بسم الله الرحمن الرحیم

ساختمان‌های داده

جلسه ۲۰

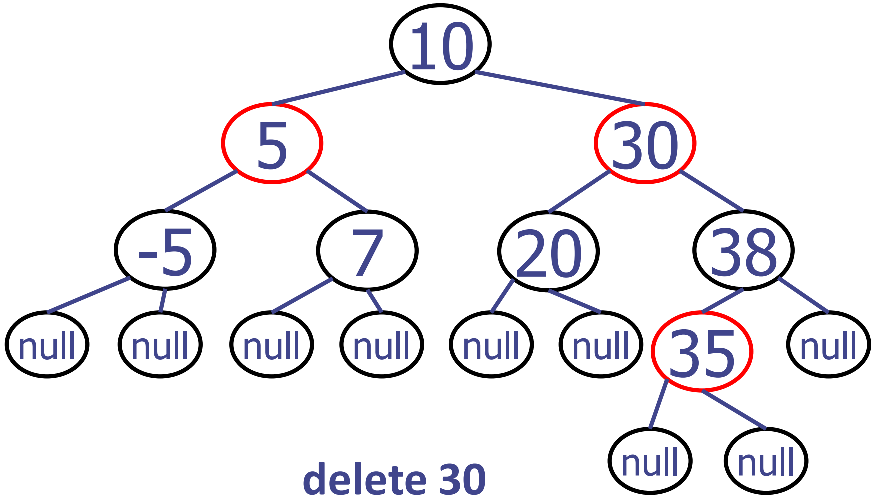
مجتبی خلیلی
دانشکده برق و کامپیوتر
دانشگاه صنعتی اصفهان

RB-Tree: Deletion

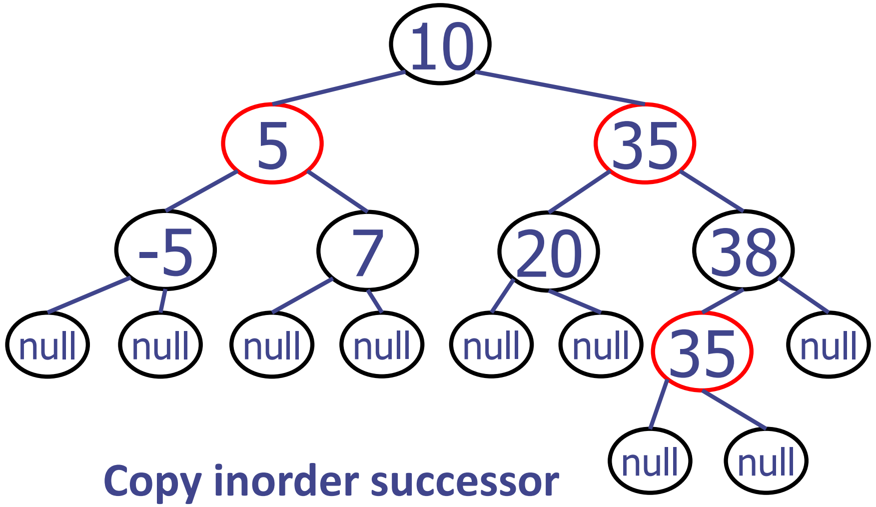
Deletion: Example 1

IUT-ECE

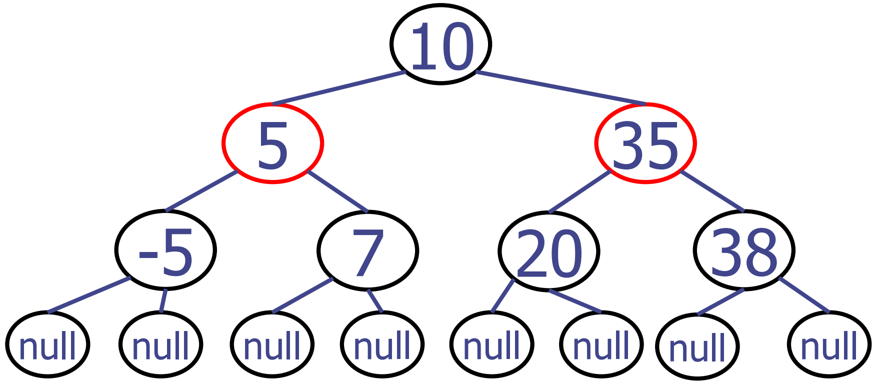
- To perform operation erase(k), we first execute the deletion algorithm for binary search trees



