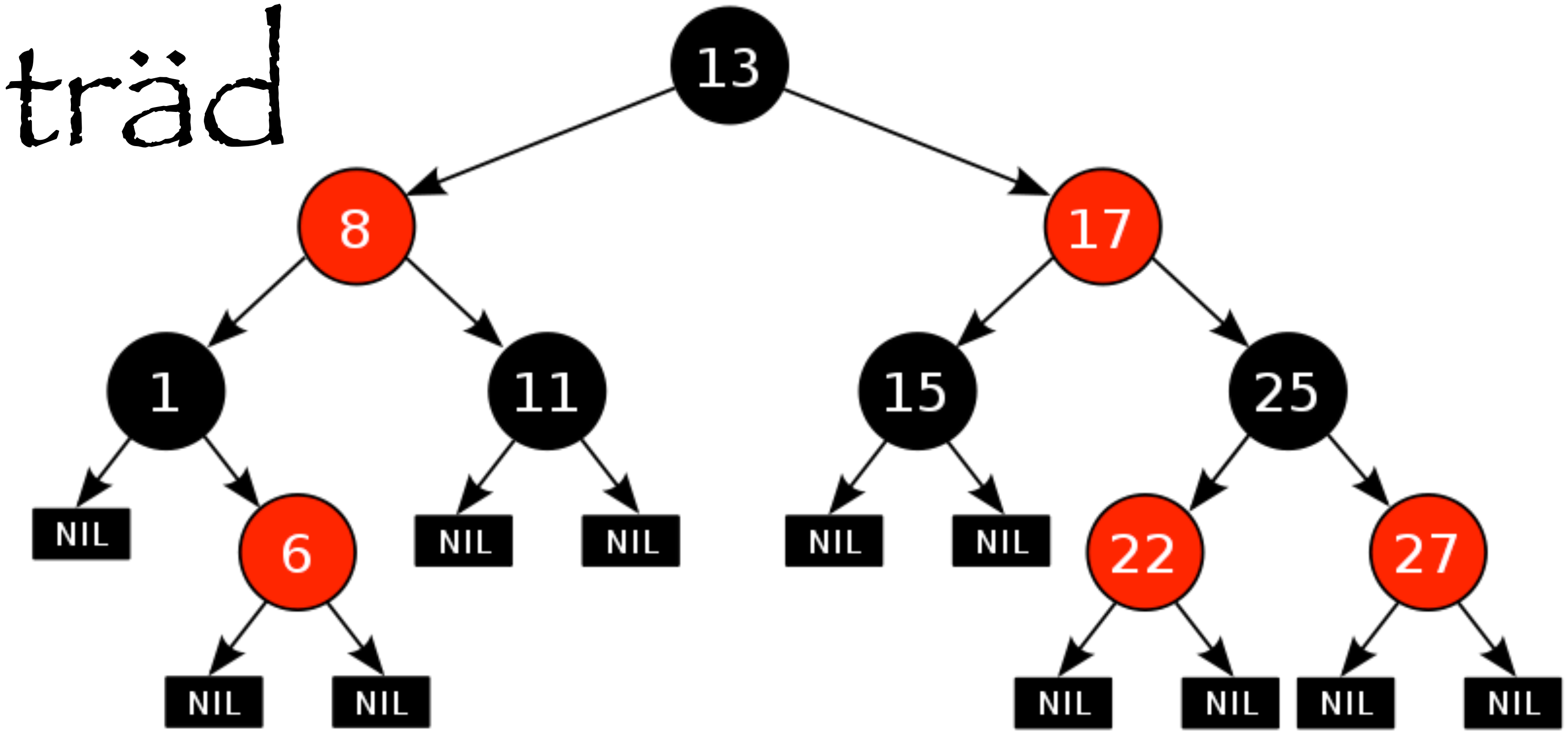
Rödsvarta träd

Koffman & Wolfgang

- kapitel 9, avsnitt 3

Rödsvarta träd



Ett rödsvart träd har följande invarianter:

- en nod är antingen röd eller svart
- roten är alltid svart
- en röd nod har alltid svarta barn
- tomma barn kallas ”löv” och är svarta
- antalet svarta noder är alltid samma,
  i varje stig från roten till ett löv
  - antalet röda noder i stigen är aldrig fler än antalet svarta
  - därför är trädet balanserat

Insättning i ett rödsvart träd

Först söker vi efter insättningspunkten precis som för alla binära sökträd

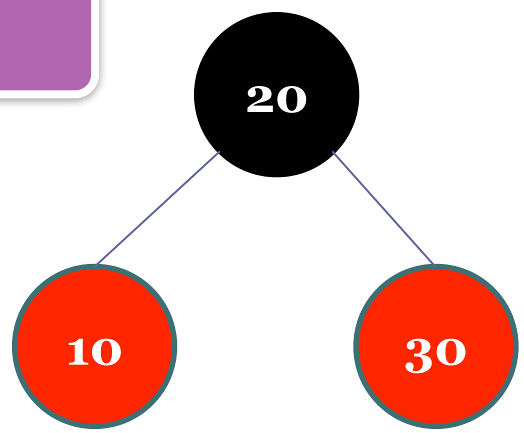
Det nya elementet ersätter ett löv och ges färgen röd

- den nya noden får två svarta löv, så antalet svarta noder i lövstigarna är oförändrat
- om föräldern är svart, så är vi klara
- annars behöver vi arrangera om trädet
- det finns tre möjliga fall som vi måste hantera

Hädanefter visar vi inte de svarta löven (de tomma noderna), men de finns alltid där

Insättning, fall 1

CASE 1

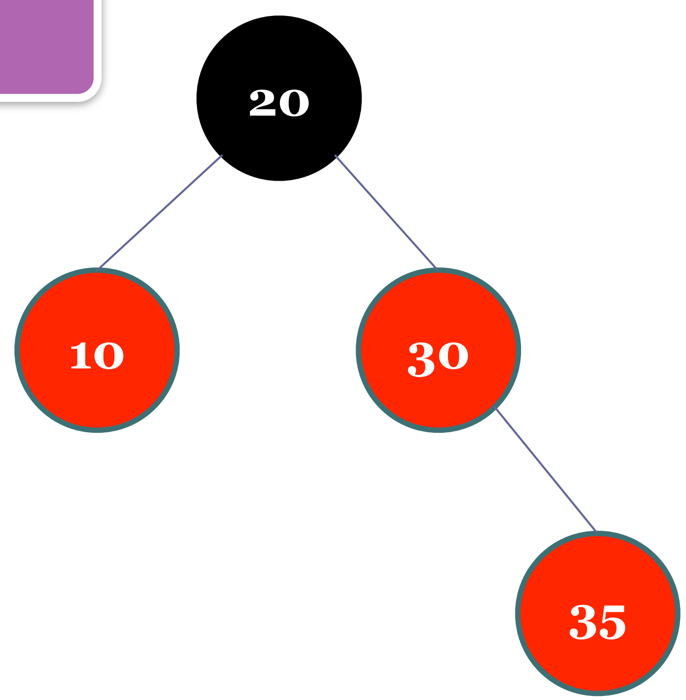


Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Insättning, fall 1

CASE 1

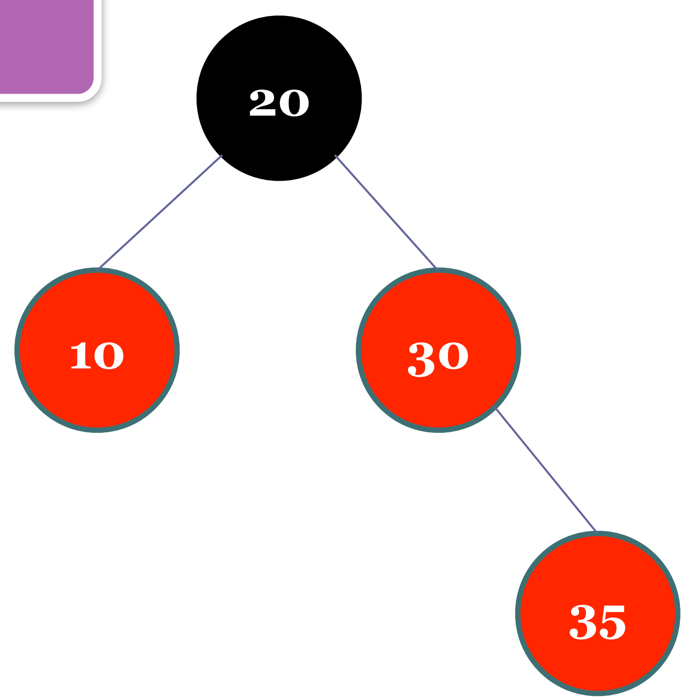


Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Insättning, fall 1

CASE 1



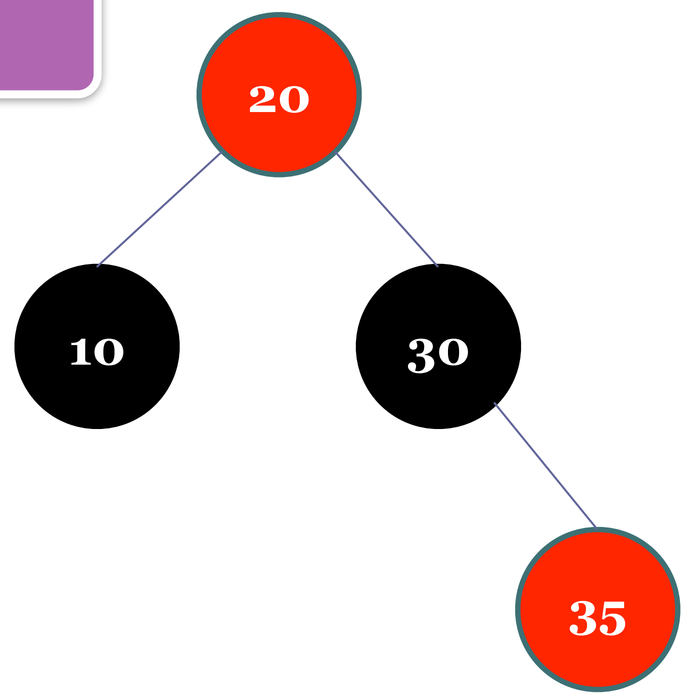
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

If a parent is red, and its sibling is also red, they can both be changed to black, and the grandparent to red

Insättning, fall 1

CASE 1



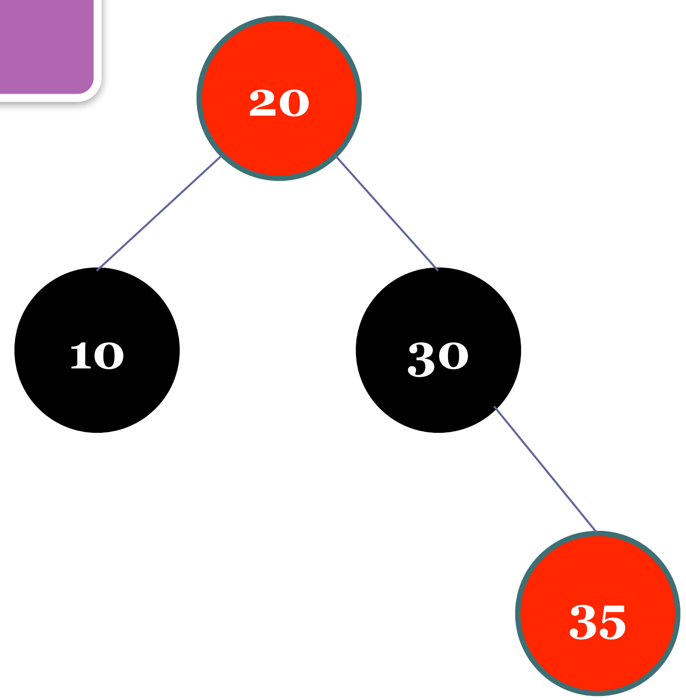
If a parent is red, and its sibling is also red, they can both be changed to black, and the grandparent to red

Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Insättning, fall 1

CASE 1

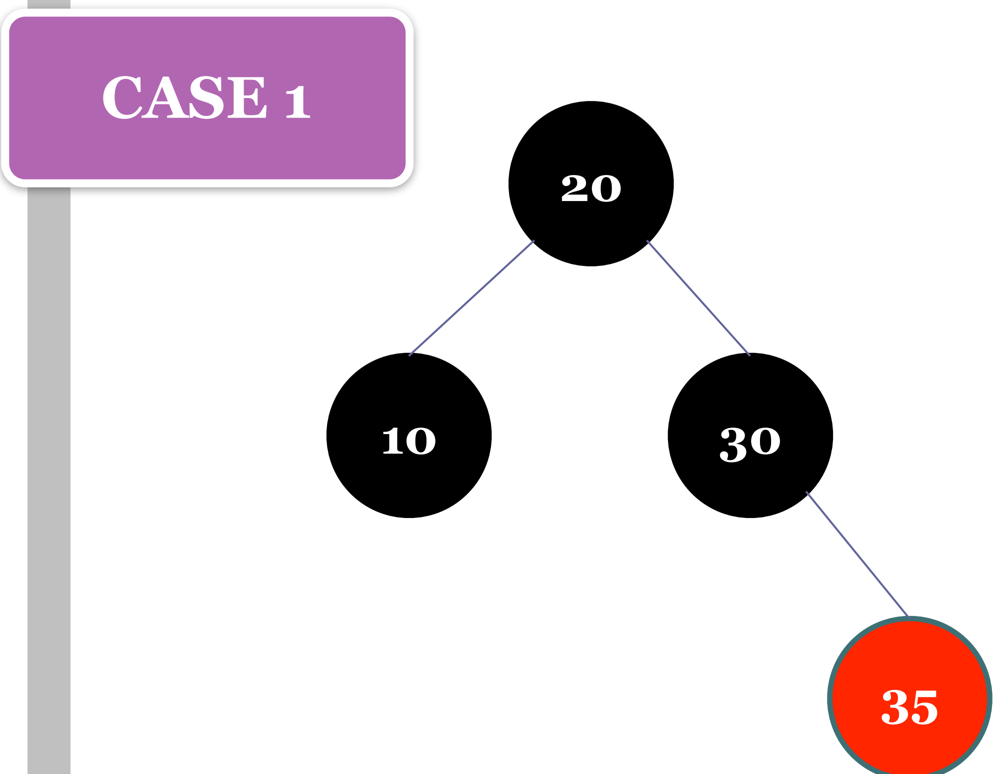


The root can be changed to black and still maintain invariant 4

Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Insättning, fall 1



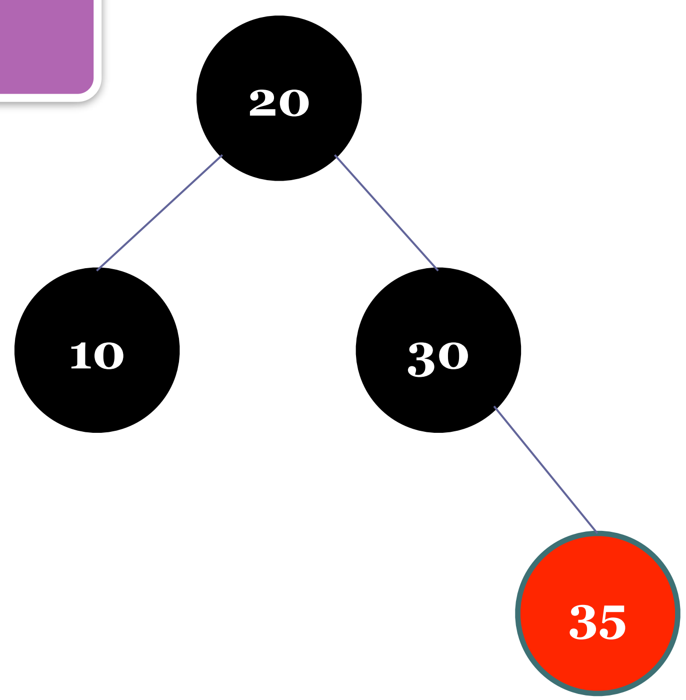
The root can be changed to black and still maintain invariant 4

Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Insättning, fall 1

CASE 1



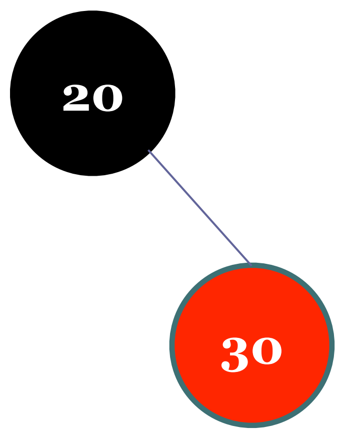
Balanced tree

Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Insättning, fall 2

CASE 2

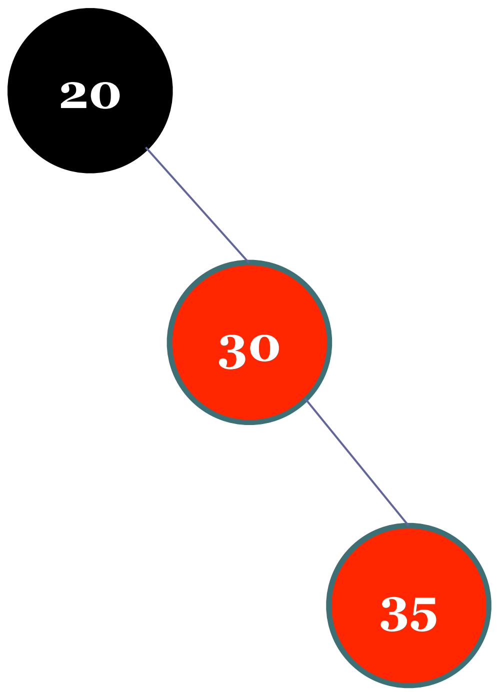


Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Insättning, fall 2

CASE 2

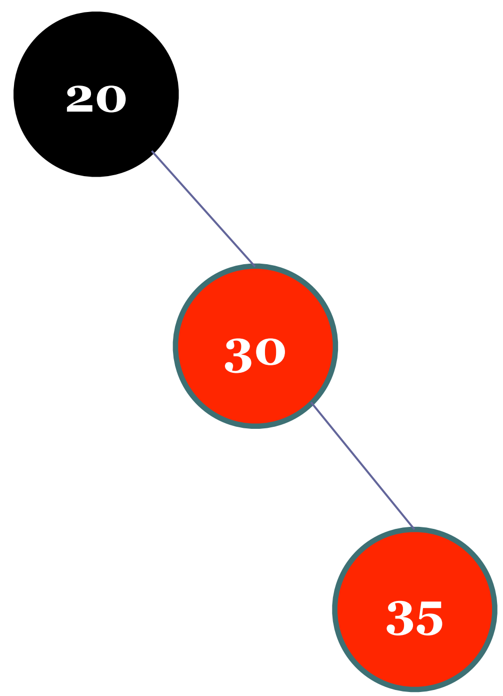


Invariants:

1. **A node is either red or black**
2. **The root is always black**
3. **A red node always has black children (a `null` reference is considered to refer to a black node)**
4. **The number of black nodes in any path from the root to a leaf is the same**

