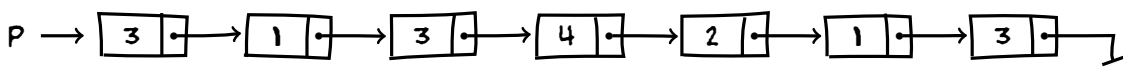
Contando o número de ocorrências

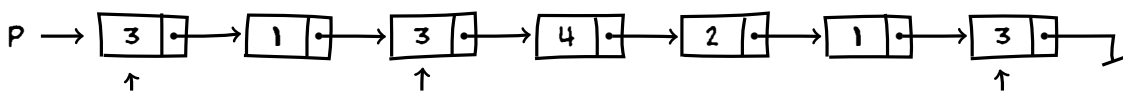
Imagine que o ponteiro `p` aponta para uma lista encadeada que pode conter elementos repetidos



A tarefa é

→ *Contar a quantidade de ocorrências do número* `k`

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




```
=>  3 ocorrências
```

Bom, dessa vez a gente não pode começar a voltar quando encontra a primeira ocorrência do `k`.

Quer dizer, a ideia é ir até o final da lista.

E daí, a gente pode fazer a contagem na volta.

Quer dizer, a chamada que vê o `Nulo` retorna 0.

E toda chamada que vê o número `k` adiciona 1 ao valor que está sendo retornado.

A coisa fica assim

```
func contaOcK-Rec (q,k)
      Se ( q == Nulo )   Retorna (0)              // começando a contar do 0
      resp = contaOcK-Rec (q->prox, k)
      Sse ( q->num == k )                         // Você é o kara?
            resp++
      Retorna (resp)
```

E a função `main()` fica assim

```
   func main ()
         resp = contaOcK-Rec (p, 3)
         Imprime ('o número de ocorrências do 3 é:', resp)
```

Legal.

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

Quer dizer, a gente também pode fazer a contagem na ida.

Para fazer isso, a gente adiciona um parâmetro `cont` à nossa função.

Daí, a função `main()` inicia a contagem passando o valor 0 ao parâmetro `cont`.

E cada chamada que vê uma ocorrência do `k` adiciona 1 ao cont.

No final, a chamada que vê o `Nulo` retorna o valor de `cont`.

E todas as demais repassam o valor retornado para a chamada anterior — (*para que ele até a função* `main()`)

A coisa fica assim

```
func contaOcK2-Rec (q,k,cont)
      Se ( q == Nulo )   Retorna (cont)           // retornando o resultado
      Sse ( q->num == k )                         // Você é o kara?
            resp = contaOcK-Rec (q->prox, k, cont+1)
      Senão
            resp = contaOcK-Rec (q->prox, k, cont)
```

◇