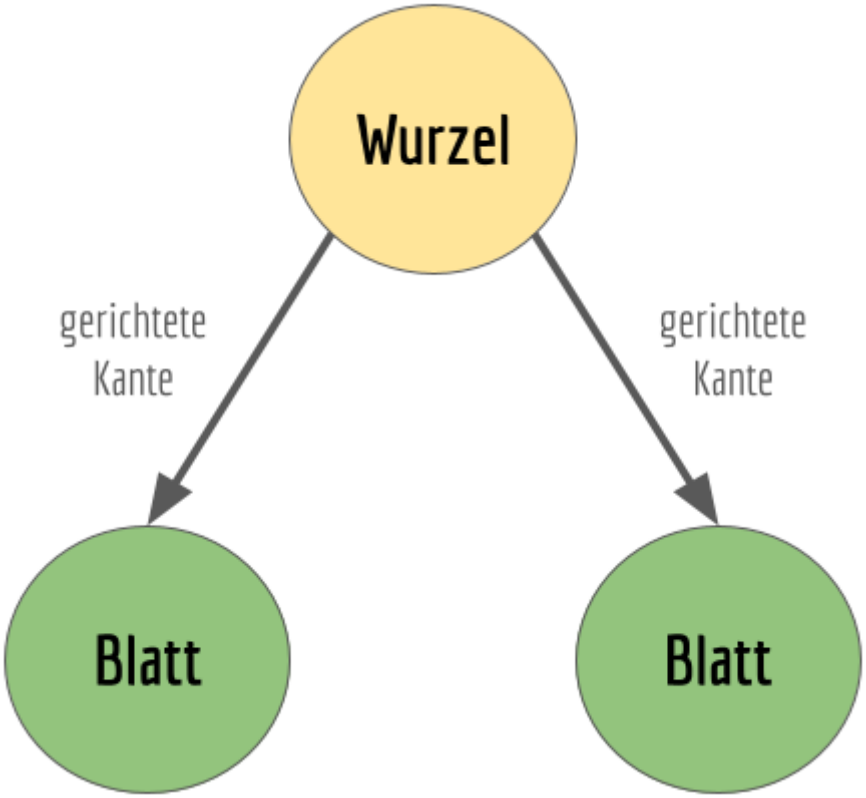
Binärbäume

Bäume sind ein sehr gutes Beispiel dafür, wie man mit Hilfe der Objektorientierung sehr eleganten Code schreiben kann. Wir betrachten an dieser Stelle vollständige und volle Binärbäume, weil diese in der Implementierung am einfachsten zu handhaben sind.

Grundbegriffe

Ein Baum besteht aus **Knoten**. Ganz oben befindet sich der **Wurzelknoten**. Mit ihm sind im Falle eines Binärbaumes zwei **Blattknoten** verbunden. Die Verbindung zwischen den Knoten bezeichnet man als **Kante**. Die Information, welche Blattknoten mit dem Wurzelknoten verbunden sind, befindet sich nur im Wurzelknoten. Der Wurzelknoten „zeigt“ damit auf die Blattknoten, weswegen man die Kante auch als **gerichtet** bezeichnet.



Hier eine einfache Implementierung in Python:

simplenode.py

```
class Node:
    def __init__(self) -> None:
        self.left = None
        self.right = None

# die drei Knoten erstellen
wurzel = Node()
blattlinks = Node()
```

