CMPE 180A DSA with Python

Trees: Priority Queues, Heaps, Binary Search Trees
Project introduction, Lab

Sanja Damjanovic **Week 5** **8/12/2025**

Week 5 Outline

- At 5.30pm Midterm (90 minutes)
- At 7pm: Lecture
- Trees: Priority queues, Heaps, Binary Search Trees
- Project introduction
- Scientific Python packages
  - NumPy
  - Matplotlib
  - Pandas
  - SciPy

CMPE180A Overview

- Weeks 5, 6, 7: HW 2, 3, 4
- Week 7: Project: selection of the topic and dataset
- Weeks 8, 9, 10: work on project
- Week 10: the final
- Week 11: project presentations, everyone will present

Term Project

Project

- Project assignment will be published on Canvas
    - Exploratory data set analysis + prediction/classification
- Project groups:
    - Randomly generated
    - Group: 2-3 students (5-6 groups)

Project Data Sets - Examples

- NYC Yellow Taxi Trip Data (https://www.kaggle.com/datasets/elemento/nyc-yellow-taxi-trip-data)
- IMDB Dataset of 50K Movie Reviews (https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews)
- Superstore Dataset (https://www.kaggle.com/datasets/vivek468/superstore-dataset-final)
- Spotify Tracks DB (https://www.kaggle.com/datasets/zaheenhamidani/ultimate-spotify-tracks-db)

Project Deliverables and Deadlines

- Week 7/8: Select a dataset you will be working on (check the project notebook)
- Week 11:
  - Upload project code and slides; write documentation directly in notebook by using text sections
  - Present results in ~ 10 minutes with 5-10 slides:
  - Slides:
    - 1st: The title slide should contain the title of your project and the names of the team members
    - 2nd slide: introduction with the dataset and list of the done
    - 3rd: dataset description
    - 4th-nth: steps in analysis and results
    - the last slide: conclusion and the overview of the presented (~ similar to the 2nd slide), and recommendations

Scientific Python

NumPy

Documentation

- numpy.org
- fundamental package for scientific computing in Python
- a Python library that provides:
    - a multidimensional array object
    - various derived objects (such as masked arrays and matrices)
    - routines for fast operations on arrays including:
        - mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more

NumPy arrays

Python objects:	• high-level number objects: integers, floating point • containers: lists (costless insertion and append), dictionaries (fast lookup)
NumPy provides:	• extension package to Python for multi-dimensional arrays • closer to hardware (efficiency) • designed for scientific computation (convenience) • Also known as *array oriented computing*

NumPy

NumPy

- NumPy reference on numpy.org
- Reading:
- https://lectures.scientific-python.org/intro/numpy/index.html
- 1.3. NumPy: creating and manipulating numerical data:
  - **1.3.1. The NumPy array object**
  - **1.3.2. Numerical operations on arrays**
  - **1.3.3. More elaborate arrays**
  - Reading

Matplotlib

Visualization with Python

- https://matplotlib.org/
- Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python
- *pyplot* provides a procedural interface to the matplotlib object-oriented plotting library. It is modeled closely after Matlab™
- Matplotlib tutorial: https://lectures.scientific-python.org/intro/matplotlib/index.html
  - Reading

Pandas

https://pandas.pydata.org/

- a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language
- contains data structures and data manipulation tools designed to make data cleaning and analysis fast and convenient in Python
- used in tandem with numerical computing tools like NumPy and SciPy etc.

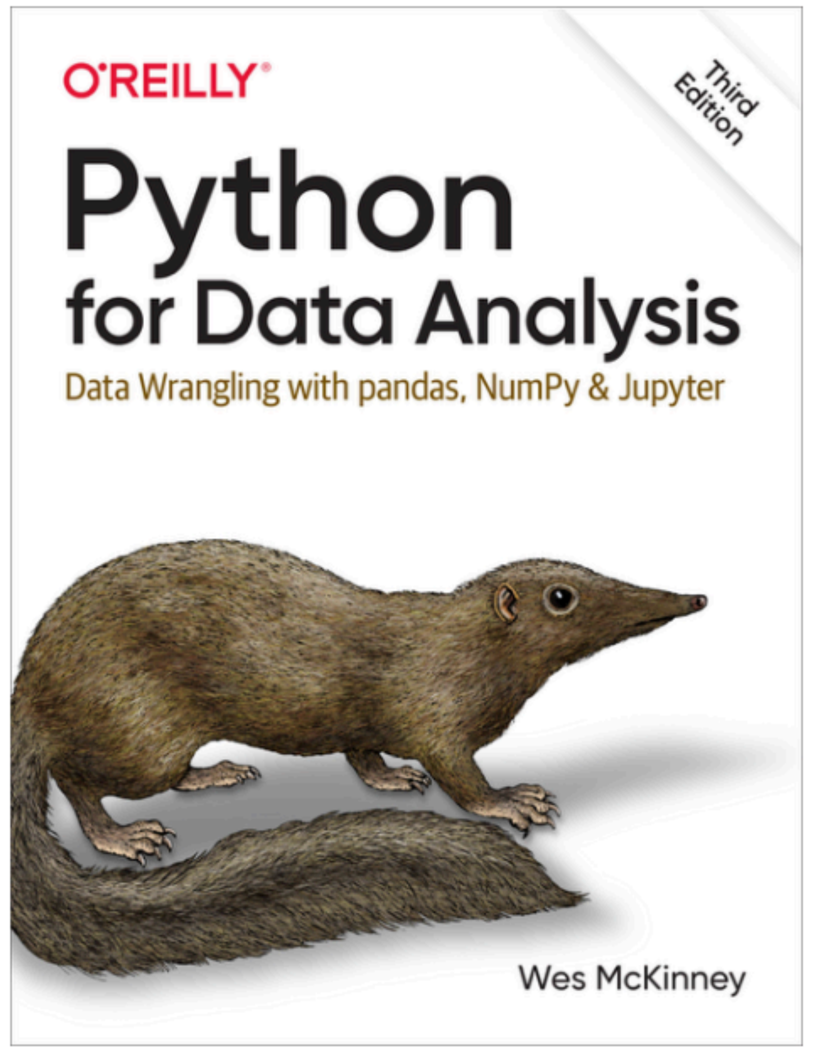
Pandas: Data Structures

- Series
  - a one-dimensional array-like object containing a sequence of values with labels
- DataFrames
  - a rectangular table of data and contains an ordered, named collection of columns, each of which can be a different value type (numeric, string, Boolean, etc.)
  - has both a row and column index

Pandas

Resources

- Reading
- Book chapter Chapter 5. Getting Started with pandas
- https://pandas.pydata.org/getting_started.html
- Pandas cheat sheet
- https://wesmckinney.com/book/



SciPy

https://scipy.org/

- SciPy: high-level scientific computing
- SciPy provides algorithms for optimization, integration, interpolation, eigenvalue problems, algebraic equations, differential equations, statistics and many other classes of problems.
- SciPy tutorial : 1.5.1, 1.5.2, 1.5.3, 1.5.4, 1.5.5, 1.5.6
  - Reading
- User Guide

Video Tutorials

https://www.youtube.com/@freecodecamp

- Pandas & Python for Data Analysis by Example – Full Course for Beginners https://youtu.be/gtjxAH8uaP0?si=EUuvel3MiXWOvS0G
- Data Analysis with Python - Full Course for Beginners (Numpy, Pandas, Matplotlib, Seaborn) https://youtu.be/r-uOLxNrNk8?si=XgUNnjTtWLlI4p4X
- Python NumPy Tutorial for Beginners https://youtu.be/QUT1VHiLmmI?si=PQr8uv60OblIhJy6
- Data Analysis with Python: Part 3 of 6 Numerical Computing with Numpy (Live Course) https://www.youtube.com/live/NlZXAytUeeE?si=8IEcOamBSV51EmRZ
- Matplotlib Crash Course https://youtu.be/3Xc3CA655Y4?si=u_6gkytDp2yKyYPr

