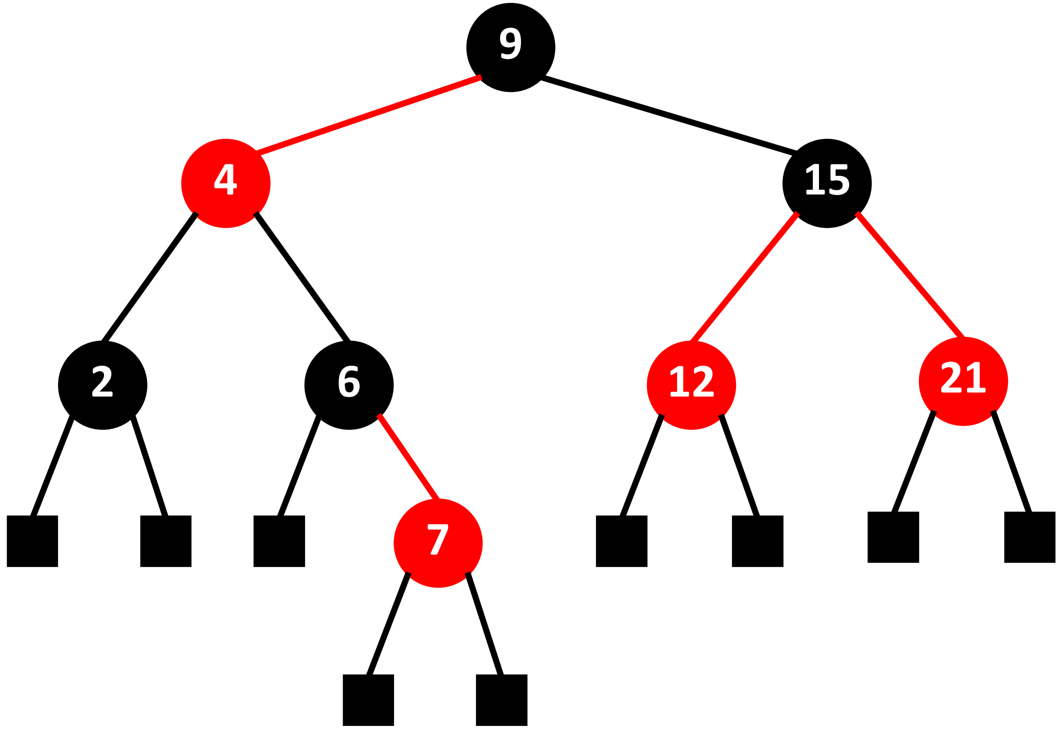
Red-Black Trees

Red-Black Trees

- A **red-black tree** is a **binary search tree** that satisfies the following properties:
  - **root property:** the **root** is **black**
  - **external property:** every **leaf** is **black**
  - **internal property:** the **children** of a **red** node are **black**
  - **depth property:** all the **leaves** have the same **black depth**

Red-Black Trees

Example:



From (2,4) to Red-Black Trees

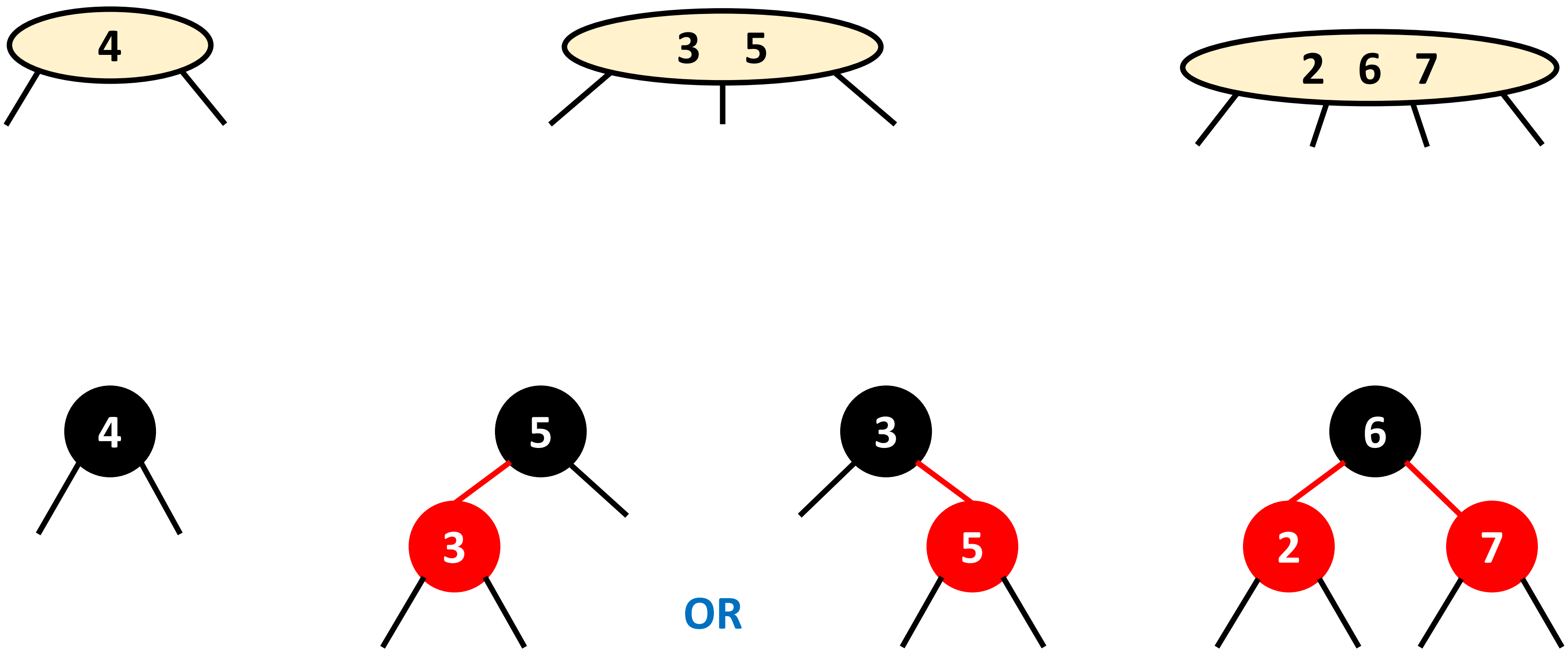
- A **red-black tree** is a representation of a (2,4) tree by means of a **binary tree** whose nodes are colored **red** or **black**

- In comparison with its associated (2,4) tree, a red-black tree has

  - **same logarithmic time performance**

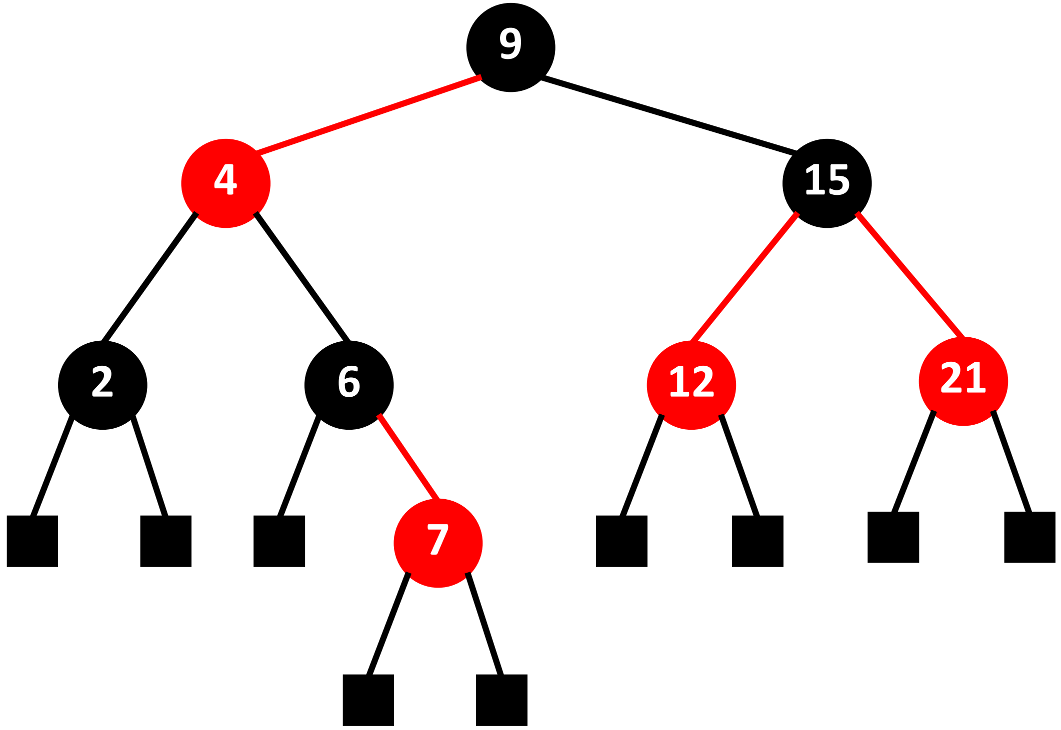
  - **simpler implementation with a single node type**

