



Theory of algorithms (5th lecture)

Pál Pusztai
pusztai@sze.hu

Outline

- Binary search trees
 - Definition
 - Tree walks
 - Querying a binary search tree
 - Insertion and deletion
- Exercises



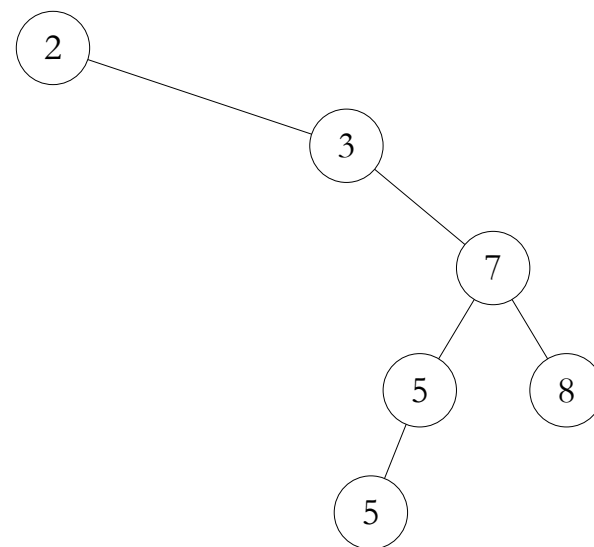
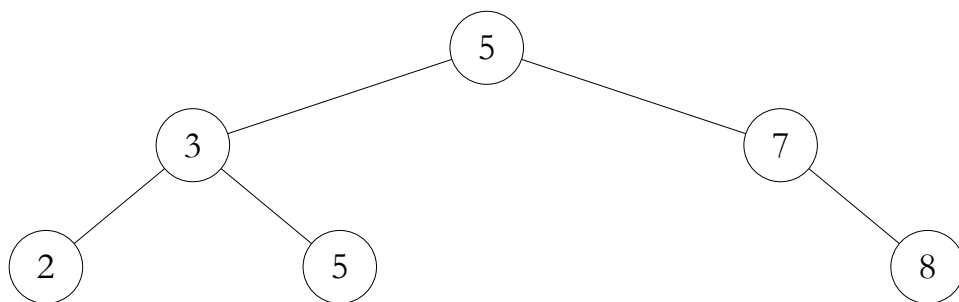
Binary search trees

The **binary trees** can be represented by a linked data structure in which each node is an object.

In addition to a *key* and satellite data, each node contains attributes *left*, *right*, and *p* that point to the nodes corresponding to its left child, its right child, and its parent, respectively.

The **binary-search-tree property**: Let x be a node in a binary search tree. If y is a node in the left subtree of x , then $y.key \leq x.key$. If y is a node in the right subtree of x , then $y.key \geq x.key$.

The **search tree** data structure supports many dynamic-set operations, including SEARCH, MINIMUM, MAXIMUM, PREDECESSOR, SUCCESSOR, INSERT, and DELETE. Thus, we can use a search tree both as a **dictionary** and as a **priority queue**.



Binary search trees

Binary search trees

INORDER-TREE-WALK(x)

```
1  if  $x \neq \text{NIL}$ 
2      INORDER-TREE-WALK( $x.\text{left}$ )
3      write  $x.\text{key}$ 
4      INORDER-TREE-WALK( $x.\text{right}$ )
```

Remarks:

- To print all the elements in a binary search tree T , we call INORDER-TREE-WALK($T.\text{root}$).
- The **inorder tree walk** prints the key of the root of a subtree between printing the values in its left subtree and printing those in its right subtree.
- A **preorder tree walk** prints the key of the root before the values in either subtree, and a **postorder tree walk** prints the root after the values in its subtrees.

