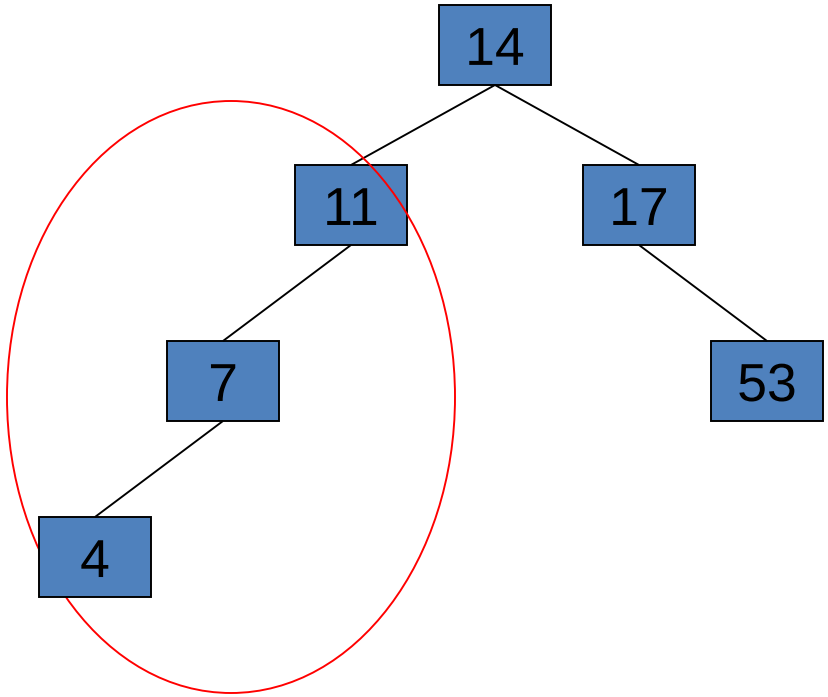
AVL TREE

.:: EJERCICIOS ::.

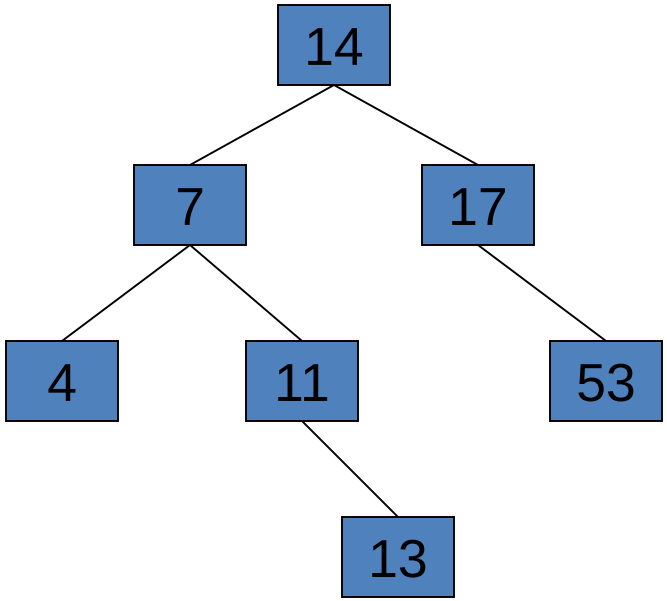
AVL Tree Example:

- **Insert 14, 17, 11, 7, 53, 4, 13 into an empty AVL tree**



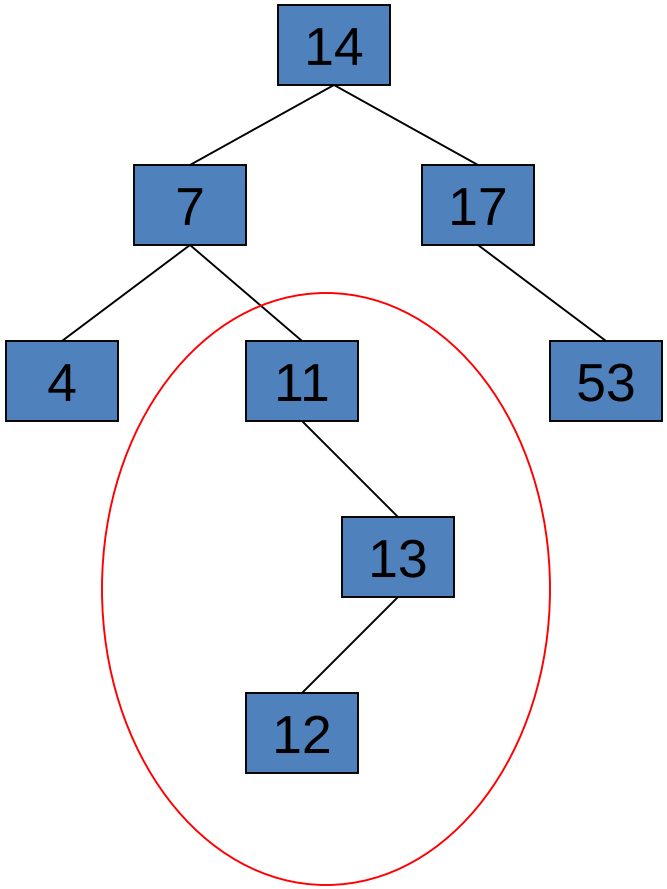
AVL Tree Example:

- **Insert 14, 17, 11, 7, 53, 4, 13 into an empty AVL tree**



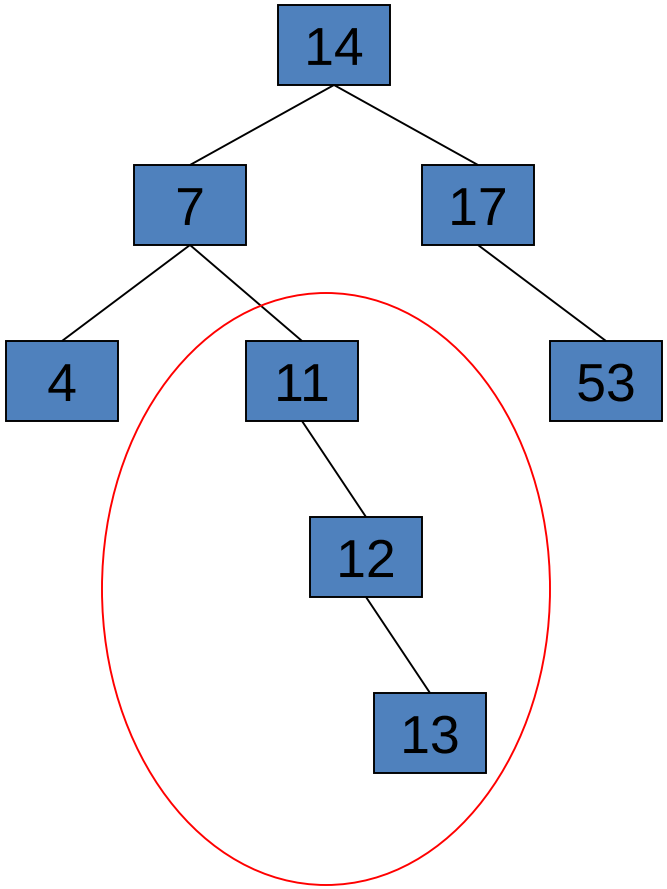
AVL Tree Example:

- **Now insert 12**



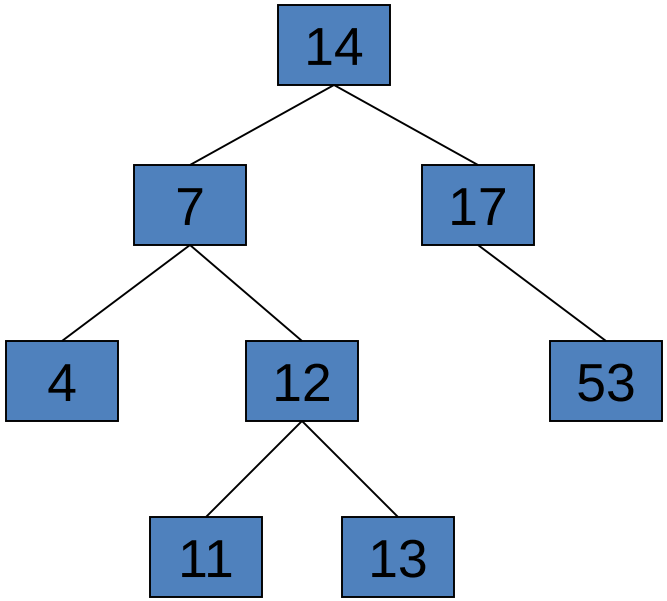
AVL Tree Example:

- **Now insert 12**



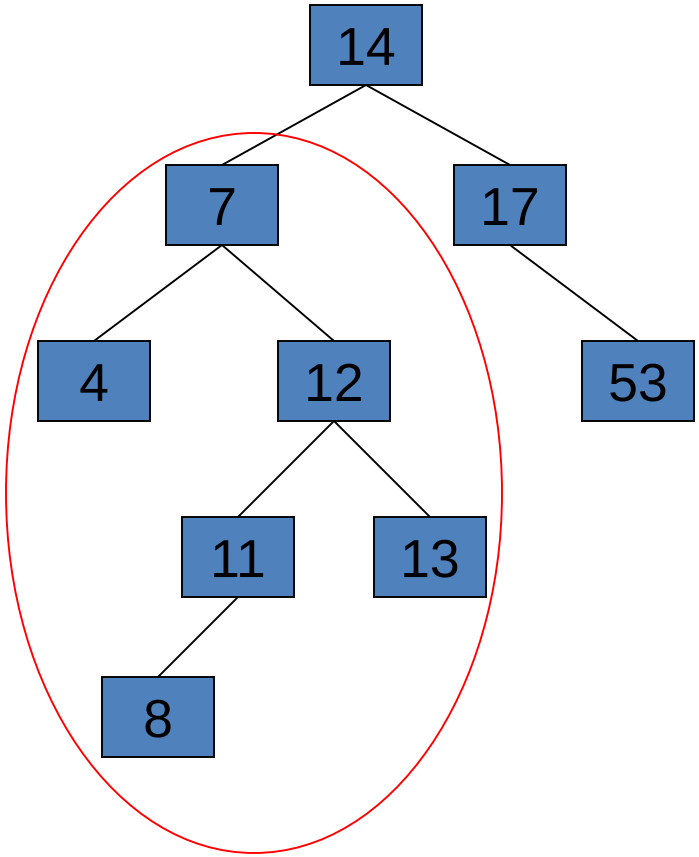
AVL Tree Example:

- **Now the AVL tree is balanced.**



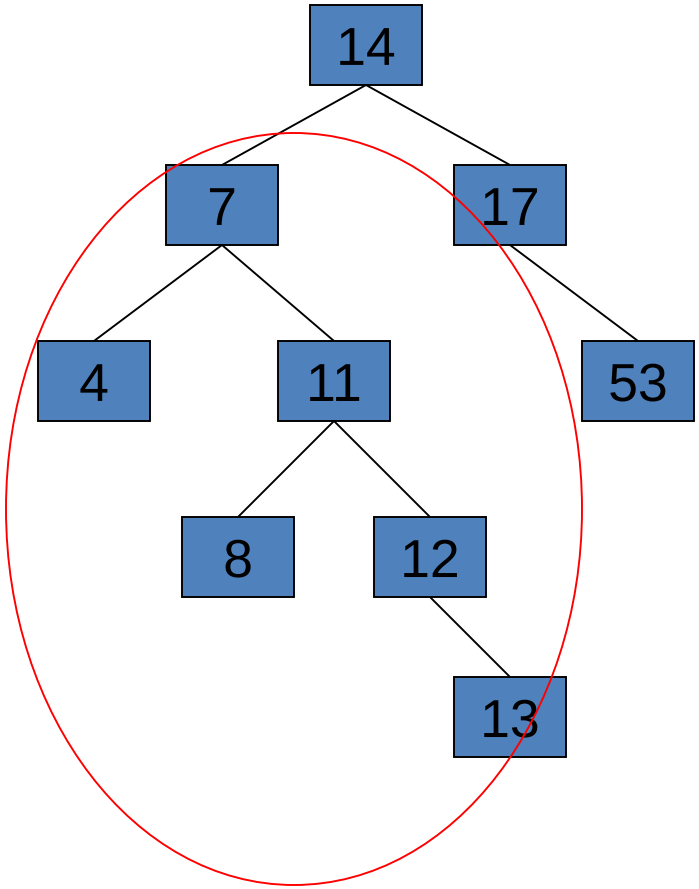
AVL Tree Example:

- **Now insert 8**



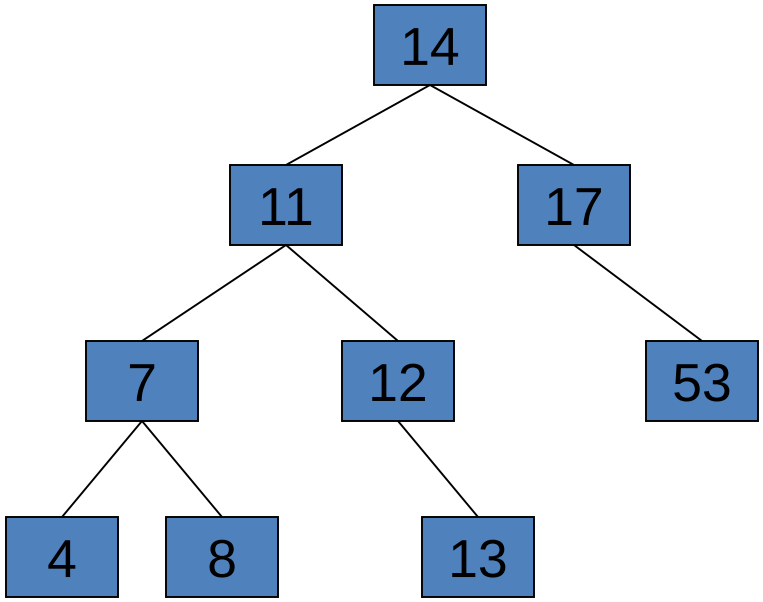
AVL Tree Example:

- **Now insert 8**



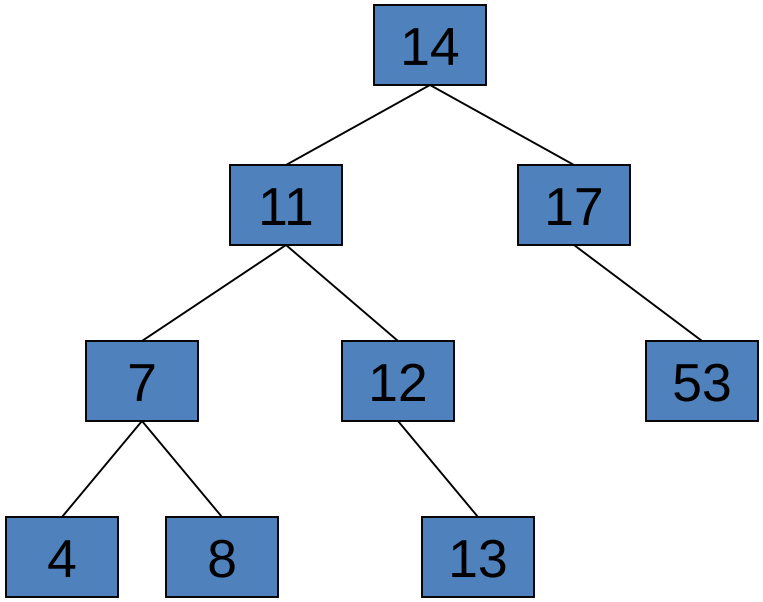
AVL Tree Example:

- **Now the AVL tree is balanced.**



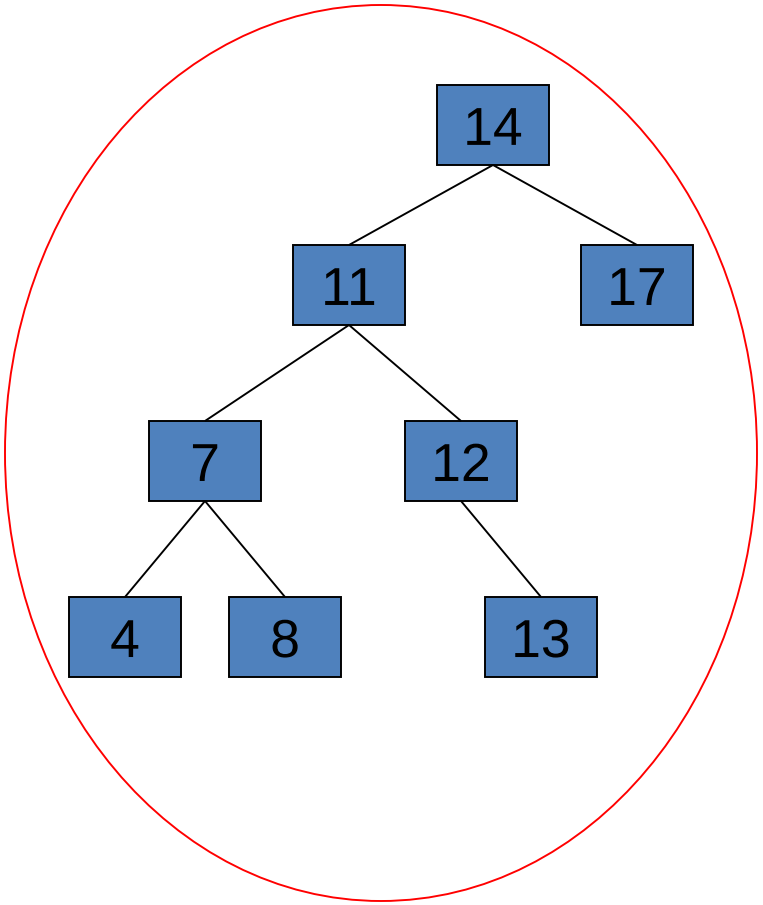
AVL Tree Example:

- **Now remove 53**



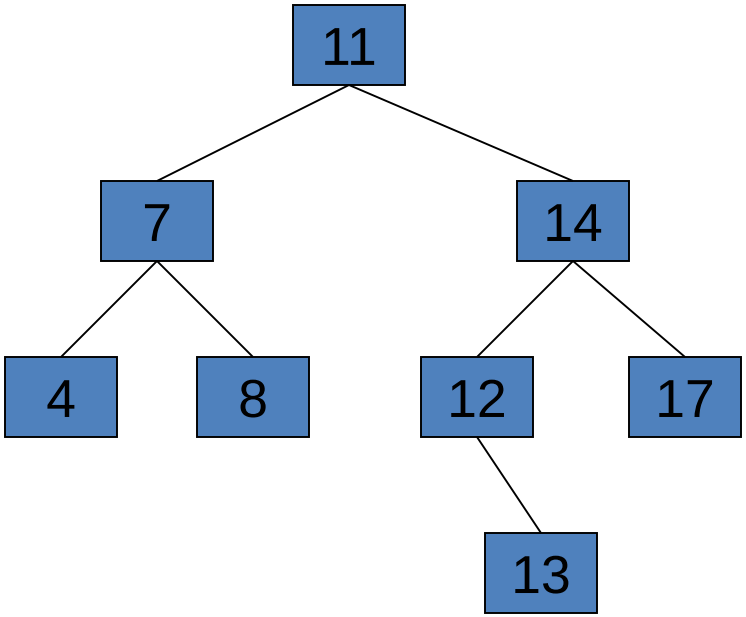
AVL Tree Example:

- **Now remove 53, unbalanced**



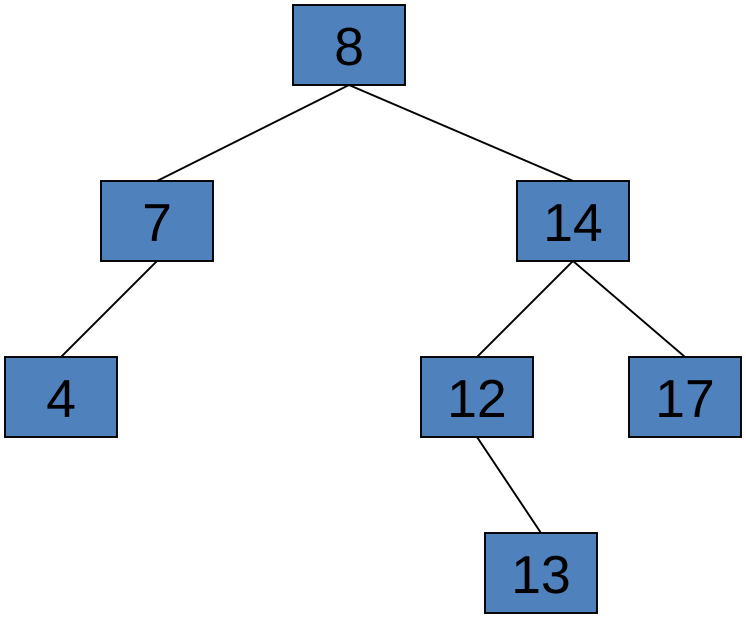
AVL Tree Example:

- **Balanced! Remove 11**



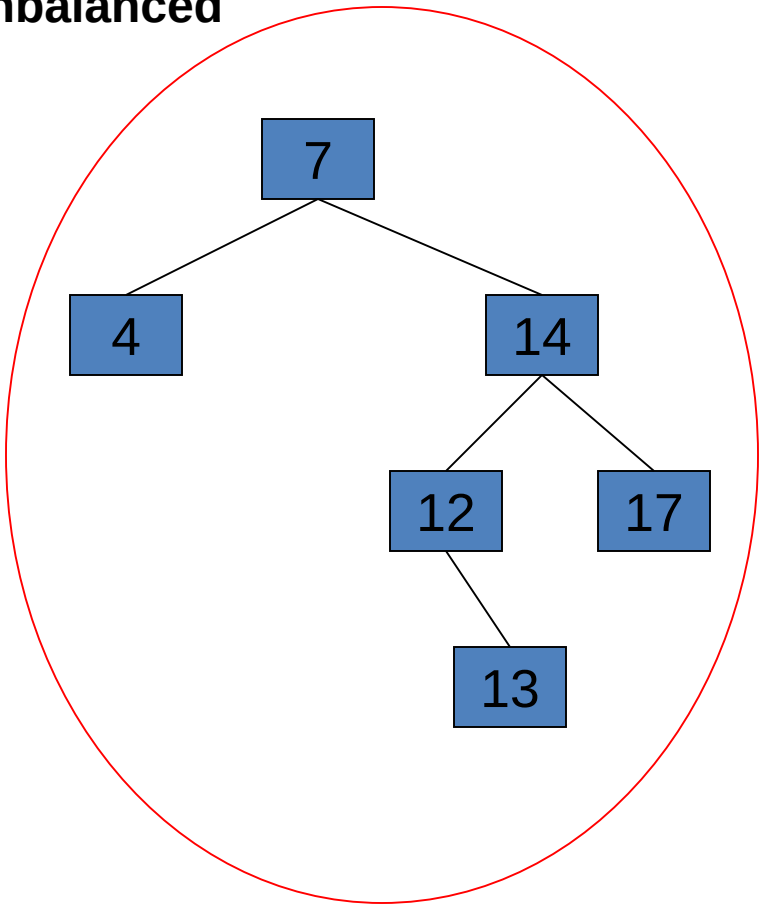
AVL Tree Example:

- **Remove 11, replace it with the largest in its left branch**



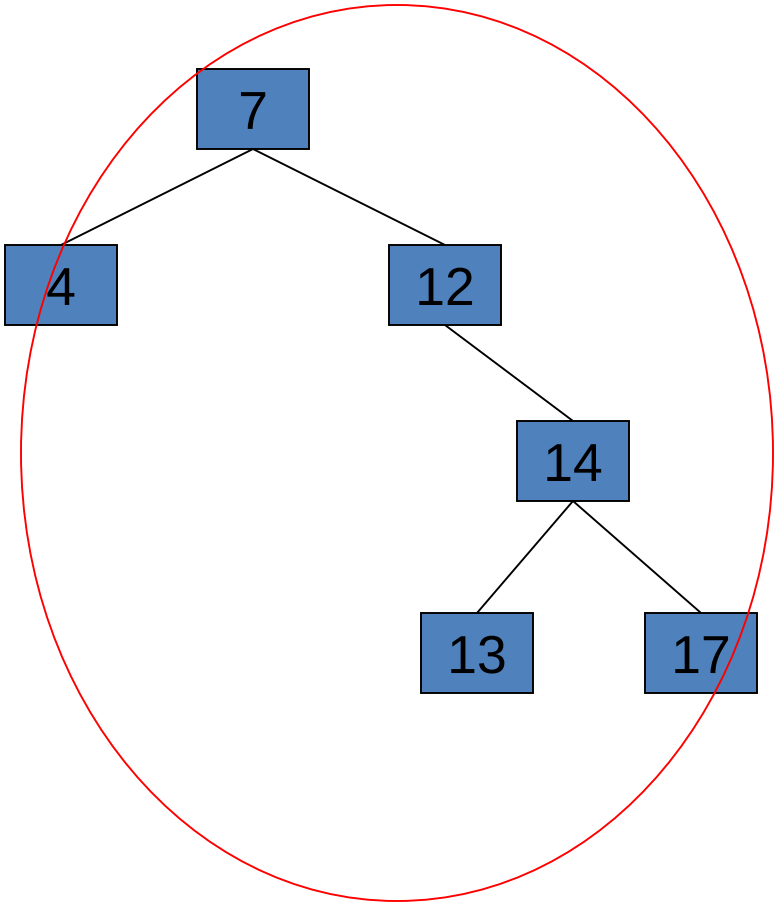
AVL Tree Example:

- **Remove 8, unbalanced**



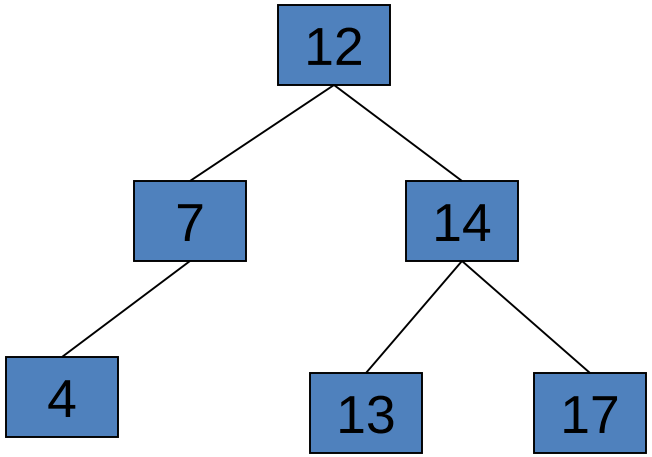
AVL Tree Example:

- **Remove 8, unbalanced**



AVL Tree Example:

- **Balanced!!**

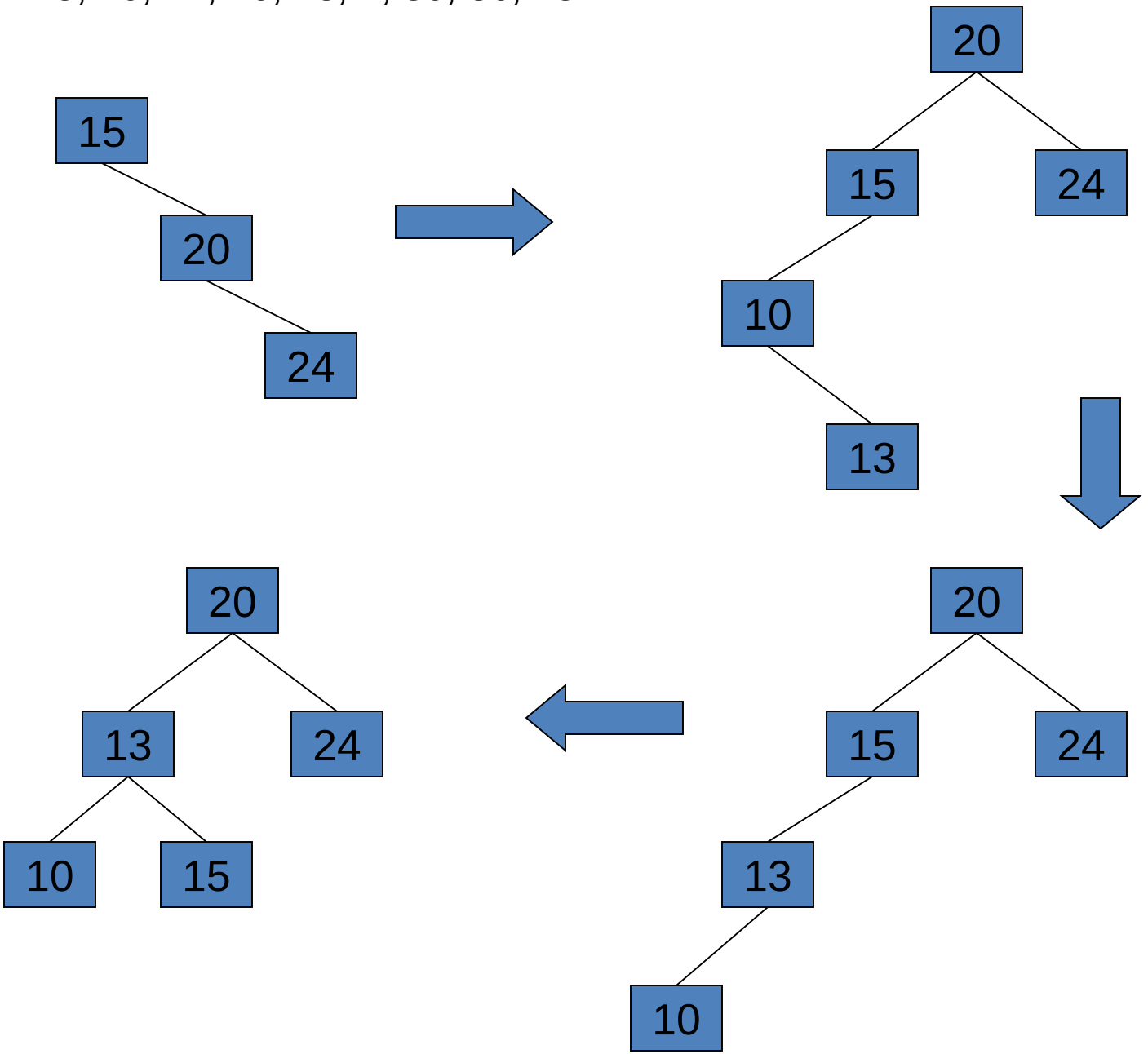


In Class Exercises

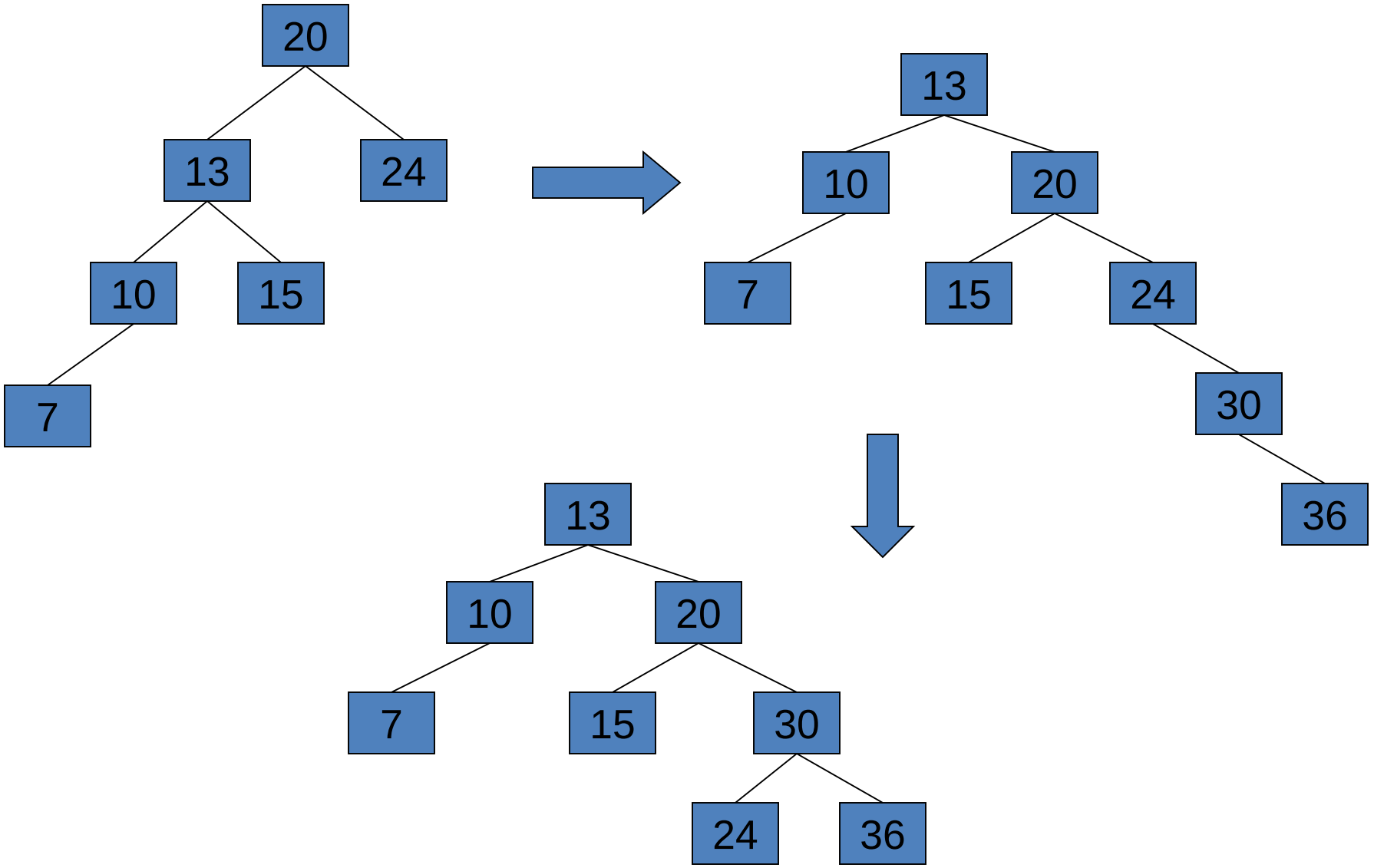
- Build an AVL tree with the following values:

  15, 20, 24, 10, 13, 7, 30, 36, 25

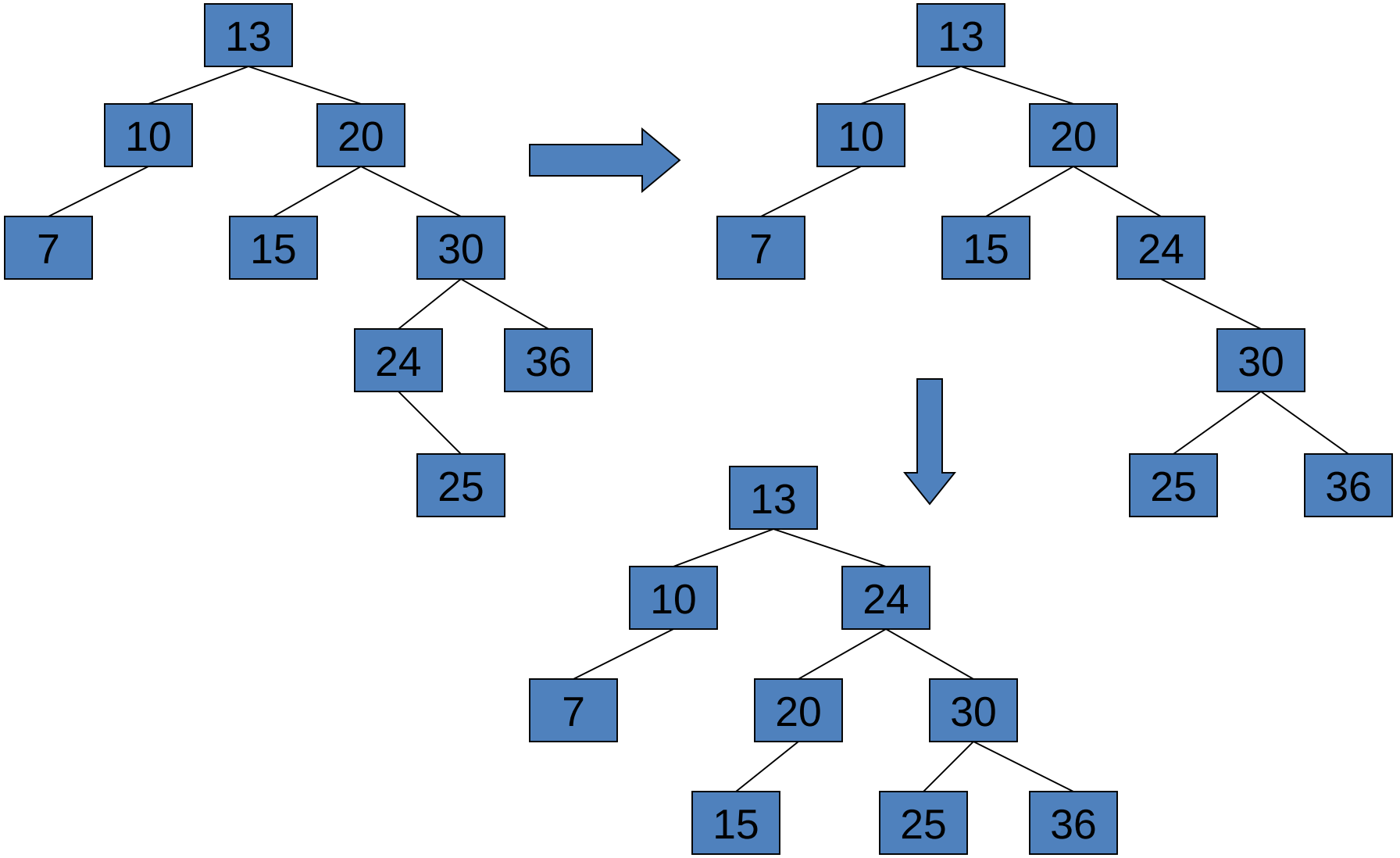
15, 20, 24, 10, 13, 7, 30, 36, 25



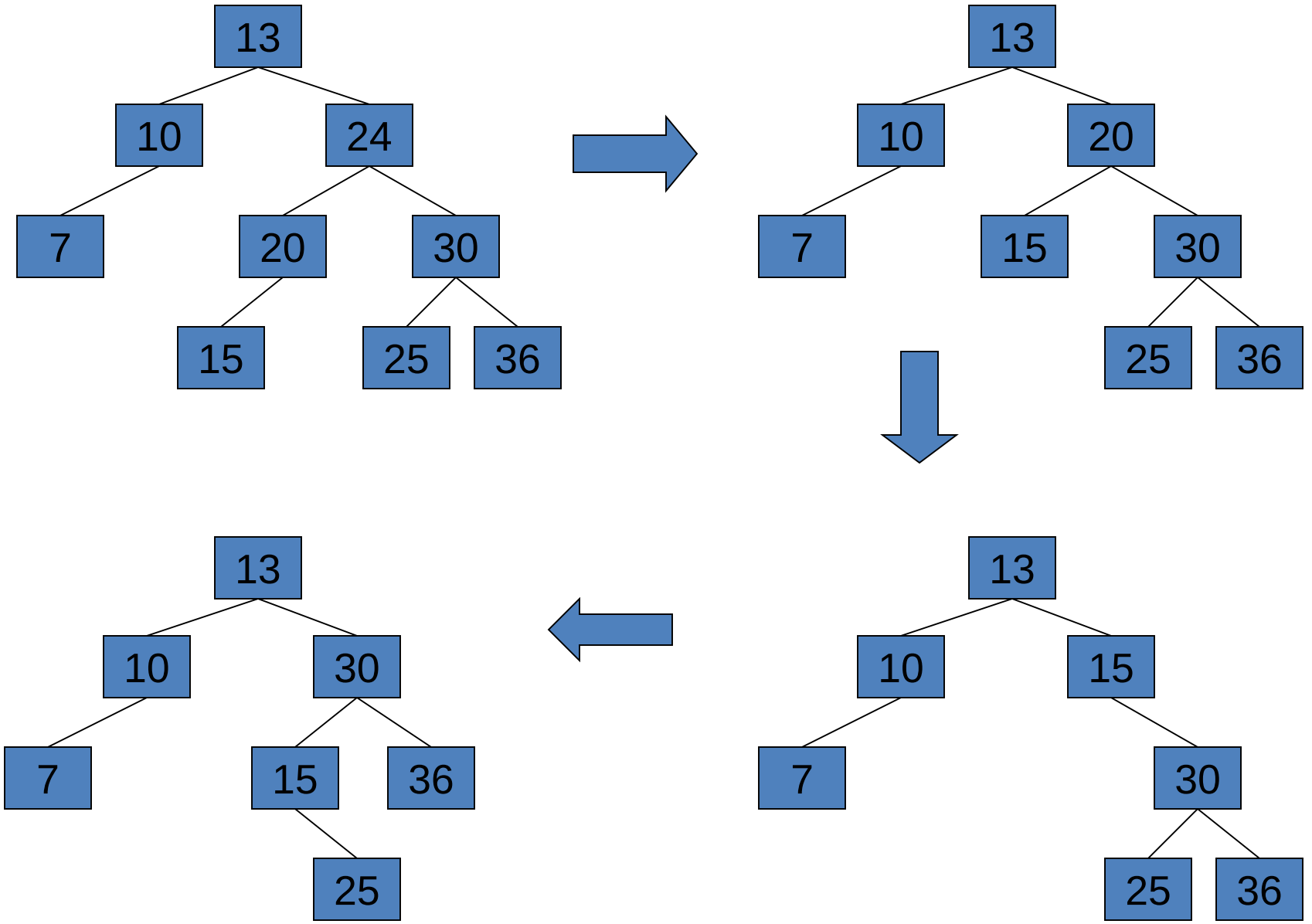
15, 20, 24, 10, 13, 7, 30, 36, 25



15, 20, 24, 10, 13, 7, 30, 36, 25



Remove 24 and 20 from the AVL tree.



AVL TREE

.:: IMPLEMENTATIONS - AVLSolution2::.

```
1   #ifndef __AVL_H__
2   #define __AVL_H__
3
4   #include <functional>
5   #include <algorithm>
6
7   template <typename T, typename R=T>
8   class AVLTree {
9       struct Node {
24
25      Node*           root;
26      int             length;
27      std::function<R(T)>  key;
28
29      void destroy(Node* n) {
30          if (n != nullptr) {
35      }
36      void rotAB(Node*& n) {
44      void rotBA(Node*& n) {
52      void balance(Node*& n) {
66      void add(Node*& n, T e) {
78  public:
79      AVLTree(std::function<R(T)> key = [](T a) { return a; })
80          : root(nullptr), length(0), key(key) {}
81      ~AVLTree() { destroy(root); }
82
83      int Height() {
86      int Size() {
89      void Add(T e) {
93
94  };
95
96  #endif
```

```
 7  template <typename T, typename R=T>
 8  class AVLTree {
 9      struct Node {
10          T     e;
11          Node* l;
12          Node* r;
13          int   h;
14
15          Node(T e) : e(e), l(nullptr), r(nullptr), h(0) {}
16
17          static int height(Node* n) {
18              return n == nullptr? -1 : n->h;
19          }
20          void updateH() {
21              h = std::max(Node::height(l), Node::height(r)) + 1;
22          }
23      };
```

```
 7  template <typename T, typename R=T>
 8  class AVLTree {
 9      struct Node {
24
25      Node*                   root;
26      int                     length;
27      std::function<R(T)>   key;
28
29      void destroy(Node* n)  {
30          if (n != nullptr)  {
35      }
36      void rotAB(Node*& n)  {
37          Node* aux = n->l;
38          n->l = aux->r;
39          n->updateH();
40          aux->r = n;
41          n = aux;
42          n->updateH();
43      }
44      void rotBA(Node*& n)  {
45          Node* aux = n->r;
