Trees and BSTs

BSTs: Extension 02

Paolo Camurati and Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

Pointers & Counters

- New functionalities can be added to BSTs by inserting new information to each node
- This information usually consists in adding for each node
  - A pointer to the parent
  - The number of nodes of the tree rooted at the current node
- This fields have to be
  - Inserted in the original data structure
  - Defined and updated (when necessary) by **all** BST manipulation functions (even the ones already analyzed)

Binary Search Trees

item → key
is an integer
(in this section)

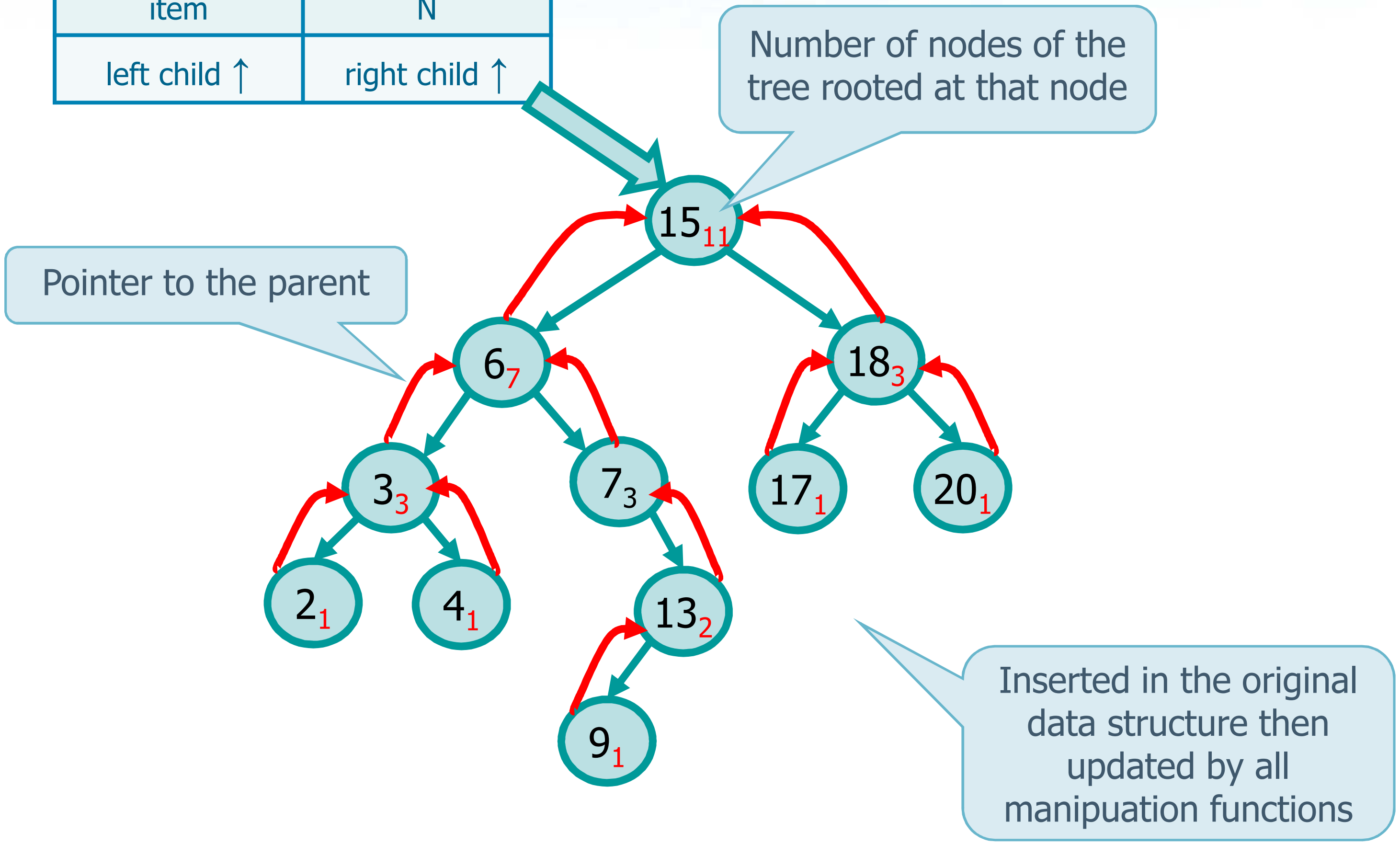
pointer to parent	
item	N
pointer to left child	pointer to right child

```
typedef struct node *link;
struct node {
  link p;
  Item item;
  int N;
  link l;
  link r;
};
```

ADT: We use functions
to compare keys, etc.