delete 30



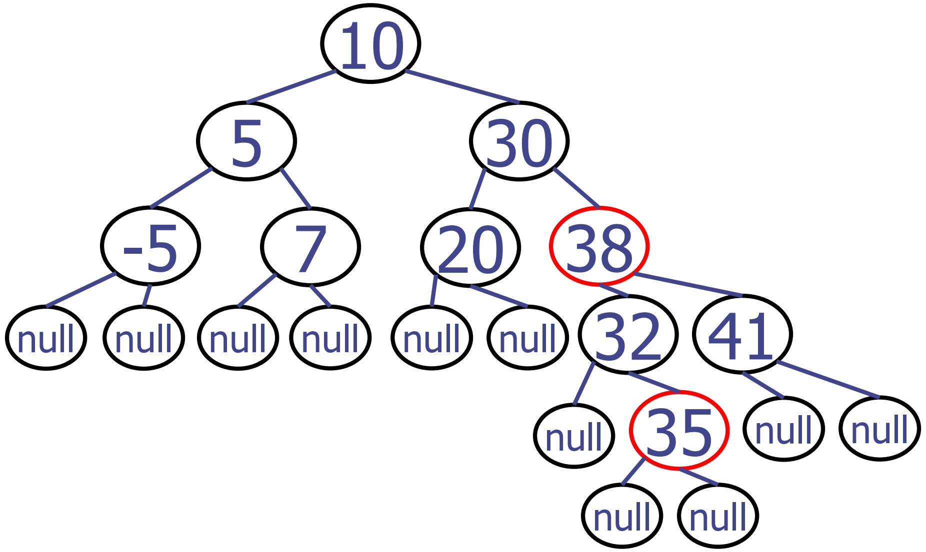
Copy inorder successor



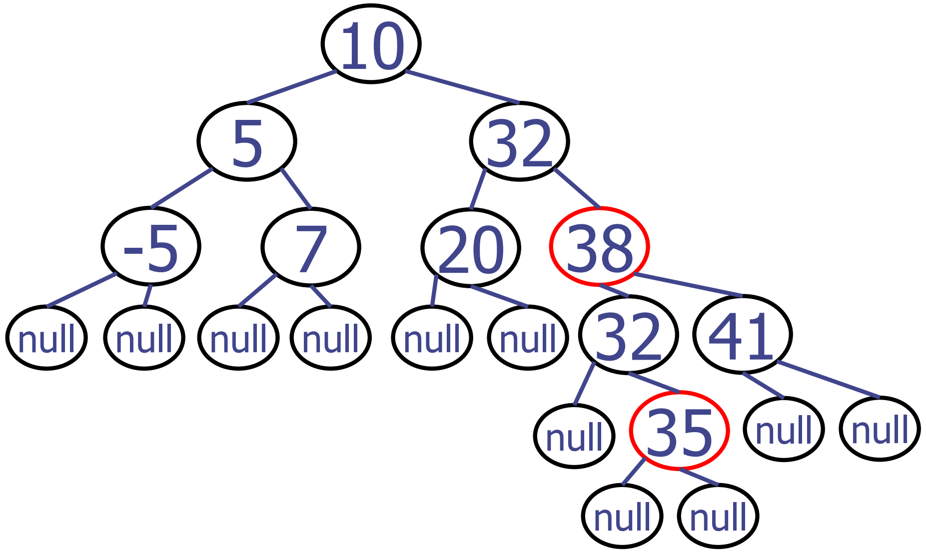
Just delete the copied 35, and color the remaining node in black. Then, we are done.

Implication:
<u>If the node to be deleted is red</u>, removing it is fine

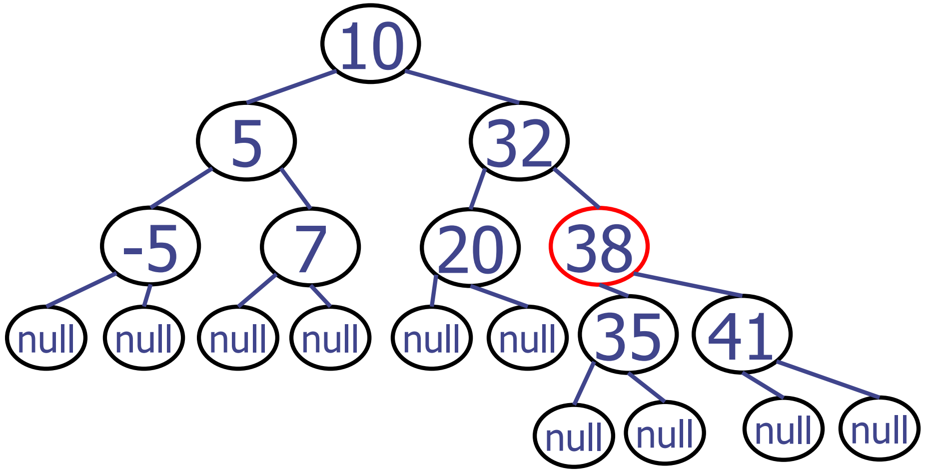
Deletion: Example 2



delete 30



Copy inorder successor

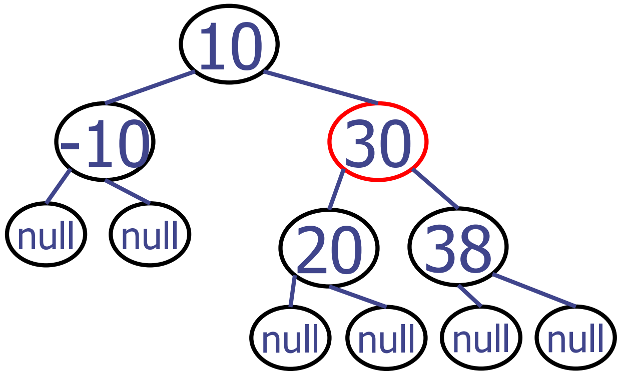


Just delete the copied 32, and color 35 with black.

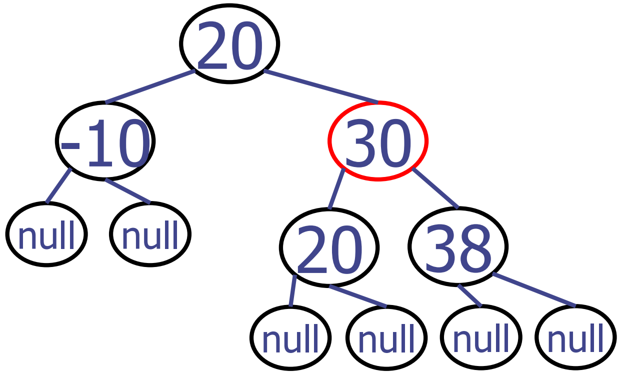
Implication: <u>**For a node (with a red child) to be deleted**</u>, delete it and change the red child's color.
(35: -1 first and +1 second. So no change)

Deletion: Example 3

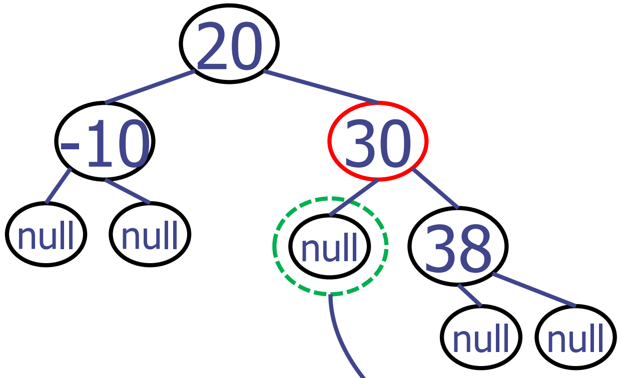
- What about deleting **a node with a black child**?



Delete 10



Copy inorder successor



Delete 20.