46          n->r = aux->l;
47          n->updateH();
48          aux->l = n;
49          n= aux;
50          n->updateH();
```

```
52      void balance(Node*& n) {
53          int delta = Node::height(n->l) - Node::height(n->r);
54          if (delta < -1) {
55              if (Node::height(n->r->l) > Node::height(n->r->r)) {
56                  rotAB(n->r);
57              }
58              rotBA(n);
59          } else if (delta > 1) {
60              if (Node::height(n->l->r) > Node::height(n->l->l)) {
61                  rotBA(n->l);
62              }
63              rotAB(n);
64          }
65      }
66      void add(Node*& n, T e) {
67          if (n == nullptr) {
68              n = new Node(e);
69              return;
70          } else if (key(e) < key(n->e)) {
71              add(n->l, e);
72          } else {
73              add(n->r, e);
74          }
75          balance(n);
76          n->updateH();
77      }
78  public:
79      AVLTree(std::function<R(T)> key = [](T a) { return a; })
80          : root(nullptr), length(0), key(key) {}
81      ~AVLTree() { destroy(root); }
```

```
66      void add(Node*& n, T e) {
67          if (n == nullptr) {
68              n = new Node(e);
69              return;
70          } else if (key(e) < key(n->e)) {
71              add(n->l, e);
72          } else {
73              add(n->r, e);
74          }
75          balance(n);
76          n->updateH();
77      }
78  public:
79      AVLTree(std::function<R(T)> key = [](T a) { return a; })
80          : root(nullptr), length(0), key(key) {}
81      ~AVLTree() { destroy(root); }
82
83      int Height() {
84          return Node::height(root);
85      }
86      int Size() {
87          return length;
88      }
89      void Add(T e) {
90          add(root, e);
91          ++length;
92      }
```

```cpp
#include <iostream>
#include "avl.hpp"

using namespace std;

int main() {
    AVLTree<int>* t = new AVLTree<int>();

    for (int i = 0; i < 1e6; ++i) {
        t->Add(i);
    }

    cout << t->Height() << endl;

    delete t;
    return 0;
}

```