A **red-black tree** is a binary search tree with one extra bit of storage per node: its **color**, which can be either RED or BLACK. By constraining the node colors on any simple path from the root to a leaf, red-black trees ensure that no such path is more than twice as long as any other, so that the tree is approximately **balanced**.

Each node of the tree now contains the attributes **color**, **key**, **left**, **right**, and **p**. If a child or the parent of a node does not exist, the corresponding pointer attribute of the node contains the value NIL. We shall regard these NILs as being pointers to leaves (external nodes) of the binary search tree and the normal, key-bearing nodes as being internal nodes of the tree.

A red-black tree is a binary tree that satisfies the following **red-black properties**:

1. Every node is either red or black.
2. The root is black.
3. Every leaf (NIL) is black. **NILs are considered to be black.**
4. If a node is red, then both its children are black. **(or both children are NIL)**
5. For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

\[ \text{If a node is red, its parent is black.} \]
Some illegal Red-Black Trees
Balance Lemma

We call the number of black nodes on any simple path from, but not including, a node \( x \) down to a leaf the **black-height** of the node, denoted \( bh(x) \). By property 5, the notion of black-height is well defined, since all descending simple paths from the node have the same number of black nodes. We define the black-height of a red-black tree to be the black-height of its root.

The following lemma shows why red-black trees make good search trees.

**Lemma 13.1**
A red-black tree with \( n \) internal nodes has height at most \( 2 \lg(n + 1) \).

An unbalanced binary search tree could have height \( n - 1 \).
Tree as unbalanced as possible
but still satisfy red-black properties

\[ f(h) = \begin{align*}
1 & \rightarrow 2 \\
3 & \rightarrow 3 \\
5 & \rightarrow 4 \\
7 & \rightarrow 5 \\
9 & \rightarrow 6
\end{align*} \]

\[ \frac{h}{2} + 1 = \frac{h}{2} + 3 \]

Minimum number of nodes:

\[ \frac{h}{2} = 2^2 - 2 \]
\[ 6 = 8 - 2 = 2^3 - 2 \]
\[ 14 = 16 - 2 = 2^4 - 2 \]
\[ 14 + 16 = 30 = 2^5 - 2 \]

Guess: \[ 30 + 32 = 62 = 2^6 - 2 \]

\[ f^{-1}(n) \]

\[ \frac{(h+3)/2}{2} - 2 \]
\[ h = \frac{(n+3)^{1/2} - 2}{2} \]

if \( n = \frac{(n+3)^{1/2}}{2} \)

\[ n+2 = \frac{(n+3)^{1/2}}{2} \]

\[ \log(n+2) = \frac{n+3}{2} \]

\[ 2 \log(n+2) = h + 3 \]

\[ 2n \log(n+2) - 3 = h \]

h must be at most 2 \[ 2 \log(n+2) - 3 \leq h \]

for n nodes.