From (2,4) to Red-Black Trees



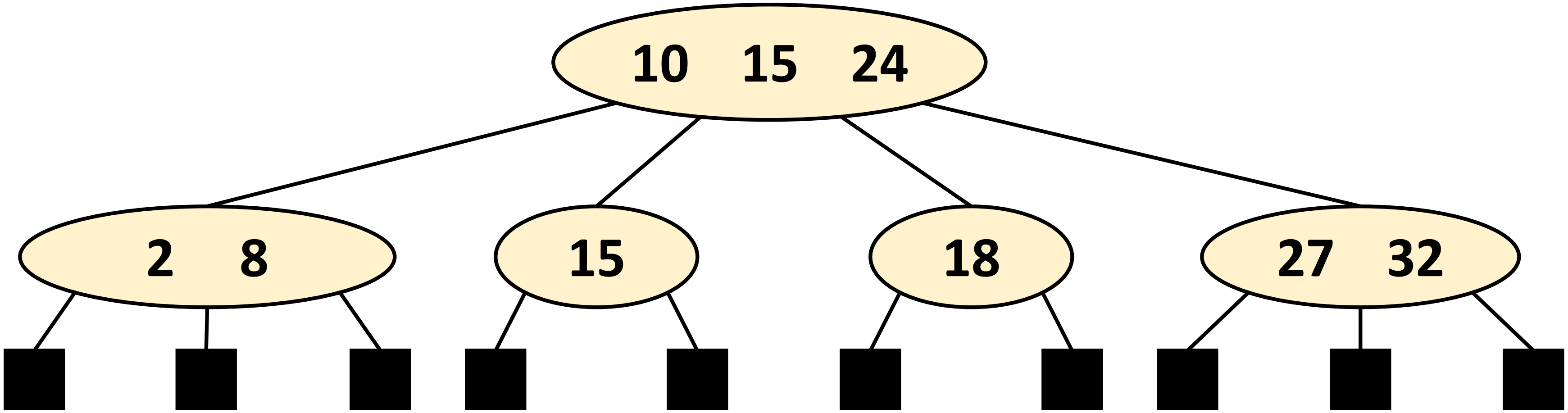
Red-Black Tree to (2,4) Tree

Example:



(2,4) Tree to Red-Black Tree

Example:



Red-Black Trees

Theorem: A red-black tree storing n items has height $O(\log n)$.

Proof:

- The **height** of a red-black tree is **at most twice** the height of its associated (2,4) tree, which is $O(\log n)$

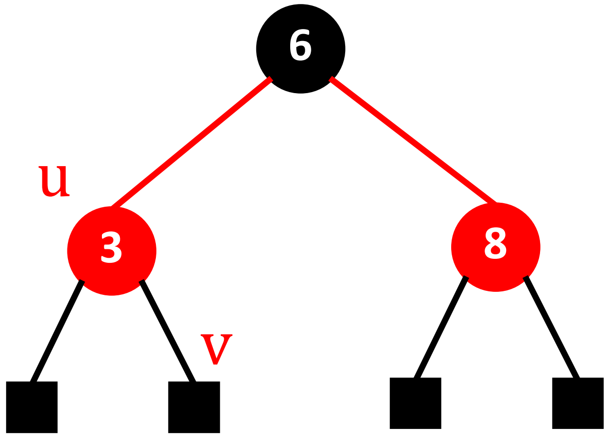
- The **search algorithm** for a **red-black tree** is the same as that for a **binary search tree**
- By the above theorem, **searching** in a red-black tree takes $O(\log n)$ time

Insertion

- To **insert** k, we execute the insertion algorithm for binary search trees and color **red** the newly inserted node v unless it is the root:
  - we preserve the **root**, **external**, and **depth properties**
  - if the **parent** u of v is **black**, we also preserve the **internal property** and we are done
  - else (u is **red**) we have a **double red** (i.e., a **violation of the internal property**), which requires a **reorganization of the tree**

