Algoritmos e estrutura de dados

Árvores

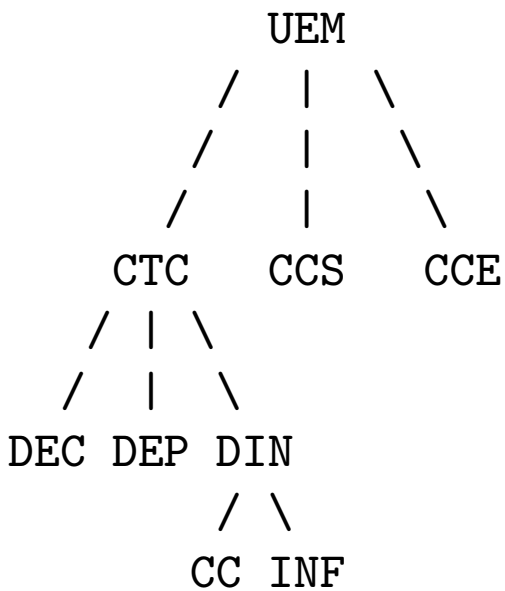
Marco A L Barbosa

Conteúdo

Árvores

Árvores

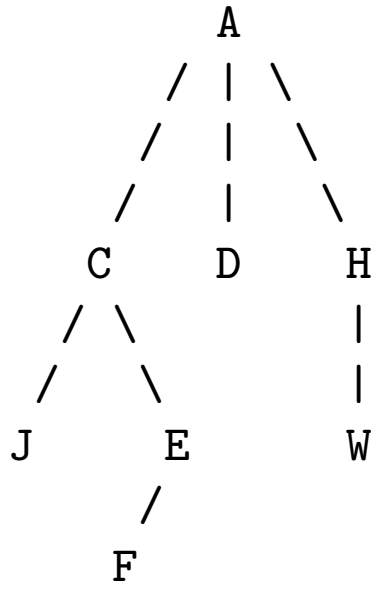
- Árvores são estrutura de dados utilizadas para representar relações hierárquicas
- Árvores são mais gerais do que listas encadeadas
- Uma **árvore** é
  - vazia; ou
  - um nó com um valor e uma lista de subárvores
- Exemplo de árvore (estrutura organizacional parcial da UEM)



Árvores

- Se x é uma árvore é y é uma subárvore de x, x é **pai** de y e y é **filho** de x
- Um nó x_n é **descendente** de um nó x_0 (e x_0 é **ancestral** de x_n) se existe nós $x_1, x_2, \cdots, x_n$ tal que x_i é pai de x_{i+1} para $0 \leq i \leq n-1$, ou seja, existe um caminho de x_0 para x_n
- A **raiz** de uma árvore é o ancestral de todos os nós dá árvore
- O **grau** de um nó é a quantidade de subárvores não vazias deste nó
- Um nó de grau zero é chamado de **folha** ou **terminal**
- Os nós que não são folhas são chamados de nós **internos**
- A **altura de um nó** x é a distância entre x e o seu descendente mais afastado. A altura de uma folha é 0
- A **altura de uma árvore** é a altura da raiz da árvore

Árvores

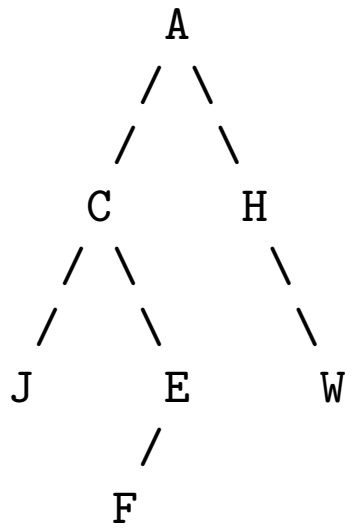


- A é a raiz da árvore
- A é pai de C e C é filho de A
- Os descentes de C são J, E, F
- A altura de C é 2 e de D é 0
- A altura da árvore é 3
- O grau de A é 3 e de W é 0
- J, F, D e W são folhas
- A, C, E e H são nós internos

