Árvores de busca binária

Neste capítulo

- Introdução
- Conceito de árvores de busca binária
- Consultas
- Inserção e eliminação
- revisão

Introdução

Uma árvore de busca é uma estrutura de dados que contem as seguintes operações: busca, máximo, mínimo, predecessor, sucessor, inserir, excluir.

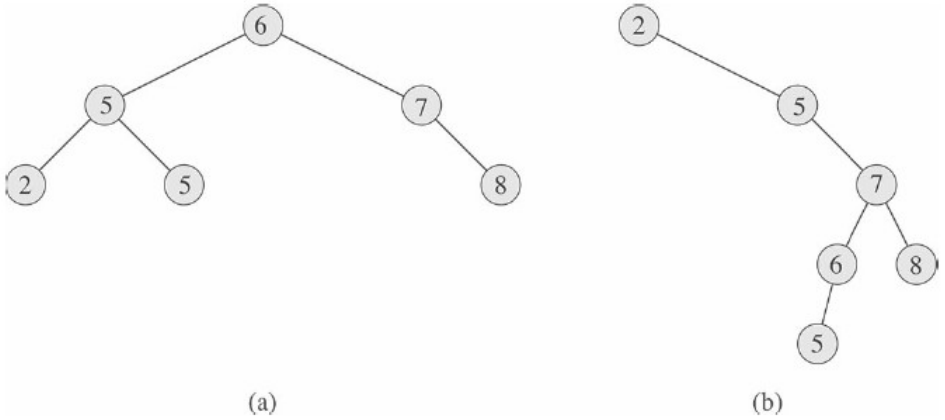
Assim, uma árvore de busca pode ser usada como um dicionário e também como uma fila de prioridades.

As operações básicas em uma árvore de busca binária demoram um tempo proporcional à altura da árvore.
Neste capítulo, vamos aprender como implementar as suas operações.

Conceito de Árvores de busca binária

Uma árvore de busca binária é organizada de tal maneira que haja no máximo dois filhos em cada nó.

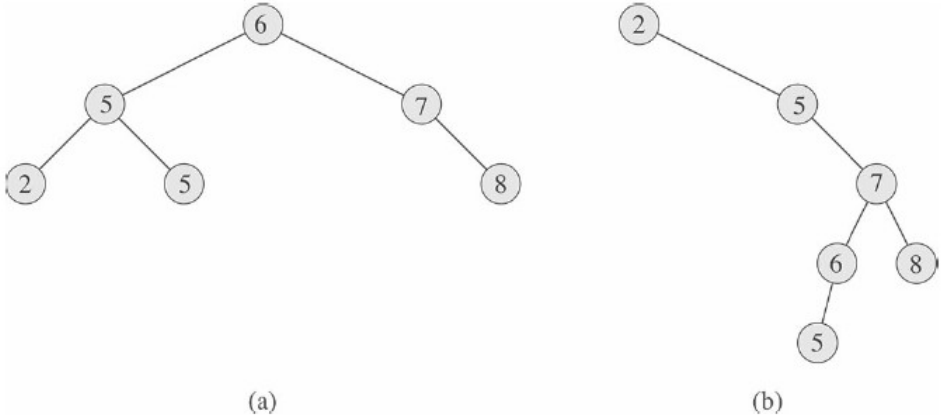
- Cada nó é um objeto
- Chave, filho a esquerda, filho a direita e pai.
- O nó raiz é o único que não possui pai.



Conceito de Árvores de busca binária

As chaves em uma árvore de busca binária são sempre armazenadas de modo a satisfazer a propriedade de árvore de busca binária:

- Todas as chaves da sub-árvore a esquerda de um nó não são maiores que o próprio nó.
- Todas as chaves da sub-árvore a direita de um nó não são menores que o próprio nó.
- Percurso em ordem, pré-ordem e pós-ordem.



Percurso em ordem

- Permite imprimir os elementos da árvore binária de forma ordenada
- Imprime a chave raiz entre as duas subárvores
- Na pré-ordem a raiz é obtida antes das subárvores
- Na pós-ordem a raiz é obtida depois das subárvores

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

Consultas em uma árvore de busca binária

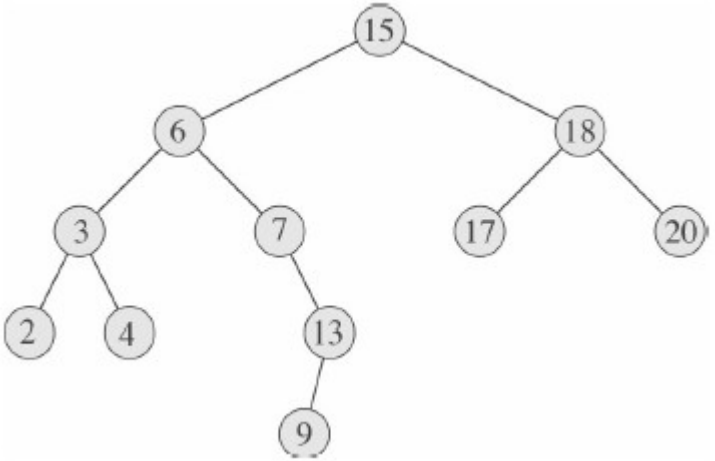
Se a árvore é de busca, então vamos buscar.

