ENGENHARIA INFORMÁTICA E DE COMPUTADORES

Algoritmos e Estruturas de Dados

(parte 14 – Árvores Binárias)

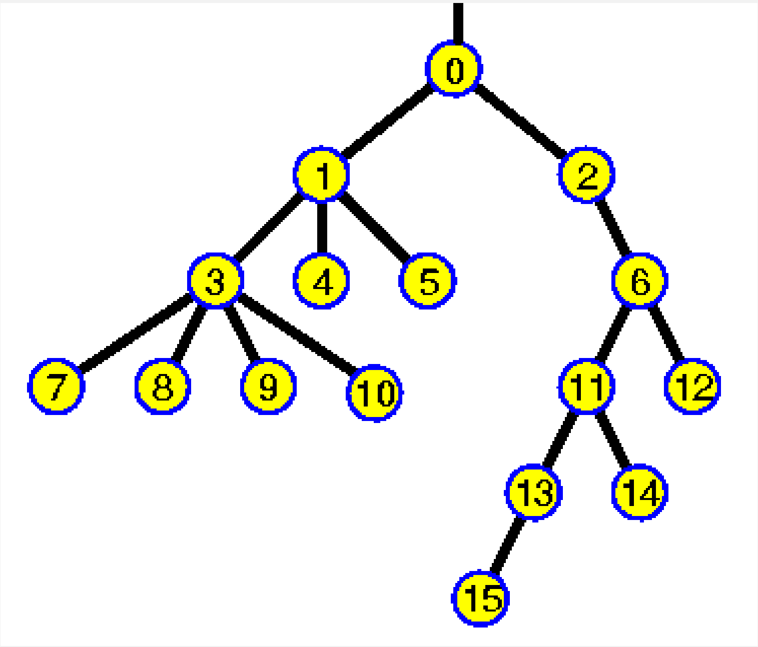
2º Semestre 2022/2023

Instituto Superior de Engenharia de Lisboa

Paula Graça

ÁRVORES

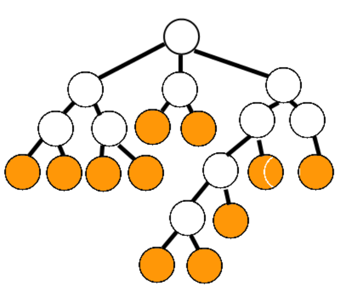
- Árvore Ordenada (N-ária)
  - É uma árvore em que para cada nó os filhos estão ordenados, ou seja, existe uma relação de ordem dos filhos (existe o 1° filho, o 2°, etc., até ao último)
  - Se cada nó não tiver mais que N filhos, então trata-se de uma árvore N-ária (ou de grau N)



DEFINIÇÕES

A árvore tem 16 nós
A árvore tem grau 4
A árvore tem altura 5
O nó 0 é a raiz
O nó 1 é intermédio
O nó 4 é folha
O nó 4 é filho do nó 1
O nó 1 é pai do nó 4
Os nós 3, 4 e 5 são irmãos

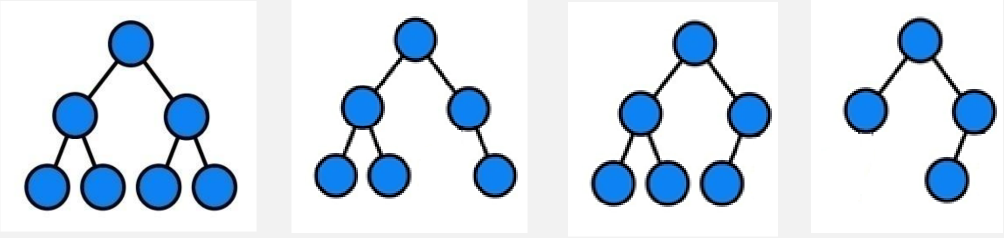
ÁRVORES



1 nó raíz
9 nós não-terminais
11 folhas

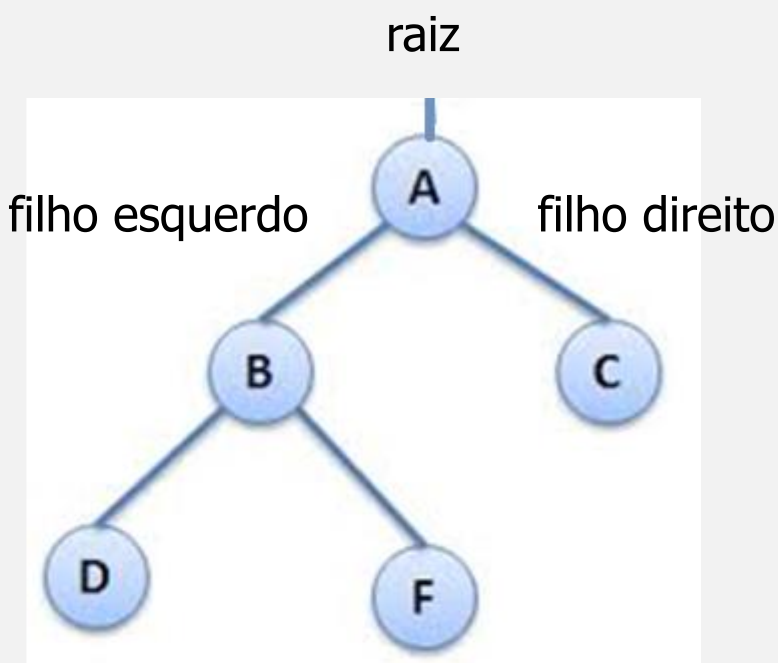
- Definições
  - A raiz (*root*) é o único nó sem ascendente
  - Nós sem filhos são designados por terminais ou folhas (*leaves*)
  - Nós com filhos (com exceção da raiz) são designados não-terminais

- Definição
  - A árvore é de tipo N-ária, sse cada nó não tiver mais que N filhos
  - Quando N=2, a árvore diz-se binária



