Datenstrukturen

Lösungen+

Version vom 19. Oktober 2019

Aufgabe 1.1

Eine Sammlung von Objekten sowie eine Menge zulässiger Operationen, die darauf zugreifen.

Aufgabe 1.2

- Integer
- Float
- Boolean

Aufgabe 1.3

(a) `L[i]` $O(1)$

(b) `L.pop(i)` $O(n)$

(c) `L.pop()` $O(1)$

(d) `L.append(77)` $O(1)$

(e) `L.insert(i, 77)` $O(n)$

(f) `for x in L: ...` $O(n)$

Aufgabe 1.4

(a) `D[k]` $O(1)$

(b) `D.del(k)` $O(1)$

(c) `for k in D:` $O(n)$
```
    print(D[k])
```

Aufgabe 1.5

Mengen sind in Python als Hash-Tabellen implementiert. Daher haben Operationen zum Nachschlagen, Einfügen und Löschen (im Mittel) die Komplexität $O(1)$.

(a) `A.add(24)` $O(1)$

(b) `27 in A` $O(1)$

(c) `A.union(B)` $O(m+n)$

Aufgabe 2.1

- UPN-Rechner (Postfix-Notation)
- Undo-Funktion in Textverarbeitungsprogrammen
- Browser-History
- Backtracking-Algorithmen

Aufgabe 2.2

- `s.peek()` liefert das oberste Element eines Stacks zurück, ohne es zu entfernen.
- `s.pop()` liefert das oberste Element eines Stacks zurück und entfernt es.

Aufgabe 2.3

(a)
```
def push(item):
    self.items.append(item)
```

(b)
```
def pop():
    return self.items.pop()
```

(c)
```
def isEmpty__():
    return self.items == []
```

Aufgabe 2.4

(a) `a * b - (c + d)` ⇒ `a b * c d + -`

(b) `c / b ** (d - a)` ⇒ `c b d a - ** /`

Aufgabe 2.5

`(bottom) m, a, k (top)`

Aufgabe 2.6

- Druckerwarteschlangen
- Simulationen (Verkehrsflüsse, Supermarktkassen, ...)
- Puffern von Maus und Tastatureingaben in Message Queues
- Puffern von Daten zwischen asynchronen Prozessen

Aufgabe 2.7

FIFO (First in – First out)

Aufgabe 2.8

(a)
```
def __enqueue__(self, item):
    self.items.insert(0, item)
```

(b)
```
def size(self):
    return len(self.items)
```

Aufgabe 2.9

```
-> s, z, f, a ->
```

Aufgabe 2.10

```
from Deque import Deque

def palindrome(string):
    D = Deque()

    for character in string:
        D.addFront(character)

    while D.size() > 1:
        if D.removeFront() != D.removeRight():
            return False

    return True
```

Aufgabe 3.1 (→ 2.11)

Die Adresse 0x2A verweist auf den Node bei Adresse 0x14.

Der Node bei 0x14 enthält den Wert 0x3A und die Adresse 0x0F.

Der Node bei 0x0F enthält den Wert 0x22 und die Adresse 0x37.

Der Node bei 0x37 enthält den Wert 0x32 und die Adresse 0x00.

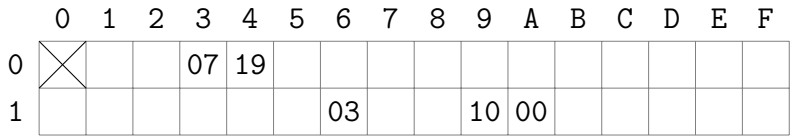
Die Adresse `0x00` kennzeichnet das Listenende.

Somit sind in der Liste die folgenden Bytes gespeichert:

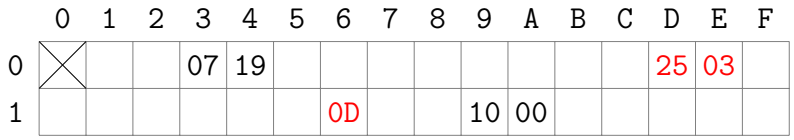
0x3A, 0x22, 0x32

Aufgabe 3.2 (→ 2.12)

vorher:

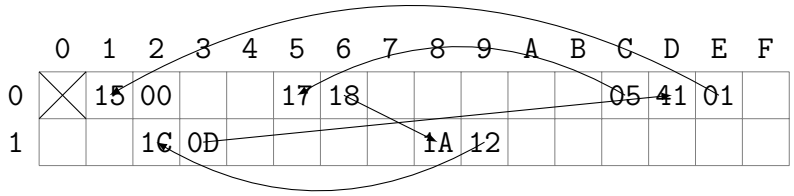


nachher:

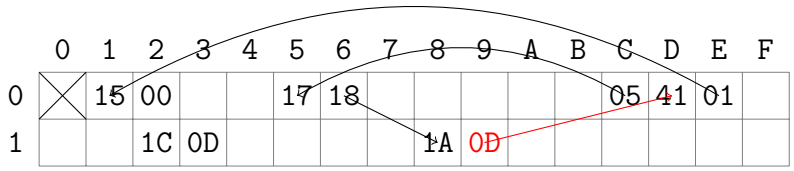


Aufgabe 3.3 (→ 2.13)

vorher:



nachher:



Aufgabe 3.4 (→ 2.14)

(a) Einfügen eines Elements: $O(1)$

(b) Suchen eines Elements: $O(n)$

(c) Löschen eines Elements: $O(n)$

Aufgabe 3.5 (→ 2.15)

Garbage Collection

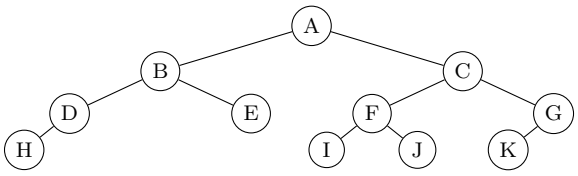
Aufgabe 3.6 (→ 2.16)

```
class Node:

    def __init__(self, data):
        self.data = data
        self.next = None

    def getData(self):
        return self.data

    def getNext(self):
        return self.next

    def setData(self, data):
        self.data = data

    def setNext(self, next):
        self.next = next
```

Aufgabe 3.7 (→ 3.1)

- HTML-Seiten
- Dateisysteme (sofern sie hierarchisch organisiert sind)
- Parse-Trees

Aufgabe 3.8 (→ 3.2)



(a) Welchen Schlüssel hat die Wurzel? *A*

(b) Welche Kinder hat der Knoten mit dem Schlüssel C? *F, G*

(c) Welche Geschwister hat der Knoten H? *keine*

(d) Welche Eltern hat der Knoten mit dem Schlüssel E? *B*

(e) Welches sind die Blätter des Baums? *H, E, I, J, K*

(f) Welches sind die inneren Knoten des Baums? *A, B, C, D, F, G*

(g) Welche Tiefe hat der Knoten mit dem Schlüssel D? *2*

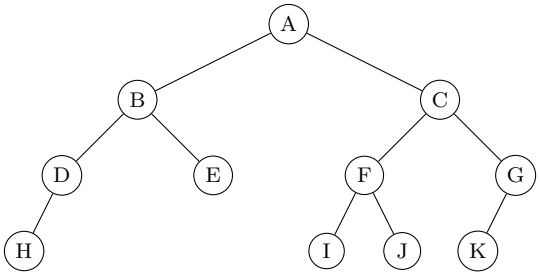
(h) Welche Höhe hat der Baum? *3*

Aufgabe 3.9 (→ 3.3)

Ein Baum ist entweder leer oder er besteht aus einer Wurzel und null oder mehr Bäumen, die jeweils durch eine Kante mit der Wurzel des Elternbaums verbunden sind.

Aufgabe 3.10 (→ 3.4)

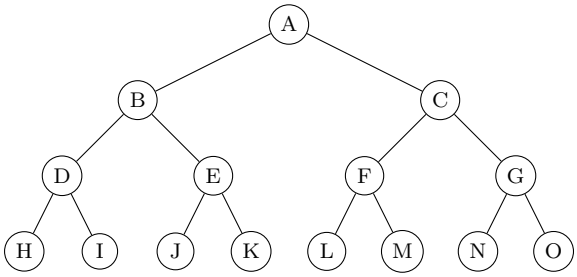
balancierter Binärbaum



Die Tiefe der Blätter unterscheiden sich höchstens um 1.

