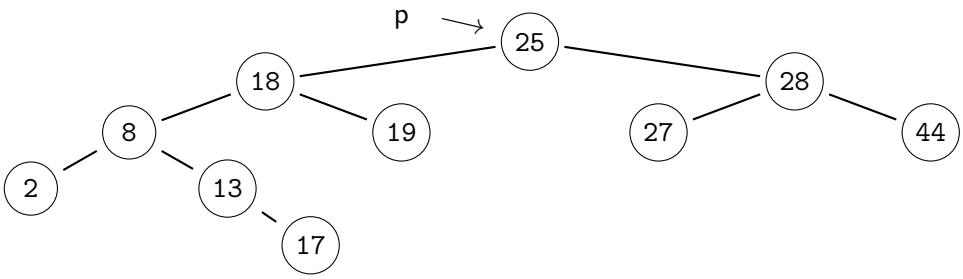
Estruturas de Dados

simulado 3

Parte I – Algoritmos recursivos para árvores

1. Contando as folhas ímpares

   Imagine que o ponteiro `p` aponta para uma árvore binária



   O problema é

   → *Contar o número de folhas ímpares na árvore*

   No exemplo acima, isso nos daria

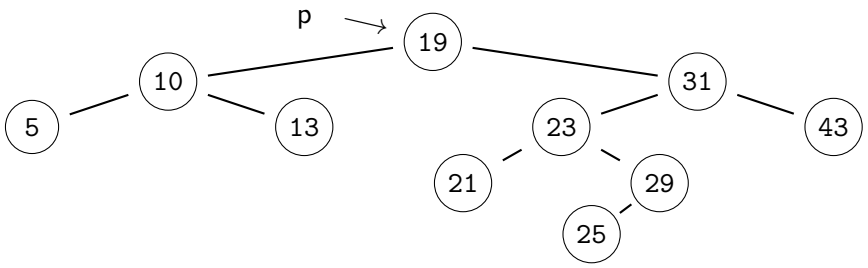
```
=>  3 folhas ímpares:  17, 19, 27
```

   a) Apresente um algoritmo recursivo que resolve esse problema.

   b) E analise o tempo de execução do seu algoritmo.

2. Imprimindo descendentes e ancestrais

   Imagine que nós temos uma árvore binária de busca apontada pelo ponteiro `p`.



   Agora considere as seguintes definições

   - os descendentes de um elemento `k` são todos os elementos que se encontram na sua subárvore esquerda e na sua subárvore direita
   - os ancestrais de um elemento `k` são todos os elementos que se encontram no caminho de `k` até a raiz — (*incluindo a raiz, mas não* `k`)

   O problema consiste em

→ *Dado um elemento* `k`

- *imprimir todos os seus descendentes na árvore*
- *e imprimir todos os seus ancestrais na árvore*

— *(se o* `k` *não está na árvore, então nada deve ser impresso)*

No exemplo acima, para `k` = 23, isso nos daria

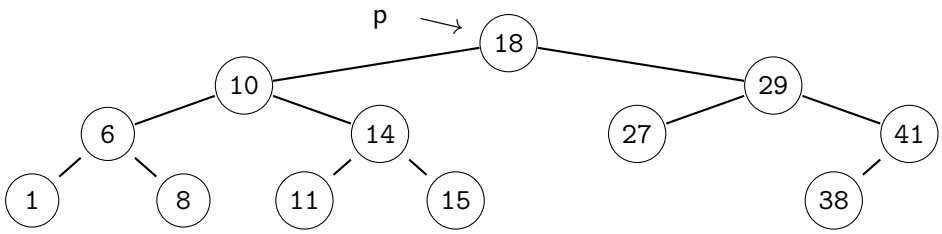
`- descendentes:` 21, 25, 29 `- ancestrais:` 31, 19

a) Apresente um algoritmo recursivo que resolve esse problema.

b) E analise o tempo de execução do seu algoritmo.

3. Inserção no nível `k`

Imagine que nós temos uma árvore binária de busca apontada pelo ponteiro `p`.



O problema consiste em

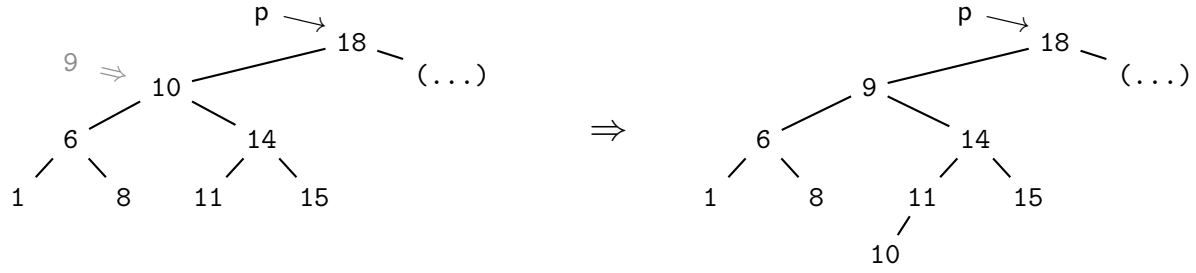
→ *Inserir um novo elemento* `x` *no nível* `k` *da árvore*

Quer dizer, nós fazemos a navegação na árvore utilizando o valor de `x`, como usual.