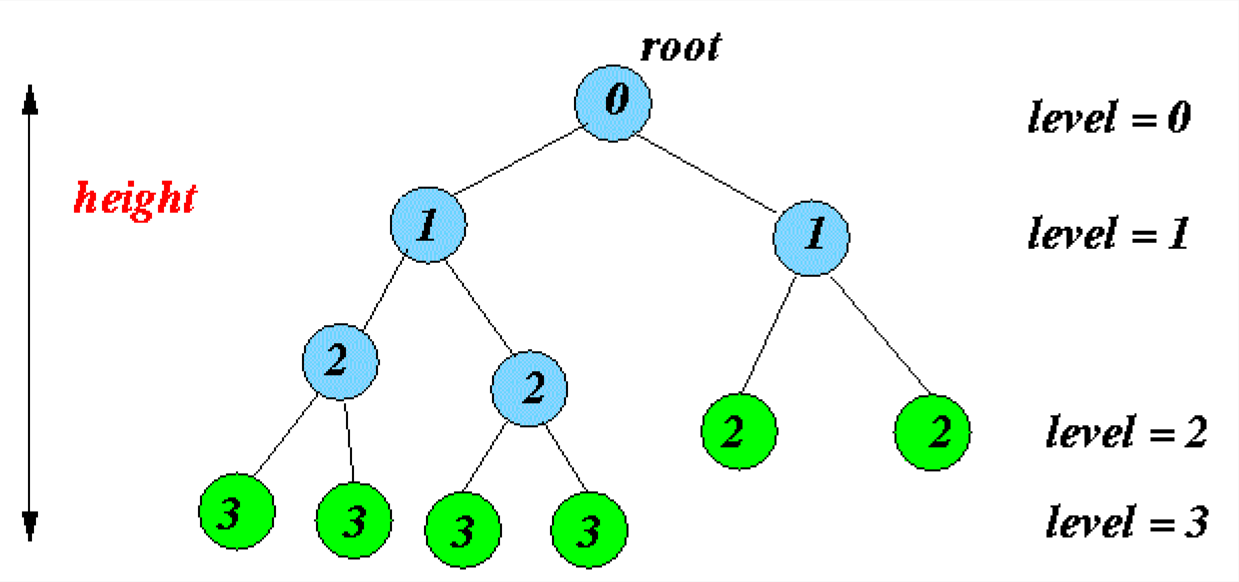
ÁRVORES

- Árvore Binária Ordenada
  - A árvore binária é um caso particular da árvore N-ária, ou seja, é uma árvore de grau 2: cada nó não poder ter mais que 2 filhos
  - Uma vez que os filhos de cada nó são ordenados, são referidos como filho esquerdo e filho direito



ÁRVORES BINÁRIAS

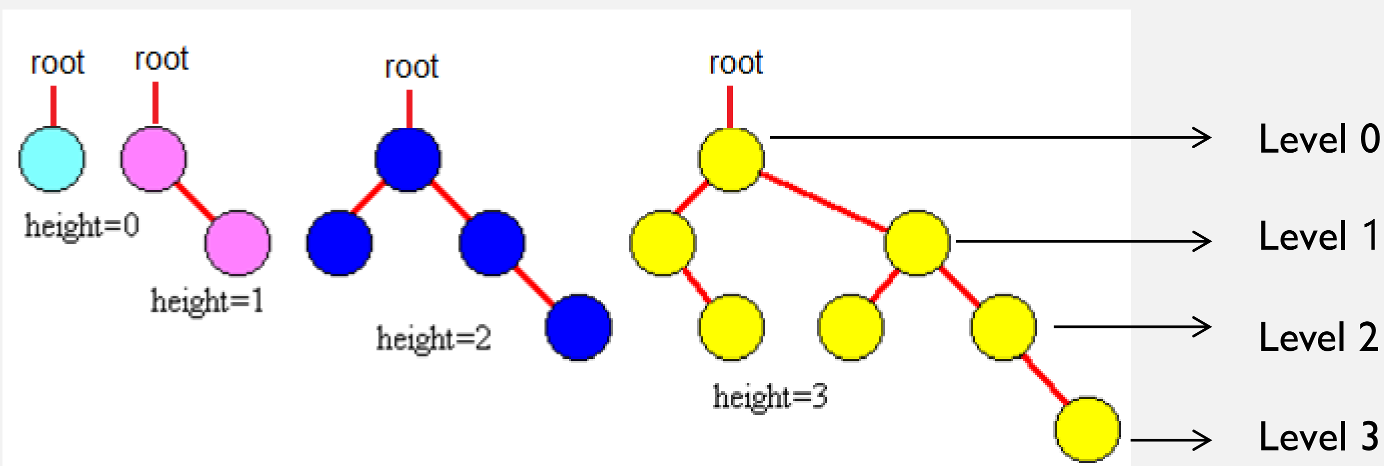
- Nível e Altura
  - O nível (*level*) de um nó é o número de arestas entre a raiz e o nó (a raiz tem nível 0)
  - A altura (*heigh*) de uma árvore é o makor dos níveis das folhas (uma árvore só com a raiz tem altura 0)



ÁRVORES BINÁRIAS

- Nível, Altura e Grau
  - A raiz tem nível de profundidade 0
  - A altura de uma árvore é o maior nível de profundidade das folhas
  - O grau de uma árvore é o número de filhos diretos suportados pelos seus nós

    (os filhos não preenchidos estão a *null*)

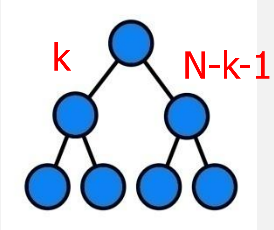


ÁRVORES BINÁRIAS - PROPRIEDADES

- Teorema:
  - Uma árvore binária com N nós não folhas possui N+1 folhas, desde que a árvore esteja completa

- Estratégia de prova:
  - Se N=1, a árvore possui um único nó (raiz e folha em simultâneo), i.e. 1 folha
  - Seja N>0 o número de nós não folhas:
    - k nós não-terminais na subárvore esquerda
    - N-k-1 (raiz) nós não-terminais na subárvore direita