Árvores binárias

Árvores binárias

- Uma árvore binária é um árvore em que cada nó tem no máximo dois filhos, um filho a esquerda e outro a direita
- Uma **árvore binária** é
  - vazia; ou
  - um nó com um valor, um subárvore binária a direita e uma subárvore binária a esquerda
- Exemplo



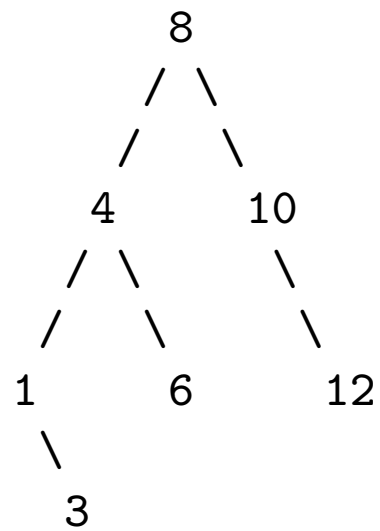
Árvores binárias de busca

Árvores binárias de busca

- Um **árvore binária de busca** é uma árvore binária em que cada nó x satisfaz a seguinte propriedade: o valor de x é
  - maior ou igual ao valor de qualquer nó na subárvore a esquerda de x
  - menor ou igual ao valor de qualquer nó na subárvore a direita de x

Árvores binárias de busca

- Exemplo



- Se substituirmos 6 por 9, a árvore deixa de satisfazer a propriedade, deixando assim de ser uma árvore binária de busca

Árvores binárias de busca

- Vamos implementar algoritmos de inserção, busca e remoção (entre outros)
- Vamos definir uma classe para representar um `No` e uma classe para representar uma árvore binária de busca

Árvores binárias de busca

```
class No(Struct):
    '''
    Representa um nó em uma árvore binária
        valor : Qualquer - representa o valor no nó
        esq : No - representa a subárvore a esquerda
        dir : No - representa a subárvore a direita
    '''
    def __init__(self, valor, esq, dir):
        self.init(valor, esq, dir)
```

Árvores binárias de busca

```
class ArvoreBB(Struct):
    '''
    Representa uma árvore binária de busca (ABB). Uma ABB é uma
    árvore com a seguinte propriedade:
        Seja x uma nó na ABB. Se y é um nó na subárvore a
        esquerda de x, então y.valor <= x.valor. Se y é uma
        nó na subárvore a direira de x, então y.valor >= x.valor
    Campos
        _raiz : No - representa a raiz da árvore
    '''
    def __init__(self):
        self.init()
        self._raiz = None
```

Árvores binárias de busca / Busca

- Buscar um elemento x na árvore
- Ideia
  - Se a raiz é `None`, então `x` não está na árvore
  - Se `x` é o valor armazenado na raiz, então `x` está na árvore
  - Se `x` é menor que o valor armazenado na raiz, faça a busca na subárvore a esquerda
  - Se `y` é maior que o valor armazenado na raiz, faça a busca na subárvore a direta
- Análise do tempo de execução
  - No pior caso, a busca será feita até a folha mais distante da raiz e o elemento não será encontrado
  - O quantidade de nós no caminho até a folha mais distante é 1 menos que a altura da árvore
  - Portanto, o tempo de execução é $O(h)$, onde h é a altura da árvore

Árvores binárias de busca / Busca

```
def _busca(self, v):
    '''
    ArvoreBB, Qualquer -> No
    Se v está na árvore, devolve o nó que contém v,
    caso contrário devolve None.
    Veja os exemplos de __contains__.
    '''
    p = self._raiz
    while p != None:
        if v == p.valor:
            return p
        elif v < p.valor:
            p = p.esq
        else:
            p = p.dir
    return None
```

