Map \ multimap \ set \ multiset are all implemented as a red-black trees.

A **red–black tree** is a kind of self-balancing binary search tree. Standard binary search tree does not do rebalancing, which makes it not suitable for worst case scenarios.

To understand why, an example speaks a 1000 words. Visit these sites:

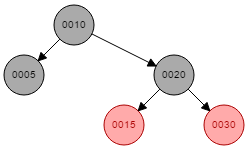
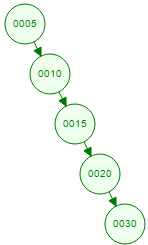
<https://www.cs.usfca.edu/~galles/visualization/RedBlack.html>

<https://www.cs.usfca.edu/~galles/visualization/BST.html>

For both of them, input numbers: [5, 10, 15, 20, 30]

This is how they should look:

Red-Black Tree: Standard Binary Search Tree:

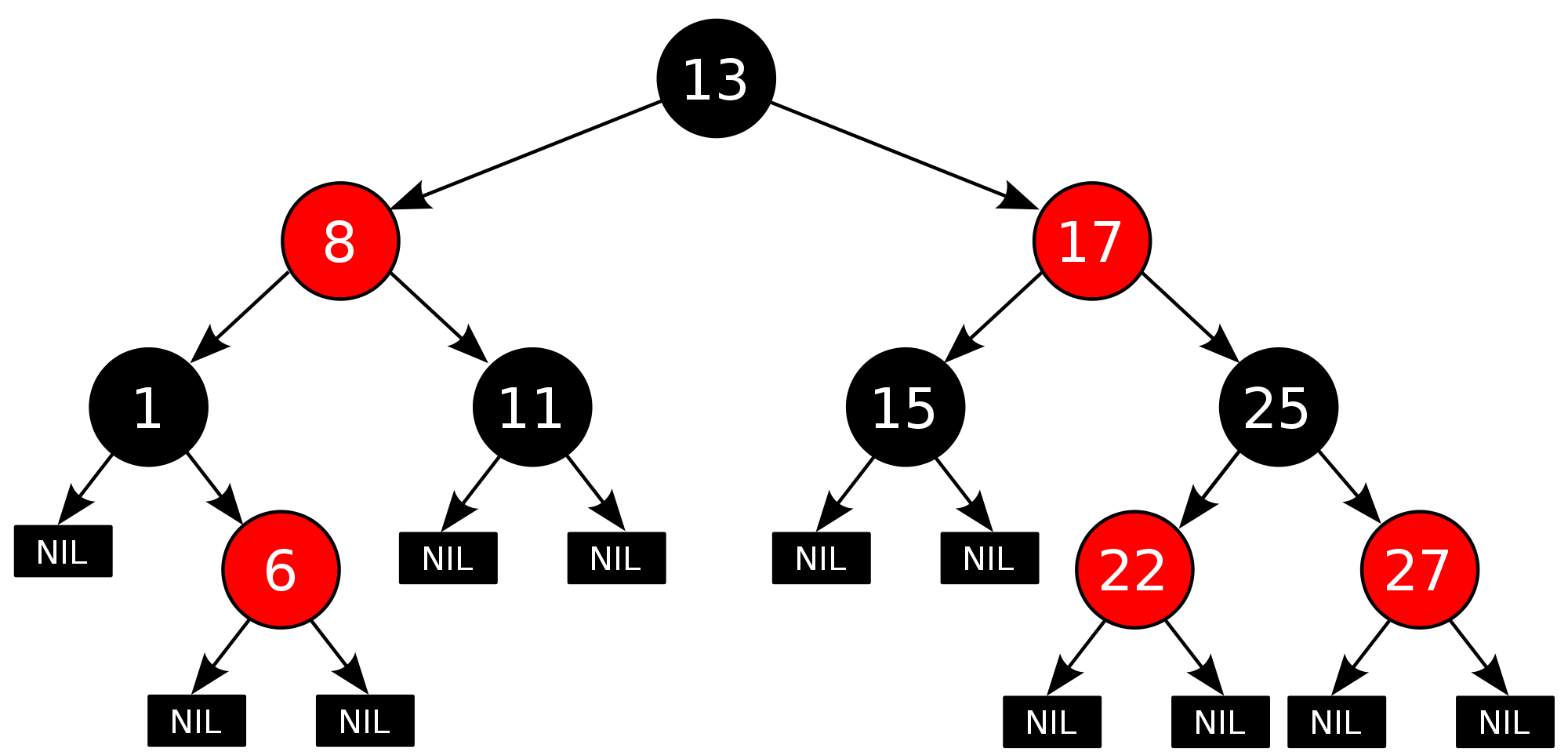
 

Consider look-up performance for value 30.

If we want to find the element that happens to be at the end of standard binary tree, it is **O(n)**. We have to search the whole tree.

For red-black tree, worst case lookup is **O(log n)**. That is much better.

Example of red-black tree:



A red-black tree with ***n*** internal nodes has height at most 2**log(*n*+1)**.

The balancing of the tree is not perfect, but it is good enough to guarantee searching in [O(log *n*)](https://en.wikipedia.org/wiki/Big-O_notation) time, where *n* is the total number of elements in the tree.

During insert and delete operations, nodes may be rotated to maintain tree balance.

Because std::map is implemented as a binary search tree, it sorts every element we put into it! Sometimes, in code we want to have mapping between key and value, but we may not need the sorting part. In that case, consider other containers (like std::unordered\_map).