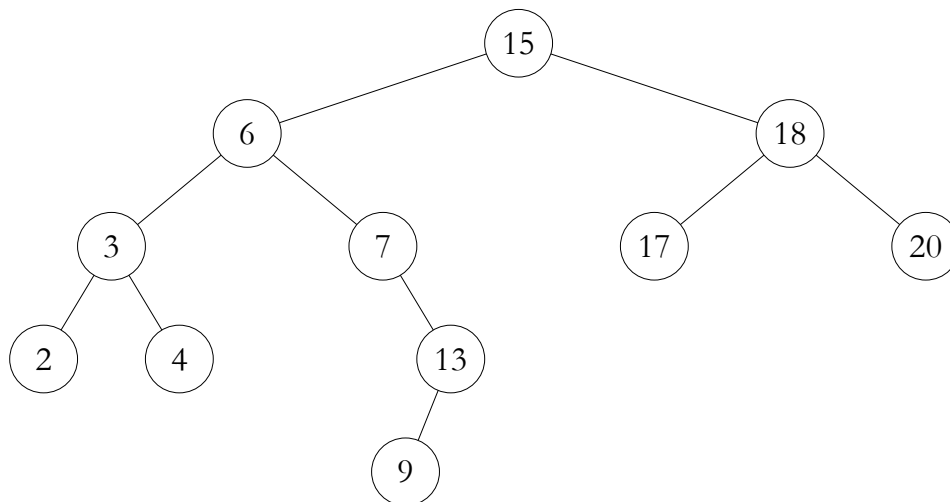
Theorem: If x is the root of an n -node subtree, then the call INORDER-TREE-WALK(x) takes $\Theta(n)$ time.

Efficiency: It takes $\Theta(n)$ time to walk an n -node binary search tree.



Exercises

- Draw binary search trees of heights 2, 3, 4, 5, and 6 with the below given keys.
 - {1, 4, 5, 10, 16, 17, 21}
- What is the order of the printed keys of the below given tree by an algorithm with preorder tree walk, and what is it with a postorder tree walk?



Binary search trees

TREE-SEARCH(x, k)

```
1  if  $x == \text{NIL}$  or  $k == x.\text{key}$ 
2      return  $x$ 
3  if  $k < x.\text{key}$ 
4      return TREE-SEARCH( $x.\text{left}, k$ )
5  else
6      return TREE-SEARCH( $x.\text{right}, k$ )
```

ITERATIVE-TREE-SEARCH(x, k)

```
1  while  $x \neq \text{NIL}$  and  $k \neq x.\text{key}$ 
2      if  $k < x.\text{key}$ 
3           $x = x.\text{left}$ 
4      else
5           $x = x.\text{right}$ 
6  return  $x$ 
```

Efficiency: The running time is $O(h)$ on a binary search tree of height h .

Exercises

- Suppose that we have whole numbers between 1 and 1000 in a binary search tree, and we want to search for the number 363. Which of the following sequences could *not* be the sequence of nodes examined?
 - 2, 252, 401, 398, 330, 344, 397, 363
 - 924, 220, 911, 244, 898, 258, 362, 363
 - 925, 202, 911, 240, 912, 245, 363



Binary search trees

TREE-MINIMUM(x)

```
1  while  $x.left \neq \text{NIL}$ 
2       $x = x.left$ 
3  return  $x$ 
```

TREE-MAXIMUM(x)

```
1  while  $x.right \neq \text{NIL}$ 
2       $x = x.right$ 
3  return  $x$ 
```

TREE-SUCCESSOR(x)

```
1  if  $x.right \neq \text{NIL}$ 
2      return TREE-MINIMUM( $x.right$ )
3   $y = x.p$ 
4  while  $y \neq \text{NIL}$  and  $x == y.right$ 
5       $x = y$ 
6       $y = y.p$ 
7  return  $y$ 
```

Efficiency: The running time of each operation is $O(h)$ on a binary search tree of height h .



