Árvores binárias de pesquisa

Fernando Lobo

Algoritmos e Estrutura de Dados

— Algumas figuras retiradas do livro Introduction to Algorithms, 3rd Edition. —

Árvores binárias de pesquisa

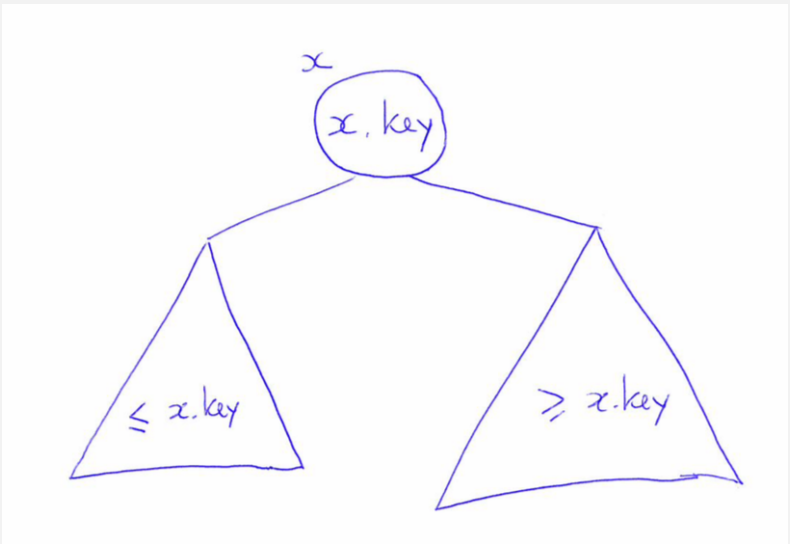
- Permite realizar operações sobre conjuntos dinâmicos em tempo $O(h)$, sendo h a altura da árvore.
  - *Search*, *Insert*, *Delete*, *Minimum*, *Maximum*, *Successor*, *Predecessor*.
- Se a árvore for equilibrada, $h = O(\lg n)$
- No pior caso, $h = O(n)$

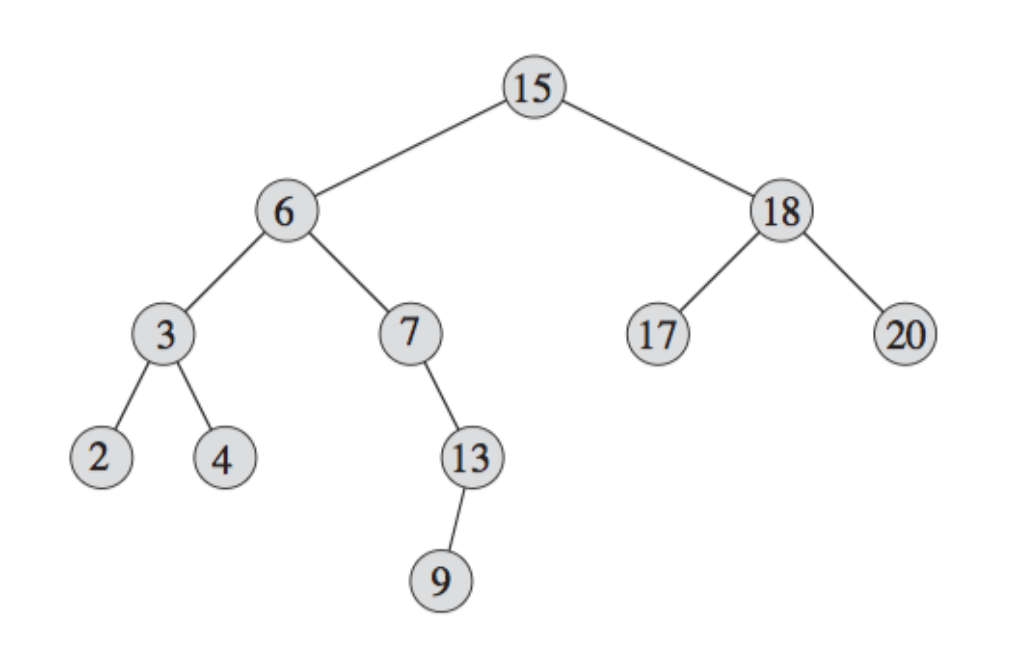
Representação

- Coleção de nós ligados de forma especial.
- Cada nó é um objecto que contém:
  - **key**: elemento do conjunto dinâmico.
  - **left**: apontador para o filho esquerdo.
  - **right**: apontador para o filho direito.
  - **p**: apontador para o pai.
- Se T é uma árvore, $T.root$ aponta para a raíz.
- A raíz é o único nó da árvore que não tem pai: $T.root.p = \text{NIL}$

