Trees and BSTs

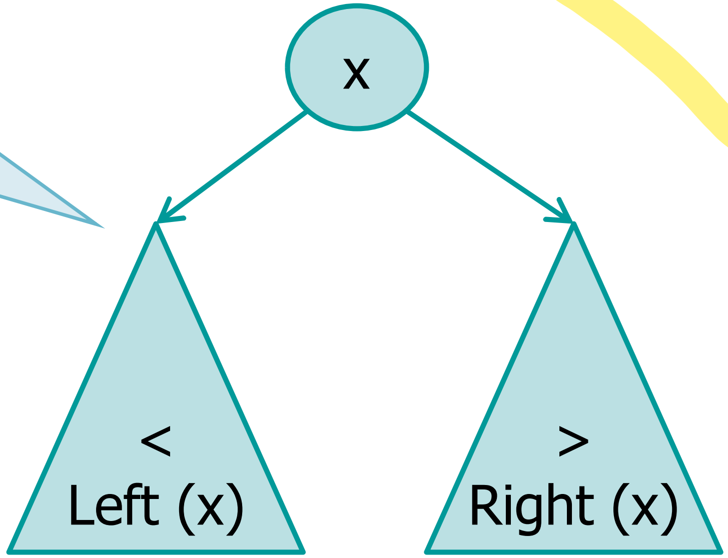
BSTs: Binary Search Trees

Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

Binary Search Trees (BSTs)

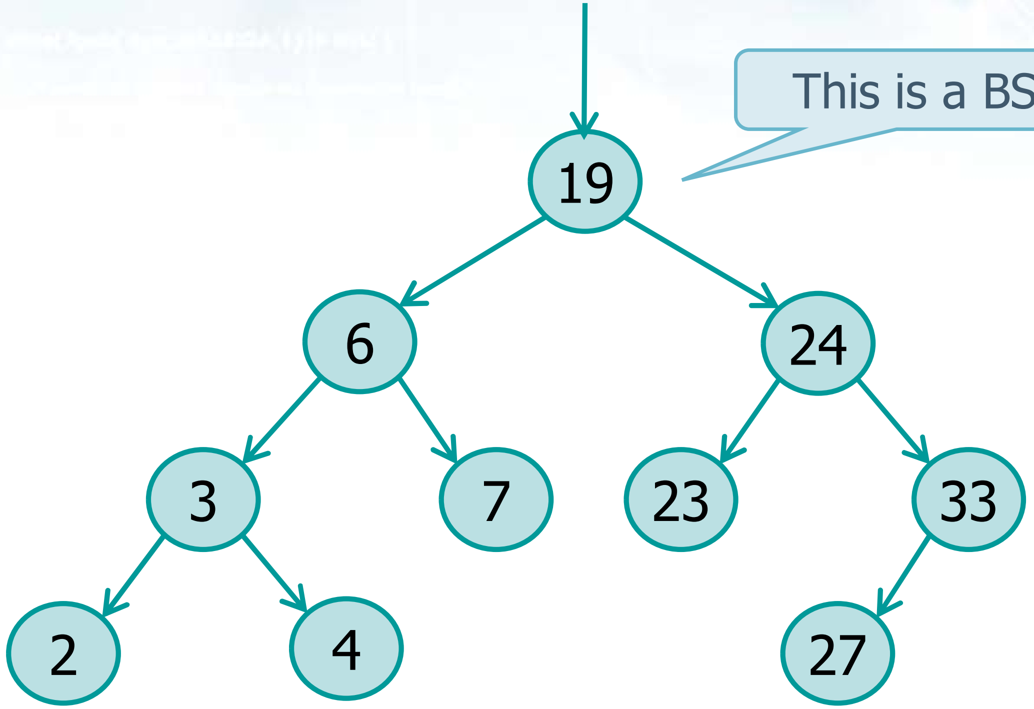
- A BST is a binary tree that forces a specific order among the key of the nodes
  - ∀ node x
    - ∀ node $y \in Left(x)$, $key[y] < key[x]$
    - ∀ node $y \in Right(x)$, $key[y] > key[x]$

Distinct keys (no need to know where to put the same key x)