Binary search trees

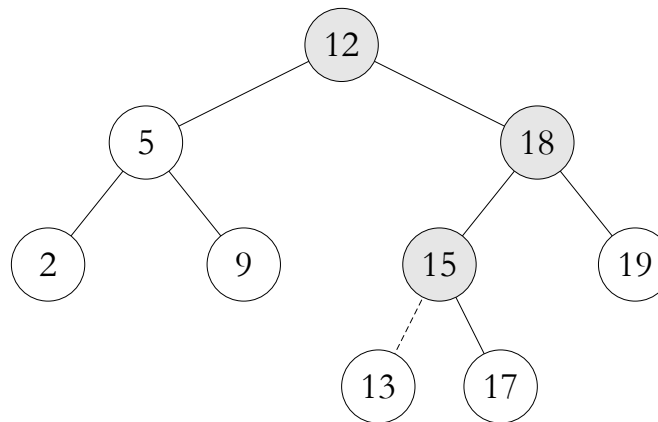
TREE-INSERT(T, z)

```
1   $y = \text{NIL}$ 
2   $x = T.\text{root}$ 
3  while  $x \neq \text{NIL}$ 
4       $y = x$ 
5      if  $z.\text{key} < x.\text{key}$ 
6           $x = x.\text{left}$ 
7      else
8           $x = x.\text{right}$ 
9   $z.p = y$ 
10 if  $y == \text{NIL}$ 
11      $T.\text{root} = z$            // tree  $T$  was empty
12 else
13     if  $z.\text{key} < y.\text{key}$ 
14          $y.\text{left} = z$ 
15     else
16          $y.\text{right} = z$ 
```

Efficiency: The running time is $O(h)$ on a binary search tree of height h .



Binary search trees



Inserting an item into a binary search tree.

Exercises

- Illustrate the operation of TREE-INSERT with to below given keys. We insert these keys into an initially empty binary search tree.
 - 2, 4, 5, 7, 1, 6, 3
- Give an order of the previous keys that results the tree with minimum height.
- What is the asymptotical "behavior" of TREE-INSERT with inserting the same keys into a tree by n times?



Binary search trees

TRANSPLANT(T, u, v)

```
1  if  $u.p == \text{NIL}$ 
2       $T.root = v$ 
3  else
4      if  $u == u.p.left$ 
5           $u.p.left = v$ 
6      else
7           $u.p.right = v$ 
8  if  $v \neq \text{NIL}$ 
9       $v.p = u.p$ 
```

This auxiliary procedure replaces the subtree rooted at node u with the subtree rooted at node v . Node u 's parent becomes node v 's parent, and u 's parent ends up having v as its appropriate child.

Efficiency: The running time is $O(1)$.



Binary search trees

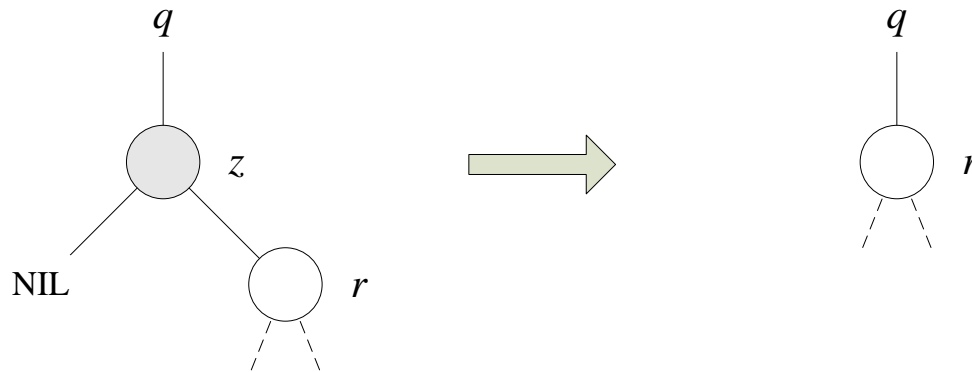
TREE-DELETE(T, z)

```
1  if  $z.left == \text{NIL}$ 
2      TRANSPLANT( $T, z, z.right$ )
3  else if  $z.right == \text{NIL}$ 
4      TRANSPLANT( $T, z, z.left$ )
5  else
6       $y = \text{TREE-MINIMUM}(z.right)$ 
7      if  $y.p \neq z$ 
8          TRANSPLANT( $T, y, y.right$ )
9           $y.right = z.right$ 
10          $y.right.p = y$ 
11     TRANSPLANT( $T, z, y$ )
12      $y.left = z.left$ 
13      $y.left.p = y$ 
```

Efficiency: The running time is $O(h)$ on a binary search tree of height h .

