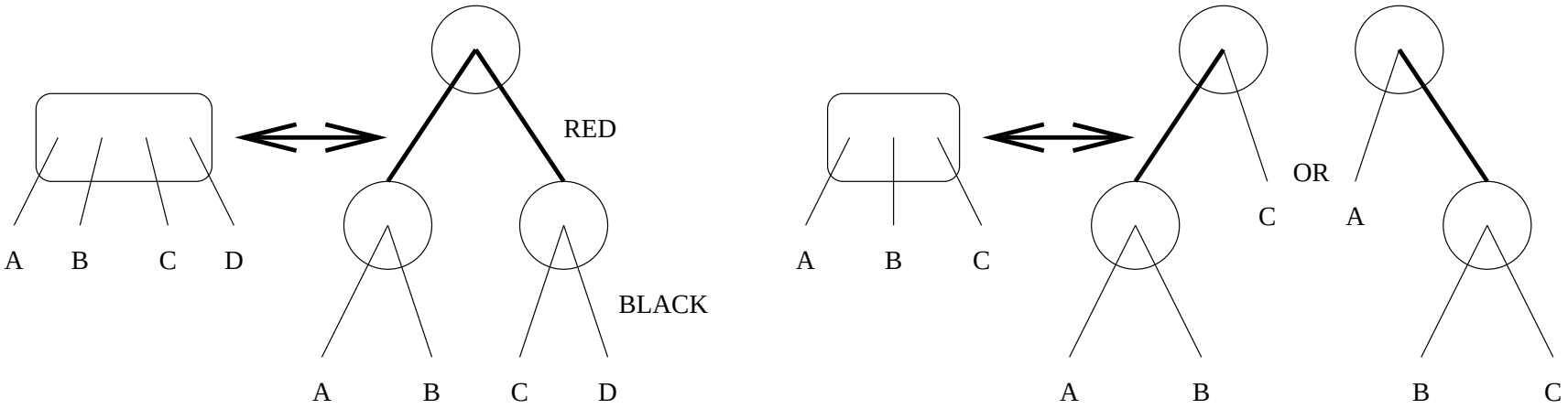
Overview

- Red-Black Trees

Red-Black Trees

- Represent $2-3-4$ trees as standard binary trees (2-nodes only) by using one extra *bit* per node.
  - Idea: represent 3- and 4-nodes as small binary trees bound together by "red" links.
    - The "black" links bound the nodes of the original $2-3-4$ tree together.



Red-Black Trees (cont'd)

- A red-black tree satisfies the following properties:

