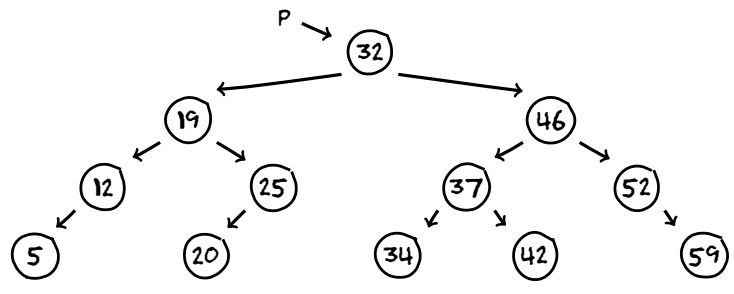
Separando pares e ímpares

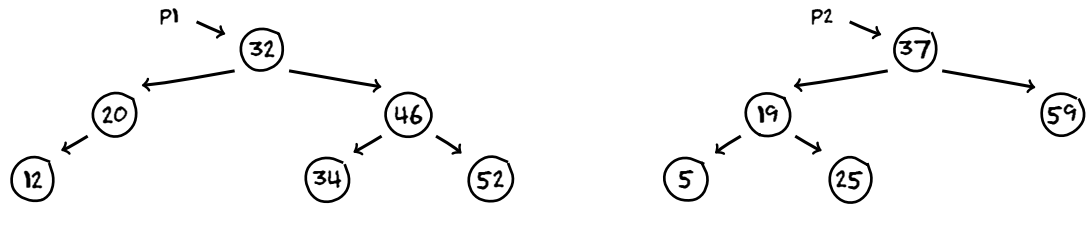
Imagine que o ponteiro p aponta para uma árvore binária de busca



A tarefa é

→ *separar os elementos pares e ímpares da árvore em duas árvores* p1 *e* p2

No exemplo acima, isso nos daria



Certo.

Mas, como é que a gente faz isso?

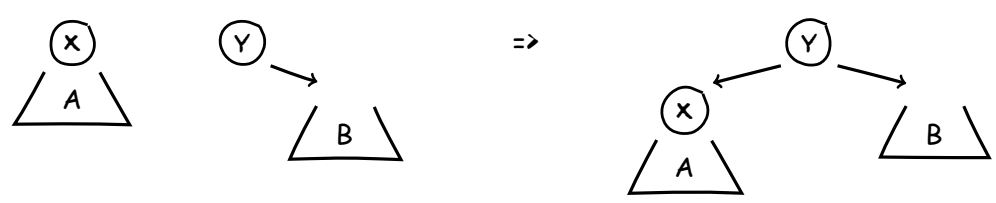
Bom, nesse exemplo a raiz da árvore é um número par

Então, depois de separar os pares e ímpares nas subárvores esquerda e direita, nós podemos montar a árvore para com a raiz da árvore e as subárvores pares de cada lado.

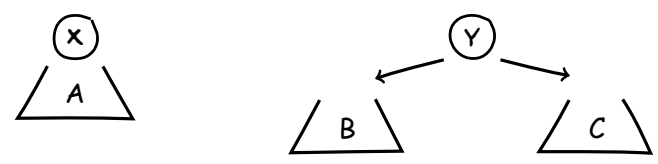
Mas, como a gente monta a árvore ímpar?

Bom, o caso em que uma das subárvores ímpares está vazia é fácil — (*basta retornar a outra*)

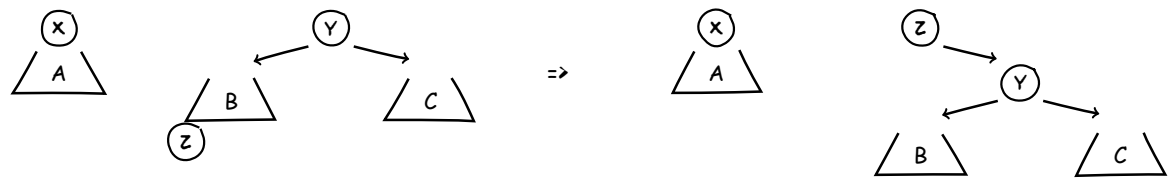
E o caso em que a subárvore ímpar da direita não tem o filho esquerdo também é fácil



Então agora, a gente considera o caso em que a subárvore ímpar da direita tem o filho esquerdo



Bom, nesse caso a gente pode remover o menor elemento do lado esquerdo (B), e colocá-lo na posição da raiz



E agora nós já sabemos o que fazer.

A coisa fica assim

```
func SPI-Rec (q)                                        // Separando Pares e Ímpares
      Se (q == Nulo)   Retorna ( Nulo, Nulo )
      a1,b1 = SPI-Rec (q->esq)
      a2,b2 = SPI-Rec (q->dir)
      Se (q->num é par)
            q->esq,dir = a1,a2
            Se (b1 == Nulo)   Retorna (q,b2)
            Sse (b2 == Nulo)   Retorna (q,b1)
            Sse (b2->esq == Nulo)
                  b2->esq:= b1;   Retorna (q,b2)
            Senão
                  z,b2 = removeMenor (b2)
                  z->esq,dir = b1,b2;   Retorna (q,z)
      Senão
      // análogo ao caso anterior ...
```