Algoritmos e Estrutura de Dados

Aula 13 – Estrutura de Dados: Árvores de Pesquisa Binária

Prof. Tiago A. E. Ferreira

Introdução

- Uma árvore de pesquisa binária são estruturas de dados que admitem operações em conjuntos dinâmicos, como:
  - Search
  - Minimum
  - Maximum
  - Predecessor
  - Sucessor
  - Insert
  - Delete

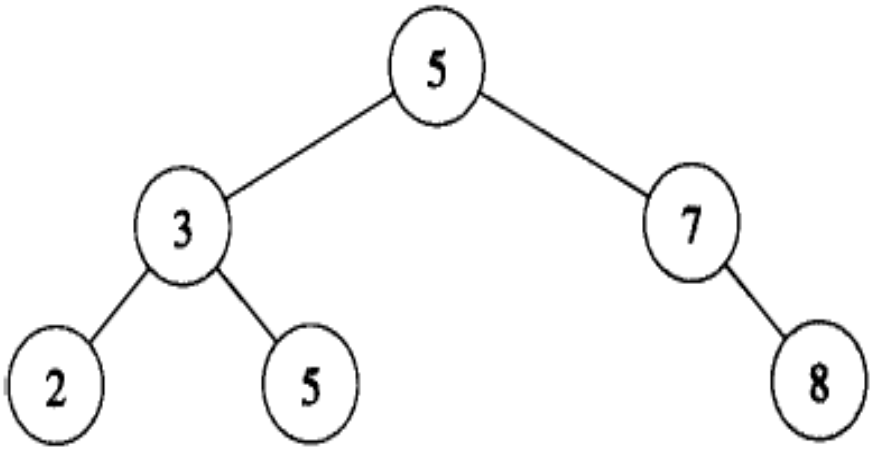
Operações Básicas X Custos

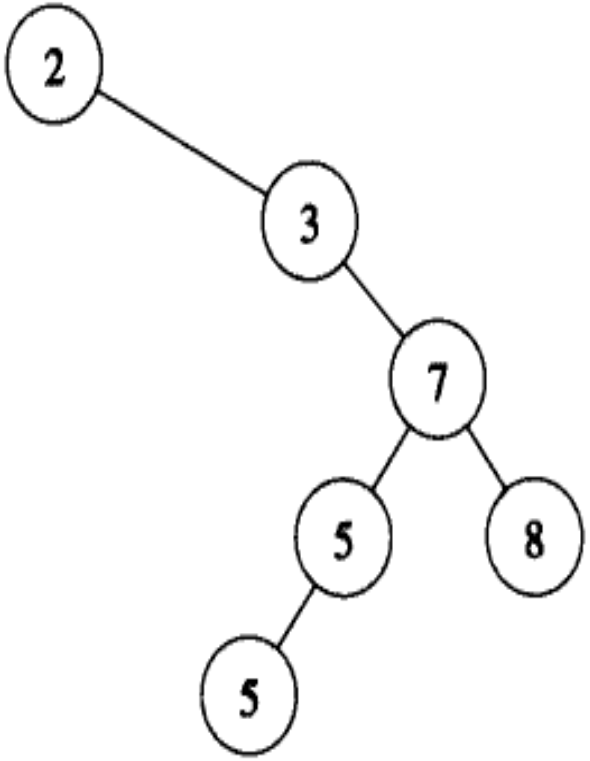
- De forma geral, as operações básicas em uma árvore binária são **proporcionais a sua altura**
  - Para uma árvore binária completa:
    - **Custo em tempo Θ(lgn)**
  - Para uma árvore que é uma cadeia linear de nodos:
    - **Custo em tempo Θ(n)**
  -

Árvore Binária

- Uma árvore binária é um conjunto de objetos nodos.
  - Cada nodo contem:
    - Filho Esquerdo
    - Filho Direito
    - Pai
    - Dados
  - Se o nodo em questão não tiver Pai ou algum Filho, os respectivos campos serão **None.**
- Em uma árvore binária:
  - Todo nodo da sub-árvore esquerda é menor (ou igual) a raiz, e todo nodo na sub-árvore direita é maior (ou igual) a raiz.

Exemplos Árvores Binárias





Percorrendo a Árvore

- Para exibir os elementos da árvore binária em ordem crescente, percorre-se a árvore com o algoritmo **em ordem**.
  - Para percorrer toda a árvore, chama-se a função passando a raiz da árvore.

```
INORDER-TREE-WALK(x)
1  if x ≠ NIL
2      then INORDER-TREE-WALK(left[x])
3           print key[x]
4           INORDER-TREE-WALK(right[x])
```

Analisando o Percurso em Ordem

- No algoritmo em Ordem, para cada nodo da árvore são chamados dois outros nodos recursivamente (esquerdo e direito).
  - Desta forma, o custo em tempo será: **$T(n) = \Theta(n)$**.