Insättning, fall 2

CASE 2



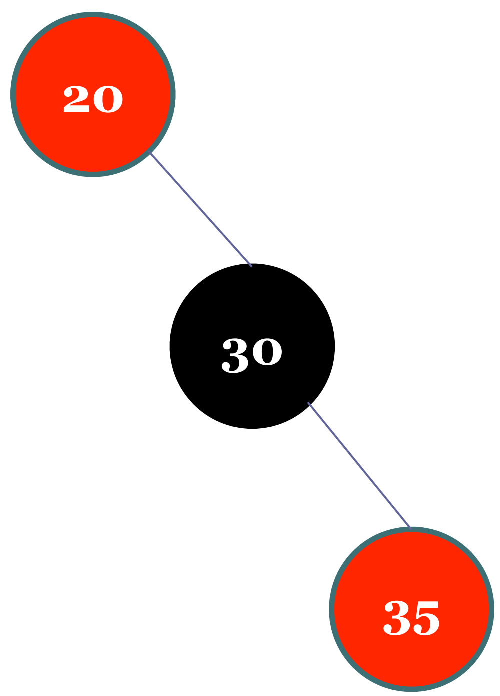
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

If a parent is red (with no sibling), it can be changed to black, and the grandparent to red

Insättning, fall 2

CASE 2



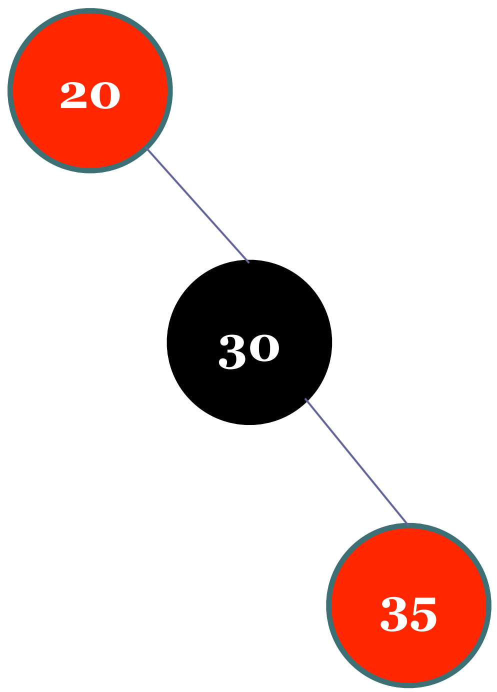
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

If a parent is red (with no sibling), it can be changed to black, and the grandparent to red

Insättning, fall 2

CASE 2



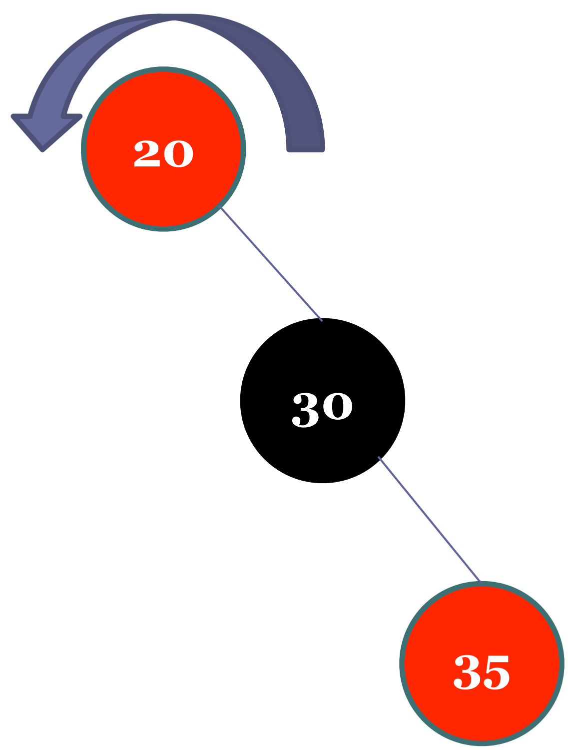
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

There is one black node on the right and none on the left, which violates invariant 4

Insättning, fall 2

CASE 2

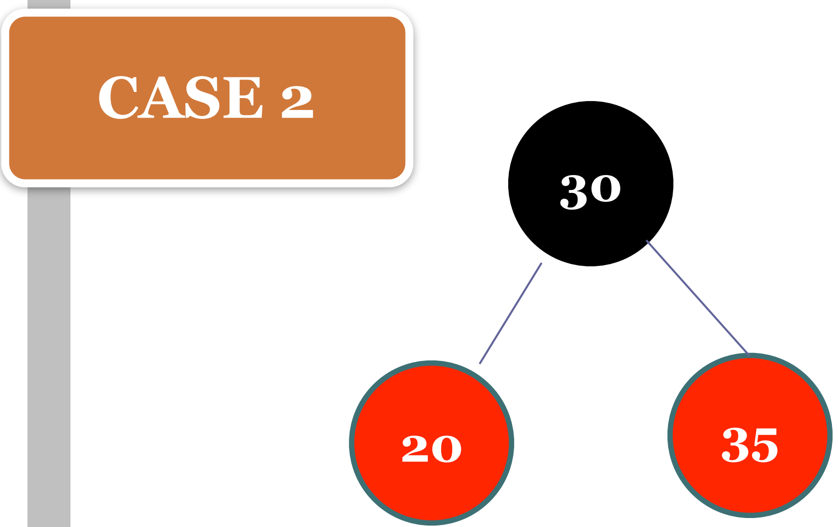


Rotate left around the grandparent to correct this

Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Insättning, fall 2



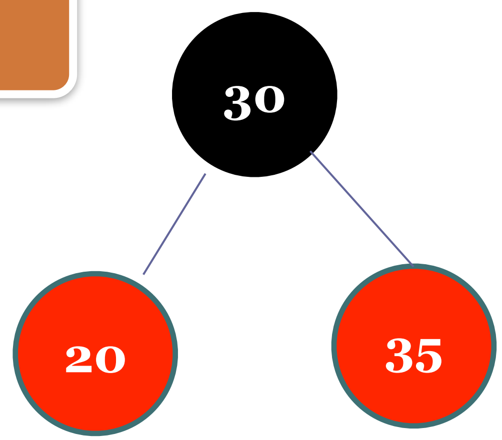
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Rotate left around the grandparent to correct this

Insättning, fall 2

CASE 2

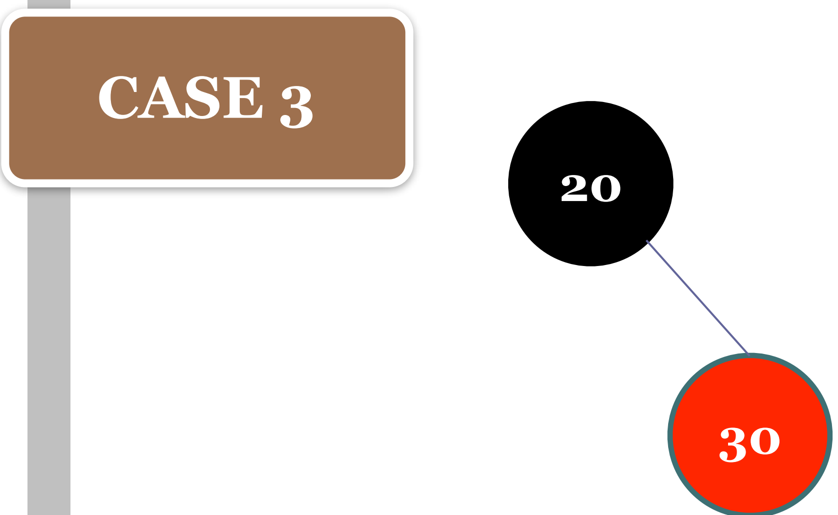


Balanced tree

Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Insättning, fall 3 (första försöket)

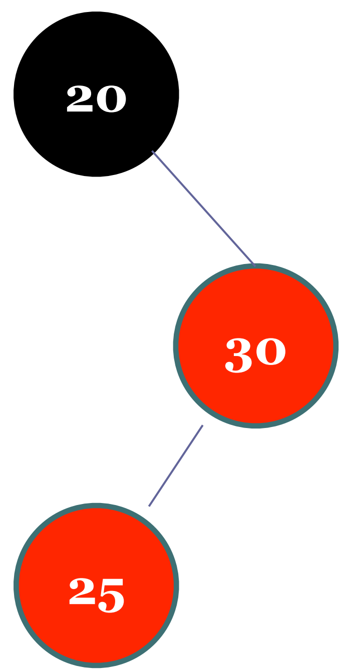


Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Insättning, fall 3 (första försöket)

CASE 3

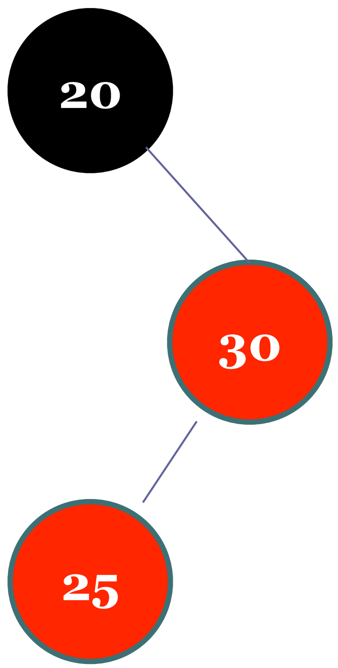


Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Insättning, fall 3 (första försöket)

CASE 3



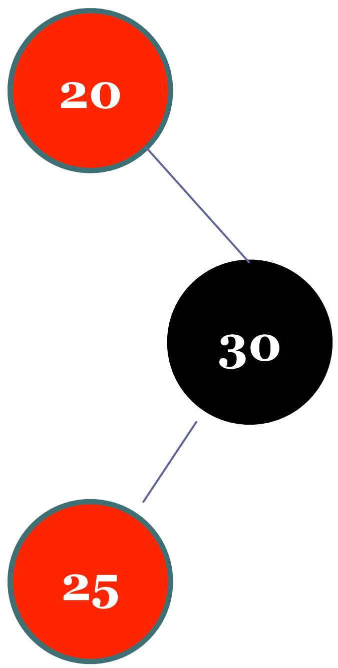
If a parent is red (with no sibling), it can be changed to black, and the grandparent to red

Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Insättning, fall 3 (första försöket)

CASE 3



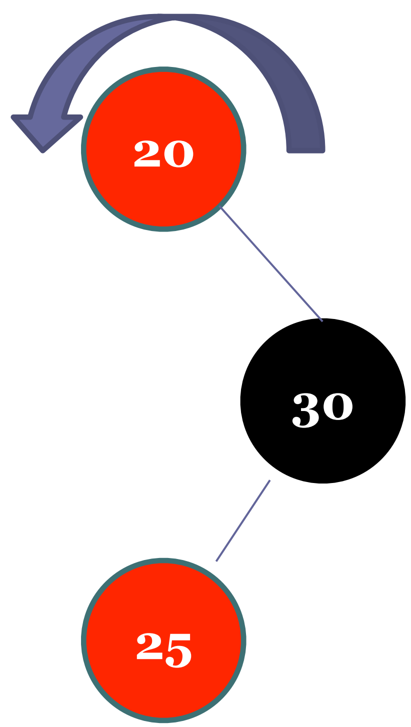
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

If a parent is red (with no sibling), it can be changed to black, and the grandparent to red

Insättning, fall 3 (första försöket)

CASE 3



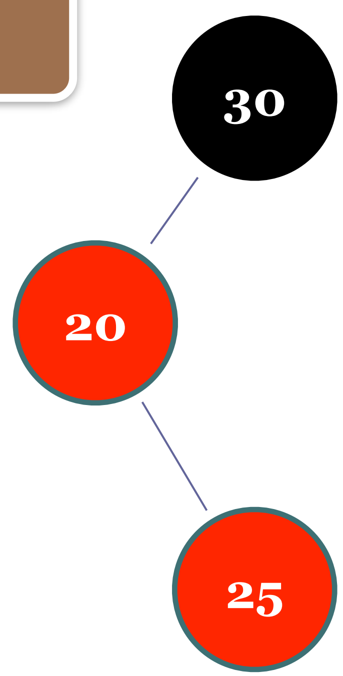
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

A rotation left does not fix the violation of #4

Insättning, fall 3 (första försöket)

CASE 3



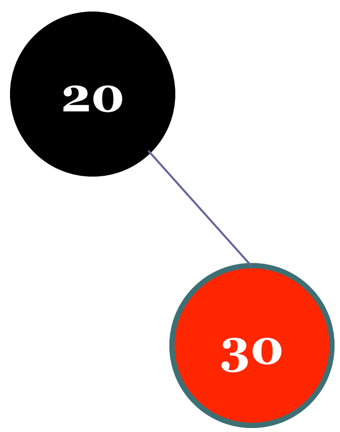
A rotation left does not fix the violation of #4

Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Insättning, fall 3, korrekt version

CASE 3



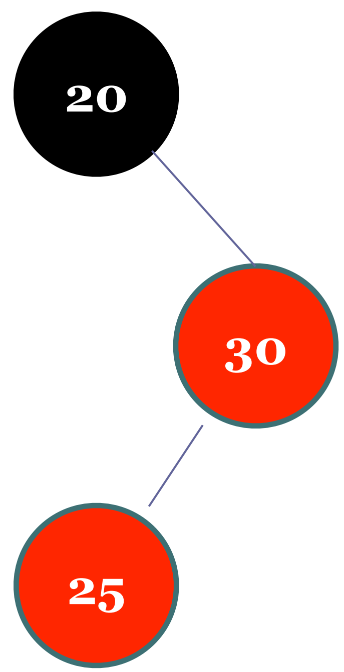
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Back-up to the beginning (don't perform rotation or change colors)

Insättning, fall 3

CASE 3



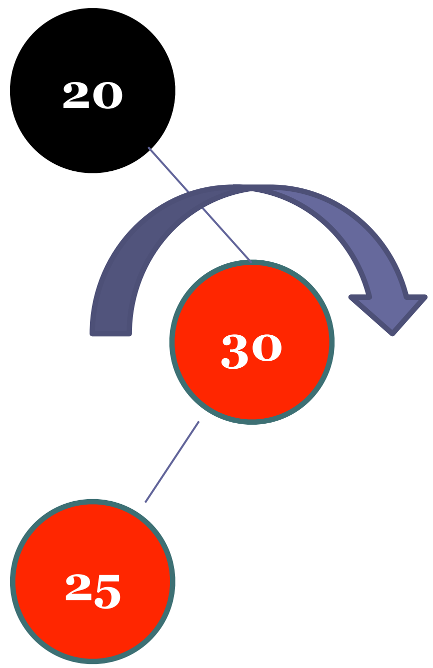
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Back-up to the beginning (don't perform rotation or change colors)

Insättning, fall 3

CASE 3



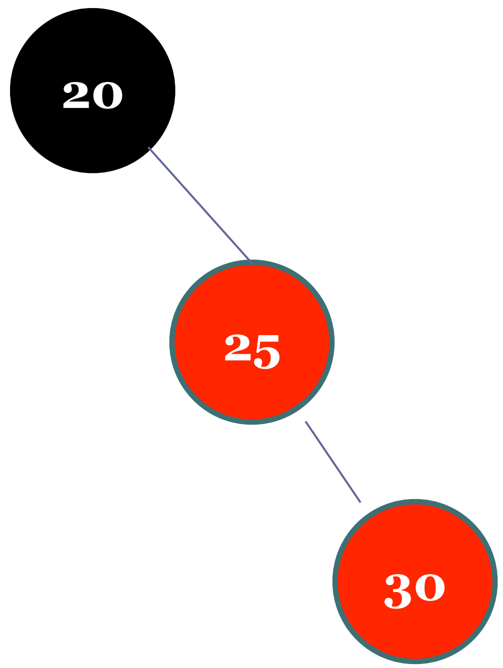
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Rotate right about the parent so that the red child is on the same side of the parent as the parent is to the grandparent

Insättning, fall 3

CASE 3



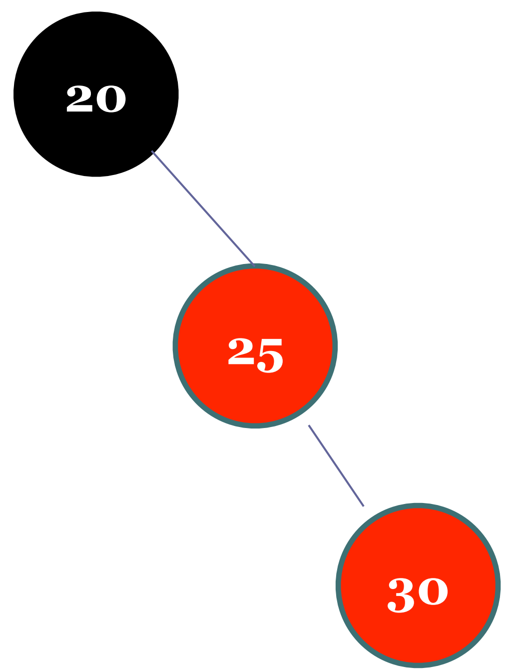
Rotate right about the parent so that the red child is on the same side of the parent as the parent is to the grandparent

Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Insättning, fall 3

CASE 3



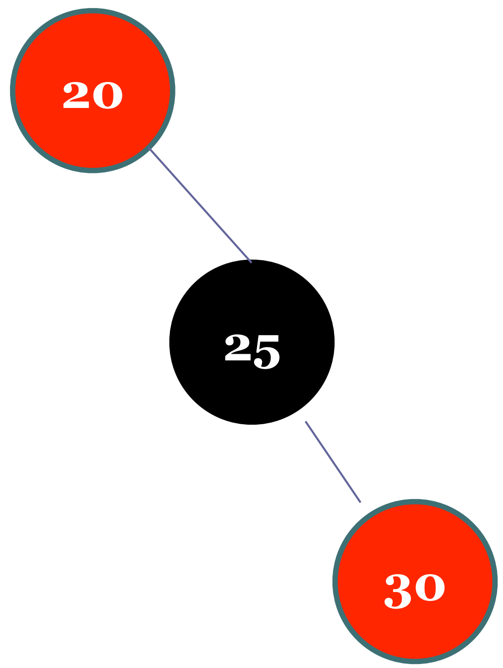
NOW, change colors

Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Insättning, fall 3

CASE 3

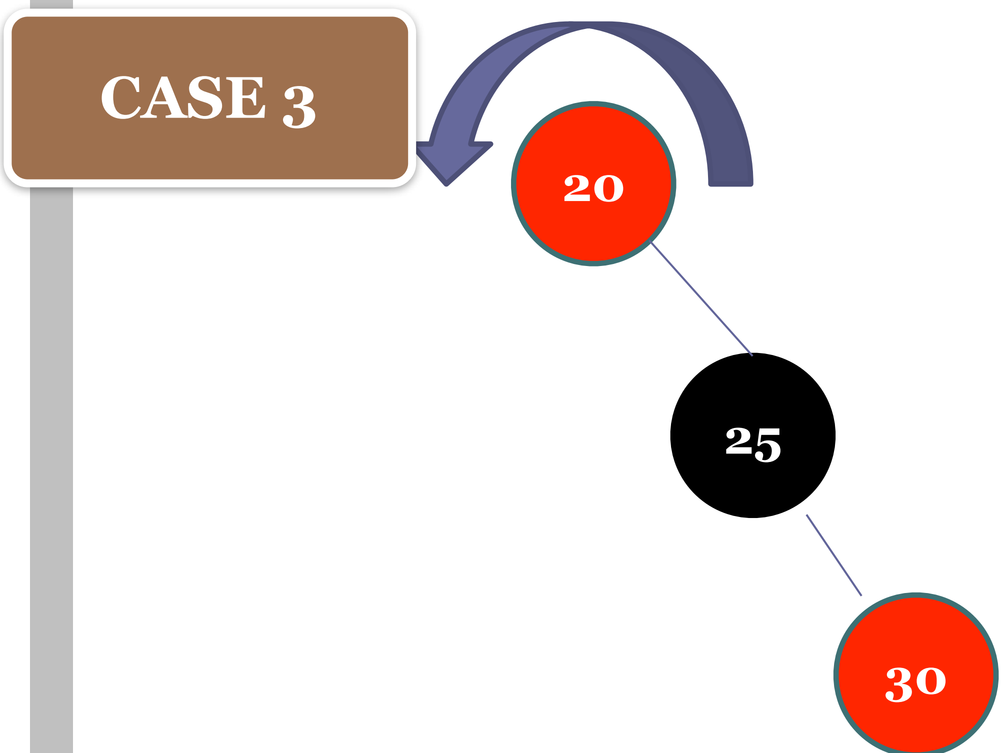


Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

NOW, change colors

Insättning, fall 3



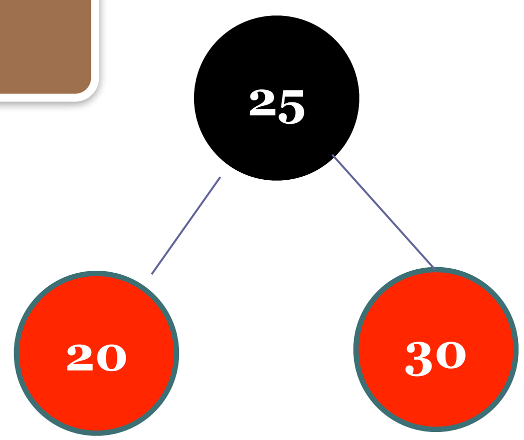
and rotate left . . .

Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Insättning, fall 3

CASE 3



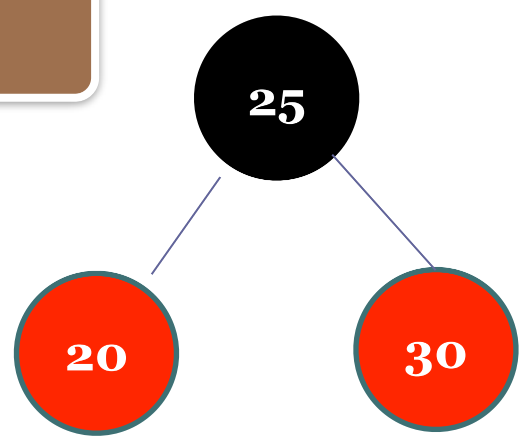
and rotate left. . .

Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Insättning, fall 3

CASE 3



Balanced tree

Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Insättning, sammanfattning

Wikipedias artikel om rödsvarta träd är bra!

- http://en.wikipedia.org/wiki/Red–black_tree

Fall 0 (case 2 i Wikipedia):

- föräldern är svart

Fall 1 (case 3 i Wikipedia):

- både föräldern och dess syskon är röda

Fall 2 (case 5 i Wikipedia):

- föräldern är röd och dess syskon svart
- dessutom är noden ett vänster-vänsterbarn / höger-högerbarn

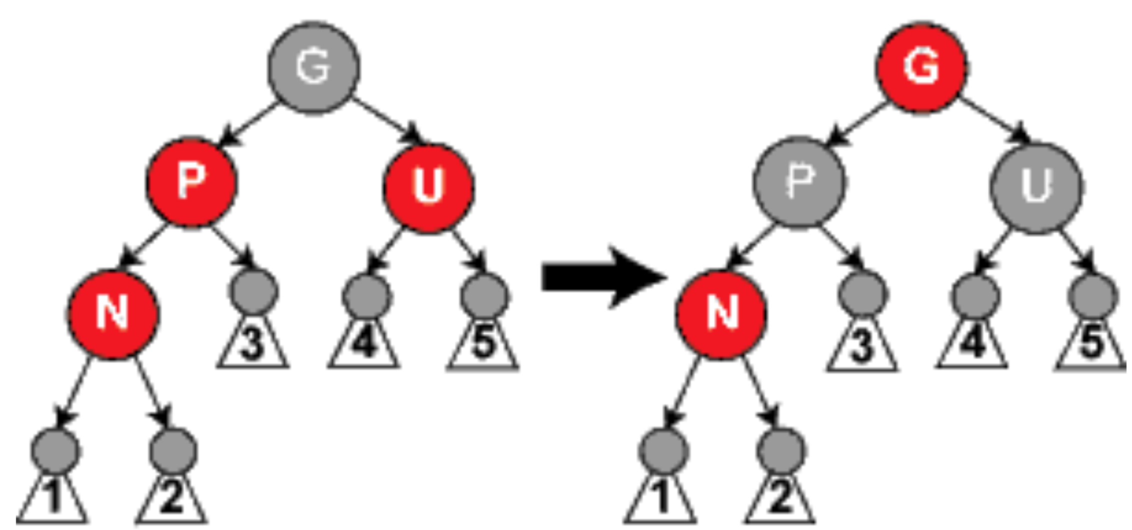
Fall 3 (case 4 i Wikipedia):

- föräldern är röd och dess syskon svart
- dessutom är noden ett höger-vänsterbarn / vänster-högerbarn

Insättning, fall 1

Både föräldern (P) och dess syskon (U) är röda:

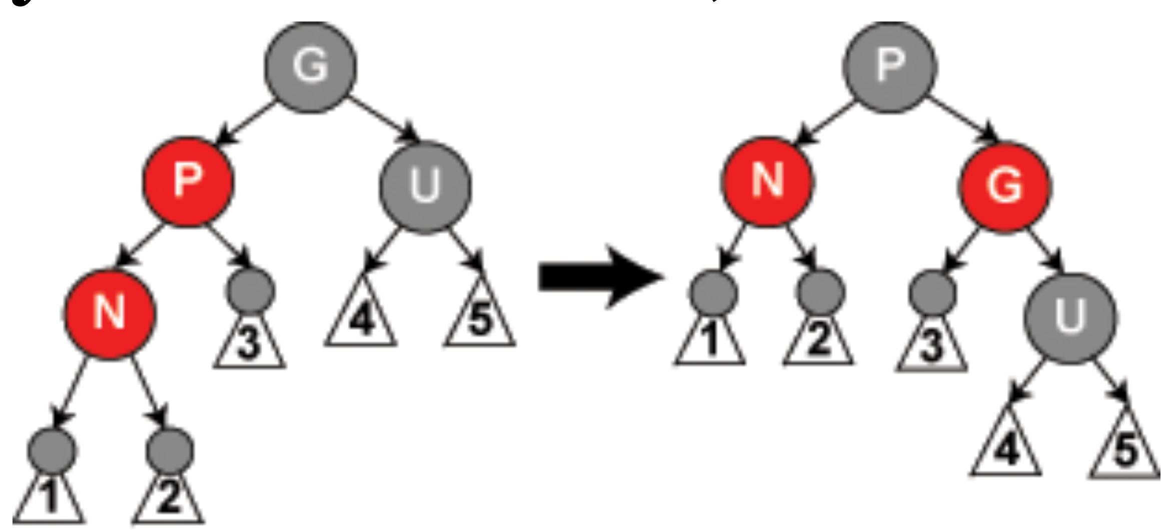
- deras förälder (G) måste då vara svart
- byt färg på dem (P, U) till svart
- byt färg på deras förälder (G) till röd
- fortsätt rekursivt uppåt i trädet
  med G som ny nod



Insättning, fall 2

Föräldern (P) är röd och dess syskon (U) är svart:

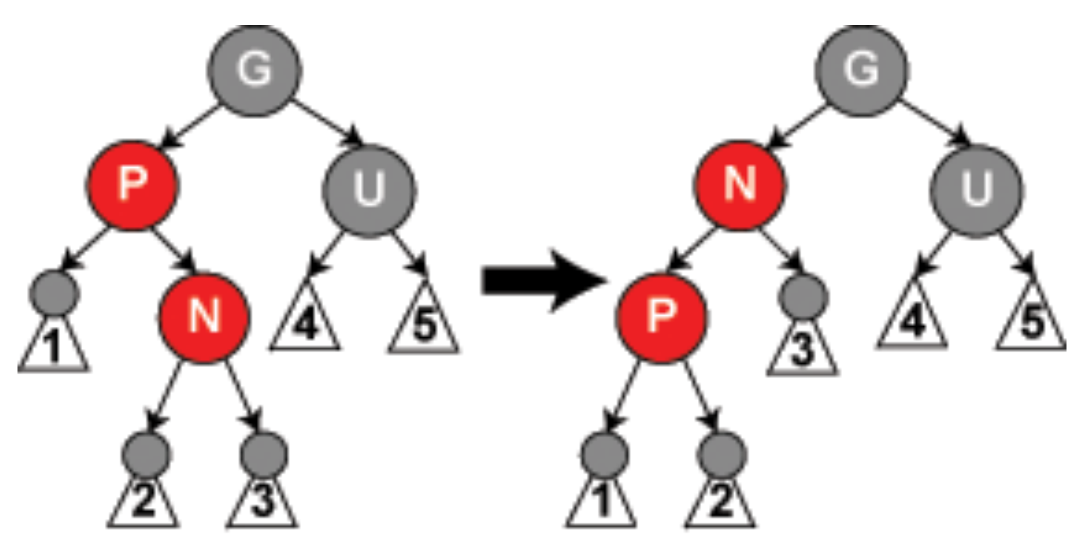
- dessutom är noden (N) ett vänster-vänsterbarn (eller ett höger-högerbarn)
- deras förälder (G) måste vara svart
- rotera runt G
- byt färg på P till svart och på G till röd
- nu är P ny svart lokal rot, och vi är klara!



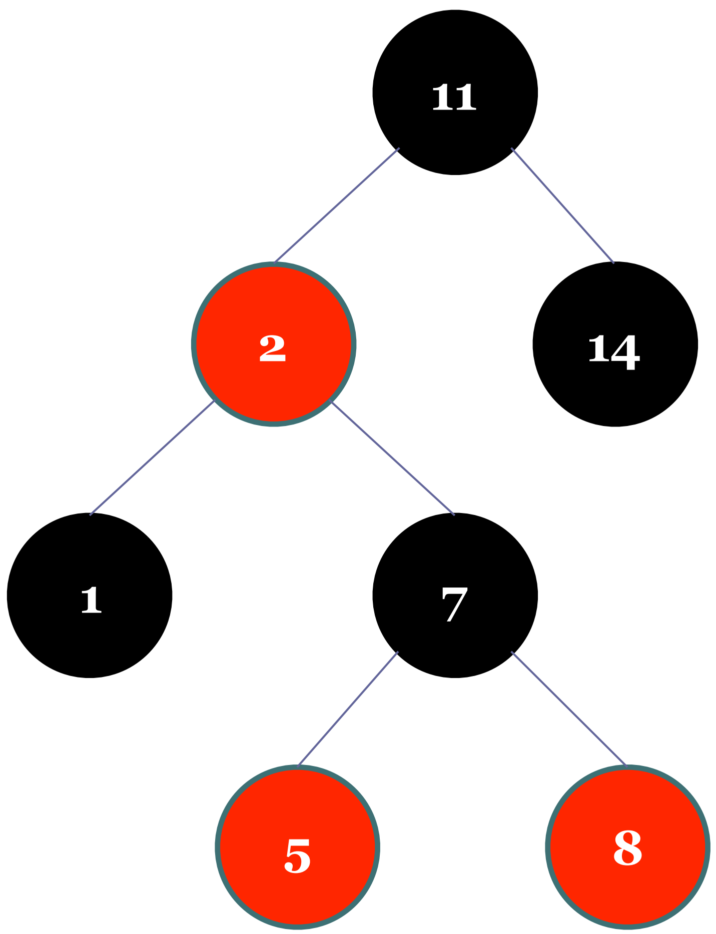
Insättning, fall 3

Föräldern (P) är röd och dess syskon (U) är svart:

- dessutom är noden (N) ett vänster-högerbarn (eller ett höger-vänsterbarn)
- deras förälder (G) måste vara svart
- rotera runt P
- nu är P ett vänster-vänsterbarn (eller höger-högerbarn) och vi kan fortsätta med fall 2



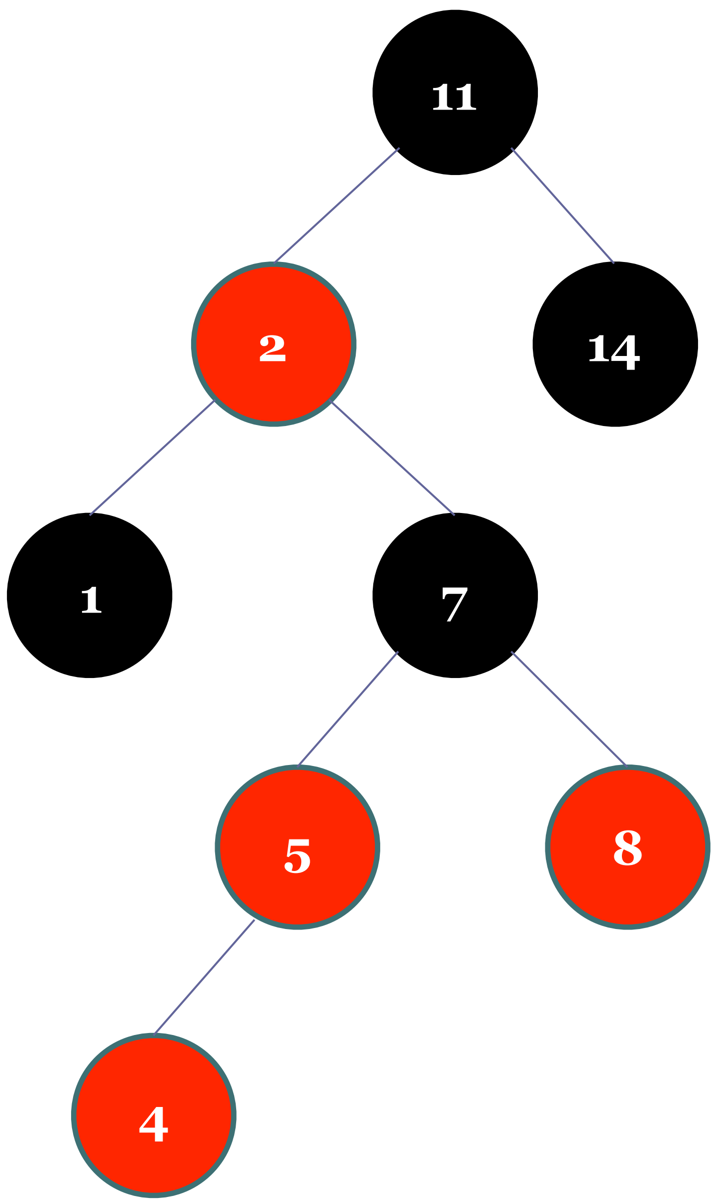
Insättning, ett enkelt exempel



Invariants:

1. **A node is either red or black**
2. **The root is always black**
3. **A red node always has black children (a `null` reference is considered to refer to a black node)**
4. **The number of black nodes in any path from the root to a leaf is the same**

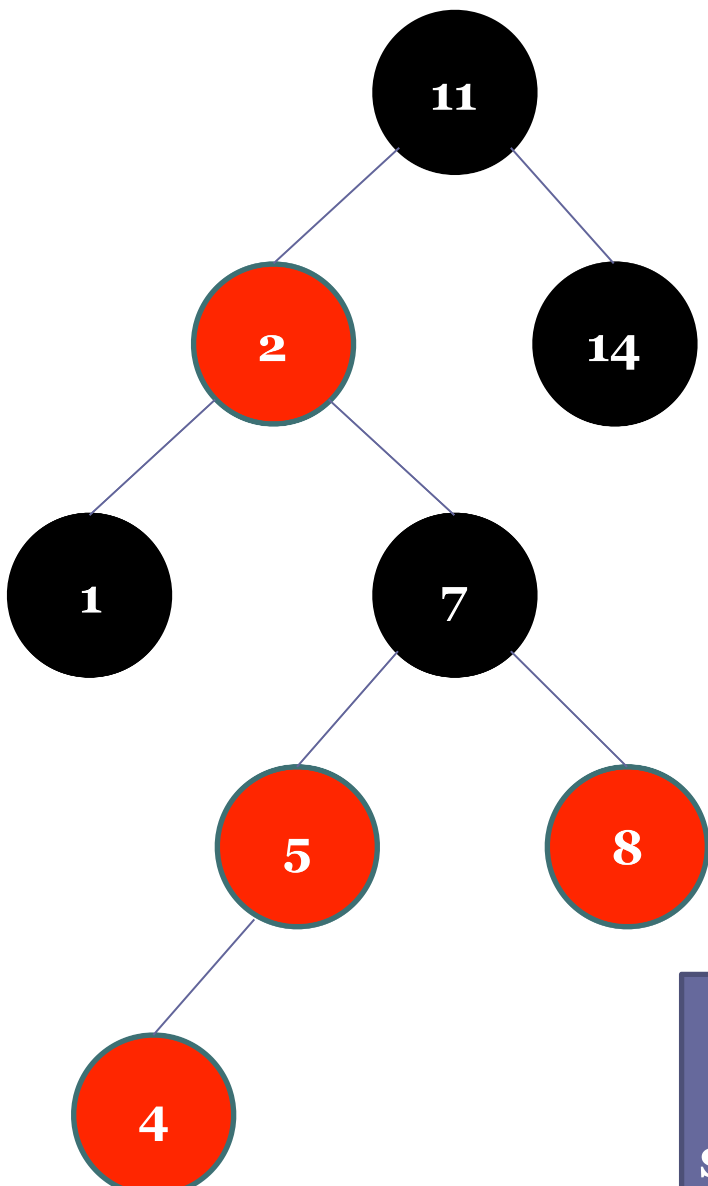
Insättning, ett enkelt exempel



Invariants:

1. **A node is either red or black**
2. **The root is always black**
3. **A red node always has black children (a `null` reference is considered to refer to a black node)**
4. **The number of black nodes in any path from the root to a leaf is the same**