N = 3 nós não folhas



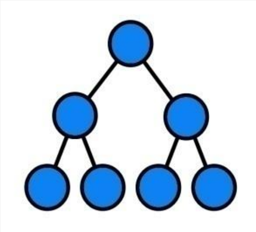
- Prova:
  - A subárvore esquerda tem k+1 folhas e a subárvore direita tem (N-k-1)+1=N-k folhas
  - Somando, a árvore tem (k+1)+(N-k)=N+1 folhas

ÁRVORES BINÁRIAS - PROPRIEDADES

- Teorema:
  - Uma árvore binária com N nós não folhas possui 2*N arestas (N-1 para os nós não-terminais e N+1 para as folhas), desde que a árvore esteja completa

- Estratégia de prova:
  - Excetuando a raiz, cada nó possui um único ascendente, pelo que só há uma aresta entre um nó e o seu ascendente
  - Há N+1 arestas para as folhas
  - Há N -1 arestas para os nós não folhas

N=3 nós não folhas



N-1=2 arestas dos nós não terminais

N+1=4 arestas das folhas

- Prova:
  - Somando, a árvore tem (N+1)+(N-1)=2*N arestas

2*N=2*3=6 arestas

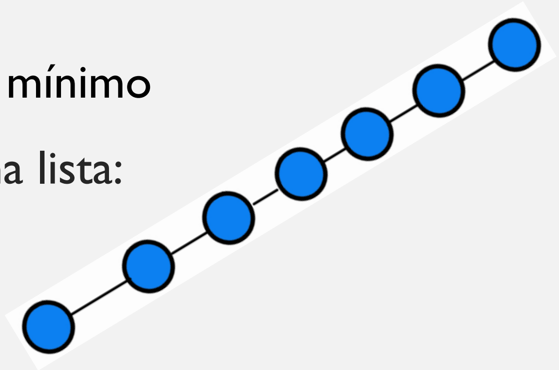
ÁRVORES BINÁRIAS - PROPRIEDADES

- Teorema:
  - Numa árvore binária com N nós, o nível das folhas varia entre $\lfloor \log_2 N \rfloor$ e N-1

- Estratégia de prova: identificar níveis máximo e mínimo
  - O nível máximo é o da árvore degenerada numa lista:

    nível N-1

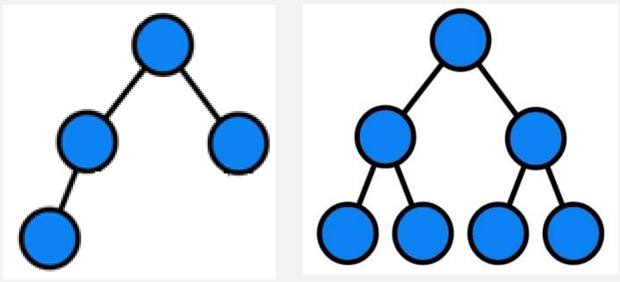
    $h(N=7) = 6$



  - O nível mínimo é o da árvore balanceada: Para todo o nó, a altura da sua sub-árvore esquerda e direita difere no máximo de 1

    $h = \log_2 N$

    $h(N=4) = 2$
    $h(N=7) = 2$

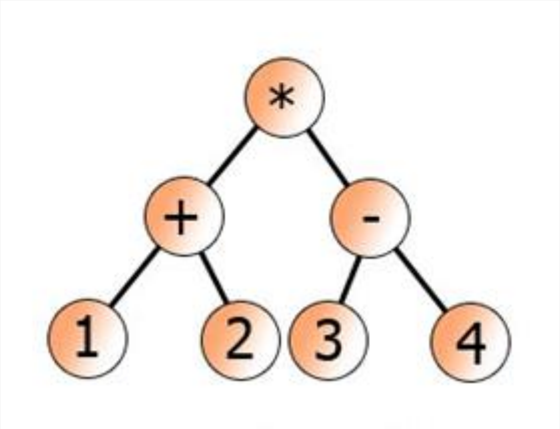


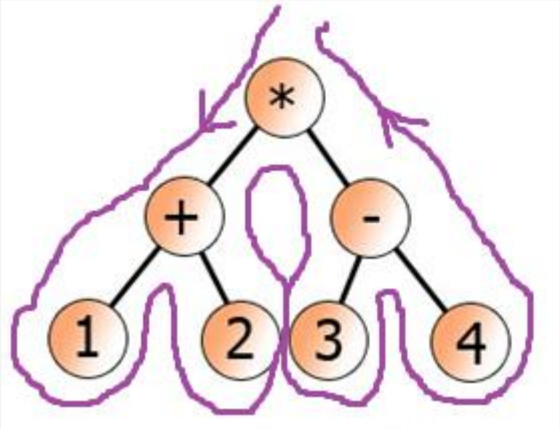
ÁRVORES BINÁRIAS - VARRIMENTO

- Existem diversas estratégias de percurso/varrimento (*transverse*) de árvores
- <u>Em profundidade</u>
  - Prefixo (*preorder* ou *deep-first*): obtém o valor do nó antes do varrimento das suas subárvores
  - Infixo (*inorder*): varre primeiro a subárvore esquerda, obtém o valor do nó e varre depois a subárvore direita
  - Sufixo (*postorder*): obtém o valor do nó depois do varrimento das suas subárvores
- <u>Em largura:</u>
  - Largura (*breadth-first*) – obtém o valor dos nós por níveis

ÁRVORES BINÁRIAS – VARRIMENTO EM PROFUNDIDADE

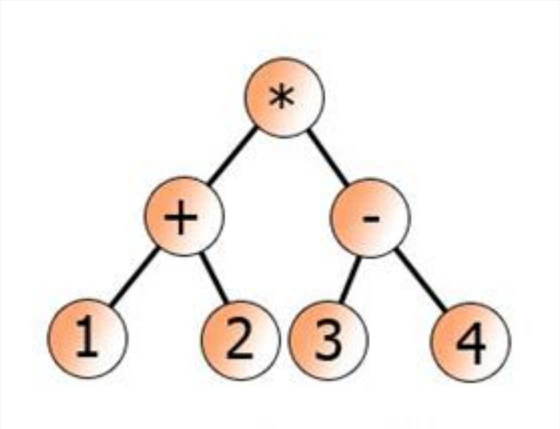
- Exemplo:

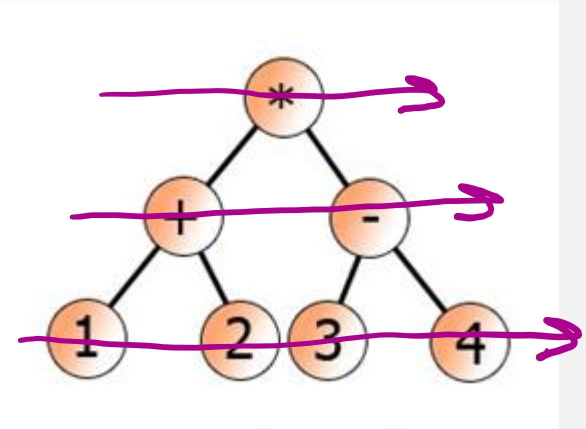




- Prefixo: * + 1 2 – 3 4
- Infixo: 1 + 2 * 3 – 4 (representação da expressão)
- Sufixo: 1 2 + 3 4 – * (ordem de cálculo)

- Exemplo:





- Largura: * + – 1 2 3 4 (varrimento por níveis)

ÁRVORES BINÁRIAS – VARRIMENTO EM PROFUNDIDADE

```
Preorder-Tree-Walk(root)
    if root ≠ NULL
        print root.value
        Preorder-Tree-Walk(root.left)
        Preorder-Tree-Walk(root.right)
```

```
Inorder-Tree-Walk(root)
    if root ≠ NULL
        Inorder-Tree-Walk(root.left)
        print root.value
        Inorder-Tree-Walk(root.right)
```

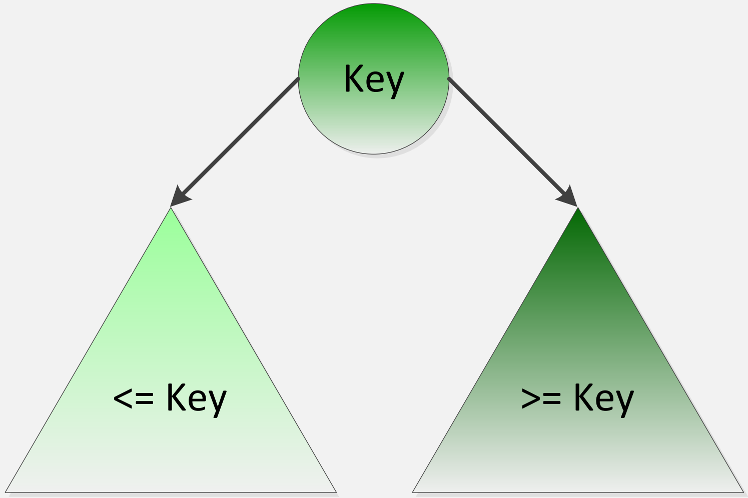
```
Postorder-Tree-Walk(root)
    if root ≠ NULL
        Postorder-Tree-Walk(root.left)
        Postorder-Tree-Walk(root.right)
        print root.value
```

ÁRVORES BINÁRIAS – VARRIMENTO EM LARGURA

```
Levels-Tree-Walk(root)
    Q = ∅    // queue inicializada a vazio
    if root ≠ NULL
        Enqueue(Q, root)
        while Q ≠ ∅
            root = Dequeue(Q)
            print root.value
            if root.left ≠ NULL
                Enqueue(Q, root.left)
            if root.right ≠ NULL
                Enqueue(Q, root.right)
```

