Type system
Abstract data types

Course: Object Oriented Programming (OOP)

CTU, FS, U12110
Matouš Cejnek

Contents

- Type system
- Abstract data types
  - Queue
  - Stack
  - Tree
  - Heap

What do we already know?

A **data type**, is a specification of a variable and what type of mathematical, relational or logical operations can be applied to it without causing an error.

Type system

- Type system is a logical system comprising a set of rules that assigns a property called a type to every object.
- Type system is commonly a part of programming language and it is built into interpreters and compilers.

Type safety

- Type safety and type soundness are the extent to which a programming language discourages or prevents type errors
- Type enforcement can be **static**, catching potential errors at compile time, or **dynamic**, associating type information with values at run-time

CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Dynamic typing vs static typing

Dynamically-typed languages perform type checking at runtime, while statically typed languages perform type checking at compile time.

Static typing	**Dynamic typing**
Errors are catched sooner	Less code
	Less effort / time

Static typing

Example languages: C, C++, Java, Rust, Go, Scala

- Static typing is used mainly by languages used for big, complex projects (difficult to debug and time consuming anyways)

Dynamic typing

Example languages: Perl, Ruby, Python, PHP, JavaScript, Erlang

- Dynamic typing is mainly popular among script language (small scripts are easy to debug and they can be created in short time)

Type system examples

Python

```
a = 1
a = 'abc'
```

Reference is retyped.

C++

```
int a = 1;
```

Type of **a** cannot be changed.

Type safety examples

Python

```
a = 1 + ’abc’
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

C++