Insättning, ett enkelt exempel



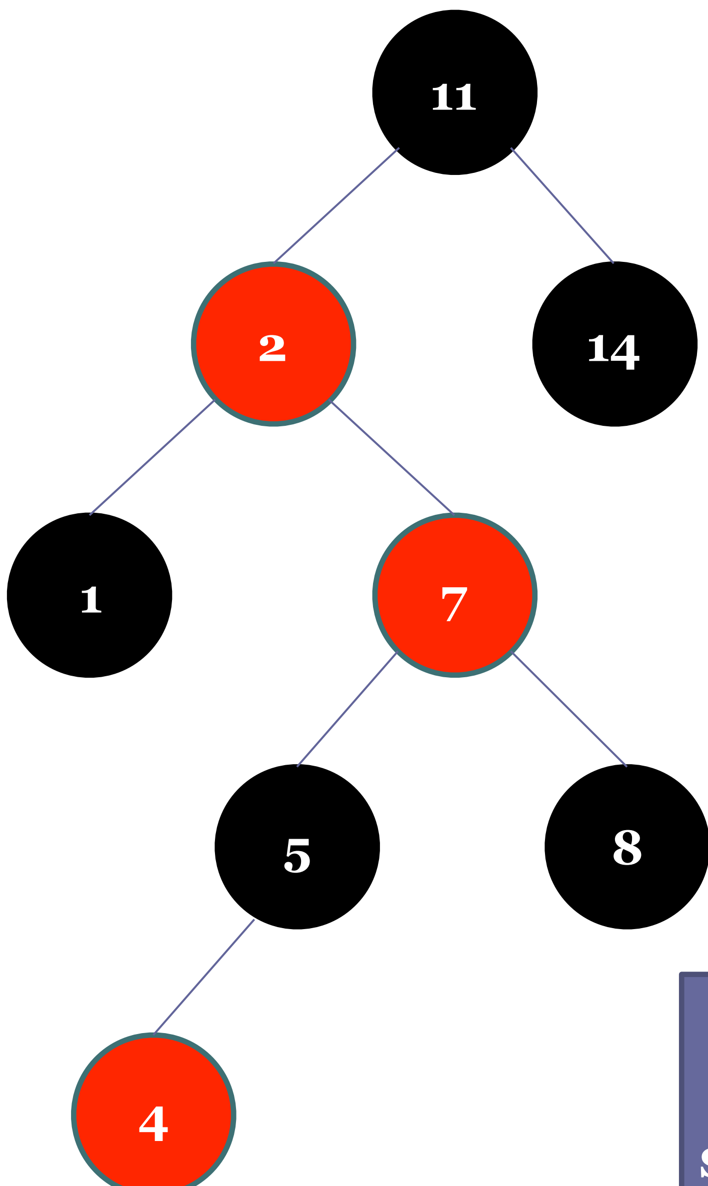
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 1

If a parent is red, and its sibling is also red, they can both be changed to black, and the grandparent to red

Insättning, ett enkelt exempel



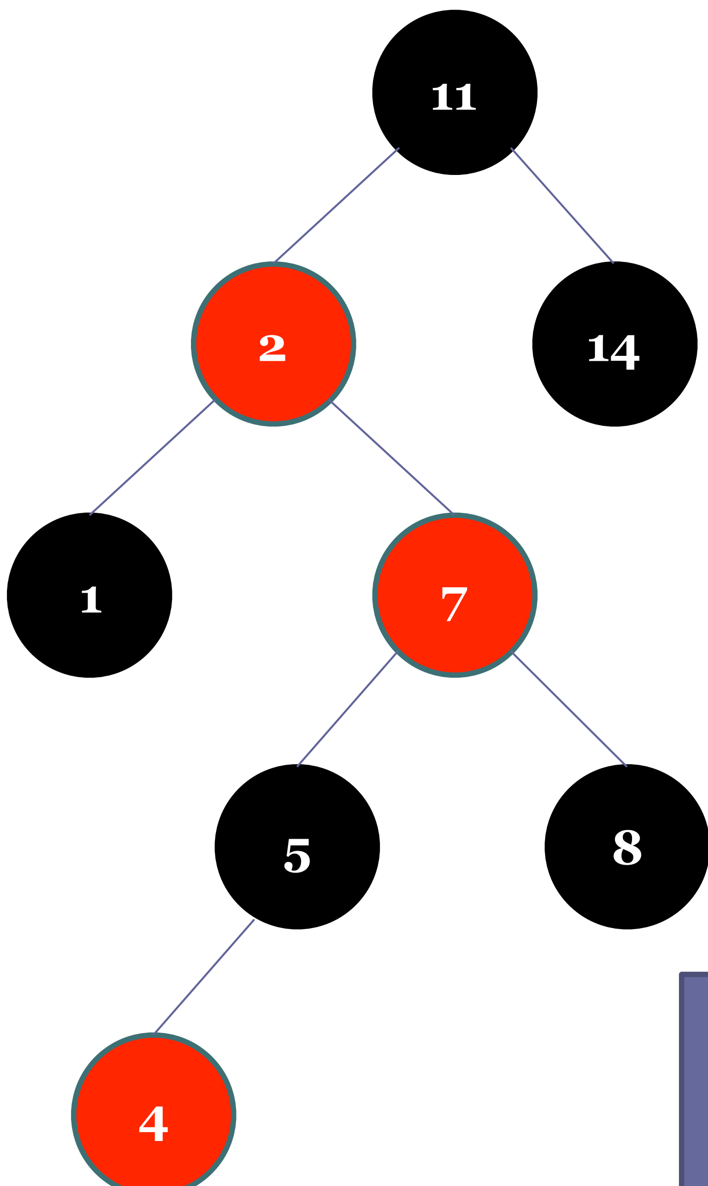
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 1

If a parent is red, and its sibling is also red, they can both be changed to black, and the grandparent to red

Insättning, ett enkelt exempel

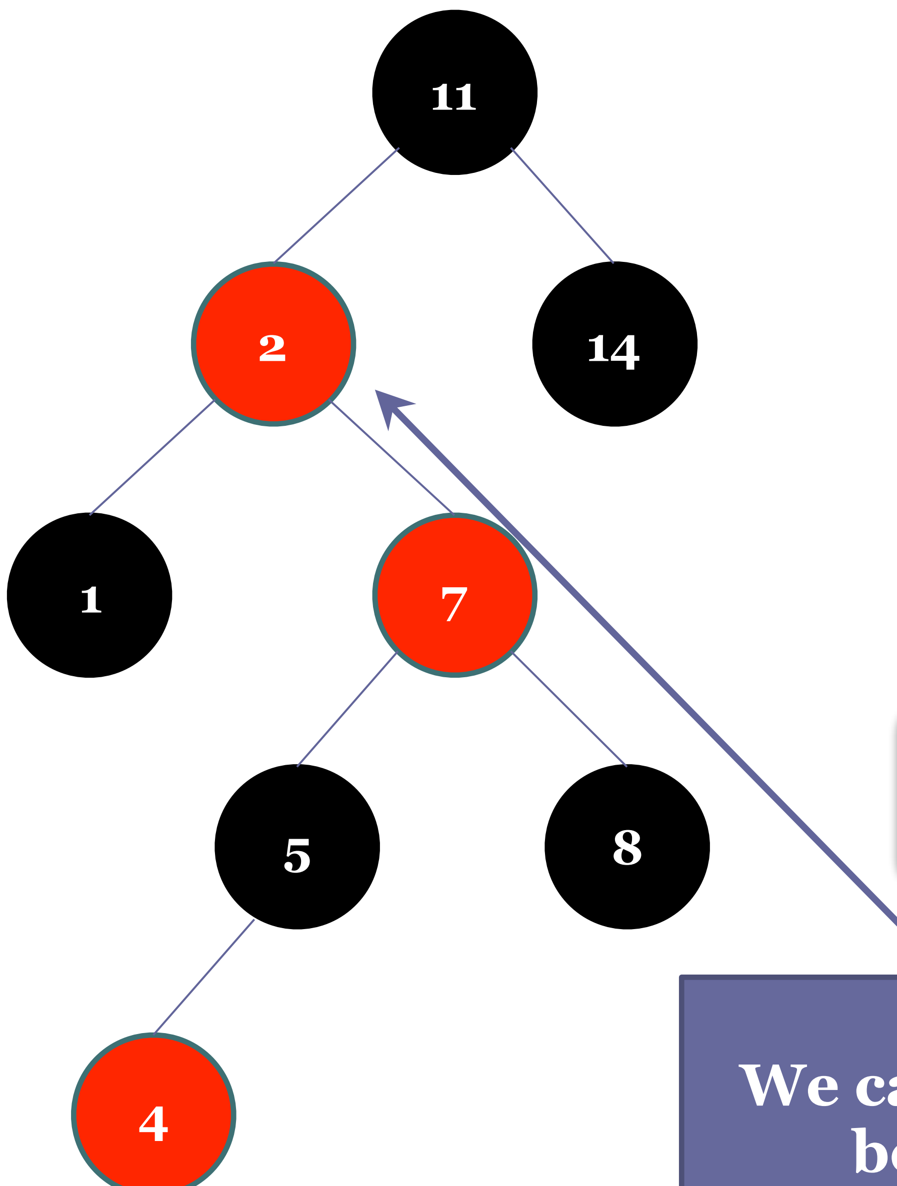


Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

The problem has now shifted up the tree

Insättning, ett enkelt exempel



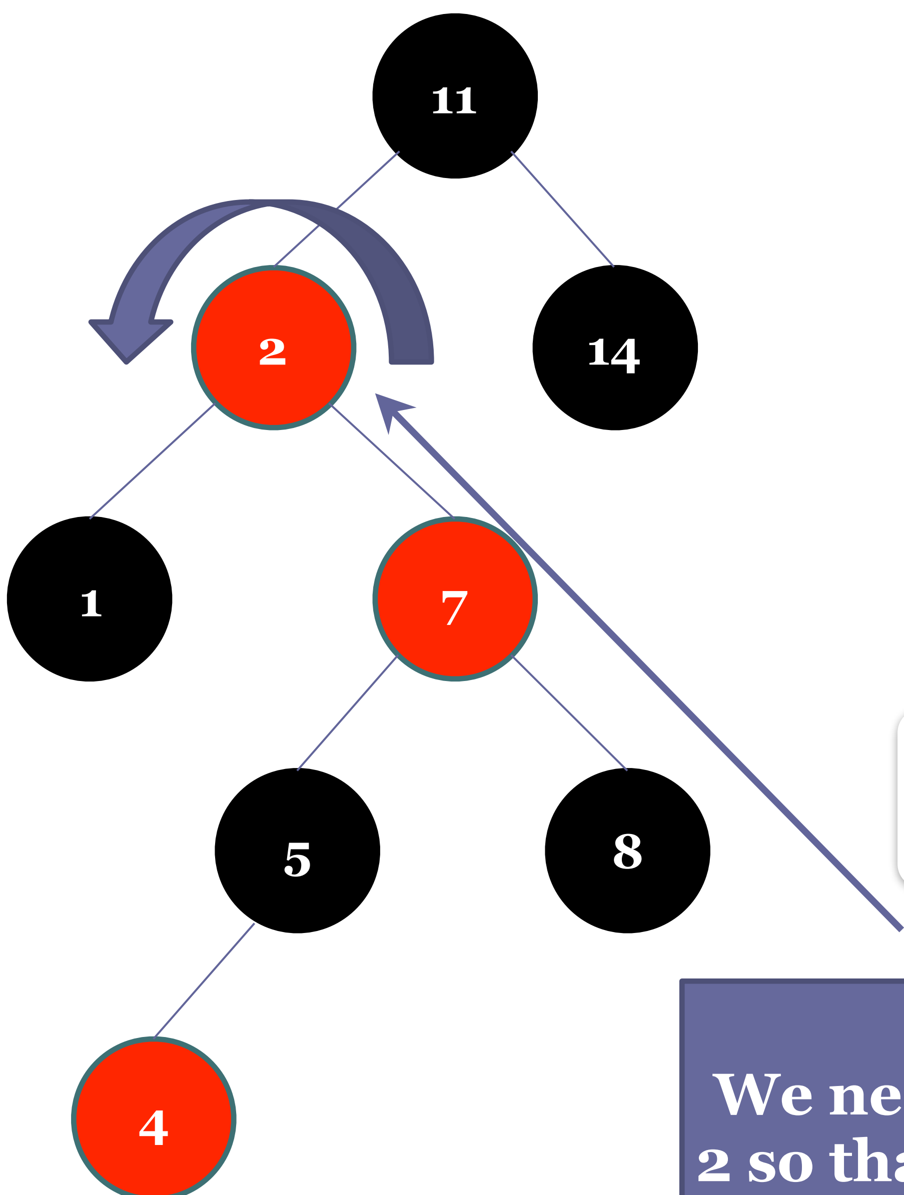
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 3

We cannot change 2 to black because its sibling 14 is already black (both siblings have to be red to do the color change)

Insättning, ett enkelt exempel



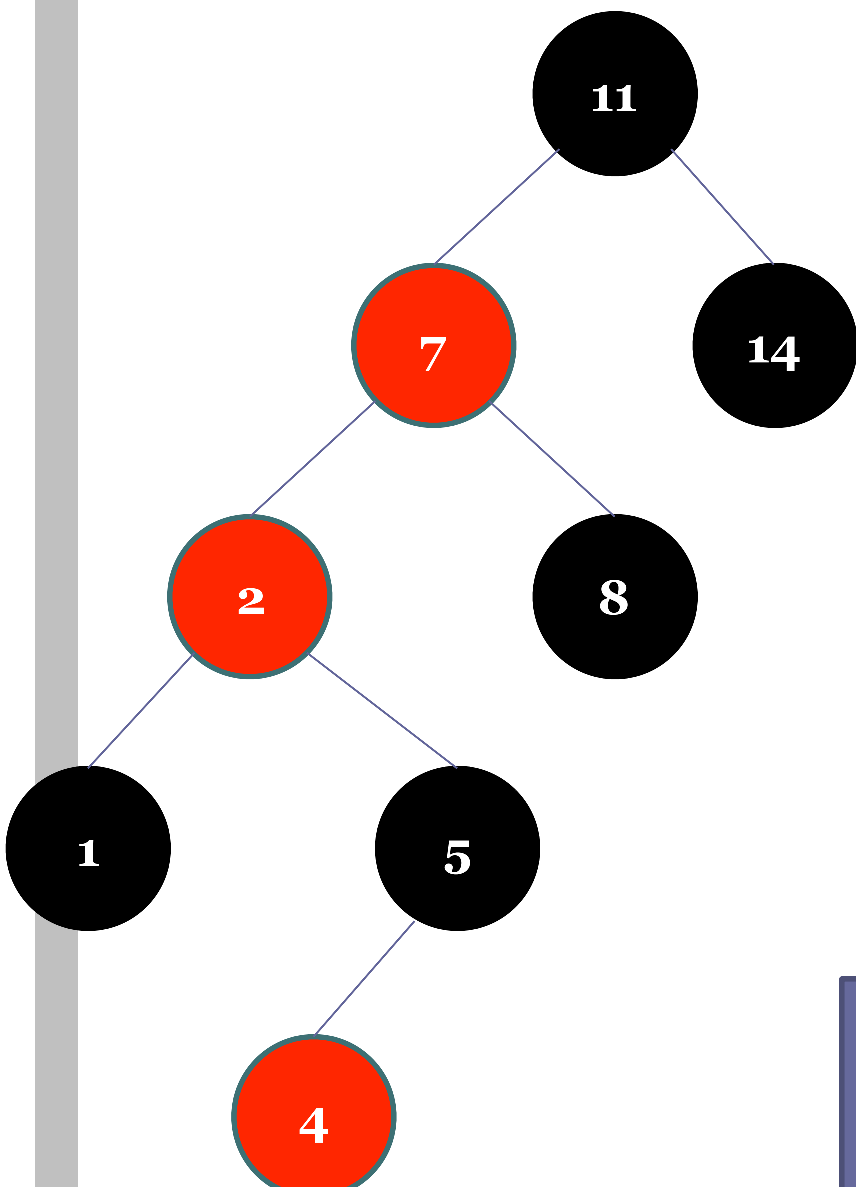
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 3

We need to rotate left around 2 so that the red child is on the same side of the parent as the parent is to the grandparent

Insättning, ett enkelt exempel



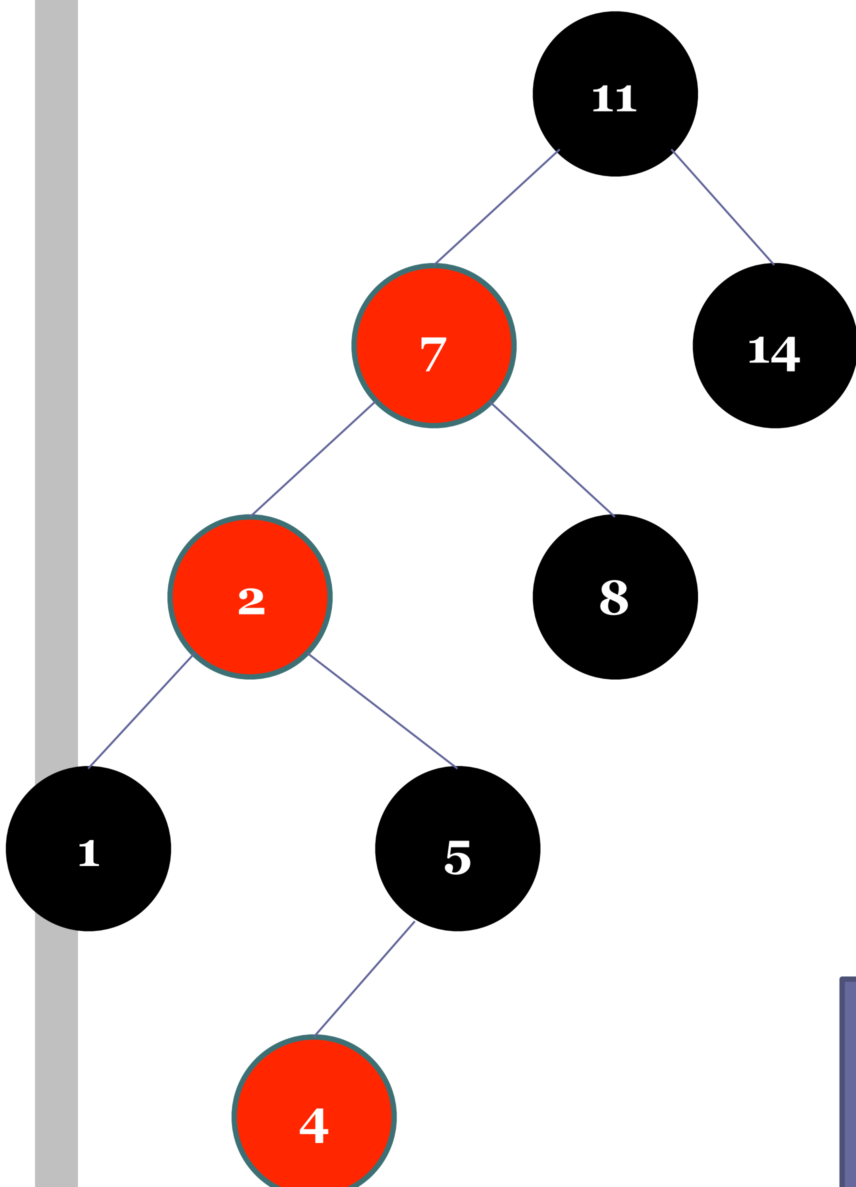
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 3