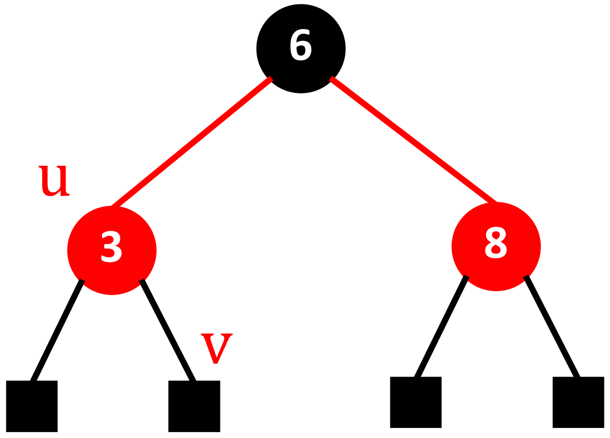
Insertion

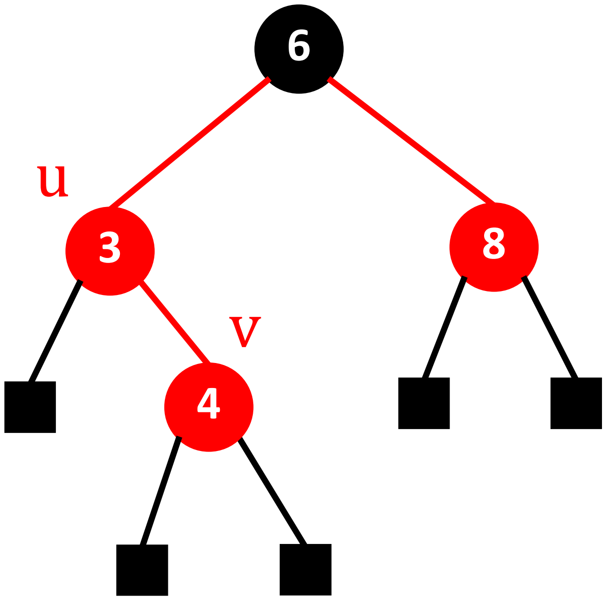
Example: insert 4



Insertion

Example: insert 4



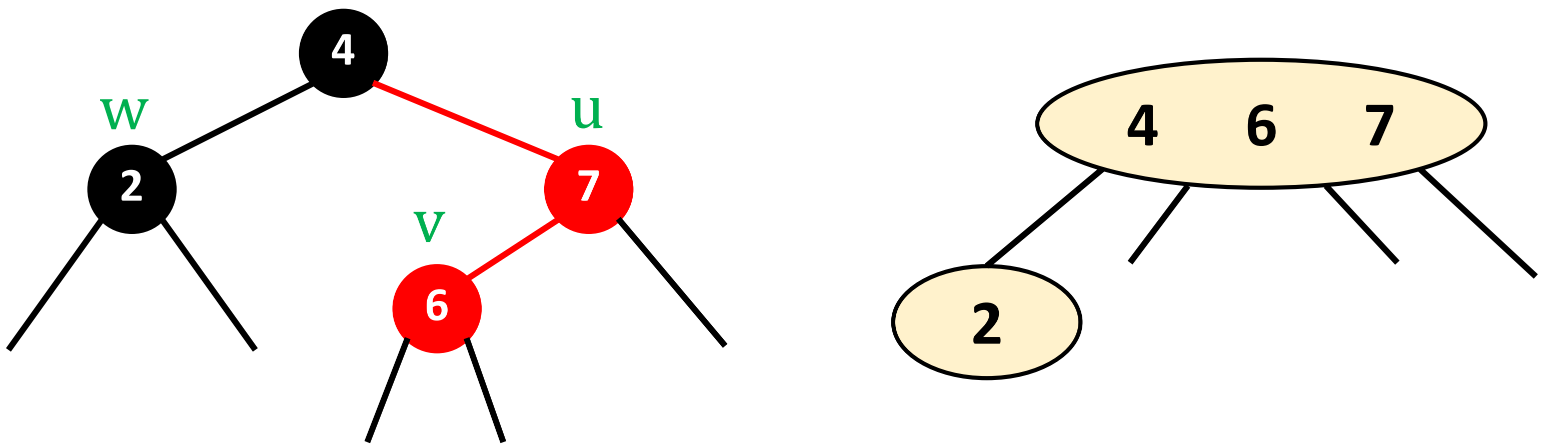


Remedying Double Red

Consider a **double red** with child v and parent u, and let w be the sibling of u.

Case 1: w is **black:**

- the double red is an incorrect replacement of 4-node
- **restructuring:** change the 4-node replacement

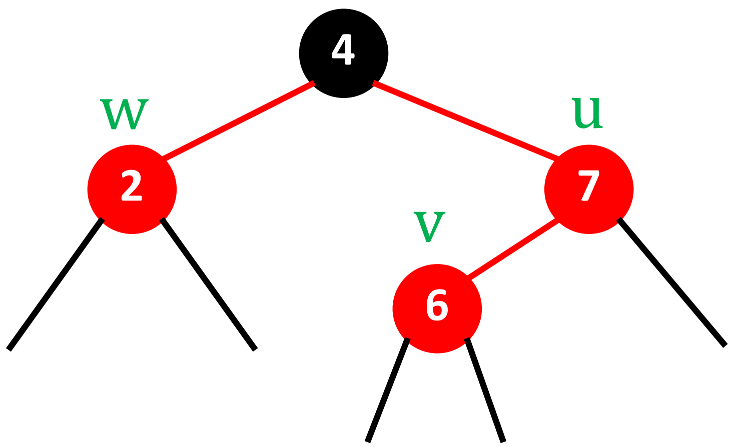


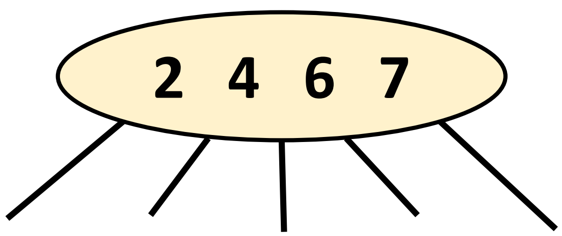
Remedying Double Red

Consider a **double red** with child v and parent u, and let w be the sibling of u.

Case 2: w is **red:**

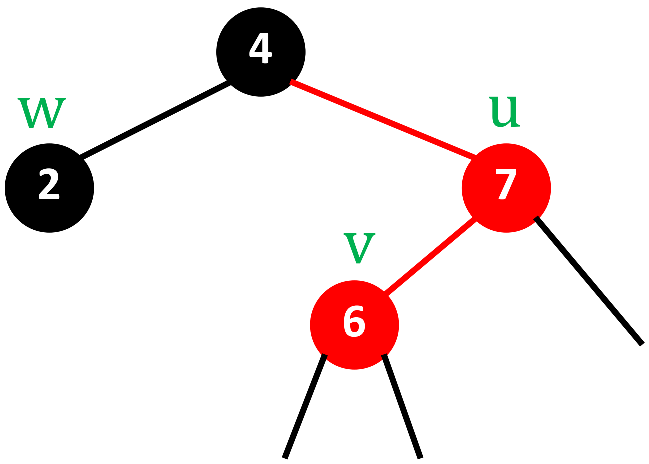
- the double red corresponds to an **overflow**
- **recoloring:** perform the equivalent of a split