```
int a = "abc";
```

```
error: invalid conversion from ‘const char*’ to ‘int’
[-fpermissive]
```

Popular type systems

There are many specific type systems. Few popular:

- Duck typing
- Nominal type system
- Structural type system

Keep in mind, many languages combine more type systems.

Nominal type system

In nominative (name-based) type system, the compatibility and equivalence of data types is determined by explicit declarations and/or the name of the types.

Duck typing

“If it walks like a duck, and it quacks like a duck, then it must be a duck.”

Duck typing is a concept related to **dynamic typing**.

When you use duck typing, you do not check types at all. Instead, you check for the presence of a given method or attribute.

Structural type system

Structural type system determines compatibility and equivalence by the type's actual structure or definition.

It is related to **static typing.**

Abstract data types

Linked list

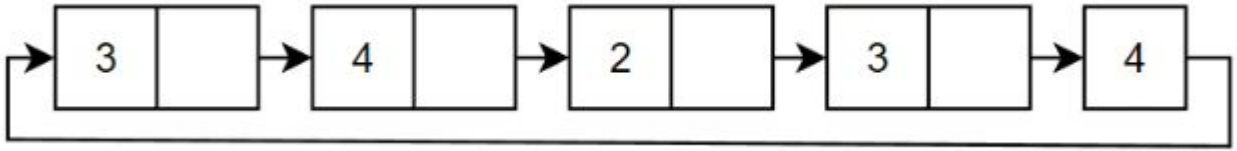
- Each element points to the next.
- Order is not given by their physical placement in memory.

Singly linked list:

CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Linked list

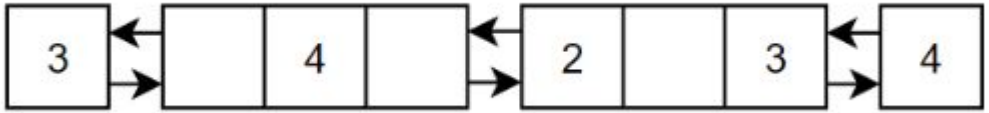
Circular linked list:



CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

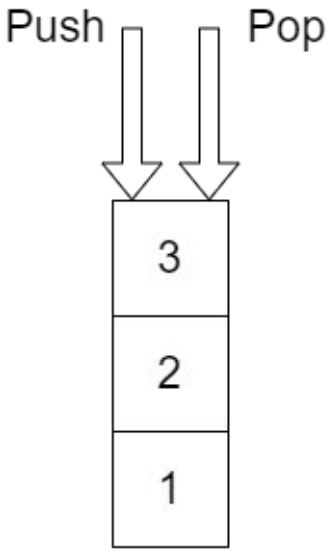
Linked list

Doubly linked list:

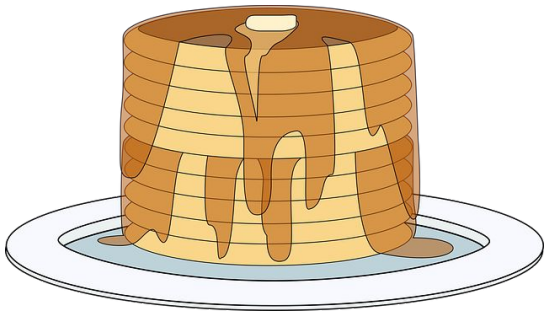


Stack (LIFO)

Stack is also called LIFO (last in, first out).

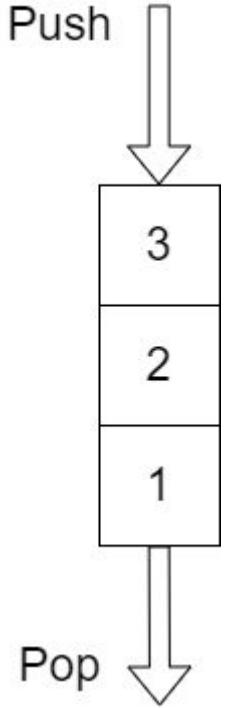


Stack of pancakes (source: pixabay)



Queue (FIFO)

Queue is also called first-in-first-out (FIFO).



Queue of cars (source: pixabay)

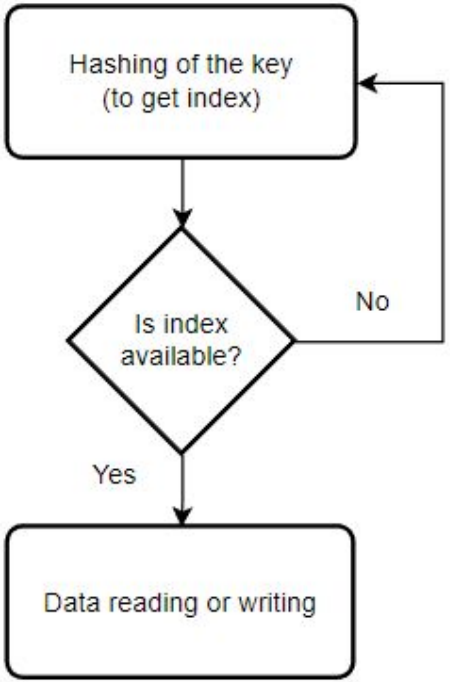
Circular queue

A Circular queue is a data structure that uses a single, fixed-size queue as if it were connected end-to-end.

