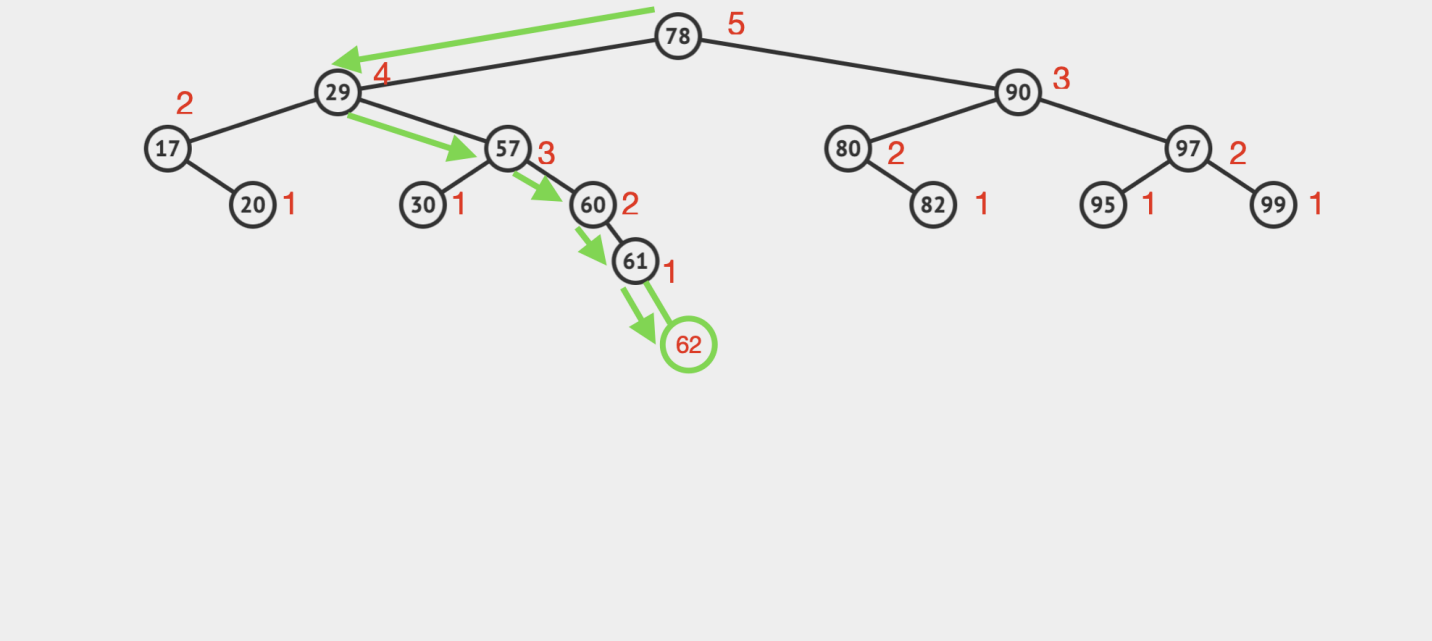
AVL

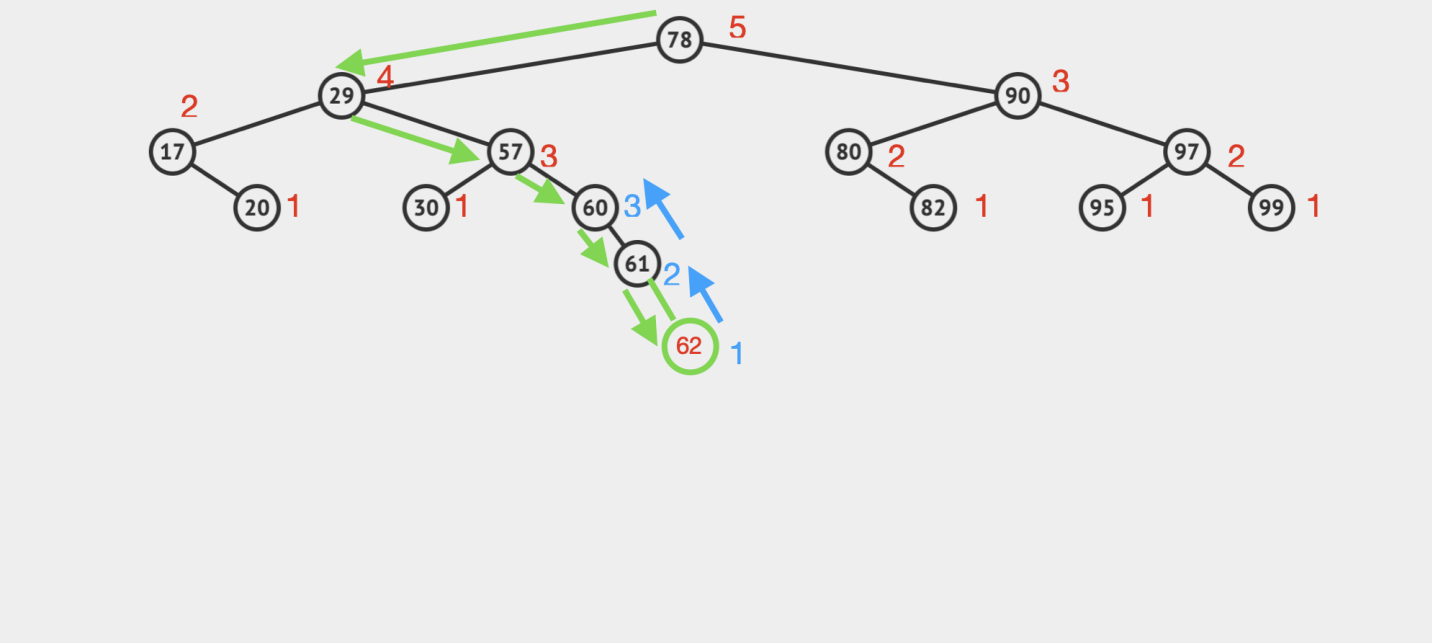
Inserare

Pasi:

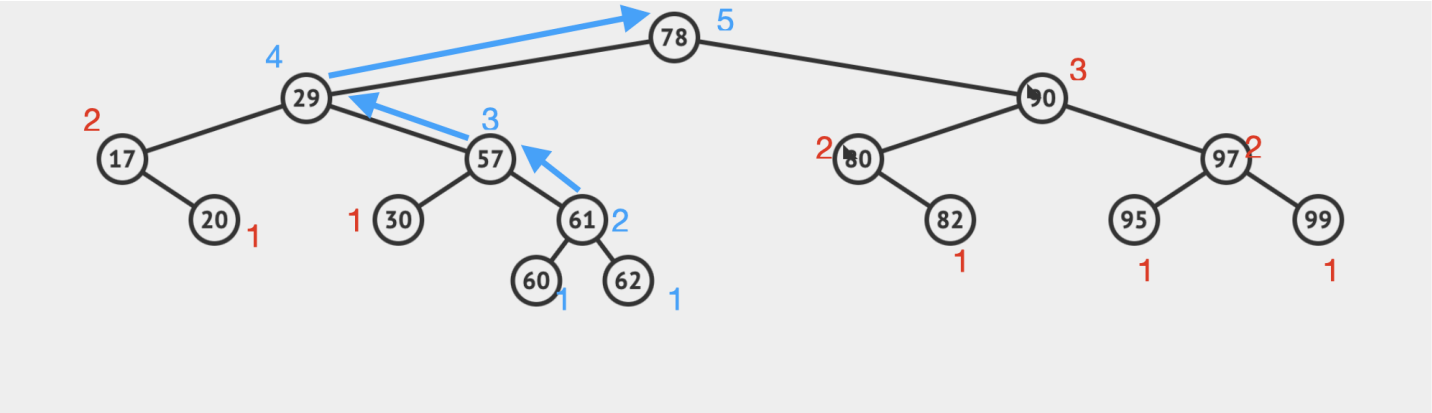
1. Traversarea arborelui de sus in jos pentru a gasi pozitia corecta pentru inserarea noului nod; se va marca drumul de la radacina la acest nou nod
2. Traversarea inversa a arborelui: de la nodul nou inserat inapoi spre radacina se pastreaza acelasi drum marcat la punctul 1; pentru fiecare nod din acest drum:
    1. actualizarea inaltimilor: height = 1 + max(height(left), height(right))
    2. calcularea balansului (inaltime_left - inaltime_right)
    3. daca modulul balansului e 2 sau mai mare, se va aplica una din cele 4 rotatii discutate



Se traverseaza arborele si se insereaza nodul la pozitia corecta; se marcheaza drumul.