Heaps, Priority Queues, Binary Search Trees

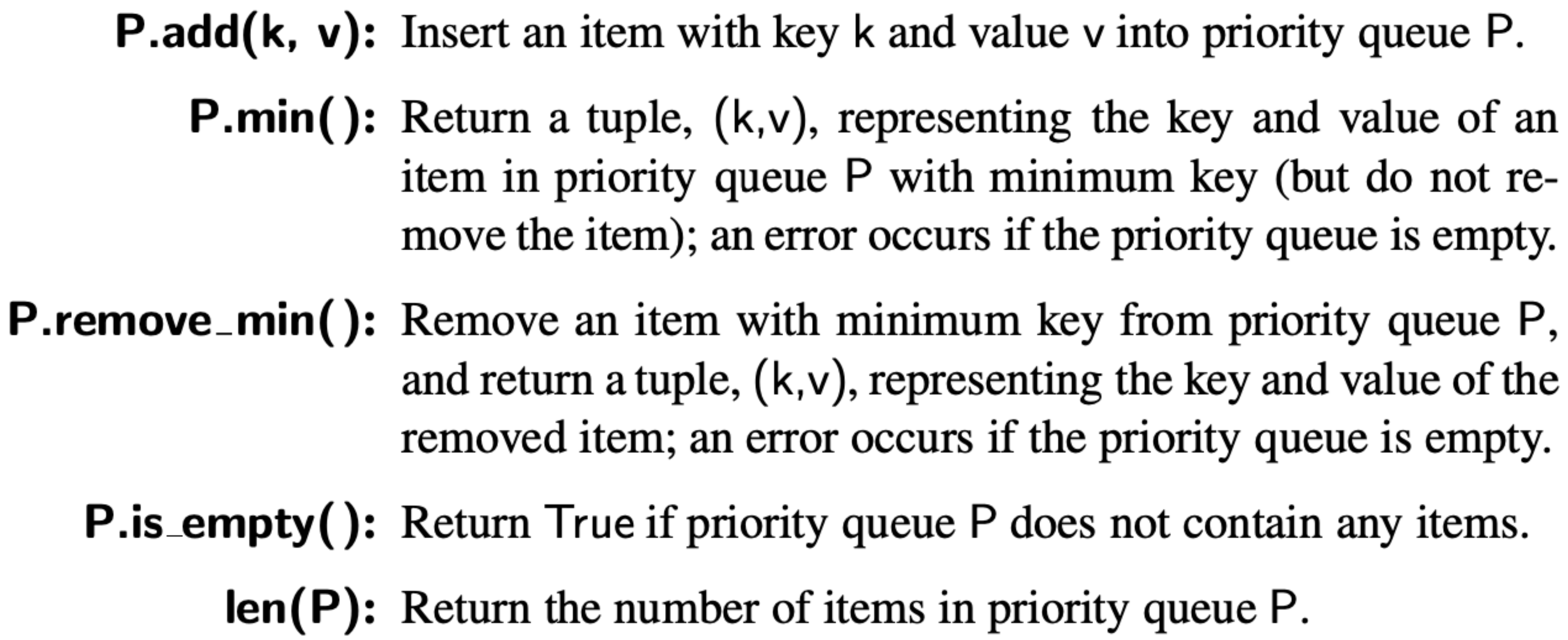
Priority Queues

Priority Queue

- FIFO - “first come, first serve”- policy is not suitable for all queues
- an air-traffic control center that has to decide which flight to clear for landing from among many approaching the airport: plane’s distance from the runway, time spent waiting in a holding pattern, or amount of remaining fuel
- Different priority criteria: flight booking

Priority Queue ADT

- an element and its priority is a key-value pair



P.add(k, v): Insert an item with key k and value v into priority queue P.

P.min( ): Return a tuple, (k,v), representing the key and value of an item in priority queue P with minimum key (but do not remove the item); an error occurs if the priority queue is empty.

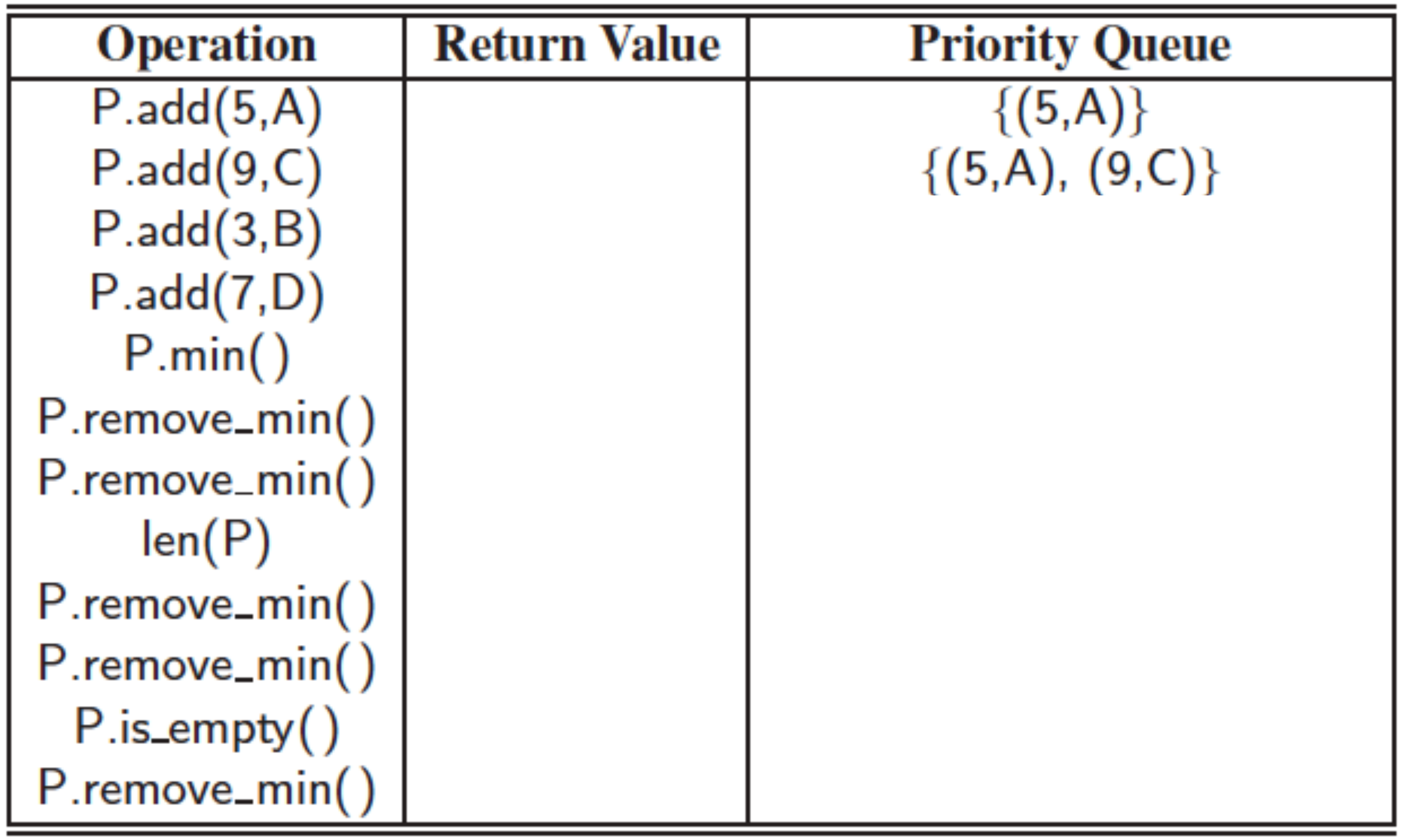
P.remove_min( ): Remove an item with minimum key from priority queue P, and return a tuple, (k,v), representing the key and value of the removed item; an error occurs if the priority queue is empty.

P.is_empty( ): Return True if priority queue P does not contain any items.

len(P): Return the number of items in priority queue P.

Priority Queue Example

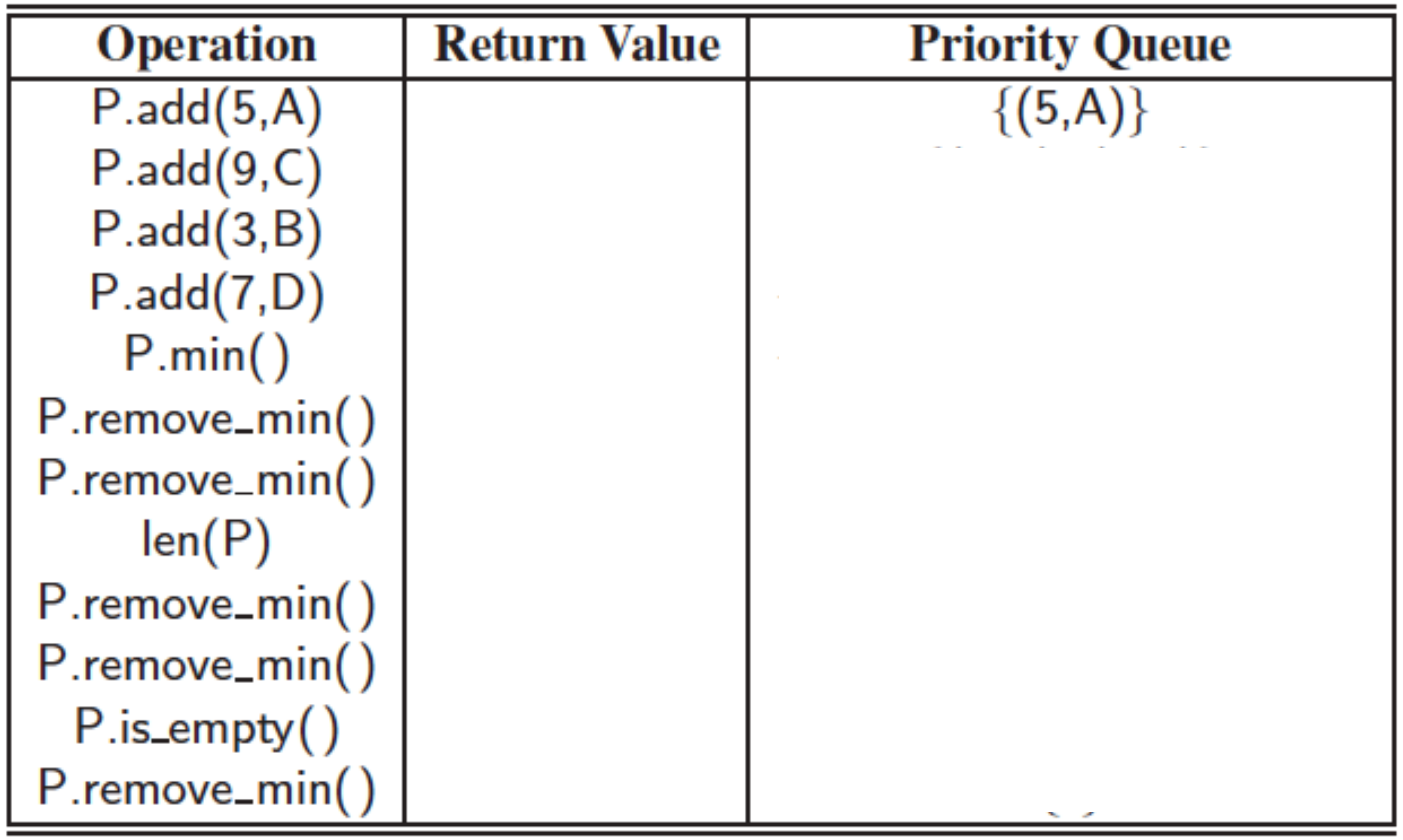
Q1: What is the last returned value in the ‘Return Value’ column?