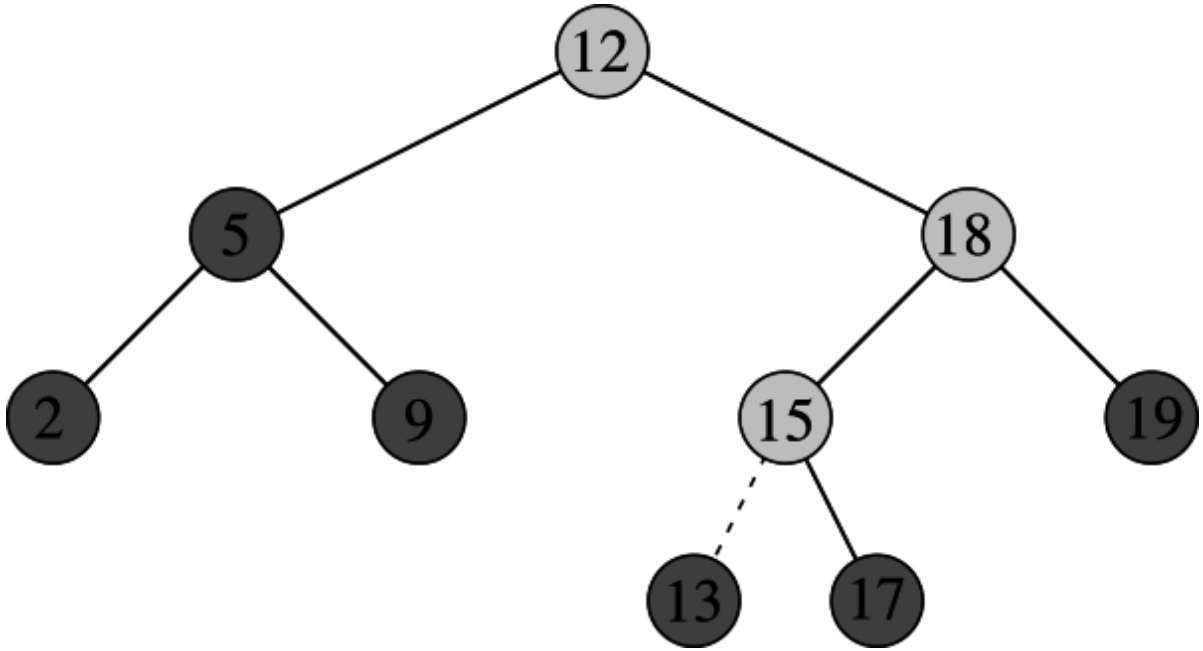
Árvores binárias de busca / Busca

```
def __contains__(self, v):
    '''
    ArvoreBB, Qualquer -> Boolean
    Devolve True se v está na árvore, False caso contrário.
    Exemplos: veja o arquivo arvores.py
    '''
    p = self._busca(v)
    return p != None
```

Árvores binárias de busca / Inserção

- Ideia
  - Procurar o lugar correto para o novo valor de forma a manter a propriedade de árvore binária de busca
- Análise do tempo de execução
  - No pior caso, a inserção será feita na folha mais distante da raiz
  - O quantidade de nós no caminho até a folha mais distante é 1 menos que a altura da árvore
  - Portanto, o tempo de execução é $O(h)$, onde h é a altura da árvore

Árvores binárias de busca / Inserção



- Os nós claros representam o caminho seguido até encontrar a posição onde o novo item deve ser inserido
- A linha tracejada indica a nova ligação criada na árvore

Árvores binárias de busca / Inserção

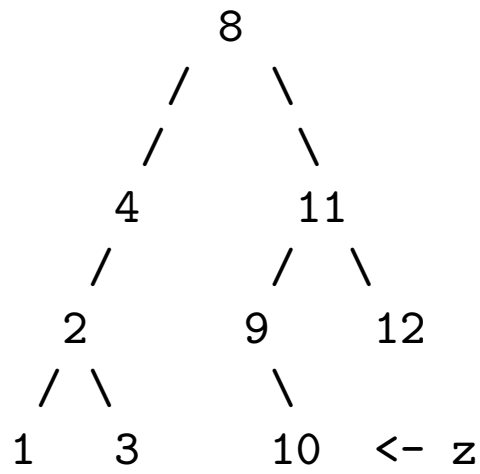
```
    def insere(self, v):
        '''
        ArvoreBB, Qualquer -> None
        Insere v na árvore.
        Exemplos: veja o arquivo arvores.py
        '''
        p = None
        # procura o local da inserção
        x = self._raiz
        while x != None:
            p = x
            if v < x.valor:
                x = x.esq
            else:
                x = x.dir
        t = No(v, None, None)
        if p == None: # estava vazia
            self._raiz = t
        elif v < p.valor:
            p.esq = t
        else:
            p.dir = t
```

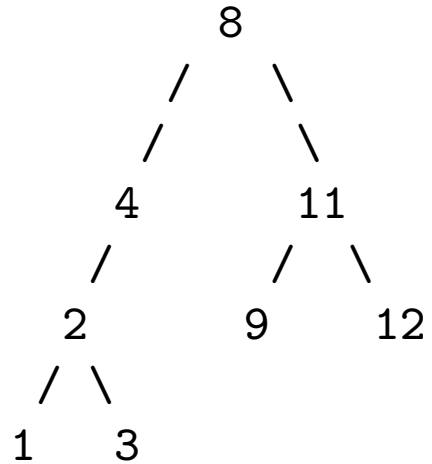
Árvores binárias de busca / Remoção

- Ideia da remoção de um nó z
  - Se z não tem filho, então apenas modifica-se o pai substituindo z por `None`
  - Se z tem apenas um filho, então elevar o filho para tomar o lugar de z alterando o pai de z para substituir z pelo seu filho
  - Se z tem dois filhos, então encontra-se o sucessor y de z, que está na subárvore a direita, e y toma a posição de x na árvore. O resto da subárvore a direita original de z torna-se a subárvore a direita de y e a subárvore a esquerda de z torna-se a subárvore a esquerda de y
    - Sucessor de um nó z é um nó y tal que o valor armazenado no nó y é o menor valor maior que o valor armazenado em z, ou seja, y é o nó com o próximo valor em ordem ordem crescente depois do valor de z
- Análise do tempo de execução
  - Semelhante a busca e inserção
  - O tempo de execução é $O(h)$, onde h é a altura da árvore