- Busca – usado para procurar um nó com determinada chave
- Mínimo e máximo - encontrar um elemento em uma árvore de busca binária cuja chave é um mínimo/máximo seguindo ponteiros de filhos da *esquerda/direita* desde a raiz até encontrarmos um valor nulo
- Predecessor e sucessor - encontra seu predecessor/sucessor na sequência ordenada determinada por um percurso de árvore em ordem.

Pseudo código da Busca

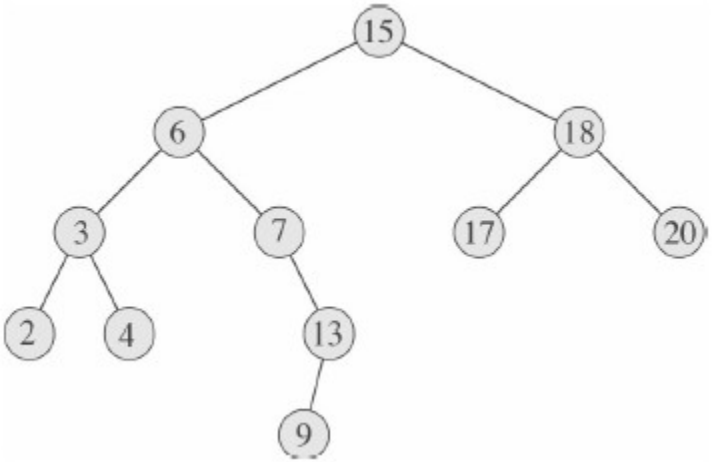
```
TREE-SEARCH(x, k)
1  if x == NIL or k == x.key
2       return x
3  if k < x.key
4       return TREE-SEARCH(x.left, k)
5  else return TREE-SEARCH(x.right, k)
```

```
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
```



Qual a complexidade algorítmica?

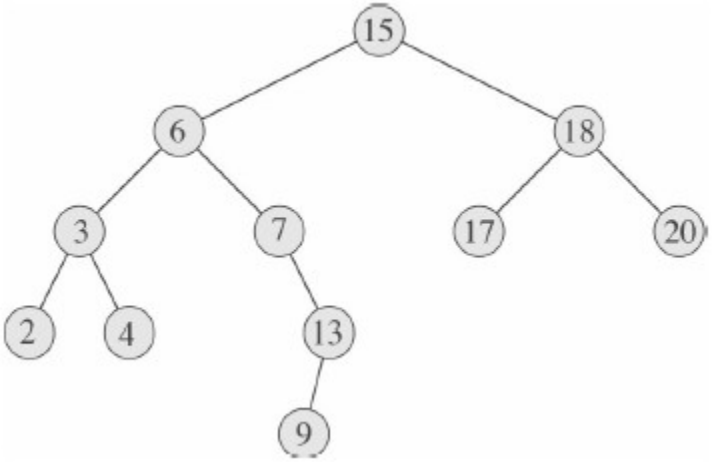
Pseudo código máximo/mínimo




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

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

Pseudo código sucessor



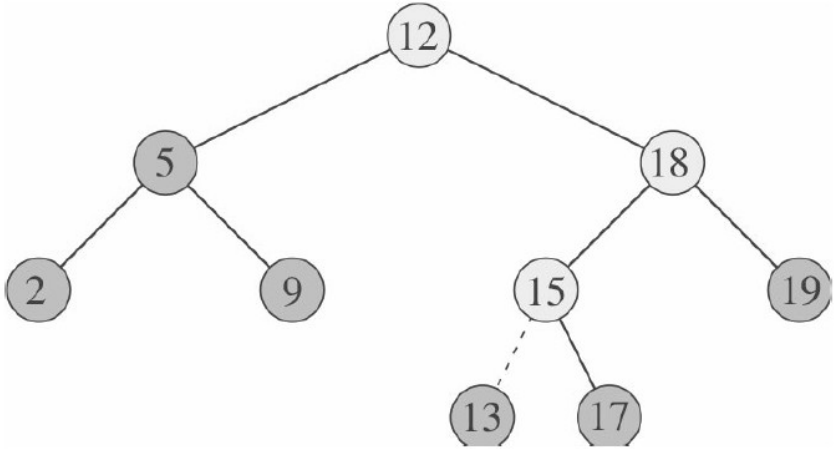

```
TREE-SUCCESSOR(x)
1  if x.right ≠ NIL
2       return TREE-MINIMUM(x.right)
3  y = x.p
4  while y ≠ NIL and x == y.right
5       x = y
6       y = y.p
7  return y
```

- Elemento sucessor é o menor dentre os maiores filhos de um nó.
- Elemento predecessor é o maior dentro os menores filhos de um nó.
- Desafio: escreva um pseudo código para o predecessor.