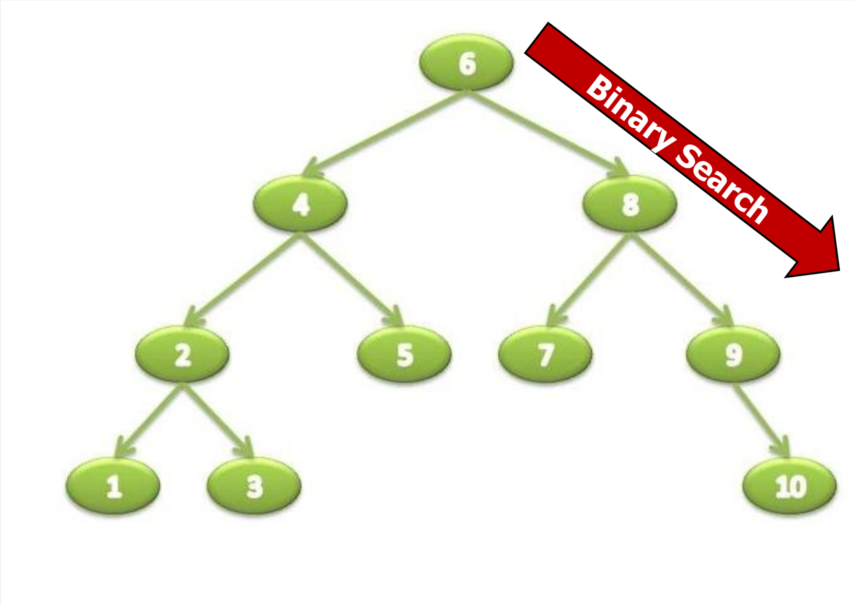
ÁRVORES BINÁRIAS DE PESQUISA

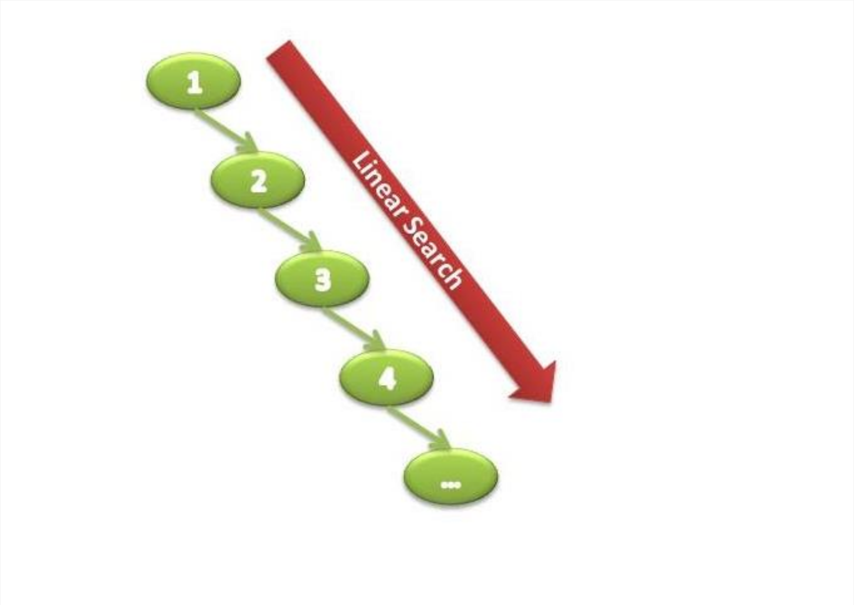
- Uma Árvore Binária de Pesquisa (BST – *Binary Search Tree*) é uma árvore binária que obedece ao seguinte critério de ordenação
  - Para todo o nó, as chaves da sua subárvore esquerda são menores ou iguais à sua chave
  - Para todo o nó, as chaves da sua subárvore direita são maiores ou iguais à sua chave



ÁRVORES BINÁRIAS DE PESQUISA

- Vantagens
  - Pesquisa mais rápida quando a árvores está balanceada: T(N) = *O* (h) = *O* (lg N)
- Desvantagens
  - A árvore pode degenerar em lista: T(N) = *O* (h) = *O* (N)





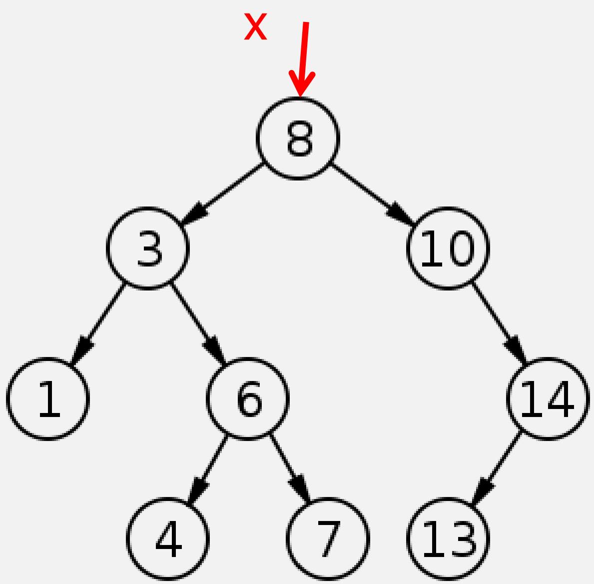
ÁRVORE BINÁRIA DE PESQUISA - ESTRUTURA

- Uma árvore binária de pesquisa é representada por uma estrutura ligada em que cada nó é um objecto
  - T.root aponta para a raiz da árvore T
  - Cada nó é composto por
    - value (chave de ordenação e eventuais dados associados)
    - left (aponta para o filho esquerdo)
    - right (aponta para o filho direito)
    - p (aponta para o pai) T.root.p ==NULL
  - As chaves obedecem à seguinte propriedade
    - Se y está na subárvore esquerda de x, então

      y.value $\leq$ x.value

ÁRVORE BINÁRIA DE PESQUISA - OPERAÇÕES

- O percurso infixo obtém as chaves com a seguinte ordem:



1 3 4 6 7 8 10 13 14

```
Inorder-Tree-Walk(x)
    if x ≠ NULL
        Inorder-Tree-Walk(x.left)
        print x.value
        Inorder-Tree-Walk(x.right)
```

ÁRVORE BINÁRIA DE PESQUISA - ANÁLISE

- O tempo de execução para percorrer uma árvore binária é dado pela seguinte recorrência (assumindo que está balanceada)

$$T(n) = 2T\left(\frac{n}{2}\right) + O(1)$$

  - Sendo $O(1)$ – custo do acesso à raiz
  - $2T(n/2)$ – custo do acesso às duas subárvores

- Resolvendo a recorrência através do teorema mestre

$$T(n) = 2T\left(\frac{n}{2}\right) + 1$$

$$a = 2; b = 2;\ f(n) = 1 \quad = n^0$$

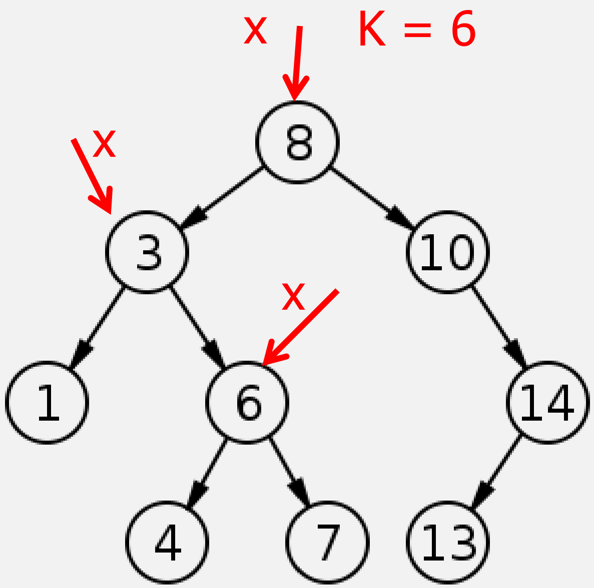
$$f(n) = O(n^{\log_b^{a-\epsilon}}) = O(n^{\log_2^{2-\epsilon}}) = O(n^{1-\epsilon})$$ Teorema mestre caso 1

$$f(n) = 1\,, \qquad para\ \varepsilon = 1$$

$$T(n) = \Theta(n)$$

- Dado o apontador para a raiz da árvore e uma chave *k*, Tree-Search devolve um apontador para o nó com essa chave, se existir, ou NIL caso contrário

```
Iterative-Tree-Search(x, k)
     while x ≠ NULL and k ≠ x.value
          if k < x.value
               x = x.left
          else x = x.right
     return x
```