Problem: A path of only 2 blacks

Regard this as “double black nodes”

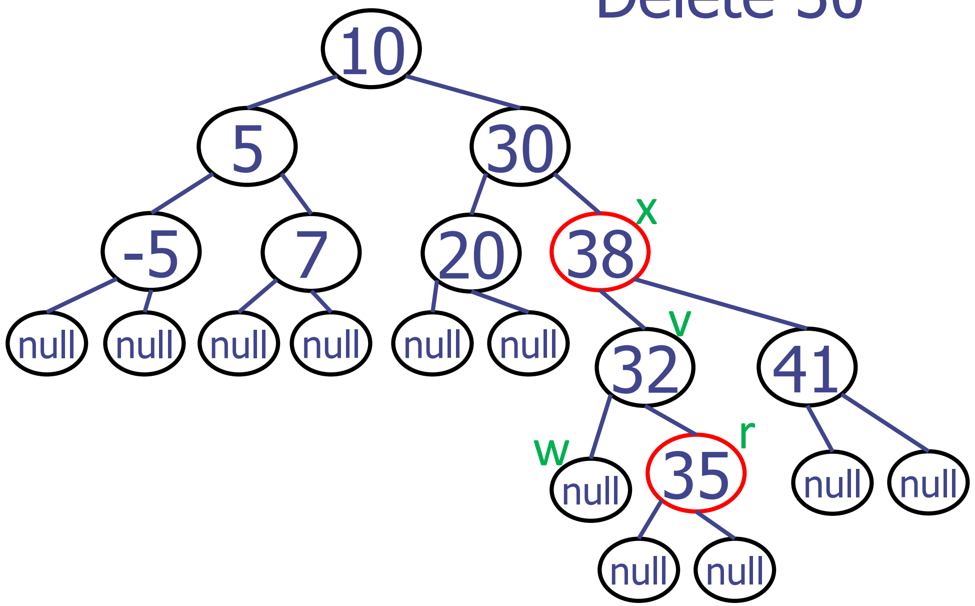
IUT-ECE

Mojtaba Khalili

Deletion

- To perform operation erase(k), we first execute the deletion algorithm for binary search trees
  - Enough to consider the removal of an entry at a node with an external child
    (To remove a node with both internal children, we first copy the inorder successor, and then …)
- Notations
  - v : the internal node removed,
    - "myself"
  - w : the external node removed,
    - "my lonely child"
  - r : the sibling of w
    - "my other child"
  - x : the parent of v
    - "my father"

Delete 30



Questions

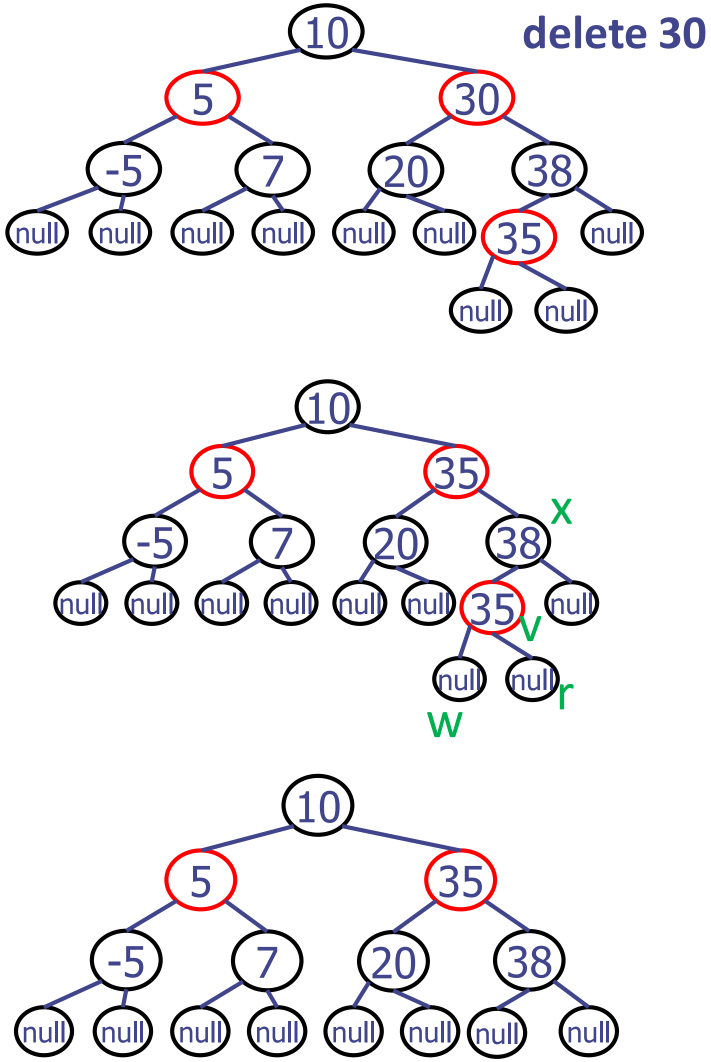
- How to handle “double black nodes”
- Are there some cases in handling those? Yes
- Are you ready for ”cases”?
- It’s really, really complex, but if you concentrate, then you can follow it.

Deletion: Algorithm Overview (1)

First, remove v and w, and make r a child of x

If either of ***v*** or ***r*** was red, we color ***r*** black and we are done (Examples 1 and 2)

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 (Examples 3)



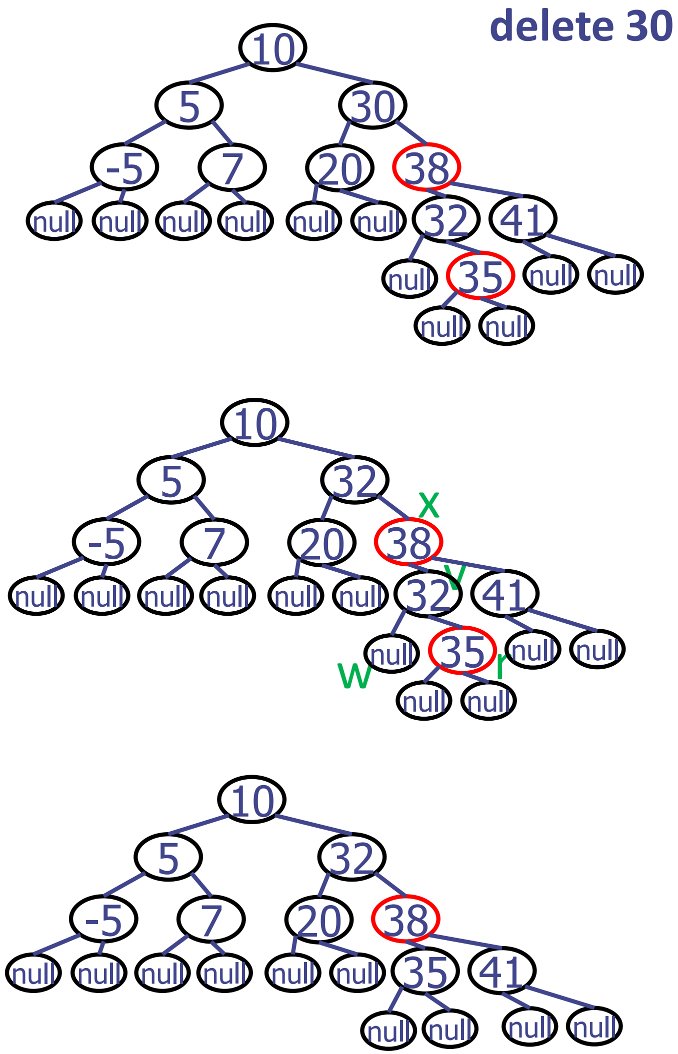
Deletion: Algorithm Overview (2)