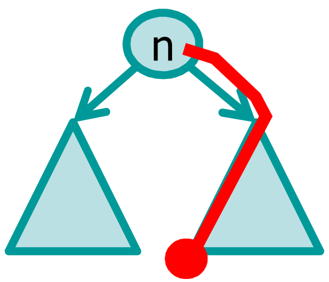
Example

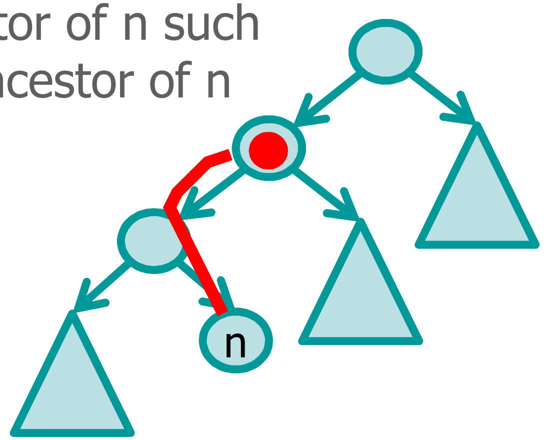
pointer to parent	
item	N
left child ↑	right child ↑



Successor of a node

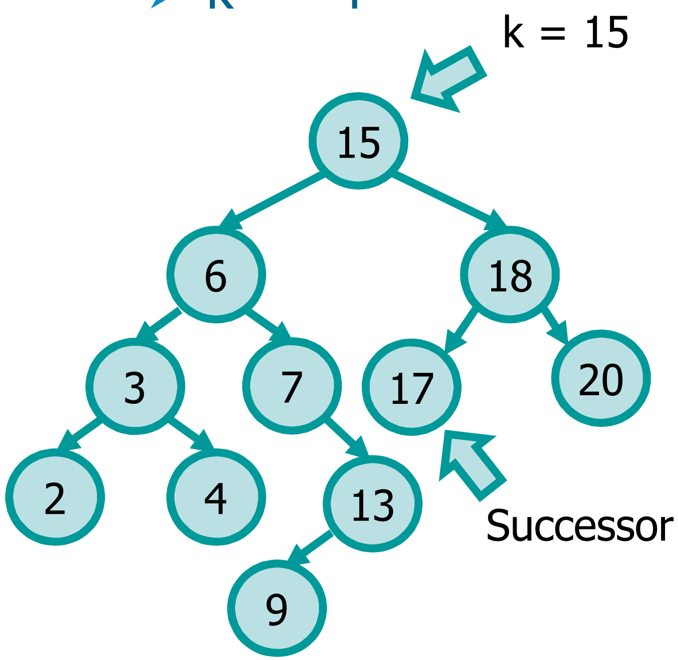
- Given a node n, find the node with the smallest key larger than the node key
- There are two cases
  - Node n has the right child
    - succ(key(n)) is the minumum value in Right(n)
  - Node n does not have the right child
    - succ(key(**n**)) is the first ancestor of n such that the left child is also an ancestor of n