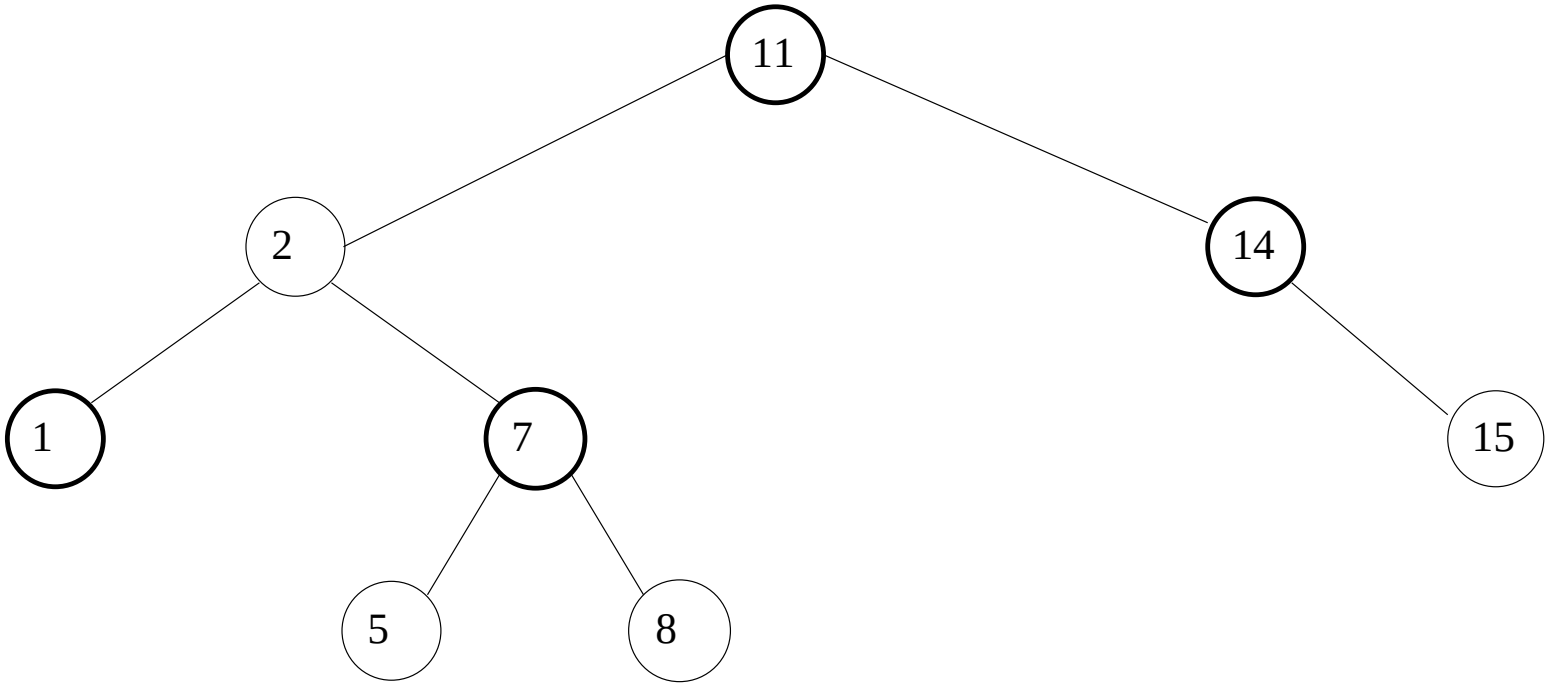
  1. Every node is either red or black, according to the color of the link from its parent.
  2. Every empty node/tree is black.
  3. If a node is red, both its children are black.
  4. Every path from a node to a descendant empty node contains the same number of black nodes.

- By restricting the way nodes are colored on any path from the root to an empty node, these properties ensure that no such path is more than twice as long as any other.

  – Thus, the tree is approximately balanced.The above properties insure that

Red-Black Trees: An Example

- In the following diagram, black nodes are drawn with a thick circle, and red nodes are drawn with a thin circle. I omit the empty (black) nodes.



Red-Black Trees: Find

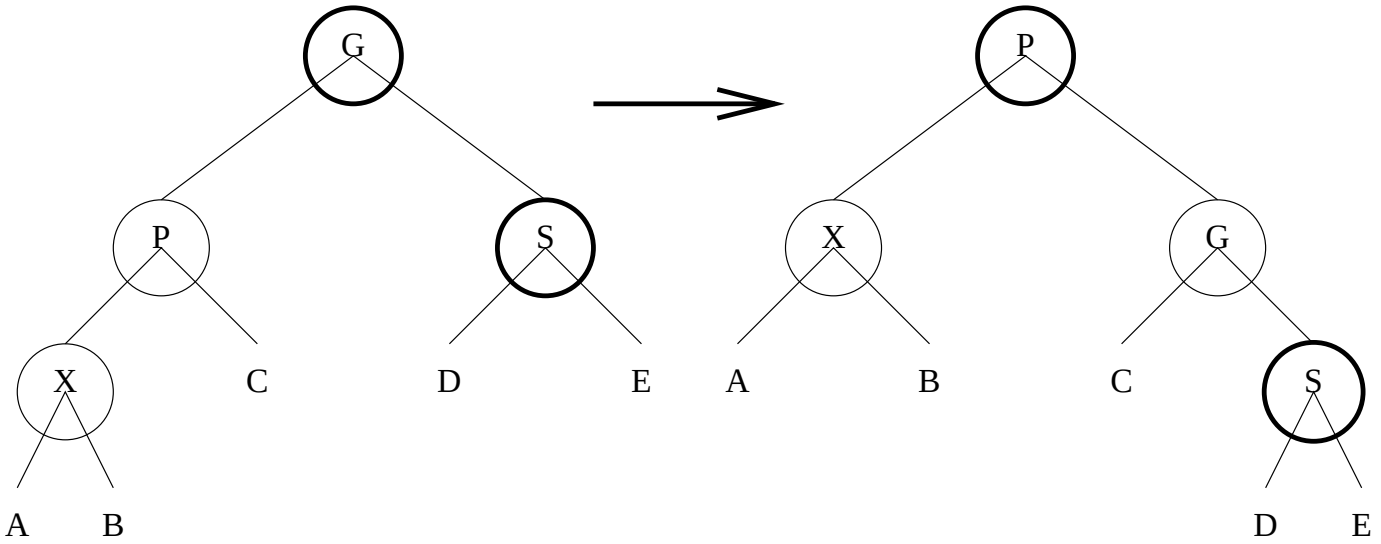
- Use the (ordinary) BST `find()`!