IUT-ECE

First, remove v and w, and make r a child of x

If either of ***v*** or ***r*** was red, we color ***r*** black and we are done (Examples 1 and 2)

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 (Examples 3)



Deletion: Algorithm Overview (2)

First, remove v and w, and make r a child of x

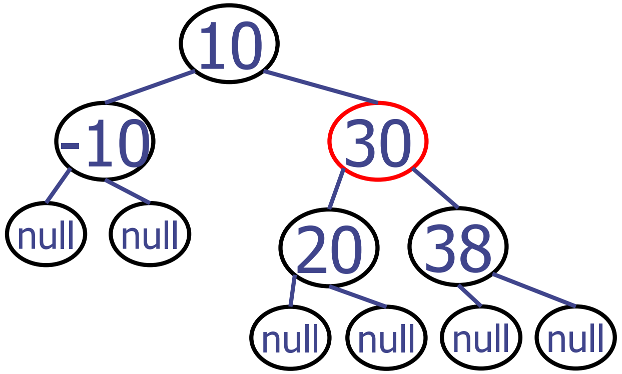
If either of ***v*** or ***r*** was red, we color ***r*** black and we are done (Examples 1 and 2)
(Let's call this Case 0)

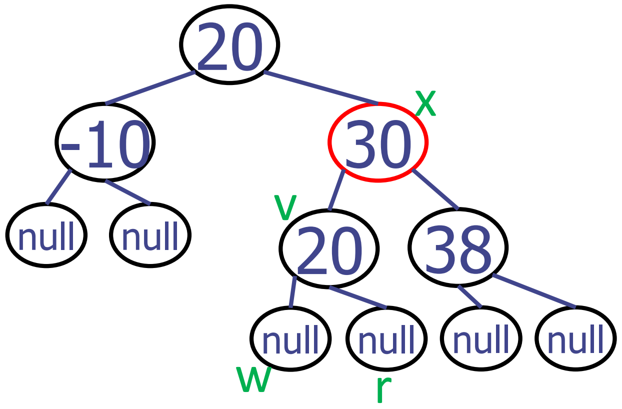
Else (***v*** and ***r*** were both black) we color ***r double black***, which is a violation of the internal property requiring a reorganization of the tree (Examples 3)

- Notations after removing v and w
  - y: sibling of r
  - z: child of y
- We now divide the cases, depending of the color of y and z

Recall: Example 3. Notations again!

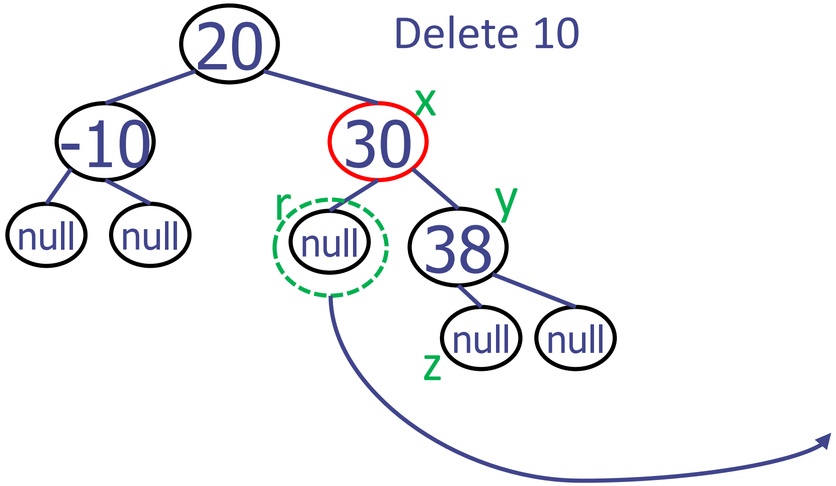
- What about deleting a node with a black child?





Delete 10

Copy inorder successor



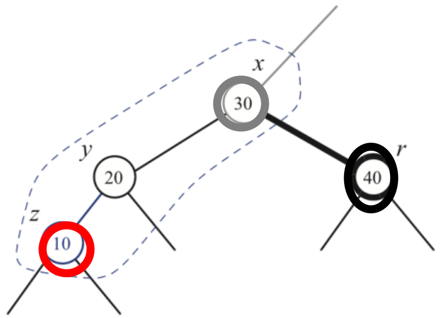
Delete 20.

Problem: A path of only 2 blacks

Regard this as “double black nodes”

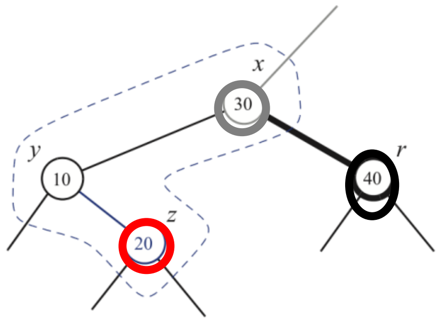
Handling Double Black Nodes: Case 1

- Case 1: The sibling y of r is black, and has a red child z
  - We perform a restructuring, and we are done



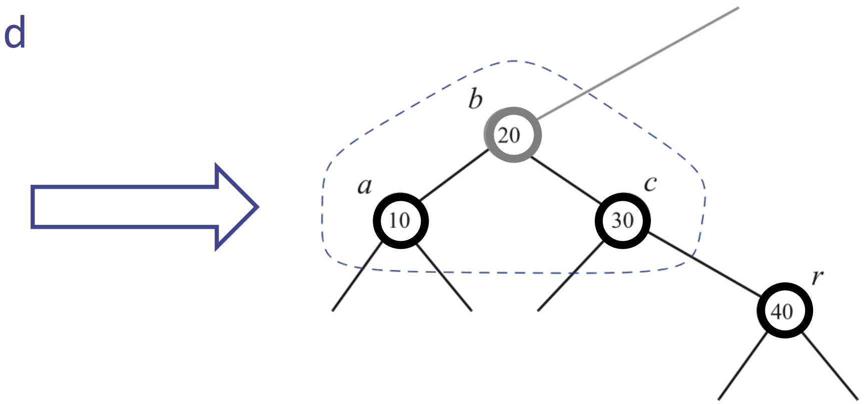
(a)

z is the left child



(b)