E daí,

- Se o nível `k` não puder ser alcançado, então a operação é abortada
- Se a posição adequada no nível `k` está vazia, então o `x` é inserido ali.
- Se a posição adequada no nível `k` estiver ocupada, então o `x` toma o lugar do elemento que está ali, e esse elemento é empurrado p/ baixo.

No exemplo acima, a inserção do `x` = 9 no nível `k` = 2, produziria o seguinte

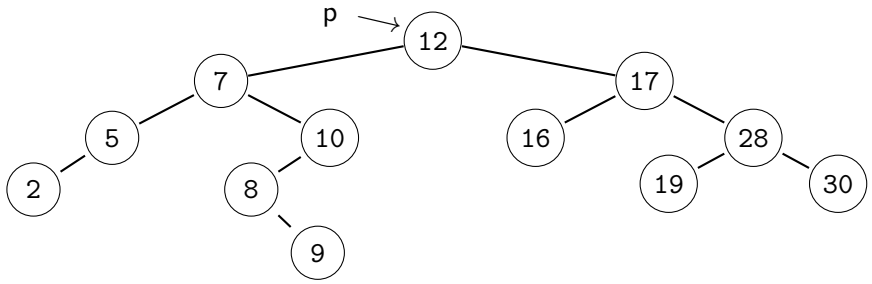


a) Apresente um algoritmo recursivo que resolve esse problema.

b) Analise o tempo de execução do seu algoritmo.

4. Removendo todos os ímpares

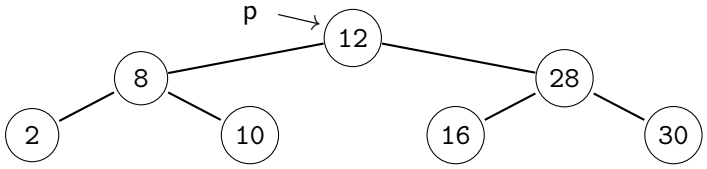
Imagine que nós temos uma árvore binária de busca apontada pelo ponteiro `p`.



O problema consiste em

→ *Remover todos os elementos ímpares da árvore*

No exemplo acima, isso nos daria



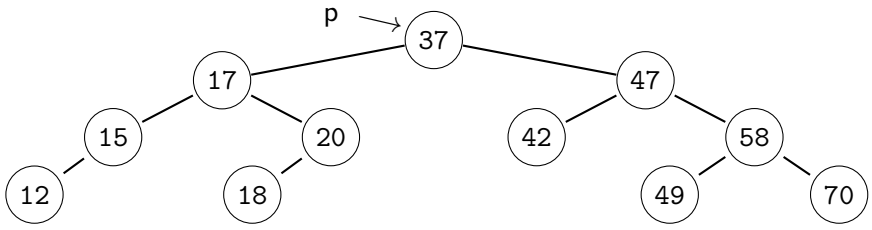
a) Apresente um algoritmo recursivo que resolve esse problema.

b) Analise o tempo de execução do seu algoritmo.

Parte II – Algoritmos iterativos para árvores

5. Busca aproximada

Imagine que `p` é um ponteiro que aponta para uma árvore binária



O problema é

→ *Imprimir o elemento mais próximo de* `k` *na árvore*

— *(se o* `k` *está na árvore, então é ele quem deve ser impresso)*

No exemplo acima, para `k` = 29, isso nos daria

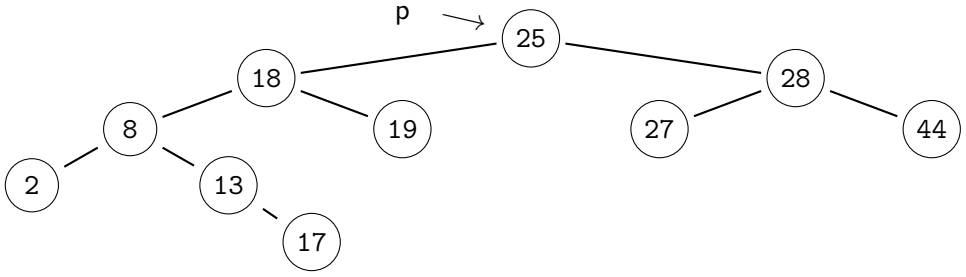
```
=>   37
```

a) Apresente um algoritmo iterativo que resolve esse problema.

b) Analise o tempo de execução do seu algoritmo.