Aufgabe 3.11 (→ 3.5)

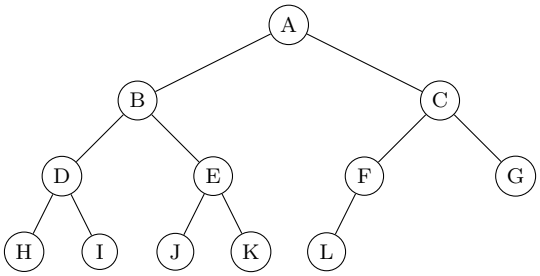
perfekter Binärbaum



Ein voller Binärbaum, dessen Blätter alle dieselbe Tiefe haben.

Aufgabe 3.12 (→ 3.6)

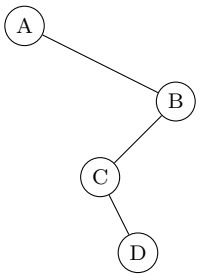
vollständiger Binärbaum



Ein balancierter Binärbaum, bei dem die tieferen Blätter alle links stehen.

Aufgabe 3.13 (→ 3.7)

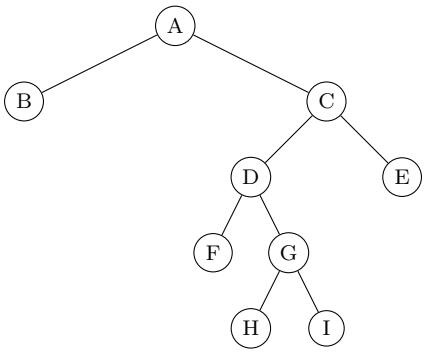
entarteteter Baum



Jeder Knoten hat höchstens ein Kind.

Aufgabe 3.14 (→ 3.8)

voller Binärbaum



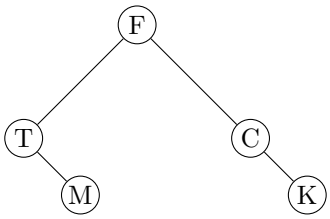
Alle inneren Knoten haben zwei Kinder.

Aufgabe 3.15 (→ 3.9)

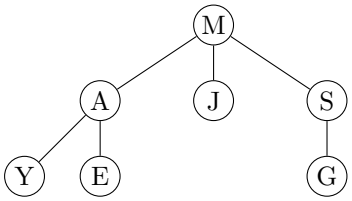
`['F',['T',[],['M',[],[]]],['C',[],['K',[],[]]]]`

Zusammengehörende Klammerpaare untereinander schreiben:

```
[ F
  [ T
    [ ]
    [ M
        [ ]
        [ ]
    ]
  ]
  [ C
    [ ]
    [ K
      [ ]
      [ ]
    ]
  ]
]
```



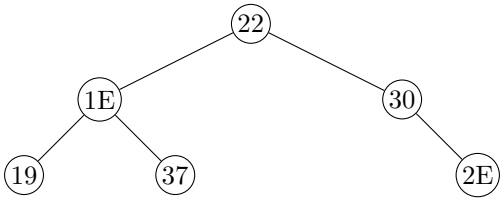
Aufgabe 3.16 (→ 3.10)



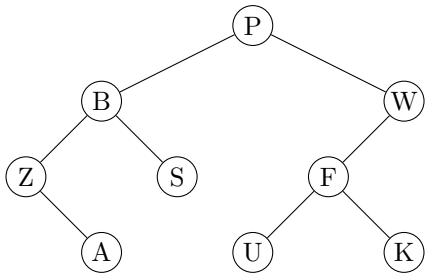

```
[ M
  [ A
    [ Y  [] [] [] ]
    [ E  [] [] [] ]
    [ ]
  ]
  [ J [] [] [] ]
  [ S
    [ ]
    [ G  [] [] [] ]
    [ ]
  ]
]
```

Aufgabe 3.17 (→ 3.11)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		00	0B	41	1E	2F	10	00	00	00	00	00	00	00	00	22
1	37	00	00	00	3A	0F	00	32	41	00	00	00	00	00	00	00
2	00	00	00	00	00	00	00	00	00	0C	14	17	00	00	46	19
3	00	00	22	04	3A	00	00	32	00	00	80	00	4C	24	46	00
4	00	00	00	00	00	00	00	02	04	00	00	00	2E	00	00	00