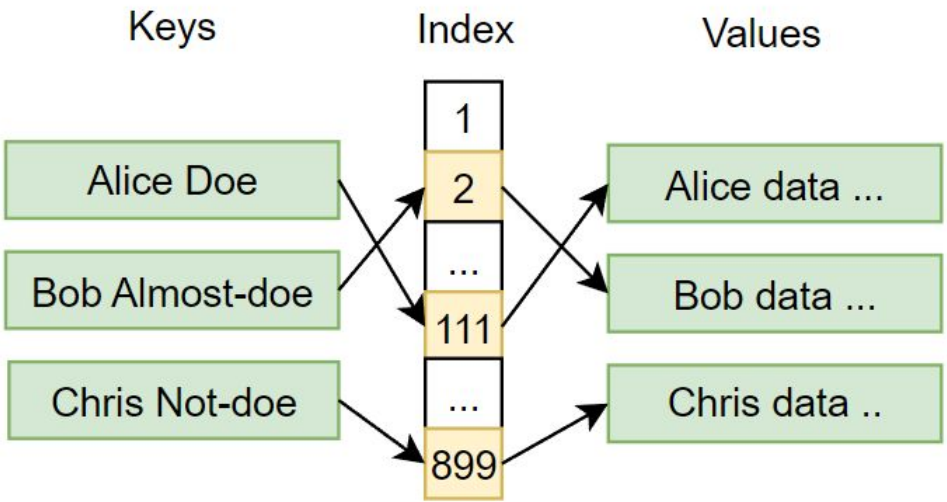
Also known as: circular queue, cyclic buffer or ring buffer

Hash table

- Index is obtained from keys via a hash function.
- Similar keys should produce different indexes.
- Keys are not stored, only index and data.



Hash table example



Tree-like abstract data types

- Tree
- Binary tree
- AVL tree
- Heap

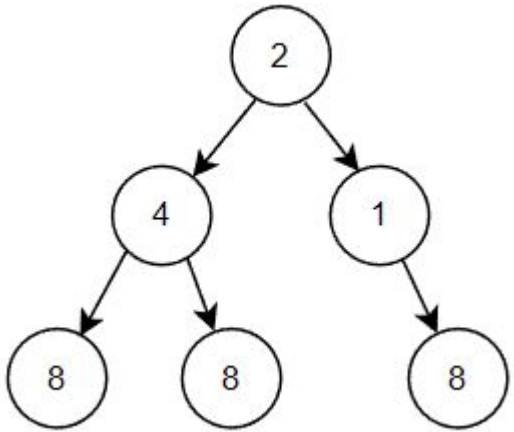
Tree

Tree is a widely used abstract data type that represents a hierarchical tree structure with a set of connected nodes.

Popular usage:

- Document Object Models (DOM tree) of XML and HTML
- File systems / Directory structures
- Natural language processing

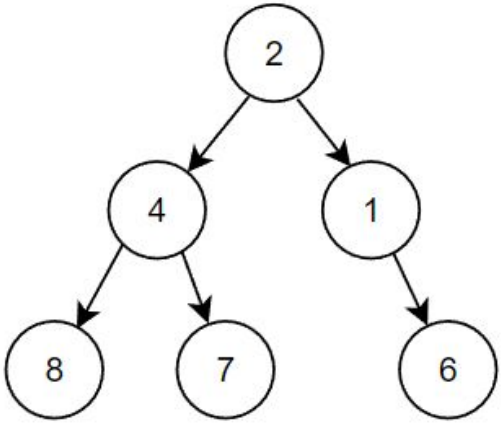
Tree terminology

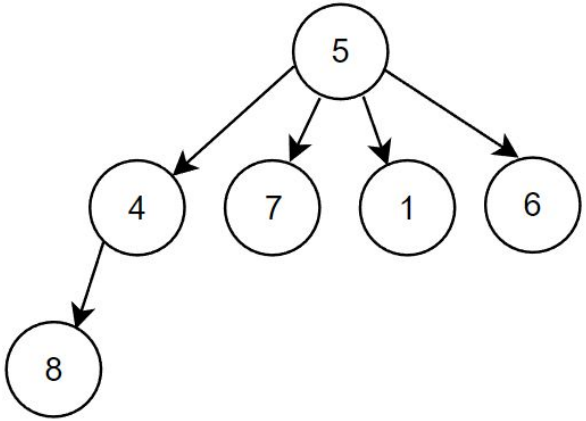