Propriedade

- Se y está na sub-árvore esquerda de x, então $y.key \leq x.key$
- Se y está na sub-árvore direita de x, então $y.key \geq x.key$





Percursos

- Podemos percorrer os nós da árvore de forma sistemática.
- 3 modos usuais:
  - **inorder**: visita a sub-árvore esquerda, depois visita a raíz, e depois visita a sub-árvore direita.
    - ★ no exemplo anterior: 2 3 4 6 7 9 13 15 17 18 20
  - **preorder**: visita a raíz, depois visita a sub-árvore esquerda, e depois visita a sub-árvore direita.
    - ★ 15 6 3 2 4 7 13 9 18 17 20
  - **postorder**: visita a sub-árvore esquerda, depois visita a sub-árvore direita, e depois visita a raíz.
    - ★ 2 4 3 9 13 7 6 17 20 18 15

Inorder

- Inorder visita os elementos por ordem crescente.

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


Search

- Dado um apontador para um nó x e um valor k, retorna um apontador para um nó da árvore que tenha $key = k$, ou NIL caso não exista nenhum $key = k$.

- Chamada inicial: TREE-SEARCH($T.root, k$)

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

Versão iterativa

```
ITERATIVE-TREE-SEARCH(x, k)
1  while x ≠ NIL and k ≠ x.key
2      if k < x.key
3          x = x.left
4      else x = x.right
5  return x
```


Minimum

- Está no nó mais à esquerda da árvore.

```
TREE-MINIMUM(x)
1  while x.left ≠ NIL
2      x = x.left
3  return x
```

Maximum

- Está no nó mais à direita da árvore.

```
TREE-MAXIMUM(x)
1  while x.right ≠ NIL
2      x = x.right
3  return x
```


Sucessor e Predecessor

Assumindo que os keys são distintos:

- O sucessor de um nó x é o nó y tal que $y.key$ é a key mais pequena que seja maior que $x.key$.
- O predecessor é definido de forma análoga. Se x é sucessor de y, então y é predecessor de x.
- Podemos obter o sucessor e predecessor de um nó sem fazer qualquer comparação, i.e. usando apenas a estrutura da árvore.

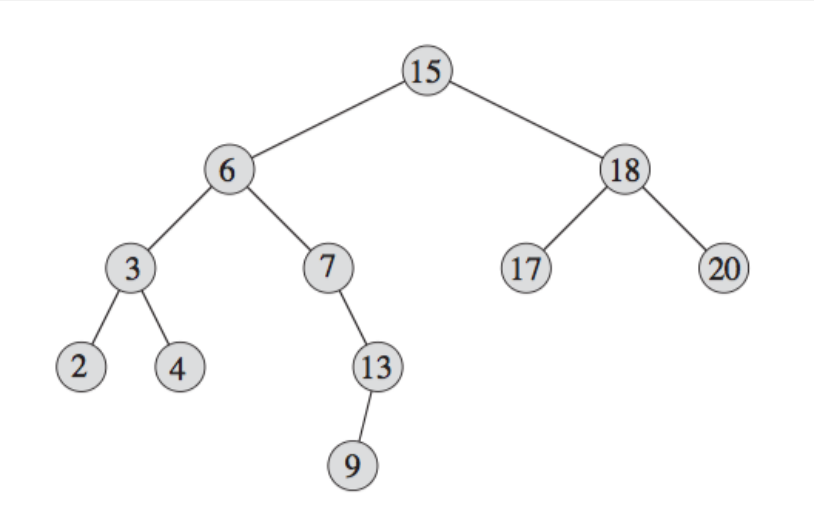
Sucessor

Como encontrar o sucessor de x? Há 2 casos:

1. x tem uma sub-árvore direita que não é vazia.
   - O sucessor de x é o mínimo dessa sub-árvore.
2. A sub-árvore direita de x é vazia.
   - Vamos subindo pela árvore a partir de x.
   - Enquanto subirmos pela esquerda (filho direito), as keys são menores que x.
   - O sucessor de x está no primeiro nó que não subiu pela esquerda.

Exemplos

- Sucessor do nó que tem $key = 15$ é o nó que tem $key = 17$.
- Sucessor do nó que tem $key = 13$ é o nó que tem $key = 15$.
- Sucessor do nó que tem $key = 20$ é NIL.