Aufgabe 3.18 (→ 3.12)



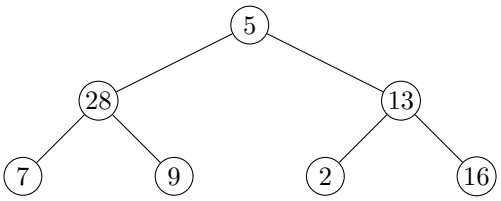
(a) Preorder: *P B Z A S W F U K*

(b) Inorder: *Z A B S P U F K W*

(c) Postorder: *A Z S B U K F W P*

Aufgabe 3.19 (→ 3.13)

`[5, 28, 13, 7, 9, 2, 16]`:



Aufgabe 3.20 (→ 3.15)

(a) $\left\lfloor \frac{76}{2} \right\rfloor = \lfloor 38 \rfloor = 38$

(b) $\left\lfloor \frac{89}{2} \right\rfloor = \lfloor 44.5 \rfloor = 44$

Aufgabe 3.21 (→ 3.16)

(a) Die Liste muss mindestens $2 \cdot 5 + 1 = 11$ Elemente haben.

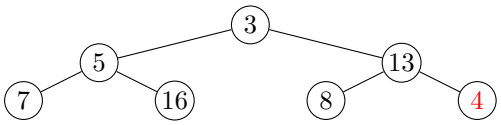
(b) Die Kindknoten haben die Indizes 10 und 11.

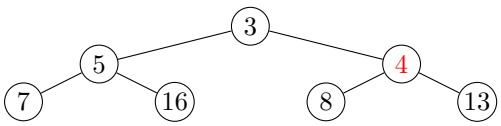
Aufgabe 3.22 (→ 3.17)

Tiefe: $\min\{k \in \mathbb{N} \colon 20 < 2^k\} = 5$

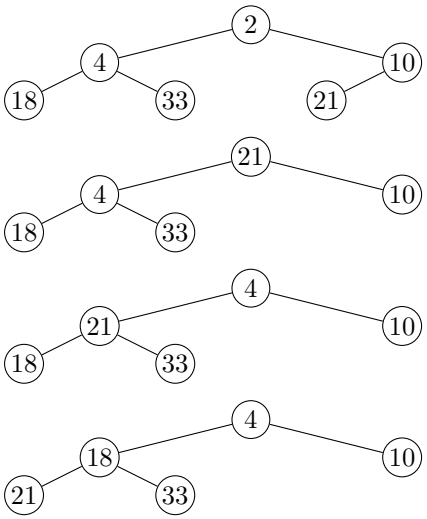
Anzahl Blätter: $20 - (2^{5-1} - 1) = 20 - 15 = 5$

Aufgabe 3.23 (→ 3.18)



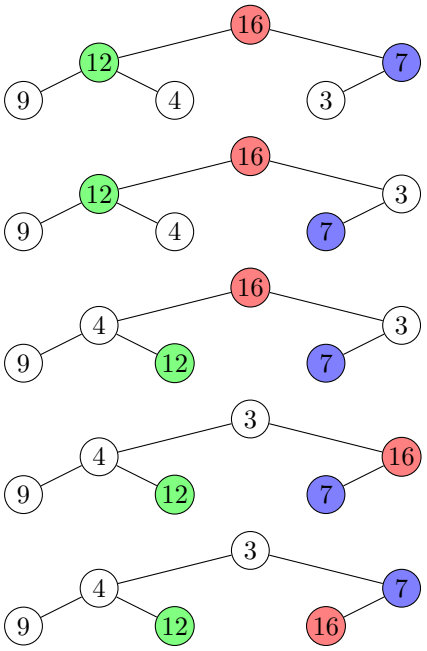


Aufgabe 3.24 ($\rightarrow$ 3.19)



Aufgabe 3.25 ($\rightarrow$ 3.20)

Da in vollständigen Binärbäumen alle Knoten mit einem Index grösser als $\lfloor 6/2 \rfloor = 3$ Blätter sind, genügt es, die 3 inneren Knoten in umgekehrter Reihenfolge so weit wie nötig nach unten zu tauschen.



7 abwärts tauschen:

12 abwärts tauschen:

16 abwärts tauschen: