- retornar o maior elemento sub-árvore esquerda

```
1. PtrNoArvore getMaxAux (PtrNoArvore *node) {
2.     PtrNoArvore ret;
3.     if((*node) -> direita == NULL) {
4.         ret = (*node);
5.         (*node) = (*node) -> esquerda;
6.         return (ret);
7.     }
8.     Return (getMaxAux(&(*node) -> direita));
9. }
```

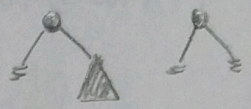
critério de parada e rearranjo da árvore

- retornar o menor elemento sub-árvore

```
1. PtrNoArvore getMinAux (PtrNoArvore * node) {
2.     PtrNoArvore ret;
3.     if((*node) -> esquerda == NULL) {
4.         ret = (*node);
5.         (*node) = (*node) -> direita;
6.         return (ret);
7.     }
8.     return (getMinAux(&(*node) -> esquerda));
9. }
```

Registre interno visando o rearranjo da árvore

* Encontramos o que remover ...



```
if ((*node) → chave == x) {
    PtrNoArvore tmp = (*node);
  /* casos 1 e 2, sub-árvore esquerda é nula,
     folhas caem aqui também */
    if ((*node) → esquerda == NULL) {
        (*node) = (*node) → direita;
    }
    // caso 3, sub-árvore direita é nula
    else if ((*node) → direita == NULL) {
        (*node) = (*node) → esquerda;
    }
    /* caso 4, nenhuma sub-árvore é nula
       usar: pegar o maior elemento da sub-árvore
       esquerda */
       tmp = getMaxAux (&(*node) → esquerda);
    // tmp = getMinAux (&(*node) → direita);
       (*node) → chave = tmp → chave
    }
    free (tmp);
    return (tmp);
}
```

* Destrutor Árvore

```
1. void DestroiArvore (PtrNOArvore *node){
2.     if ((*node) != NULL){
3.         DestroiArvore (&(*node)->esquerda);
4.         DestroiArvore (&(*node)->direita);
5.         free (*node);
6.         *node = NULL;
7.     }
8. }
```