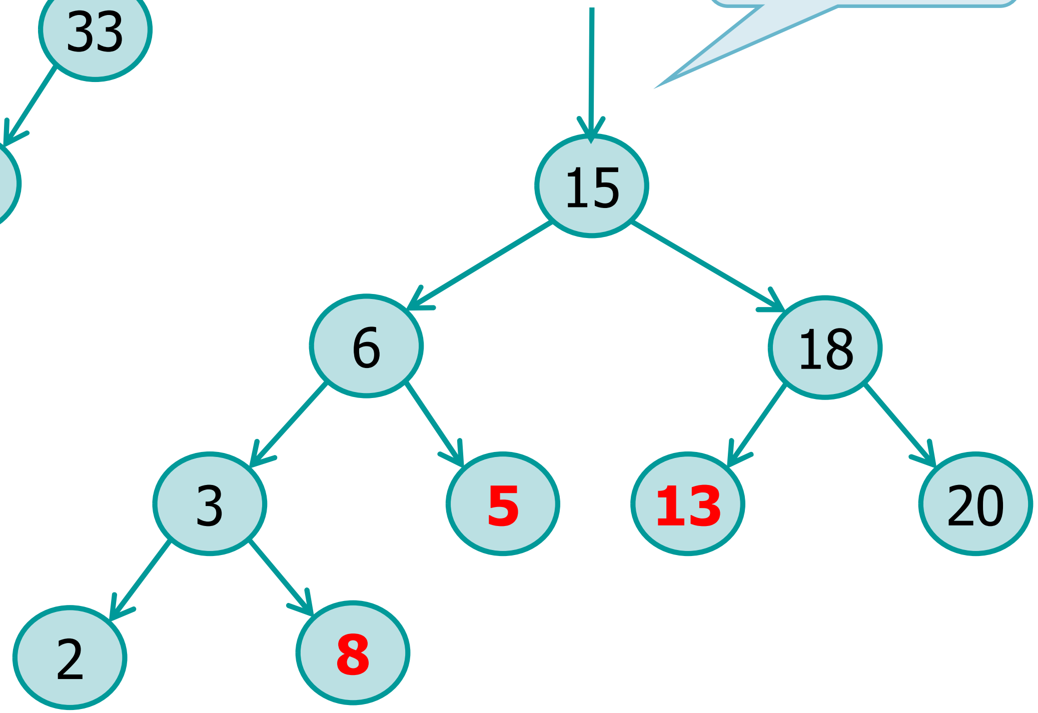


Examples

This is a BST



This is **not** a BST

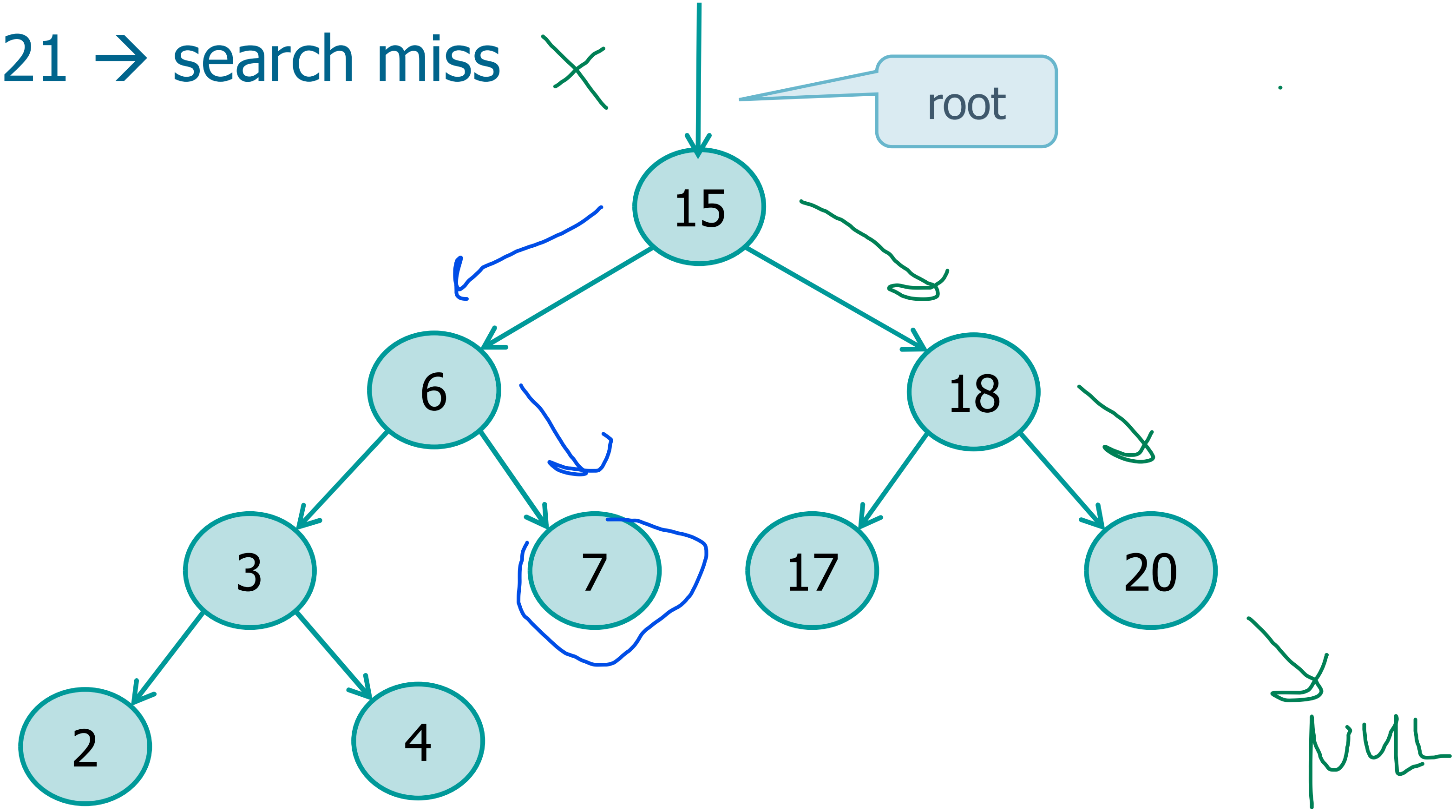


Search

- Given a BST, we can use a recursive procedure to search for a node storing the desired key
  - Visit the tree from the root
  - We terminate the search if
    - We find the key in the current node (search hit) or
    - We reach an empty tree (NULL pointer, i.e., search miss)
  - Recur from the current node on
    - The left sub-tree if the searched key is smaller than the key of the current node
    - The right sub-tree otherwise

Example

- Given the following BST look for
  - key = 7 → search hit
  - key = 20 → search hit
  - key = 21 → search miss



