um nó folha.

Exemplo: $k = 36$

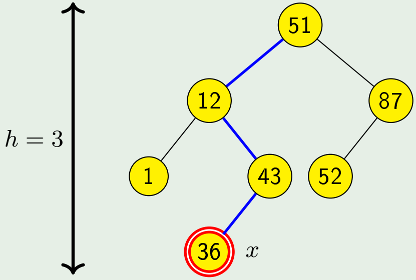
- **Exemplo:** considere $k = 36$



- A **altura** de um dado nó z em uma árvore binária T é definida como sendo o **número de arestas (ou arcos) do maior caminho** de z até um nó folha.

Exemplo: $k = 36$

- **Exemplo:** considere $k = 36$



- A **altura** de um dado nó z em uma árvore binária T é definida como sendo o **número de arestas (ou arcos) do maior caminho** de z até um nó folha.
- No pior caso o tempo de execução da operação de busca é $O(h)$, onde h é altura da árvore que corresponde a altura do nó raiz.

- Tempo de busca por um determinado valor (também o mínimo ou o máximo)
  - Melhor caso: $O(1)$
  - Caso médio: $O(\log n)$
  - Pior caso: $O(n)$

Operação de busca

Pseudo-código

```
function Tree-Search(x, k)
    if x == NULL or k == x.key then
        return x
    end if
    if k < x.key then
        return Tree-Search(x.left, k)
    else
        return Tree-Search(x.right, k)
    end if
end function
```

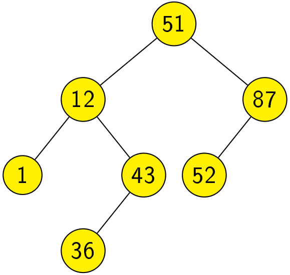
Sumário

1 Introdução

2 Árvores Binárias
- Operação de Busca
- Operação de Inserção
- Operação de Remoção
- Propriedades
- Percurso

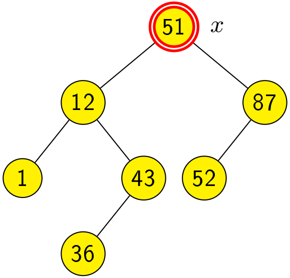
Operação de inserção

- Seja z, com $z.key = 55$, o nó a ser inserido.



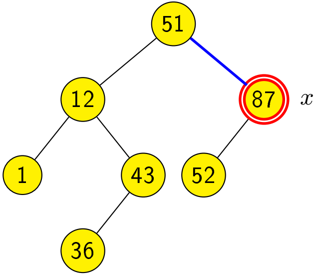
Operação de inserção

- Seja z, com $z.key = 55$, o nó a ser inserido.



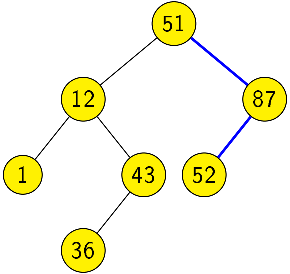
Operação de inserção

- Seja z, com $z.key = 55$, o nó a ser inserido.



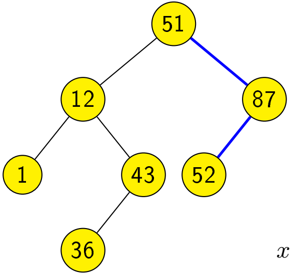
Operação de inserção

- Seja z, com $z.key = 55$, o nó a ser inserido.



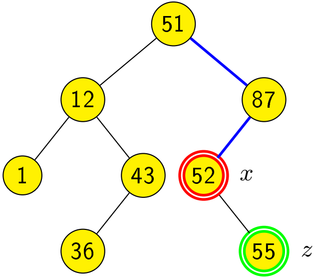
Operação de inserção

- Seja z, com $z.key = 55$, o nó a ser inserido.



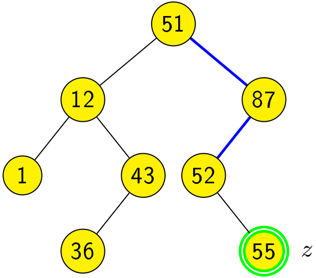
$x = \texttt{NULL}$

- Seja z, com $z.key = 55$, o nó a ser inserido.



Operação de inserção

- Seja z, com $z.key = 55$, o nó a ser inserido.



- Para inserir um novo nó z em uma árvore binária de busca, começamos na raiz e percorremos a árvore comparando a chave do nó a ser inserido com a chave do nó atual x.
- Se a chave for menor que a chave do nó atual, seguimos para a subárvore esquerda; se for maior, seguimos para a subárvore direita.
- Repetimos esse processo até encontrar um nó vazio/`NULL` (um ponto de inserção), onde z será inserido.
- Então, adicionamos z como filho esquerdo ou direito do nó pai (y) do nó vazio dependendo se a chave de z é menor ou maior que a chave de y.

- Tempo para inserir um determinado valor
  - Melhor caso: $O(1)$
  - Caso médio: $O(\log n)$
  - Pior caso: $O(n)$

Operação de inserção

Pseudo-código

```
procedure Tree-Insert(T, z)
    y ← NULL
    x ← T.root
    while x ≠ NULL do
        y ← x
        if z.key < x.key then
            x ← x.left
        else
            x ← x.right
        end if
    end while
    z.p ← y
    if y == NULL then
        T.root = z
    else if z.key < y.key then
        y.left ← z
    else
        y.right ← z
    end if
end procedure
```

Sumário

1 Introdução

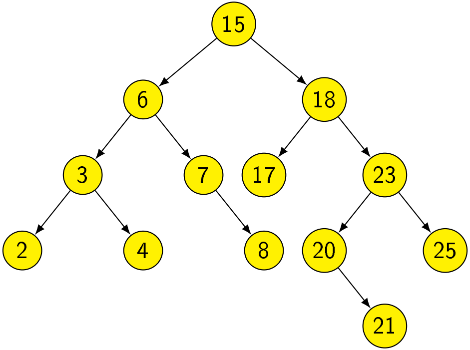
2 Árvores Binárias
- Operação de Busca
- Operação de Inserção
- Operação de Remoção
- Propriedades
- Percurso

Casos

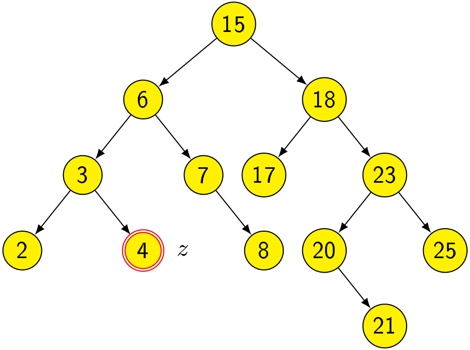
A remoção de um nó z de uma árvore binária de busca envolve considerar os seguintes casos:

1. se o nó z for uma folha ($z.left = z.right = \texttt{NULL}$), simplesmente removemos o nó.
2. se o nó z tiver apenas um filho ($z.left \neq \texttt{NULL}$ ou $z.right \neq \texttt{NULL}$), ligamos o pai do nó z ao único filho de z.
3. se o nó z tiver dois filhos ($z.left \neq \texttt{NULL}$ e $z.right \neq \texttt{NULL}$), encontramos o sucessor (y) do nó z (o nó mínimo da subárvore direita enraizada em z):
   1. Se y é exatamente o filho da direita de z, substituímos z por y
   2. Caso contrário, substituímos o nó z por y e então removemos y da posição original.

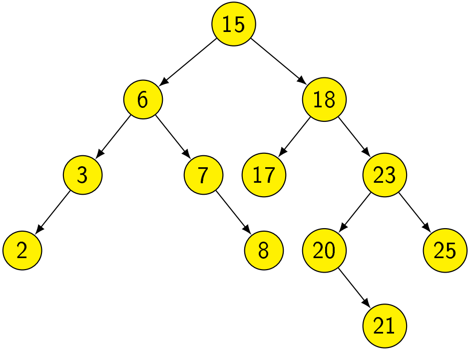
1. se o nó z for uma folha ($z.left = z.right =$ NULL), simplesmente removemos o nó.



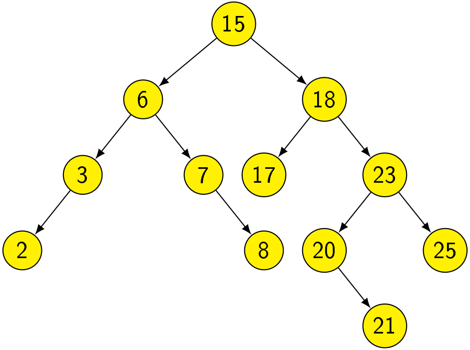
1. se o nó z for uma folha ($z.left = z.right =$ NULL), simplesmente removemos o nó.



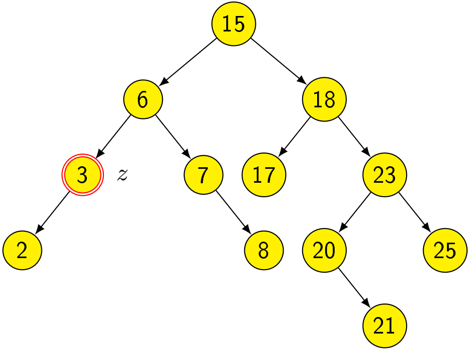
1. se o nó z for uma folha ($z.left = z.right =$ NULL), simplesmente removemos o nó.



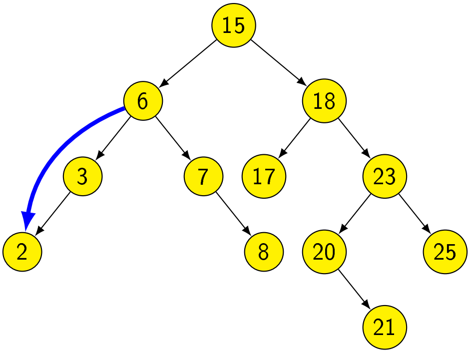
2. se o nó z tiver apenas um filho ($z.left \neq \texttt{NULL}$ ou $z.right \neq \texttt{NULL}$), ligamos o pai do nó z ao único filho de z.



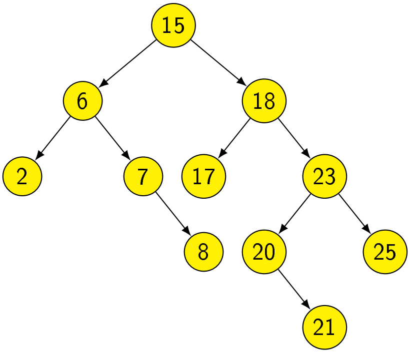
2. se o nó z tiver apenas um filho ($z.left \neq$ `NULL` ou $z.right \neq$ `NULL`), ligamos o pai do nó z ao único filho de z.



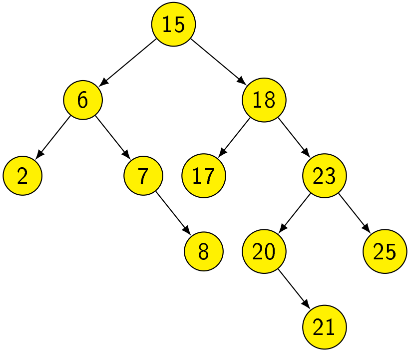
2. se o nó z tiver apenas um filho ($z.left \neq \texttt{NULL}$ ou $z.right \neq \texttt{NULL}$), ligamos o pai do nó z ao único filho de z.



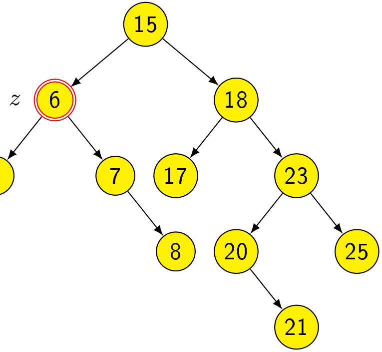
2. se o nó z tiver apenas um filho ($z.left \neq$ `NULL` ou $z.right \neq$ `NULL`), ligamos o pai do nó z ao único filho de z.



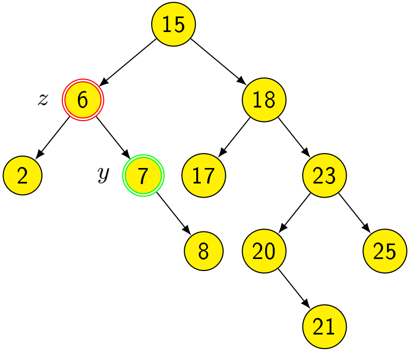
3. se o nó z tiver dois filhos ($z.left \neq$ NULL e $z.right \neq$ NULL), encontramos o sucessor (y) do nó z (o nó mínimo da subárvore direita enraizada em z):
   a. Se y é exatamente o filho da direita de z, substituímos z por y



