Scripting und Algorithmen

Binärsuche

Harald Lampesberger

FH OÖ, Department für Sichere Informationssysteme
SAL2VO, SS23, Version: 27. Februar 2023

Warum Suchen?

Fundamentale Aufgabe Datenorganisation

- Reminder: Computerspeicher sind historisch linear aufgebaut
- Eine klassische Aufgabe ist die Suche eines Elementes in einer Datenstruktur
- zB Key-Value-Paare oder Datenbank-Records, die sequenziell abgespeichert sind

„Suche“ kann aber auch mehr sein

- Suche nach einer Lösung für ein numerisches Berechnungsproblem
- Suche nach einem git-commit, der einen Bug in das Projekt eingeschleppt hat
- ...

Linear Search

```
primes = [2, 3, 5, 7, 11, 13, 17, 19, 23]

def linear_search(value):
    for idx, elem in enumerate(primes):
        if elem == value:
            return idx
    return None
```

Lineare Datenstruktur wird elementweise durchlaufen

- Definieren wir $n =$ `len(primes)`
- Anzahl der Schleifendurchläufe im best case $= 1$
- Anzahl der Schleifendurchläufe im worst case $= n$
- Geht das vielleicht schneller?

Teile (und Herrsche)
Divide (and Conquer)

Intuition Binärsuche

Intuition: Telefonbuch

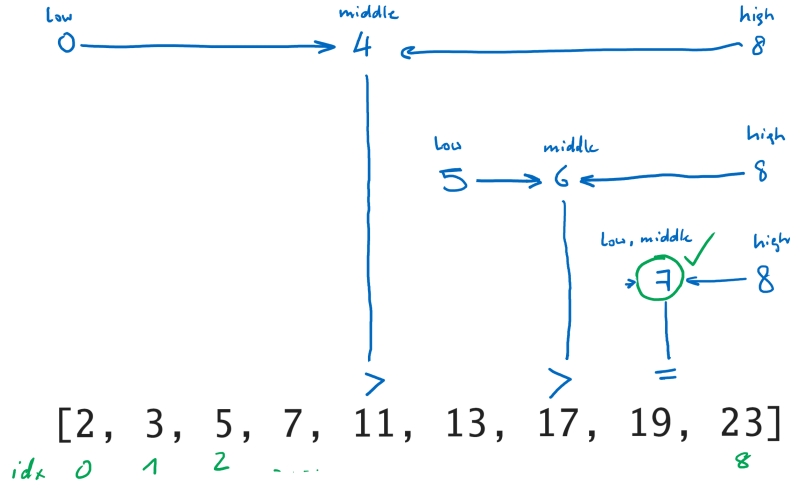
- Wir suchen die Telefonnummer (Value) zu einem Namen (Key), Namen sind sortiert

1. Wir halbieren das Buch; haben wir exakt den Key gefunden sind wir fertig
2. Wenn wir nur mehr eine Seite haben stoppen wir; Key nicht gefunden
3. Ist der gesuchte Key größer als die Hälfte, wiederhole 1. mit der oberen Hälfte
4. Ist der gesuchte Key kleiner als die Hälfte, wiederhole 1. mit der niedrigeren Hälfte

Wir können das wiederholende Teilen/Halbieren als rekursiven Algorithmus beschreiben

Beispiel

Wir suchen den Index von 19



Recursive Binary Search

```
def binary_search(lst, low, high, value):
    middle = (low + high) // 2
    if value == lst[middle]:
        return middle
    elif low == high:
        return None
    elif value >  lst[middle]:
        return binary_search(lst, middle + 1, high, value)
    else:
        return binary_search(lst, low, middle - 1, value)

last_idx = len(primes) - 1
binary_search(primes, 0, last_idx, 19)
```

Diskussion

Binärsuche ist schnell und skaliert super

- Definieren wir n als Länge des Suchraums
- Im worst case sind $\log n$ Schritte notwendig

Suchraum muss sortiert sein

- Vorher sortieren kostet Zeit und Speicher

Rekursive Variante hat Einschränkungen

- Python erlaubt standardmäßig Rekursionstiefe von 1.000
- Jede Rekursion kann aber auf Schleife umprogrammiert werden