Recursive implementation

We suppose that the key is a string

Root node

Searched key

```
node_t *search_r (node_t *root, char *key) {

  if (root == NULL)
    return (NULL);

  if (strcmp(key, root->key) < 0)
    return (search_r (root->l, key));

  if (strcmp(key,root->key) > 0)
    return (search_r (root->r, key));

  return root;
}
```

Search miss

Left recursion

Right recursion

Search hit

Iterative implementation

We suppose that the key is a string

Root node

Searched key

```
node_t *search_i (node_t *root, char *key) {

  while (root != NULL) {
   if (strcmp (key, root->key) == 0)
     return (root);

   if (strcmp (key, root->key) < 0)
     root = root->l;
   else
     root = root->r;
  }

  return (root);
}
```

Search hit

Move down left

Move down right

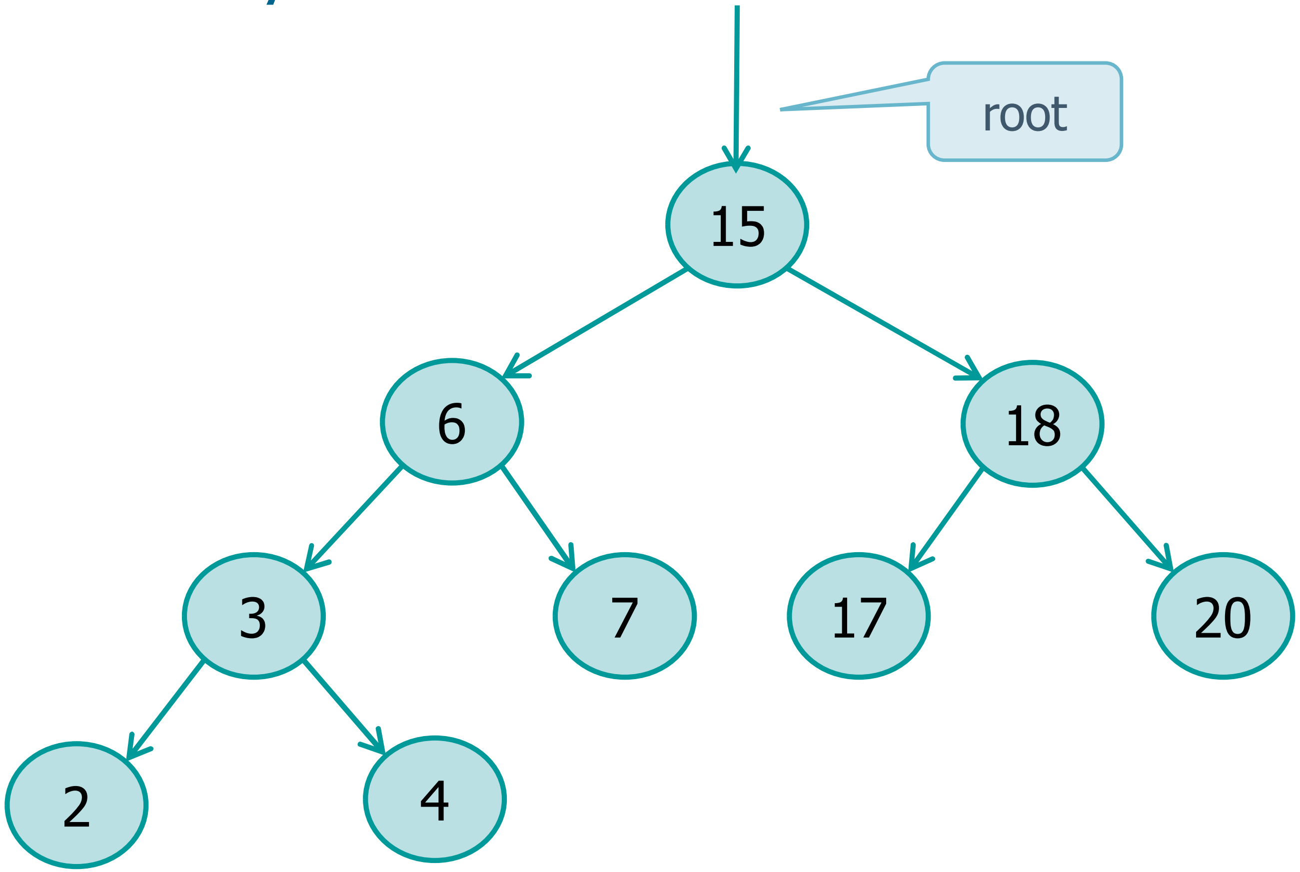
Search miss

Minimum and Maximum

- Find the minimum key in a given BST
  - If the BST is empty return NULL
  - Follow pointers onto **left** sub-trees until they exist
  - Return last key encountered
- Find the maximum ley in a given BST
  - If the BST is empty return NULL
  - Follow pointers onto **right** sub-trees until they exist
  - Return last key encountered

Example

- Given the following BST look for
  - Minimun → key = 2
  - Maximum → key = 20



Recursive implementation

```
node_t *min_r (node_t *root) {
  if (root == NULL)
    return (NULL);
  if (root->l == NULL)
    return (root);
  return min_r (root->l);
}
```

Empty BST

Termination condition

Left recursion

```
node_t *max_r (node_t *root) {
  if (root == NULL)
    return (NULL);
  if (root->r == NULL)
    return (root);
  return max_r (root->r);
}
```

Empty BST

Termination condition

Right recursion

Iterative implementation

```
node_t *min_i (node_t *root) {
  if (root == NULL)
    return (NULL);
  while (root->l != NULL)
    root = root->l;
  return (root);
}
```

