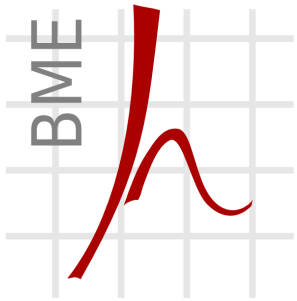
Binary trees

Basics of Programming 1



Department of Networked Systems and Services
G. Horváth, A.B. Nagy, Z. Zsóka, P. Fiala, A. Vitéz

2 December, 2020

Content

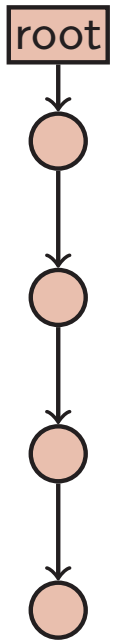
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Chapter 1

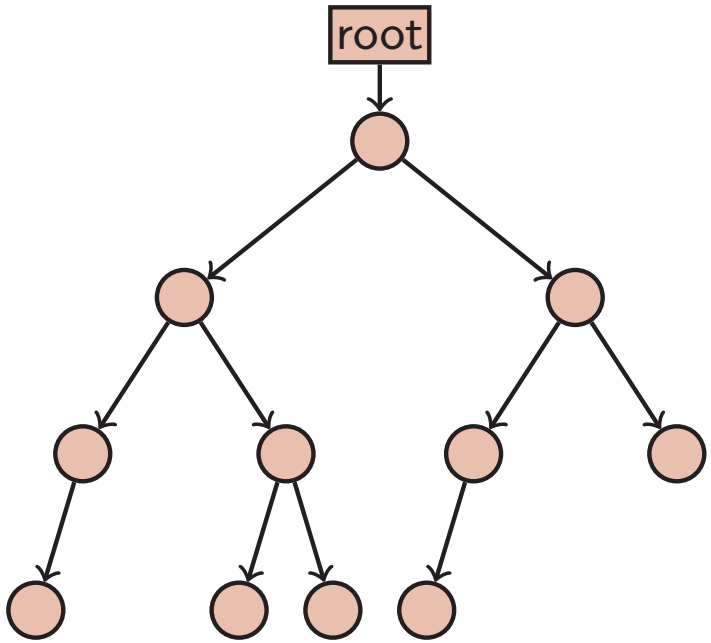
Binary trees

Trees

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



$K = 1$ (linked list)

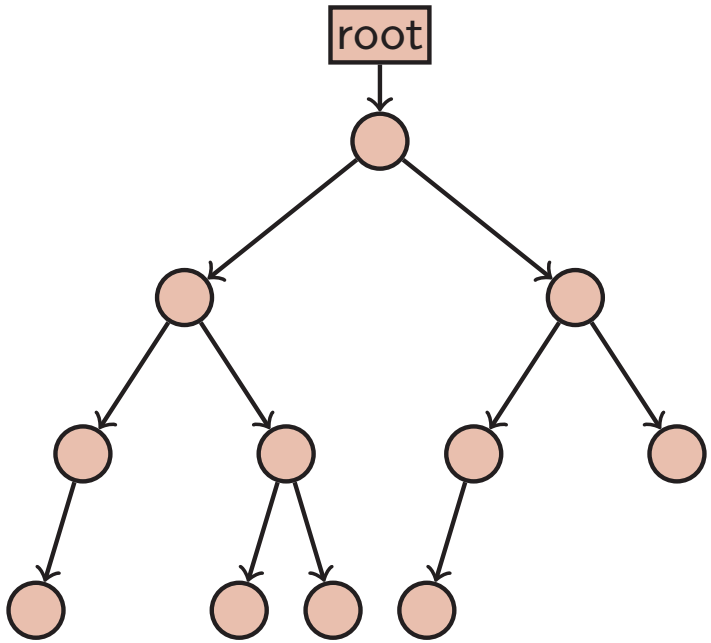


$K = 2$ (binary tree)

- An acyclic graph
- Every node has exactly one incoming edge
- K-ary tree: every node has at most K outgoing edges

Binary trees

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- Declaration of the binary tree data structure

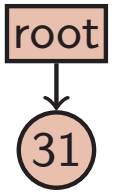
```
1  typedef struct tree {
2    int data;
3    struct tree *left, *right;
4  } tree_elem, *tree_ptr;
```

link

- Typically we typedef not only the struct, but also the pointer

Binary search trees

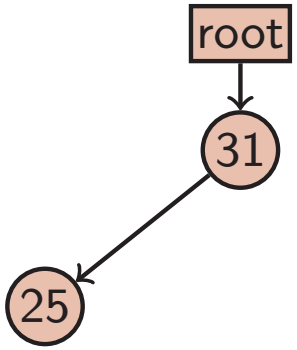
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- Sub-tree to the left: only elements smaller than the node
- Sub-tree to the right: only elements greater than the node
- The structure of the tree depends on the insertion order of the elements!

Binary search trees

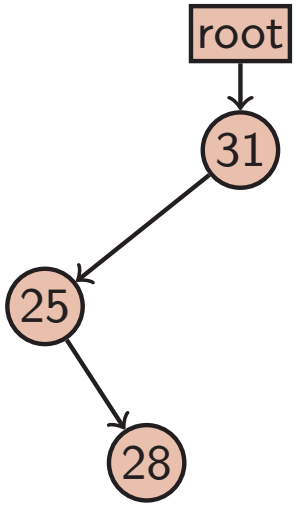
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- Sub-tree to the left: only elements smaller than the node
- Sub-tree to the right: only elements greater than the node
- The structure of the tree depends on the insertion order of the elements!

Binary search trees

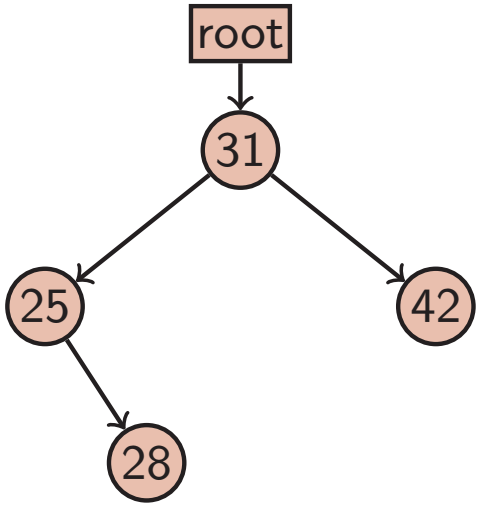
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- Sub-tree to the left: only elements smaller than the node
- Sub-tree to the right: only elements greater than the node
- The structure of the tree depends on the insertion order of the elements!

Binary search trees

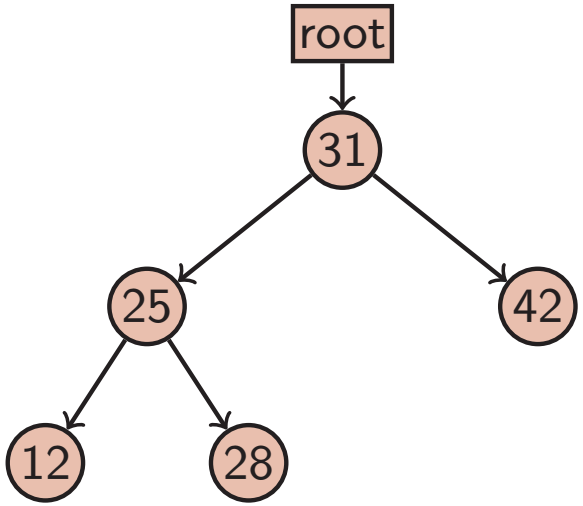
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- Sub-tree to the left: only elements smaller than the node
- Sub-tree to the right: only elements greater than the node
- The structure of the tree depends on the insertion order of the elements!

Binary search trees

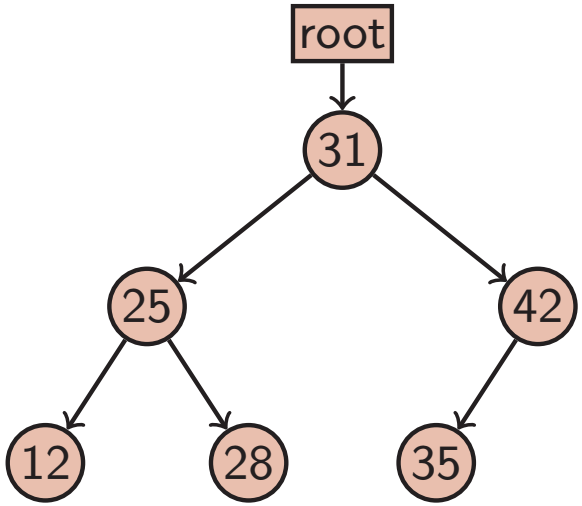
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- Sub-tree to the left: only elements smaller than the node
- Sub-tree to the right: only elements greater than the node
- The structure of the tree depends on the insertion order of the elements!

Binary search trees

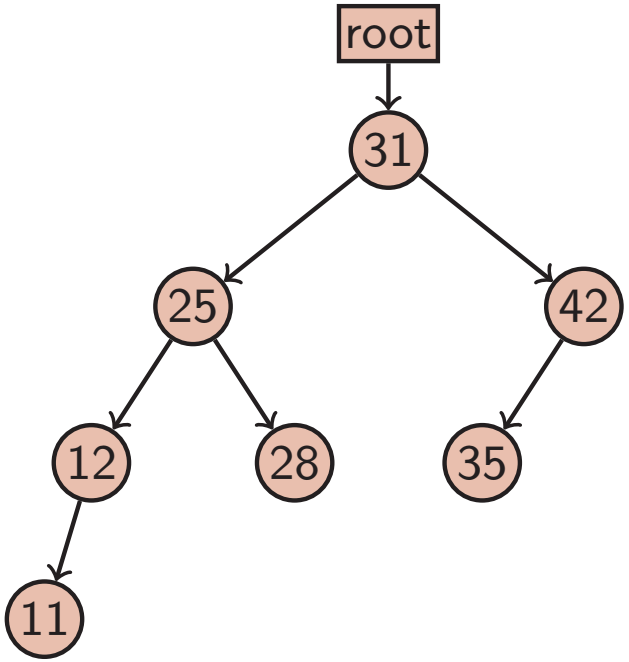
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- Sub-tree to the left: only elements smaller than the node
- Sub-tree to the right: only elements greater than the node
- The structure of the tree depends on the insertion order of the elements!