Árvores binárias de busca / Remoção

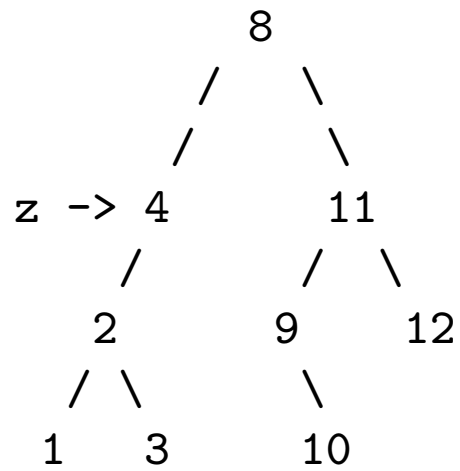
- Exemplo: remoção de um nó sem filhos

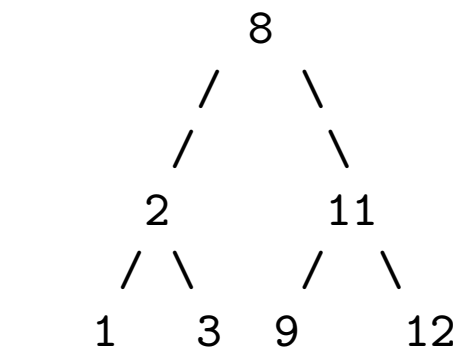




Árvores binárias de busca / Remoção

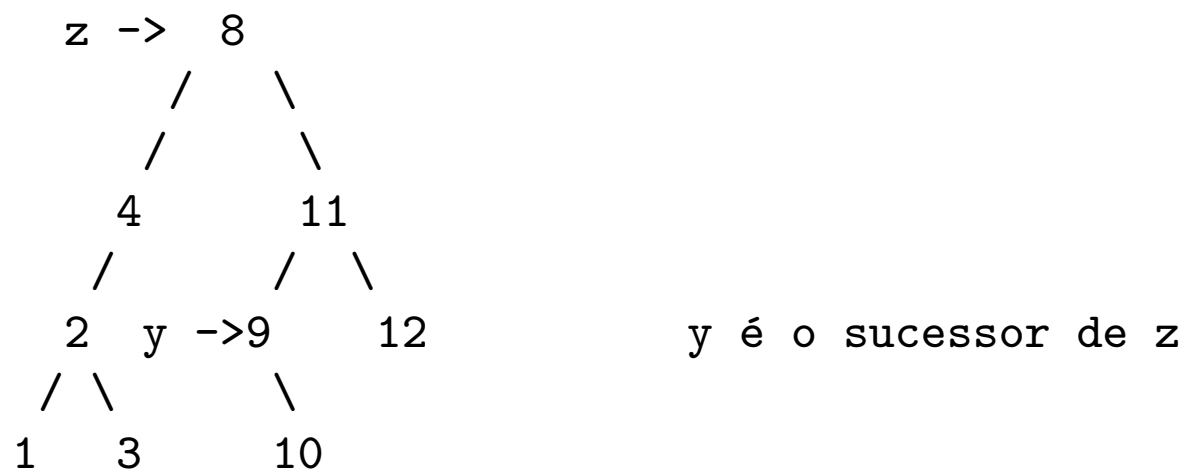
- Exemplo: remoção de um nó com 1 filho

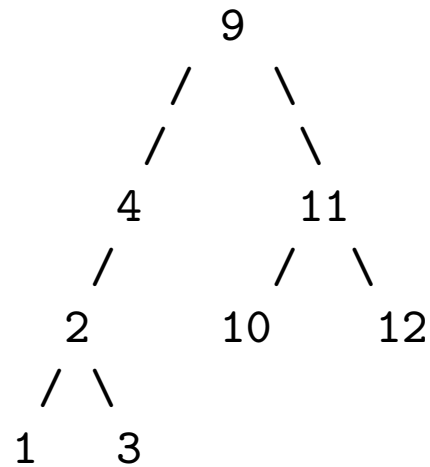




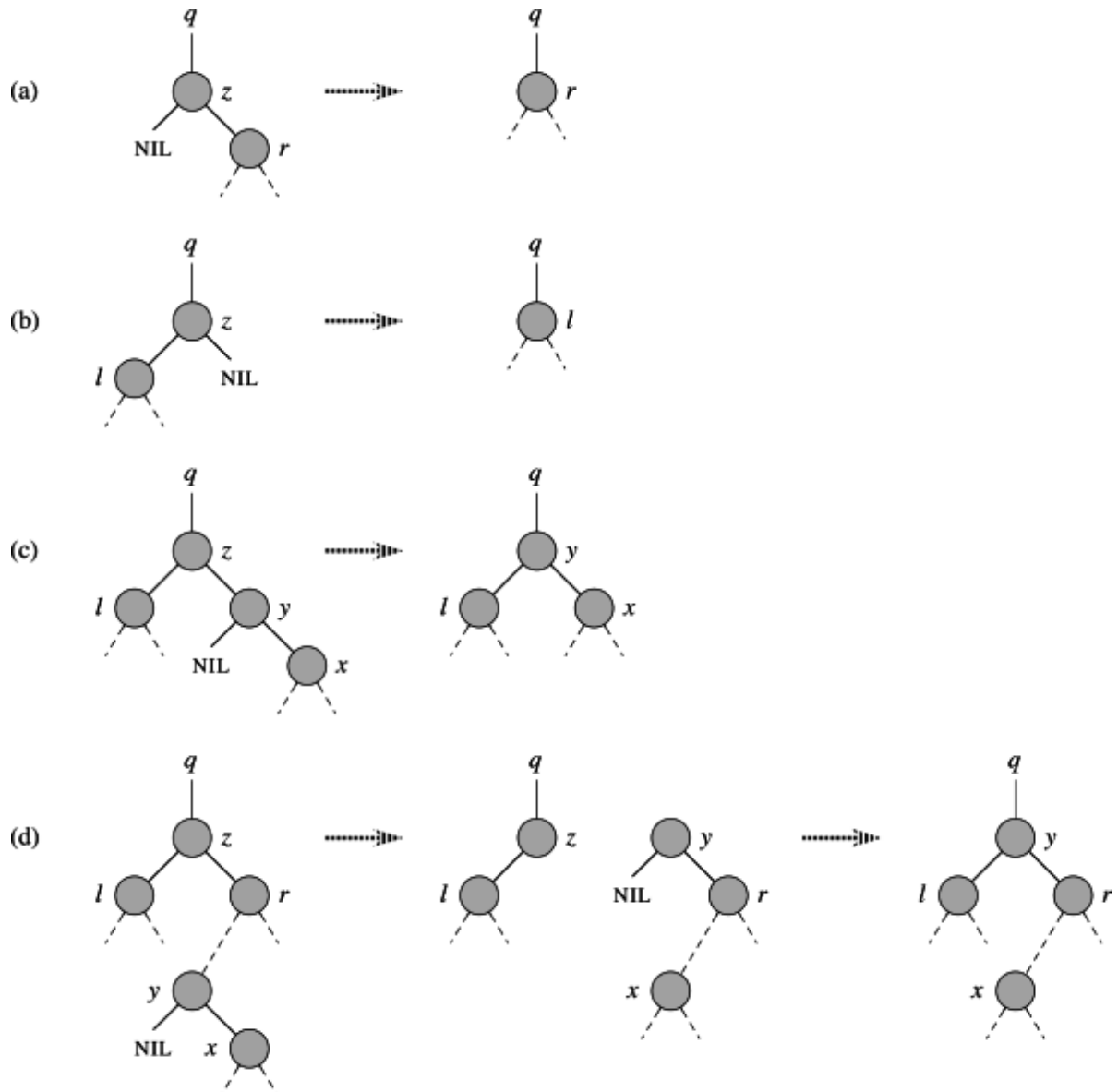
Árvores binárias de busca / Remoção

- Exemplo: remoção de um nó com 2 filhos





Árvores binárias de busca / Remoção



Árvores binárias de busca / Remoção

```
def _remove_no(self, z):
    '''
    ArvoreBB, No -> None
    Remove o nó z da árvore.
    Veja os exemplos de remove.
    '''
    if z.esq == None:
        self._transplanta(z, z.dir)
    elif z.dir == None:
        self._transplanta(z, z.esq)
    else:
        y = self._no_minimo(z.dir)
        p = self.pai(y)
        if p != z:
            self._transplanta(y, y.dir)
            y.dir = z.dir
        self._transplanta(z, y)
        y.esq = z.esq
```

Árvores binárias de busca / Remoção

```
def _transplanta(self, u, v):
    '''
    ArvoreBB, No, No -> None
    Altera o pai de u para ter como filho o v no lugar do u.
    Se u não tem pai, então v passa a ser a raiz da árvore.
    Veja os exemplos de remove.
    '''
    if self._raiz == u:
        self._raiz = v
    else:
        p = self.pai(u)
        if p.esq == u:
            p.esq = v
        else:
            p.dir = v
```

Tempo de execução

Tempo de execução

- Uma árvore binária é **balanceada** se, em cada um de seus nós, as subárvores esquerda e direita tem aproximadamente a mesma altura
- Uma árvore binária balanceada com n nós tem altura próxima de $\log_2 n$
- Portanto, em uma árvore binária de busca balanceada, o tempo de execução das operações de busca, inserção e remoção é $O(\log_2 n)$
- A altura máxima de uma árvore binária com n nós é $n - 1$ (se a árvore for como uma lista)
- Portanto, em uma árvore binária de busca no pior caso o tempo de execução das operações de busca, inserção e remoção é $O(n)$
- Desta forma é necessário manter a árvore balanceada após inserção e remoção para manter o tempo de execução das operações em $O(\log_2 n)$

Árvores AVL

Árvores AVL

- Uma **árvore AVL** é uma árvore binária de busca auto-balanceada
- A altura de cada subárvore de um nó qualquer difere de no máximo 1
- As operações de balanceamento, busca, inserção e remoção tem tempo de execução $O(\log_2 n)$

Referências

- Projeto de algoritmos / Árvores binárias. Paulo Feofiloff.
- Projeto de algoritmos / Árvores binárias de busca. Paulo Feofiloff.
- Algoritmos: Teoria e prática. 3ª edição. Cormen, Thomas H at all.
- Algoritmos: Teoria e prática. 2ª edição. Cormen, Thomas H at all.