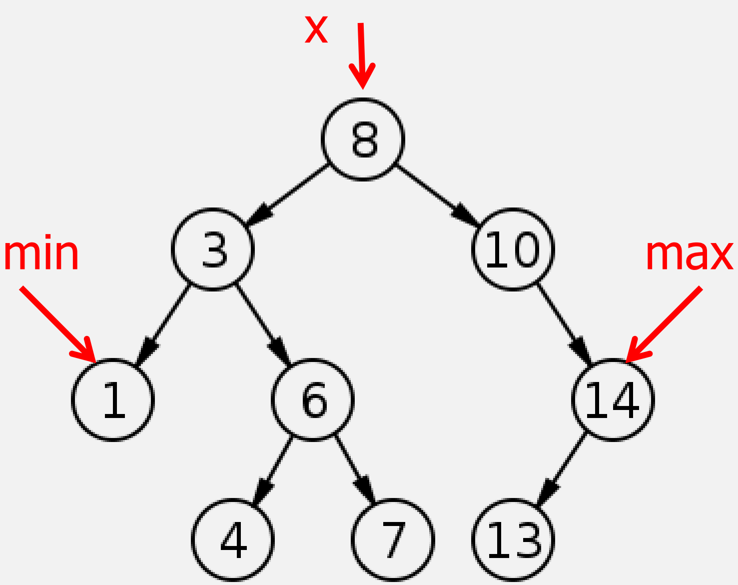
T(n) = O(h)

ÁRVORE BINÁRIA DE PESQUISA - OPERAÇÕES

- Tree-Minimum e Tree-Maximum, devolvem respectivamente, o nó mais à esquerda (menor chave) e o nó mais à direita da árvore (maior chave)

```
Tree-Minimum(x)
    while x.left ≠ NULL
        x = x.left
    return x
```

```
Tree-Maximum(x)
    while x.right ≠ NULL
        x = x.right
    return x
```



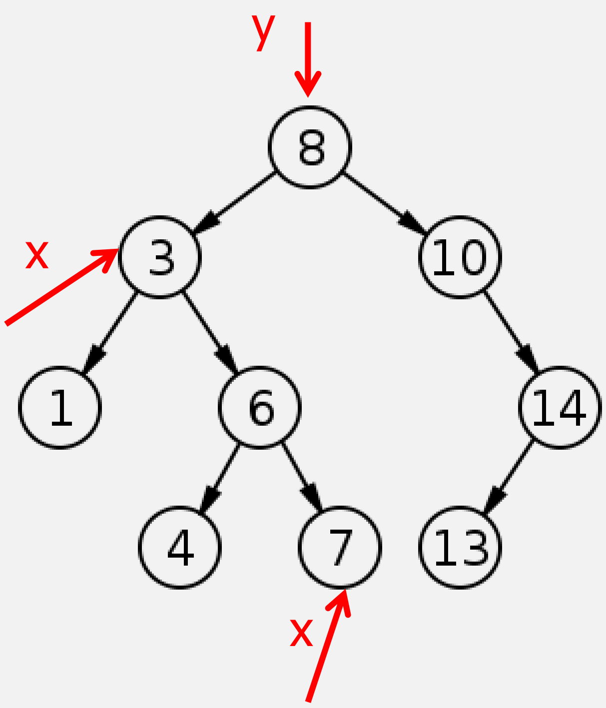
$T(n) = O(h)$

Os algoritmos assumem que a árvore não está vazia

ÁRVORE BINÁRIA DE PESQUISA - OPERAÇÕES

- Dado um nó numa árvore binária de pesquisa, Tree-Successor devolve o nó por ordem do percurso infixo (chave de valor seguinte se não houver repetições)

```
Tree-Successor(x)
    if x.right ≠ NULL
        return Tree-Minimum(x.right)
    y = x.p
    while y ≠ NULL and x == y.right
        x = y
        y = y.p
    return y
```

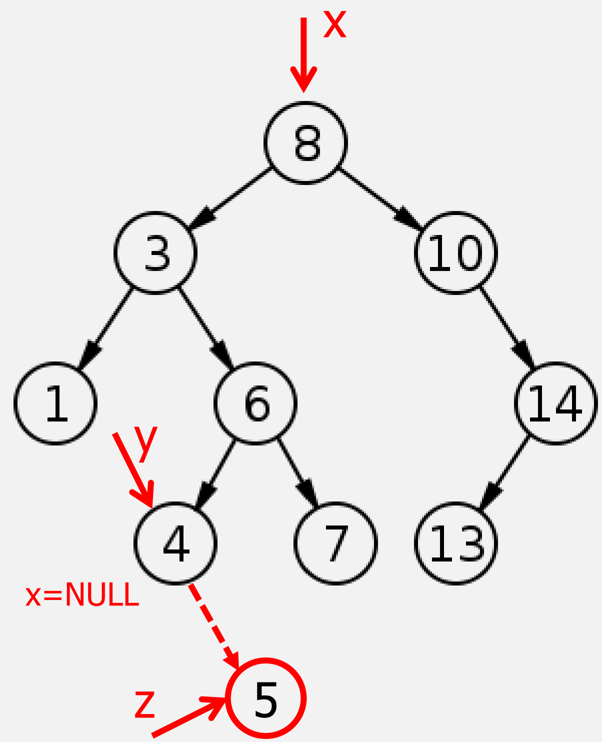


$T(n) = O(h)$

ÁRVORE BINÁRIA DE PESQUISA - OPERAÇÕES

- Tree-Add insere um novo nó z na raiz, ou como folha na subárvore esquerda ou subárvore direita de y, após procurar o nó onde inserir de acordo com a ordenação

