

# Abgabe ADS2

## Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig und ohne die Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen - einschließlich Tabellen, Karten, Abbildungen etc., die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Werken und Quellen (dazu zählen auch Internetquellen) entnommen wurden, sind in jedem einzelnen Fall mit exakter Quellenangabe kenntlich gemacht worden. Zusätzlich versichere ich, dass ich beim Einsatz von generativen IT-/KI-Werkzeugen (z.B. ChatGPT, BARD, Dall-E oder Stable Diffusion) diese Werkzeuge in einer Rubrik "Übersicht verwendeter Hilfsmittel" mit ihrem Produktnamen, der Zugriffsquelle (z. B. URL) und Angaben zu genutzten Funktionen der Software sowie Nutzungsumfang vollständig angeführt habe. Wörtliche sowie paraphrasierende Übernahmen aus Ergebnissen dieser Werkzeuge habe ich analog zu anderen Quellenangaben gekennzeichnet. Mir ist bekannt, dass es sich bei einem Plagiat um eine Täuschung handelt, die gemäß der Prüfungsordnung sanktioniert werden wird. Ich versichere, dass ich die vorliegende Arbeit oder Teile daraus nicht bereits anderweitig innerhalb und außerhalb der Hochschule als Prüfungsleistung eingereicht habe.

## Implementierung einer effizienten Datenstruktur für das Predecessor-Problem

### Algorithmen und Datenstrukturen 2

Fachhochschule Bielefeld – Campus Minden  
Bachelor of Computer Science  
Wintersemester 2025/2026  
Prof. Dr. Jörg Brunsmann

## Teammitglieder

- Clemens Maas, 1260892

## Inhaltsverzeichnis

1. Einleitung
2. Evaluation existierender Lösungsansätze
3. Konzept der Datenstruktur
4. Implementierung
5. Optimierungen/Varianten (optional)
6. Tests und Evaluation
7. Benutzerdokumentation
8. Lessons Learned
9. Ausblick
10. Literaturverzeichnis
11. Anhang: Hilfsmittel

## 1. Einleitung

Das Predecessor-Problem ist ein grundlegendes Problem der Informatik: Gegeben eine dynamische Menge von Ganzzahlen, finde effizient den größten Wert, der kleiner oder gleich einem gegebenen Query-Wert ist.

Typische Anwendungen:

- Datenbanken: Range Queries und Indexstrukturen
- Netzwerke: Routing-Tabellen
- Algorithmen: Sweep-Line-Algorithmen / Event Scheduling

## Aufgabenstellung

Implementierung einer Datenstruktur für eine geordnete Menge von Ganzzahlen im Bereich ( $[0, 2^w - 1]$ ) mit folgenden Operationen:

Operation	Beschreibung	Rückgabewert
insert(x)	Fügt Wert x ein	void
delete(x)	Löscht Wert x	void
search(x)	Prüft ob x vorhanden	bool
indexOf(x)	Liefert Index von x	i oder -1
getAt(i)	Liefert Wert an Index i	x oder null

Operation	Beschreibung	Rückgabewert
deleteAt(i)	Löscht Wert an Index i	x oder null
insertAt(i, x)	Fügt x an Index i ein	void
predecessor(z)	Größter Wert $\leq z$ (inkl. Index)	(i, x) oder -1, -1
successor(z)	Kleinster Wert $\geq z$ (inkl. Index)	(i, x) oder -1, -1
min()	Minimaler Wert	x oder null
max()	Maximaler Wert	x oder null
isEmpty()	Prüft ob leer	bool
size()	Anzahl Elemente	n

## 2. Evaluation existierender Lösungsansätze

### 2.1 Sortiertes Array

- Insert/Delete:  $O(n)$  (Verschieben)
  - Search/Pred/Succ:  $O(\log n)$
  - getAt:  $O(1)$
  - Speicher:  $O(n)$
- Bewertung: ☷ Ungünstig bei häufigen Updates.

### 2.2 Hash Table

- Insert/Delete/Search:  $O(1)$  erwartet
  - Pred/Succ/getAt: nicht effizient (keine Ordnung)
- Bewertung: ☷ Für das Predecessor-Problem ungeeignet.

### 2.3 (Unbalancierter) BST

- durchschnittlich:  $O(\log n)$
  - Worst-Case:  $O(n)$
- Bewertung: △ Keine Worst-Case-Garantie.

### 2.4 AVL-Tree

- $O(\log n)$  garantiert (strenger balanciert)
- Bewertung: ☹ Sehr gut, aber höhere Implementierungskomplexität.

### 2.5 Red-Black Tree

- Insert/Delete/Search/Pred/Succ:  $O(\log n)$  garantiert
  - getAt/indexOf mit Order Statistics:  $O(\log n)$
  - Speicher:  $O(n)$
- Bewertung: ☷ Gewählt – gute Balance aus garantierter Laufzeit und Implementierbarkeit.