Red-Black Trees: Insert

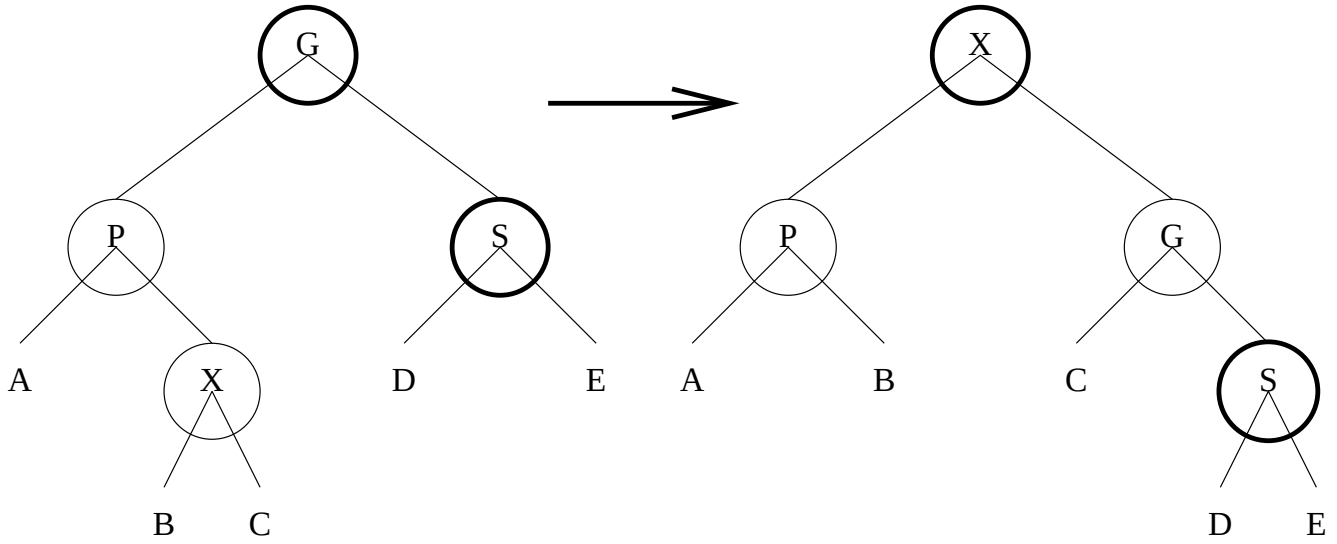
- On the way down, when we see a node X that has two red children, we make X red and its children black.
  - Thus, the number of black nodes on the paths below X remain unchanged.
  - However, if X's parent is red, this introduces two consecutive red nodes.
    - A single or double rotation can eliminate the consecutive red nodes.

Red-Black Trees: Insert (cont'd)

- Single rotation + Coloring changes
  - Goal: eliminate consecutive red nodes without changing the number of black nodes along any path.



  - Why can't S be red?

Red-Black Trees: Insert (cont'd)

- Double rotation + Coloring changes
  - Goal: eliminate consecutive red nodes without changing the number of black nodes along any path.