Binary search trees

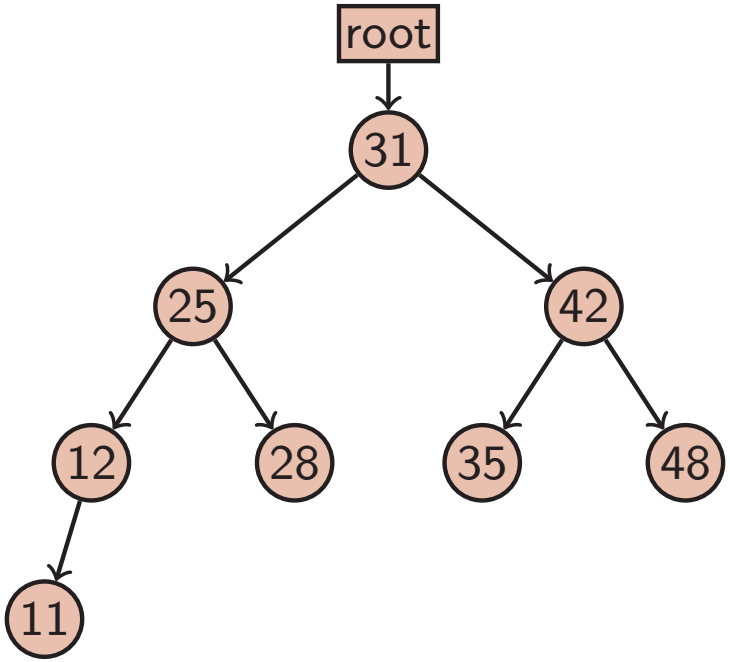
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- Sub-tree to the left: only elements smaller than the node
- Sub-tree to the right: only elements greater than the node
- The structure of the tree depends on the insertion order of the elements!

Binary search trees

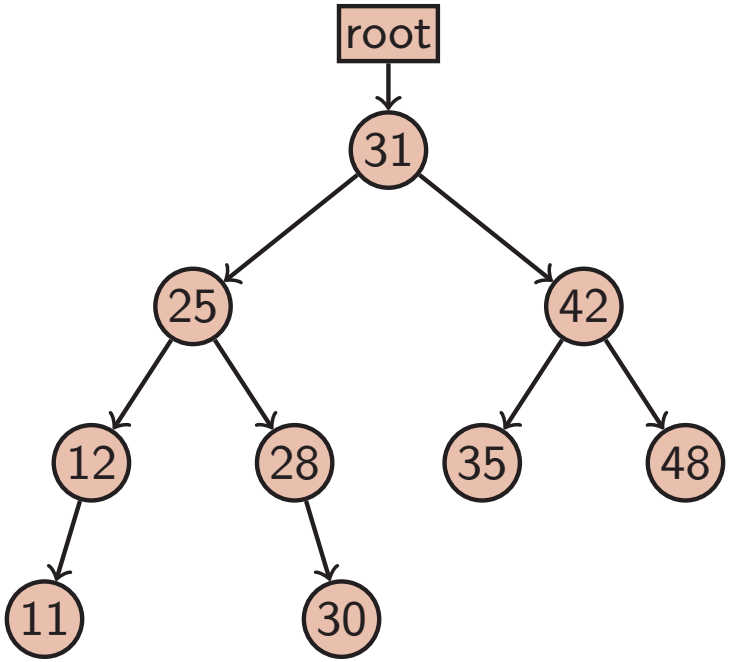
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- Sub-tree to the left: only elements smaller than the node
- Sub-tree to the right: only elements greater than the node
- The structure of the tree depends on the insertion order of the elements!

Binary search trees

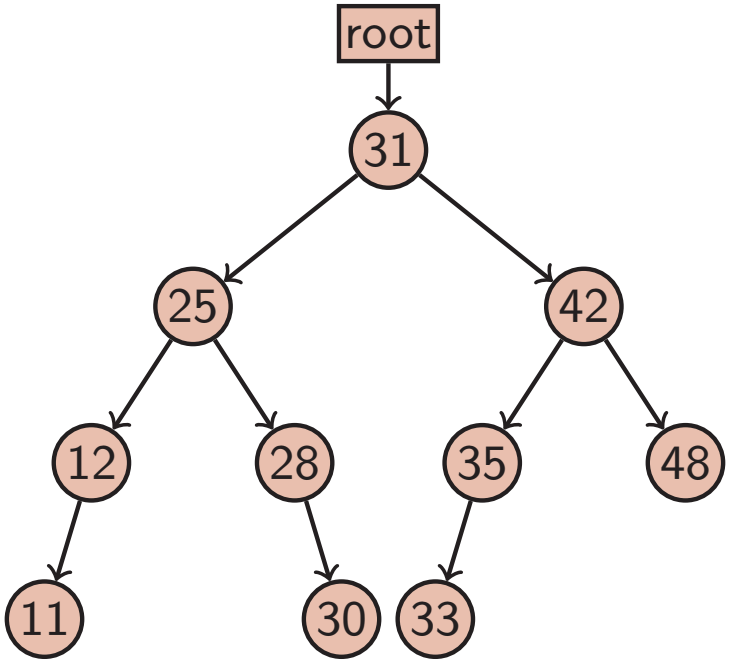
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- Sub-tree to the left: only elements smaller than the node
- Sub-tree to the right: only elements greater than the node
- The structure of the tree depends on the insertion order of the elements!

Binary search trees

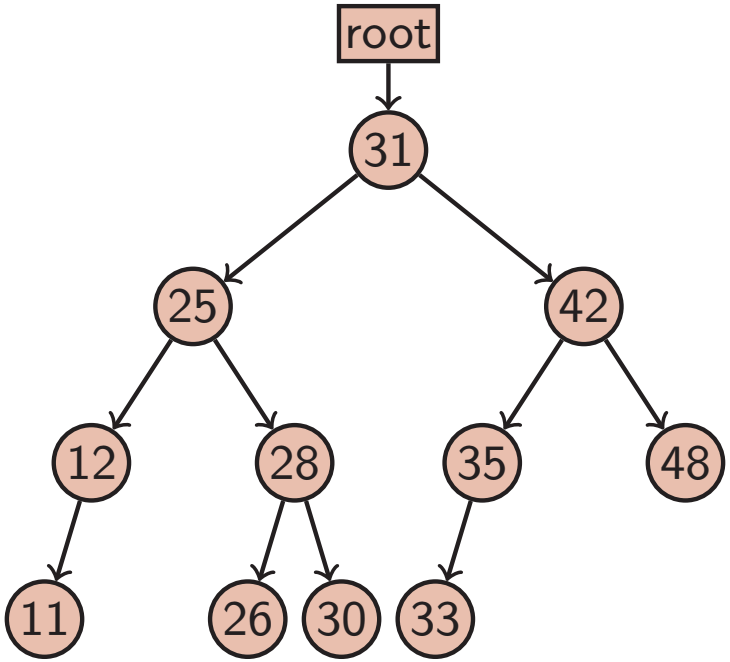
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- Sub-tree to the left: only elements smaller than the node
- Sub-tree to the right: only elements greater than the node
- The structure of the tree depends on the insertion order of the elements!

Binary search trees

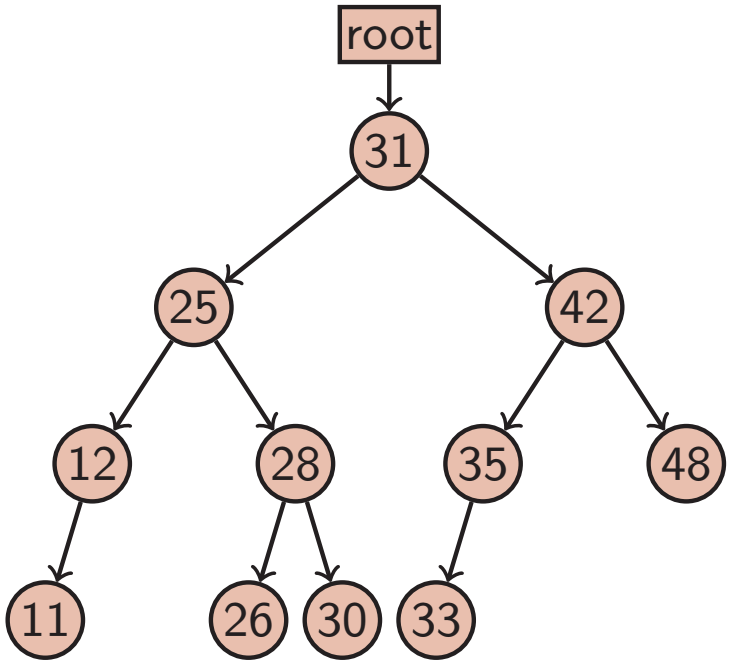
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



31 25 28 42 12 35 11 48 30 33 26

- Sub-tree to the left: only elements smaller than the node
- Sub-tree to the right: only elements greater than the node
- The structure of the tree depends on the insertion order of the elements!