Operation	Return Value	Priority Queue
P.add(5,A)		{(5,A)}
P.add(9,C)		{(5,A), (9,C)}
P.add(3,B)		
P.add(7,D)		
P.min( )		
P.remove_min( )		
P.remove_min( )		
len(P)		
P.remove_min( )		
P.remove_min( )		
P.is_empty( )		
P.remove_min( )		

Priority Queue Example

What is the last returned value in the ‘Return Value’ column?



Operation	Return Value	Priority Queue
P.add(5,A)		{(5,A)}
P.add(9,C)		
P.add(3,B)		
P.add(7,D)		
P.min( )		
P.remove_min( )		
P.remove_min( )		
len(P)		
P.remove_min( )		
P.remove_min( )		
P.is_empty( )		
P.remove_min( )		

Priority Queue implementation using List

- Insertion into the list O(n)
- Sorting O(n*log(n))

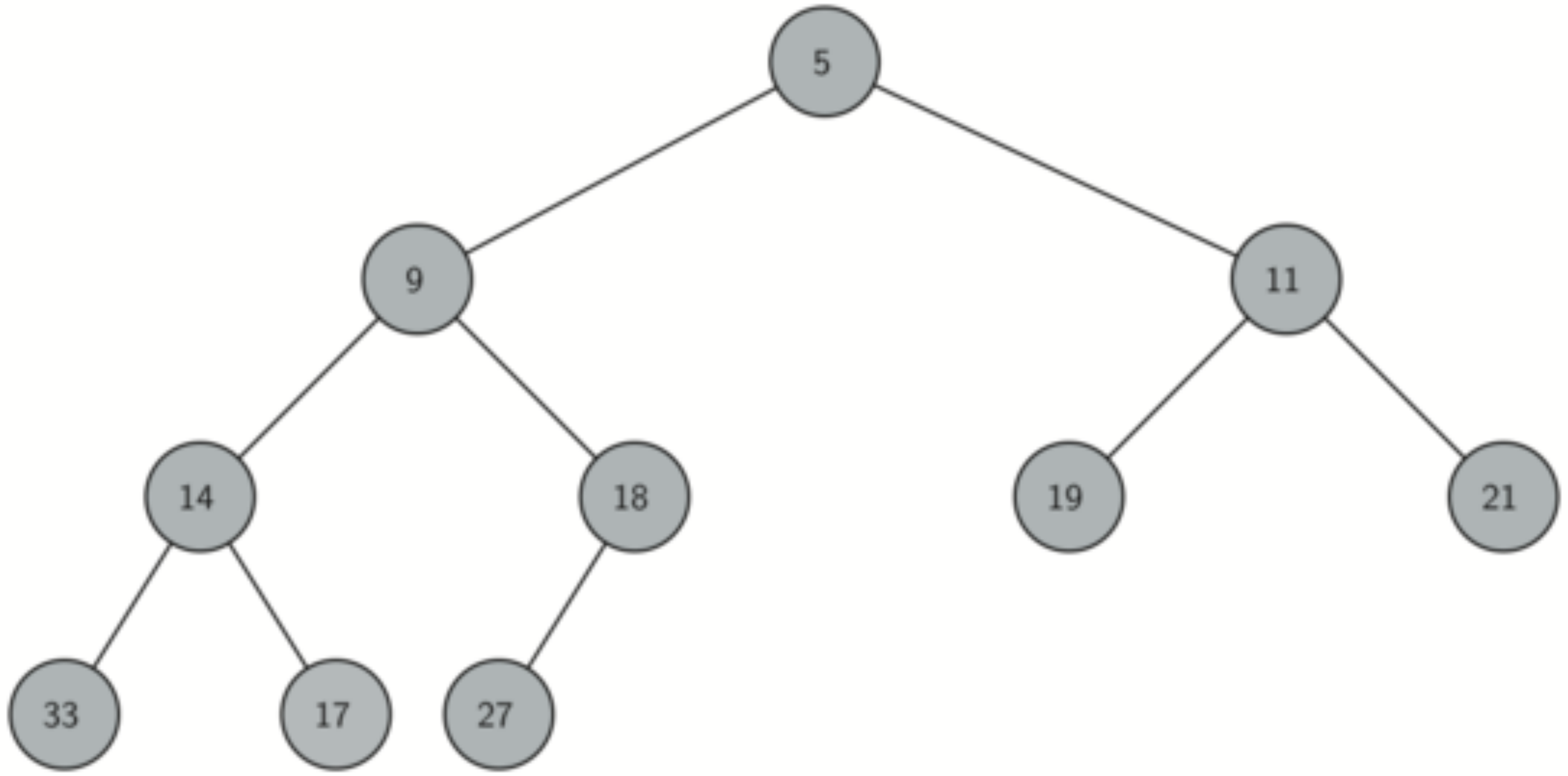
Heaps

Heap

Min-heap: the root node holds the minimum value

- A heap is a binary tree storing keys at its nodes and satisfying the following properties:
- Heap-Order Property: for every internal node v other than the root,

  $key(v) \geq key(parent(v))$



Max-heap: The root node holds

the minimum value

Priority Queue implementation using Tree

Min-heap (or max-heap)

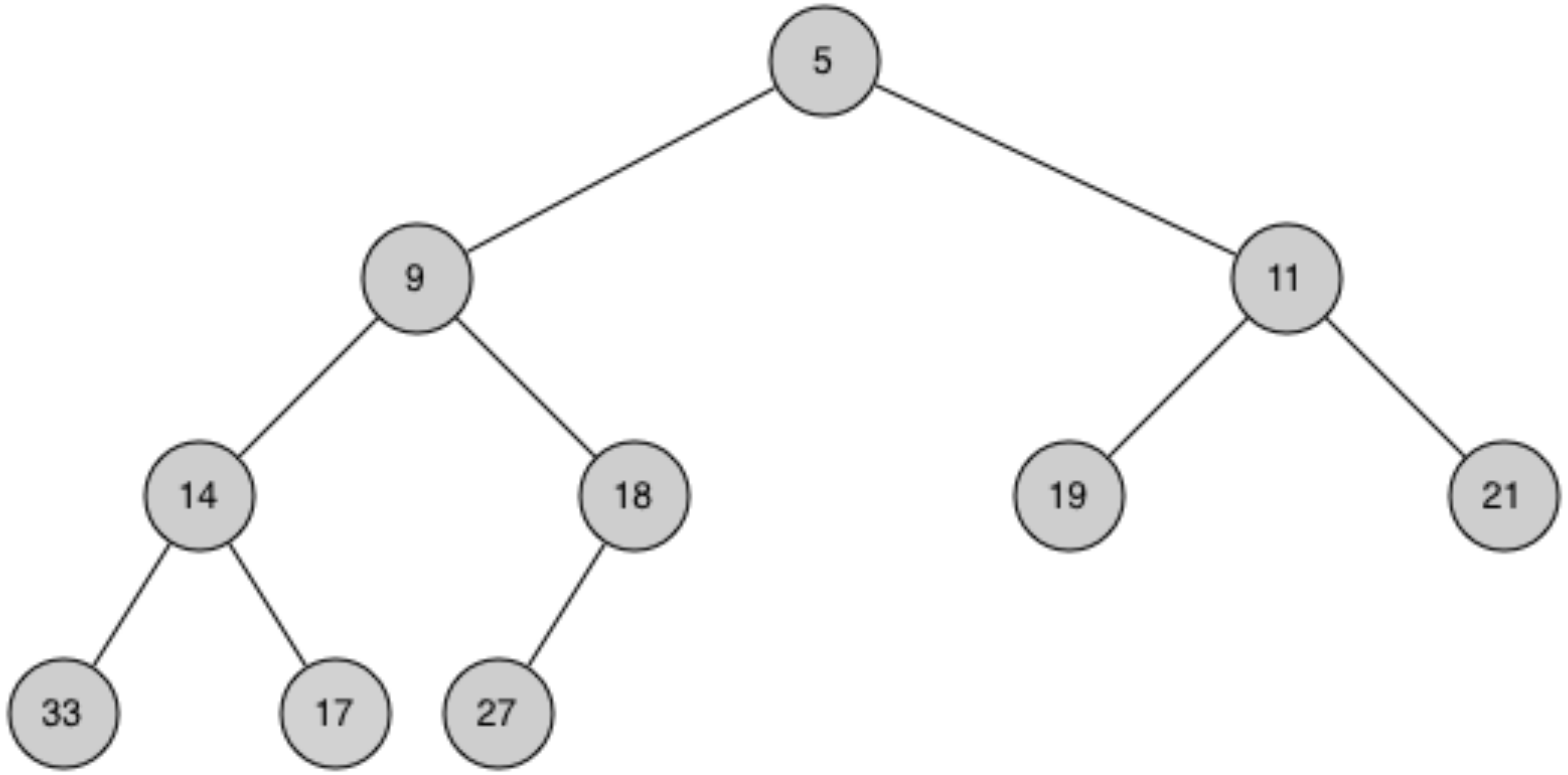
- Insertion into a min-heap (or max-heap): O(log(n))
- Lookup for the min (or max) element: O(1)
- Deletion: O(log (n))

Heap Application

- Sequence of information (strings, integers) presented in ‘streaming’ fashion:
    - Sequence is of the unknown length or length>k
    - Computer the k longest strings in the sequence seen so far:
        - Min-heap: find-min, remove-min, insert