Empty BST

Move down

Return result

```
node_t *max_i (node_t *root) {
  if (root == NULL)
    return (NULL);
  while (root->r != NULL)
    root = root->r;
  return (root);
}
```

Empty BST

Move down

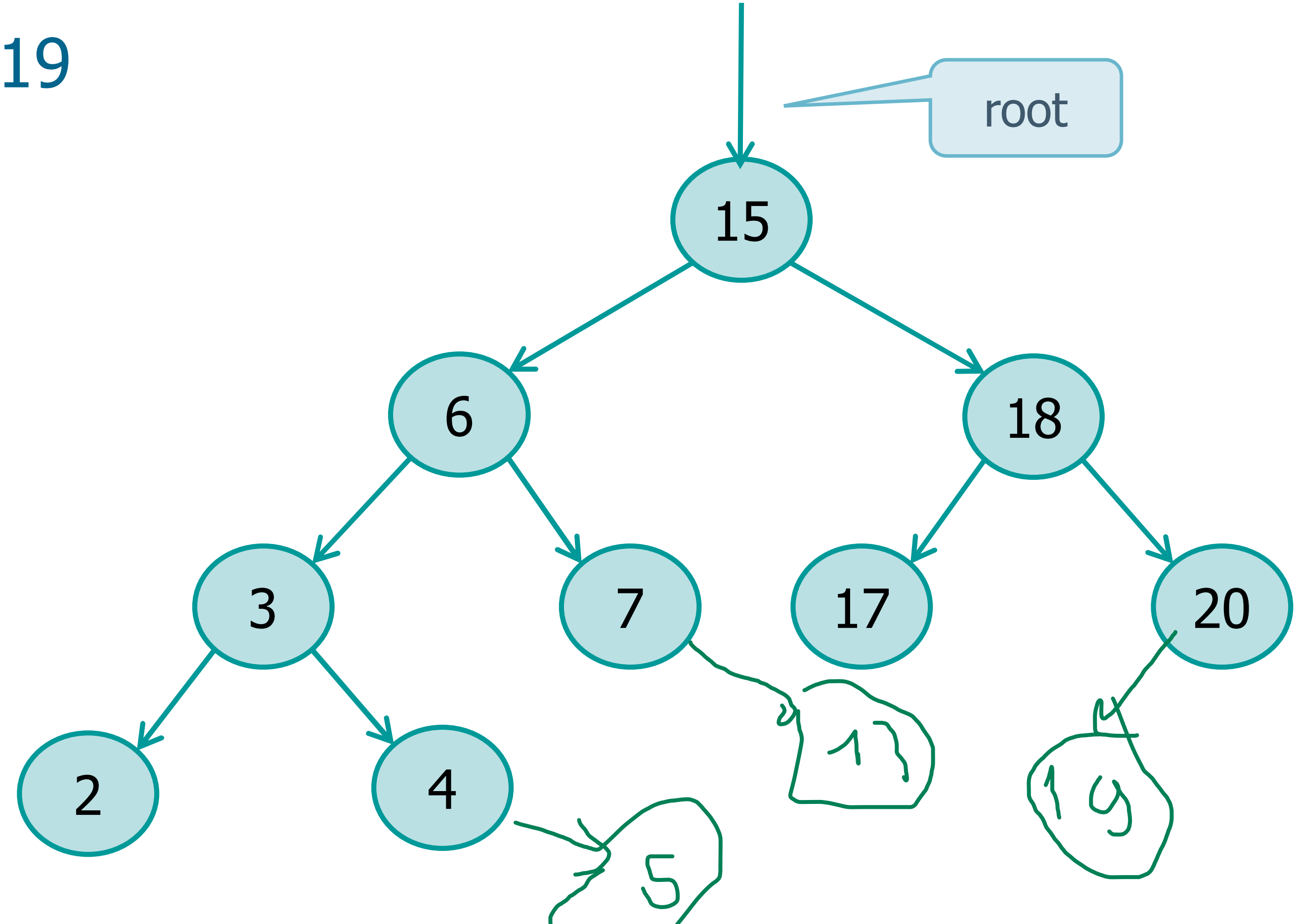
Return result

Leaf Insert

- Insert into a BST a node storing a new item
- The BST property must be maintained
  - If the BST is empty
    - Create a new tree node with the new key and return its pointer
  - Recursion
    - Insert into the left sub-tree if the item key is less than the current node key
    - Insert into the right sub-tree if the item key is larger than the current node key
- Notice that in all cases the new node in on a BST leaf (terminal node with no children)

Example

- Given the following BST insert
  - key = 5
  - key = 13
  - key = 19



Recursive implementation

Function **new_node** creates a new node

BST root

Key

Termination condition: Insert a new node

```
node_t *insert_r (node_t *root, char *key) {

  if (root == NULL)
    return (new_node(key, NULL, NULL));

  if (strcmp (key, root->key) < 0)
    root->l = insert_r (root->l, key);
  else
    root->r = insert_r (root->r, key);

  return root;
}
```