Searching an element in the tree

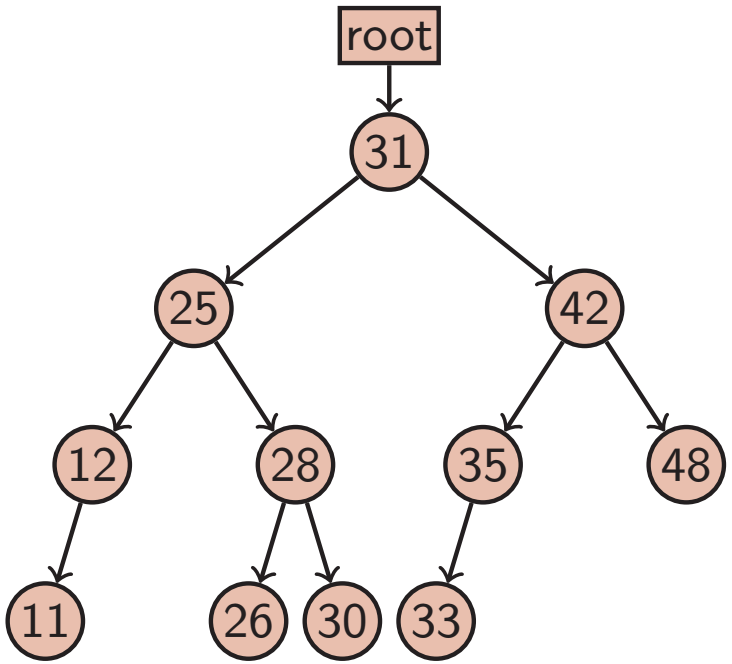



```
1  tree_ptr find(tree_ptr root,
2                int data)
3  {
4    while (root != NULL &&
5      data != root->data)
6    {
7      if (data < root->data)
8        root = root->left;
9      else
10       root = root->right;
11   }
12   return root;
13 }
```

link

Searching an element in the tree

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



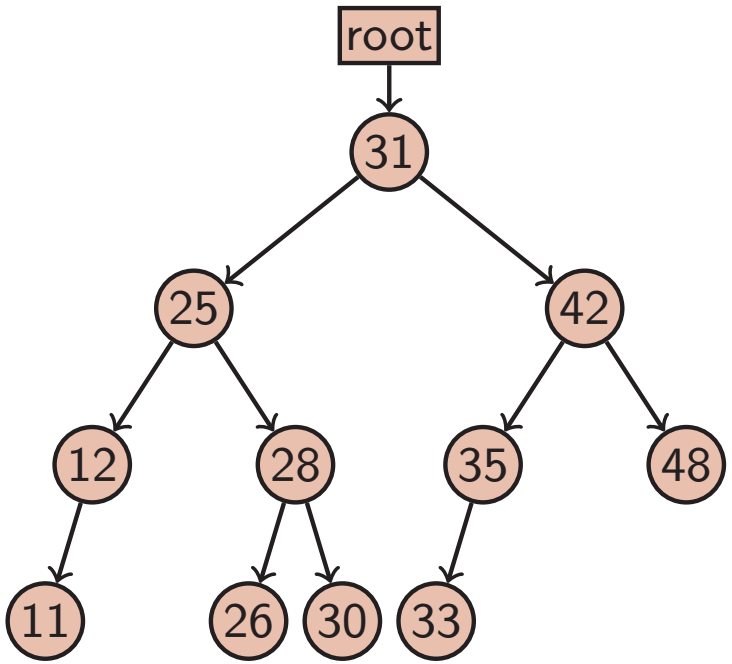

```
1  tree_ptr find(tree_ptr root,
2                int data)
3  {
4    while (root != NULL &&
5      data != root->data)
6    {
7      if (data < root->data)
8        root = root->left;
9      else
10       root = root->right;
11   }
12   return root;
13 }
```

link

- This is not recursive yet

Searching an element in the tree

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



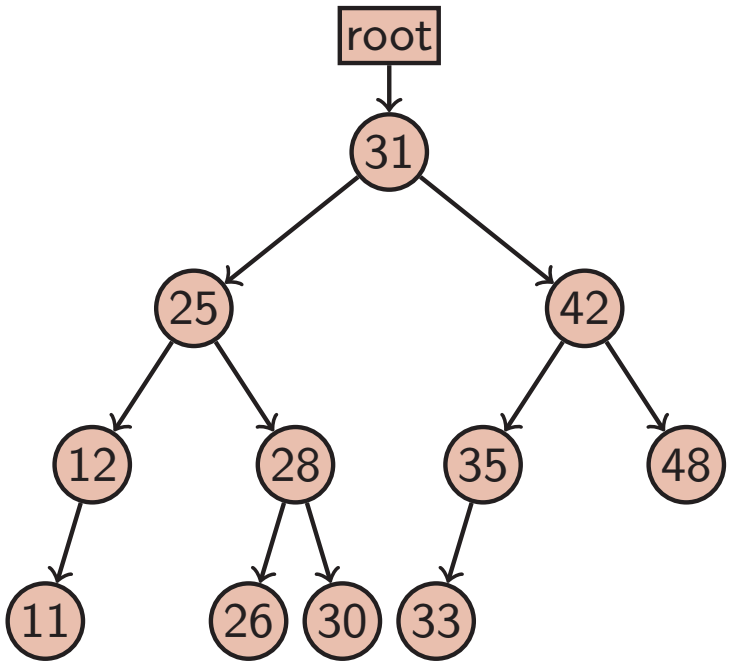

```
1  tree_ptr find(tree_ptr root,
2                int data)
3  {
4    while (root != NULL &&
5           data != root->data)
6    {
7      if (data < root->data)
8        root = root->left;
9      else
10       root = root->right;
11   }
12   return root;
13 }
```

link

- This is not recursive yet
- In a depth-d tree the max. number of steps is d

Searching an element in the tree

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

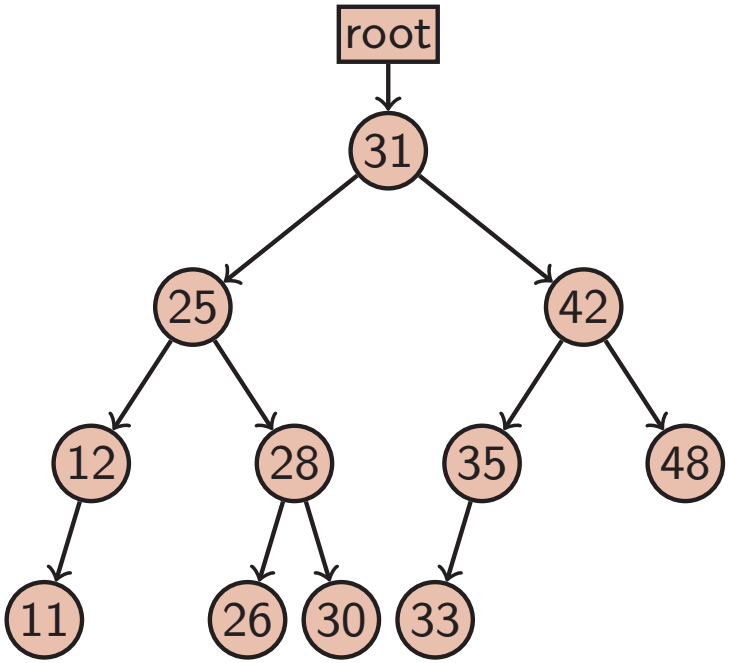



```
1  tree_ptr find(tree_ptr root,
2                int data)
3  {
4    while (root != NULL &&
5           data != root->data)
6    {
7      if (data < root->data)
8        root = root->left;
9      else
10       root = root->right;
11   }
12   return root;
13 }
```

link

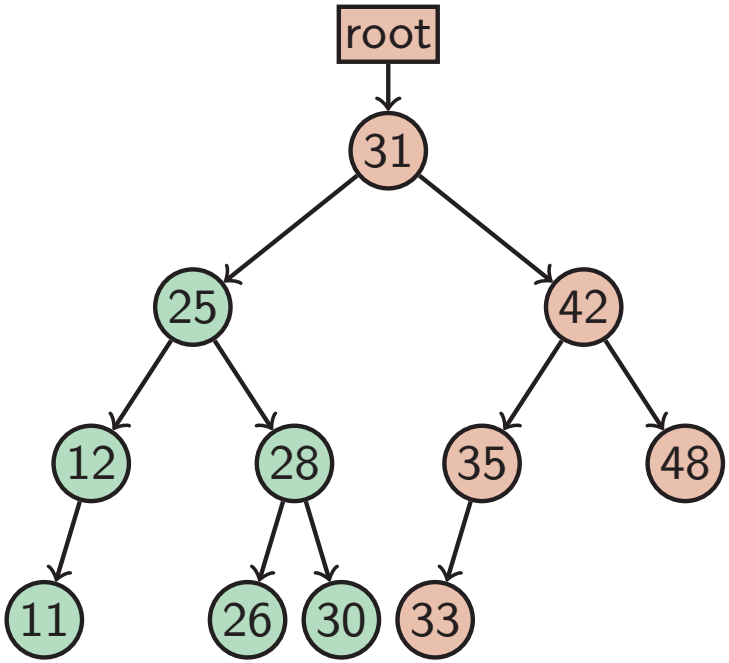
- This is not recursive yet
- In a depth-d tree the max. number of steps is d
- If the tree is balanced and has n elements $\Rightarrow \approx \log_2 n$ steps!