### 2.6 van Emde Boas Tree

- $O(\log \log u)$ , aber Speicher  $O(u)$
- Bewertung: ☷ für w=32/64 praktisch zu groß.

---

## 3. Konzept der Datenstruktur

### 3.1 Red-Black Tree mit Order Statistics

Gewählter Ansatz: Augmentierter Red-Black Tree, bei dem jeder Knoten die Größe seines Teilbaums speichert (`size`).

#### 3.1.1 Node-Struktur (konzeptionell)

```

const Node = struct {
    value: T,
    color: Color,      // Red/Black
    left: ?*Node,
    right: ?*Node,
    parent: ?*Node,
    size: usize,        // Teilbaumgröße
};

}

```

### 3.1.2 Invarianten

Red-Black Eigenschaften (klassisch):

1. Root ist schwarz
2. Rote Knoten haben schwarze Kinder
3. Alle Pfade von einem Knoten zu null-Blättern enthalten gleich viele schwarze Knoten

Order Statistics:

- node.size = 1 + size(left) + size(right)
- Größen müssen bei Insert/Delete/Rotation korrekt aktualisiert werden.

### 3.1.3 Pseudocode (zentral)

Select (getAt):

```

SELECT(node, i):
    left_size = SIZE(node.left)
    if i == left_size: return node
    if i < left_size: return SELECT(node.left, i)
    return SELECT(node.right, i - left_size - 1)

```

Rank (indexOf):

```

RANK(node):
    rank = SIZE(node.left)
    current = node
    while current.parent != null:
        if current == current.parent.right:
            rank += 1 + SIZE(current.parent.left)
        current = current.parent
    return rank

```

## 4. Implementierung

### 4.1 Technologie-Stack

- Programmiersprache: Zig 0.15 (0.15.2 beim Language Server)
- Single-threaded, explizite Speicherverwaltung via Allocator

### 4.2 Modul-Struktur (relevant)

Die Hauptdatenstruktur ist `src/rb_tree.zig`. Zusätzliche Dateien (z. B. Varianten/Experimente) sind optional und nicht notwendig, um die ADT-Funktionalität zu erfüllen.

### 4.3 Wichtige Design-Entscheidungen

1. Generische Implementierung: `RedBlackTree(comptime T: type)`
2. Explizites Memory Management über `Allocator`
3. Min/Max Caching (`min_cache`, `max_cache`) für O(1) Zugriff auf Minimum/Maximum.
4. `size`-Tracking pro Knoten für `getAt` / `indexOf`.

## 5. Optimierungen/Varianten (optional)

Dieser Abschnitt beschreibt optionale Erweiterungen/Varianten, die nicht zwingend Teil der Basislösung sind.

## 5.1 Node Pooling (Variante)

Eine mögliche Variante ist Node Pooling (Freelist), um bei Insert/Delete-Zyklen weniger oft `allocator.create/destroy` aufzurufen. Diese Technik kann konstante Faktoren in update-lastigen Workloads verbessern, ändert aber die asymptotischen Komplexitäten nicht.

## 5.2 Bulk-/SIMD-Ideen

Bulk-Operationen und SIMD sind potenziell sinnvoll bei sehr speziellen Workloads, stehen aber nicht im Fokus der geforderten ADT-Operationen und der Evaluation in Abschnitt 6.

# 6. Tests und Evaluation

## 6.1 Unit Tests (Kurzüberblick)

Wir testen u. a. Insert/Delete/Search sowie Index-Operationen und Predecessor/Successor inkl. Randfällen (leerer Baum, 1 Element, Out-of-Bounds).

Aufruf der Tests mit:

```
zig run test
```

## 6.2 Evaluation (Aufgabe 1.3)

**Ziel:** Laufzeit und genutzten Speicherplatz messen, grafisch darstellen und mit der Theorie vergleichen.

### 6.2.1 Messmethodik

- Benchmark erzeugt `bench.csv`.
- Mehrere Samples pro Datensatzgröße  $n$  (Median + Streuung) für stabilere Aussagen.
- Plots werden mit Python/Matplotlib erzeugt.
- Warmup um die Streuung zu minimieren.

### 6.2.2 Laufzeit-Plots

- `plots/time_search_hit.png`
- `plots/time_predecessor.png`
- `plots/time_cycles_insert_delete.png`

**Erwartung (Theorie):** Für Search/Predecessor/Updates erwarten wir  $O(\log n)$  pro Operation aufgrund der Baumhöhe eines balancierten Suchbaums. **Beobachtung (Messung):** Die gemessene `ns/op` wächst deutlich langsamer als linear mit  $n$  und ist konsistent mit dem erwarteten logarithmischen Verhalten.