Left recursion

Right recursion

Assign (new) pointer onto parent pointer on the way back

Iterative implementation

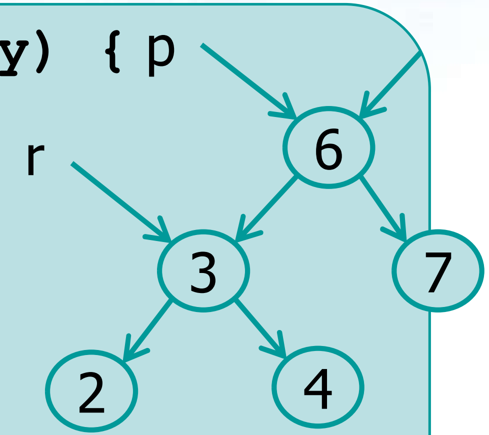
- BST insert can be also be performed using an iterative procedure
  - Find the position first
  - Then add the new node
- As we cannot assign the new pointer on the way back (on recursion) we need two pointers
  - Please remind the ordered list implementation
    - The visit was perfomed either using two pointers or the pointer of a pointer to assign the new pointer to the pointer of the previous element

either recursion or we need the pointer to pointer to operate on previous level

Iterative implementation

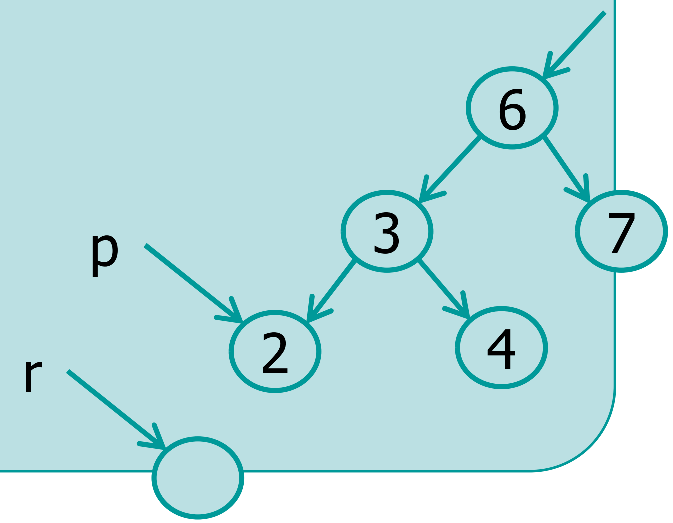
```
node_t *insert_i (node_t *root, char *key) {
  node_t *p, r;

  if (root == NULL) {
    return (new_node(key, NULL, NULL));
  }
  r = root;
  p = r;
  while (r != NULL) {
    p = r;
    r = (strcmp(key,r->key)<0) ? r->l : r->r;
  }
  r = new_node (key, NULL, NULL);
  if (strcmp (key, p->key) < 0)
    p->l = r;
  else
    p->r = r;
  return root;
}
```



Move left or move right

Create link with parent in the right direction



Node Extract

- Given a BST delete a node with a given key
  - We have to recursively search the key into the BST
  - If we find it
    - Then, we must delete it
    - Otherwise, the key is not in the BST and we just return
- Search is performed as before and it is followed by the procedure to delete the node

Node Extract

- To sum up we have to
  - If the BST is empty
    - Return doing nothing
  - If the current node is the one with the desired key, then apply one of the following three basic rules
    - Rule 1: If the node has no children, simply remove it
    - Rule 2: If the node has one child, then move the child one level higher in the tree to substitute the erased node in the tree with its child
    - Rule 3: If the node has two children, find
      - The greatest node in its left subtree or
      - The smallest node in its right subtree

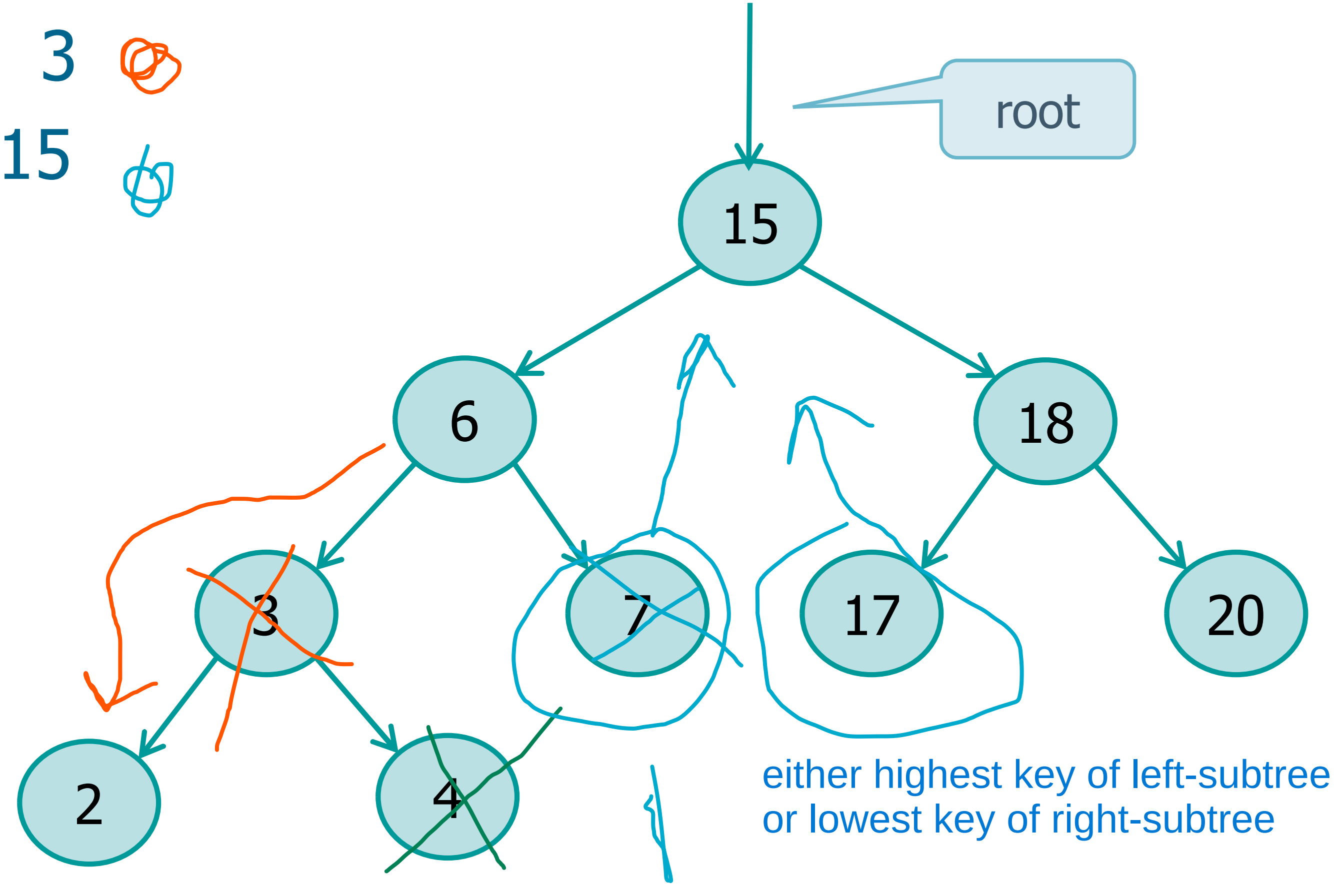
      and substitute the erased node with it