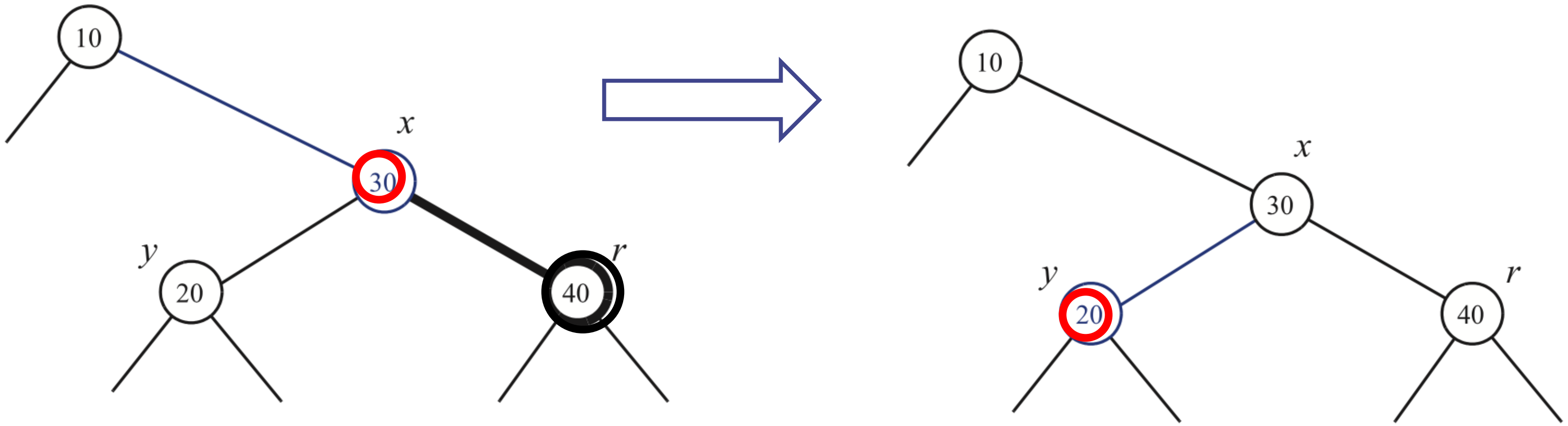
z is the right child



Double black node solved?

Handling Double Black Nodes: Case 2

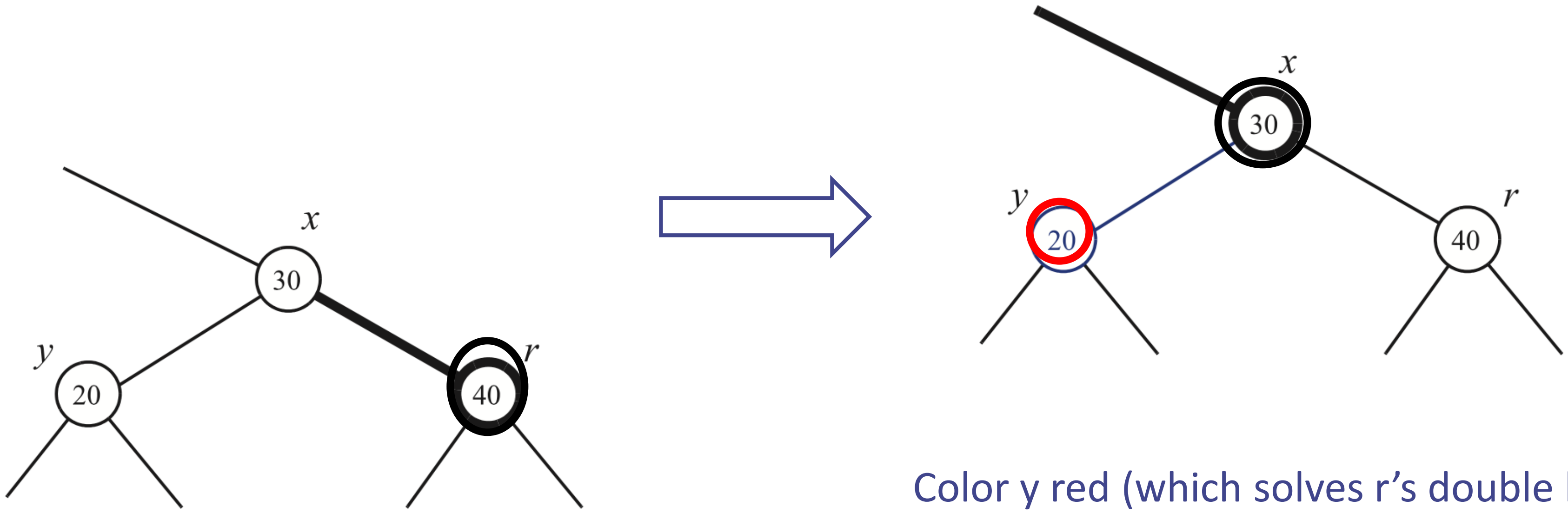
- Case 2: The sibling y of r is black, and y's both children are black
  - We perform a recoloring
  - Case 2-1: x (r's parent) is red



Color x black and color y red

Handling Double Black Nodes: Case 2

- Case 2: The sibling y of r is black, and y's both children are black
  - We perform a recoloring
  - Case 2-2: x (r's parent) is black