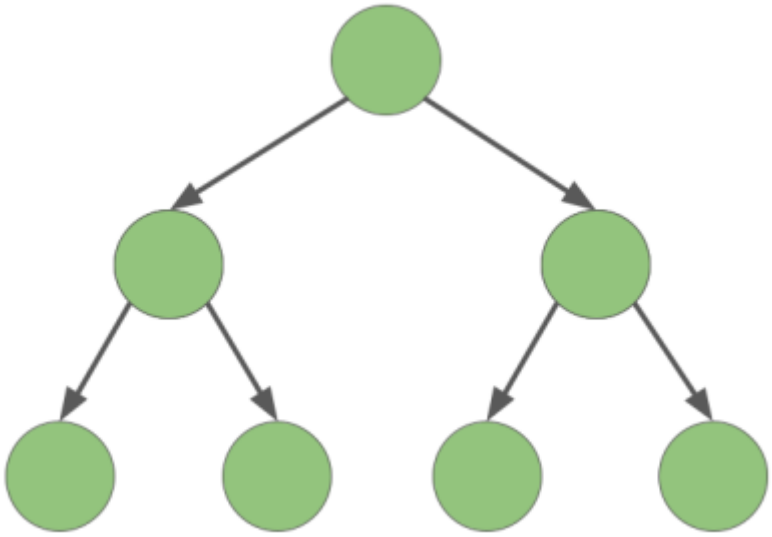
```
blattrechts = Node()

# Verweise vom Wurzelknoten auf die Blätter erstellen (Baum aus den
Knoten bauen)
wurzel.links = blattlinks
wurzel.rechts = blattrechts
```

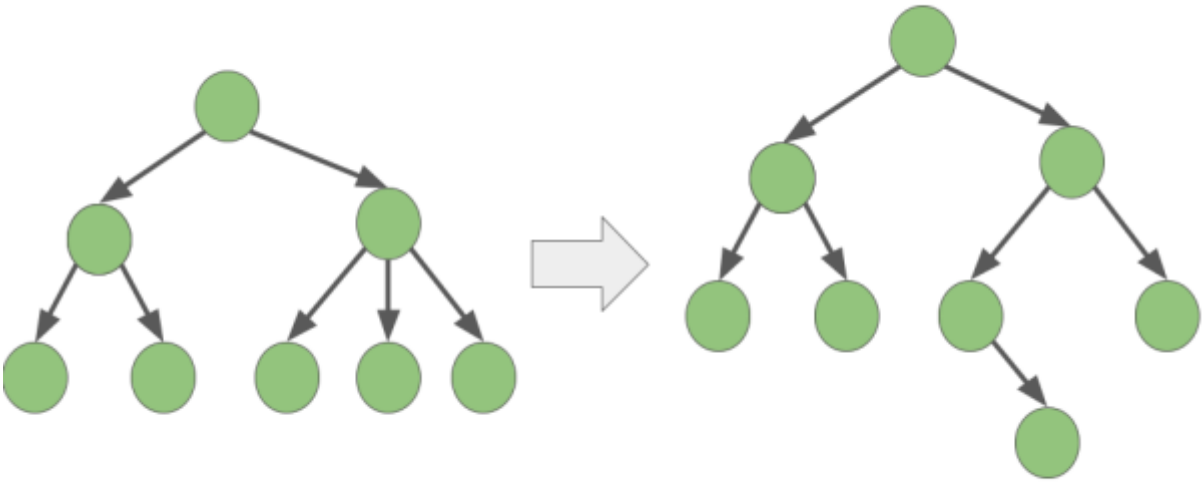
Normalerweise trägt ein Knoten nicht nur Information darüber, welche anderen Knoten mit ihm verbunden sind, sondern speichert zusätzlich Daten. Das werden wir weiter unten bei den Gewichten von Kanten noch in einer Anwendung sehen.

Spezialfall Binärbäume

Ein Binärbaum besitzt immer zwei Blattknoten an einem übergeordneten Knoten. Bei einem vollständigen Binärbaum sind zum alle Ebenen komplett ausgefüllt:

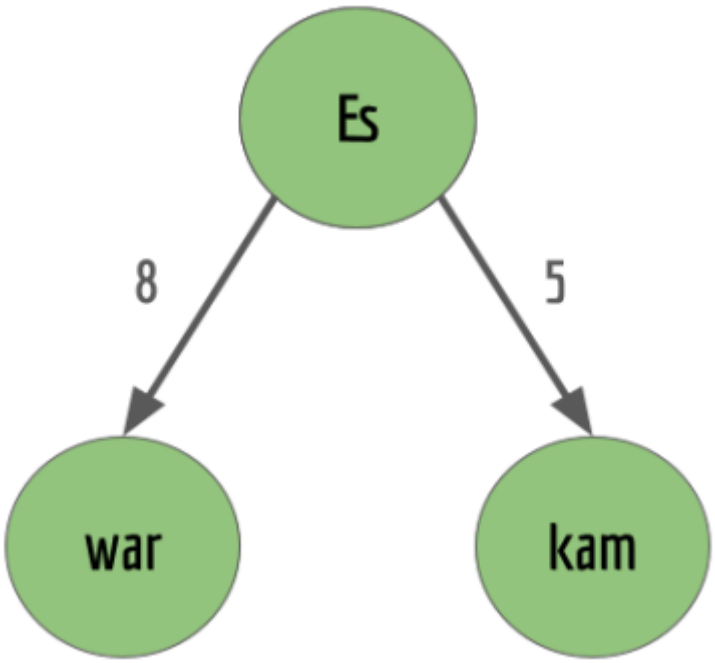


Alle denkbaren Bäume (mit z.B. mehr Blättern pro Knoten) lassen sich durch Umhängen von Knoten zu einem Binärbaum umformen, der jedoch dann nicht immer vollständig ist.



Kanten und Gewicht

Eine Kante kann ein Gewicht bekommen, das z.B. angibt, wie wahrscheinlich es ist, dass Daten, die in einem Baum gespeichert sind, aufeinander folgen.



Auf diese Weise könnte in einem Sprachmodell hinterlegt sein, wie wahrscheinlich es ist, welches Wort auf den Satzanfang „Es“ folgt. Hier einmal ein Beispielimplementierung für den letzten Baum.

treedataweighted.py

```
class Node:
    def __init__(self, data) -> None:
        self.left = None
        self.right = None
```

```
        self.leftweight = 0
        self.rightweight = 0
        self.data = data

# Array für die Knoten des Baumes anlegen
nodes[]

# Knoten anlegen
nodes[0] = Node("Es")
nodes[1] = Node("war")
nodes[2] = Node("kam")

# Knoten verbinden
nodes[0].left = nodes[1]
nodes[0].right = nodes[2]

# Gewichte der Kanten setzen
nodes[0].leftweight = 8
nodes[0].rightweight = 5
```

Ein kleines Sprachmodell

Hier sehr ihr ein Beispiel für ein „Sprachmodell“, welches Märchenanfänge generiert und von all dem Gebrauch macht, was wir bisher über Binärbäume besprochen haben. Alle wesentlichen Aktionen, die in großen Sprachmodellen stattfinden, lassen sich hier erleben, z.B. dass immer Interaktion mit Menschen notwendig sind, um ein Sprachmodell zu optimieren.