Node Extract

- If the current node is not the one with the desired key
  - Recur onto the left sub-tree in the key is smaller than the node's key
  - Recur onto the right sub-tree in the key is smaller than the node's key

Example

- Given the following BST delete the folliwing keys in the given order
  - key = 4
  - key = 3
  - key = 15



Recursive implementation

```
node_t *delete_r (node_t *root, char *key) {
  link p;
  char *val;

  if (root == NULL)
    return (root);

  if (strcmp (key, root->key) < 0) {
    root->l = delete_r (root->l, key);
    return (root);
  }
  if (strcmp (key, root->key) > 0) {
    root->r = delete_r (root->r, key);
    return (root);
  }
```

Empty BST

Left recursion

Right recursion

Recursive implementation

this code is reached if
strcmp returns 0

Node found

```
p = root;
if (root->r == NULL) {
   root = root->l;
   free (p);
   return (root);
}
if (root->l == NULL) {
   root = root->r;
   free (p);
   return (root);
}
root->l = max_delete_r (&val, root->l);
root->key = val;
return (root);
}
```

Rule 1 or 2
Right child = NULL

Rule 1 or 2
Left child = NULL

in the case above, if 15
root->l=6,
free(root=7)
15 value changed to 7
return root 7;

Rule ~~1~~ 3
Node with 2 children
(find max into left sub-tree)

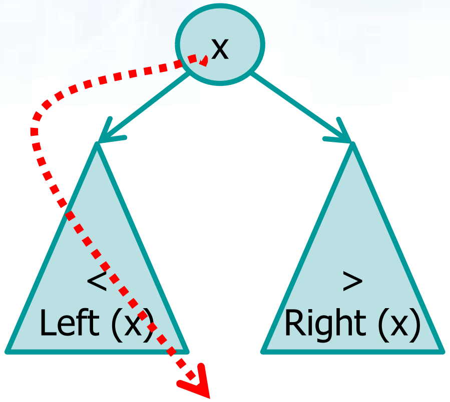
Recursive implementation

Find and delete maximum value into left sub-tree

```
node_t *max_delete_r (
  char *val, node_t *root) {
  link tmp;

  if (root->r == NULL) {
    *val = root->key;
    tmp = root->l;
    free (root);
    return (tmp);
  }

  root->r = max_delete_r (val, root->r);
  return (root);
}
```



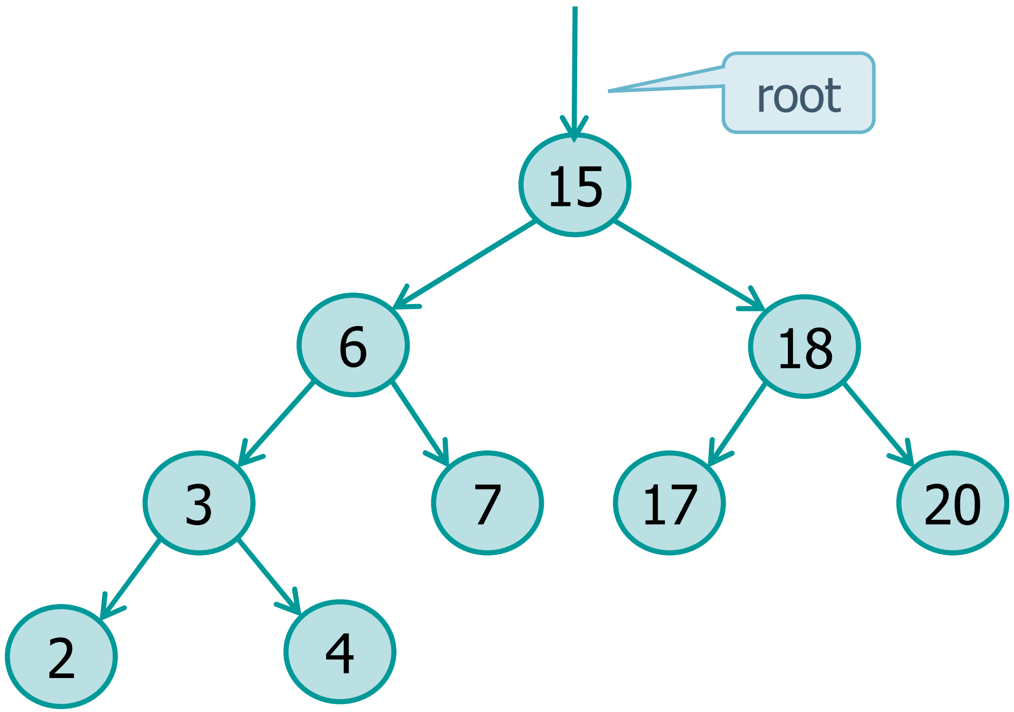
Node found
Free the node and return pointer to left child