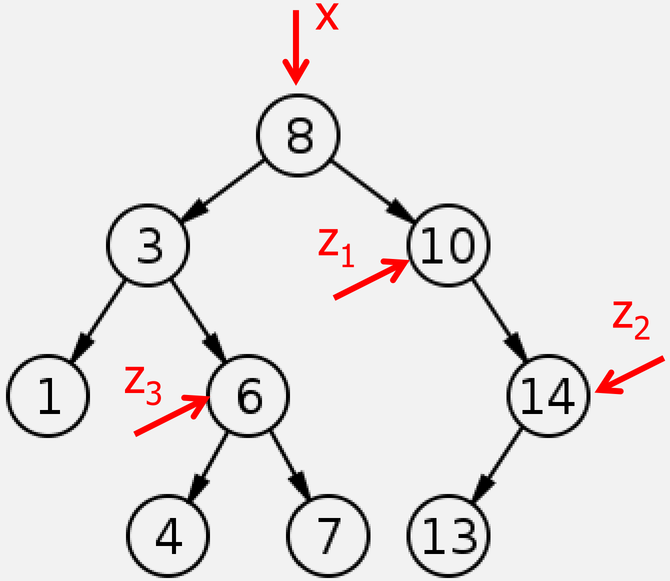
```
Tree-Add(T, z)
    y = NULL      // y = parent of x
    x = T.root
    while x ≠ NULL
        y = x
        if z.value <= x.value
            x = x.left
        else x = x.right
    z.p = y
    if y == NULL // tree T was empty
        T.root = z
    else if z.value <= y.value
        y.left = z
    else y.right = z
```



$T(n) = O(h)$

ÁRVORE BINÁRIA DE PESQUISA - OPERAÇÕES

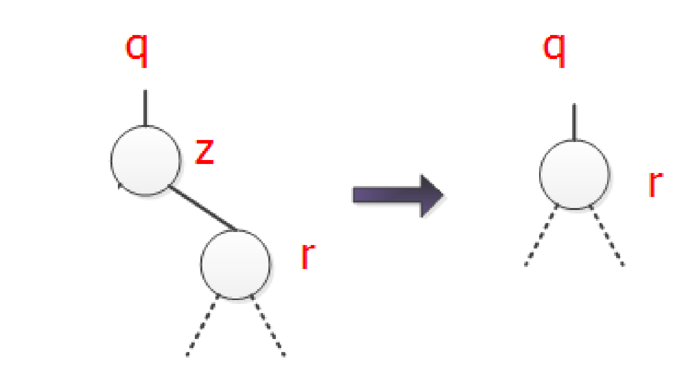
- Tree-Remove remove o nó z, após procurar a sua posição a partir da raiz de acordo com a ordenação
- São considerados três casos
    1. O nó **z** não tem filho esquerdo
    2. O nó **z** não tem filho direito
    3. O nó **z** tem filho esquerdo e direito



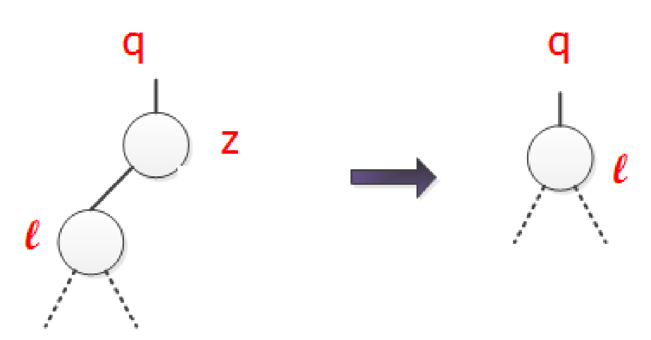
ÁRVORE BINÁRIA DE PESQUISA - OPERAÇÕES

- A remoção de um nó **z** numa árvore binária de pesquisa processa-se da seguinte forma:

  1. Se z não tem filho esquerdo, então substitui-se z pelo seu filho direito, que poderá ou não ser NULL



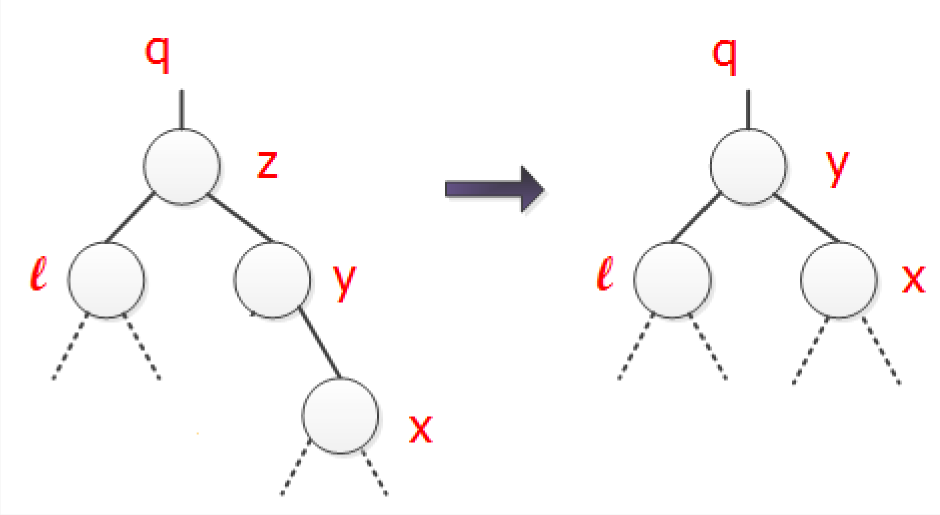
  2. Se z não tem filho direito, então substitui-se z pelo seu filho esquerdo, que poderá ou não ser NULL