In-order traversal




```
1  void inorder(tree_ptr root)
2  {
3    if (root == NULL)
4      return;
5    inorder(root->left);
6    printf("%d ", root->data);
7    inorder(root->right);
8  }
```

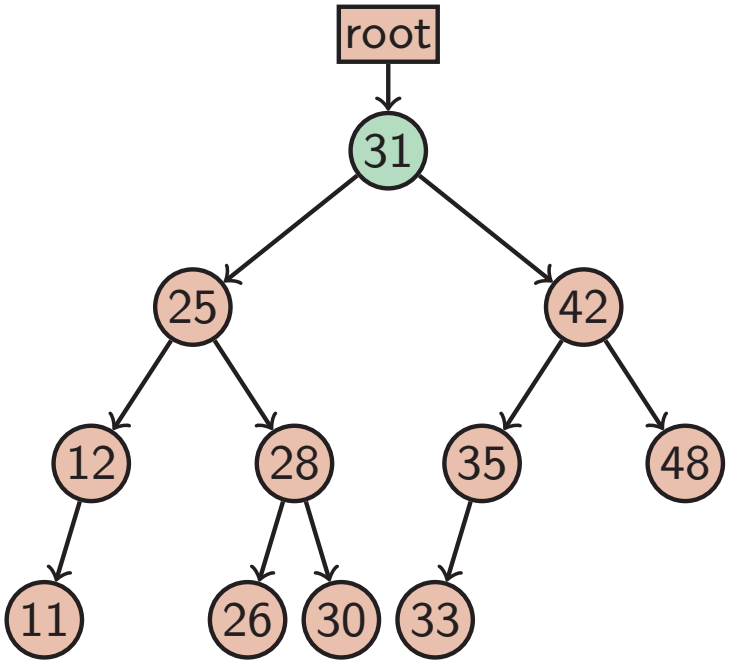
In-order traversal




```
1  void inorder(tree_ptr root)
2  {
3    if (root == NULL)
4      return;
5    inorder(root->left);
6    printf("%d ", root->data);
7    inorder(root->right);
8  }
```

```
11 12 25 26 28 30
```

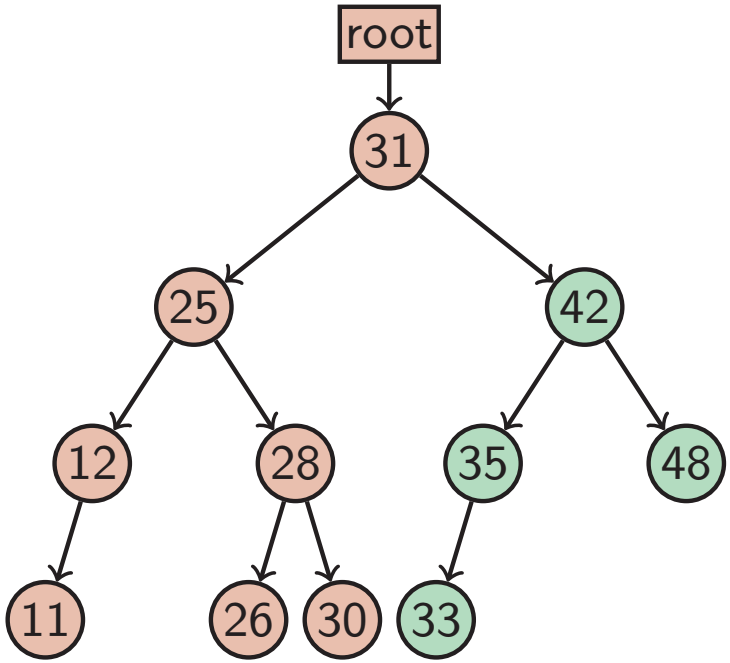
In-order traversal




```
1  void inorder(tree_ptr root)
2  {
3    if (root == NULL)
4      return;
5    inorder(root->left);
6    printf("%d ", root->data);
7    inorder(root->right);
8  }
```

```
11 12 25 26 28 30 31
```

In-order traversal

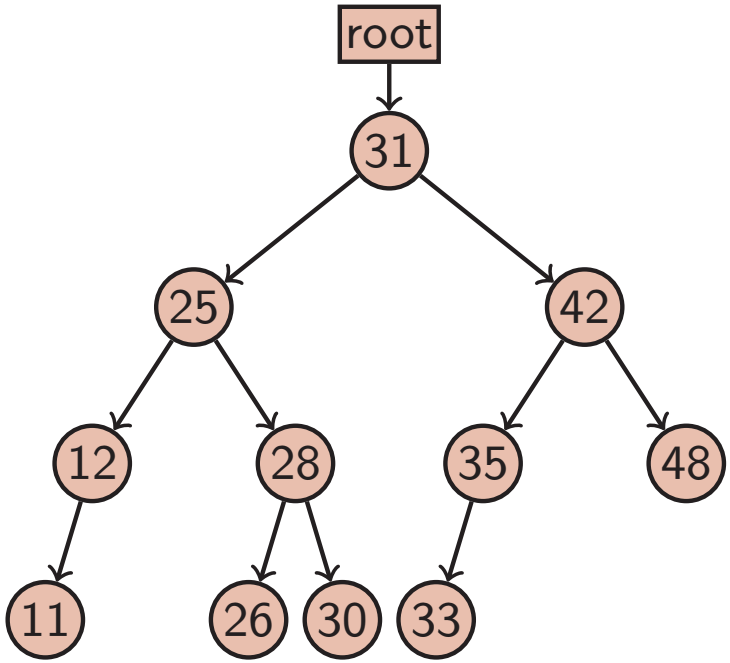



```
1  void inorder(tree_ptr root)
2  {
3    if (root == NULL)
4      return;
5    inorder(root->left);
6    printf("%d ", root->data);
7    inorder(root->right);
8  }
```

```
11 12 25 26 28 30 31 33 35 42 48
```

In-order traversal

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES




```
1  void inorder(tree_ptr root)
2  {
3    if (root == NULL)
4      return;
5    inorder(root->left);
6    printf("%d ", root->data);
7    inorder(root->right);
8  }
```

```
11 12 25 26 28 30 31 33 35 42 48
```

- in-order traversal
  1. left sub-tree
  2. root element
  3. right sub-tree

With this traversal the nodes are visited in increasing order of their values

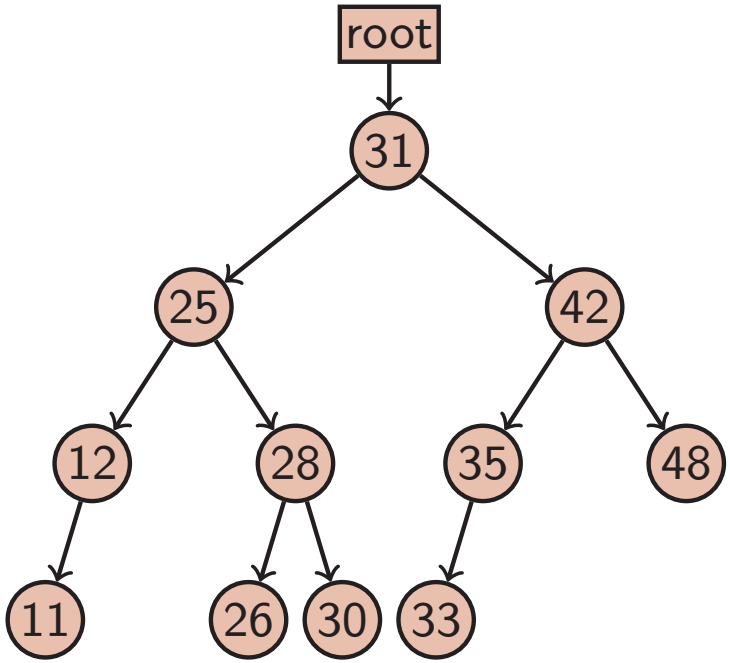
In-order traversal

An other implementation of the in-order traversal:

```
1  void inorder(tree_ptr root)
2  {
3    if (root->left != NULL)
4      inorder(root->left);
5    printf("%d ", root->data);
6    if (root->right != NULL)
7      inorder(root->right);
8  }
```

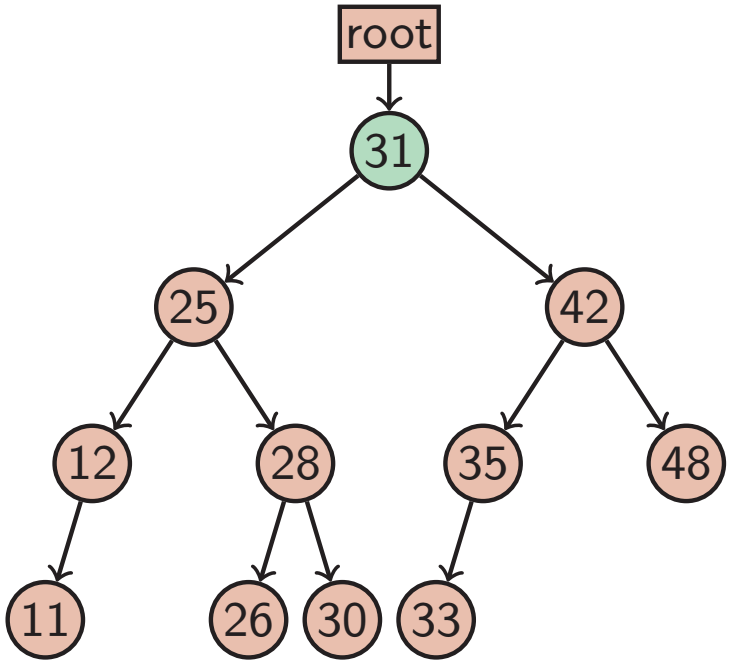
But in this case the caller has not make sure that `root != NULL` holds

Pre-order traversal




```
1  void preorder(tree_ptr root)
2  {
3    if (root == NULL)
4      return;
5    printf("%d ", root->data);
6    preorder(root->left);
7    preorder(root->right);
8  }
```

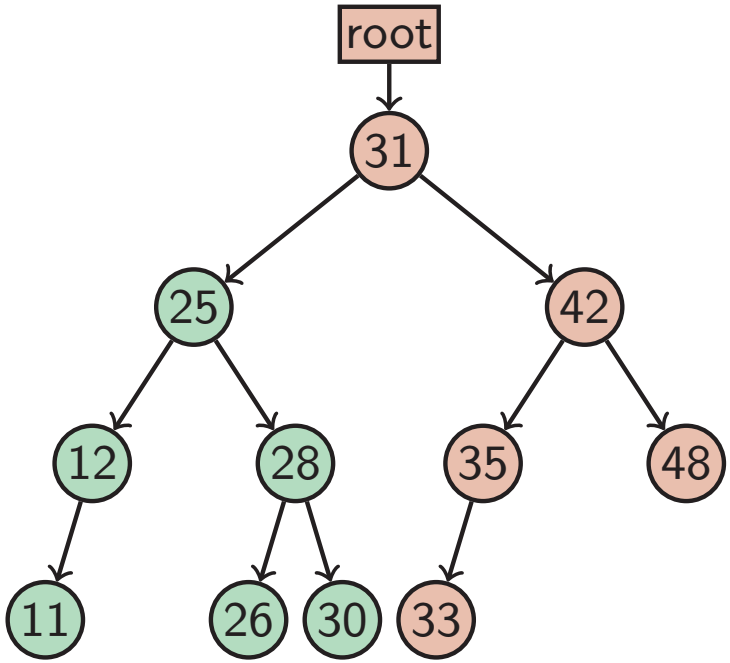
Pre-order traversal




```
1  void preorder(tree_ptr root)
2  {
3    if (root == NULL)
4      return;
5    printf("%d ", root->data);
6    preorder(root->left);
7    preorder(root->right);
8  }
```

```
31
```