Recur until there is no right child

Alternative solution: Find and delete minimum value into right sub-tree

Sorting and Median

- Given a BST
  - An in-order visit delivers keys in ascending order
  - Ascending order
    - 2 3 4 6 7 15 17 18 20



Sorting and Median

- Given a BST
  - The (inferior) **median key** of a set of n elements is the element stored in position $\lfloor (n+1)/2 \rfloor$ in the ordered sequence of the element set

Ascending order

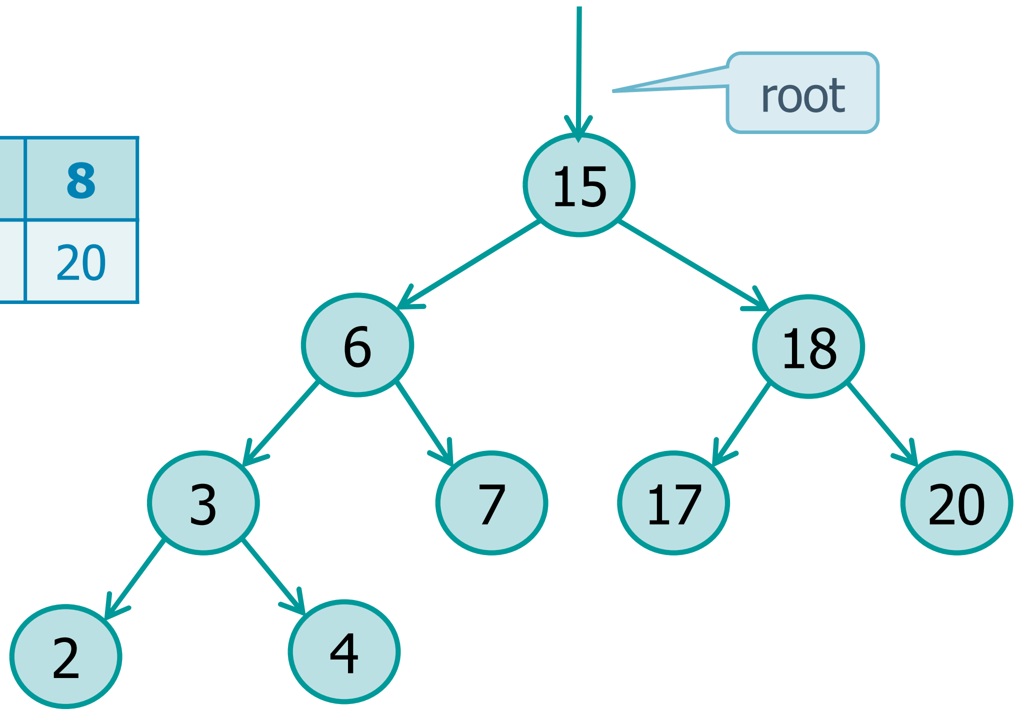
0	1	2	3	4	5	6	7	8
2	3	4	6	7	15	17	18	20

$\lfloor \frac{n+1}{2} \rfloor = \lfloor \frac{9+1}{2} \rfloor = 5$

→ position 5

→ element of index 4

→ 7 is the median key



Sorting and Median

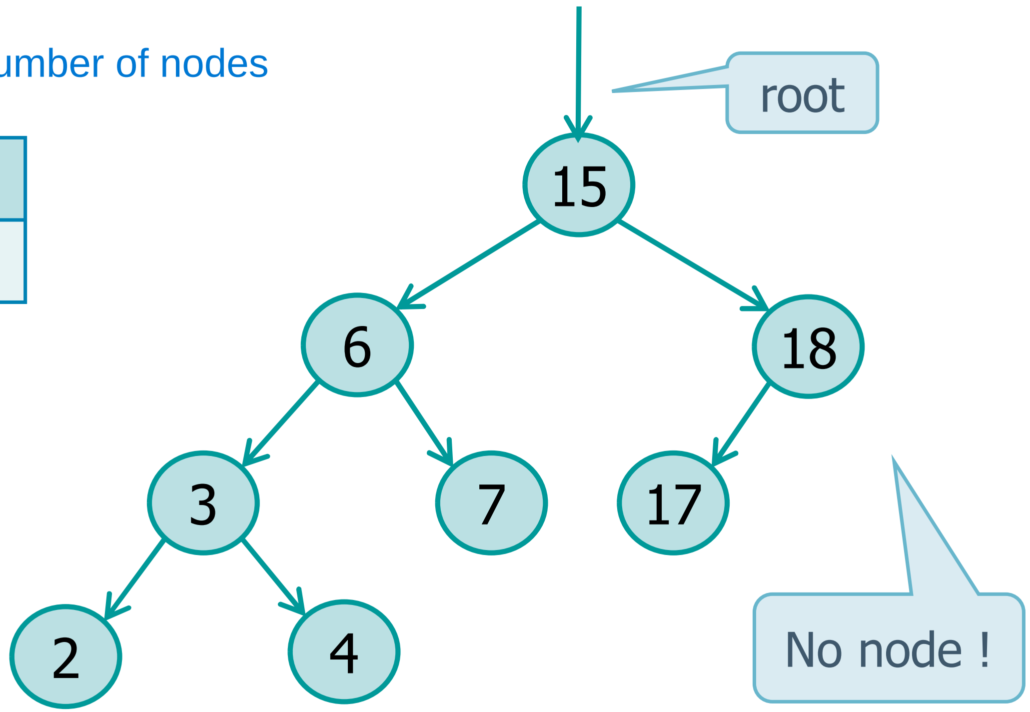
- Given a BST
  - The (inferior) **median key** of a set of n elements is the element stored in position $\lfloor (n+1)/2 \rfloor$ in the ordered sequence of the element set