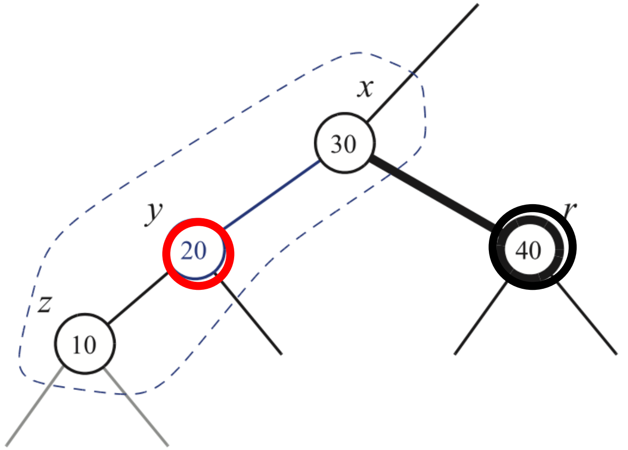


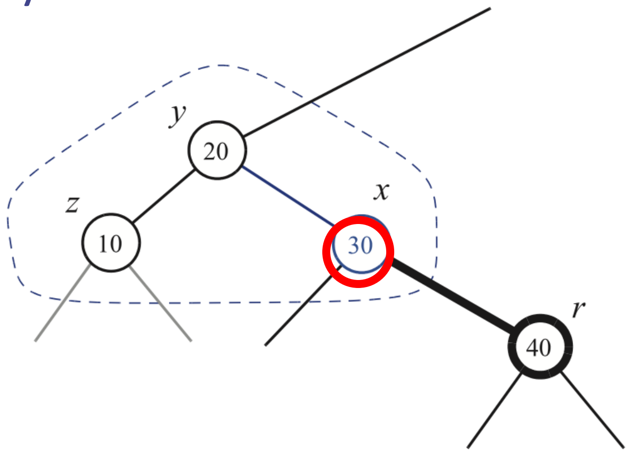
Color y red (which solves r's double black),
and make x "double black"
(propagates the double black up),
then reconsider the cases for x

Handling Double Black Nodes: Case 3

IUT-ECE

- Case 3: The sibling y of r is red
  - We perform adjustment
    - If y is the *right* child of x, then let z be the *right* child of y
    - If y is the *left* child of x, then let z be the *left* child of y
  - Case 3-1: z is the left child of v





Perform restructuring
Make y be the parent of x
Color y black and x red
(double black not yet solved)
→ The sibling of r is black (why?)
→ Case 1 or Case 2 applies

- Case 3-2: z is the right child of y → Similarly, we apply

Double Black Node Handling: Summary

- The algorithm for remedying a double black node r with sibling y considers three cases

  Case 1: y is black and has a red child
  - We perform a restructuring, and we are done

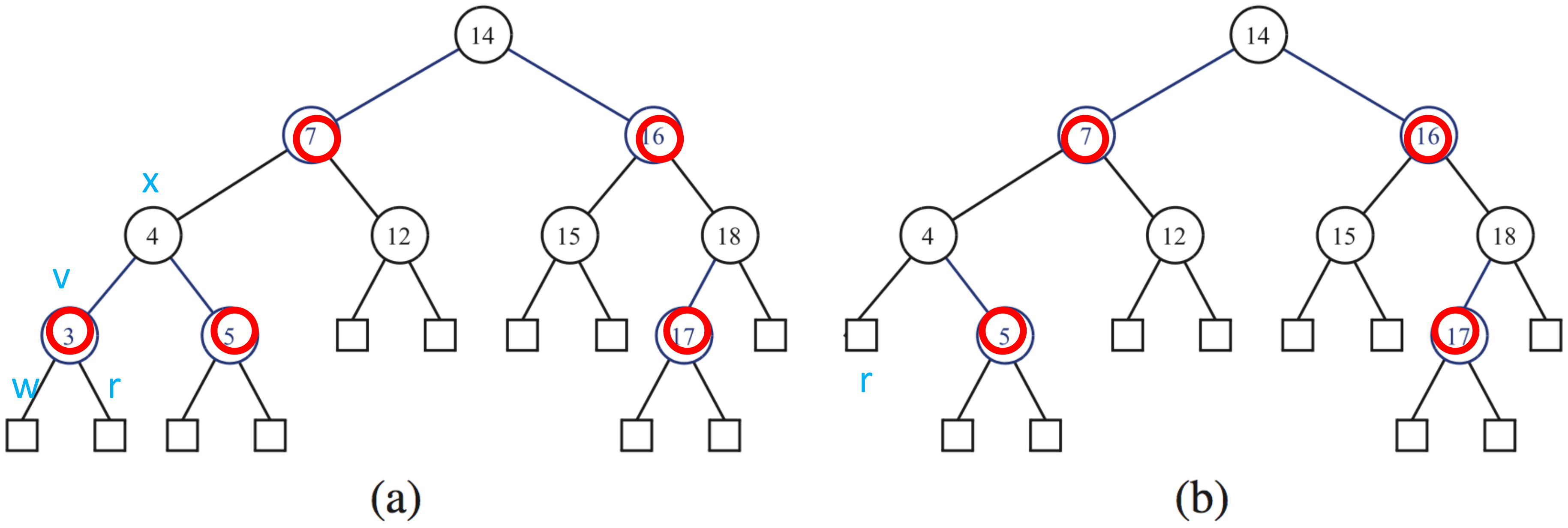
  Case 2: y is black and its children are both black
  - We perform a recoloring, which may propagate up the double black violation

  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

- Deletion in a red-black tree takes $O(\log n)$ time

Example: Remove 3

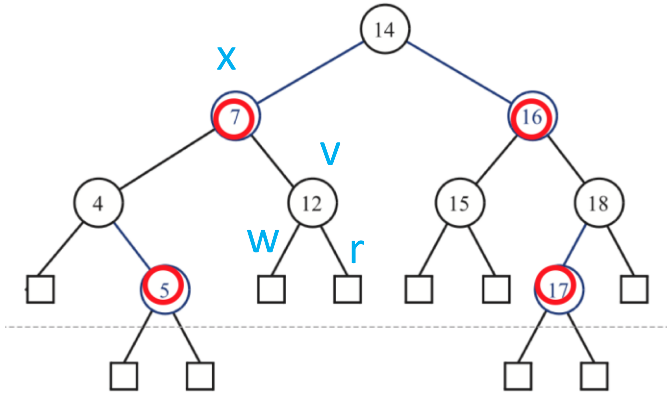
IUT-ECE



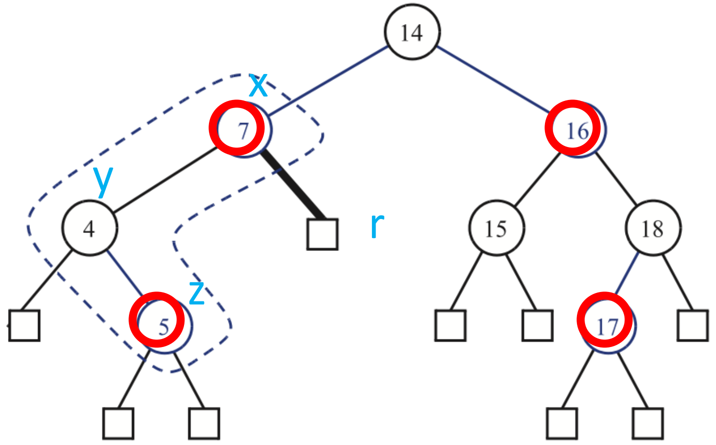
- v is red → Case 0 (either v or r is red)
- Remove v and w and color r black