littlelanguangemodell.py

```
import random

class Node:

    # Konstruktor, nur data muss einen Wert haben
    def __init__(self, data) -> None:
        self.left = None
        self.right = None
        self.leftweight = 0
        self.rightweight = 0
        self.data = data

    # zufällig durch den Binärbaum gehen
    # Daten des jeweiligen Knotens ausgeben
    # Aktuellen Weg in einer Liste merken (false = links, true = right)

    def walkRandom(self):
        print(self.data)
```

```
        select = random.uniform(0,100)
        if select > 50:
            nextNode = self.left
            direction = False
        else:
            nextNode = self.right
            direction = True
        if nextNode:
            way.append(direction)
            nextNode.walkRandom()
        else:
            return

    # Anhand der Gewichte der Kanten durch den Baum gehen
    # Bei gleichem Gewicht zufälligen Knoten wählen

    def walkWeighted(self):
        print(self.data)
        if self.leftweight > self.rightweight:
            nextNode = self.left
        elif self.leftweight < self.rightweight:
            nextNode = self.right
        elif self.leftweight == self.rightweight:
            select = random.uniform(0,100)
            if select > 50:
                nextNode = self.left
            else:
                nextNode = self.right
        if nextNode:
            nextNode.walkWeighted()
        return

    # Anhand des Weges (way) durch den Baum gehen
    # Gewichte der Kanten entsprechend setzen

    def setWeight(self, currentdepth):
        if currentdepth >= len(way):
            return
        if way[currentdepth]:
            self.rightweight += 1
            nextNode = self.right
        else:
            self.leftweight += 1
            nextNode = self.left
        if nextNode:
            nextNode.setWeight(currentdepth+1)
        else:
            return

# Mega unschöne Methode, um die Gewichte im Baum formatiert anzuzeigen
```

```
def displayWeight():
    print("              ", nodes[0].leftweight, nodes[0].rightweight)
    print("            ",nodes[1].leftweight, nodes[1].rightweight,
nodes[2].leftweight, nodes[2].rightweight)
    print("        ", nodes[3].leftweight, nodes[3].rightweight,
nodes[4].leftweight, nodes[4].rightweight, nodes[5].leftweight,
nodes[5].rightweight, nodes[6].leftweight, nodes[6].rightweight)
    print(nodes[7].leftweight, nodes[7].rightweight,
nodes[8].leftweight, nodes[8].rightweight, nodes[9].leftweight,
nodes[9].rightweight, nodes[10].leftweight,
nodes[10].rightweight,nodes[11].leftweight, nodes[11].rightweight,
nodes[12].leftweight, nodes[12].rightweight, nodes[13].leftweight,
nodes[13].rightweight, nodes[14].leftweight, nodes[14].rightweight)
    print()
    return

# data enthält die Daten der Knoten in Reihenfolge der Baumlevel
0,1,1,2,2,2,2 ...
data = ["Es","war einmal", "begab sich zu der Zeit", "ein Müller", "ein
Königssohn", "als Wesen der Erde innewohnten", "der Fantasiewesen",
"der", "welcher", "der", "welcher", "die", "welche", "die", "welche",
"in die Welt zog", "ausging", "in die Welt zog", "ausging", "in die
Welt zog", "ausging", "in die Welt zog", "ausging", "der Fantasie der
Kinder",  "der Fantasie der Kinder", "der Fantasie der Kinder", "der
Fantasie der Kinder", "der Fantasie der Kinder", "der Fantasie der
Kinder", "der Fantasie der Kinder", "der Fantasie der Kinder"]

# nodes ist eine Liste der Knoten
nodes = []

# z ist eine Hilfsvariable zum Aufbau des Binärbaumes
z = 1

# Binärbaum bauen
for i in range(0,len(data)):
    nodes.append(Node(data[i]))

for i in range(0,14):
    nodes[i].left = nodes[i+z]
    nodes[i].right = nodes[i+z+1]
    z+=1

# Mainmethode
while True:
    print()
    choice = int(input("Was willst du tun?\n1: Training\n2: Stand
abrufen\n3: Ende\n\nDeine Wahl: "))
    print()
    if choice == 1:
        print()
```

```
        way = []
        nodes[0].walkRandom()
        print()
        innerchoice = int(input("1: Dieser Anfang ist ok\n2: Dieser
Anfang ist nicht ok\n\nDeine Eingabe: "))
        if innerchoice == 1:
            nodes[0].setWeight(0)
    elif choice == 2:
        displayWeight()
        nodes[0].walkWeighted()
    elif choice == 3:
        break
    else:
        print("Ungültige Eingabe!")
        print()

print("Programmende!")
```

From:
https://schule.riecken.de/ - **Unterrichtswiki**

Permanent link:
https://schule.riecken.de/doku.php?id=informatik:algorithmisch:python:binaerbaeume&rev=1721486221

Last update: **2024/07/20 14:37**