- **Teorema 12.1**
  - Se x é a raiz de uma sub-árvore com n nodos, então a chamada da função INORDER-TREE-WALK irá demorá o tempo $T(n) = \Theta(n)$.

Consulta em uma Árvore Binária

- A função mais comum sobre uma Árvore Binária é a consulta de uma chave.
  - Como consultas sobre uma árvore binária, pode-se citar as funções:
    - Search
    - Maximum
    - Minimum
    - Predecessor
    - Sucessor

Função Tree-Search

- Para pesquisar uma elemento sobre uma árvore binária utiliza-se a função:

```
TREE-SEARCH(x, k)
1  if x = NIL or k = key[x]
2      then return x
3  if k < key[x]
4      then return TREE-SEARCH(left[x], k)
5      else  return TREE-SEARCH(right[x], k)
```

- Observe que a pesquisa pela chave gera um caminho descendente na árvore
  - Logo, o custo em tempo será O(h), onde h é a altura ou profundidade da árvore.

Função Tree-Search

- Também é possível gerar uma versão iterativa da função de pesquisa:

```
ITERATIVE-TREE-SEARCH(x, k)
1  while x ≠ NIL and k ≠ key[x]
2        do if k < key[x]
3              then x ← left[x]
4              else x ← right[x]
5  return x
```

Funções de Mínimo e Máximo

- A chave mínima em uma árvore binária será o nodo mais a esquerda.

```
TREE-MINIMUM(x)
1  while left[x] ≠ NIL
2        do x ← left[x]
3  return x
```

- A chave máxima em uma árvore binária será o nodo mais a direita.

```
TREE-MAXIMUM(x)
1  while right[x] ≠ NIL
2        do x ← right[x]
3  return x
```

Analisando As Funções Max. e Min.

- Para uma árvore binária com altura (ou profundidade) h
  - As funções de pesquisa de máximo e mínimo terão custo em tempo de **O(h)**.

Sucessor

- Dada uma árvore binária sem repetição de chaves (ou elementos).
  - Para um certo nodo **x**, o seu **sucessor** será o nodo que tiver a menor chave maior que a chave de x.

```
TREE-SUCCESSOR(x)
1  if right[x] ≠ NIL
2     then return TREE-MINIMUM(right[x])
3  y ← p[x]
4  while y ≠ NIL and x = right[y]
5        do x ← y
6           y ← p[y]
7  return y
```

Sucessor

- A função Tree-Sucessor divide-se em dois casos:
  - Quando a sub-arvore direita do nodo x é não nula
    - Assim, o sucessor de x será o mínimo da sub-árvore direita de x.
  - Quando a sub-arvore direita do nodo x é nula
    - Assim, o sucessor de x, se existir, será o primeiro ancestral de x cujo seu filho esquerdo também é ancestral de x.
- O custo em tempo é O(h), onde h é a altura da árvore.

Predecessor

- Dada uma árvore binária sem repetição de chaves (ou elementos).
  - Para um certo nodo **x**, o seu **predecessor** será o nodo que tiver a maior chave menor que a chave de x.
  - Assim, para um nodo X
    - Se a sub-árvore esquerde de X é não nula, então o predecessor é o máximo da sub-árvore esquerda.
    - Se a sub-árvore esquerda de X é nula, então o predecessor, se existir, será o ancestral de x cujo filho direito também seja ancestral de x.

Inserir e Deletar

- Dada uma árvore binária, é possível inserir e/ou deletar nodos da árvore.
  - Porém, as propriedades da árvore devem ser mantidas!

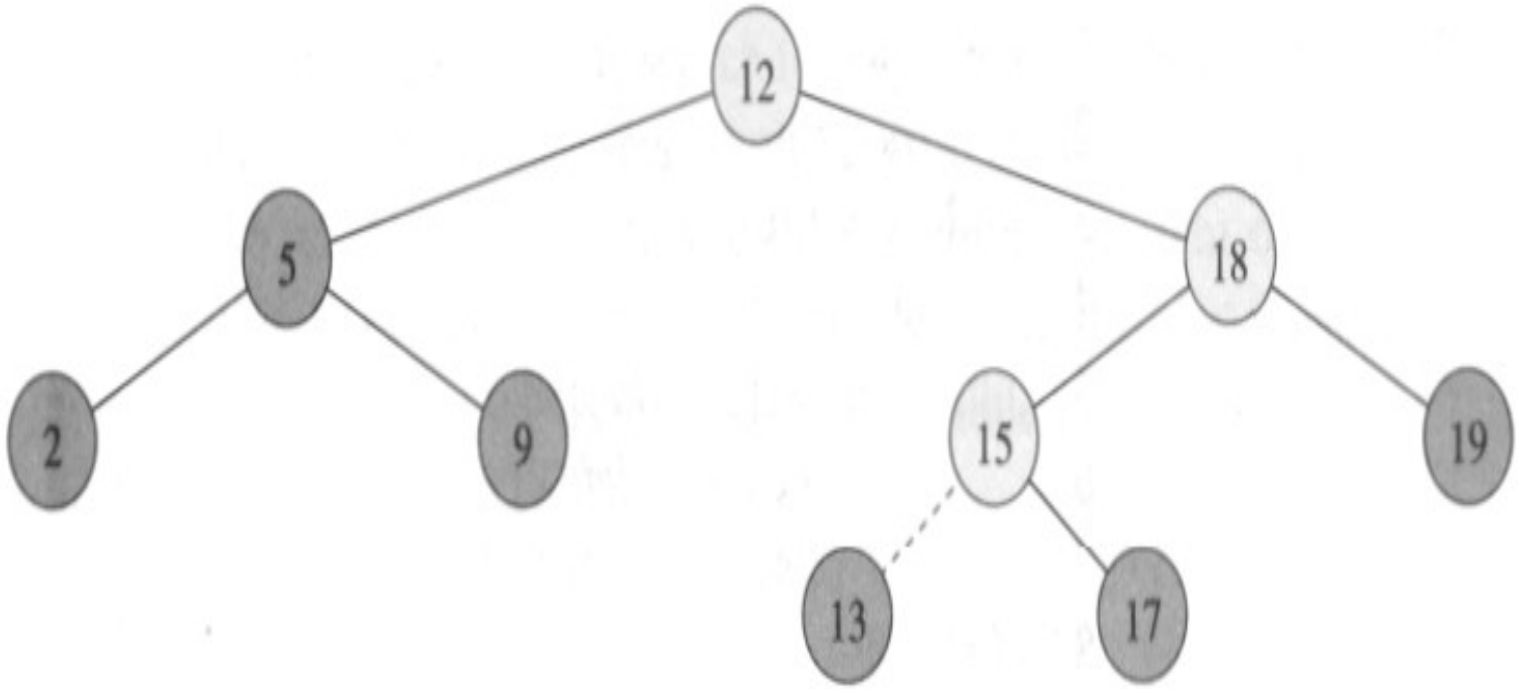
Inserir

- Para inserir um nodo z (chave[z]=v, e esq[z]=dir[z]=None) na árvore T, utiliza-se a função:

```
TREE-INSERT(T, z)
 1  y ← NIL
 2  x ← root[T]
 3  while x ≠ NIL
 4        do y ← x
 5           if key[z] < key[x]
 6              then x ← left[x]
 7              else x ← right[x]
 8  p[z] ← y
 9  if y = NIL
10     then root[T] ← z
11     else if key[z] < key[y]
12             then left[y] ← z
13             else right[y] ← z
```