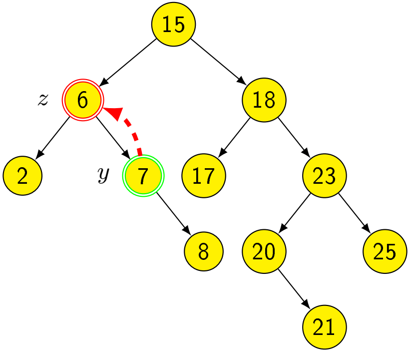
3. se o nó z tiver dois filhos ($z.left \neq$ NULL e $z.right \neq$ NULL), encontramos o sucessor (y) do nó z (o nó mínimo da subárvore direita enraizada em z):
   a. Se y é exatamente o filho da direita de z, substituímos z por y



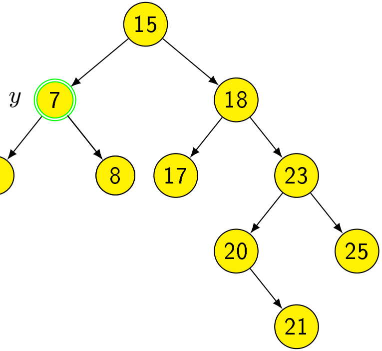
3. se o nó z tiver dois filhos ($z.left \neq$ NULL e $z.right \neq$ NULL), encontramos o sucessor (y) do nó z (o nó mínimo da subárvore direita enraizada em z):
   a. Se y é exatamente o filho da direita de z, substituímos z por y



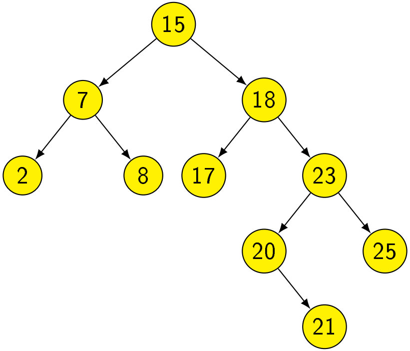
3. se o nó z tiver dois filhos ($z.left \neq$ NULL e $z.right \neq$ NULL), encontramos o sucessor (y) do nó z (o nó mínimo da subárvore direita enraizada em z):
   a. Se y é exatamente o filho da direita de z, substituímos z por y



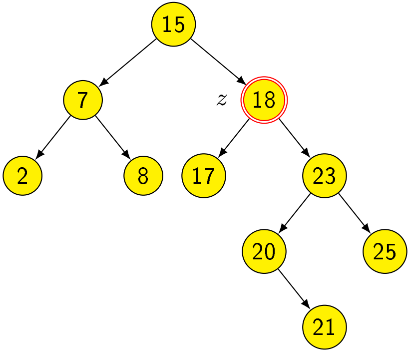
- ❸ se o nó z tiver dois filhos ($z.left \neq$ NULL e $z.right \neq$ NULL), encontramos o sucessor (y) do nó z (o nó mínimo da subárvore direita enraizada em z):
  - ⓐ Se y é exatamente o filho da direita de z, substituímos z por y



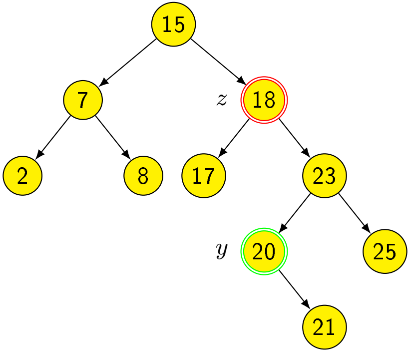
3. se o nó z tiver dois filhos ($z.left \neq$ NULL e $z.right \neq$ NULL), encontramos o sucessor (y) do nó z (o nó mínimo da subárvore direita enraizada em z):
   b. Caso contrário, substituímos o nó z por y e então removemos y da posição original.



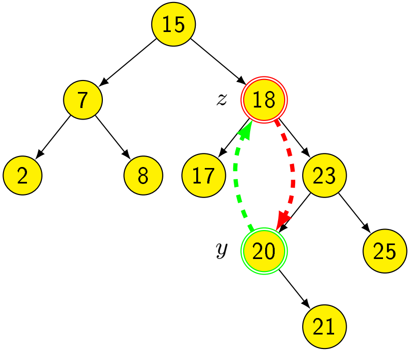
3. se o nó z tiver dois filhos ($z.left \neq$ NULL e $z.right \neq$ NULL), encontramos o sucessor (y) do nó z (o nó mínimo da subárvore direita enraizada em z):
   - b. Caso contrário, substituímos o nó z por y e então removemos y da posição original.