6. Imprimindo o k e os 5 que vem depois

Imagine que p é um ponteiro que aponta para uma árvore binária



O problema é

→ *Imprimir o elemento* `k`
e os 5 elementos que vem depois dele na árvore — (de acordo com a ordem crescente)

No exemplo acima, para `k` = 13, isso nos daria

```
=>  13, 17, 18, 19, 25, 27
```

a) Apresente um algoritmo iterativo que resolve esse problema.

b) E analise o tempo de execução do seu algoritmo.

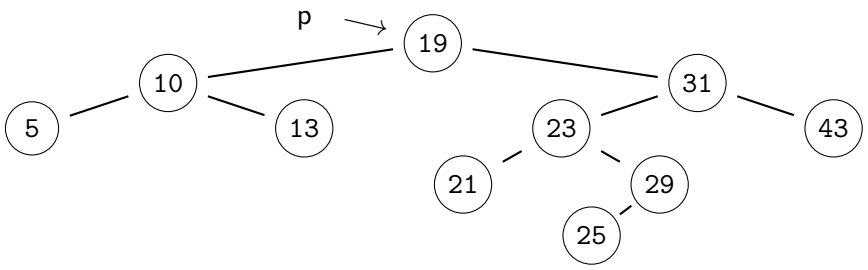
Dica: Use o truque do sinal de trânsito:

- você desce até o `k` ajustando as luzes
- e depois você usa as luzes para percorrer a árvore

Ou então você pode usar o truque da cordinha.

7. Remover o k e toda a sua descendência

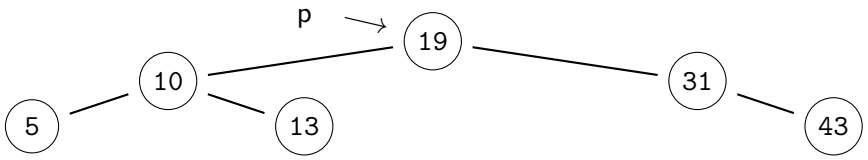
Imagine que o ponteiro p aponta para uma lista encadeada



O problema é

→ *Remover o elemento* `k` *e toda a sua descendência*
— *(se o* `k` *não está na árvore, então ninguém deve ser removido)*

No exemplo acima, para `k` = 23, isso nos daria



a) Apresente um algoritmo iterativo que resolve esse problema.

b) E analise o tempo de execução do seu algoritmo.

Parte III – Árvores balanceadas

8. Balanceando a quantidade de elementos

Imagine que alguém teve uma ideia diferente para balancear a árvore binária.

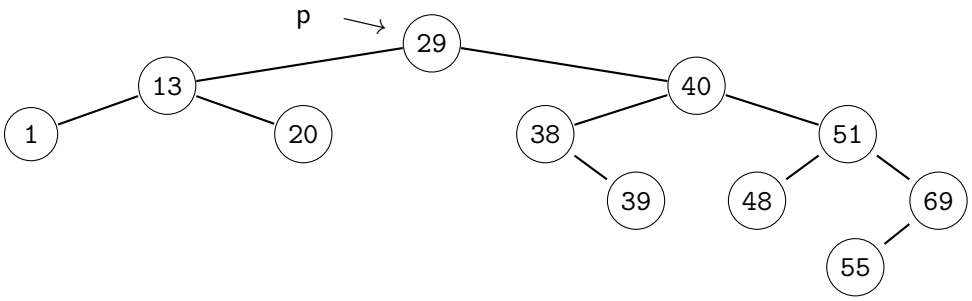
Quer dizer, ao invés de olhar para a altura de cada lado, a ideia é olhar para o número de elementos de cada lado.

Mais especificamente, a definição fica assim

Definição: *Nós dizemos que uma árvore binária é* BQE *se para todo elemento x da árvore*

- *o número de elementos à esquerda de x*
  é no máximo o dobro do número no lado direito mais 1
- *o número de elementos à direita de x*
  é no máximo o dobro do número no lado esquerdo mais 1

Por exemplo, a árvore abaixo é BQE



a) Apresente um algoritmo que implementa a operação de inserção na árvore BQE.

O seu algoritmo precisa garantir que a árvore continua BQE após a inserção do novo elemento.

b) Apresente um algoritmo que implementa a operação de remoção na árvore BQE.

O seu algoritmo precisa garantir que a árvore continua BQE após a remoção do elemento.

c) A propriedade BQE é suficiente para garantir a altura $O(\log n)$ da árvore.

Se você acha que não, apresente um contra-exemplo.