We need to rotate left around 2 so that the red child is on the same side of the parent as the parent is to the grandparent

Insättning, ett enkelt exempel



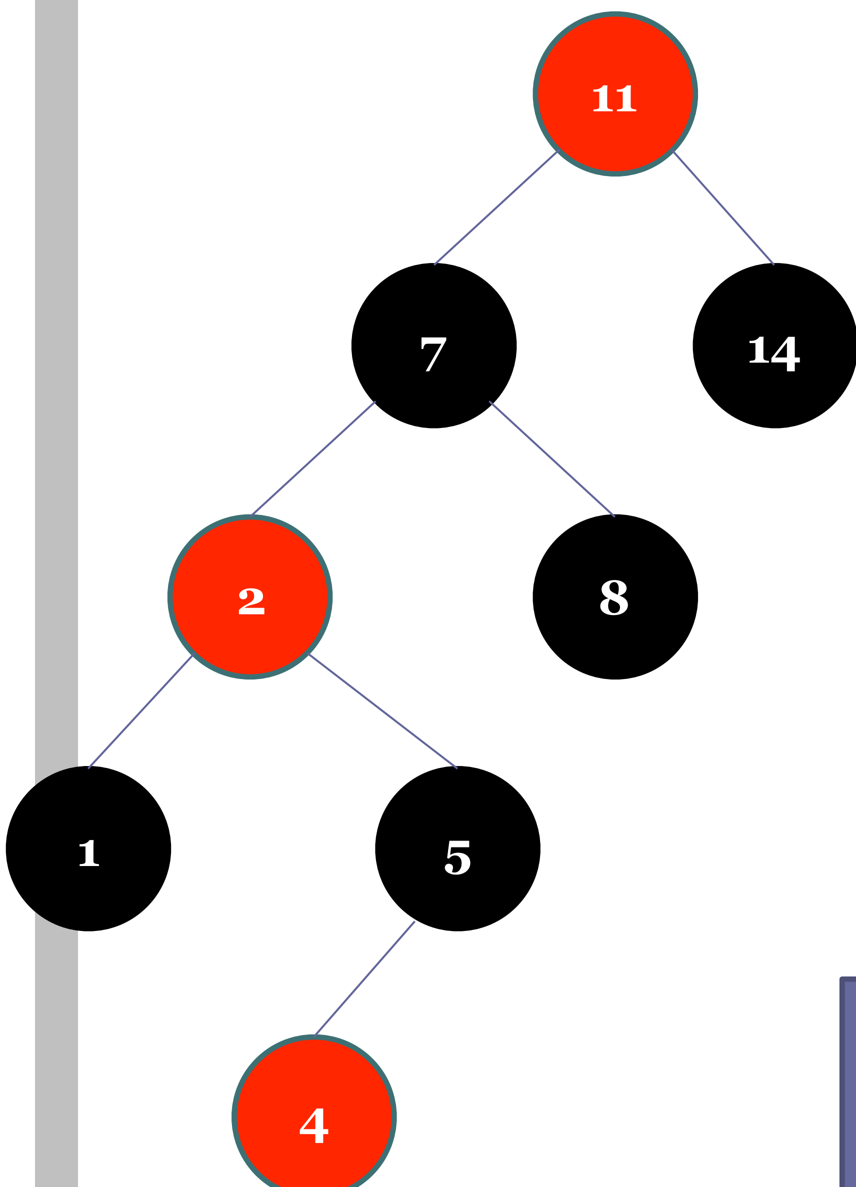
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 3

Change colors

Insättning, ett enkelt exempel



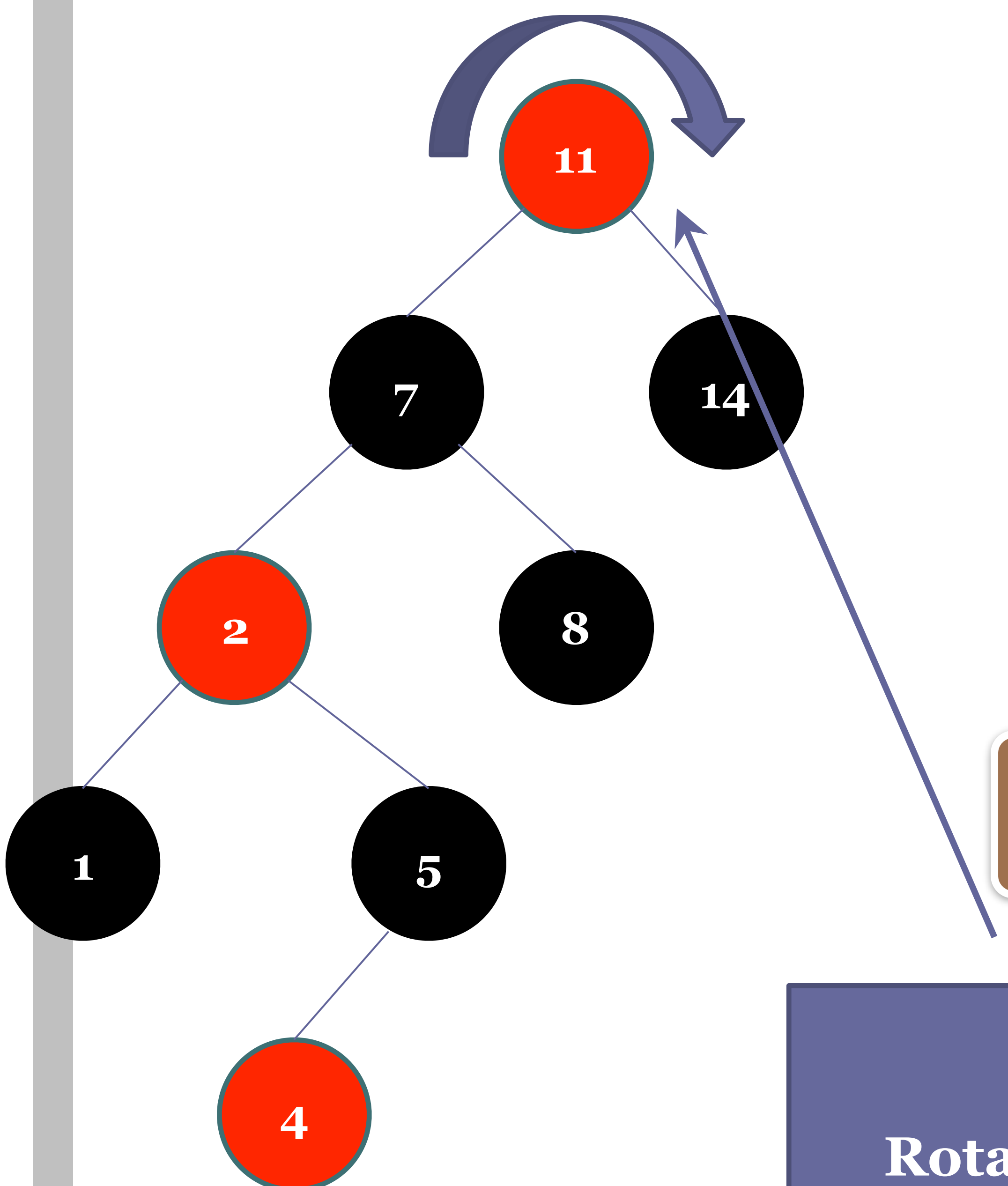
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 3

Change colors

Insättning, ett enkelt exempel



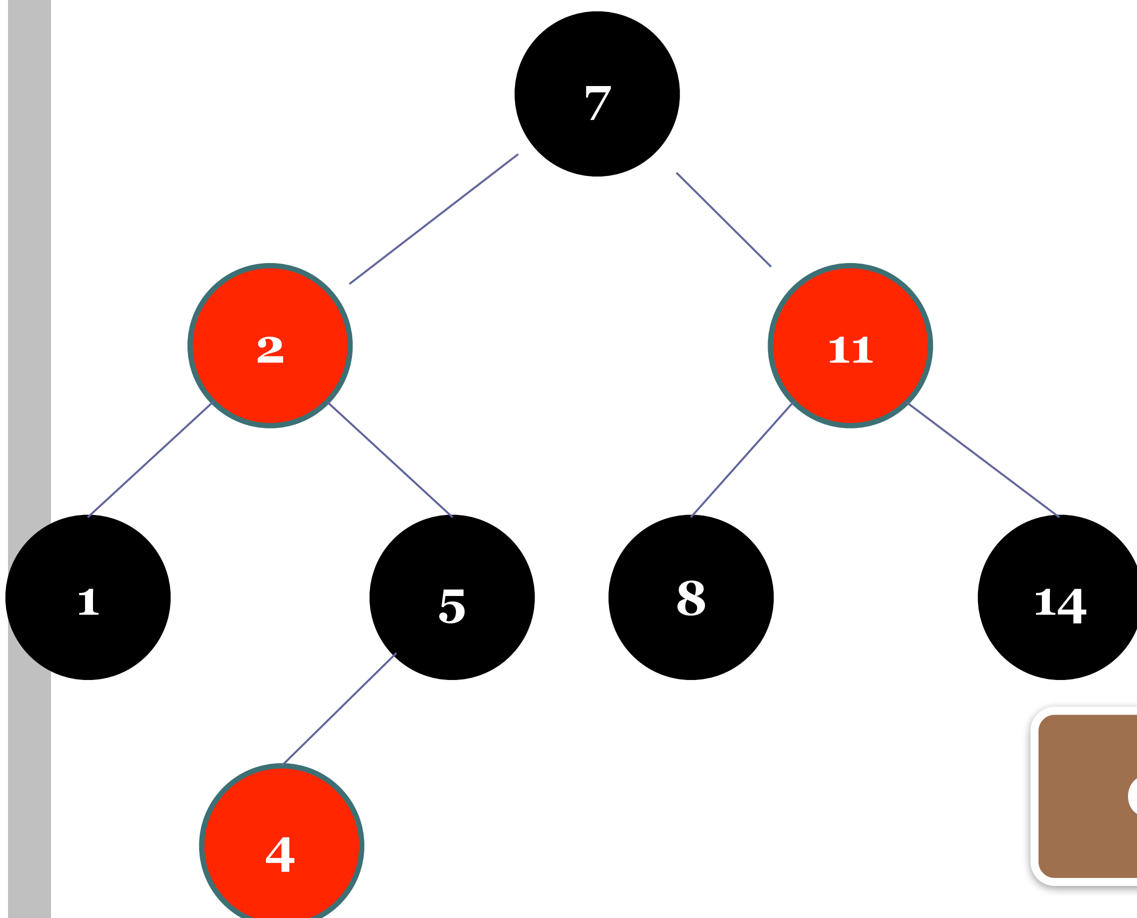
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 3

Rotate right around 11 to restore the balance

Insättning, ett enkelt exempel



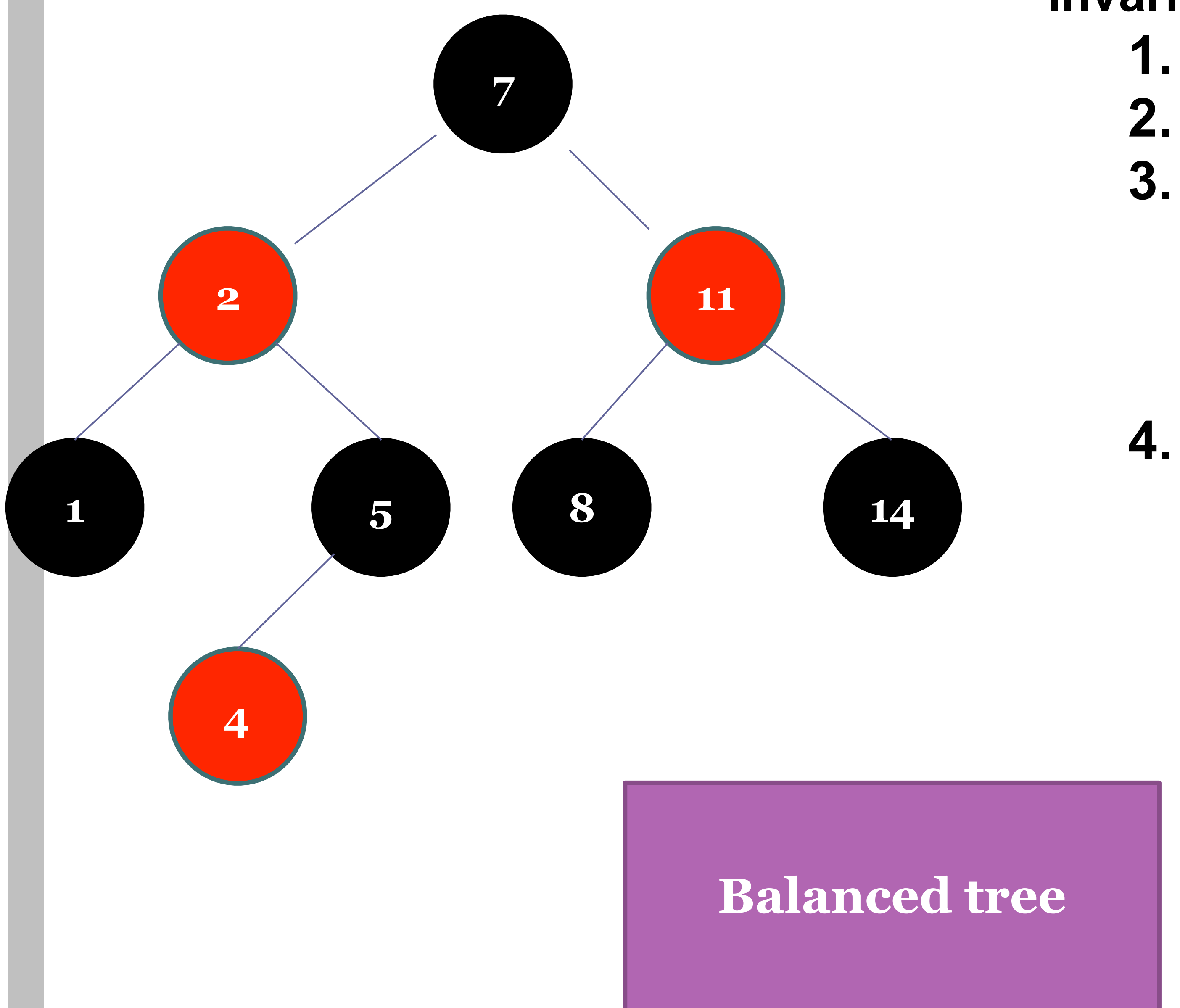
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 3

Rotate right around 11 to restore the balance

Insättning, ett enkelt exempel



Invariants:

1. **A node is either red or black**
2. **The root is always black**
3. **A red node always has black children (a `null` reference is considered to refer to a black node)**
4. **The number of black nodes in any path from the root to a leaf is the same**

Ett större rödsvart exempel

Nu ska vi bygga ett rödsvart träd för orden i

”The quick brown fox jumps over the lazy dog”

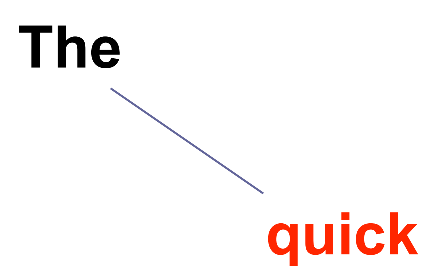
The quick…

The

Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

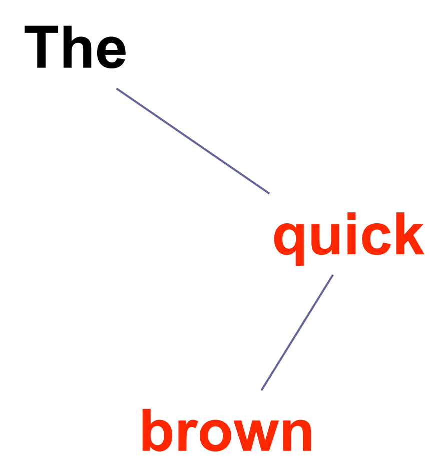
The quick…



Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

The quick brown…

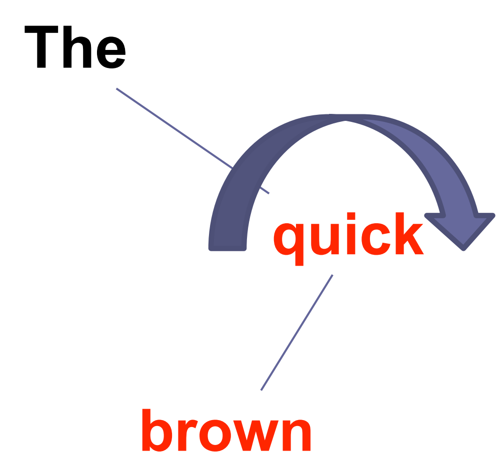


Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 3

The quick brown…



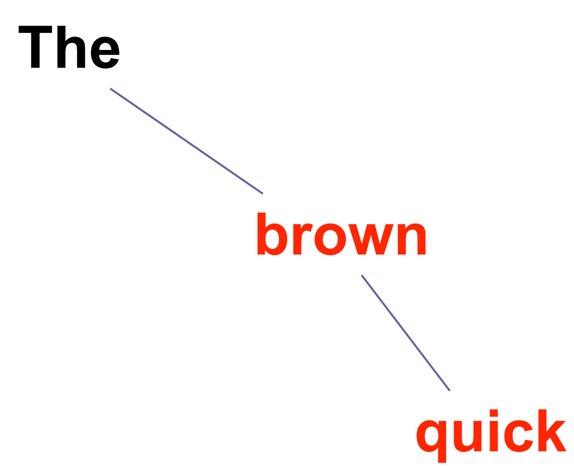
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 3

Rotate so that the child is on the same side of its parent as its parent is to the grandparent

The quick brown…

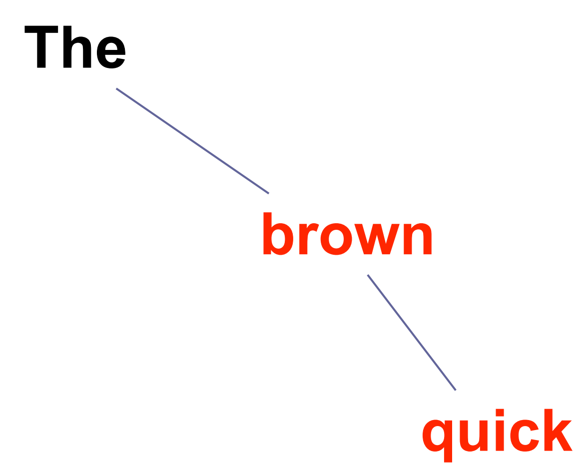


Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 3

The quick brown…



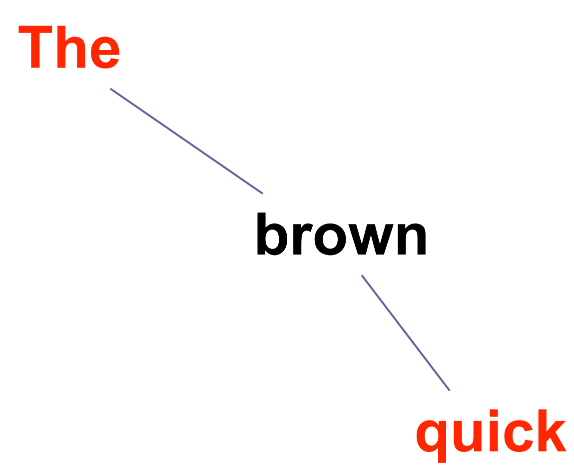
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 3

Change colors

The quick brown…



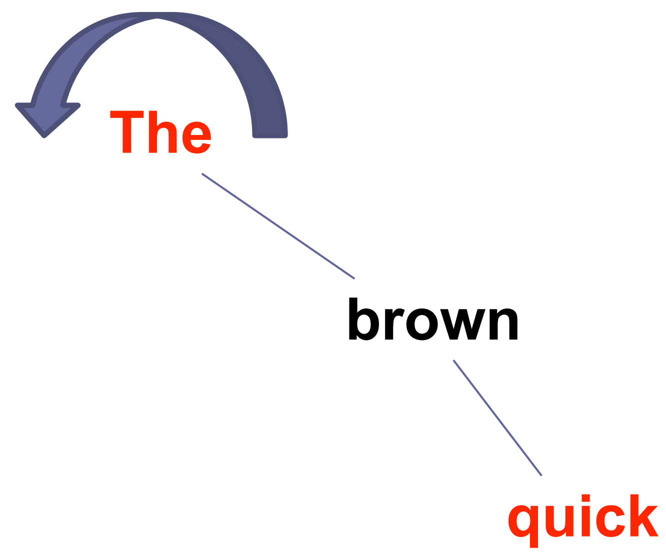
CASE 3

Change colors

Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

The quick brown...