Zusätzlich werden Speedup-Grafiken als „paired speedup“ dargestellt, um Varianz zu reduzieren:

- `plots/speedup_search_hit_paired.png`
- `plots/speedup_predecessor_paired.png`
- `plots/speedup_cycles_paired.png`

### 6.2.3 Speicher-Plot ( $O(n)$ -Check)

- `plots/memory_insert_peak.png`

**Erwartung (Theorie):**  $O(n)$  Speicher, da pro Element ein Node existiert. **Beobachtung (Messung):** Der Peak-Speicher wächst proportional zu  $n$  und folgt der Referenzlinie  $n * \text{sizeof}(\text{Node})$ , wodurch  $O(n)$  bestätigt wird.

# 7. Benutzerdokumentation

## 7.1 Kompilierung

```
zig build
```

Optional (optimiert):

```
zig build -Doptimize=ReleaseFast
```

## 7.2 Verwendung (Beispiel | Deutliches Beispiel ist in `main.zig` vorhanden)

```

const std = @import("std");
const rb = @import("rb_tree");

pub fn main() !void {
    var gpa = std.heap.GeneralPurposeAllocator(.{}){};
    defer _ = gpa.deinit();

    var tree = rb.RedBlackTree(i32).init(gpa.allocator());
    defer tree.deinit();

    try tree.insert(10);
    try tree.insert(5);
    try tree.insert(15);

    if (tree.search(10)) {
        std.debug.print("Found 10!\n", .{});
    }

    if (tree.getValueAt(1)) |value| {
        std.debug.print("Element at index 1: {}\n", .{value});
    }

    if (tree.predecessor(12)) |pred| {
        std.debug.print("Predecessor of 12: index={}, value={}\n", .{pred, pred});
    }
}

```

Hinweis: predecessor liefert ein Tupel { i64, T } .

### 7.3 Benchmarks ausführen und visualisieren

```

zig build bench
python scripts/bench_plot.py

```

Die Plots werden unter plots/ abgelegt, relativ vom Pfad aus dem man das Skript startet.

python dependencies installierbar mit

```

pip install -r scripts/requirements.txt

```

### 7.4 Tests ausführen

```

zig build test --summary-all

```

### 7.5 Beispiel aufrufen (oder direkt als Binary nach dem bauen)

```

zig build run

```

## 8. Lessons Learned

- Korrekte size-Updates sind essenziell (insbesondere bei Rotationen), sonst brechen getAt / indexOf .
- Microbenchmarks haben sehr oft eine Streuung; mehrere Samples und robuste Kennzahlen waren notwendig um eine gute Visualisierung zu bekommen.
- RB-Tree Delete-Fixup ist der fehleranfälligste Teil; Tests auf Randfälle sind entscheidend. (Hier habe ich auch die meiste Zeit verbracht)
- Immer mal wieder den Zig-Cache leeren bringt etwas. Viele Fehler haben sich dadurch behoben.

## 9. Ausblick

- Weitere Workloads (z. B. gemischte Operationen mit festen Anteilen) könnten die Messungen näher an reale Anwendungen bringen.
- Optional: Node Pooling Variante könnte konstante Faktoren bei Insert/Delete-Zyklen reduzieren.

## 10. Quellen

Quellen habe ich beim aufrufen + nachtraeglich aus meinem Browser kopiert und Minimax zum formatieren gegeben:

Kategorie · Thema	Link
Grundproblem · Predecessor problem	<a href="https://en.wikipedia.org/wiki/Predecessor_problem">https://en.wikipedia.org/wiki/Predecessor_problem</a>
Theorie/Survey · Optimal Bounds for the Predecessor Problem (Beame & Fich)	<a href="https://homes.cs.washington.edu/~beame/papers/stocpred.pdf">https://homes.cs.washington.edu/~beame/papers/stocpred.pdf</a>
Theorie/Survey · Predecessor Search (PDF)	<a href="https://repositorio.uchile.cl/bitstream/handle/2250/178791/Predecessor-Search.pdf">https://repositorio.uchile.cl/bitstream/handle/2250/178791/Predecessor-Search.pdf</a>
BST-Grundlagen · Binary search tree	<a href="https://en.wikipedia.org/wiki/Binary_search_tree">https://en.wikipedia.org/wiki/Binary_search_tree</a>
Balancierte BST · AVL tree	<a href="https://en.wikipedia.org/wiki/AVL_tree">https://en.wikipedia.org/wiki/AVL_tree</a>
Balancierte BST · Red-black tree	<a href="https://en.wikipedia.org/wiki/Red%20black_tree">https://en.wikipedia.org/wiki/Red%20black_tree</a>
Mehrweg-Bäume · B-tree	<a href="https://en.wikipedia.org/wiki/B-tree">https://en.wikipedia.org/wiki/B-tree</a>
Mehrweg-Bäume · B-Trees (Lecture Notes)	<a href="https://www.cs.gordon.edu/courses/cs321/lectures/Btrees.html">https://www.cs.gordon.edu/courses/cs321/lectures/Btrees.html</a>
Integer-Strukturen · van Emde Boas tree	<a href="https://en.wikipedia.org/wiki/Van_Emde_Boas_tree">https://en.wikipedia.org/wiki/Van_Emde_Boas_tree</a>
Integer-Strukturen · X-fast trie	<a href="https://en.wikipedia.org/wiki/X-fast_trie">https://en.wikipedia.org/wiki/X-fast_trie</a>
Integer-Strukturen · Y-fast trie	<a href="https://en.wikipedia.org/wiki/Y-fast_trie">https://en.wikipedia.org/wiki/Y-fast_trie</a>
Integer-Strukturen · Predecessor Data Structures (Slides/PDF)	<a href="https://massivedatasets.wordpress.com/wp-content/uploads/2011/02/predecessor_2011.pdf">https://massivedatasets.wordpress.com/wp-content/uploads/2011/02/predecessor_2011.pdf</a>
Randomized · Skip list	<a href="https://en.wikipedia.org/wiki/Skip_list">https://en.wikipedia.org/wiki/Skip_list</a>
Randomized · Treap	<a href="https://en.wikipedia.org/wiki/Treap">https://en.wikipedia.org/wiki/Treap</a>
Order Statistics · CLRS Ch. 14 (Augmenting Data Structures)	<a href="https://euroinformatica.ro/documentation/programming/!!!Algorithms_CORMEN!!!/DDU0081.html">https://euroinformatica.ro/documentation/programming/!!!Algorithms_CORMEN!!!/DDU0081.html</a>
Order Statistics · CLRS Ch. 14 (PDF Notes)	<a href="https://sites.math.rutgers.edu/~ajl213/CLRS/Ch14.pdf">https://sites.math.rutgers.edu/~ajl213/CLRS/Ch14.pdf</a>
Order Statistics · CLRS 14.1 (Solutions)	<a href="https://walkccc.me/CLRS/Chap14/14.1/">https://walkccc.me/CLRS/Chap14/14.1/</a>
Benchmarking · Reducing variance (Google Benchmark)	<a href="https://github.com/google/benchmark/blob/main/docs/reducing_variance.md">https://github.com/google/benchmark/blob/main/docs/reducing_variance.md</a>
Benchmarking · Random interleaving (Google Benchmark)	<a href="https://google.github.io/benchmark/random_interleaving.html">https://google.github.io/benchmark/random_interleaving.html</a>
Benchmarking · User guide (Google Benchmark)	<a href="https://google.github.io/benchmark/user_guide.html">https://google.github.io/benchmark/user_guide.html</a>
Zig · Language Reference	<a href="https://ziglang.org/documentation/master/">https://ziglang.org/documentation/master/</a>
Zig · Standard Library docs	<a href="https://ziglang.org/documentation/master/std/">https://ziglang.org/documentation/master/std/</a>
Zig Bench · doNotOptimizeAway (Ziggit thread)	<a href="https://ziggit.dev/t/how-to-use-std-mem-donotoptimizeaway/4677">https://ziggit.dev/t/how-to-use-std-mem-donotoptimizeaway/4677</a>
Zig Bench · Constant folding pitfall (blog)	<a href="https://pyk.sh/blog/2025-12-08-bench-fixing-constant-folding">https://pyk.sh/blog/2025-12-08-bench-fixing-constant-folding</a>

Kategorie	Thema	
Zig 0.15 T/O	Reader/Writer overhaul	<a href="https://dev.to/bkataru/zig-0151-io-overhaul-understanding-the-new-readerwriter-interfaces-30oe">https://dev.to/bkataru/zig-0151-io-overhaul-understanding-the-new-readerwriter-interfaces-30oe</a>
Matplotlib	fill_between demo	<a href="https://matplotlib.org/stable/gallery/lines_bars_and_markers/fill_between_demo.html">https://matplotlib.org/stable/gallery/lines_bars_and_markers/fill_between_demo.html</a>

## 11. Anhang: Hilfsmittel

### KI-Werkzeuge

#### MiniMax 2.1 (lokal gehostetes LLM)

- Wurde nicht zur direkten Code-Generierung benutzt
- Genutzte Funktionen:
  - Debugging-Hilfe (Fehler-Analyse)
  - Verbesserung der Code-Kommentierung und Dokumentation (Rechtschreibfehler + "bessere Sprache")
  - Ideenfindung und "Diskussion" zu Optimierungen
- Nutzungsumfang:
  - iterativ während Implementierung und Evaluation

### Software/Tools

- Zig (v0.15 / zls v0.15.2)
- Python 3 + Matplotlib
- Git
- TeamCity (CI-Pipeline im lokalen Netzwerk)