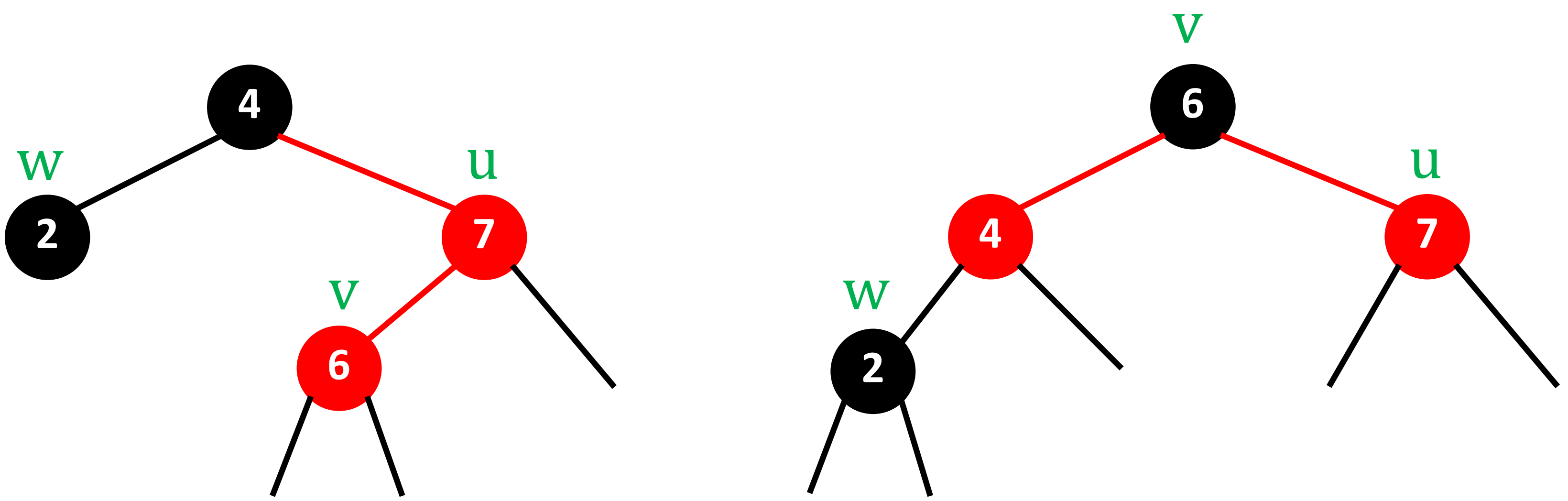
Restructuring

- A restructuring remedies a **child-parent double red** when the **parent red** node has a **black sibling**
- The **internal property** is restored and the **other properties are preserved**



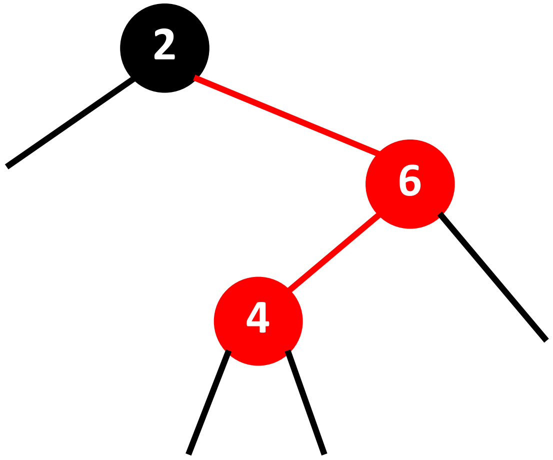
Restructuring

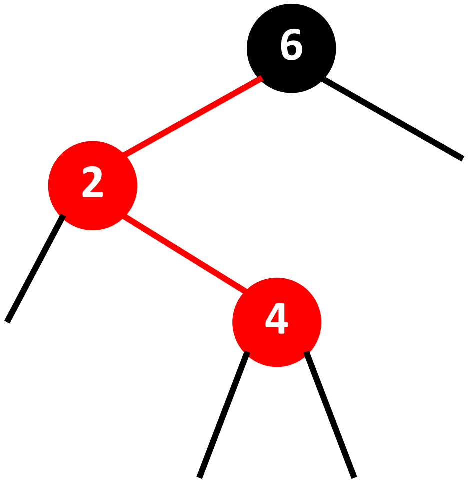
- A restructuring remedies a **child-parent double red** when the **parent red** node has a **black sibling**
- The **internal property** is restored and the **other properties are preserved**