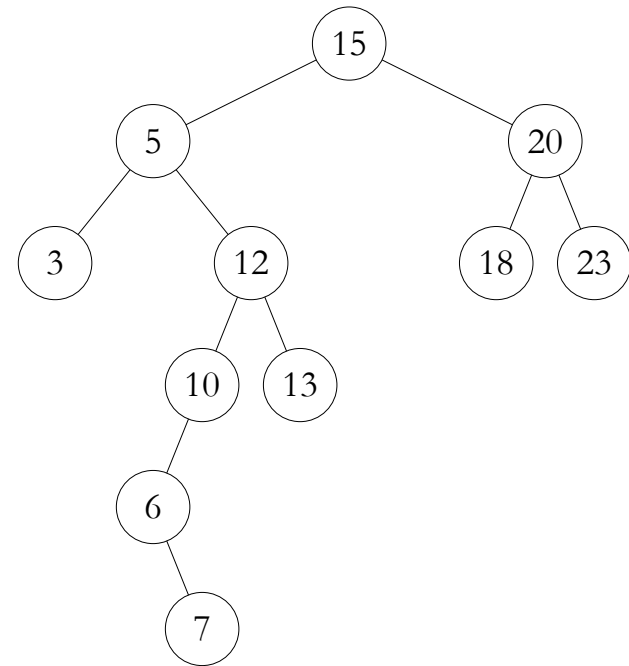
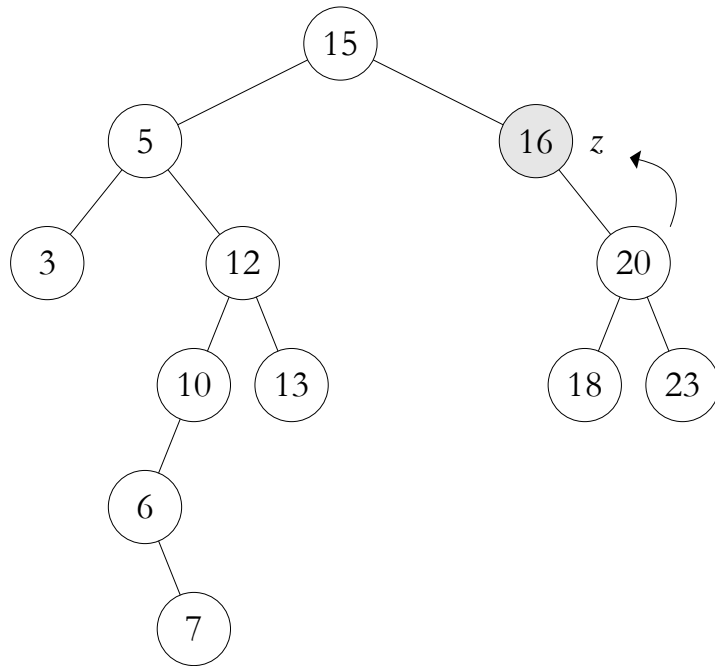