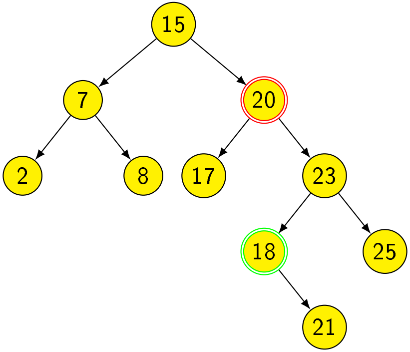
3. se o nó z tiver dois filhos ($z.left \neq$ NULL e $z.right \neq$ NULL), encontramos o sucessor (y) do nó z (o nó mínimo da subárvore direita enraizada em z):
   b. Caso contrário, substituímos o nó z por y e então removemos y da posição original.



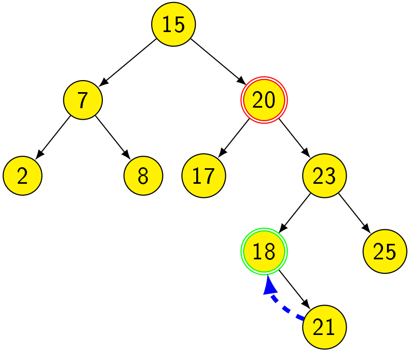
3. se o nó z tiver dois filhos ($z.left \neq$ NULL e $z.right \neq$ NULL), encontramos o sucessor (y) do nó z (o nó mínimo da subárvore direita enraizada em z):
   - b. Caso contrário, substituímos o nó z por y e então removemos y da posição original.



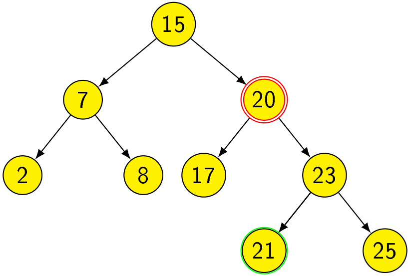
3. se o nó z tiver dois filhos ($z.left \neq$ NULL e $z.right \neq$ NULL), encontramos o sucessor (y) do nó z (o nó mínimo da subárvore direita enraizada em z):
   - b. Caso contrário, substituímos o nó z por y e então removemos y da posição original.



3. se o nó z tiver dois filhos ($z.left \neq$ NULL e $z.right \neq$ NULL), encontramos o sucessor (y) do nó z (o nó mínimo da subárvore direita enraizada em z):
   - b. Caso contrário, substituímos o nó z por y e então removemos y da posição original.



3. se o nó z tiver dois filhos ($z.left \neq$ NULL e $z.right \neq$ NULL), encontramos o sucessor (y) do nó z (o nó mínimo da subárvore direita enraizada em z):
   b. Caso contrário, substituímos o nó z por y e então removemos y da posição original.



- Tempo para remover um determinado valor
  - Melhor caso: $O(1)$
  - Caso médio: $O(\log n)$
  - Pior caso: $O(n)$

Operação de Remoção

Pseudo-código

```
procedure Tree-Delete(T, z)
    if z.left = NULL then
        Transplant(T, z, z.right)
    else if z.right = NULL then
        Transplant(T, z, z.left)
    else
        y ← Tree-Minimum(z.right)
        if y.p ≠ z then
            Transplant(T, y, y.right)
            y.right ← z.right
            y.right.p ← y
        end if
        Transplant(T, z, y)
        y.left ← z.left
        y.left.p ← y
    end if
end procedure
```

```
procedure Transplant(T, u, v)
    if u.p = NULL then
        T.root ← v
    else if u = u.p.left then
        u.p.left ← v
    else
        u.p.right ← v
    end if
    if v ≠ NULL then
        v.p = u.p
    end if
end procedure
```

```
function Tree-Minimum(x)
    a ← x;
    while a.left ≠ NULL do
        a ← a.left
    end while
    return a
end function
```

Sumário

1 Introdução

2 Árvores Binárias

- Operação de Busca
- Operação de Inserção
- Operação de Remoção
- Propriedades
- Percurso

- Propriedades de árvores binárias
  - Uma árvore binária com n elementos tem $n-1$ ramos
  - Uma árvore binária de altura h tem no mínimo $h+1$ elementos e no máximo $\sum_{i=0}^{h} 2^i = 2^{h+1} - 1$
  - Uma árvore binária de altura h com $2^{h+1} - 1$ elementos é denominada árvore cheia
  - A altura de uma árvore binária com n elementos $(n > 0)$ é no máximo $n-1$ e no mínimo $\log n - 1$
  - Em uma árvore binária cheia (cada nó, exceto folha, possui dois descendentes), a quantidade total de folhas é 2^h