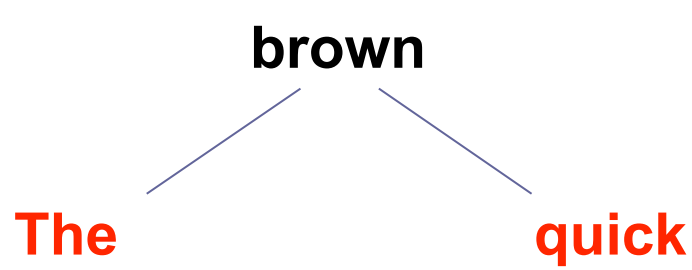
CASE 3

Rotate left

Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

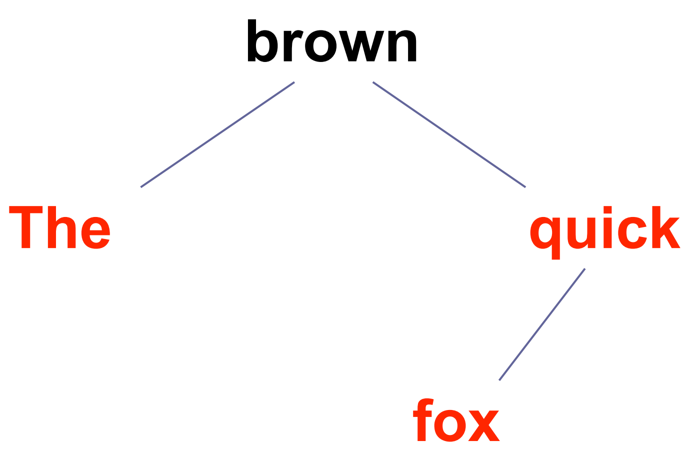
The quick brown...



Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

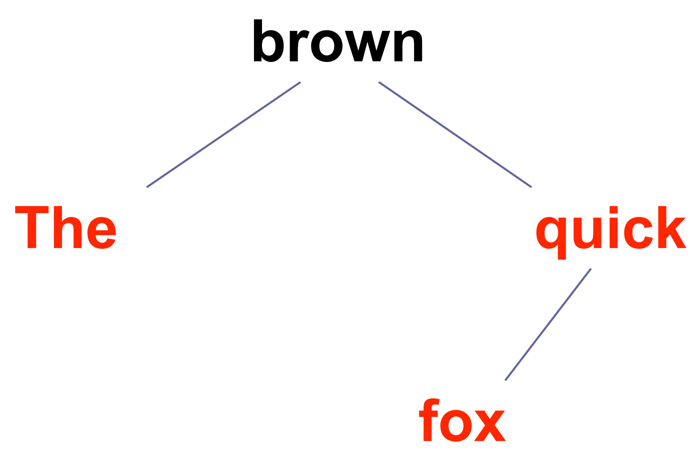
The quick brown fox...



Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

The quick brown fox...



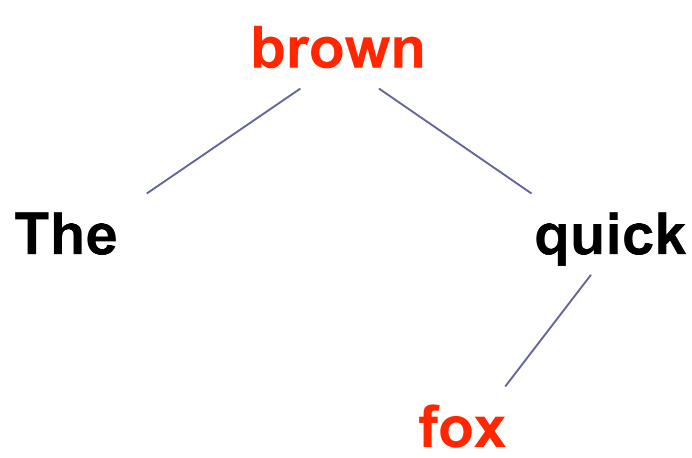
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 1

fox's parent and its parent's sibling are both red. Change colors.

The quick brown fox…



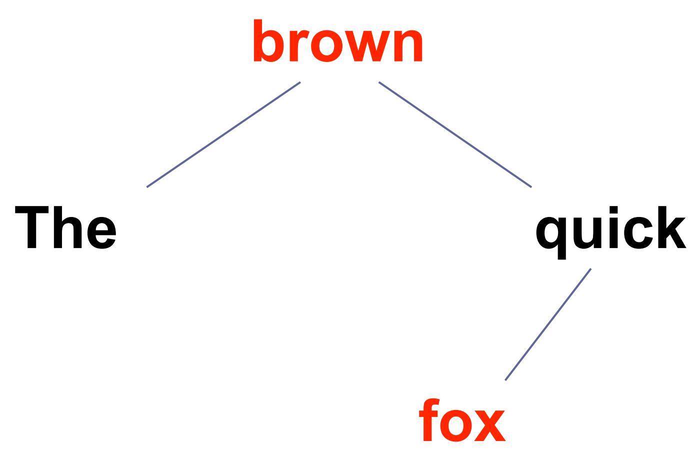
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 1

***fox*'s parent and its parent's sibling are both red. Change colors.**

The quick brown fox...



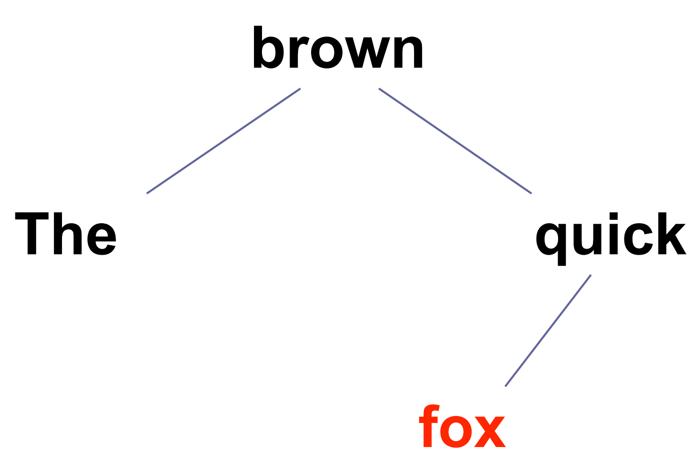
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 1

We can change *brown*'s color to black and not violate #4

The quick brown fox...



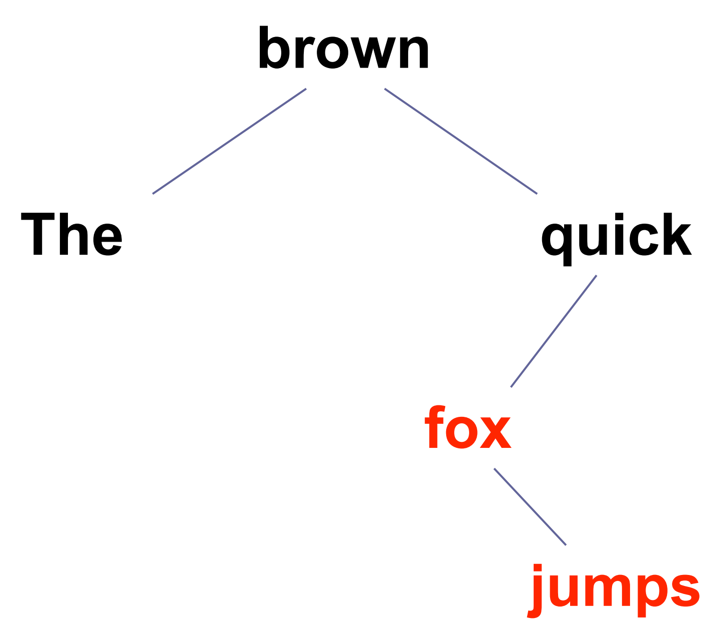
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 1

We can change *brown*'s color to black and not violate #4

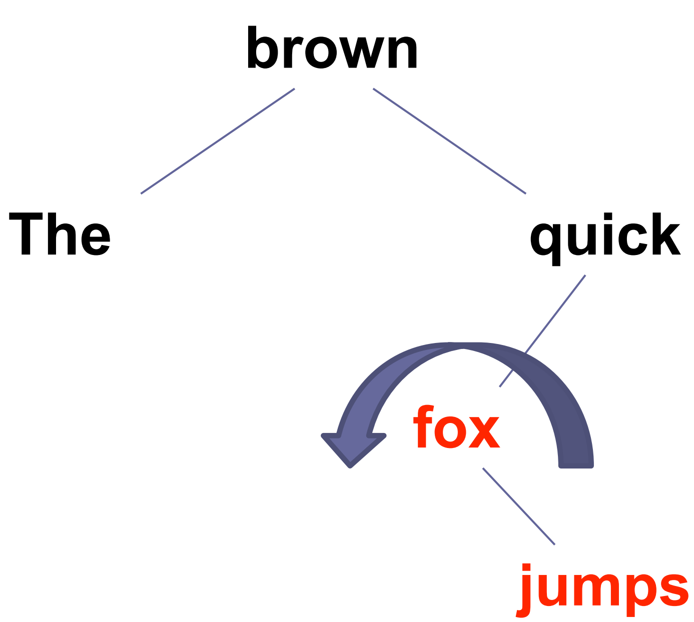
The quick brown fox jumps...



Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

The quick brown fox jumps…



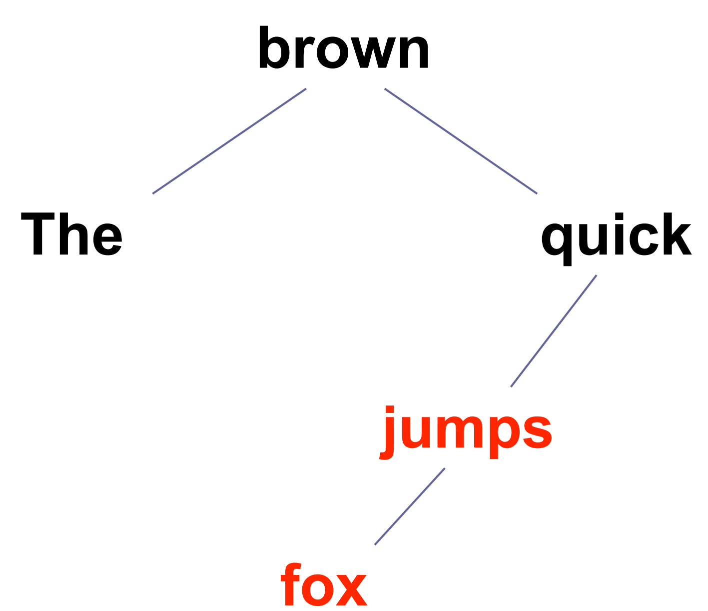
CASE 3

Rotate so that red child is on same side of its parent as its parent is to the grandparent

Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

The quick brown fox jumps…

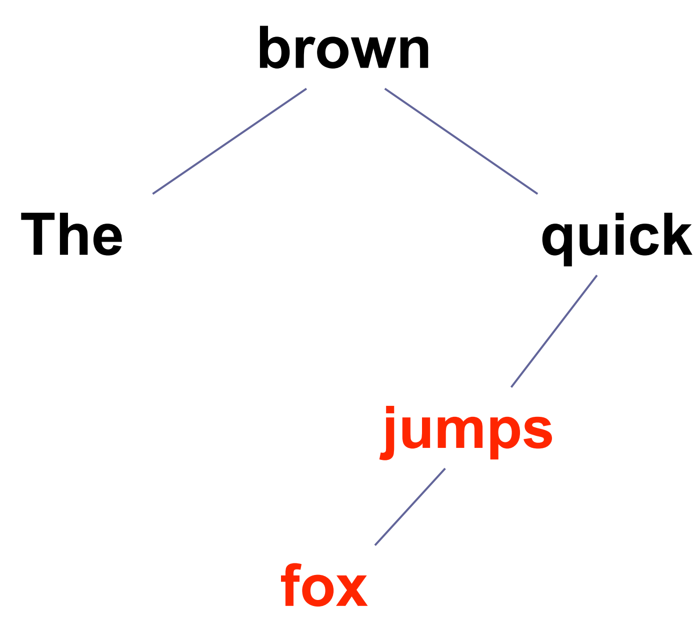


CASE 3

Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

The quick brown fox jumps…



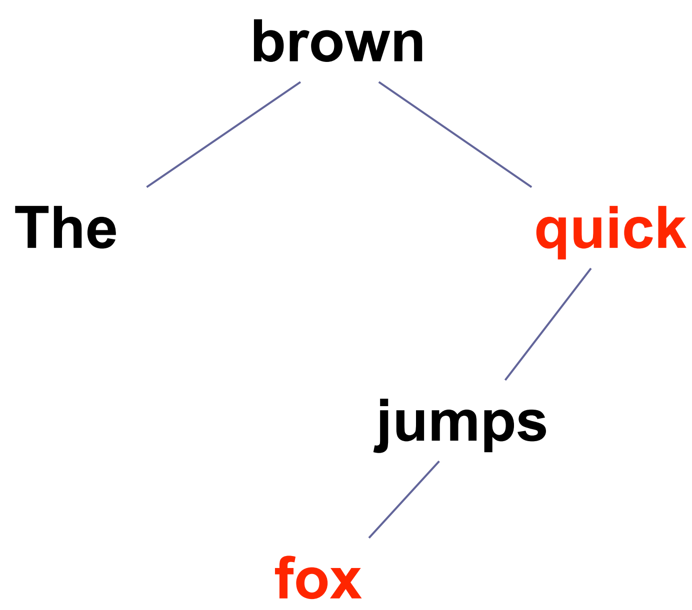
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 3

Change *fox*'s parent and grandparent colors

The quick brown fox jumps…



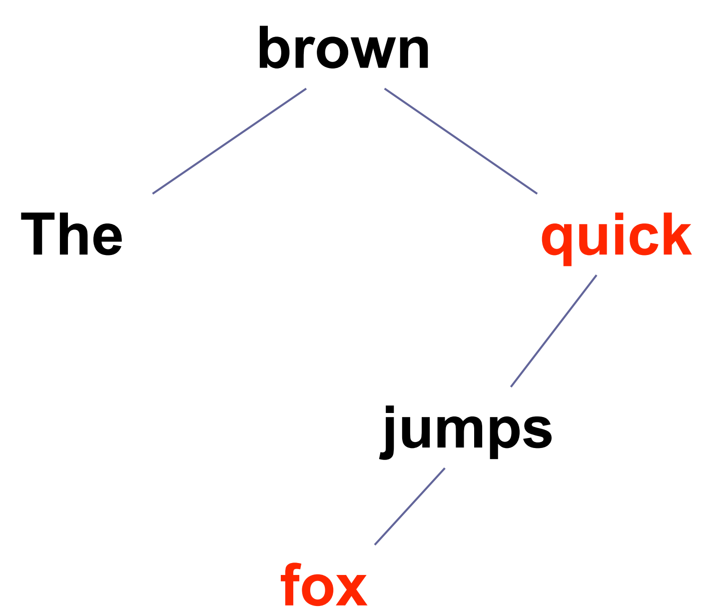
CASE 3

Change *fox*'s parent and grandparent colors

Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

The quick brown fox jumps…



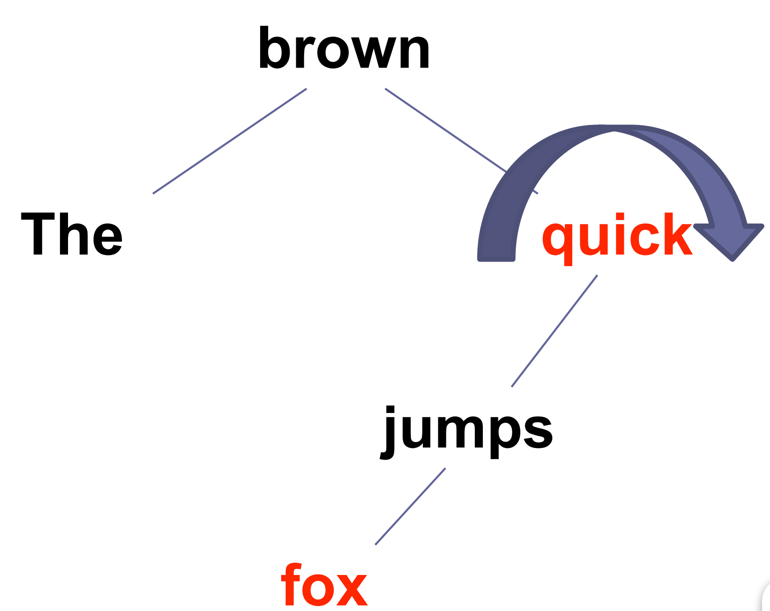
CASE 3

Rotate right about *quick*

Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

The quick brown fox jumps...