Inserção e eliminação em uma árvore de busca binária

Inserção e eliminação provocam mudanças na árvore de busca binária. A estrutura de dados deve ser modificada para refletir essa mudança.

- A propriedade de árvore de busca binária deve continuar válida.
- O procedimento de inserção vai precisar de dois ponteiros.
- O caminho percorrido é indicado pelos nós mais claros.
- A linha tracejada indica o nó inserido na posição adequada.



Pseudo código para inserção

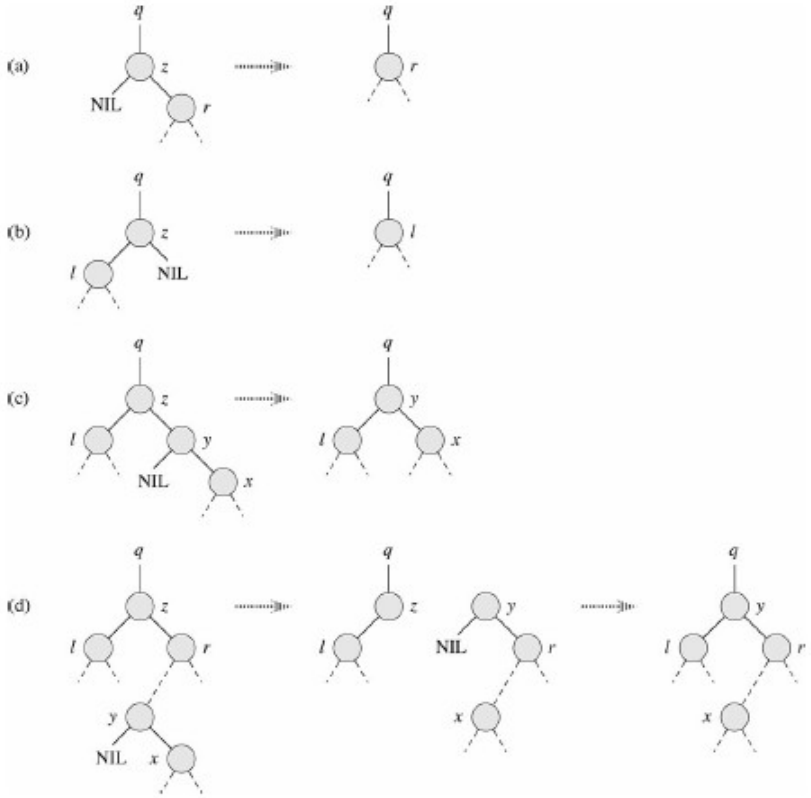
```
TREE-INSERT(T, z)
 1  y = NIL
 2  x = T.root
 3  while x ≠ NIL
 4       y = x
 5       if z.key < x.key
 6            x = x.left
 7       else x = x.right
 8  z.p = y
 9  if y == NIL
10       T.root = z          // tree T was empty
11  elseif z.key < y.key
12       y.left = z
13  else y.right = z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

- z é o nó a ser inserido, tal que inicialmente z→left = z→ right = Nulo;
- y e x são nós ponteiros auxiliares usados para percorrer a árvore.
- O percurso começa pela raiz.
- Complexidade da operação é O(h)

Estratégia para eliminação

A estratégia para eliminar um nó *z* de uma árvore de busca binária *T* tem quatro casos.

A. O nó z não tem nenhum filho a esquerda
B. O nó z tem um filho a esquerda mas nenhum filho a direita
C. O nó z tem dois filhos
D. O nó z tem dois filhos e o sucessor y está enraizado em r



Pseudo código para eliminação

```
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
```

```
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
```

Exercícios

- Trace árvores de busca binária de alturas 2, 3, 4, 5 e 6 para o conjunto de chaves {1, 4, 5, 10, 16, 17, 21}
- Dê algoritmos recursivos que executem percursos de árvores em pré ordem e pós- ordem no tempo O(n) em uma árvore de n nós.
- Escreva o procedimento TREE-PREDECESSOR.
- Crie uma versão da árvore de busca binária orientada a objeto.

Revisão

Atenção, chegou a hora da revisão.

- Busca – usado para procurar um nó com determinada chave
- Mínimo e máximo - encontrar um elemento em uma árvore de busca binária cuja chave é um mínimo/máximo seguindo ponteiros de filhos da *esquerda/direita* desde a raiz até encontrarmos um valor nulo
- Predecessor e sucessor - encontra seu predecessor/sucessor na sequência ordenada determinada por um percurso de árvore em ordem.