The Heap Order Property

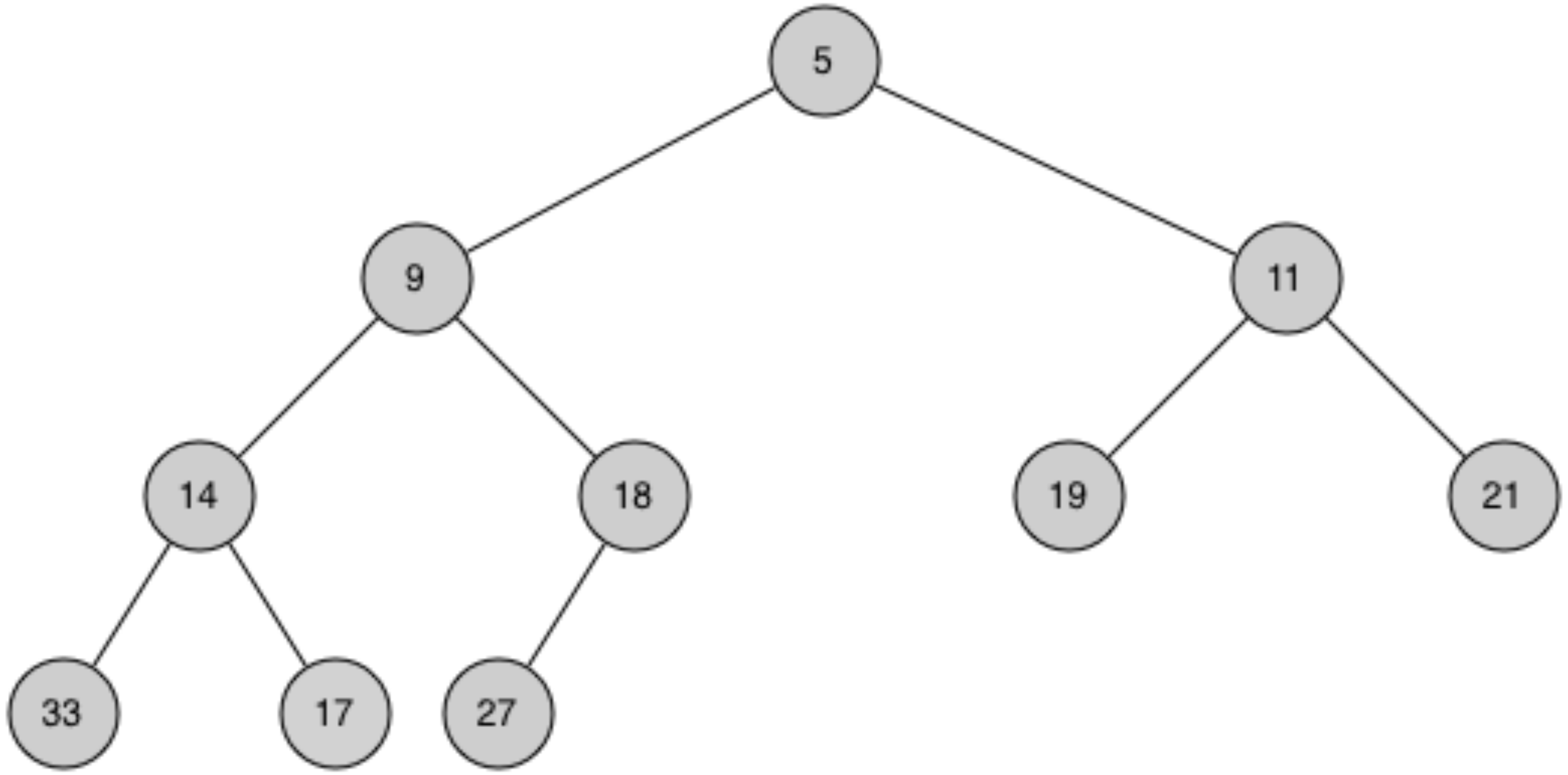
A Complete Binary Tree



Q2: What is a list representation of this tree?

The Heap Order Property

A Complete Binary Tree

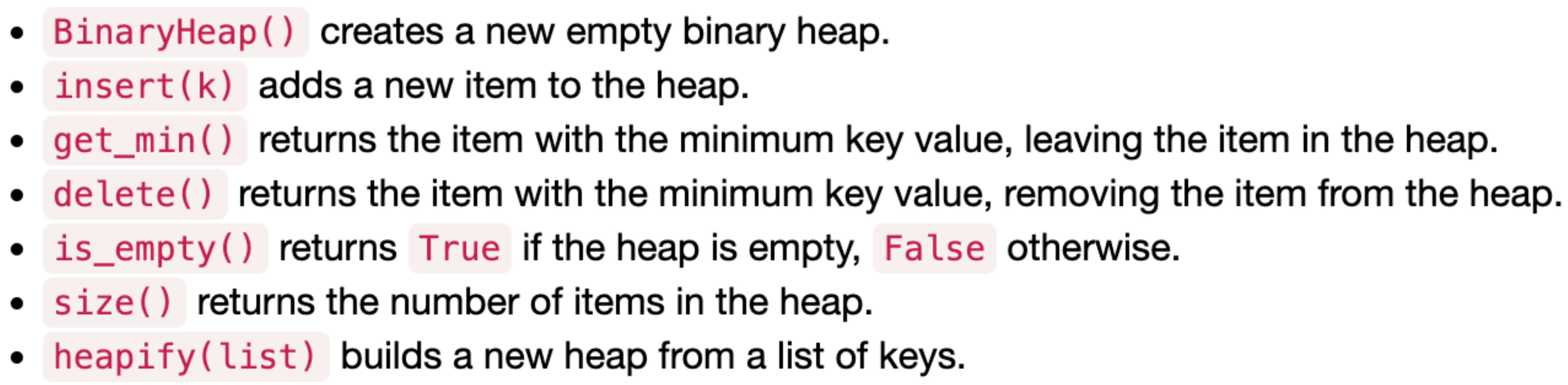


List Representation:

Binary Heap

- A specialized binary tree:
    - A **complete** binary tree (every level - except the last one- is completely filled, and all nodes are as far left as possible)
    - Max-heap: the children of the node at index I are at 2*I+1 and 2*I+2 indices
    - **Max-heap** supports **O(log(n)) insertion** and **O(1)** time **look up** for the max element and O(log(n)) deletion of max element
    - **Search** for any key is **O(n)**
    - **Min-heap**: supports **O(1) lookup** for the minimum element

Heap Operation



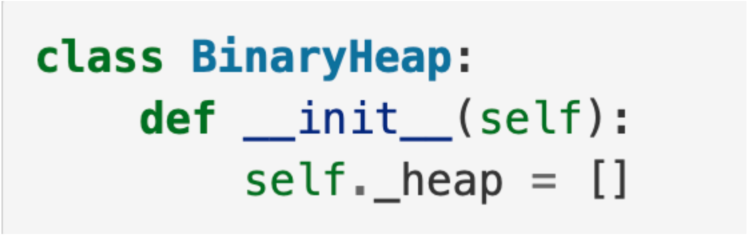
- `BinaryHeap()` creates a new empty binary heap.
- `insert(k)` adds a new item to the heap.
- `get_min()` returns the item with the minimum key value, leaving the item in the heap.
- `delete()` returns the item with the minimum key value, removing the item from the heap.
- `is_empty()` returns `True` if the heap is empty, `False` otherwise.
- `size()` returns the number of items in the heap.
- `heapify(list)` builds a new heap from a list of keys.

Heap in Python

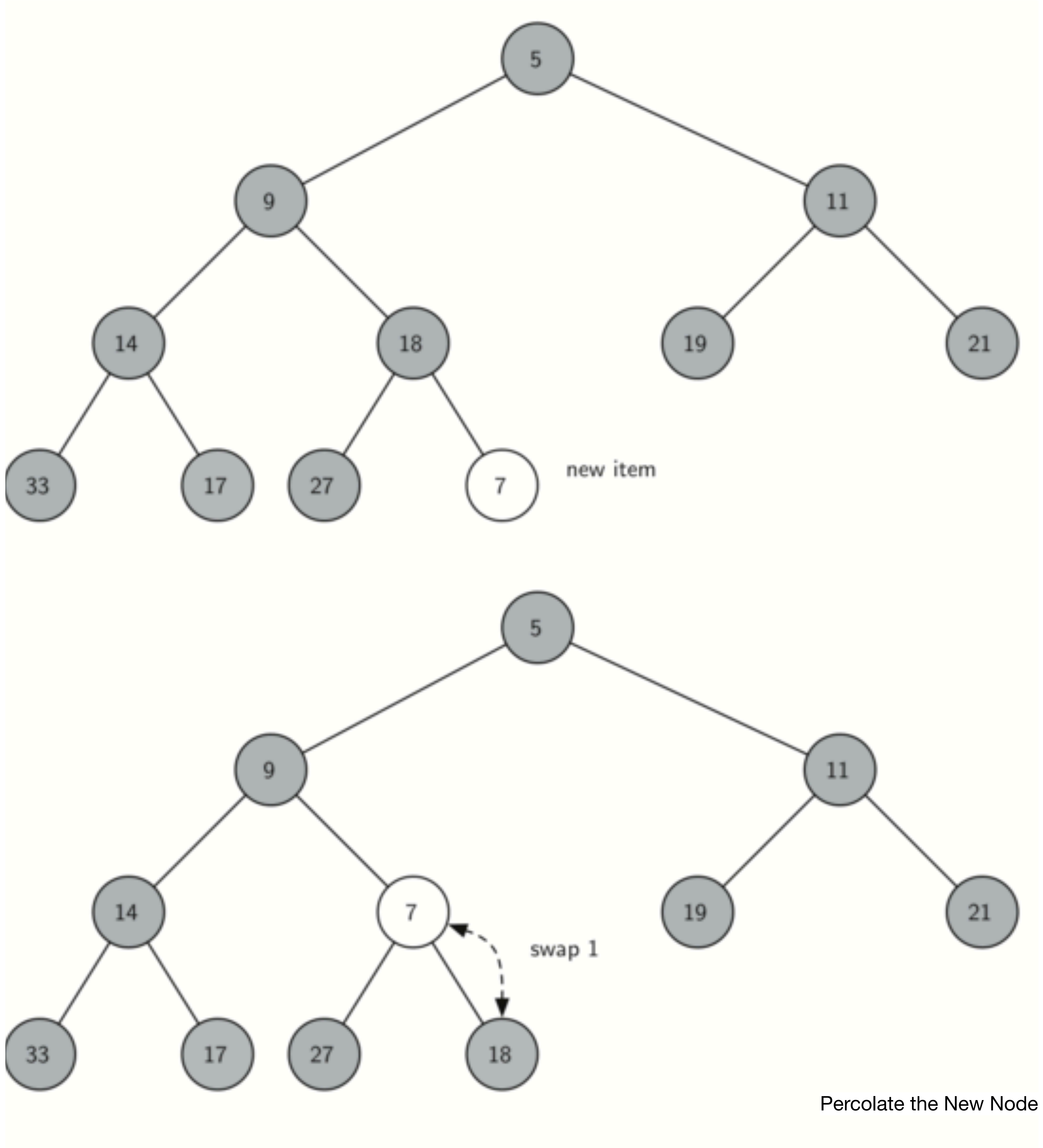
import heapq

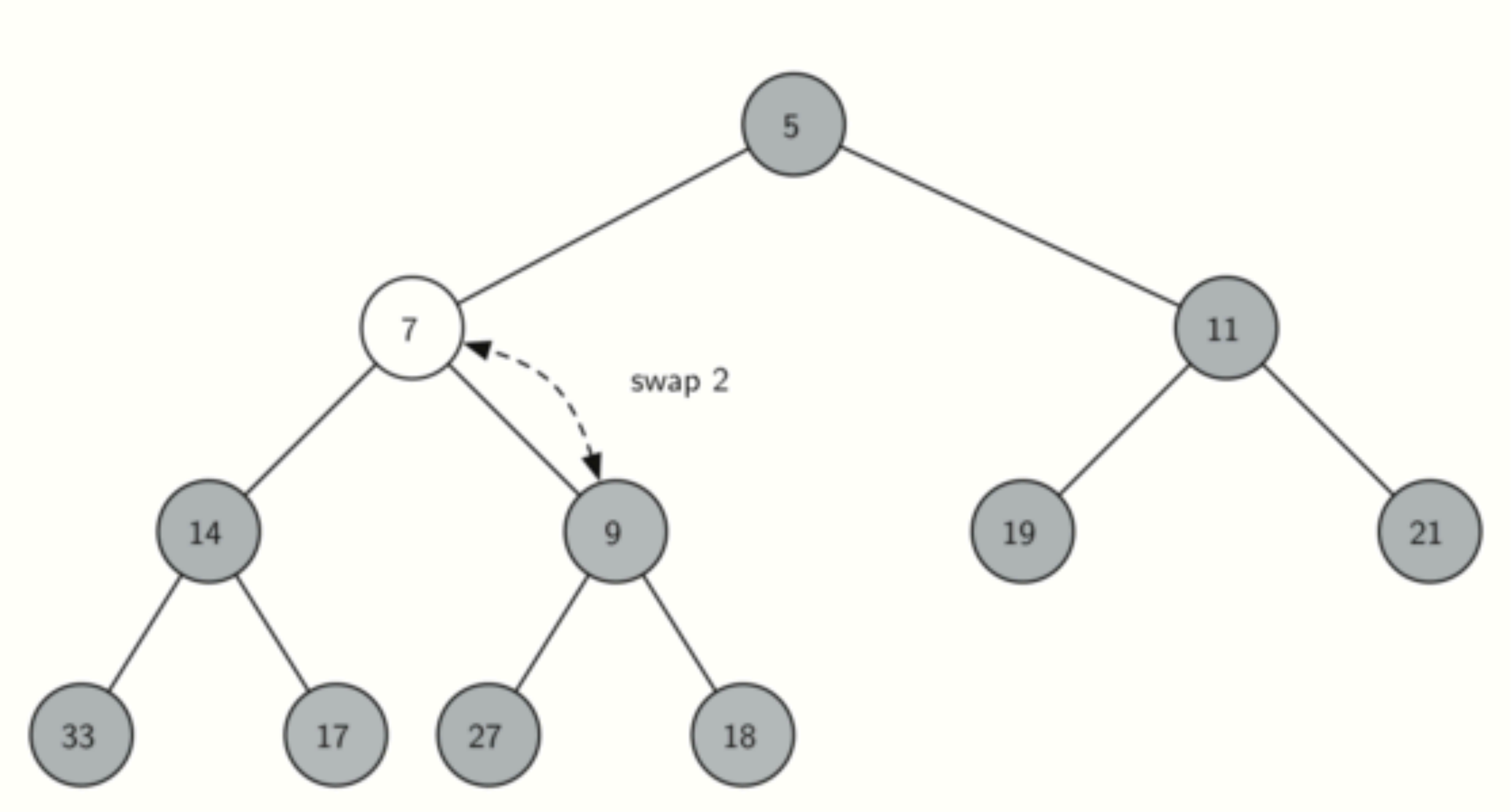
- https://docs.python.org/3/library/heapq.html#basic-examples
-
- **Heapq** module
    - heapq.heapify(L) : transforms elements of L into a min-heap in-place
    - heapq.nlargest(k,L), heapq.nsmallest(k,L) : returns k largest/smallest elements
    - heapq.heappush(h,e)
    - heapq.heappop(h)
    - heapq.heappushpop(h,a)
    - e=h[0]: returns the smallest element

Insertion



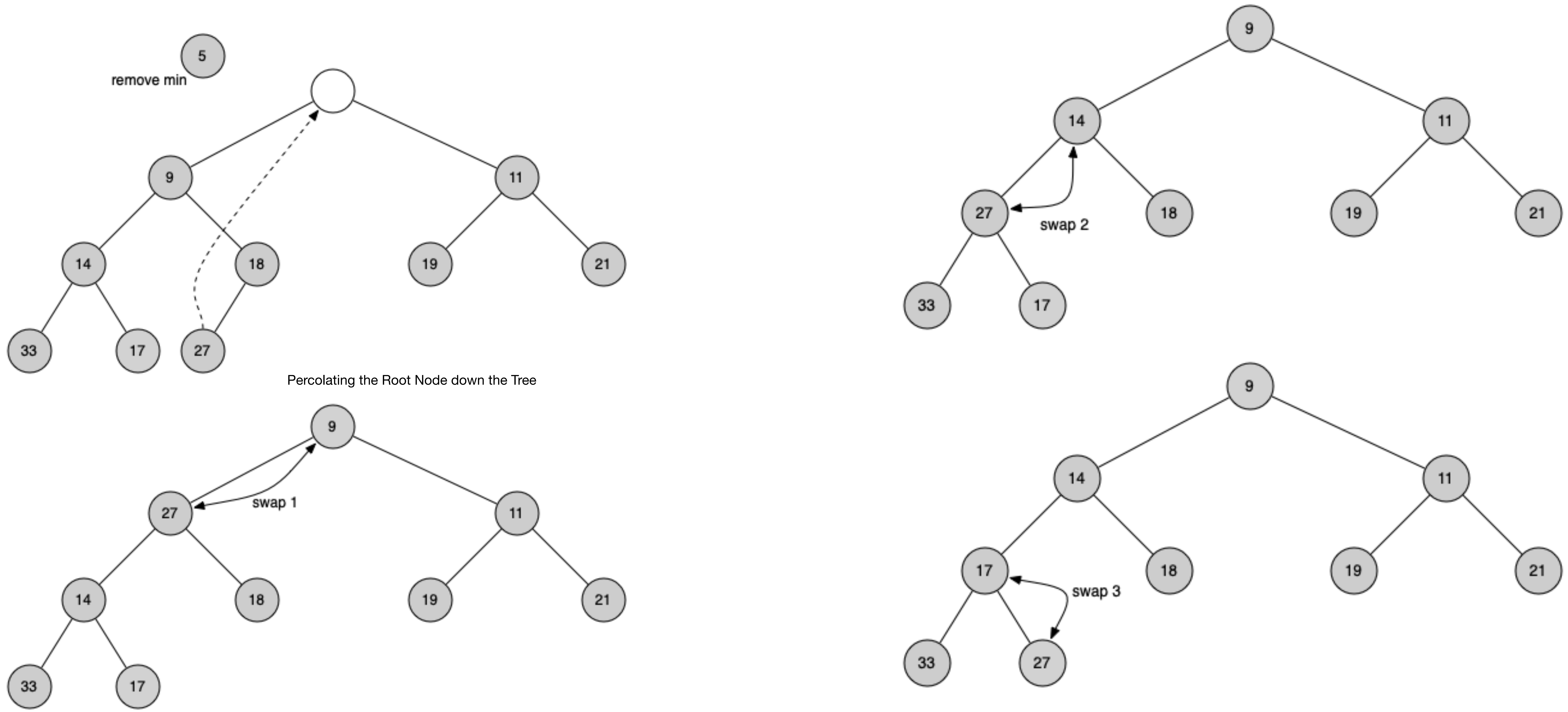
```
class BinaryHeap:
    def __init__(self):
        self._heap = []
```





Percolate the New Node up to Its Proper Position

Min removal



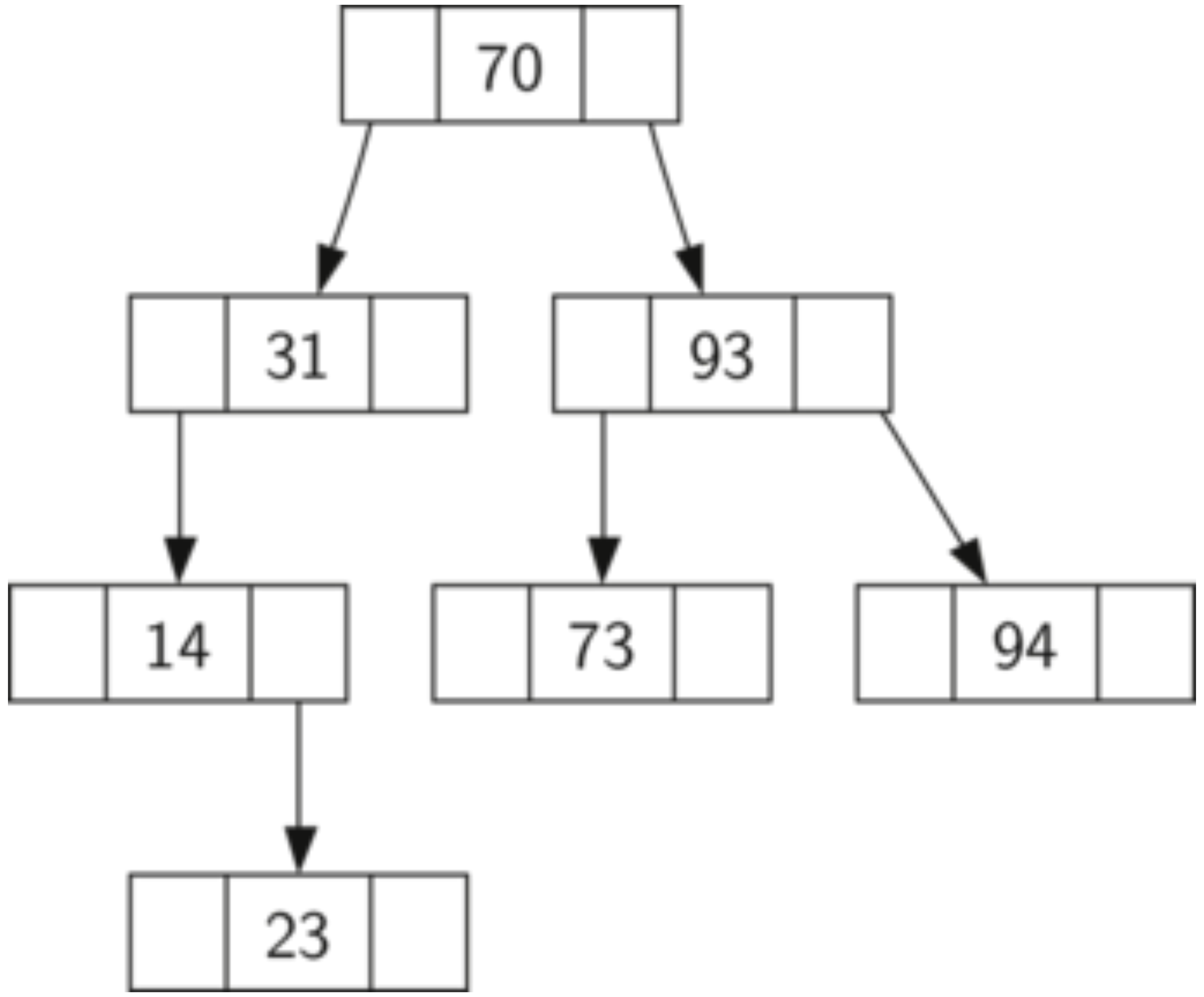
Percolating the Root Node down the Tree

Binary Search Tree

Binary Search Tree (BST) Property

- Keys that are less than the parent are found in the left subtree, and keys that are greater than the parent are found in the right subtree

- Showing **keys** only

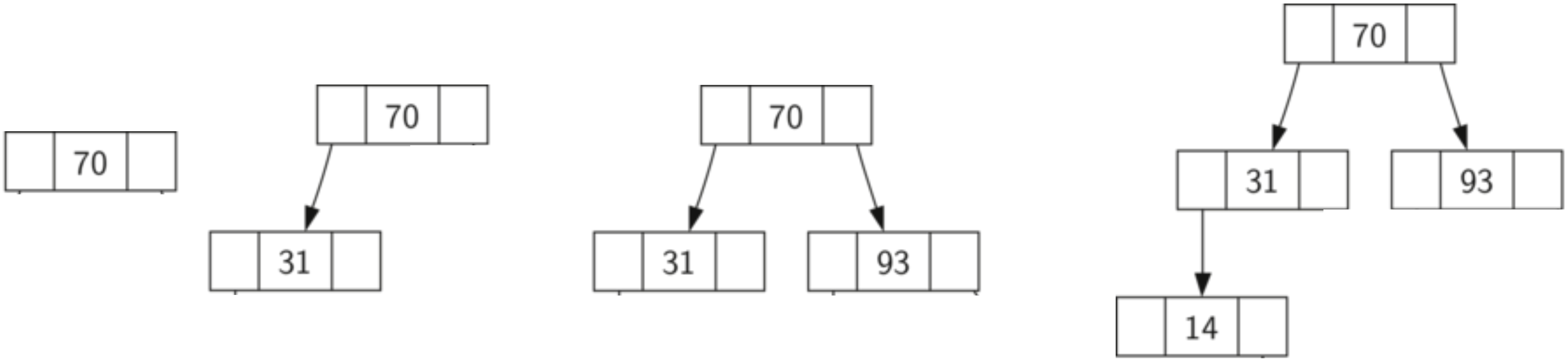


Note: Play Slideshow

BST creation

70, 31, 93, 14, 23, 73, 94

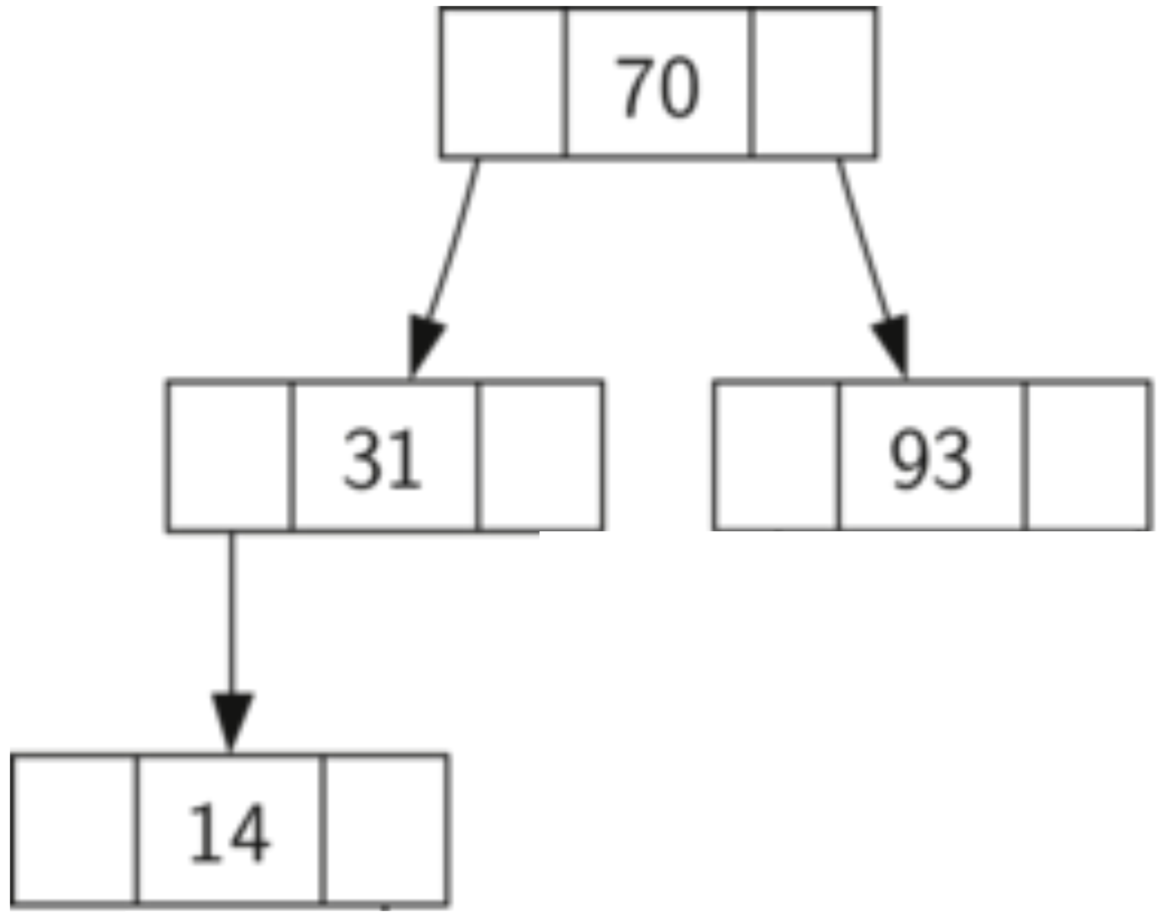
- Inserting nodes one by one:

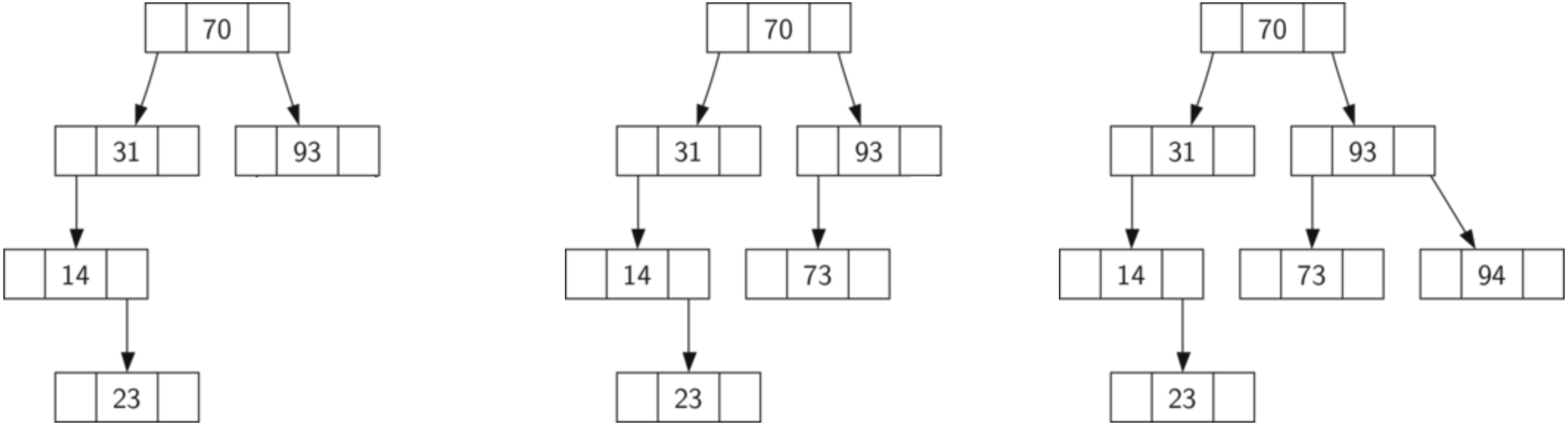


BST creation

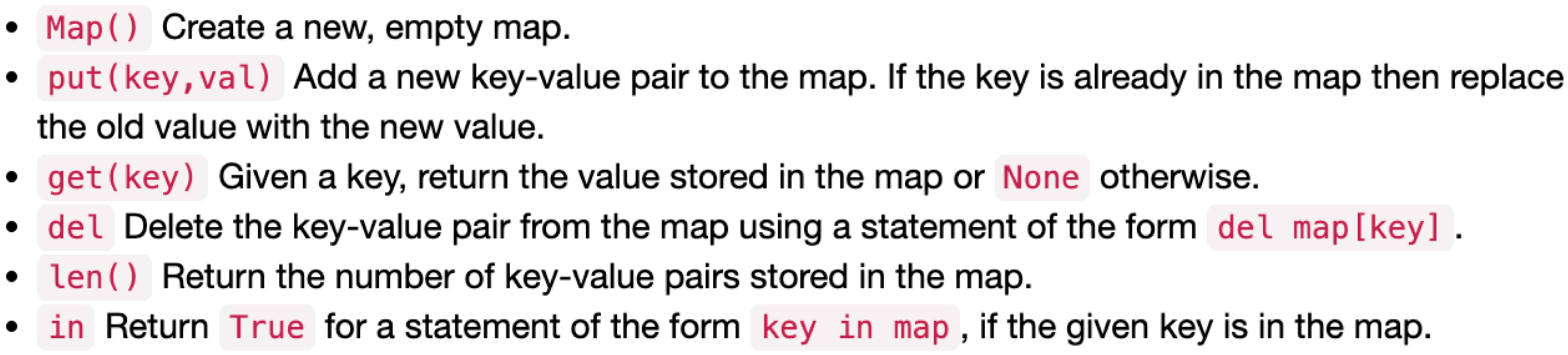
70, 31, 93, 14, 23, 73, 94

- Inserting nodes one by one:



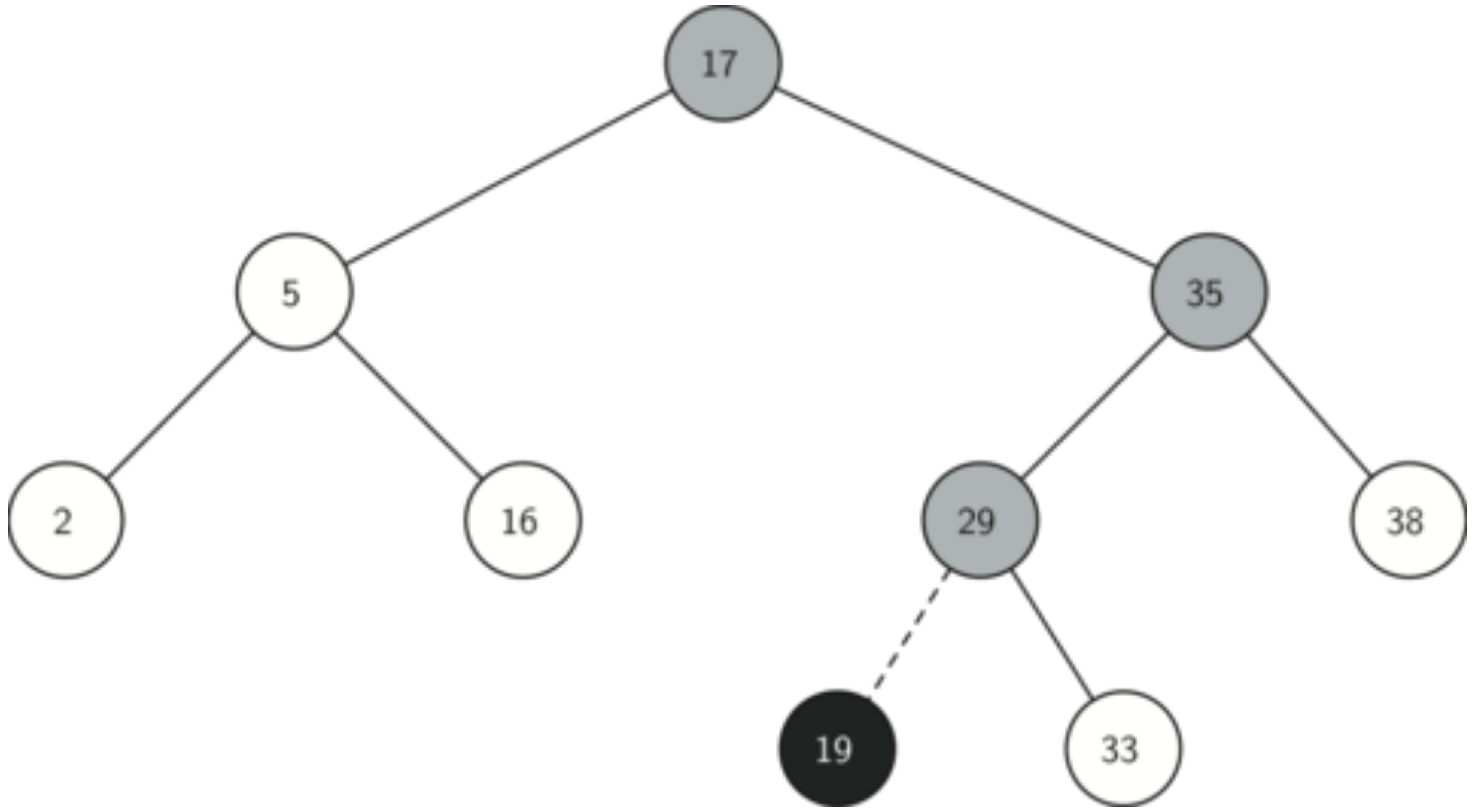


Search Tree Operations



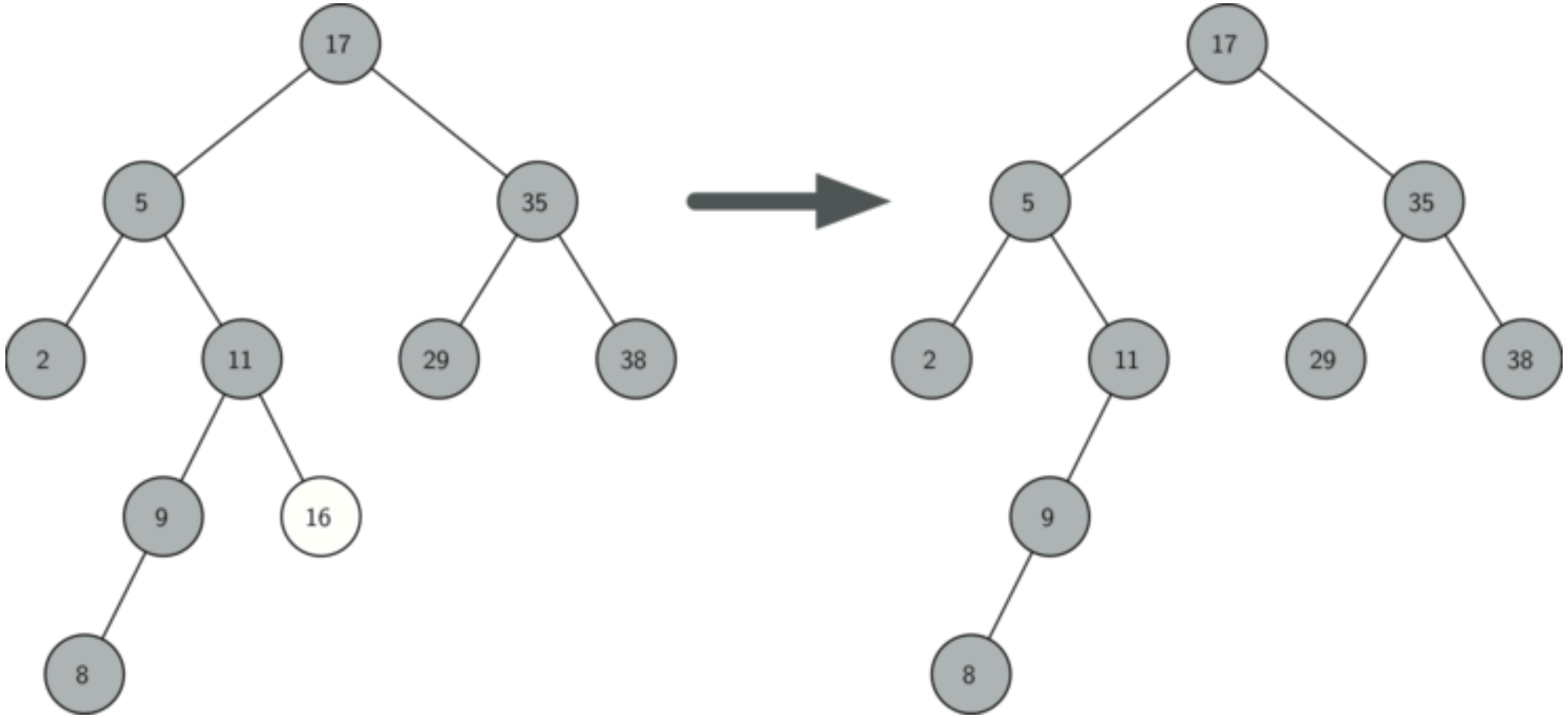
- `Map()` Create a new, empty map.
- `put(key,val)` Add a new key-value pair to the map. If the key is already in the map then replace the old value with the new value.
- `get(key)` Given a key, return the value stored in the map or `None` otherwise.
- `del` Delete the key-value pair from the map using a statement of the form `del map[key]`.
- `len()` Return the number of key-value pairs stored in the map.
- `in` Return `True` for a statement of the form `key in map`, if the given key is in the map.

Inserting a Node with Key = 19



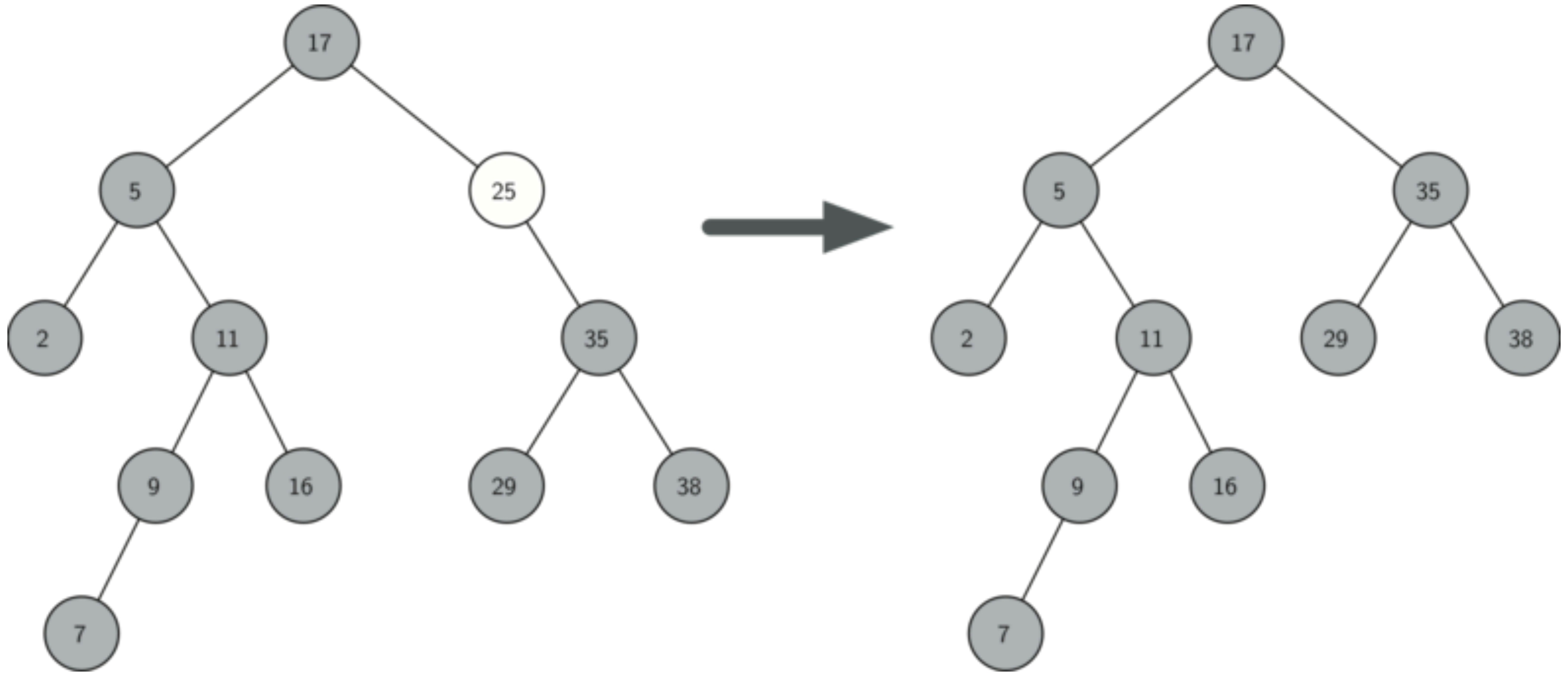
Deleting Node 16, a Node without Children

•



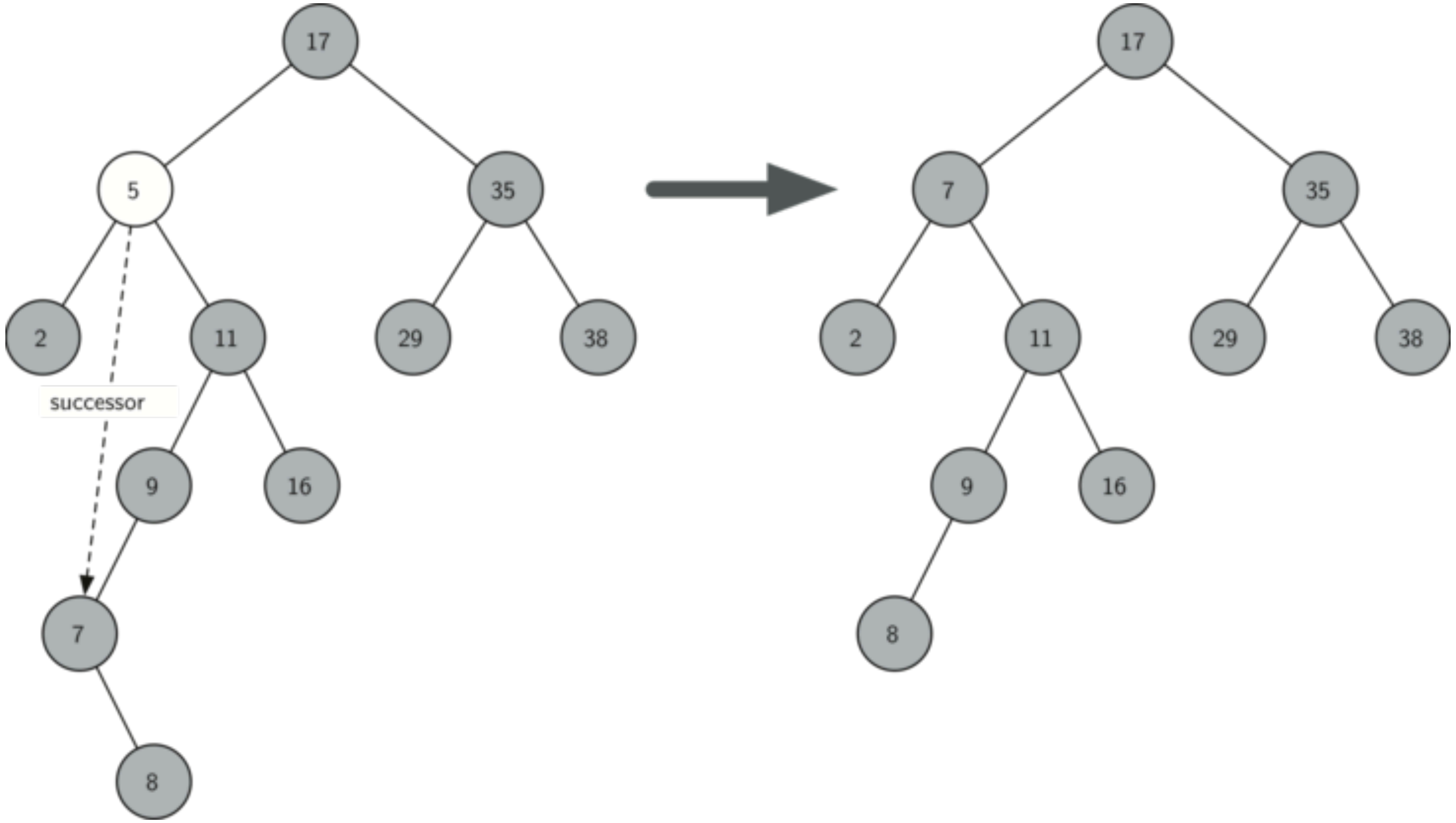
Deleting Node 25, a Node That Has a Single Child

-



Deleting Node 5, a Node with Two Children

- Replace the node with the smallest element from the right subtree (or the largest element from the left subtree)


17
5
35
2
11
29
38
successor
9
16
7
8
17
7
35
2
11
29
38
9
16
8