Nodes are connected with **edge**s. Every node has 0 or more **children**, and 0 or 1 **parent**. The node without parent is called **root**. Terminal nodes are called **leaf** nodes.

CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

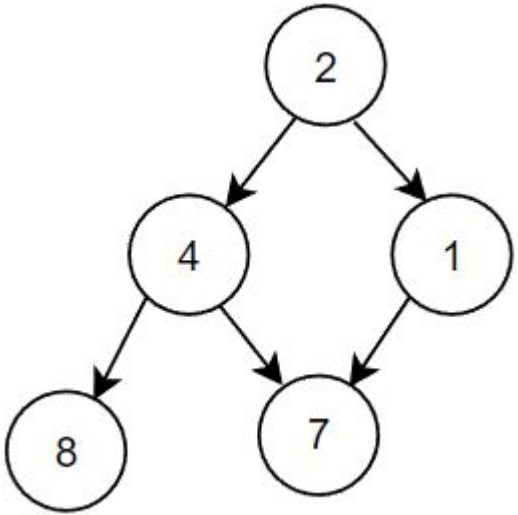
Tree examples





Non-tree example

Multiple parents!



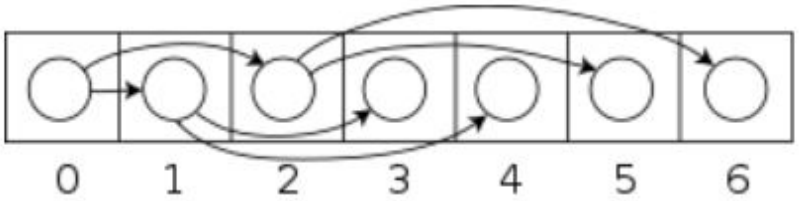
CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Tree features

- **Breadth** - number of leaves
- **Depth** - number of levels
- **Ordered / Unordered**
- **Binary tree** - tree with only two children per node

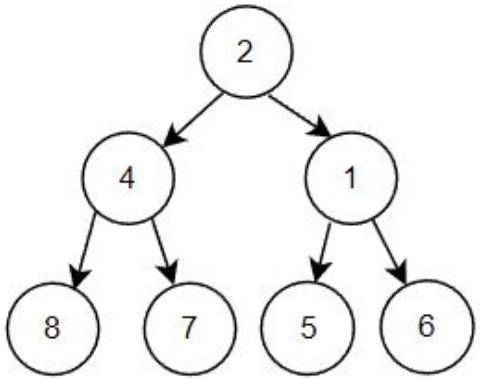
Binary tree

- **A full binary tree** - every node has two children
- **A complete binary tree** - every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

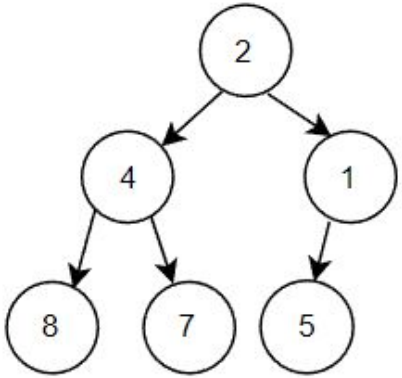


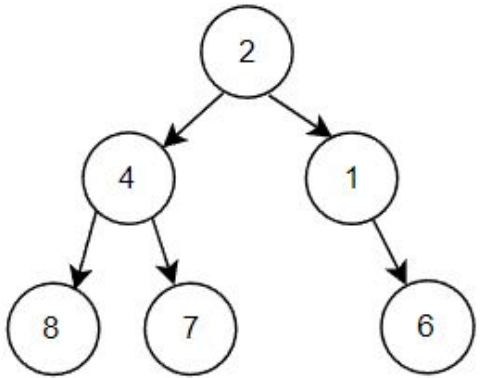
Binary tree

Full, complete



Complete





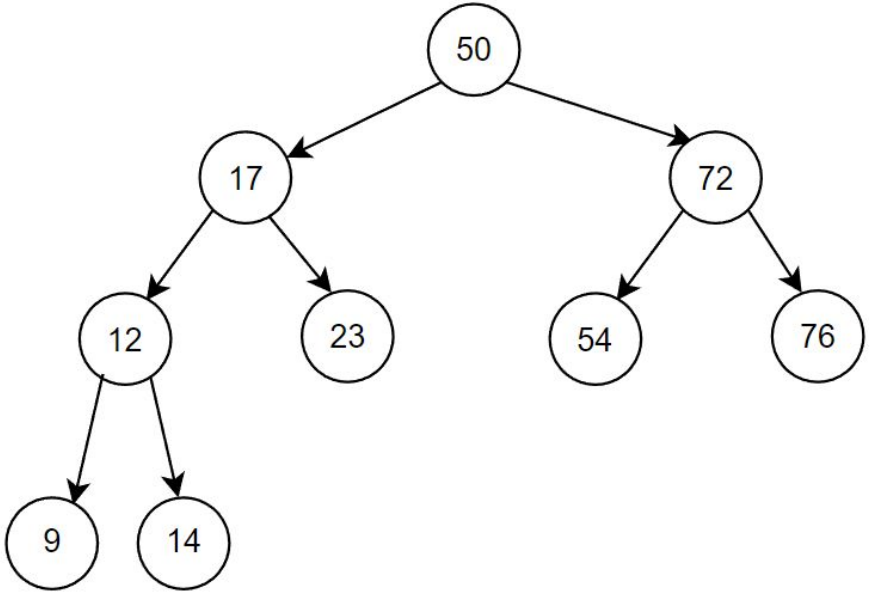
CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

AVL tree

- It is named after inventors Adelson-Velsky and Landis (1962).
- It is a self-balancing binary search tree.
- It is binary tree.
- Left child < right child
- abs(height(**Left subtree**) - (height(**right subtree**)) <= 1

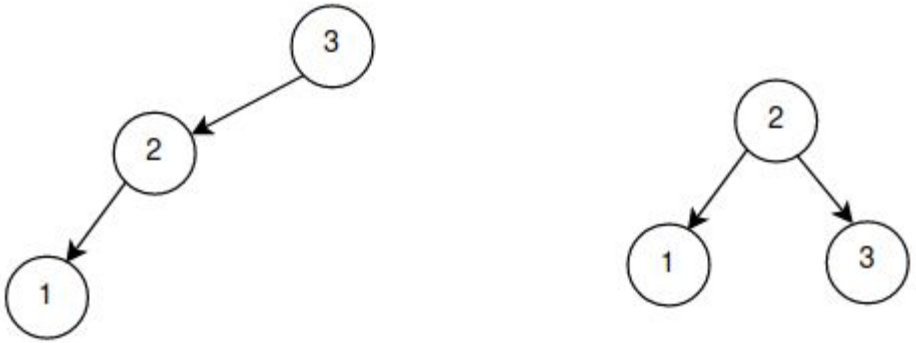
CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

AVL tree example



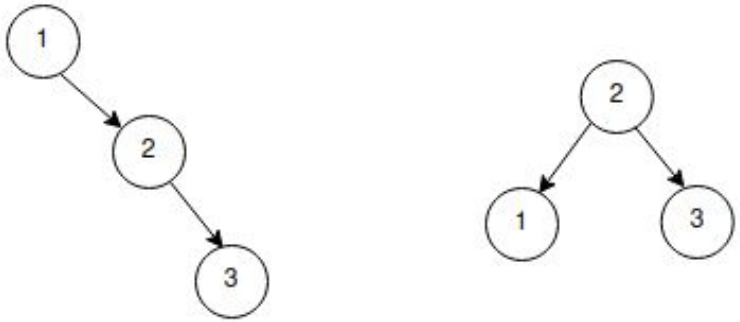
AVL tree - R rotation

Insertion of 3, 2, 1 leads to LL imbalance => R-rotation:



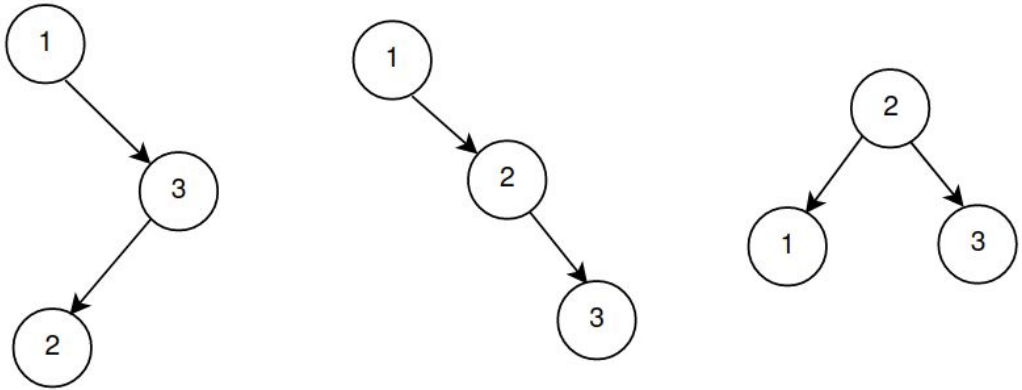
AVL tree - L rotation

Insertion of 1, 2, 3 leads to RR imbalance => L-rotation:



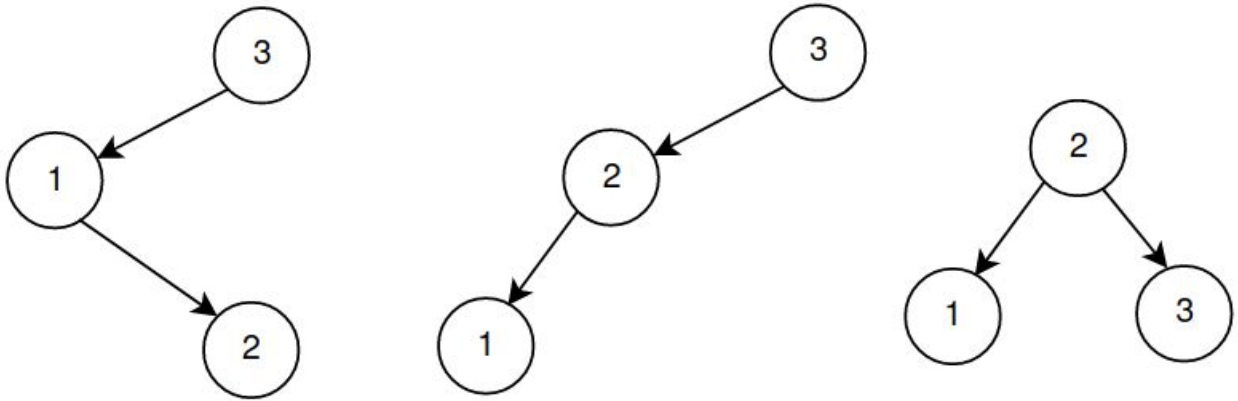
AVL tree - RL rotation

Insertion of 1, 3, 2 leads to RL imbalance => RL-rotation:



AVL tree - LR rotation

Insertion of 3, 1, 2 leads to LR imbalance => LR-rotation:



Heap