Pre-order traversal

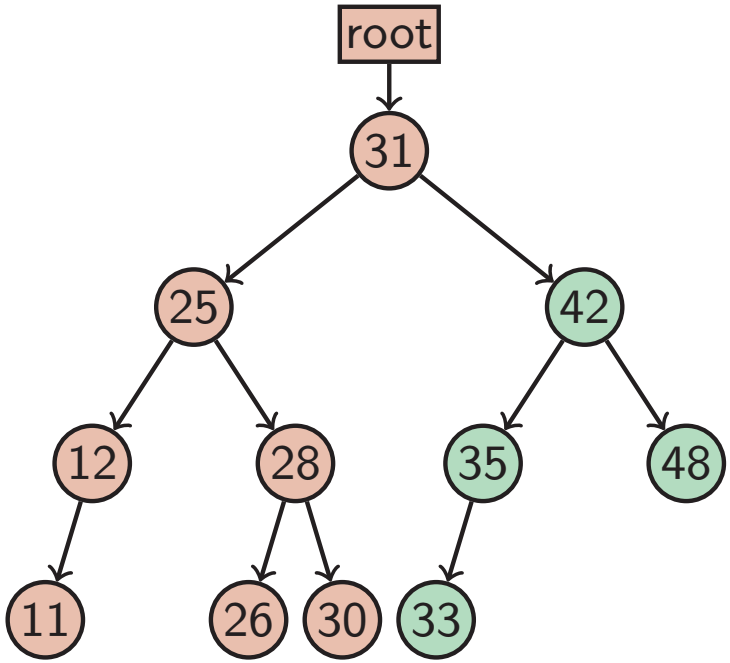



```
1  void preorder(tree_ptr root)
2  {
3    if (root == NULL)
4      return;
5    printf("%d ", root->data);
6    preorder(root->left);
7    preorder(root->right);
8  }
```

```
31 25 12 11 28 26 30
```

Pre-order traversal

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

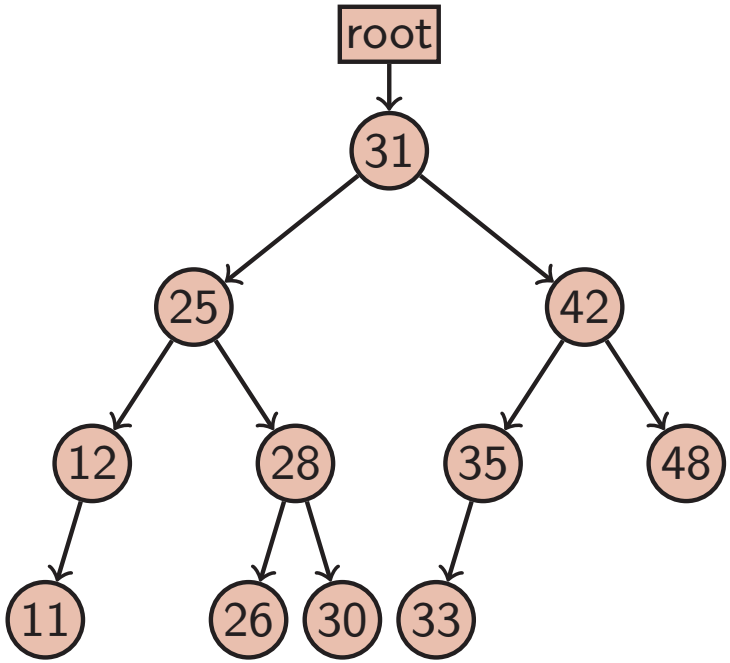



```
1  void preorder(tree_ptr root)
2  {
3    if (root == NULL)
4      return;
5    printf("%d ", root->data);
6    preorder(root->left);
7    preorder(root->right);
8  }
```

```
31 25 12 11 28 26 30 42 35 33 48
```

Pre-order traversal

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES




```
1  void preorder(tree_ptr root)
2  {
3    if (root == NULL)
4      return;
5    printf("%d ", root->data);
6    preorder(root->left);
7    preorder(root->right);
8  }
```

```
31 25 12 11 28 26 30 42 35 33 48
```

- pre-order traversal
  1. root element
  2. left sub-tree
  3. right sub-tree

Saving the elements of the tree in this order, and building it again, the structure of the tree can be fully reconstructed.

Building a tree

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Inserting a new node to the tree

```
1  tree_ptr insert(tree_ptr root, int data)
2  {
3    if (root == NULL) {
4      root = (tree_ptr)malloc(sizeof(tree_elem));
5      root->data = data;
6    }
7    else if (data < root->data)
8      root->left = insert(root->left, data);
9    else
10     root->right = insert(root->right, data);
11   return root;
12 }
```

link

Building a tree

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Inserting a new node to the tree

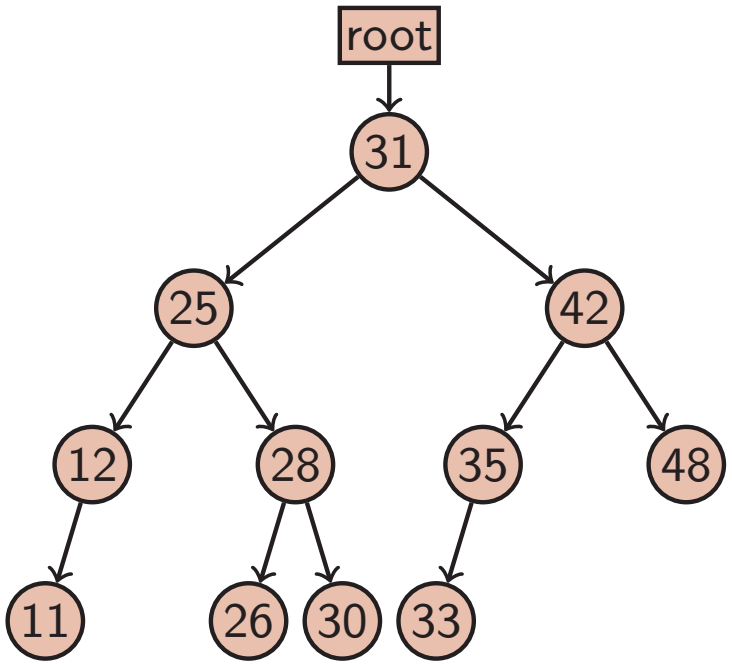
```
1  tree_ptr insert(tree_ptr root, int data)
2  {
3    if (root == NULL) {
4      root = (tree_ptr)malloc(sizeof(tree_elem));
5      root->data = data;
6    }
7    else if (data < root->data)
8      root->left = insert(root->left, data);
9    else
10     root->right = insert(root->right, data);
11   return root;
12 }
```

link

Usage of this function:

```
1  tree_ptr root = NULL;
2  root = insert(root, 2);
3  root = insert(root, 8);
4  ...
```

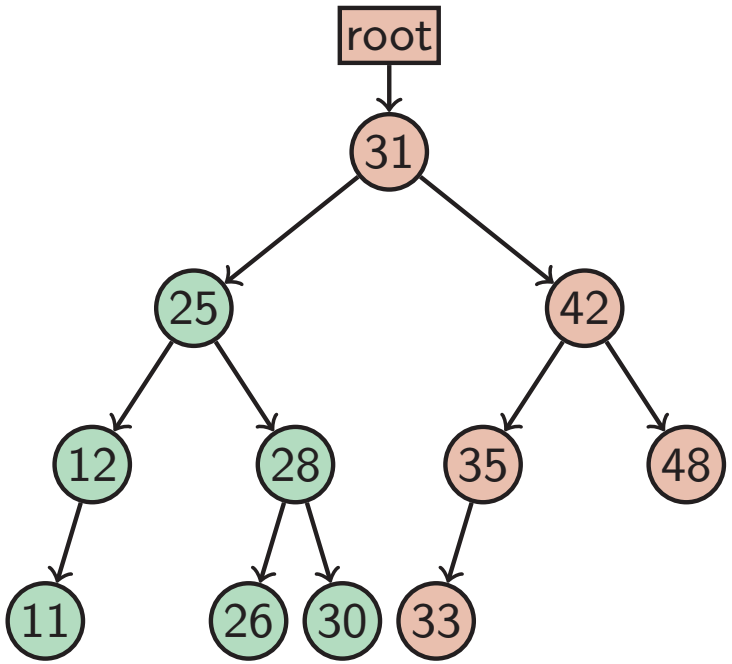
Post-order traversal




```
1  void postorder(tree_ptr root)
2  {
3    if (root == NULL)
4      return;
5    postorder(root->left);
6    postorder(root->right);
7    printf("%d ", root->data);
8  }
```

Post-order traversal

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

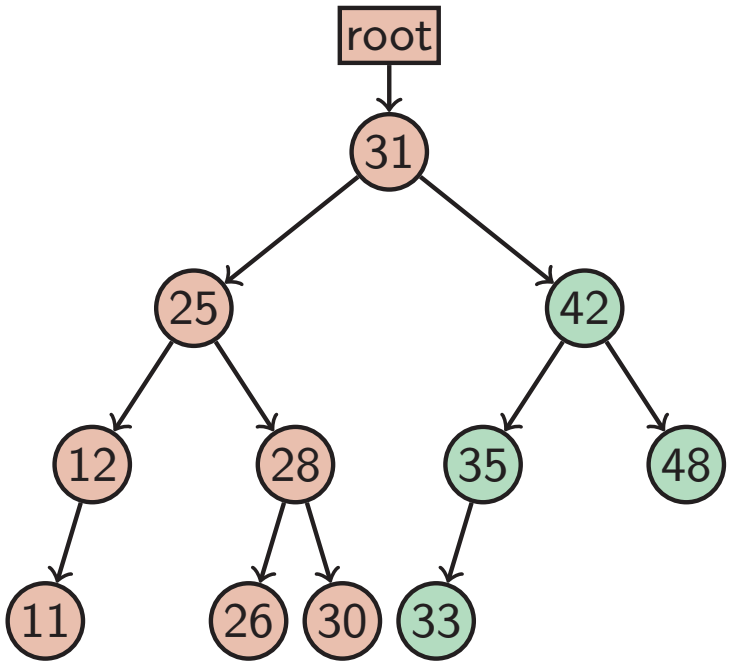



```
1  void postorder(tree_ptr root)
2  {
3    if (root == NULL)
4      return;
5    postorder(root->left);
6    postorder(root->right);
7    printf("%d ", root->data);
8  }
```

```
11 12 26 30 28 25
```

