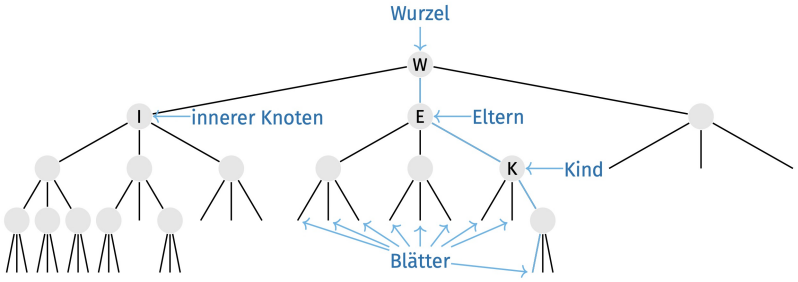
→ Bäume / Trees

→ veralgemeinerte Listen, zusammenhängende dag (dag = directed acyclic graph)



- ***Ordnung*** des Baumes: Maximale Anzahl Kindknoten (hier: 3)
- ***Höhe*** des Baumes: Maximale Pfadlänge Wurzel zu Blatt (hier: 4)

https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/

https://www.geeksforgeeks.org/complete-binary-tree/

h = max länge root zu wurzel

→ BST

node { key ; node* left; node* right; }

in C++ ptr, hier in Python schreiben wir einfach node left oder self.node left siehe weiter unten

Es gilt immer: alle key von current → left sind kleiner als current.key
alle key von current → right sind grösser

⇒ symmetrischer Vorgänger (Nachfolger) ist grösste key im linken Teilbaum (kleinste key im rechten Teilbaum)

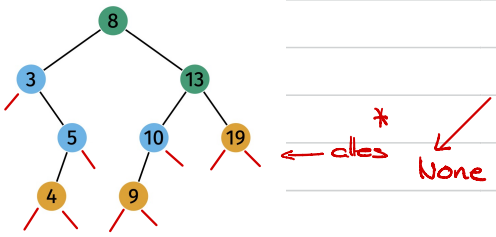
⇒ extract(key) / delete(key) mit
↳ 1 Child ⇒ ptr von parent auf child bewegen
↳ 2 Child ⇒ mit sym NF ersetzen

https://stackoverflow.com/questions/2990486/deletion-procedure-for-a-binary-search-tree

→ remove node

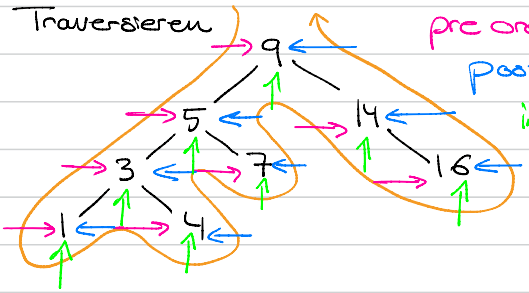
Drei Fälle möglich
- Knoten hat keine Kinder
- Knoten hat ein Kind
- Knoten hat zwei Kinder

[Blätter zählen hier nicht] *



* alles → None

→ trees traversieren:



preorder	9, 5, 3, 1, 4, 7, 14, 16	V, V.left, V.right
postorder	1, 4, 3, 7, 5, 16, 14, 9	V.left, V.right, V
inorder/Symm.	1, 3, 4, 5, 7, 7, 14, 16	V.left, V, V.right

postorder, inorder are DFS

inorder wird bei einem BST immer aufsteigend sein.