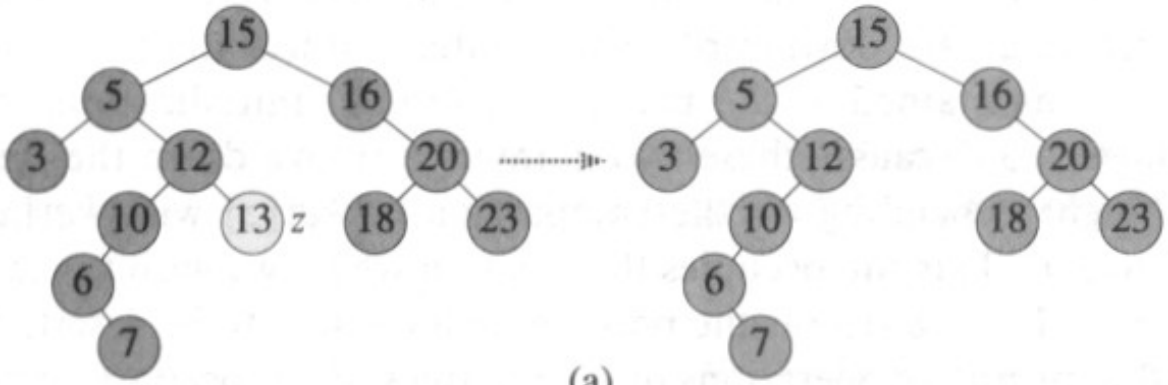
Exemplo de inserção

- Dado que desejamos inserir a chave 13

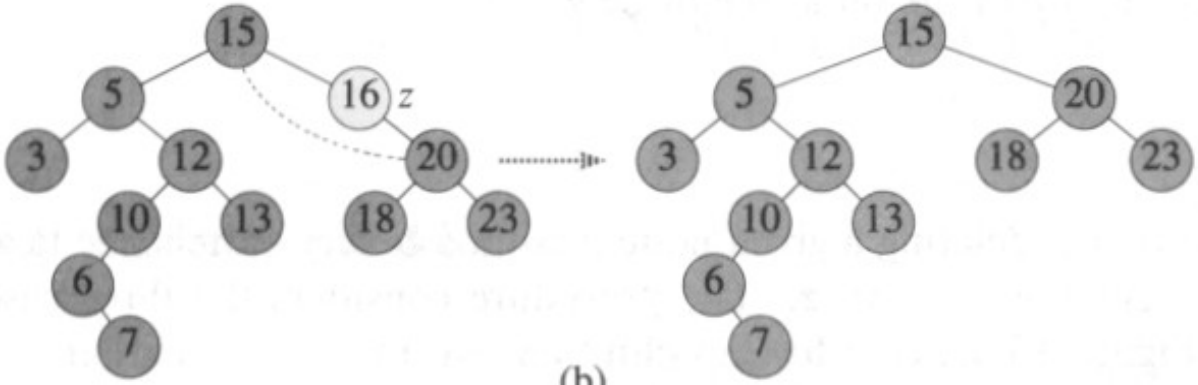


Deletar

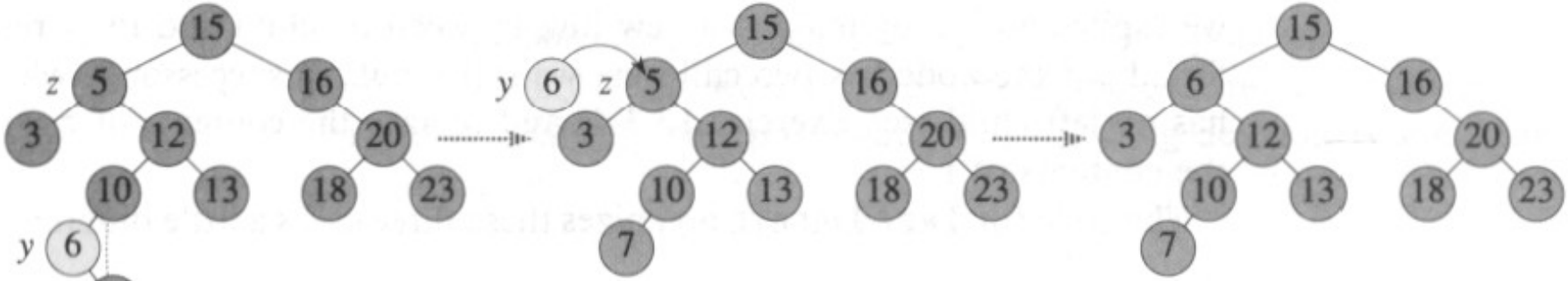
- Há três casos a considerar para deletar um nodo de uma árvore binária
  - Se o nodo não tem filhos
  - Se o nodo tem apenas um filho
  - Se o nodo tem dois filhos


15
5
16
3
12
20
10
13
z
18
23
6
7

(a)


15
5
16
z
3
12
20
10
13
18
23
6
7

(b)


15
z
5
16
3
12
20
10
13
18
23
y
6
7

(c)

Função Deletar

```
TREE-DELETE(T, z)
 1  if left[z] = NIL or right[z] = NIL
 2      then y ← z
 3      else y ← TREE-SUCCESSOR(z)
 4  if left[y] ≠ NIL
 5      then x ← left[y]
 6      else x ← right[y]
 7  if x ≠ NIL
 8      then p[x] ← p[y]
 9  if p[y] = NIL
10      then root[T] ← x
11      else if y = left[p[y]]
12              then left[p[y]] ← x
13              else right[p[y]] ← x
14  if y ≠ z
15      then key[z] ← key[y]
16           ▷ If y has other fields, copy them, too.
17  return y
```

Exercícios Práticos

- Exercício : Implementar uma classe árvore binária com as funções search, maximum, minimum, sucessor, predecessor, inserir e deletar.