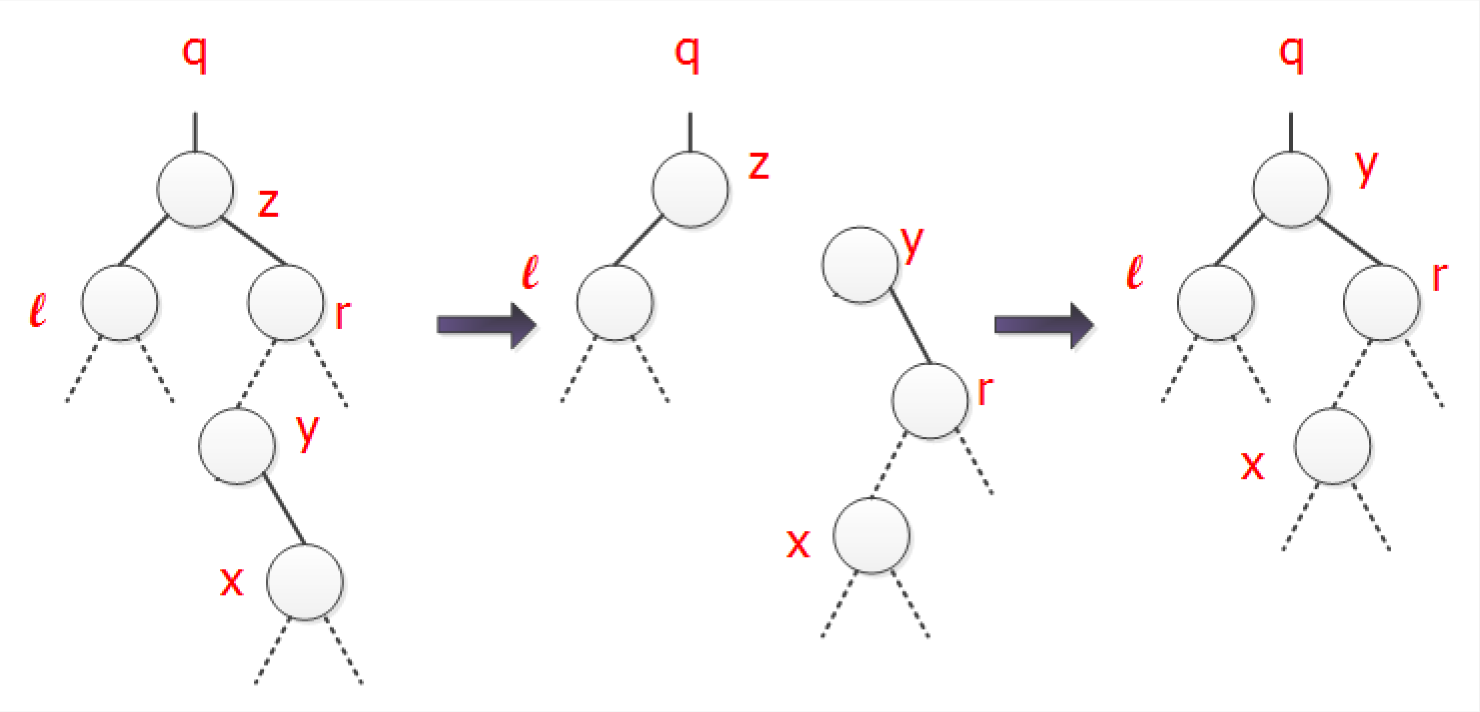
ÁRVORE BINÁRIA DE PESQUISA - OPERAÇÕES

3. Se z tem filho esquerdo e direito, procura-se o seu sucessor y (nó mais à esquerda da sua subárvore direita), o qual não tem filho esquerdo.

- Pretende-se que y seja retirado substituindo z na árvore
  - Se y é o filho direito de z, substitui-se z por y



- Se y não é o filho direito de z, então
  - Substitui-se y pelo seu filho direito e liga-se y à subárvore direita de z, e
  - Depois substitui-se z por y



ÁRVORE BINÁRIA DE PESQUISA - OPERAÇÕES

- Algoritmo Tree-Remove

```
Tree-Remove(T, z)
    if z.left == NULL
        Transplant(T, z, z.right)
    else if z.right == NULL
        Transplant(T, z, z.left)
    else
        y = Tree-Minimum(z.right)
        if y.p ≠ z
            Transplant(T, y, y.right)
            y.right = z.right
            y.right.p = y
        Transplant(T, z, y)
        y.left = z.left
        y.left.p = y
```

$T(n) = O(h)$

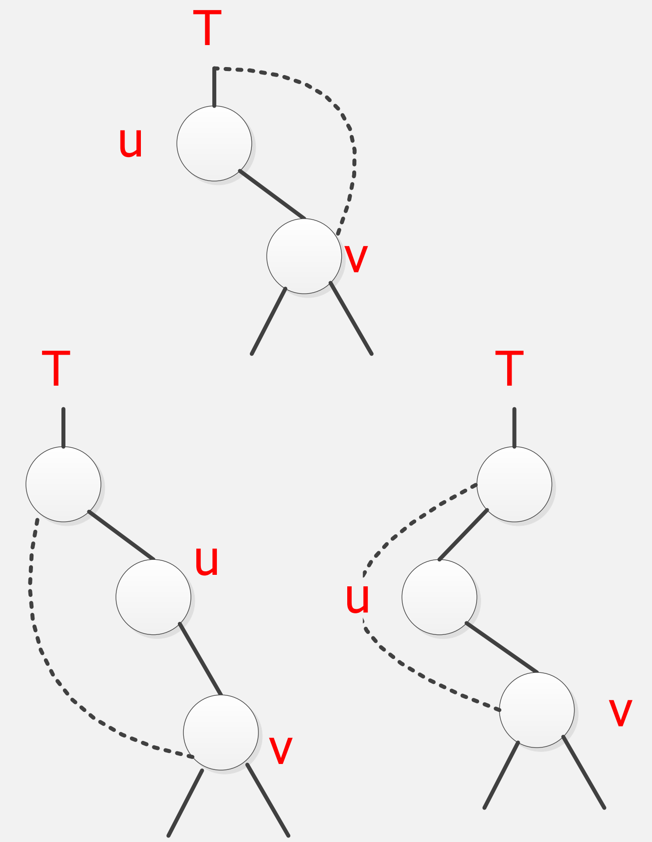
ÁRVORE BINÁRIA DE PESQUISA - OPERAÇÕES

- Algoritmo Transplant
  - Substitui a subárvore u, dado o apontador para o seu pai, pela subárvore v

(está ilustrado o caso de u não ter filho

esquerdo, mas o algoritmo funciona igual

no caso de u não ter filho direito)




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