Post-order traversal

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

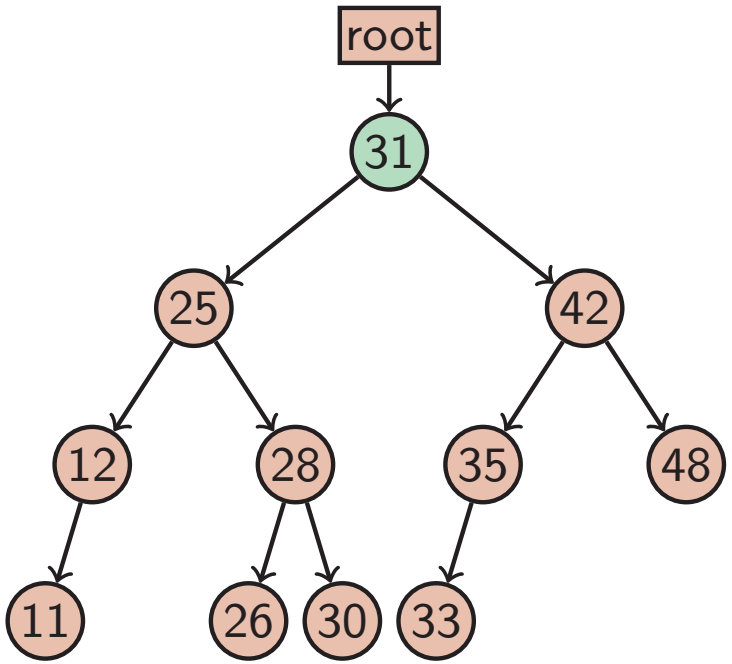



```
1  void postorder(tree_ptr root)
2  {
3    if (root == NULL)
4      return;
5    postorder(root->left);
6    postorder(root->right);
7    printf("%d ", root->data);
8  }
```

```
11 12 26 30 28 25 33 35 48 42
```

Post-order traversal

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

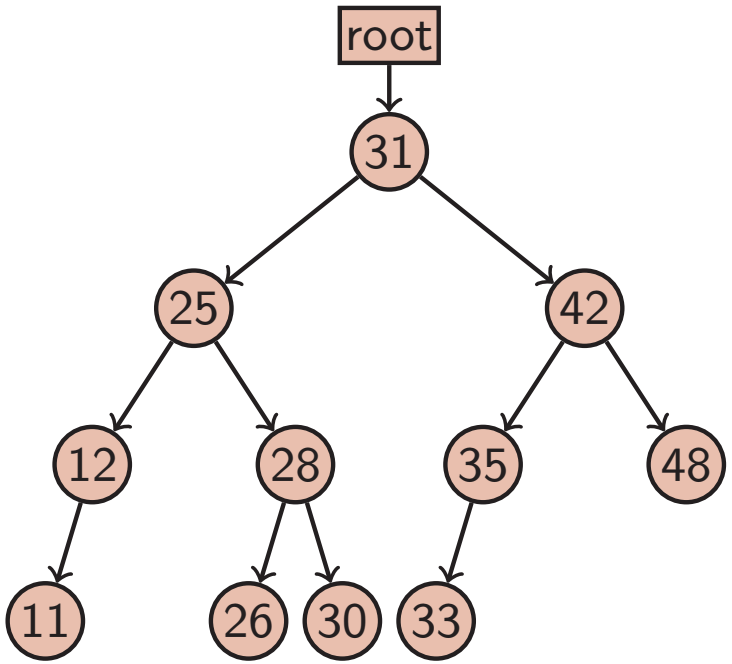



```
1  void postorder(tree_ptr root)
2  {
3    if (root == NULL)
4      return;
5    postorder(root->left);
6    postorder(root->right);
7    printf("%d ", root->data);
8  }
```

```
11 12 26 30 28 25 33 35 48 42 31
```

Post-order traversal

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



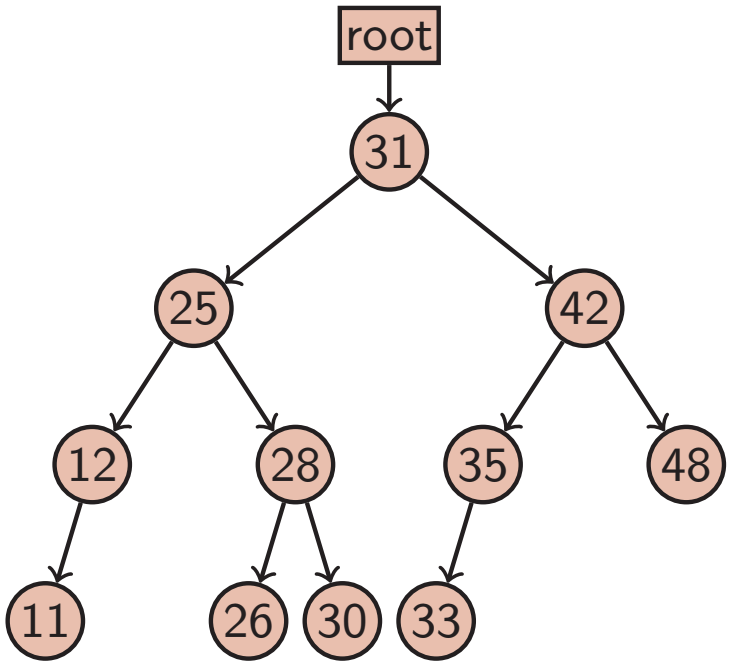

```
1  void postorder(tree_ptr root)
2  {
3    if (root == NULL)
4      return;
5    postorder(root->left);
6    postorder(root->right);
7    printf("%d ", root->data);
8  }
```

```
11 12 26 30 28 25 33 35 48 42 31
```

- post-order traversal
  1. left sub-tree
  2. right sub-tree
  3. root element

Post-order traversal

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES




```
1  void postorder(tree_ptr root)
2  {
3    if (root == NULL)
4      return;
5    postorder(root->left);
6    postorder(root->right);
7    printf("%d ", root->data);
8  }
```

```
11 12 26 30 28 25 33 35 48 42 31
```

- post-order traversal
  1. left sub-tree
  2. right sub-tree
  3. root element

In this order the leaves of the tree are visited first → application: releasing/deleting a tree

Deleting a tree by post-order traversal

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
void delete(tree_ptr root)
{
  if (root == NULL) /* empty tree: nothing to delete */
    return;
  delete(root->left);    /* post-order traversal */
  delete(root->right);
  free(root);
}
```

link

Deleting a tree by post-order traversal

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
void delete(tree_ptr root)
{
  if (root == NULL) /* empty tree: nothing to delete */
    return;
  delete(root->left);    /* post-order traversal */
  delete(root->right);
  free(root);
}
```

link

A program segment (without memory leaks):

```c
tree_ptr root = NULL;
root = insert(root, 2);
root = insert(root, 8);
...
delete(root);
root = NULL;
```

Simple algorithms on binary trees

- Write a recursive function (max. 10 lines), that

Simple algorithms on binary trees

- Write a recursive function (max. 10 lines), that
  - determines the depth of a tree

Simple algorithms on binary trees

- Write a recursive function (max. 10 lines), that
  - determines the depth of a tree
  - calculates the count / the sum / the average of the values stored in the nodes of the tree

Simple algorithms on binary trees

- Write a recursive function (max. 10 lines), that
  - determines the depth of a tree
  - calculates the count / the sum / the average of the values stored in the nodes of the tree
- Write a iterative function (max. 10 lines), that

Simple algorithms on binary trees

- Write a recursive function (max. 10 lines), that
  - determines the depth of a tree
  - calculates the count / the sum / the average of the values stored in the nodes of the tree
- Write a iterative function (max. 10 lines), that
  - computes the minimum and the maximum of the values stored in the nodes