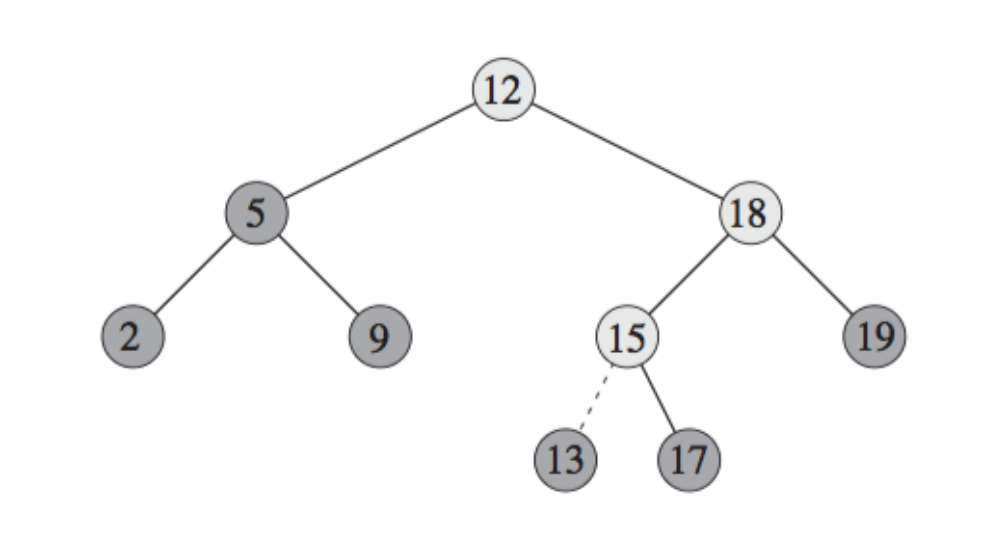
```
TREE-SUCCESSOR(x)
1  if x.right ≠ NIL
2      return TREE-MINIMUM(x.right)
3  y = x.p
4  while y ≠ NIL and x == y.right
5      x = y
6      y = y.p
7  return y
```

- TREE-PREDECESSOR(x) faz-se de forma análoga.

Inserir um elemento

- Para inserir um elemento v na árvore, o procedimento recebe um nó z com o atributo *key* já inicializado com o valor v.
- Começa na raíz e vai andando pela esq ou dta consoante $z.key$ seja $<$ ou $\geq$ que o nó corrente.
- Mantém 2 apontadores: um para x (nó corrente), outro para y (pai do nó corrente).
- z vai acabar por ser uma nova folha da árvore.

Exemplo de inserção de $key = 13$



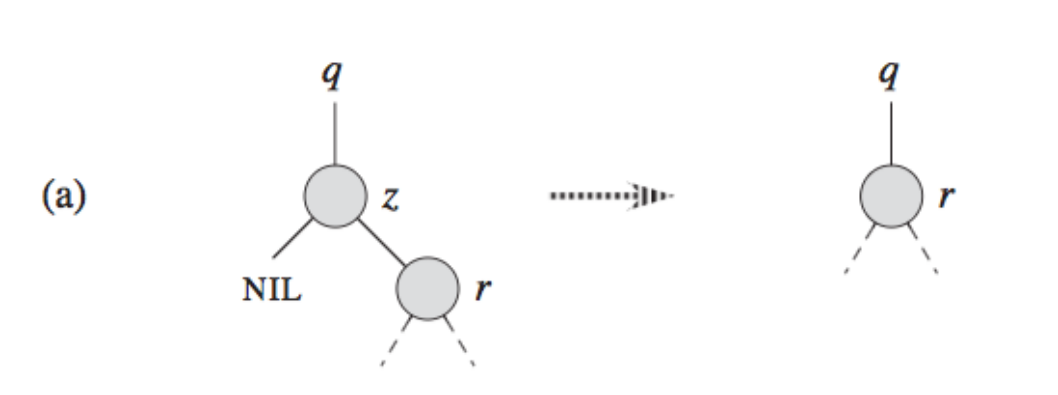

```
TREE-INSERT(T, z)
 1   y = NIL
 2   x = T.root
 3   while x ≠ NIL
 4         y = x
 5         if z.key < x.key
 6               x = x.left
 7         else x = x.right
 8   z.p = y
 9   if y == NIL
10         T.root = z          // tree T was empty
11   elseif z.key < y.key
12         y.left = z
13   else y.right = z
```

Apagar um nó z da árvore

Há 4 casos:

caso a) : z não tem filho esquerdo. (Filho dto pode ser NIL...)