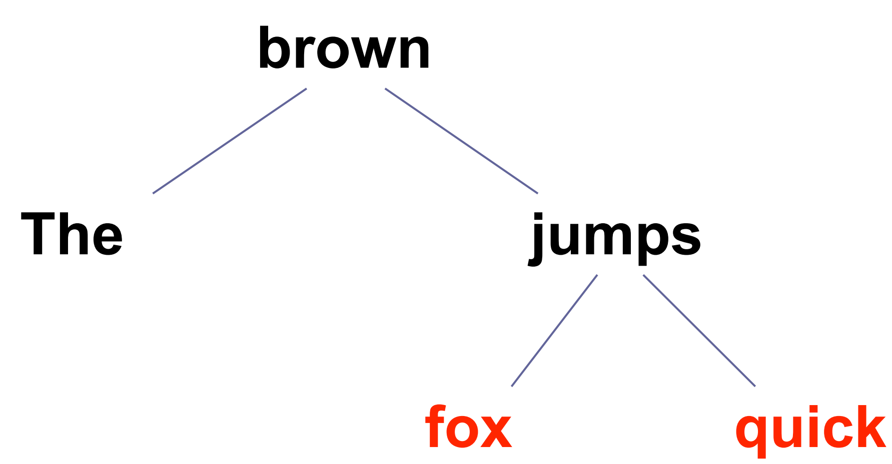
CASE 3

Rotate right about *quick*

Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

The quick brown fox jumps…



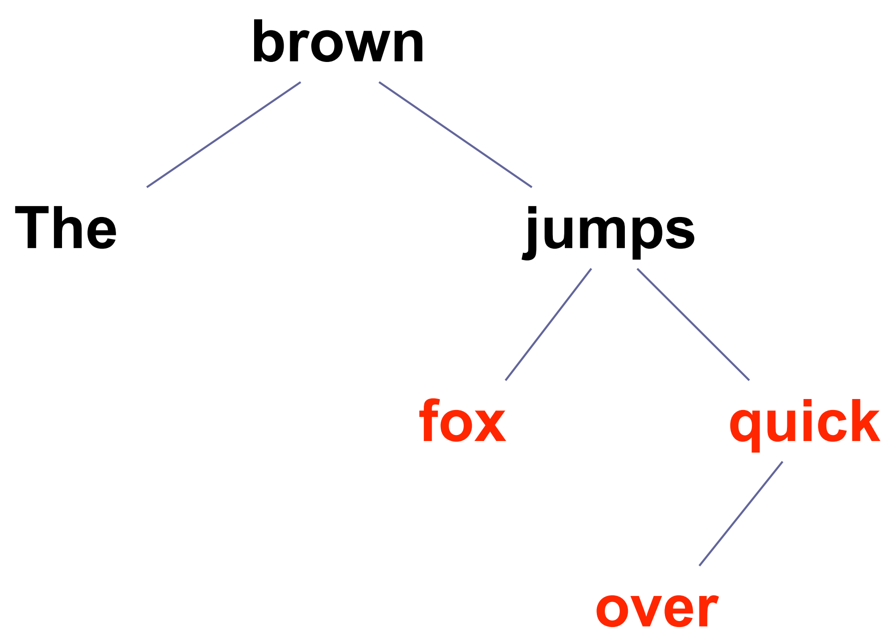
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 3

Rotate right about *quick*

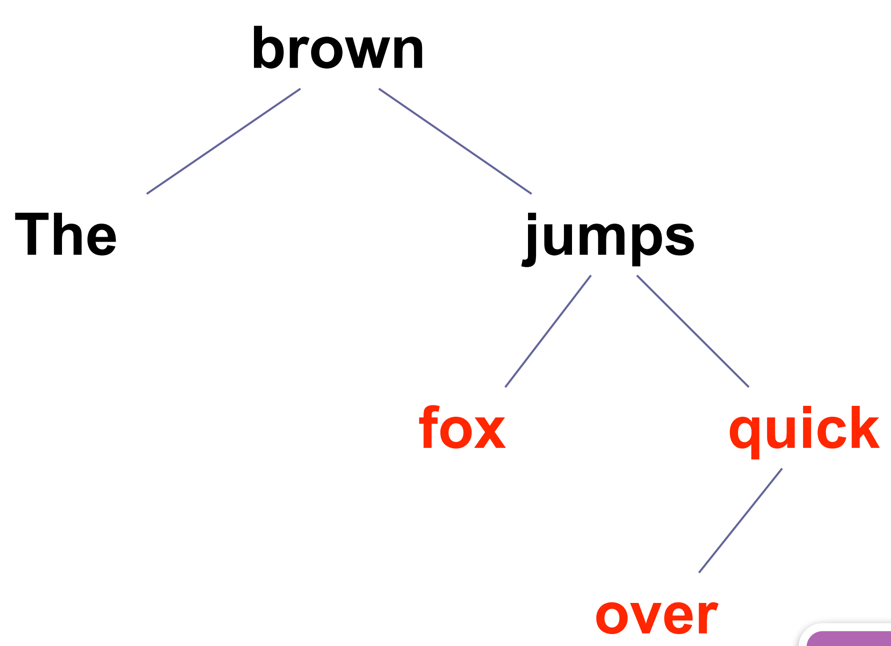
The quick brown fox jumps over…



Invariants:

1. **A node is either red or black**
2. **The root is always black**
3. **A red node always has black children (a `null` reference is considered to refer to a black node)**
4. **The number of black nodes in any path from the root to a leaf is the same**

The quick brown fox jumps over…



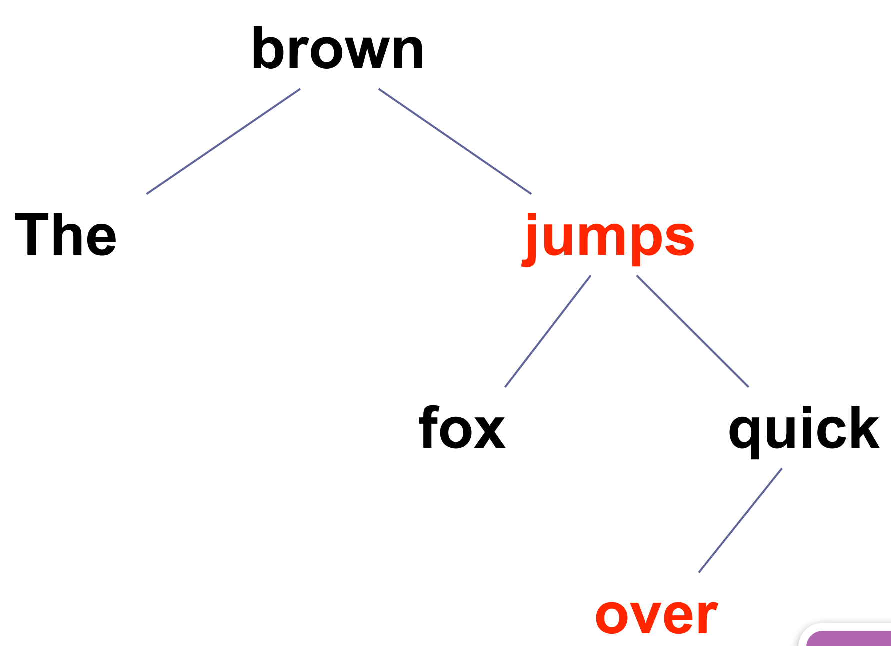
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 1

Change colors of parent, parent's sibling and grandparent

The quick brown fox jumps over…



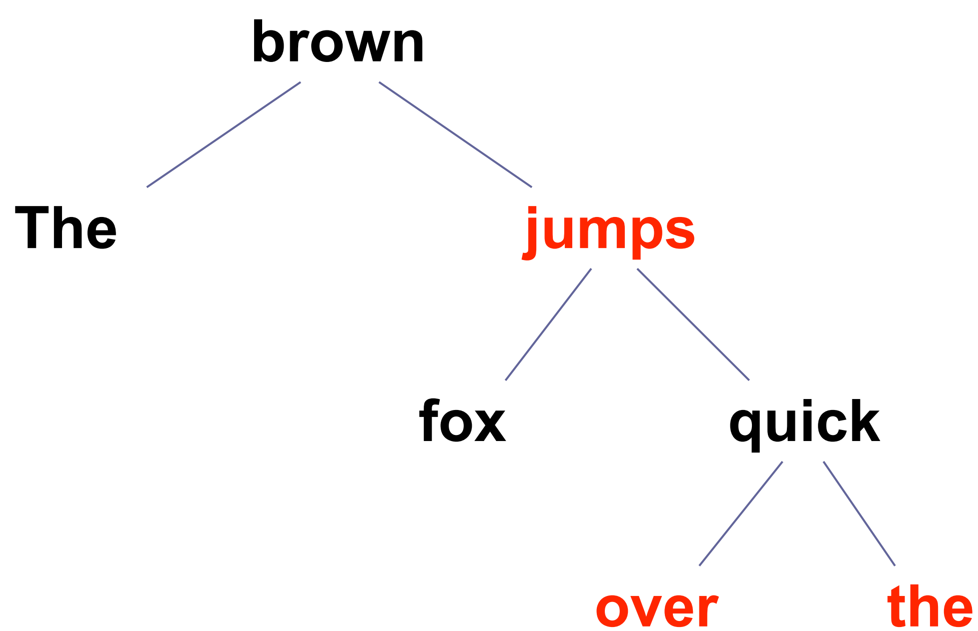
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 1

Change colors of parent, parent's sibling and grandparent

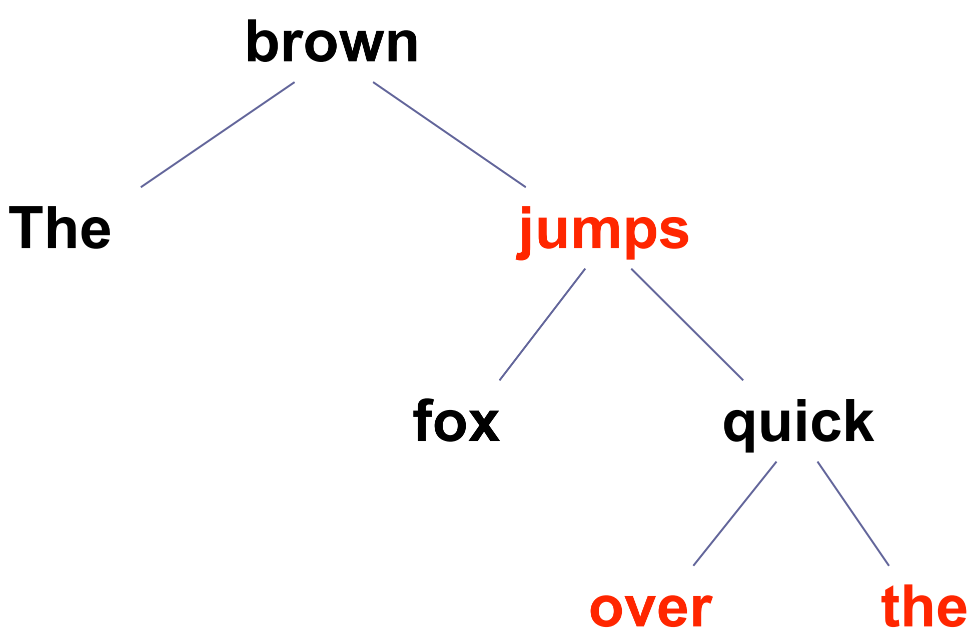
The quick brown fox jumps over the…



Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

The quick brown fox jumps over the…

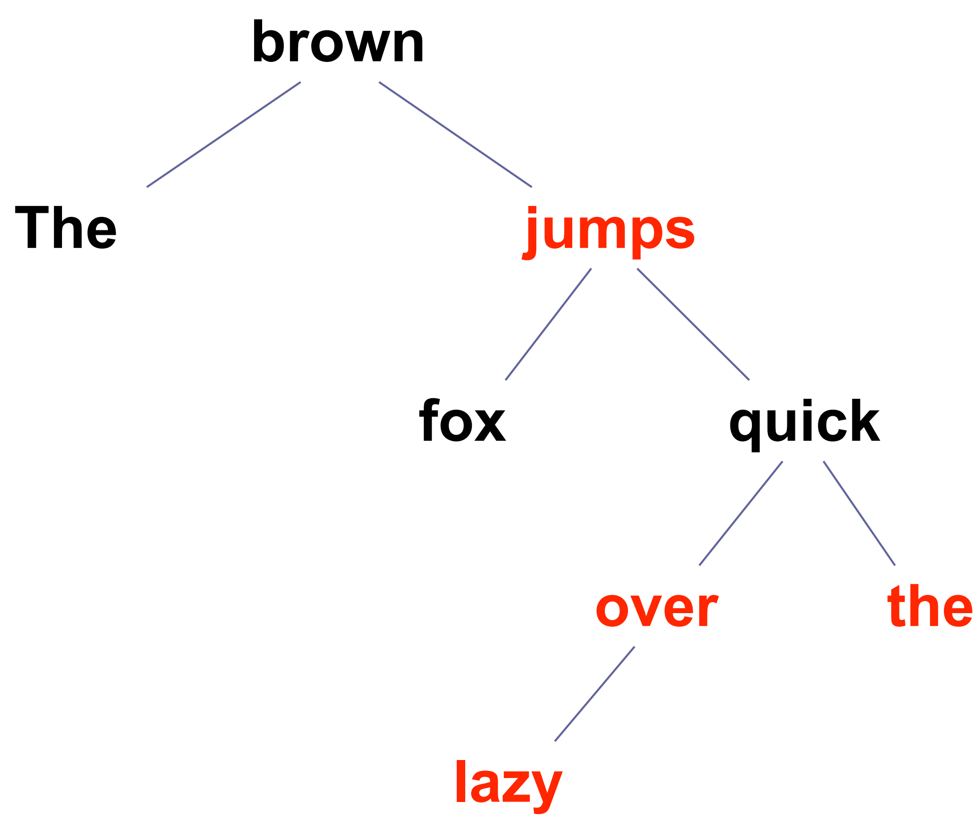


Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

No changes needed

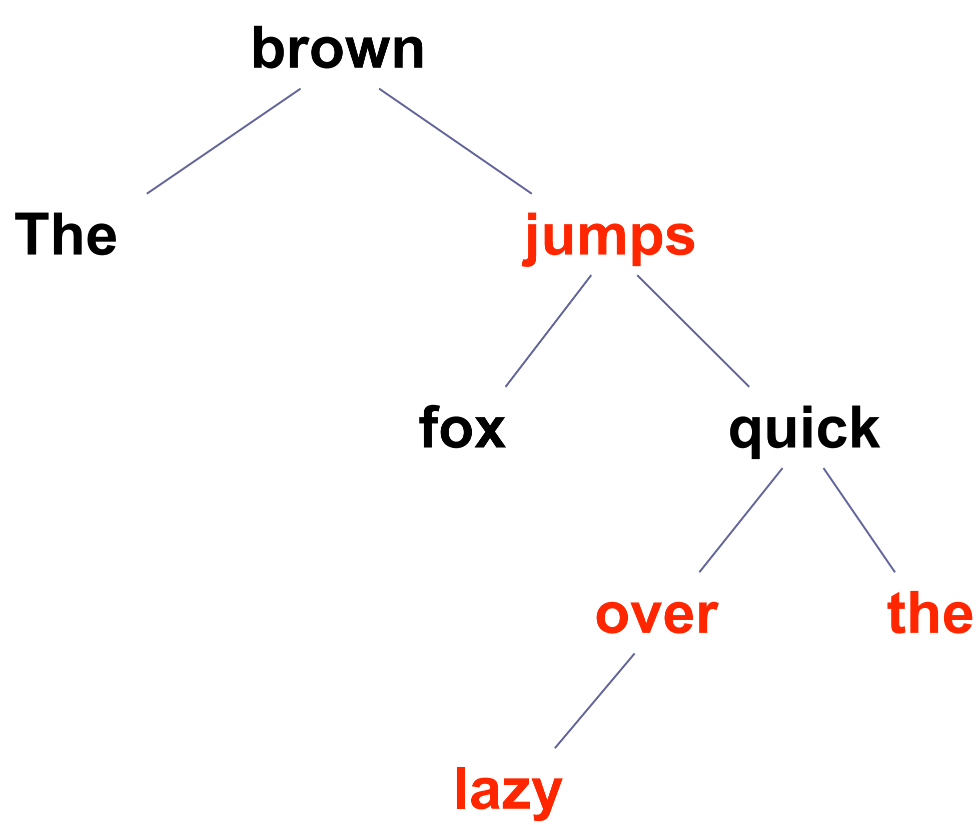
quick brown fox jumps over the lazy…



Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

quick brown fox jumps over the lazy...



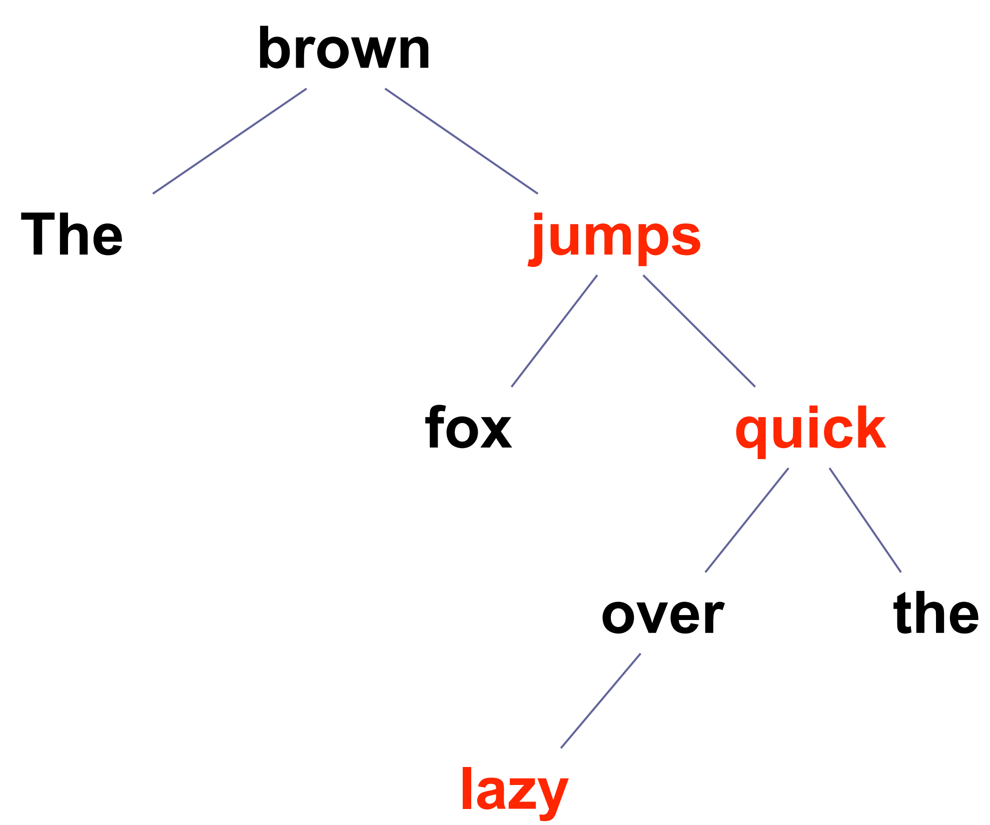
Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 1