Sumário

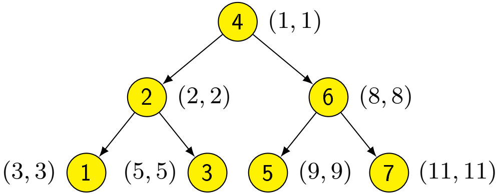
1 Introdução

2 Árvores Binárias
- Operação de Busca
- Operação de Inserção
- Operação de Remoção
- Propriedades
- Percurso

- Pré-ordem (Raiz $\rightarrow$ Esquerda $\rightarrow$ Direita)
  - A raiz é visitada primeiramente
  - Após, as sub-árvores da esquerda à direita são processadas em pré-ordem

```
void prefix(Node* tree){
  if (tree != NULL){
    printf("%d", tree->item);
    prefix(tree->left);
    prefix(tree->right);
  }
}
```

- Pré-ordem (Raiz $\rightarrow$ Esquerda $\rightarrow$ Direita)

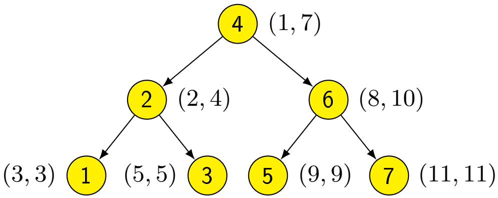


- Percurso: 4, 2, 1, 3, 6, 5, 7

- Ordem simétrica/Em ordem (Esquerda $\rightarrow$ Raiz $\rightarrow$ Direita)
  - Primeiramente, a sub-árvore esquerda é percorrida em ordem simétrica
  - Após, a raiz é visitada
  - Por último, a sub-árvore direita é percorrida em ordem simétrica

```
void infix(Node* tree){
  if (tree != NULL){
    infix(tree->left);
    printf("%d", tree->item);
    infix(tree->right);
  }
}
```

- Ordem simétrica/Em ordem (Esquerda $\rightarrow$ Raiz $\rightarrow$ Direita)

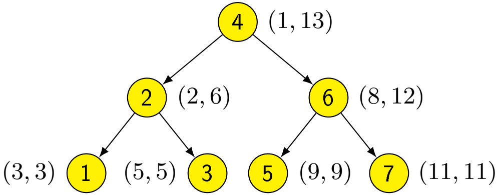


- Percurso: 1, 2, 3, 4, 5, 6, 7

- Pós-ordem (Raiz $\rightarrow$ Esquerda $\rightarrow$ Direita)
  - A raiz é a última a ser visitada
  - Todas as subárvores da esquerda até a direita são percorridas em pós-ordem

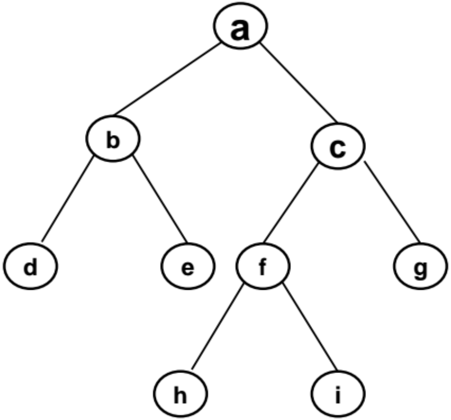
```
void posfix(Node* tree){
  if (tree != NULL){
    posfix(tree->left);
    posfix(tree->right);
    printf("%d", tree->item);
  }
}
```

- Pós-ordem (Esquerda $\rightarrow$ Direita $\rightarrow$ Raiz)



- Percurso: 1, 3, 2, 5, 7, 6, 4

Exercício: para a árvore abaixo, faça os três tipos de percurso



Referências I

Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.
Introduction to Algorithms.
Third edition, The MIT Press, 2009.

Schouery, R. C. S.
Árvores Binárias de Busca. Estrutura de Dados.
Slides. Engenharia de Computação. Unicamp, 2019.
https://ic.unicamp.br/~rafael/cursos/2s2019/mc202/slides/unidade17-arvores-binarias-handout.pdf

Oliva, J. T.
Pilhas Encadeadas. AE22CP – Algoritmos e Estrutura de Dados II.
Notas de Aula. Engenharia de Computação. Dainf/UTFPR/Pato Branco, 2024.