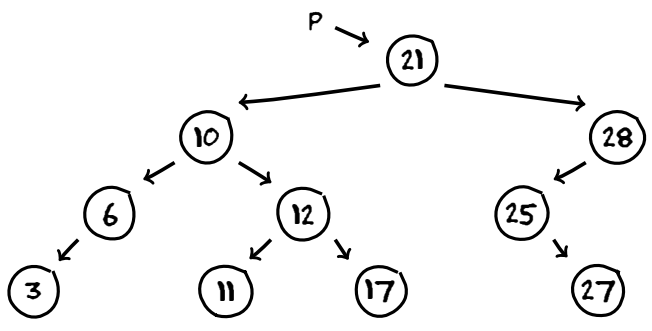
Impressão por níveis

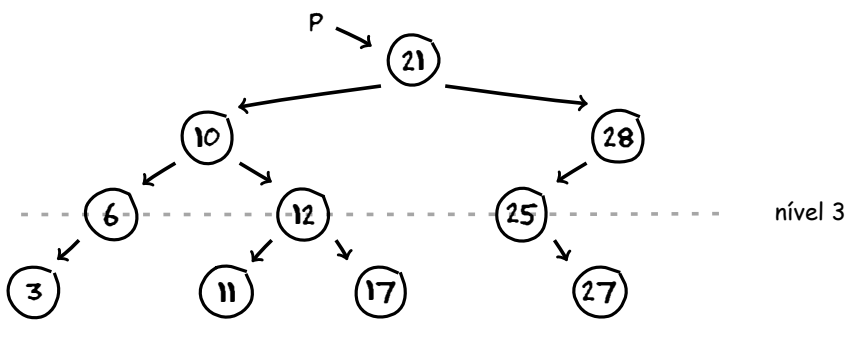
Imagine que o ponteiro `p` aponta para uma árvore binária de busca



A tarefa é

→ *Imprimir todos os elementos do nível* `k`

No exemplo acima, para `k = 3`, isso nos daria




```
=>  6,   12,   15
```

Daí, o raciocínio recursivo fica assim

1. Se o elemento atual está no nível `k`,
   então a gente imprime ele e acabou-se
2. Senão
   a gente imprime os elementos do nível `k` na subárvore esquerda
   e imprime os elementos do nível `k` na subárvore direita

Certo.

Mas, como é que a gente sabe o nível em que a gente está?

Bom, passando essa informação como parâmetro para a função.

A coisa fica assim

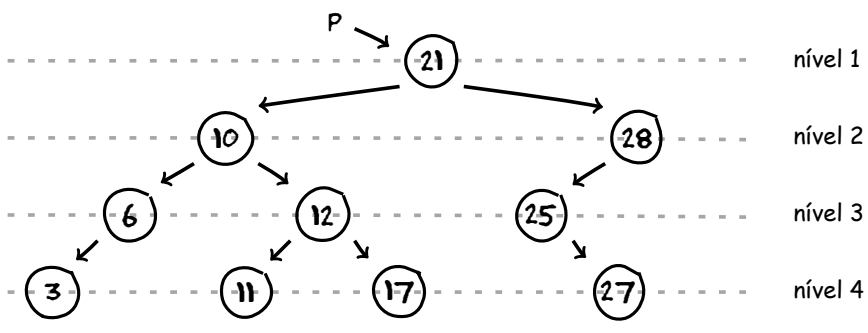
```
func INKA-Rec (q,k,nivel)                    // Imprime o Nível K da Árvore

      Se ( q == Nulo )   Retorna

      Se (nivel == k)
            Imprime (q->num)
            Retorna
      Senão
            INKA-Rec (q->esq,k,nivel+1)
            INKA-Rec (q->dir,k,nivel+1)
```

Legal.

Agora imagine que nós queremos imprimir todos os níveis da árvore




```
=>  21
    10,   28
     6,   12,   25
     3,   11,   17,   27
```

Bom, daí basta colocar um laço na função `main()`

```
main ()

      alt == altArv-Rec (p)

      Para k = 1 Até alt
            INKA-Rec (p,k,1)
```

Mas, a gente também pode fazer tudo recursivo.

Quer dizer, imagine que nós modificamos a função `INKA-Rec ()` para que ela retorne

- `k`, se algum elemento foi impresso
- -1, se não há ninguém no nível `k`

Daí, a gente pode escrever a função de impressão por níveis assim

```
func IPN-Rec (q,k)                           // Imprime Por Níveis

      resp = INKA-Rec (q->esq,k,nivel+1)

      Se (resp == -1)   Retorna
      Senão               INP-Rec (q,k+1)
```

E a função `main()` fica assim

```
main ()

      IPN-Rec (p,1)
```

Quer dizer, a recursão foi usada para simular o laço de repetição.

Legal, não é?

◇