Ascending order

if odd number of nodes

0	1	2	**3**	4	5	6	7
2	3	4	**6**	7	15	17	18

$\lfloor \frac{n+1}{2} \rfloor = \lfloor \frac{8+1}{2} \rfloor = 4$
→ position 4
→ element of index 3
→ 6 is the median key

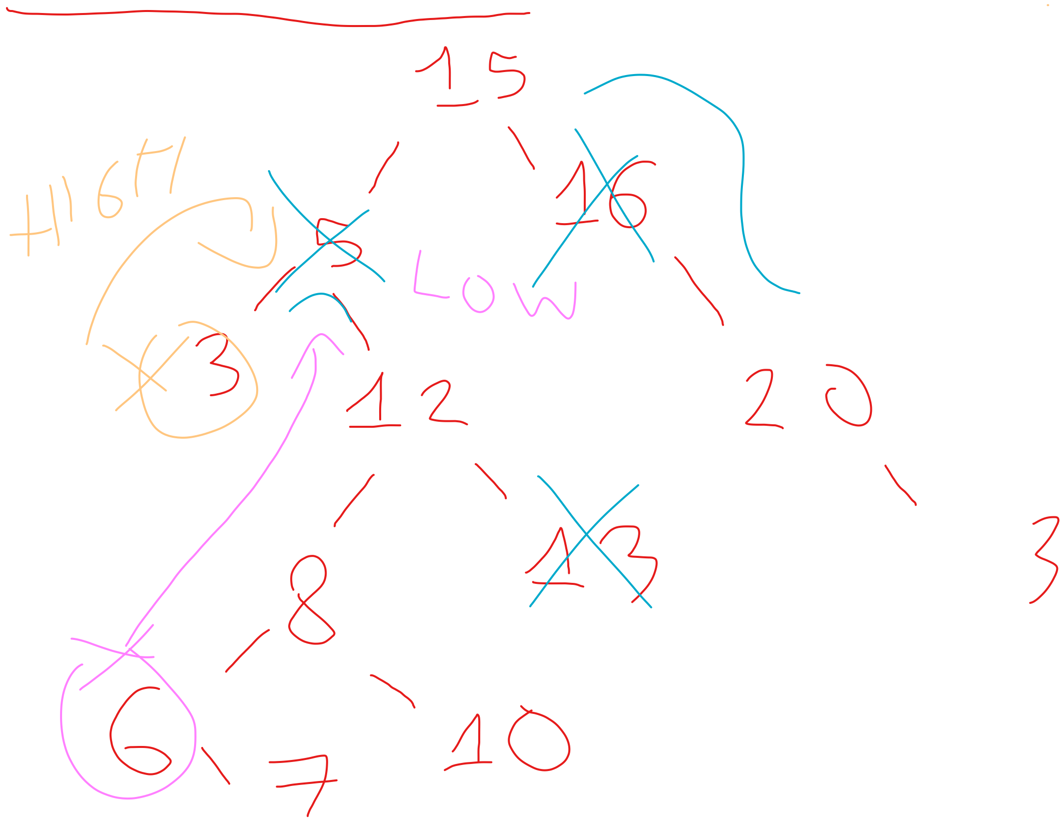


Complexity

- Operations on BSTs have complexity
  - $T(n) = O(h)$
    - Where h is the height of the tree
- The height of a tree is equal to
  - Tree fully balanced with n nodes
    - $Height\ h = \alpha(log_2 n)$
  - Tree completely unbalanced with n nodes
    - $Height\ h = \alpha(n)$
  - $O(log_2 n) \leq T(n) \leq O(n)$

Exercise

- Given an initially empty BST perform the following insertions (+) and extractions (–)
  - +15 +16 +5 +3 +12 +20 +13 +8 +10 +23 +6 +7 –13 –16 – 5



Exercise

- Suppose numbers between 1 and 1000 are stored in a BST, and we want to search for the key 363
- Which of the following sequences could 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
  - 2 399 387 219 266 382 385 278 363
  - 935 278 347 621 392 358 363

H L

check if this sequence is monotone

Exercise

- Suppose numbers between 1 and 1000 are stored in a BST, and we want to search for the key 363
- Which of the following sequences could be the sequence of nodes examined?
  - 2 252 401 398 330 344 397 363 — OK
  - 924 220 911 244 898 258 362 363 — OK
  - 925 202 911 240 912 245 363 — NO
  - 2 399 387 219 266 382 385 278 363 — NO
  - 935 278 347 621 392 358 363 — OK

The BST Library (for the laboratory)

- The BST library includes the following modules