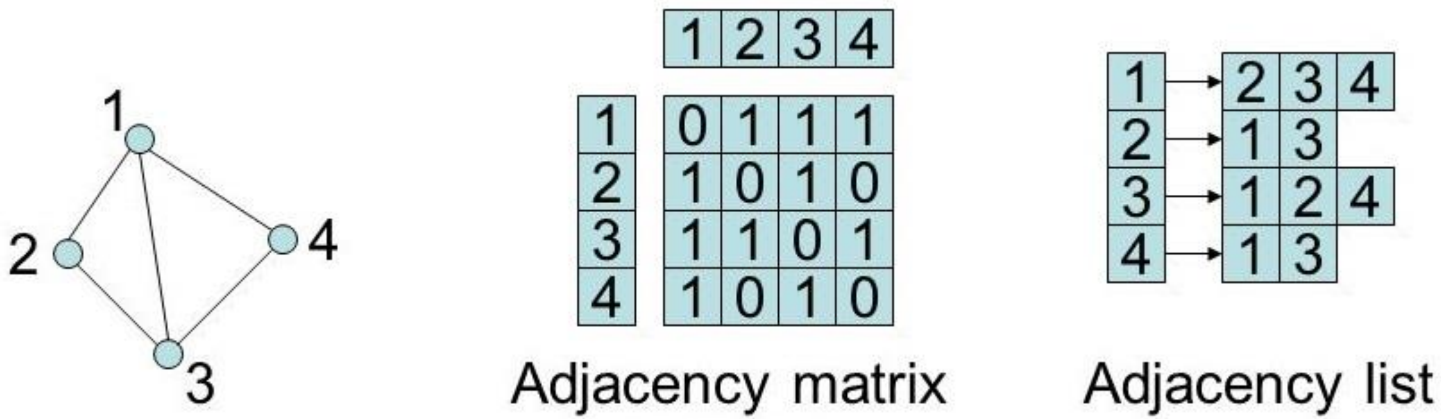
# Data Structures and algorithms – Lab 9

Iulia-Cristina Stanica
iulia.stanica@gmail.com

# Roadmap
# Graphs (part 2), Trees

- Graph implementation using linked lists
- Binary trees

# Graph implementation using linked lists



|   | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 |
| 2 | 1 | 0 | 1 | 0 |
| 3 | 1 | 1 | 0 | 1 |
| 4 | 1 | 0 | 1 | 0 |

Adjacency matrix

|   |   |   |   |
|---|---|---|---|
| 1 → | 2 | 3 | 4 |
| 2 → | 1 | 3 |   |
| 3 → | 1 | 2 | 4 |
| 4 → | 1 | 3 |   |

Adjacency list

Graph Algorithms

- Uses an array of lists! (a list for each node)
- The functions add and remove from the LinkedList class are used for the functions addEdge and removeEdge.

# Exercise 1

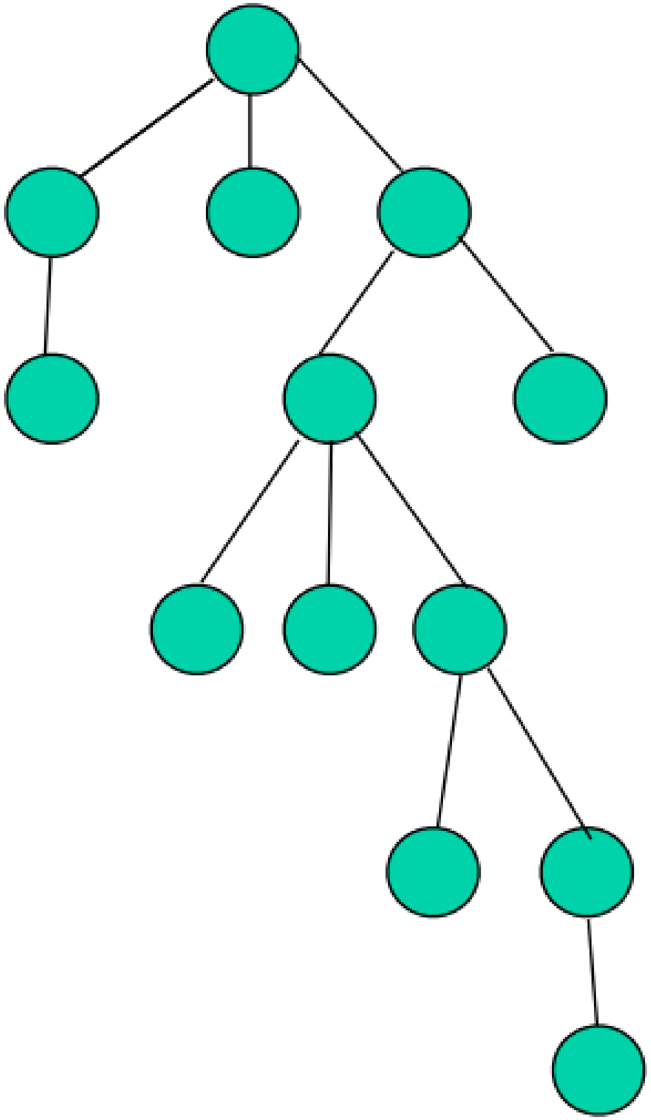
- Add the missing lines in the removeEdge function. Test its effect.