Restructuring

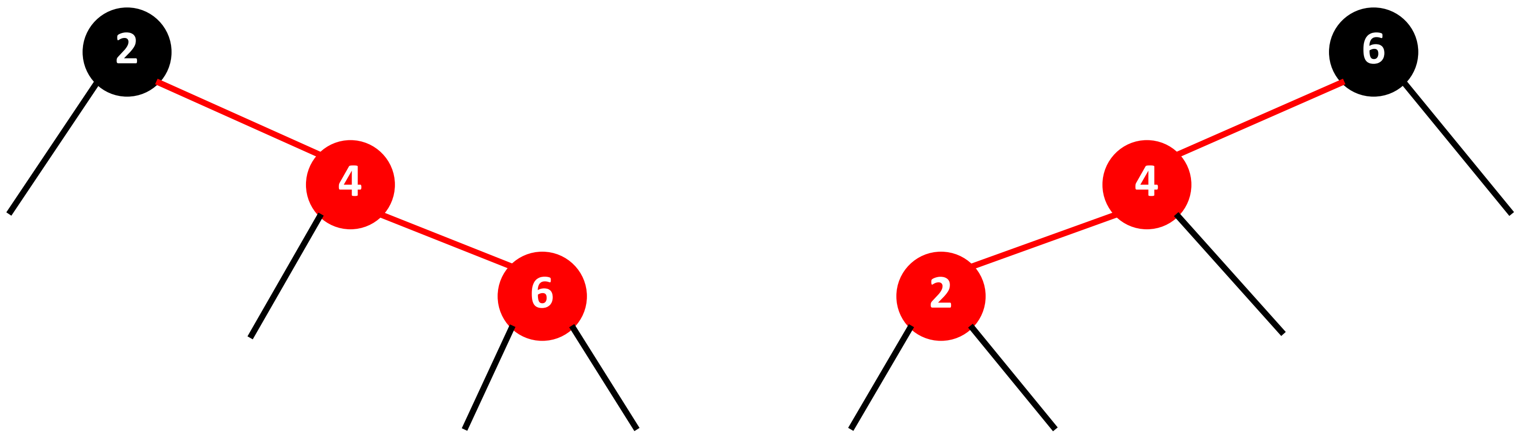
- There are **four restructuring configurations** depending on whether the **double red nodes** are **left** or **right** children:





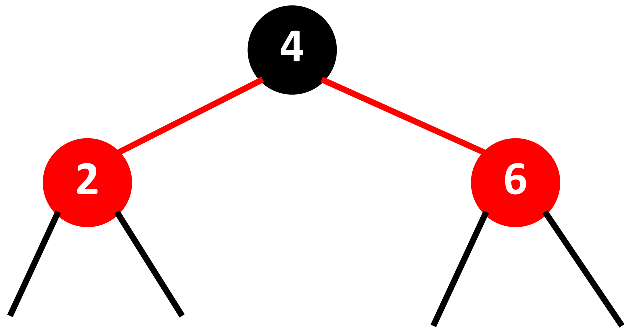
Restructuring

- There are **four restructuring configurations** depending on whether the **double red nodes** are **left** or **right** children:



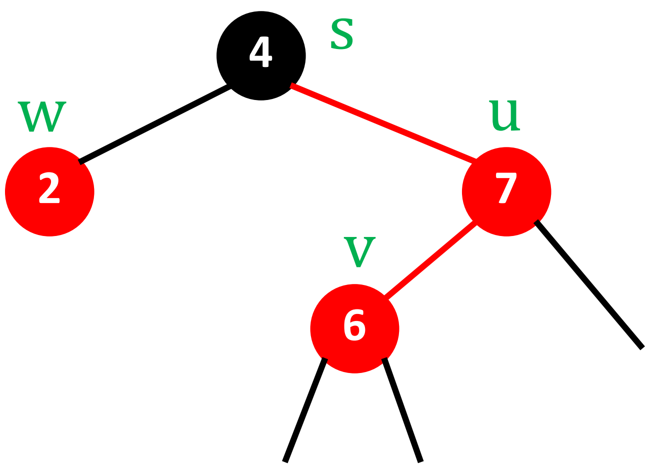
Restructuring

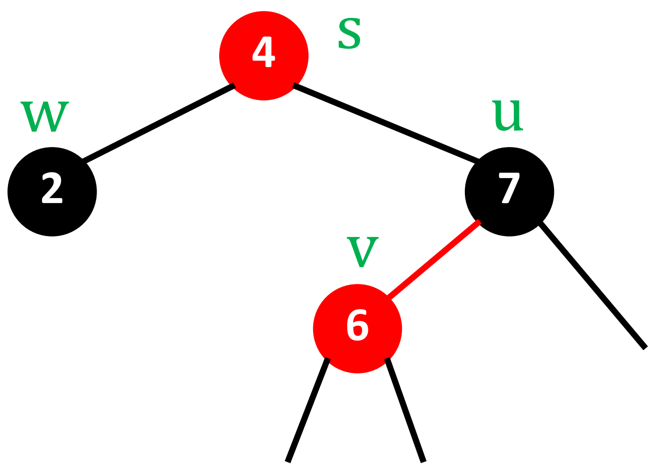
- There are **four restructuring configurations** depending on whether the **double red nodes** are **left** or **right** children:



Recoloring

- A recoloring remedies a **child-parent double red** when the **parent red node** has a **red sibling**
- The **parent** u and its **sibling** w become **black** and the grandparent s becomes **red**, unless it is the root
- The **double red violation** may propagate to the grandparent s





https://www.youtube.com/watch?v=qvZGUFHWChY

https://www.youtube.com/watch?v=95s3ndZRGbk

https://www.youtube.com/watch?v=5IBxA-bZZH8

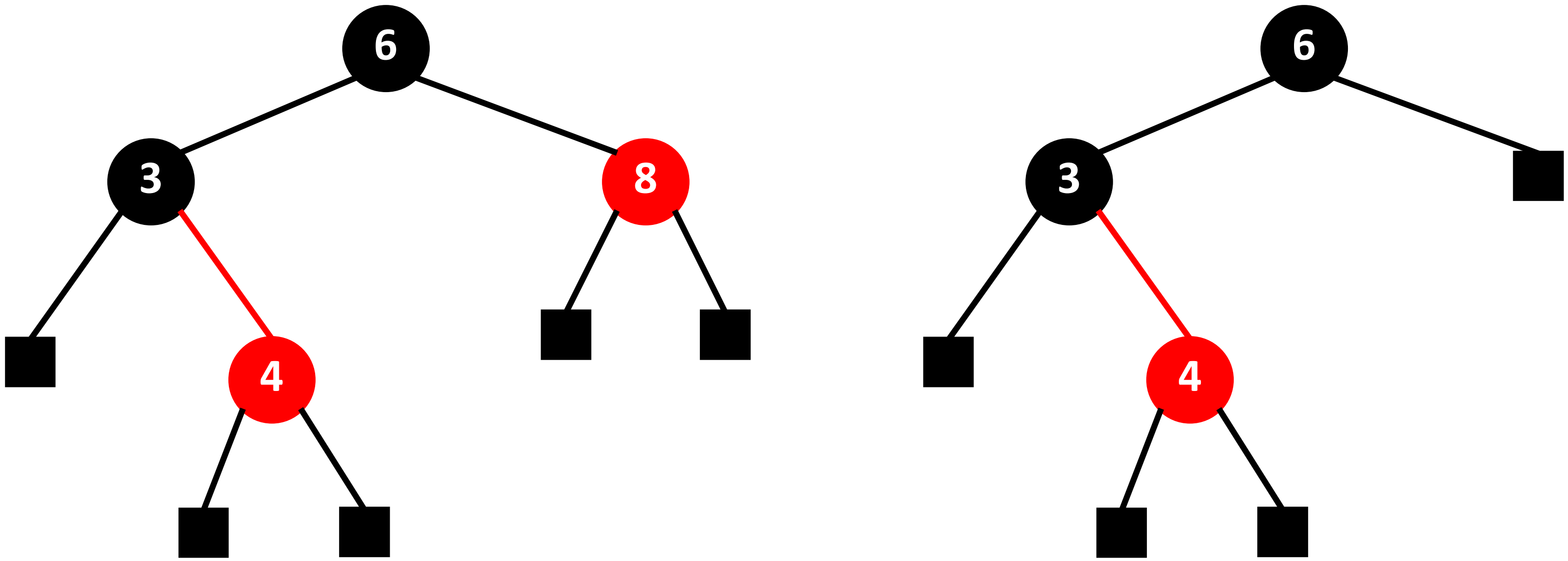
https://www.youtube.com/watch?v=A3JZinzkMpk

Deletion (Optional)

- To perform operation **remove**(k), we **first** execute the deletion algorithm for binary search trees
- Let v be the **internal node removed**, w the **external node removed**, and r the **sibling** of w
- If either v of r was **red**, we color r **black** and we are done
- Else (v and r were both **black**) we color r **double black**, which is a **violation of the internal property** requiring a **reorganization** of the tree

Deletion

Example:



Remedying Double Black

The algorithm for remedying a **double black** node w with sibling y considers **three cases**

Case 1: y is **black** and has a **red child**

- We perform a **restructuring**, equivalent to a **transfer** , and we are done

Case 2: y is **black** and its **children** are **both black**

- We perform a **recoloring**, equivalent to a **fusion**, which may propagate up the double black violation

Case 3: y is **red**

- We perform an **adjustment**, equivalent to choosing a different representation of a 3-node, after which either Case 1 or Case 2 applies