- **Heap** = bin. tree + heap condition

heap con min heap ⇒ parent → key ≤ left → key
max ≥ right → key

vollständig bis auf letzte Ebene (von links aufgefüllt)

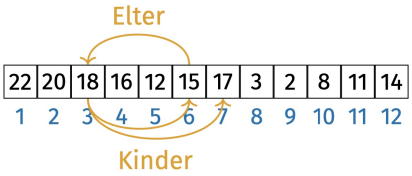
array heap[]: children(i) = {2i, 2i+1} / parent(i) = $\lfloor i/2 \rfloor$

⇒ Insert(key) puts key in heap[0] and bubbles down
⇒ heapify / bubble for min heap
if (! heap con) → swap with smaller child
repeat until heapcon == true i.e. so lässt man den falschen key den heap runter versickern

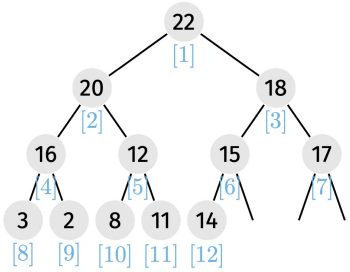
⇒ delete(key) = extractmin(key) auf Teilbaum mit root in key
exmin(key): key entfernen, El. an heap[last] an position von key
d.h. eig. key und letzte position swappen und das ganze versickern.

Baum → Array:

- Kinder$(i) = \{2i, 2i+1\}$
- Elter$(i) = \lfloor i/2 \rfloor$



Abhängig vom Startindex![2]



→ heap ist eine Art ein Array schlau zu interpretieren bzw. die Struktur Heap kann schlau in einem Array gespeichert werden.

[2] Für Arrays, die bei 0 beginnen: $\{2i, 2i+1\} \to \{2i+1, 2i+2\}$, $\lfloor i/2 \rfloor \to \lfloor (i-1)/2 \rfloor$

Algorithm Sift-Up(A, m)

Input: Array A with at least m keys and heap structure on $A[1, \ldots, m-1]$
Output: Array A with heap structure on $A[1, \ldots, m]$

```
v ← A[m] // new key
c ← m // index current node (child)
p ← ⌊c/2⌋ // index parent node
while c > 1 and v > A[p] do
    A[c] ← A[p] // key parent node → key current node
    c ← p // parent node → current node
    p ← ⌊c/2⌋
A[c] ← v // place new key
```

Algorithm SiftDown(A, i, m)

Input: Array A with heap structure for the children of i. Last element m.
Output: Array A with heap structure for i with last element m.

```
while 2i ≤ m do
    j ← 2i; // j left child
    if j < m and A[j] < A[j+1] then
        j ← j+1; // j right child with greater key
    if A[i] < A[j] then
        Swap(A[i], A[j])
        i ← j; // keep sinking down
    else
        i ← m; // sift down finished
```

Gute links:

https://randerson112358.medium.com/min-heap-deletion-step-by-step-1e05ff9d3932

https://www.geeksforgeeks.org/introduction-to-heap-data-structure-and-algorithm-tutorials/

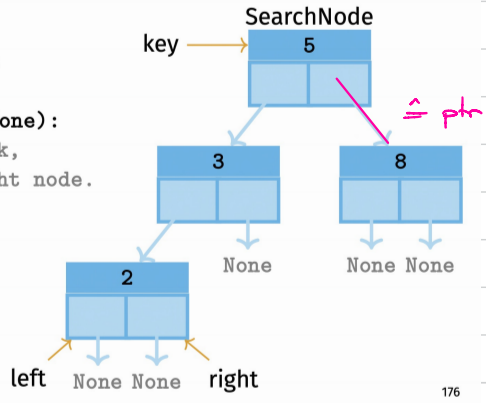
https://www.baeldung.com/cs/binary-tree-max-heapify

https://stackoverflow.com/questions/9755721/how-can-building-a-heap-be-on-time-complexity

→ Nodes in python

```
class SearchNode:
# implicit entries key, left, right

  def __init__(self, k, l=None, r=None):
    # Constructor that takes a key k,
    # and optionally a left and right node.
    self.key = k
    self.left, self.right = l, r
```



≙ ptr auf right child

(sehen wir später im Kurs genauer)

Class Variables — Declared inside the class definition (but outside any of the instance methods). They are not tied to any particular object of the class, hence shared across all the objects of the class. Modifying a class variable affects all objects instance at the same time.

Instance Variable — Declared inside the constructor method of class (the `__init__` method). They are tied to the particular object instance of the class, hence the contents of an instance variable are completely independent from one object instance to the other.

https://medium.com/python-features/class-vs-instance-variables-8d452e9abcbd