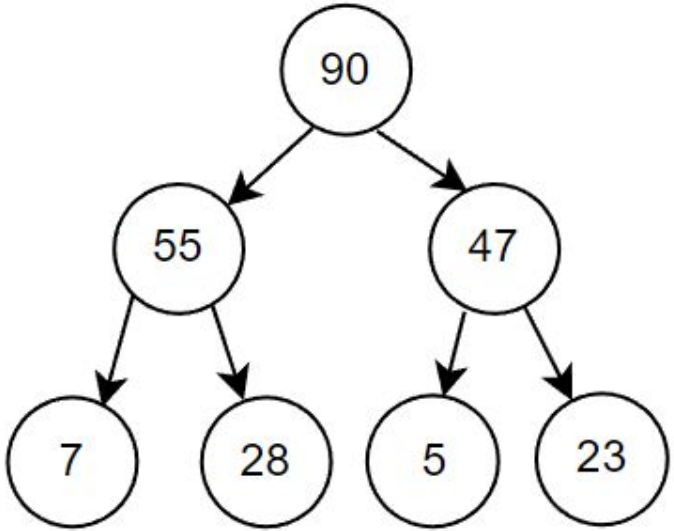
(Binary) Heap is an **complete** and **ordered** binary tree.

According to order we recognize:

- Min heap
- Max heap

CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Heap

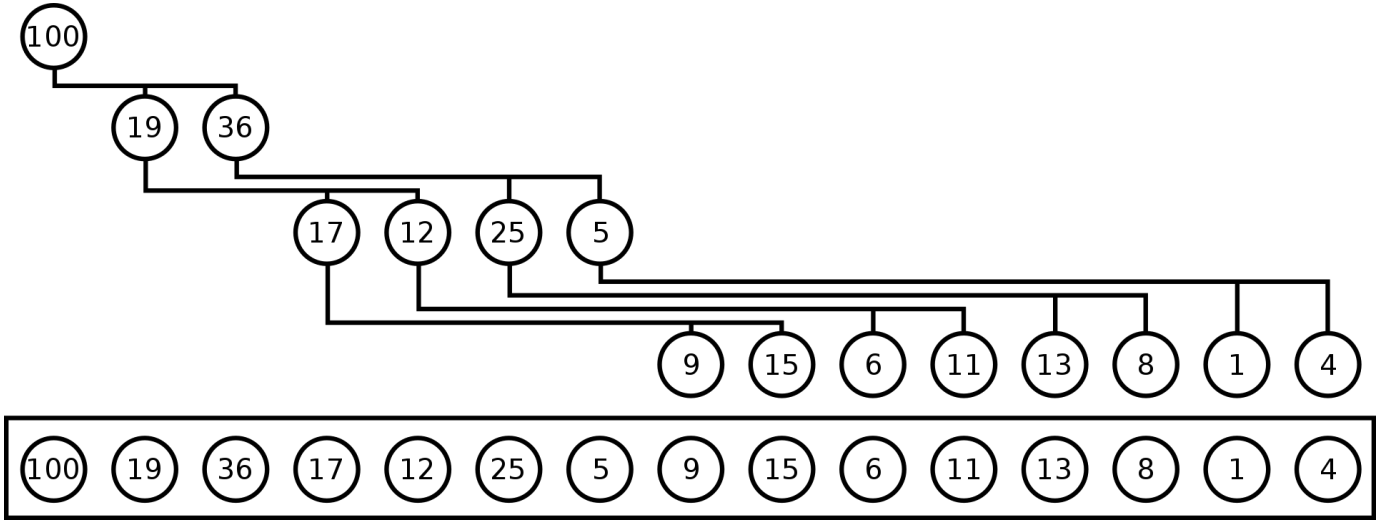


Heap

Heap is used for:

- Sorting (heap sort)
- Priority queue
- Dijkstra’s algorithm (internet routing)

Heap array implementation



CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

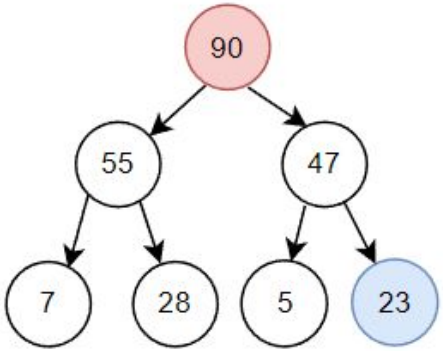
Heap - root removal

Root removal is one of the most basic operations.

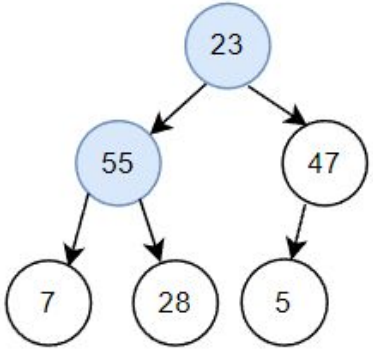
1. Replace root with the most right leave (to keep the heap complete)
2. Sift down the new root till placed correctly (heap ordered)

Heap - root removal

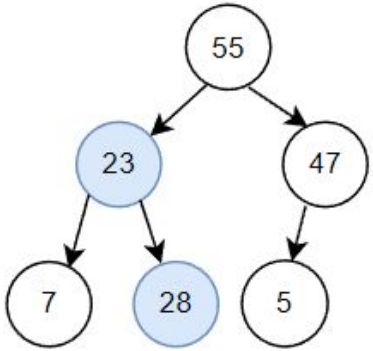
1. Remove root and replace with the last leaf



2. Sift down



3. Sift down



New valid heap