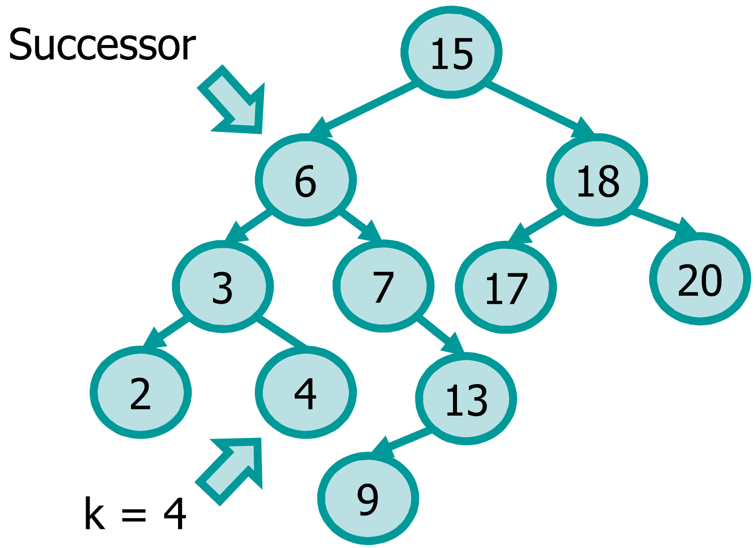




Examples

- Given the following tree which node is the successor of node with key
  - k = 15
  - k = 4





Implementation

```
link search_succ_r (link root, Item item, link z) {
  link p;
  if (root == z)
    return z;
  if (item_less (item, root->item))
    return search_succ_r (root->l, item, z);
  if (item_less (root->item, item))
    return search_succ_r (root->r, item, z);
  if (root->r != z) {
    return min_r (root->r, z);
  } else {
    p = root->p;
    while (p != z && root == p->r) {
      root = p; p = p->p;
    }
    return p;
  }
}
```

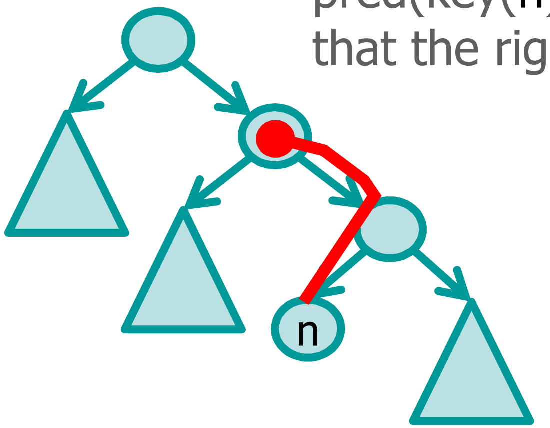
Root node

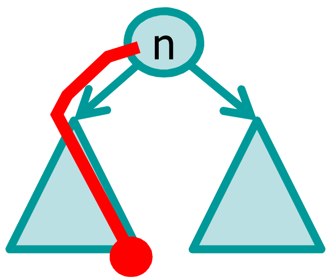
Search node

Search successor

Predecessor of a node

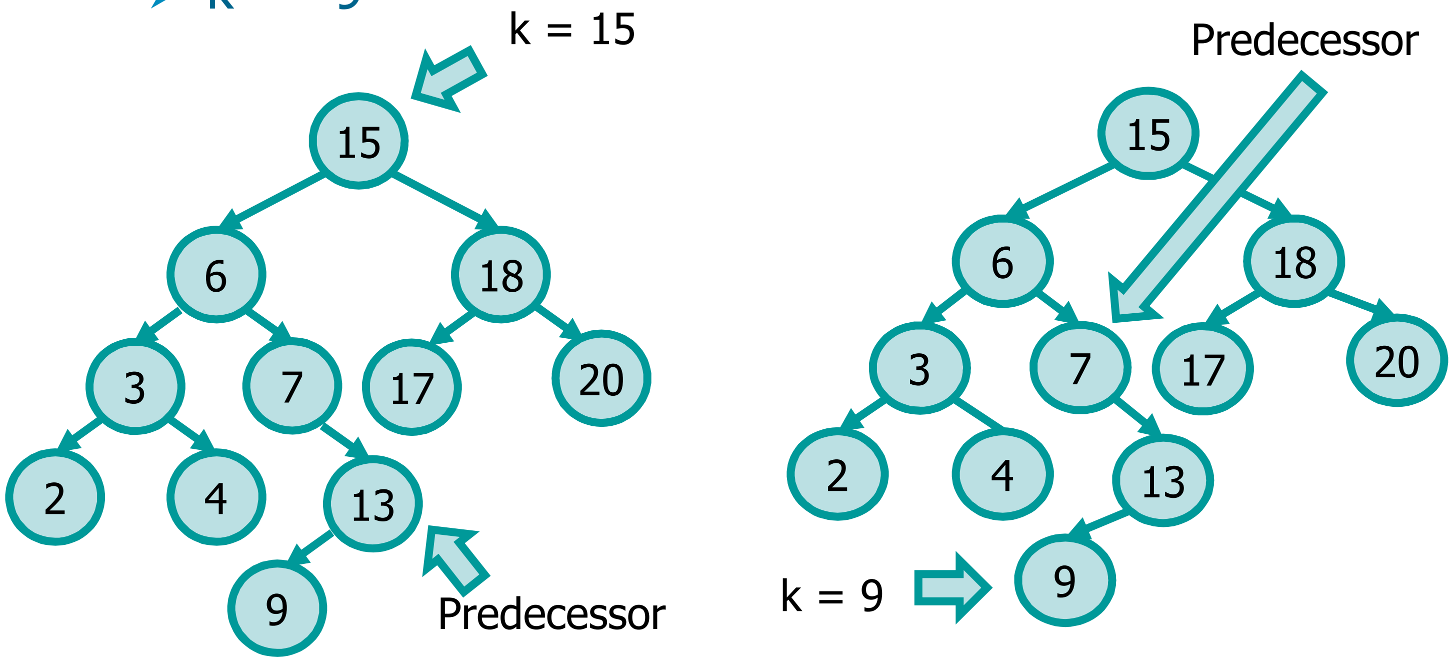
- Given a node n, find the node with the largest key smaller than the node key
- There are two cases
  - Node n has the left child
    - pred(key(n)) is the maximum value in Left(n)
  - Node n does not have the left child
    - pred(key(n)) is the first ancestor of n such that the right child is also an ancestor of n





Examples

- Given the following tree which node is the predecessor of node with key
  - k = 15
  - k = 9



Implementation

Root node

Search node

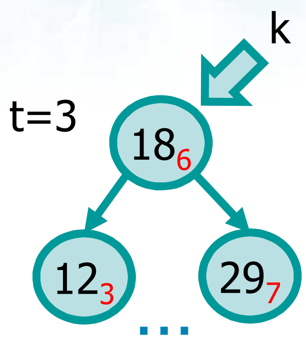
Search successor

```
link search_pred_r (link root, Item item, link z) {
  link p;
  if (root == z) return z;
  if (item_less (item, root->item))
    return search_pred_r (root->l, item, z);
  if (item_less (root->item, item))
    return search_pred_r (root->r, item, z);
  if (root->r != z) {
    return max_r (root->l, z);
  } else {
    p = root->p;
    while (p != z && root == p->l) {
      root = p; p = p->p;
    }
    return p;
  }
}
```

Select

- Select the item with the k-th smallest key
  - We use a zero-based indexing notation
  - For example, selecting the key k=0 means to select the item with the smallest key
- Given the node root
  - We define **t** the number of nodes of the **left** sub-tree (reported in the left sub-tree root)

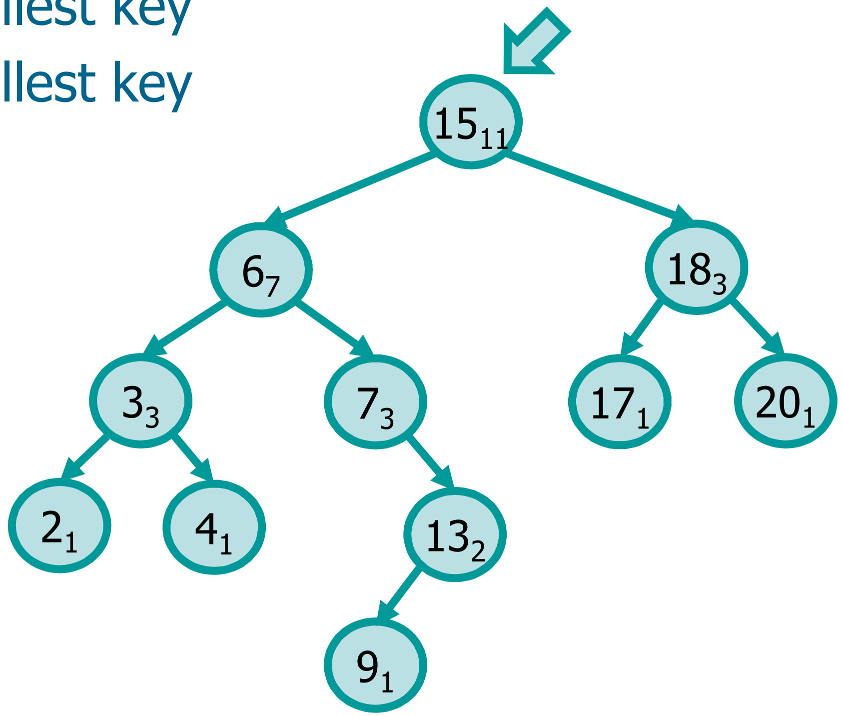
Select



- If
  - k = t
    - The root stores the k-th smallest key
    - Return the root pointer
  - k < t
    - The left sub-tree includes "enough" nodes
    - Recur into the left sub-tree to look-for the smallest k-th key
  - k > t
    - The left sub-tree does not include "enough" nodes
    - Recur on the right sub-tree
    - Set **k** to **(k-t-1)**, and look for the (k-t-1)-th smallest key

Examples

- Given the folllowing BST select the key with
  - k = 3 → 4-th smallest key
  - k = 8 → 9-th smallest key
  - k = 6 → 7-th smallest key



Implementation

Root node

```
link select_r (link root, int k, link z) {
  int t;

  if (root == z)
    return z;

  t = (root->l == z) ? 0 : root->l->N;

  if (k < t)
    return select_r (root->l, k, z);
  if (k > t)
    return select_r (root->r, k-t-1, z);

  return root;
}
```

Partition

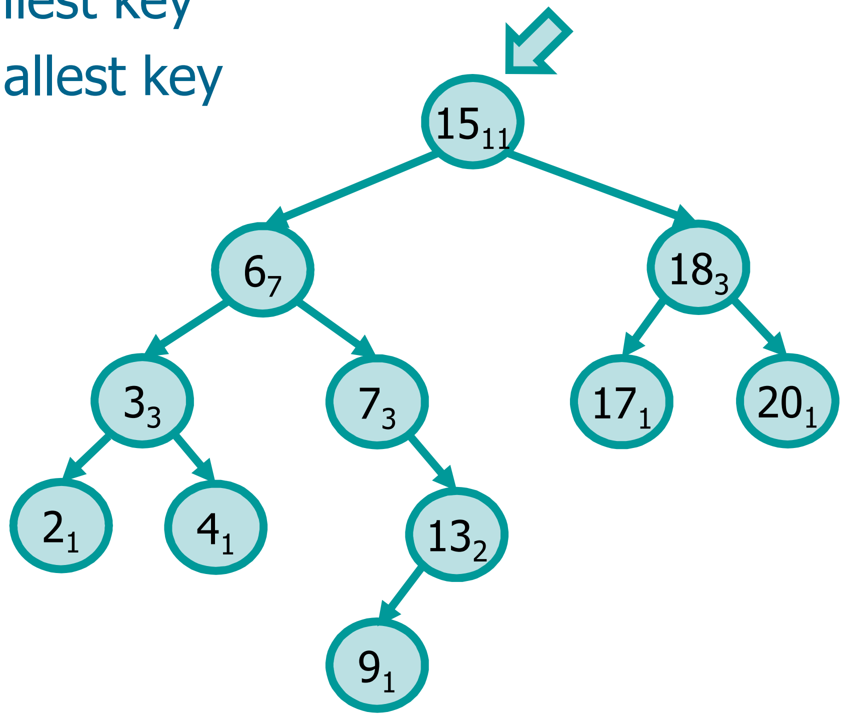
- Restructuring the tree, forcing the smallest k-th key into the root
- Consider the sub-tree root node
  - $k = t$
    - Return and rotate
  - $k < t$
    - Recur on the left sub-tree, partition with respect to the smallest k-th key, at the end right-rotation
  - $k > t$
    - Recur on the right sub-tree, partition with respect to the smallest (k-t-1)-th key, at the end left rotation

Partition

- Partitioning is often performed around the median key

Examples

- Given the folllowing BST partition it with respect to the key with
  - k = 4 → 5-th smallest key
  - k = 9 → 10-th smallest key



Implementation

Root node

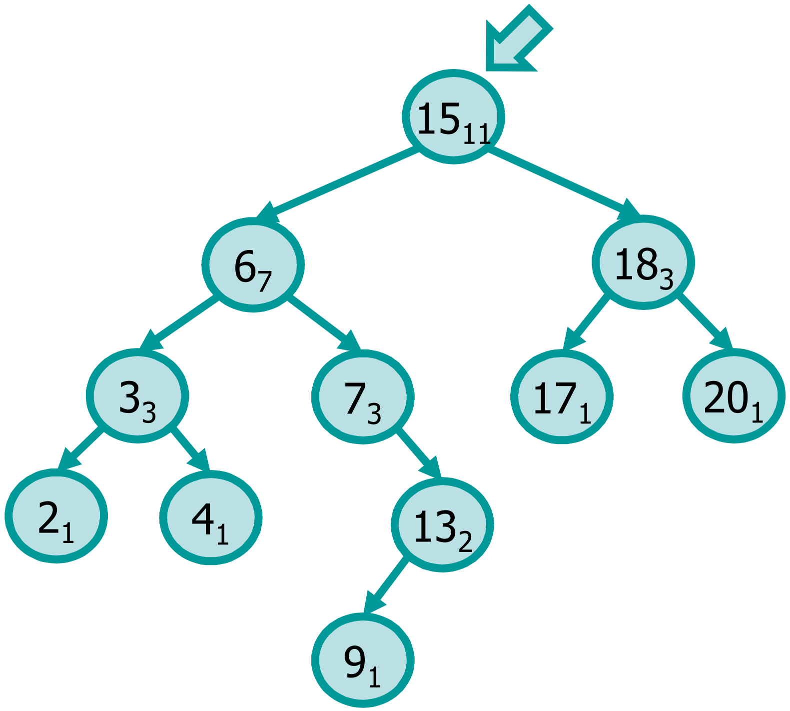
```
link part_r (link root, int k, link z) {
  int t;

  if (root == z)
    return z;

  t = (root->l == z) ? 0 : root->l->N;
  if (k < t) {
    root->l = part_r (root->l, k);
    root = rotR (root);
  }
  if (k > t) {
    root->r = part_r (root->r, k-t-1);
    root = rotL (root);
  }

  return root;
}
```

Delete a node: Version 2

- To delete from a BST a node with an item with a given key k, it is possible to use the partition function
  - If NULL or sentinel is reached
    - He key is not in the tree, just return
  - If the node with the item belongs to one sub-tree
    - Recursively delete such a sub-tree
  - If it is the root
    - Delete the node
    - The new root is the succ or pred of the deleted item
    - Rotate one of them up to the root
    - Combine the two sub-trees into the new root