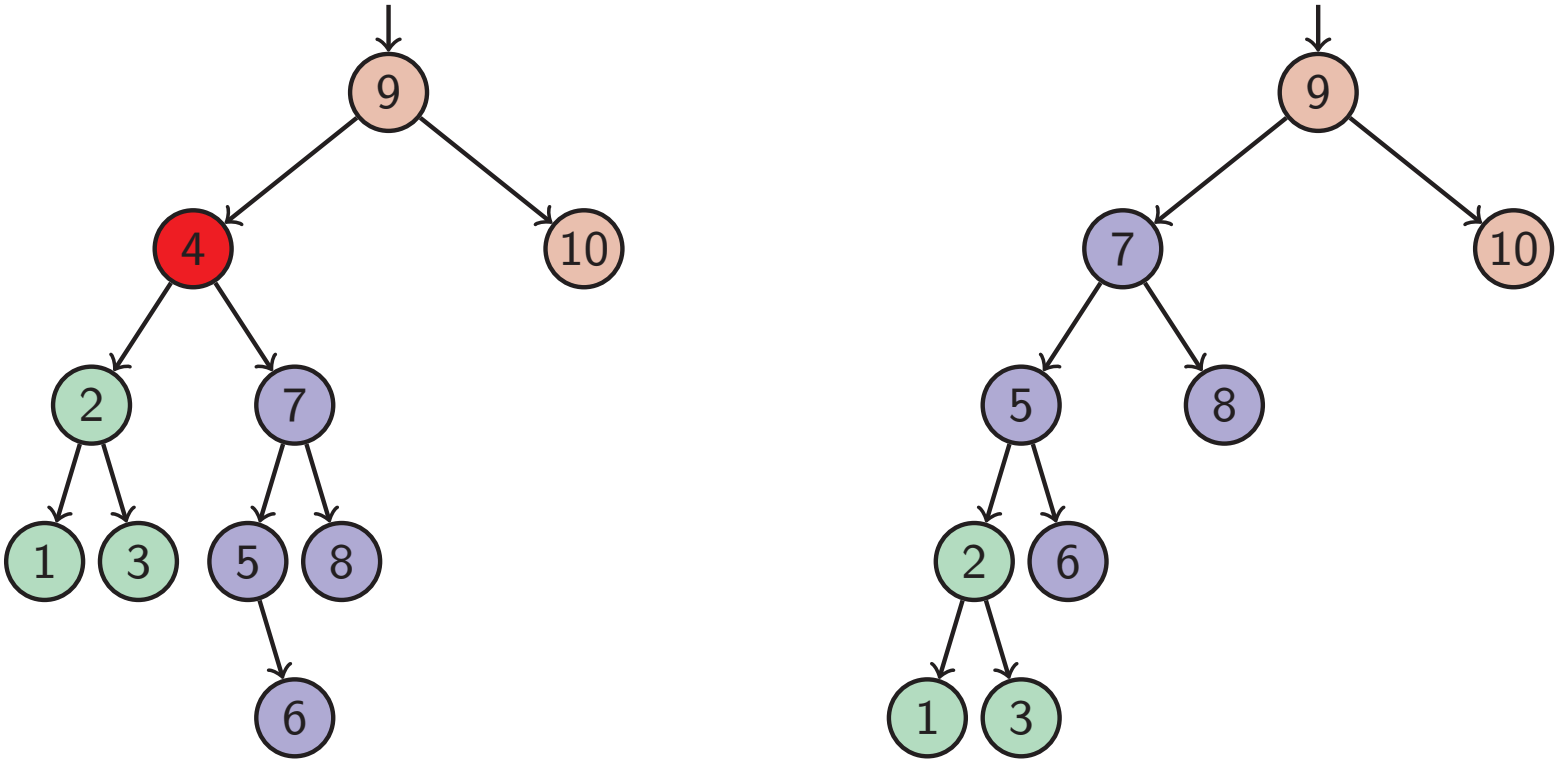
Simple algorithms on binary trees

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Write a recursive function (max. 10 lines), that
  - determines the depth of a tree
  - calculates the count / the sum / the average of the values stored in the nodes of the tree
- Write a iterative function (max. 10 lines), that
  - computes the minimum and the maximum of the values stored in the nodes
  - returns the pointer to the node storing the maximal / minimal value of the tree

Deleting an element from a search tree – naively

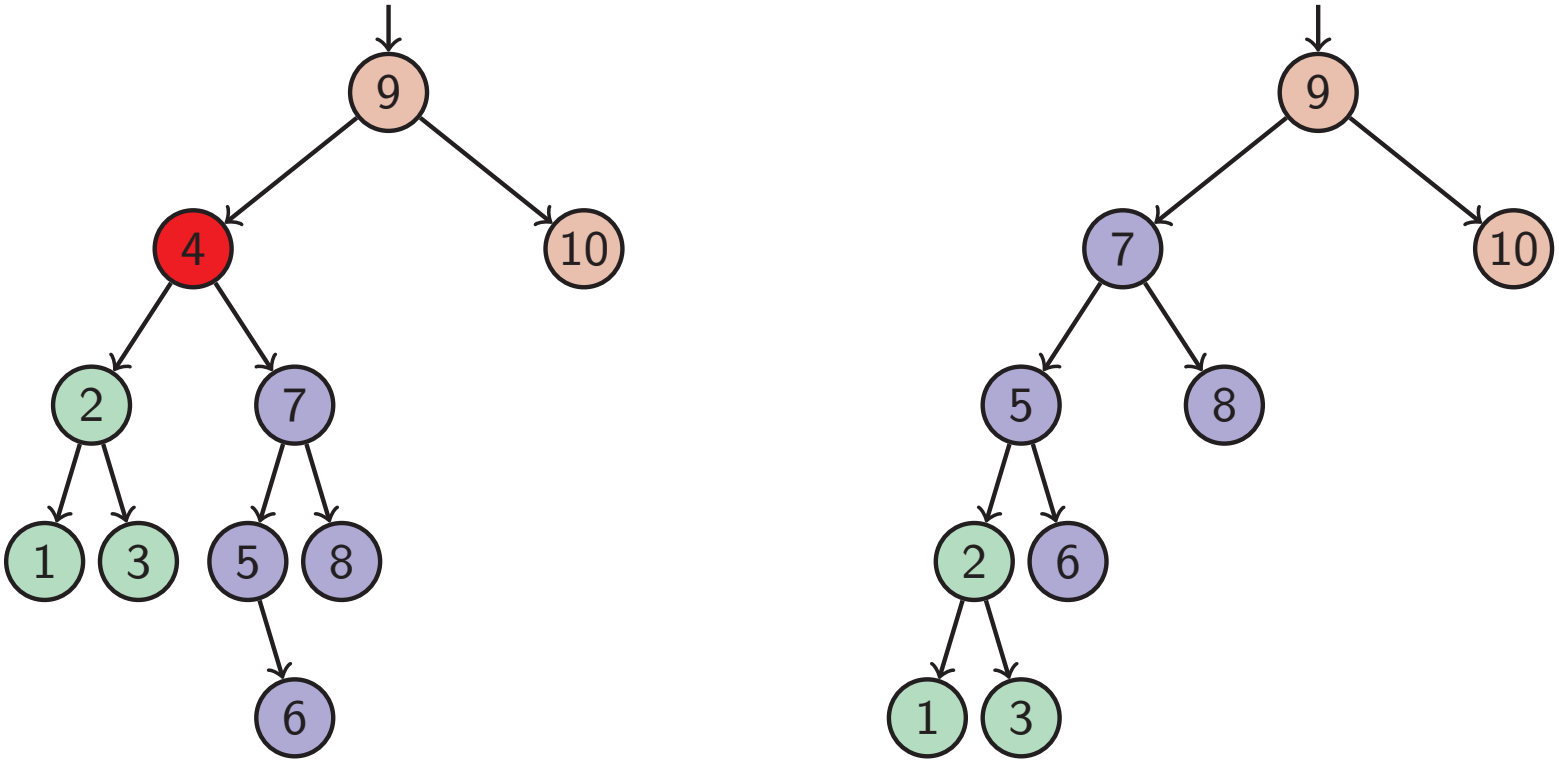
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- The right sub-tree is moved to the place of the deleted node
- The left sub-tree is inserted to below the minimal element of the right sub-tree

Deleting an element from a search tree – naively

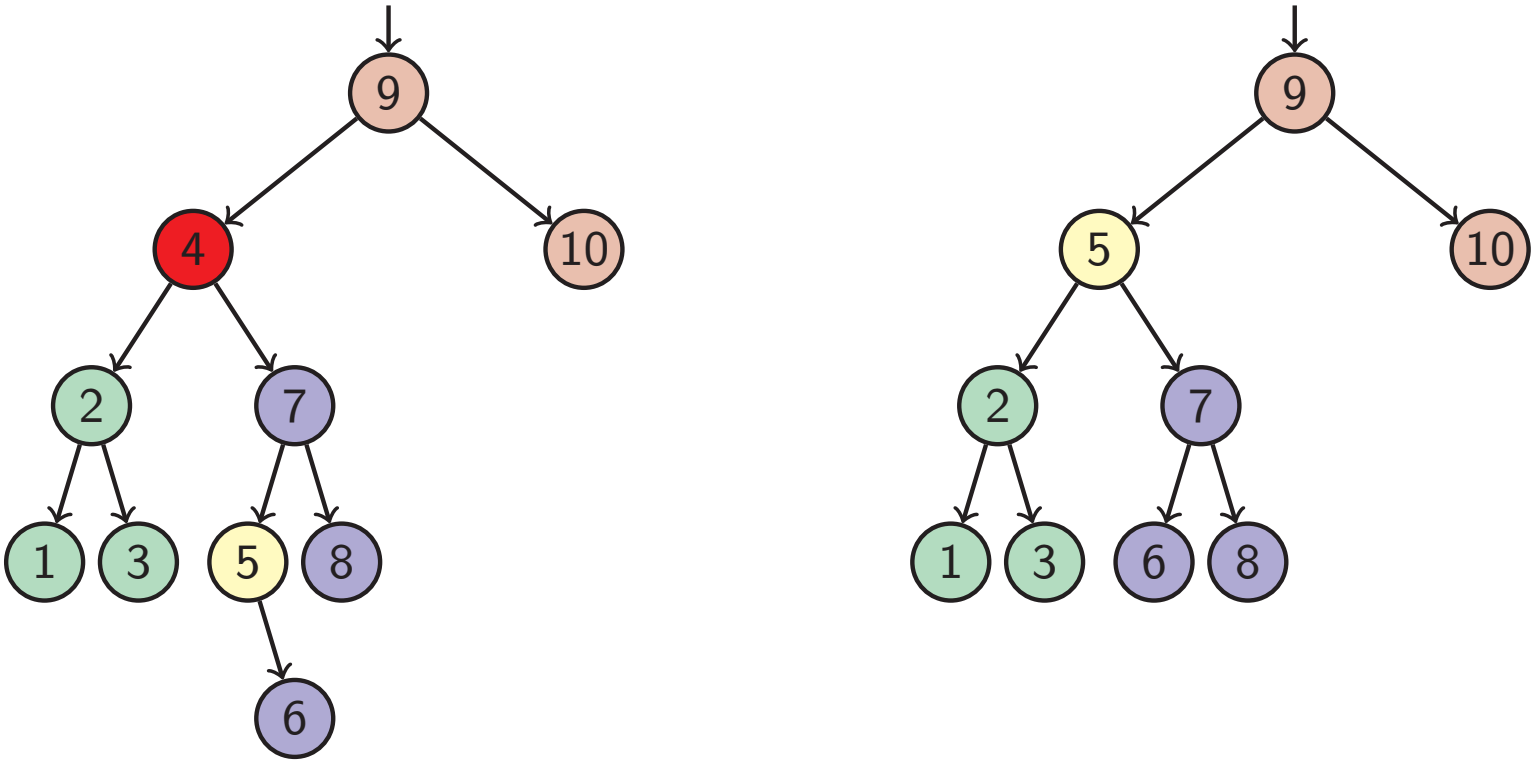
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- The right sub-tree is moved to the place of the deleted node
- The left sub-tree is inserted to below the minimal element of the right sub-tree
- The tree is getting imbalanced!