Example: Remove 12

IUT-ECE

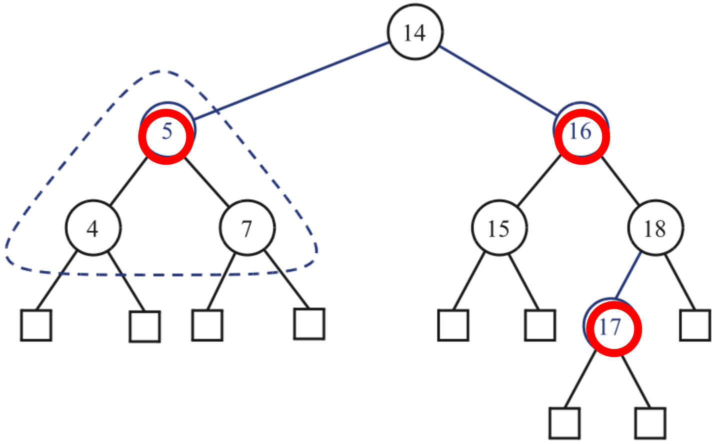


(b)



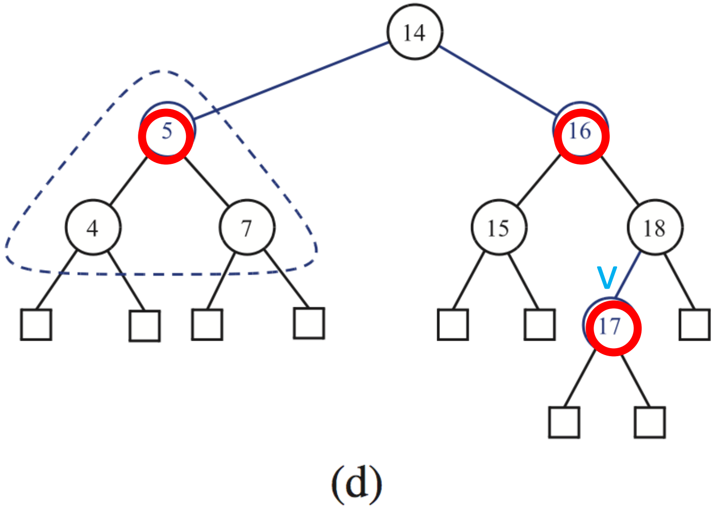
(c)

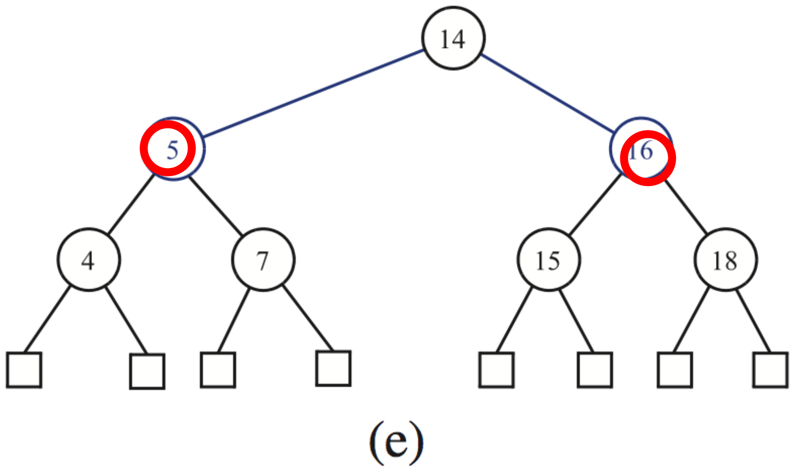
- None of v and r is red → Not Case 0
- y is black, which has red child
  - → Case 1, restructuring



(d)

Example: Remove 17

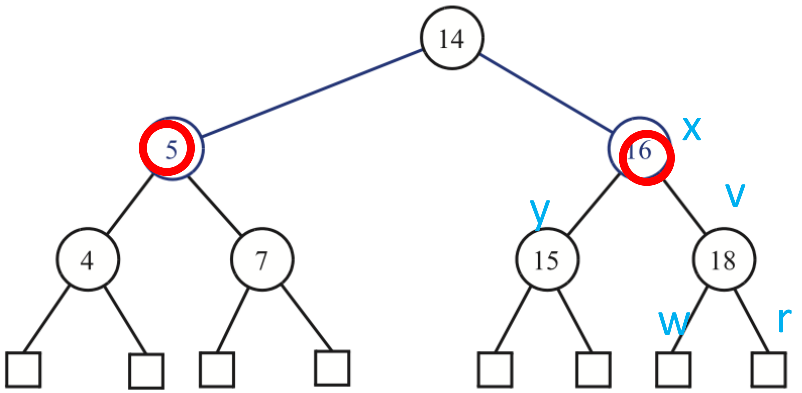




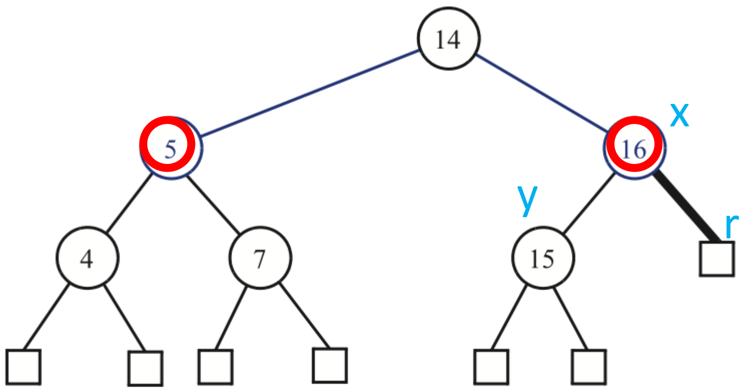
- v is red → Case 0

Example: Remove 18

IUT-ECE

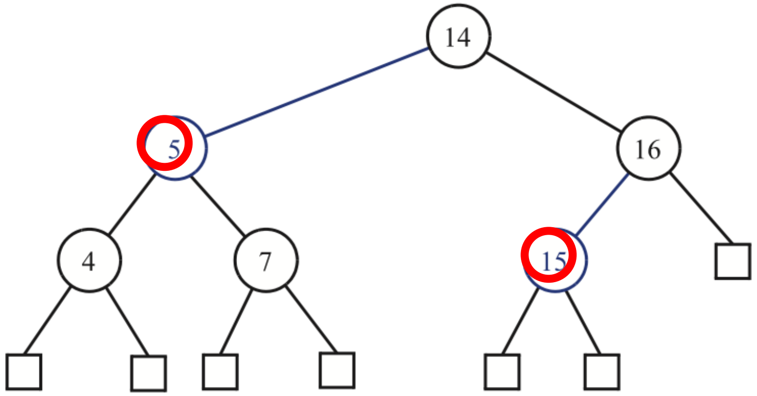


(e)



