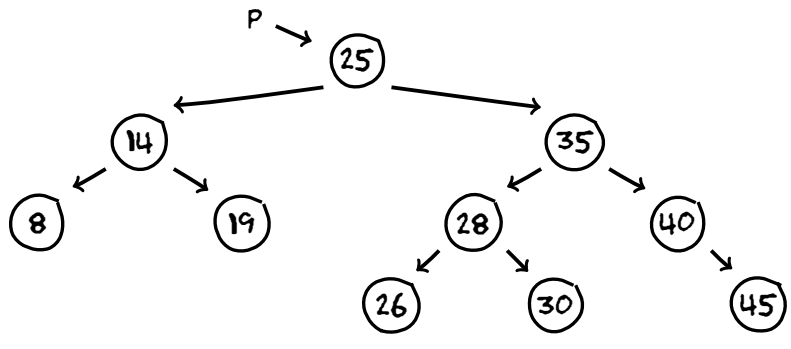
Imprimindo uma faixa de elementos

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



A tarefa é

→ *imprimir os elementos que tem valor entre* `k` *e* ℓ

No exemplo acima, para `k` = `17` e $\ell = 37$, isso nos daria

=¿ 19, 25, 26, 28, 30, 35

Certo.

Nós já temos uma função que encontra o menor elemento maior que `k`.

Então, nós podemos colocar o dedo sobre ele, e imprimir o seu valor.

Mas, e depois?

Bom, depois a gente tem que ir para o próximo elemento.

Só que, o próximo elemento é o menor elemento que é maior do que o elemento atual.

Então, a gente pode usar a nossa função outra vez.

A coisa fica assim

```
func IFE-Arv (p,k,ℓ)                                    // Maior menor que k
      q = MnK-Arv (p,k)
      Enquanto (q != Nulo)
            Imprime (q->num)
            q = MnK-Arv (p,q->num)
```

Legal.

Mas, essa não é a única maneira de fazer as coisas.

Quer dizer, a gente também pode escrever uma função que encontra o próximo elemento sem começar outra vez na raiz.

Porque muitas vezes o próximo elemento vai estar ali por perto.

Existem basicamente dois casos

1. Quando o filho direito do elemento atual não é `Nulo`,
   o próximo é o menor elemento na subárvore
2. Quando o filho direito é `Nulo`, a gente precisa subir na árvore
   — (*utilizando o ponteiro pai*)
   Na subida, existem dois casos
   - Se a gente chegou no pai vindo do filho esquerdo, então o pai é o próximo elemento
   - Se a gente chegou no pai vindo do filho direito, então é preciso continuar subindo

A coisa fica assim

```
func proxEl-Arv (q)
      Se (q->dir != Nulo)                               // caso 1:  descida
            px = q->dir
            Enquanto (px->esq != Nulo)
                  px = px->esq
            Retorna (px)
      Senão                                             // caso 2:  subida
            aux = q;   pai = q->pai
            Enquanto (pai != Nulo)
                  Se (p->esq == aux)   Quebra
                  Senão
                        aux = pai;   pai = pai->pai     // ha ha ha ...
            Retorna (pai)
```

E agora que nós temos essa função, nós podemos resolver o problema outra vez

```
func IFE-Arv2 (p,k,ℓ)                                   // Maior menor que k
      q = MnK-Arv (p,k)
      Enquanto (q != Nulo)
            Imprime (q->num)
            q = proxEl-Arv (q)
```

◇