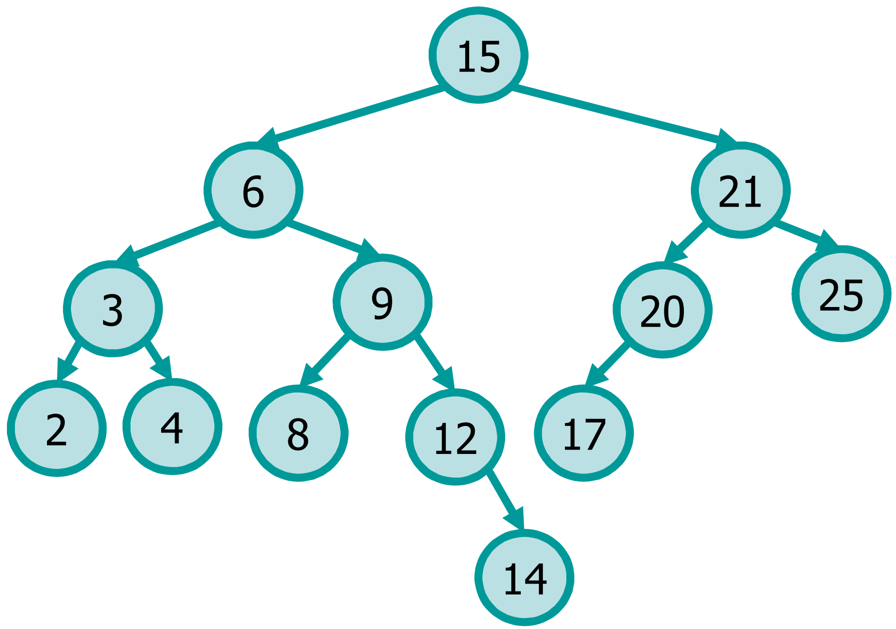
Examples

- Given the following BST delete nodes with key 7 ad 18



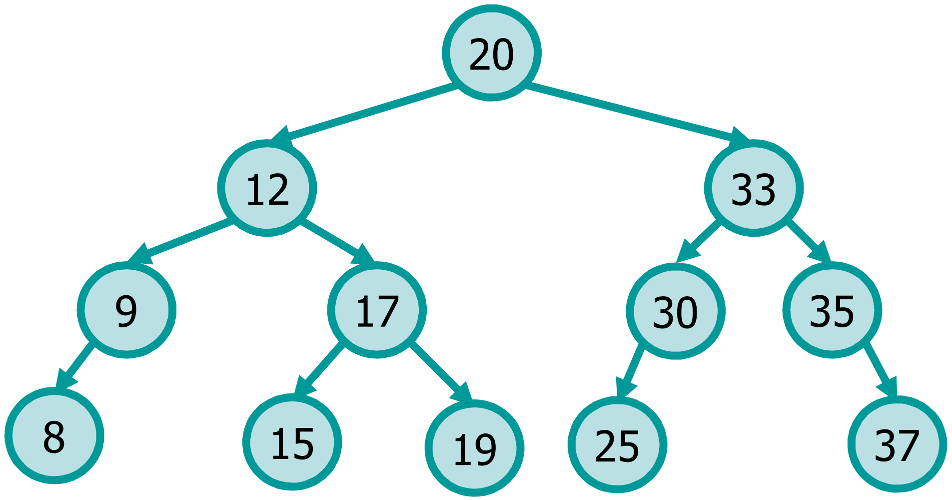
Exercise

- Given the following BST select the key with
  - k = 5 → 6-th smallest key
  - k = 9 → 10-th smallest key



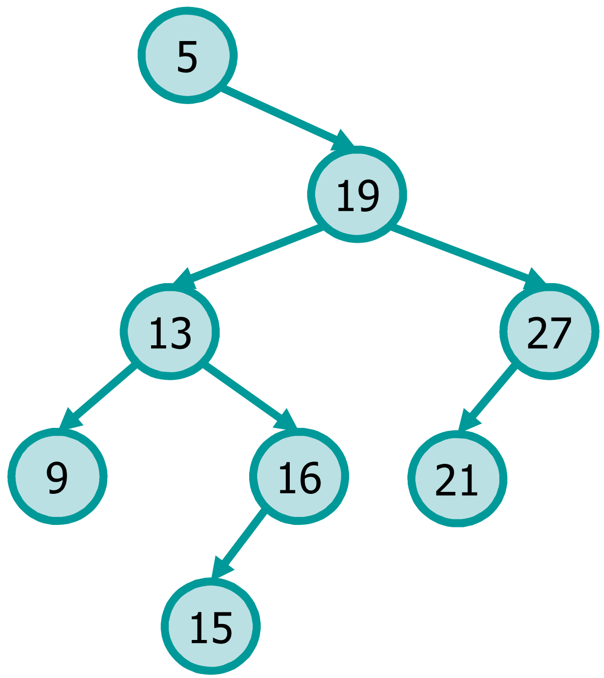
Exercise

- Given the following BST select the key with
  - k = 5 → 6-th smallest key
  - k = 10 → 11-th smallest key



Exercise

- Given the folllowing BST partition it with respect to the key with
  - k = 4 → 5-th smallest



Exercise

- Given the following BST partition it with respect to the key with
  - k = 6 → 7-th smallest