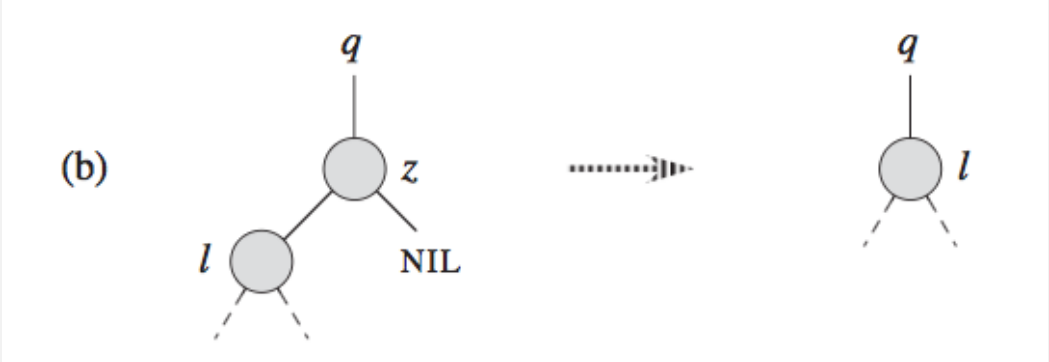
- substitui z pelo seu filho direito.



Apagar um nó z da árvore

caso b) : z só tem um filho e esse filho é o esquerdo.

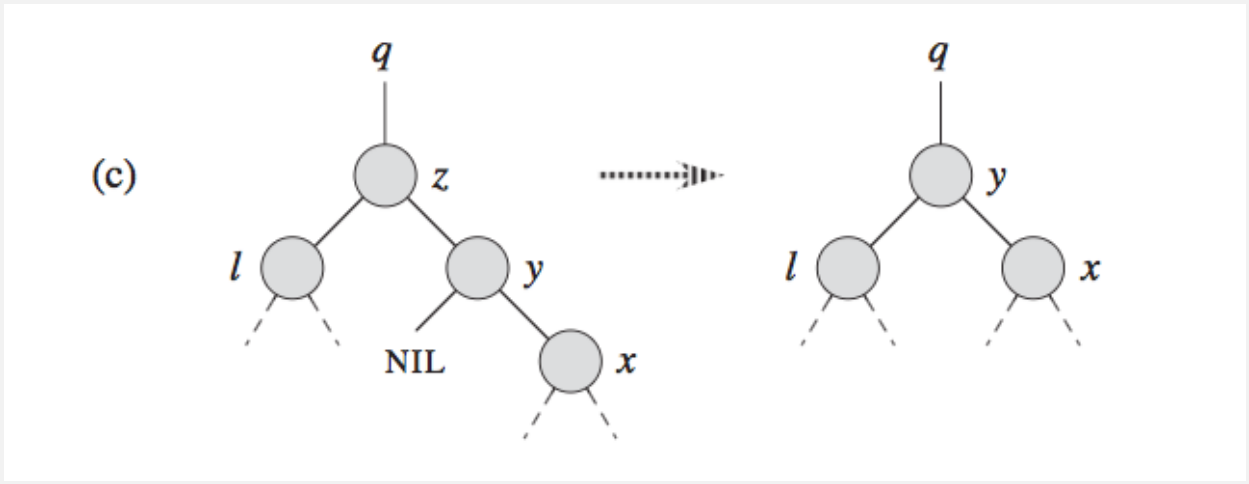
- z é substituído pelo filho esquerdo.



Apagar um nó z da árvore

caso c) : z tem 2 filhos e o filho dto é o sucessor de z.