Binary search trees



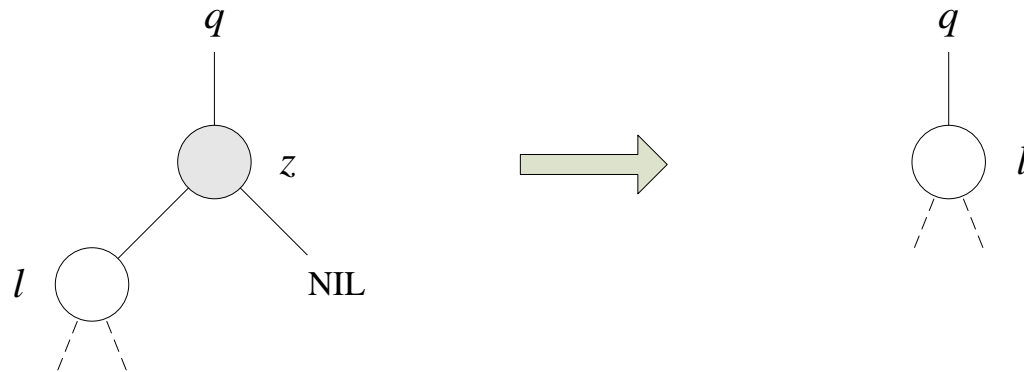
Deleting a node z from a binary search tree (case a)

Binary search trees



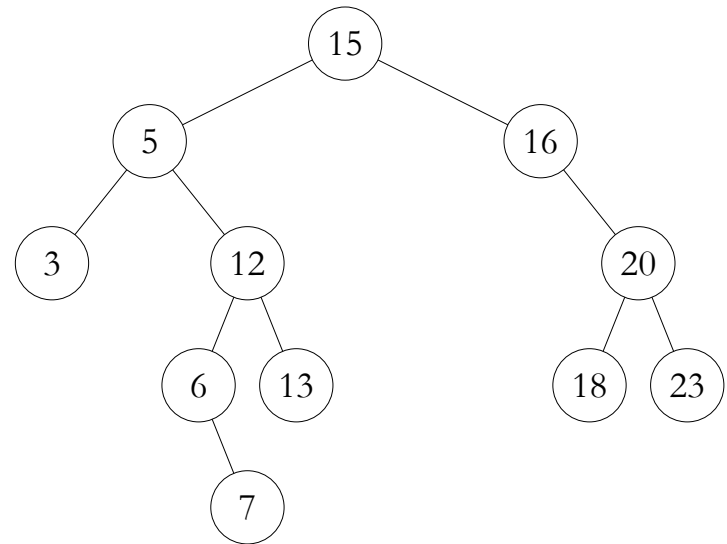
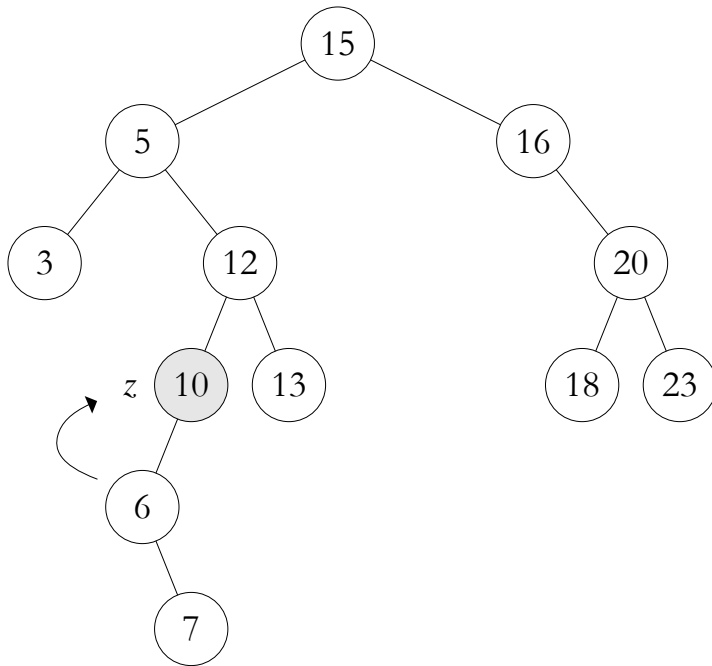
Deleting a node z from a binary search tree (case a)

