3.4 Suchbäume

Bisher stand der Inhalt beliebig im Baum. Ordnet man ihn, entsteht eine Struktur, in der man sehr schnell suchen kann.

Definition: Binärer Suchbaum

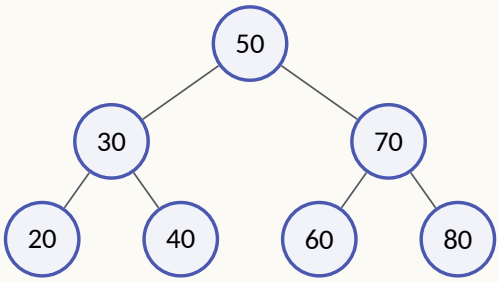
Ein binärer Baum heißt **Suchbaum**, wenn für **jeden** Knoten gilt:

- alle Inhalte im **linken** Teilbaum sind **kleiner** als der Inhalt des Knotens,
- alle Inhalte im **rechten** Teilbaum sind **größer**.

Die Bedingung muss für jeden Knoten gelten, nicht nur für die Wurzel — und für den ganzen Teilbaum, nicht nur für die direkten Nachfolger.

Beispiel: Aufbau durch Einfügen

Fügt man nacheinander 50, 30, 70, 20, 40, 60, 80 ein, entsteht:



Links steht immer das Kleinere, rechts das Größere — und zwar im ganzen Teilbaum.

Suchbaum aus 50, 30, 70, 20, 40, 60, 80 — in dieser Reihenfolge eingefügt.

Gymnasium Ottweiler

Merke

Der **Inorder**-Durchlauf eines Suchbaums liefert die Inhalte in **aufsteigender Reihenfolge**. Für den Baum oben: 20, 30, 40, 50, 60, 70, 80.

Das ist keine Zufälligkeit, sondern die Definition: links steht das Kleinere, also wird es zuerst besucht. Ein Suchbaum sortiert, indem man ihn richtig ausliest.

> In der Webfassung steht hier ein Bedienfeld zum Ausprobieren: **Suchbaum-Werkzeug**.

3.4.1 Suchen

```python
def suchen(knoten, wert):
    if knoten is None:
        return False                    # nicht gefunden
    if wert == knoten.inhalt:
        return True
    if wert < knoten.inhalt:
        return suchen(knoten.links, wert)
    return suchen(knoten.rechts, wert)
```

```
function Suchen(knoten: TBaumZeiger; wert: Integer): Boolean;
begin
  if knoten = nil then
    Result := False
  else if wert = knoten^.inhalt then
    Result := True
  else if wert < knoten^.inhalt then
    Result := Suchen(knoten^.links, wert)
  else
    Result := Suchen(knoten^.rechts, wert);
end;
```

Beispiel: Die 60 suchen

```
50 → 60 ist größer → nach rechts
70 → 60 ist kleiner → nach links
60 → gefunden
```

Drei Vergleiche bei sieben Einträgen. Jeder Vergleich halbiert, was noch zu durchsuchen bleibt — genau wie bei der binären Suche in Abschnitt 4.1.2.

3.4.2 Einfügen

Eingefügt wird immer als neues Blatt. Man sucht die Stelle, an der der Wert *stünde*, wenn er da wäre — und hängt ihn dort an.

```
def einfuegen(knoten, wert):
    if knoten is None:
        return BaumKnoten(wert)          # hier ist der Platz
    if wert < knoten.inhalt:
        knoten.links = einfuegen(knoten.links, wert)
    elif wert > knoten.inhalt:
        knoten.rechts = einfuegen(knoten.rechts, wert)
    return knoten                        # wert == inhalt: nichts tun
```

Stolperstein

Die Reihenfolge beim Einfügen bestimmt die Form des Baums. Fügt man 20, 30, 40, 50, 60, 70, 80 **der Größe nach** ein, hängt jeder Knoten rechts am vorigen — der Baum entartet zur Liste, und die Suche braucht statt drei Vergleichen bis zu sieben.

Der Suchbaum ist also nur so gut wie seine Form. Es gibt Verfahren, die ihn beim Einfügen selbsttätig wieder ausgleichen — AVL-Bäume und Rot-Schwarz-Bäume. Sie stehen in keinem der beiden Lehrpläne, geben aber ein lohnendes Thema für eine Facharbeit ab.

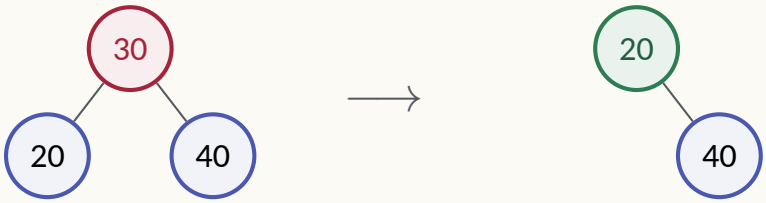
3.4.3 Löschen

Das Löschen ist der einzige unangenehme Fall, weil drei Situationen zu unterscheiden sind:

Fall	Vorgehen
Der Knoten ist ein **Blatt**	einfach entfernen
Der Knoten hat **einen** Nachfolger	dieser rückt samt seinem ganzen Teilbaum nach
Der Knoten hat **zwei** Nachfolger	siehe unten

Im dritten Fall darf der Knoten nicht einfach verschwinden — es muss jemand nachrücken, der die Suchbaumbedingung nicht verletzt. Dafür kommen genau zwei Knoten in Frage: der **größte Knoten des linken Teilbaums** oder der **kleinste des rechten**. Wir nehmen den ersten. Sein Inhalt wandert in den zu löschenden Knoten, und er selbst wird an seiner alten Stelle gelöscht — was leicht ist, denn er hat höchstens einen Nachfolger.

Beispiel: Die 30 löschen



vorher nachher

Die 30 hat zwei Nachfolger. Der größte Knoten ihres linken Teilbaums ist die 20 — sie rückt nach, ihr alter Platz wird frei. Probe über den Inorder-Durchlauf: vorher 20, 30, 40, nachher 20, 40. Aufsteigend, die Bedingung hält.

Löschen eines Knotens mit zwei Nachfolgern.

Merke

Prüfen Sie jede Veränderung an einem Suchbaum mit dem Inorder-Durchlauf. Kommt dabei keine aufsteigende Folge heraus, ist der Baum kaputt.

Arbeitsauftrag 5

13. Bauen Sie einen Suchbaum aus 55, 20, 80, 10, 35, 70, 95, 30, 40 auf. Zeichnen Sie ihn und geben Sie Tiefe und Knotenzahl an.
14. Geben Sie zu Ihrem Baum alle drei Durchläufe an. Welcher ist aufsteigend sortiert?
15. Löschen Sie daraus die 20. Zeichnen Sie das Ergebnis und prüfen Sie es mit dem Inorder-Durchlauf.
16. Implementieren Sie `einfuegen` und `suchen` in einer Sprache Ihrer Wahl und lesen Sie den Baum inorder aus. Sie haben damit nebenbei ein Sortierverfahren geschrieben.
17. In welcher Reihenfolge müssten Sie 1 bis 7 einfügen, damit ein möglichst *flacher* Baum entsteht? Welche Tiefe erreichen Sie?

Selbstcheck

Können Sie ohne nachzusehen angeben, welcher Durchlauf einen Suchbaum sortiert ausliest — und begründen, warum das aus der Definition folgt? Und können Sie erklären, weshalb die Rekursion auf Bäumen so viel natürlicher wirkt als auf Feldern?