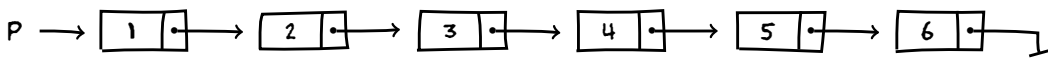
Invertendo a ordem da lista

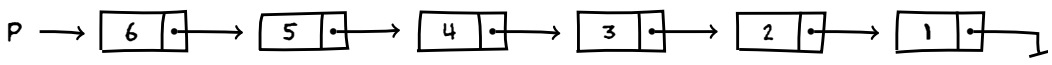
Imagine que o ponteiro p aponta para uma lista encadeada



A tarefa é

→ *inverter a ordem dos elementos da lista*

No exemplo acima, isso nos daria



Bom, aqui o raciocínio recursivo natural é o seguinte

1. primeiro a gente inverte a ordem dos elementos no restante da lista
2. e depois, a gente coloca o 1o elemento na última posição
   — (*fazendo uma chamada recursiva*)

E a coisa fica assim

```
func inverte-Rec (q)

      Se ( q == Nulo  OU  q->prox == Nulo )          // já inverti ...
            Retorna (q)

      px  = inverte-Rec (q->prox)
      q->prox = px
      q  = PNF-Rec (q)                               // Primeiro No Fim
      Retorna (q)
```

O problema é que essa função executa em tempo $O(n^2)$.

Porque cada elemento da lista faz uma chamada à função `PNF-Rec()`.

E a função `PNF-Rec()` executa em tempo $O(n)$ — (*porque?*)

E agora?

Bom, agora é a hora para uma pequena esperteza.

Quer dizer, a ideia é fazer a nossa função retornar um ponteiro u para o último elemento da lista.

Porque daí, cada chamada pode colocar o seu elemento no fim, sem chamar a função `PNF-Rec()`.

A coisa fica assim

```
func inverte2-Rec (q)

      Se (q == Nulo)   Retorna (Nulo)              // caso da lista vazia

      Se (q-prox == Nulo)   Retorna (q, q)         // o último elemento

      px  = inverte2-Rec (q->prox)
      q->prox = Nulo                               // coloca o 1o no fim
      u->prox = q                                  //
      Retorna (px)
```

◇