Binary search trees



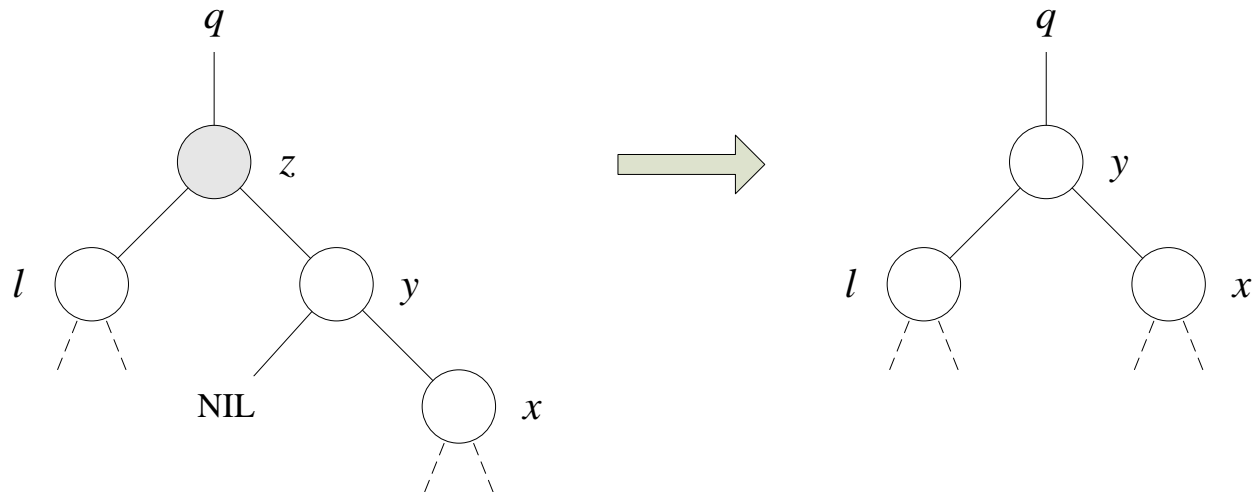
Deleting a node z from a binary search tree (case b)

Binary search trees



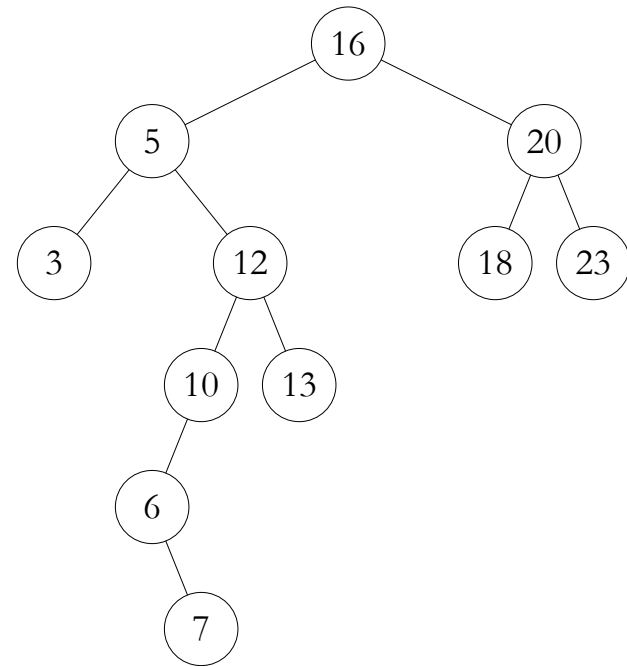
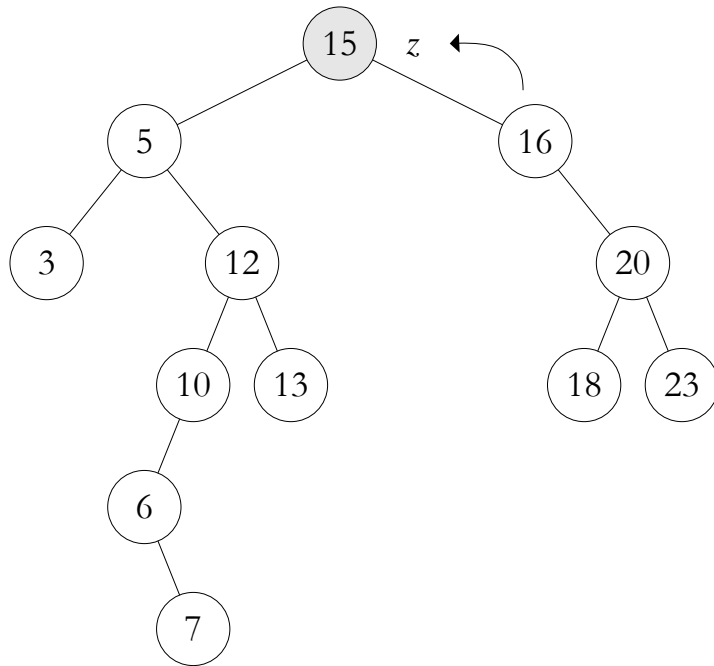
Deleting a node z from a binary search tree (case b)