Deleting an element from a search tree – clever

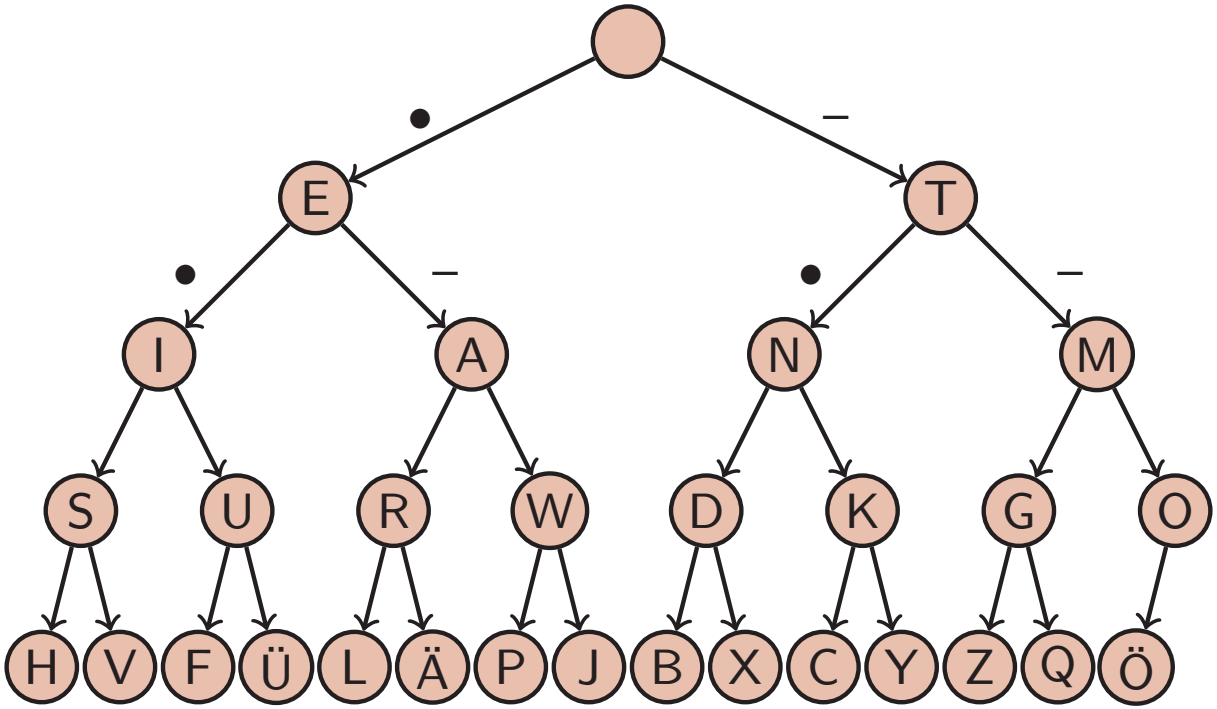
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- The minimal element of the right sub-tree is moved to the place of the deleted node
- This element could have only a right sub-tree, it is moved one level up, to its old place

Morse decoding tree

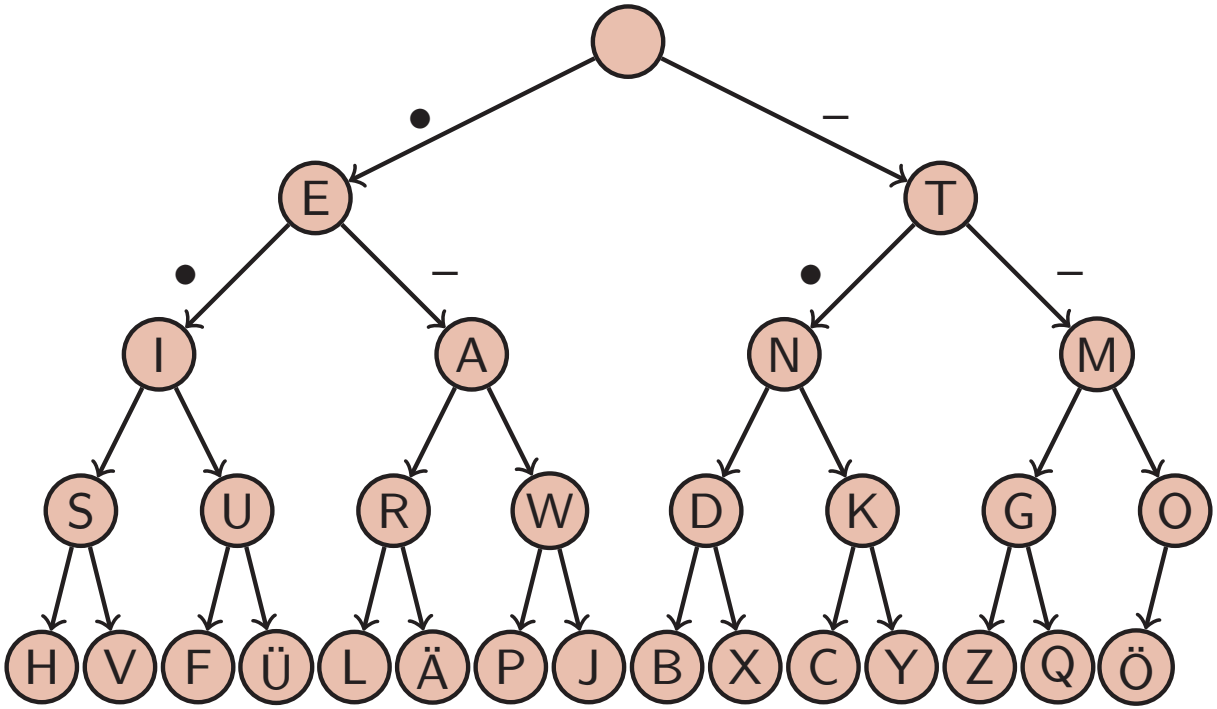
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



SOSOS: ••• – – – ••• – – – •••

Morse decoding tree

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

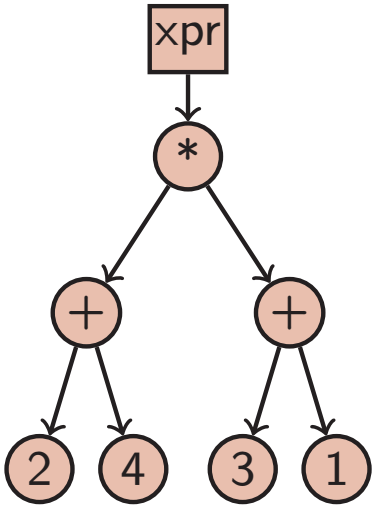


SOSOS: ●●● – – – ●●● – – – ●●●

The response: ●●●● ●– ●●●● ●– ●●●● ●–

Evaluating mathematical expressions

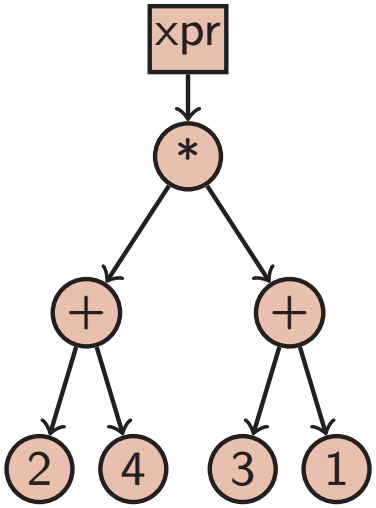
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- Storing math expressions in a tree
- Leaves $\rightarrow$ numeric constants
- Branches $\rightarrow$ two-operand operators
- In the example: $(2 + 4) * (3 + 1)$

Evaluating mathematical expressions

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- Storing math expressions in a tree
- Leaves $\rightarrow$ numeric constants
- Branches $\rightarrow$ two-operand operators
- In the example: $(2 + 4) * (3 + 1)$

```
1  int eval(tree_ptr xpr)
2  {
3    char c = xpr->data;
4    if (isdigit(c))      /* stopping condition */
5      return c - '0';
6    if (c == '+')
7      return eval(xpr->left) + eval(xpr->right);
8    if (c == '*')
9      return eval(xpr->left) * eval(xpr->right);
10 }
```

link

Evaluating functions

Let us introduce variable x as a leaf node as well:

```
1  double feval(tree_ptr xpr, double x)
2  {
3    char c = xpr->data;
4    if (isdigit(c))
5      return c - '0';
6    if (c == 'x')
7      return x;
8    if (c == '+')
9      return feval(xpr->left, x) + feval(xpr->right, x);
10   if (c == '*')
11     return feval(xpr->left, x) * feval(xpr->right, x);
12 }
```

link

Evaluating the derivative of a function

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let us take the derivative of the function! The rules are:

- $c' = 0$
- $x' = 1$
- $(f + g)' = f' + g'$
- $(f \cdot g)' = f' \cdot g + f \cdot g'$

```
1  double deval(tree_ptr xpr, double x)
2  {
3    char c = xpr->data;
4    if (isdigit(c))      /* stopping condition */
5      return 0.0;
6    if (c == 'x')      /* stopping condition */
7      return 1.0;
8    if (c == '+')
9      return deval(xpr->left, x) + deval(xpr->right, x);
10   if (c == '*')
11     return deval(xpr->left, x) * feval(xpr->right, x) +
12       feval(xpr->left, x) * deval(xpr->right, x);
13 }
```

link

Thank you for your attention.