(f)

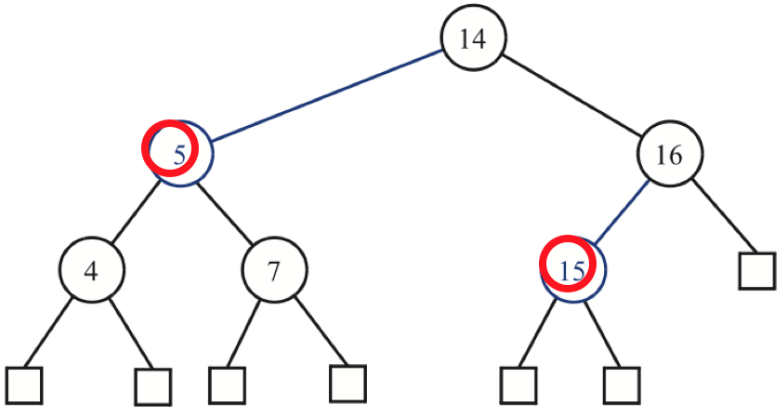
- None of v and r is red → Not Case 0
- y is black, having both black children →Case 2
  - x is red →Case 2-1, recoloring between x and y

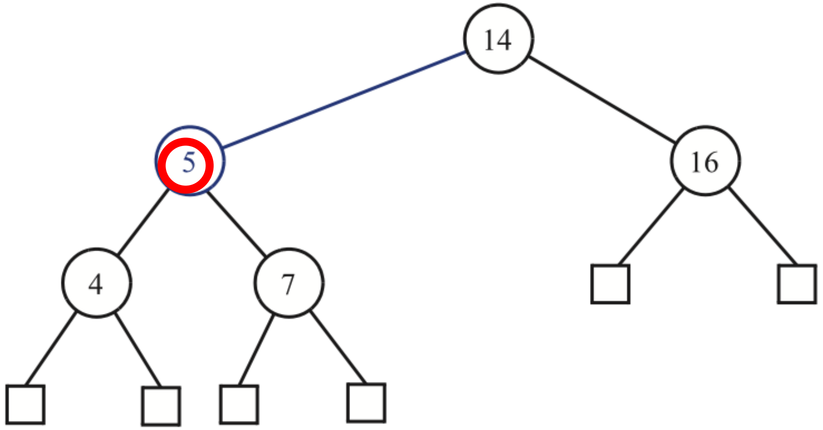


(g)

Example: Remove 15

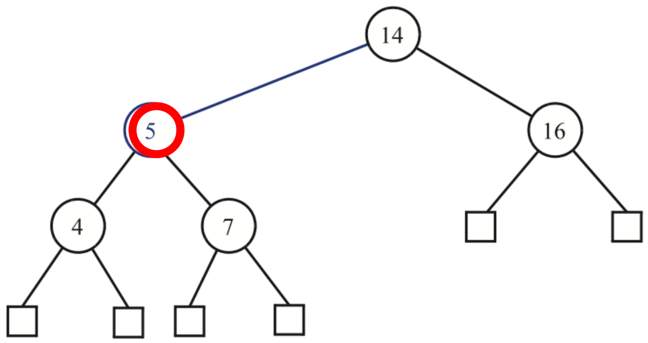
IUT-ECE



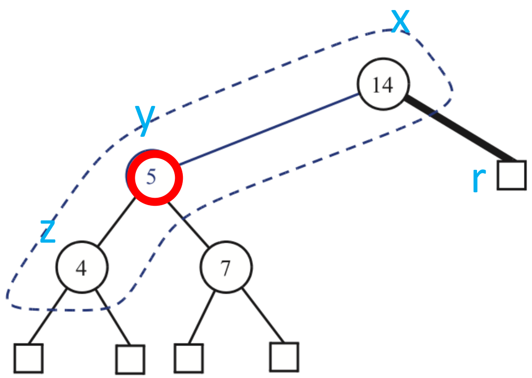


- Case 0 (now you know, right?)

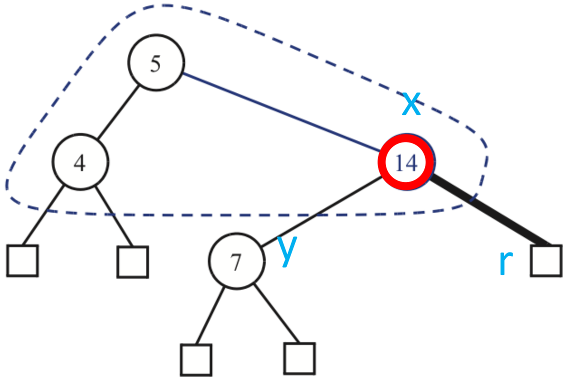
Example: Remove 16



(h)

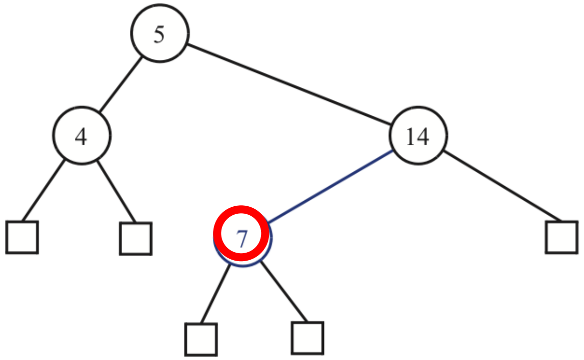


(i)



(j)

- y is red → Case 3
- y is the left child of x, thus z is node 4 (left child of y) → Case 3-1
- Adjustment → node 14 becomes double black → new y (sibling of x)
- y has both black children, and x is red
  - → Case 2-1, recoloring, then we're done



(k)