Because *over* and *the* are both red, change parent, parent's sibling and grandparent colors

quick brown fox jumps over the lazy…

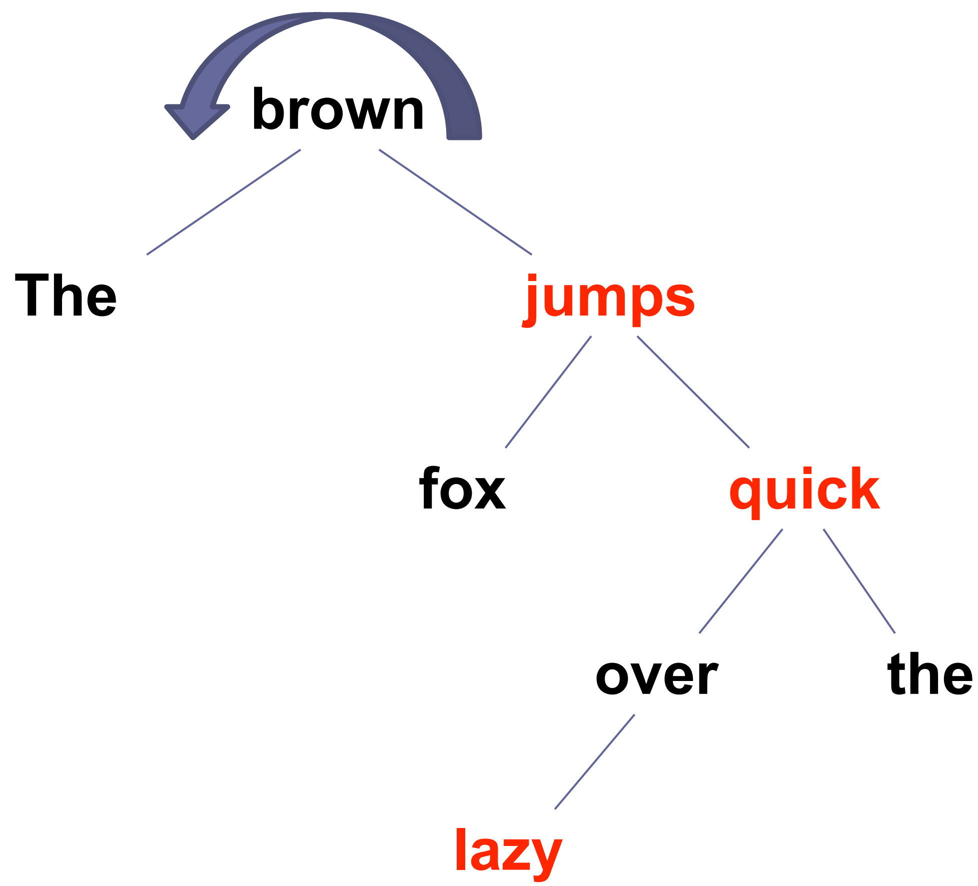


Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 2

quick brown fox jumps over the lazy…

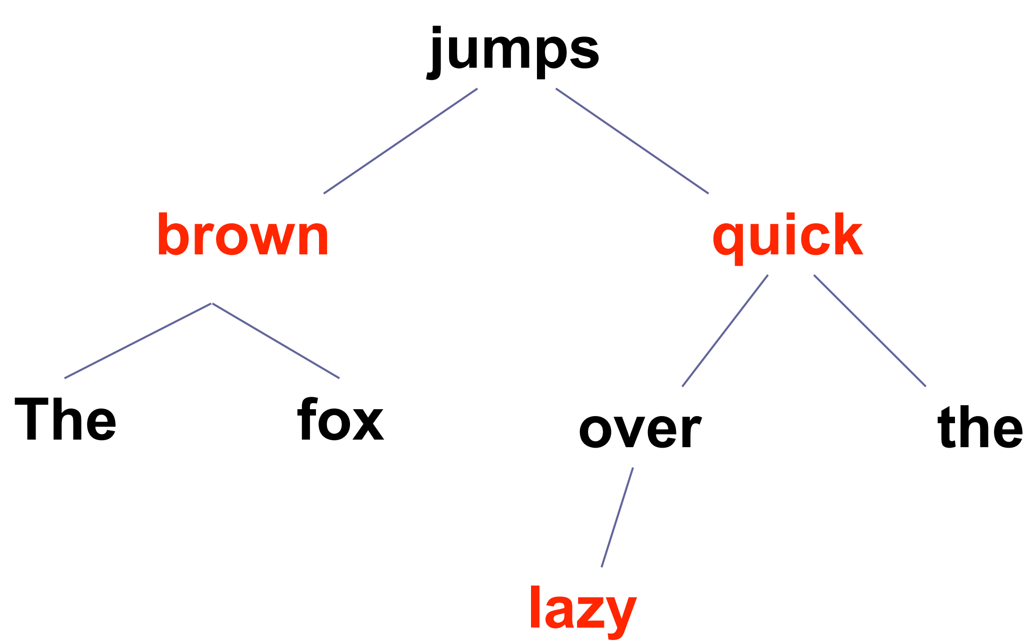


Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 2

quick brown fox jumps over the lazy...

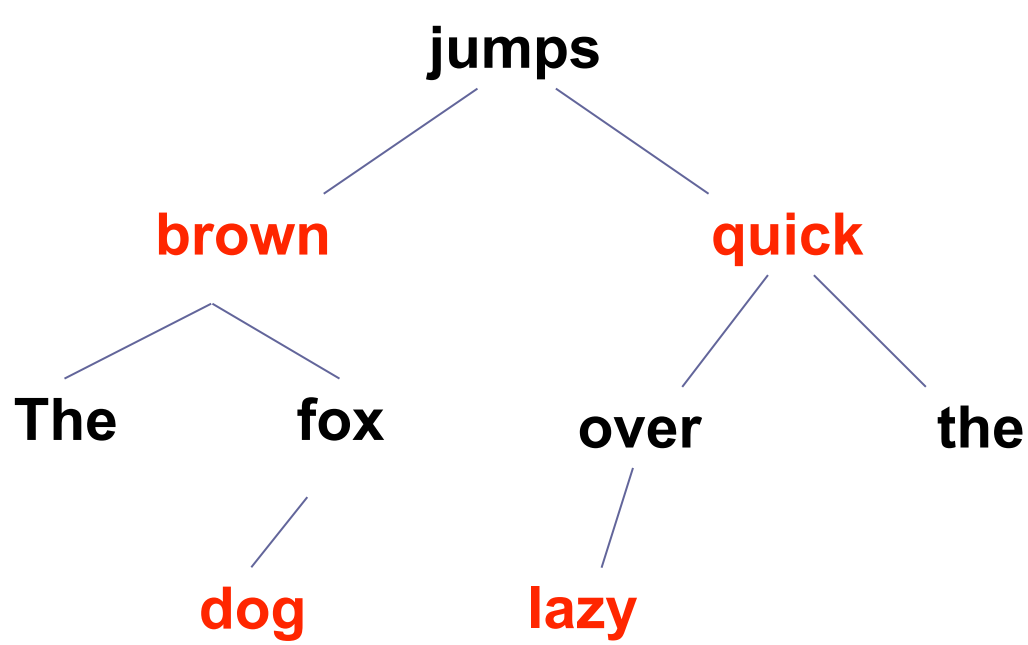


Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

CASE 2

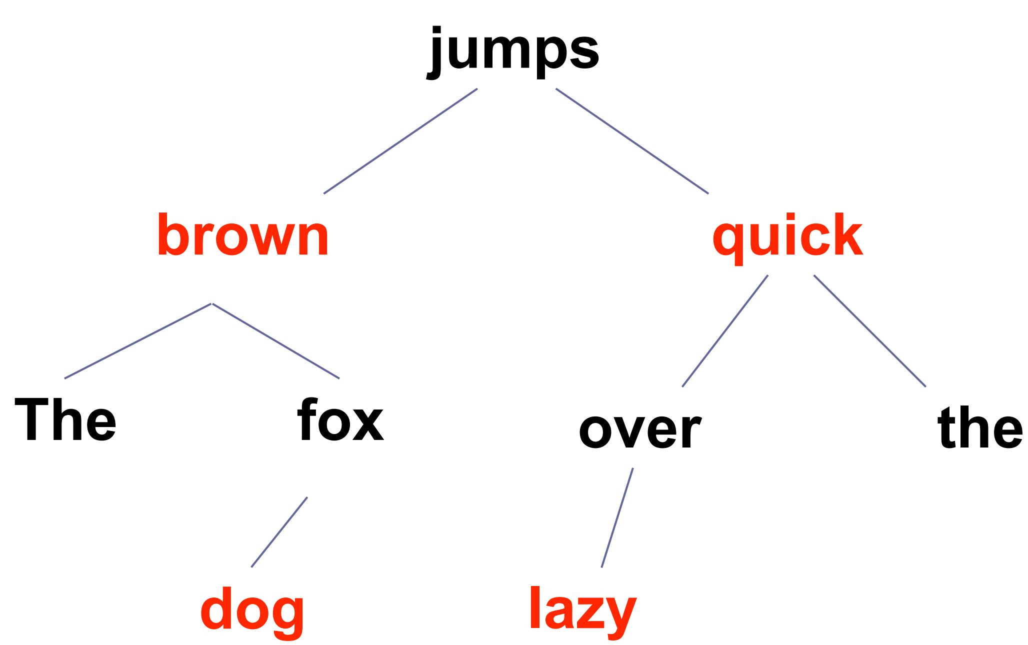
uick brown fox jumps over the lazy dog



Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

...ick brown fox jumps over the lazy dog!

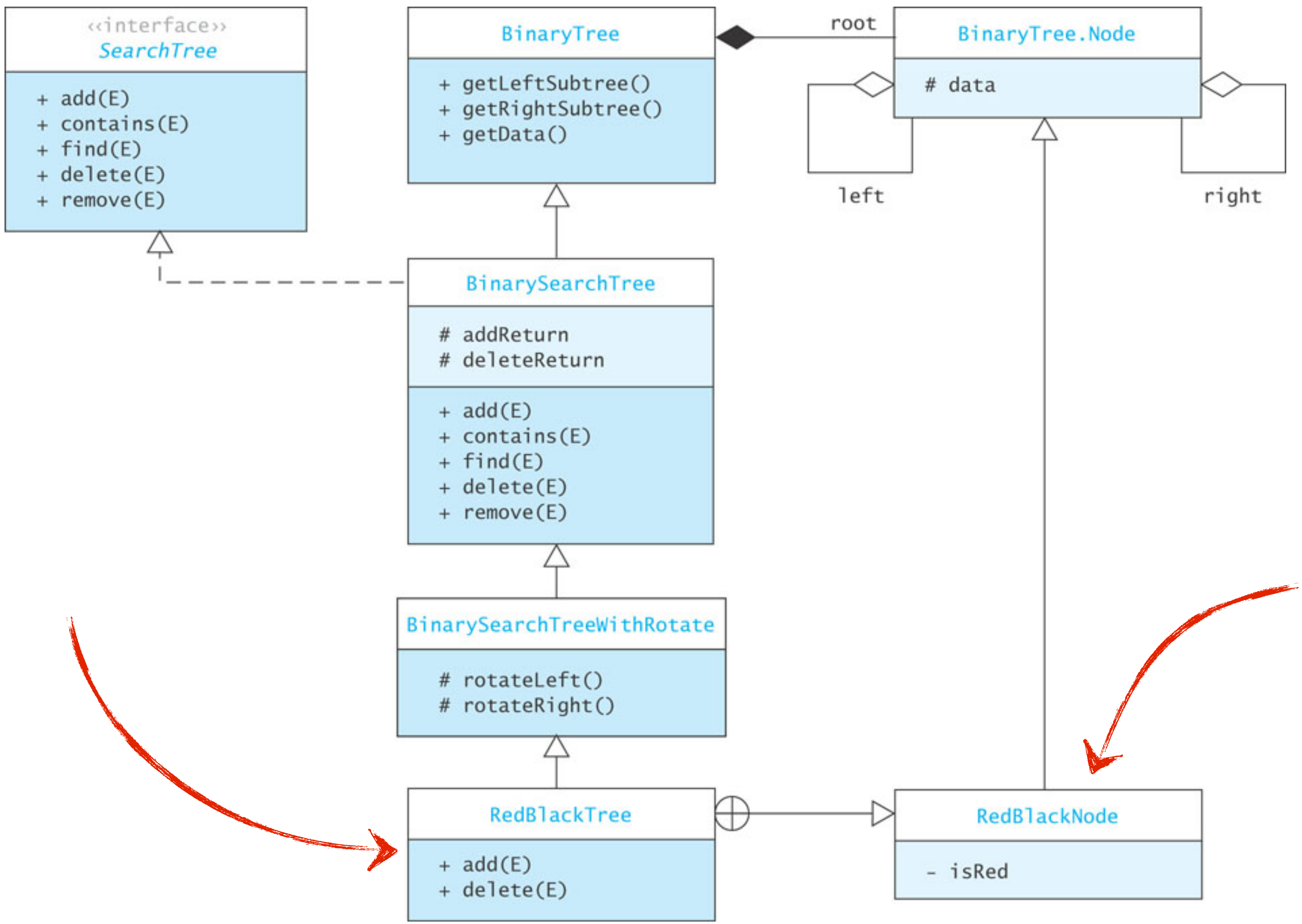


Balanced tree

Invariants:

1. A node is either red or black
2. The root is always black
3. A red node always has black children (a `null` reference is considered to refer to a black node)
4. The number of black nodes in any path from the root to a leaf is the same

Rödsvarta träd, klassdiagram



Insättning, implementering

Insättning kan implementeras enkelt om noderna har en referens till sin förälder

- det blir ungefär som en dubbellänkad lista
- det krävs mer utrymme

Bokens algoritm sätter in noden i barnbarnslöven

- dvs, ”root” refererar till förälderns förälder (G)
- när algoritmen ser en svart nod med två barn på väg ner genom trädet, så färgas noden röd och barnen svarta
- om det blir färgproblem i slutändan så fixas dem på vägen upp genom trädet

Algorithm for Red-Black Tree Insertion

```
1.  if the root is null
2.      Insert a new Red-Black node and color it black.
3.      Return true.
4.  else if the item is equal to root.data
5.      The item is already in the tree; return false.
6.  else if the item is less than root.data
7.      if the left subtree is null
8.          Insert a new Red-Black node as the left subtree and color it red.
9.          Return true.
10.     else
11.         if both the left child and the right child are red

12.             Change the color of the children to black and change local
                root to red.
13.         Recursively insert the item into the left subtree.
14.         if the left child is now red
15.             if the left grandchild is now red (grandchild is an "out-
                side" node)
16.                 Change the color of the left child to black and
                    change the local root to red.
17.                 Rotate the local root right.
18.             else if the right grandchild is now red (grandchild is an
                "inside" node)
19.                 Rotate the left child left.
20.                 Change the color of the left child to black and
                    change the local root to red.
21.                 Rotate the local root right.
22. else
23.     Item is greater than root.data; process is symmetric and is left as an
        exercise.
24. if the local root is the root of the tree
25.     Force its color to be black.
```

Algorithm for Red-Black Tree Insertion

```
1.  if the root is null
2.      Insert a new Red-Black node and color it black.
3.      Return true.
4.  else if the item is equal to root.data
5.      The item is already in the tree; return false.
6.  else if the item is less than root.data
7.      if the left subtree is null
8.          Insert a new Red-Black node as the left subtree and color it red.
9.          Return true.
10.     else
11.         if both the left child and the right child are red
12.             Change the color of the children to black and change local
                root to red.
13.         Recursively insert the item into the left subtree.
14.         if the left child is now red
15.             if the left grandchild is now red (grandchild is an "out-
                side" node)
16.                 Change the color of the left child to black and
                    change the local root to red.
17.                 Rotate the local root right.
18.             else if the right grandchild is now red (grandchild is an
                "inside" node)
19.                 Rotate the left child left.
20.                 Change the color of the left child to black and
                    change the local root to red.
21.                 Rotate the local root right.
22. else
23.     Item is greater than root.data; process is symmetric and is left as an
        exercise.
24. if the local root is the root of the tree
25.     Force its color to be black.
```

Fall 1: båda föräldrarna är röda

Algorithm for Red-Black Tree Insertion

```
if the root is null
    Insert a new Red-Black node and color it black.
    Return true.
else if the item is equal to root.data
    The item is already in the tree; return false.
else if the item is less than root.data
    if the left subtree is null
        Insert a new Red-Black node as the left subtree and color it red.
        Return true.
    else
        if both the left child and the right child are red

            Change the color of the children to black and change local
            root to red.
        Recursively insert the item into the left subtree.
        if the left child is now red
            if the left grandchild is now red (grandchild is an "out-
            side" node)
                Change the color of the left child to black and
                change the local root to red.
                Rotate the local root right.
            else if the right grandchild is now red (grandchild is an
            "inside" node)
                Rotate the left child left.
                Change the color of the left child to black and
                change the local root to red.
                Rotate the local root right.
else
    Item is greater than root.data; process is symmetric and is left as an
    exercise.
if the local root is the root of the tree
    Force its color to be black.
```

Fall 1: båda föräldrarna är röda

Fall 2: vänster-vänsterbarn; förälderns syskon är svart

Algorithm for Red-Black Tree Insertion

```
if the root is null
    Insert a new Red-Black node and color it black.
    Return true.
else if the item is equal to root.data
    The item is already in the tree; return false.
else if the item is less than root.data
    if the left subtree is null
        Insert a new Red-Black node as the left subtree and color it red.
        Return true.
    else
        if both the left child and the right child are red

            Change the color of the children to black and change local
            root to red.
        Recursively insert the item into the left subtree.
        if the left child is now red
            if the left grandchild is now red (grandchild is an "out-
            side" node)
                Change the color of the left child to black and
                change the local root to red.
                Rotate the local root right.
            else if the right grandchild is now red (grandchild is an
            "inside" node)
                Rotate the left child left.
                Change the color of the left child to black and
                change the local root to red.
                Rotate the local root right.
else
    Item is greater than root.data; process is symmetric and is left as an
    exercise.
if the local root is the root of the tree
    Force its color to be black.
```

Fall 1: båda föräldrarna är röda

Fall 2: vänster-vänsterbarn; förälderns syskon är svart

Fall 3: vänster-högerbarn; förälderns syskon är svart

Borttagning från ett rödsvart träd

Om noden har två icke-tomma barn:

- ersätt noden med inorder-föregångaren
- ta bort inorder-föregångarens nod istället
- (precis som för vanliga sökträd)

Alltså kan vi anta att noden har max ett icke-tomt barn:

- om noden är röd så har den inga barn och vi behöver inte göra något mer
- om noden är svart och har ett rött barn så flyttar vi upp det röda barnet och färgar det svart
- om noden är svart och inte har några barn så måste trädet balanseras

Effektivitet hos rödsvarta träd

Höjden på ett rödsvart träd har en övre gräns:

- maximalt är höjden $2 \cdot \log_2 n + 2$, vilket är logaritmiskt, $O(\log n)$
- precis om med AVL-träd så det i medeltal mycket bättre än så
- empiriska studier har visat att medelkostnaden för att söka i ett slumpmässigt rödsvart träd är $1{,}002 \cdot \log_2 n$

Javas API har klasserna TreeMap och TreeSet som är implementerad med rödsvarta träd