# Exercise 2

- Check if a graph is Hamiltonian or not by using this linked list implementation.

# Trees

- **Def**: a data structure that simulates a hierarchical tree structure, with a root value and subtrees of children with a parent node, represented as a set of linked nodes.
- A connected graph where we have just one path connecting each pair of nodes i and j.

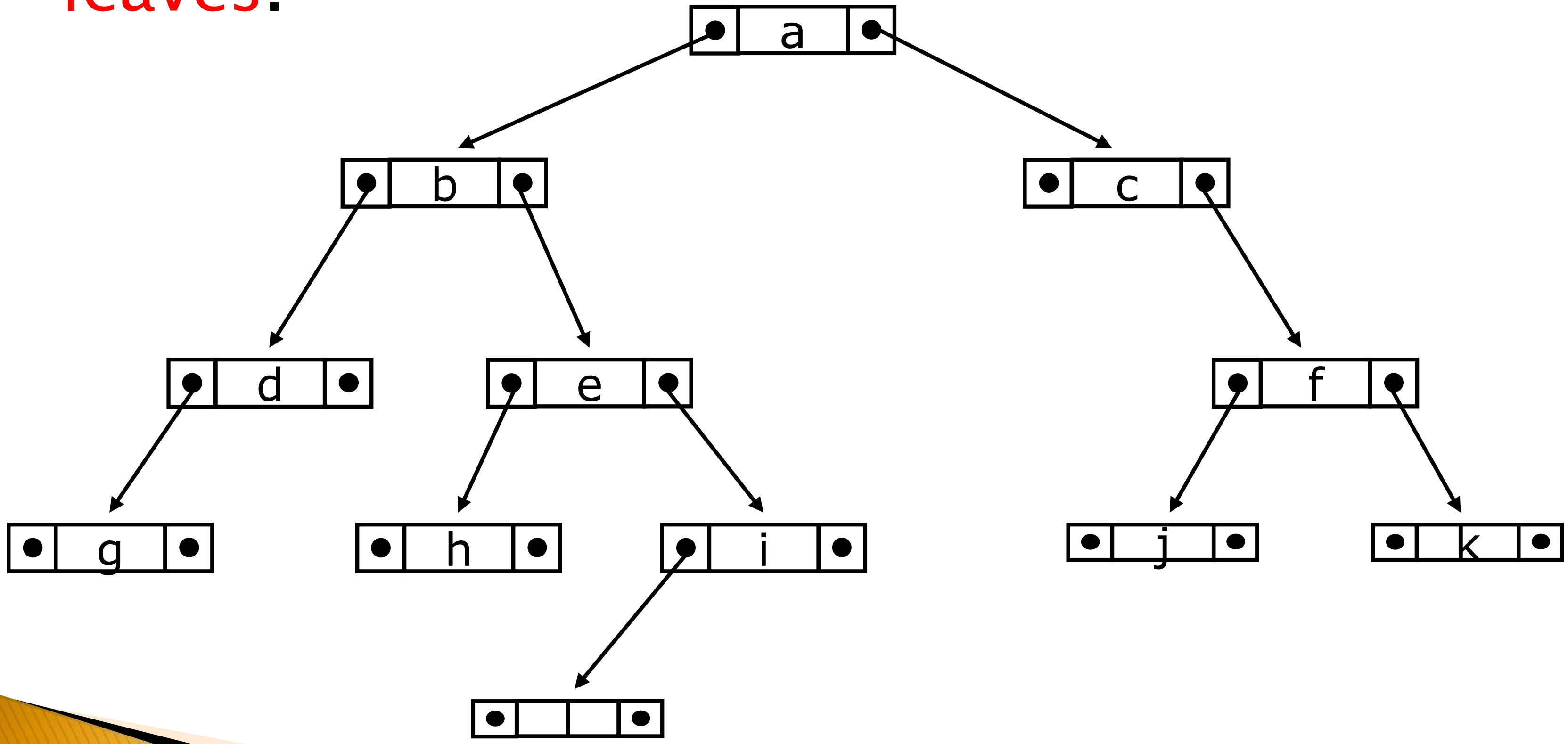


# Binary trees

- **Def:** A binary tree is composed of zero or more nodes, but each node has maximum 2 descendants: a left one and a right one.
- Each node has:
  - A value (of a certain type)
  - A pointer to the left child (can be NULL)
  - A pointer to the right child (can be NULL)

# Binary trees

- If it is not empty, the binary tree has a root node.