Binary search trees



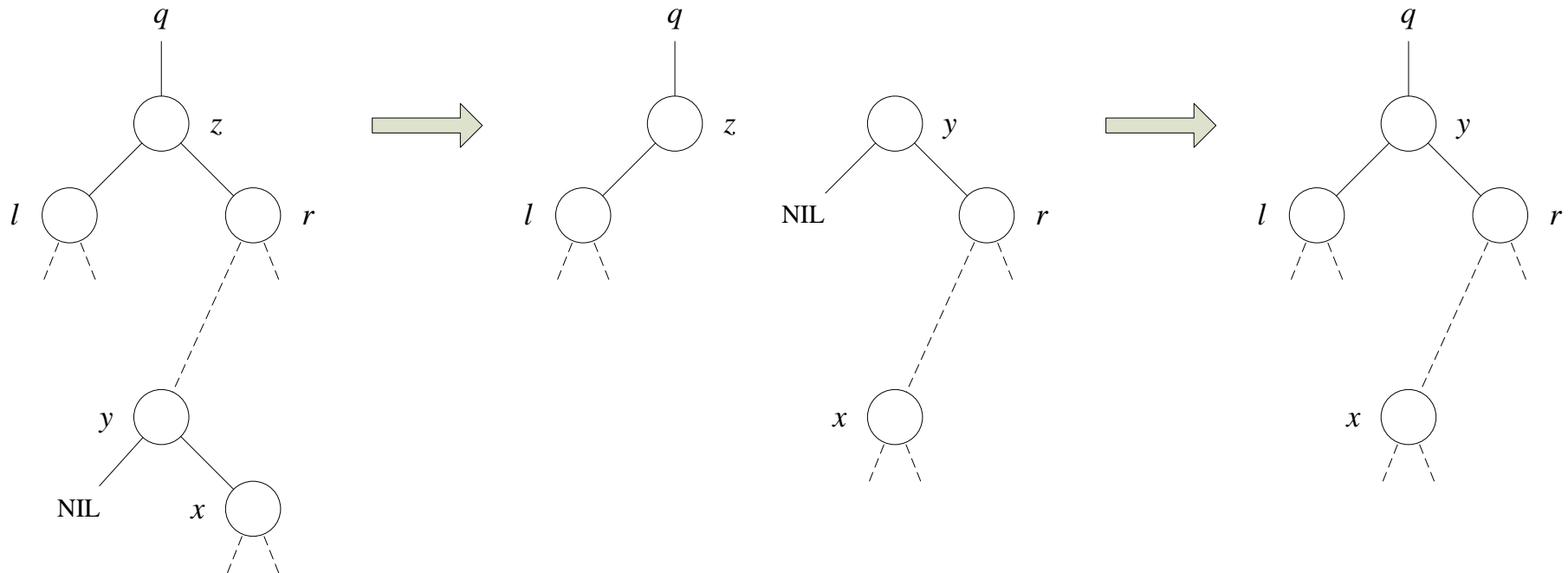
Deleting a node z from a binary search tree (case c)