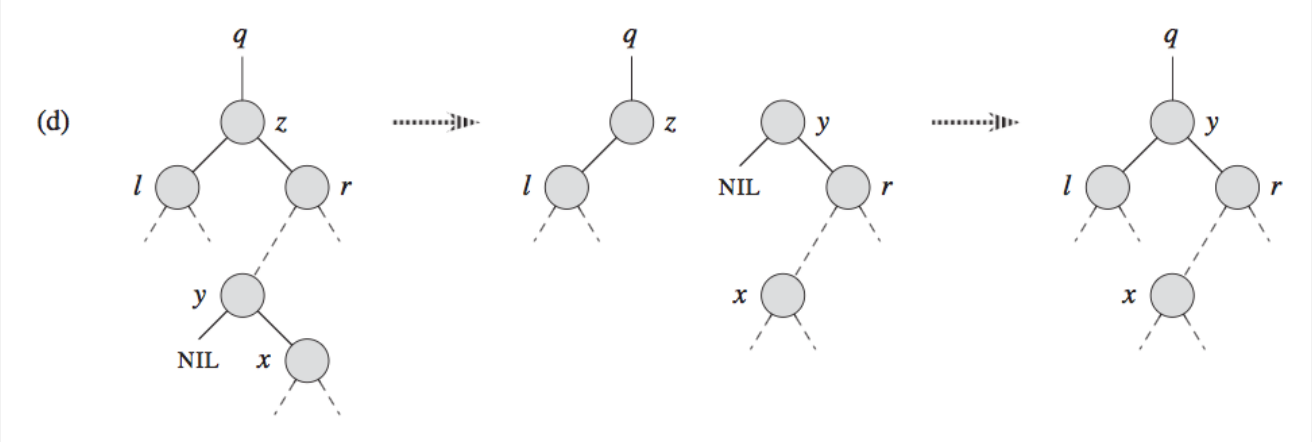
- Substitui-se z por y (o sucessor de z).
  - filho dto de y fica na mesma.
  - filho esq e y passa a ser $z.left$



Apagar um nó z da árvore

caso d) : z tem 2 filhos e o filho dto não é o sucessor de z.

- Acha-se o sucessor de z. Seja ele y. (y terá de estar na sub-árvore dta de z e não pode ter filho esq.)
  - substitui y pelo seu filho direito (x na figura)
  - depois substitui z por y.



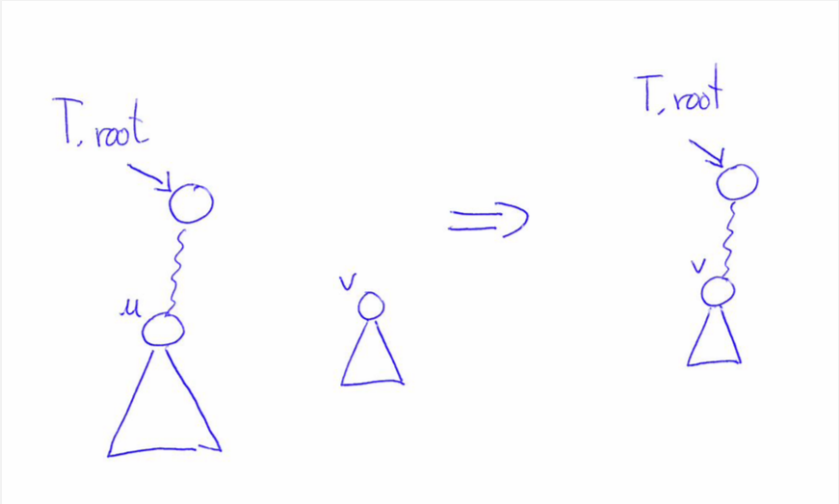
```
TREE-DELETE(T, z)
 1  if z.left == NIL
 2      TRANSPLANT(T, z, z.right)
 3  elseif z.right == NIL
 4      TRANSPLANT(T, z, z.left)
 5  else y = TREE-MINIMUM(z.right)
 6      if y.p ≠ z
 7          TRANSPLANT(T, y, y.right)
 8          y.right = z.right
 9          y.right.p = y
10      TRANSPLANT(T, z, y)
11      y.left = z.left
12      y.left.p = y
```


Sobre o código TREE-DELETE

- Faz uso da função TRANSPLANT(T, u, v) $\rightarrow$ ver próximo slide
- linhas 1–2: caso a)
- linhas 3–4: caso b)
- linha 5: encontra o sucessor de z
- linha 6: distingue caso c) e d)
- linhas 10–12: caso c)
- linhas 7–12: caso d)

TRANSPLANT(T, u, v)

- Substitui a sub-árvore que tem u como raíz pela sub-árvore que tem v como raíz.
  - pai de u passa a ser o pai de v
  - o pai de u (agora de v) tem v como filho dto ou esq, dependendo de u ter sido filho esq ou dto.




```
TRANSPLANT(T, u, v)
1  if u.p == NIL
2        T.root = v
3  elseif u == u.p.left
4        u.p.left = v
5  else u.p.right = v
6  if v ≠ NIL
7        v.p = u.p
```

Sobre o código TRANSPLANT

- linhas 1–2: é o caso de u ser raíz de T.
  (Caso contrário, u é filho esq ou dto de seu pai.)
- linhas 3–4: actualiza o pai de u se for filho esquerdo.
- linha 5: actualiza o pai de u se for filho direito.
- linhas 6–7: actualiza o pai de v caso $v \neq \text{NIL}$