Pornind de la nodul nou inserat in directie inversa pe acelasi drum marcat anterior, se vor actualiza inaltimile. In acelasi timp se va calcula balansul acestor noduri (inaltime urmas stanga - inaltime urmas dreapta) si unde se va gasi o valoare egala cu 2 sau -2 se va aplica o rotatie. Nodul 60 are balans -2: urmasul dreapta are inaltime 2 si nu are urmas stanga (deci inaltimea urmasului stanga e considerata 0).



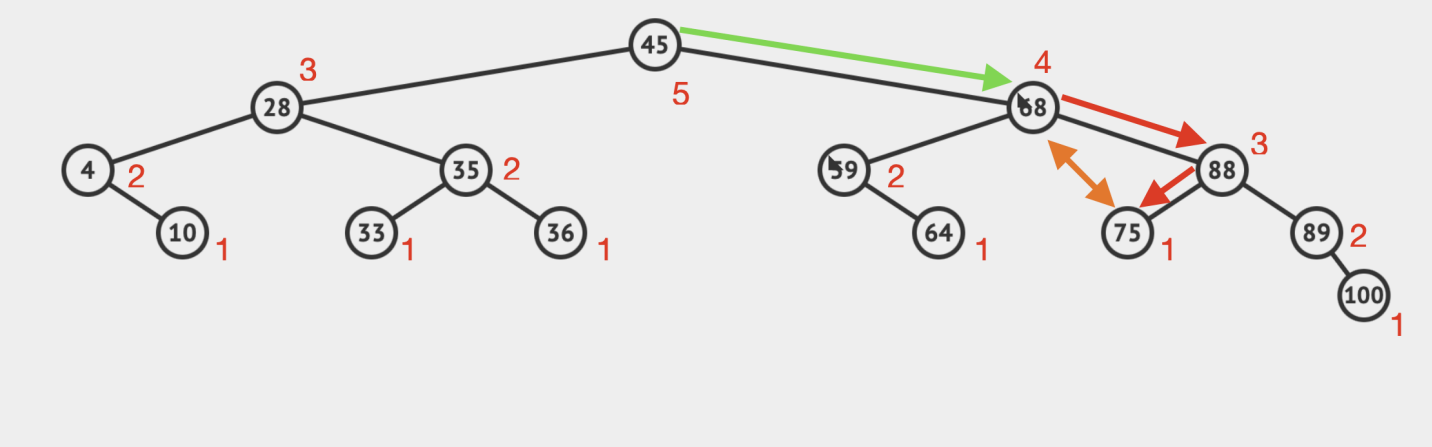
Vom aplica rotatie la nodul 60.

Stergere

Stergerea nodului cu valoare x

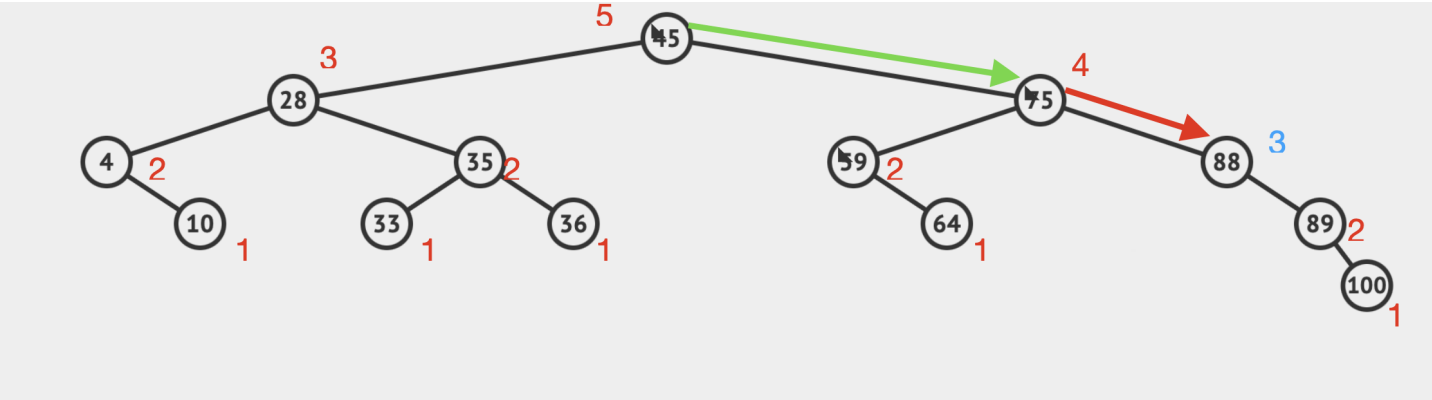
Pasi:

1. Traversarea arborelui de sus in jos pentru a gasi nodul ce trebuie eliminat (nodul x)
2. Daca a fost gasit, avem unul din cele 3 cazuri:
    1. x nu are urmasi, deci poate fi eliminat direct
    2. x are un singur urmas y, care ii va lua locul in arbore; nodul x va primi valoarea lui y, iar y va fi sters
    3. x are 2 urmasi:
        1. se va cauta cel mai mic nod y din tot arborele dreapta (x->right) al lui x; pentru asta se incepe din x si se merge prima data in dreapta, iar mai apoi doar spre stanga pana se va gasi un nod frunza (nod fara urmasi)
        2. se va inlocui valoarea lui x cu cea a nodului y si y va fi sters
    4. se va marca drumul de la radacina la nodul sters (dupa caz nodul sters poate fi x, ca la punctul 2.1, sau y, ca la punctele 2.2, 2.3)
3. Traversarea inversa a arborelui: de la nodul sters inapoi spre radacina se pastreaza acelasi drum marcat la punctul 2; pentru fiecare nod din acest drum:
    1. actualizarea inaltimilor: height = 1 + max(height(left), height(right))
    2. calcularea balansului (inaltime_left - inaltime_right)
    3. daca modulul balansului e 2 sau mai mare, se va aplica una din cele 4 rotatii discutate

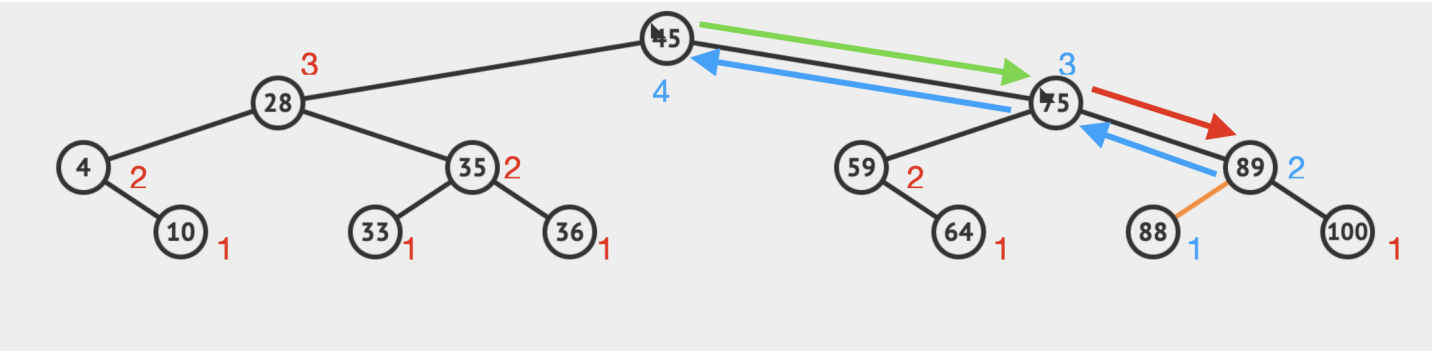


Stergerea nodului 68.

Se traverseaza arborele pana se gaseste nodul cautat. In cazul acesta el are 2 urmasi, deci se aplica cazul 2.3. Pornind de la 68 se cauta nodul cel mai mic al arborelui sau stang, in cazul acesta 75. Se traverseaza mai departe de la 68 la 75 marcand drumul si se inlocuieste cheia nodului 68 cu cea a lui 75. La final, nodul frunza 75 va fi eliminat.



Se incepe traversarea arborelui in directie inversa de la nodul eliminat pana la radacina, respectand drumul marcat anterior. Se actualizaeaza inaltimile pe acest drum si se calculeaza balansul. Ca si la inserare, daca balansul e 2 sau -2, se va aplica una din rotatii. Nodul 88 va avea balans -2, deoarece urmasul stanga e NULL si cel dreapta are inaltime 2.



Se aplica rotatie pe nodul 88 dupa care se continua parcurgerea drumului spre radacina, actualizand inltimile unde e nevoie si calculand balansul.

Rotations

T1, T2 and T3 are subtrees of the tree, rooted with y (on the left side) or x (on the right side)

```
     y                               x
    / \     Right Rotation          /  \
   x   T3   - - - - - - - - >     T1   y
  / \       < - - - - - - - -         / \
 T1  T2     Left Rotation            T2  T3
```

Keys in both of the above trees follow the following order keys(T1) < key(x) < keys(T2) < key(y) < keys(T3) So BST property is not violated anywhere.