Binary search trees



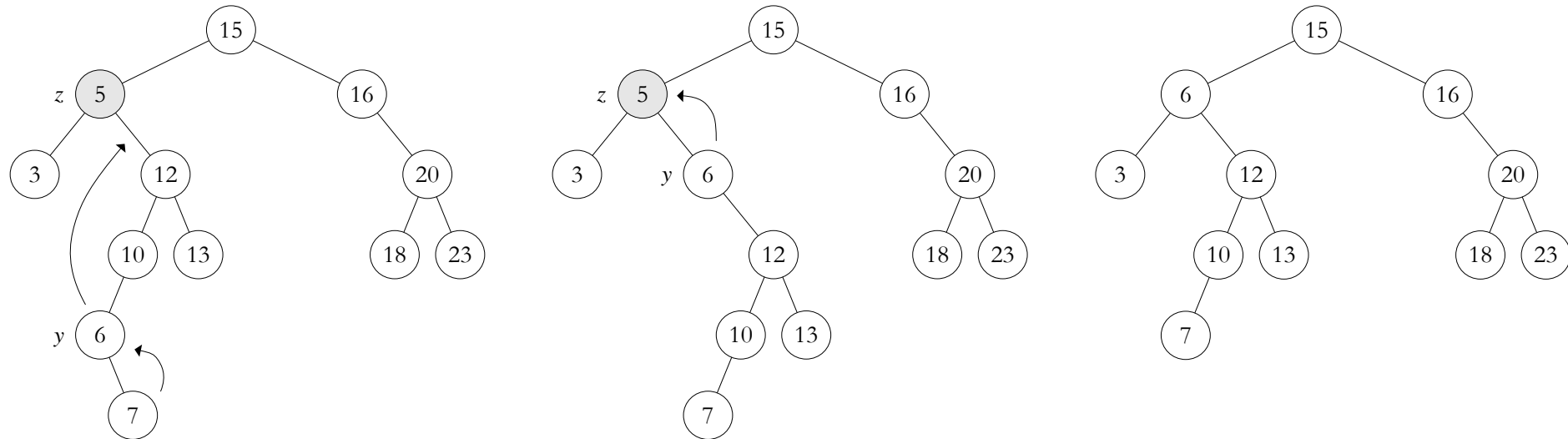
Deleting a node z from a binary search tree (case c)

Binary search trees



Deleting a node z from a binary search tree (case d)

Binary search trees



Deleting a node z from a binary search tree (case d)

Binary search trees

When both insertion and deletion are used to create a binary search tree little is known about the average height of it, but create it by insertion alone, the analysis becomes more tractable.

Let us given n distinct keys and we built a binary search tree with inserting the keys in random order into an initially empty tree.

If each of the $n!$ permutations of the input keys is equally likely then we call the tree **randomly built binary search tree**.

Theorem: The expected height of a randomly built binary search tree on n distinct keys is $O(\lg n)$.



Exercises

- What binary search tree is built with inserting keys 4, 10, 7, 9, 15, 2, 5, 8, 6, 1, 3 into an initially empty tree? Give the trees after deleting keys 1, 5, 7, and 4.
- Is it true or not that if a node of a binary search tree has two children then its successor has no left child and its predecessor has no right child?
- We can sort a given set of n numbers by first building a binary search tree containing these numbers (using TREE-INSERT repeatedly to insert the numbers one by one) and then printing the numbers by an inorder tree walk. What are the worst-case and best-case running times for this sorting algorithm?
- Is the operation of deletion “commutative” in the sense that deleting x and then y from a binary search tree leaves the same tree as deleting y and then x ? Argue why it is or give a counterexample.