- The nodes which have no children are called leaves.



# Representation

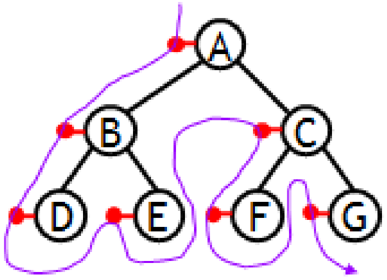
```
template <typename T> class BinaryTree
 {
    public:
     BinaryTree();
     ~BinaryTree();
    private:
     BinaryTree<T> *leftNode;
     BinaryTree<T> *rightNode;
     T *pData;
};
```

For the nodes and the data, memory is allocated dynamically, but not in the constructor!

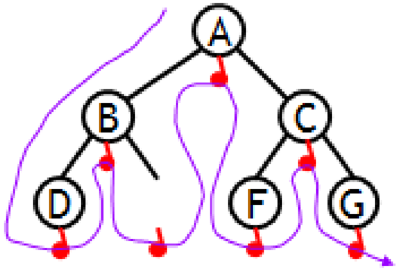
```
BinaryTree<T> *node = new BinaryTree<T>(); delete node;
T *pData = new T; delete pData;
```

# Tree traversals

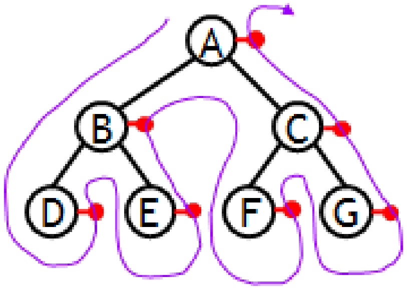
- In preorder, the root is visited *first:* root, left, right
- In inorder, the root is visited *in the middle:* left, root, right
- In postorder, the root is visited *last:* left, right, root