T1, T2, T3 and T4 are subtrees.

Height of a node:

```
longest path to an ancestor leaf +1
```

OR defined recursively:

```
if node is leaf:
    height=1;
else
    height = 1 + max(height(left), height(right))
```

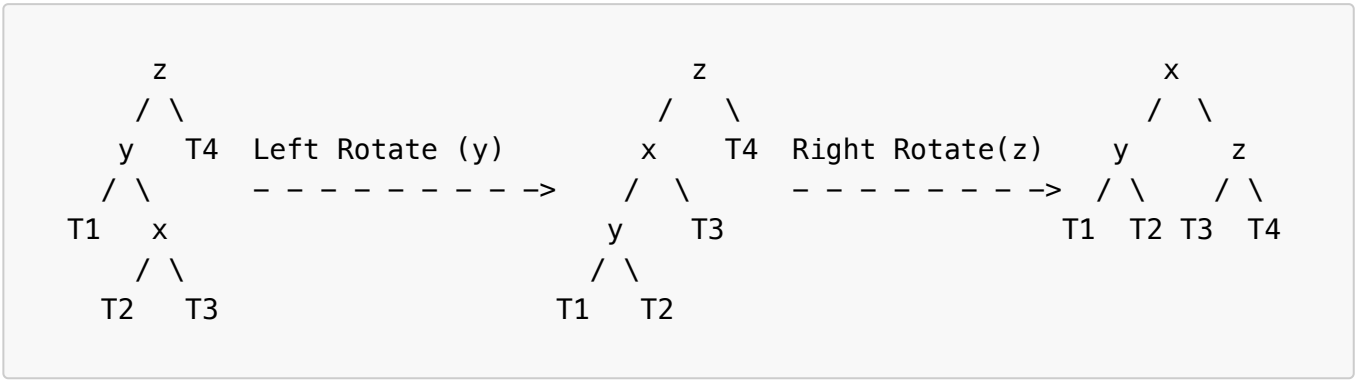
Balance of a node:

```
height(left) - height(right)
```

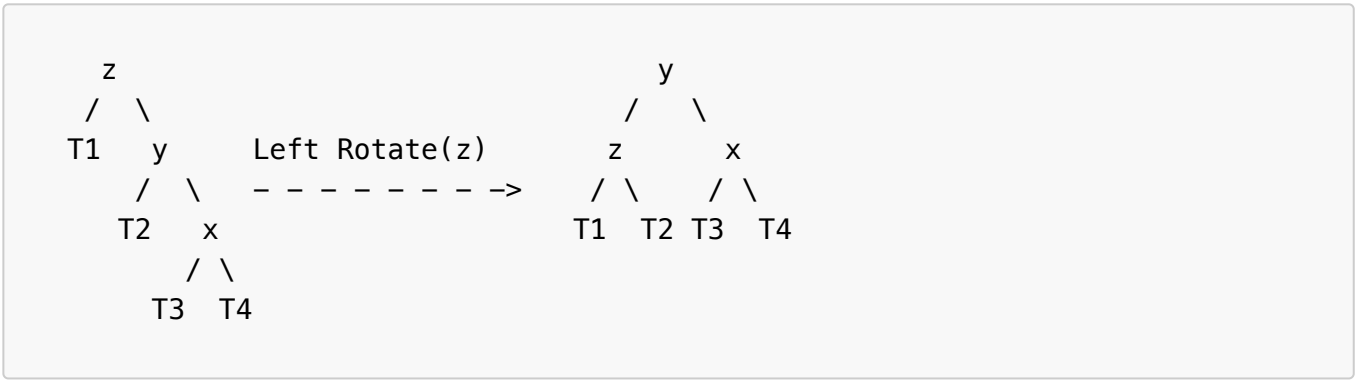
1. Left Left Case balance(z) == 2 and balance(y) >= 0

```
          z                                      y
         / \                                   /   \
        y   T4      Right Rotate (z)          x      z
       / \          - - - - - - - - ->      /  \    /  \
      x   T3                               T1  T2  T3  T4
     / \
   T1   T2
```

2. Left Right Case balance(z) == 2 and balance(y) < 0



3. Right Right Case balance(z) == -2 and balance(y) <= 0



4. Right Left Case balance(z) == -2 and balance(y) > 0