A B D E C F G

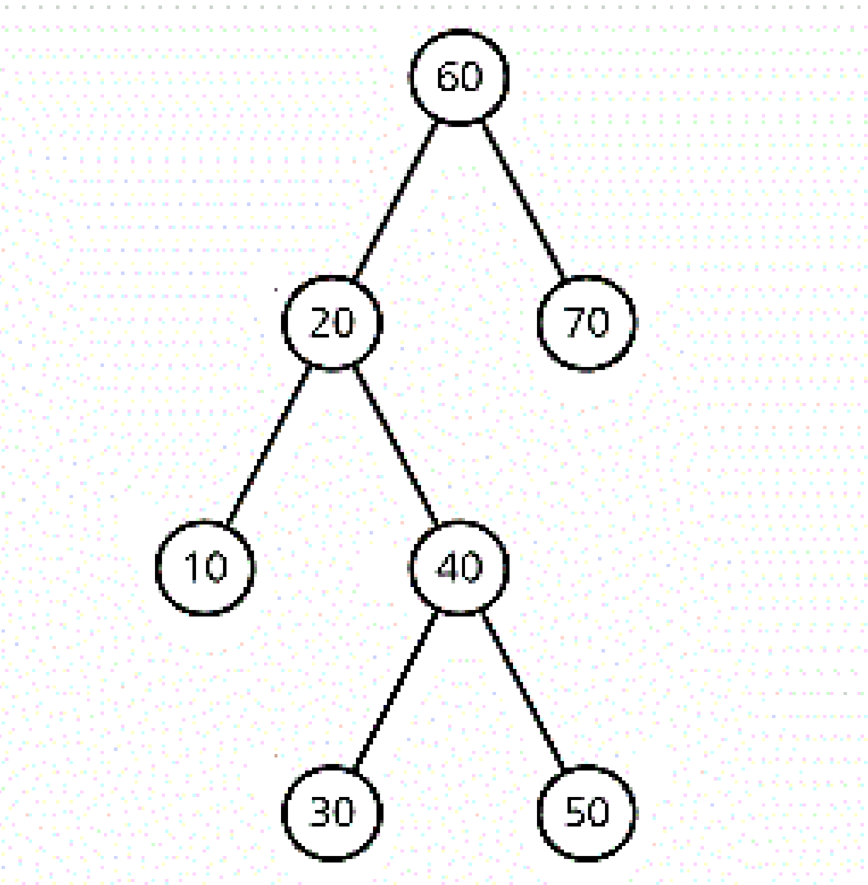


D B E A F C G



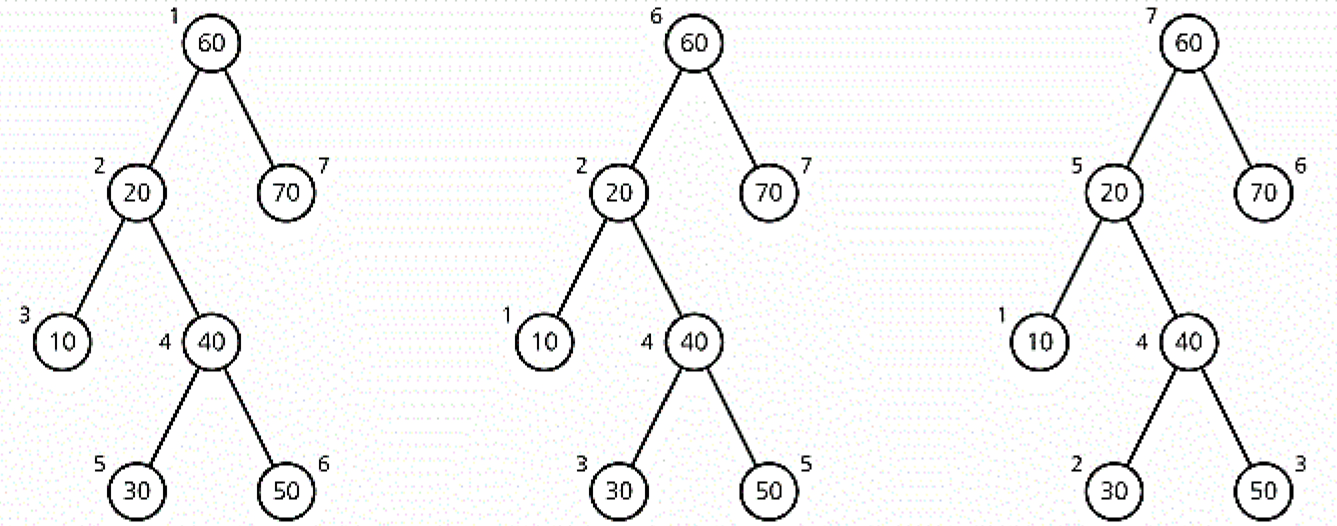
D E B F G C A

# Tree traversals



Hint: apply the algorithm recursively

# Tree traversals



(a) Preorder: 60, 20, 10, 40, 30, 50, 70 (b) Inorder: 10, 20, 30, 40, 50, 60, 70 (c) Postorder: 10, 30, 50, 40, 20, 70, 60

(Numbers beside nodes indicate traversal order.)

Pre In Post

# Homework

- Choose a method from the BinaryTree class and explain its implementation.
- Add a function which calculates the number of levels of the tree.