Foundations of Information Retrieval

& Web Search

Artifact: Localized B-Tree Index Construction

Course Code: 4594144453
Semester: 1403/1404 (Sem 2)

Engineered by:

Seyed Parsa Qazi MirSaeed

Department of Computer Engineering

Supervisor:

Dr. Hadi Saboohi

Assistant Professor
Islamic Azad University, Karaj Branch
(Postdoctoral Research Fellow, University of Malaya)

December 6, 2025

Context & Objectives

This technical report documents the implementation of a high-performance Index Construction engine (v0.2.0). It is designed to complement the theoretical framework provided in Manning, Raghavan, and Schütze (2008).

Specific alignment with Course Syllabus:

Chapter 2 (Term Vocabulary): I addressed the specific challenge of Persian token normalization and sorting (e.g., handling پ, چ, ژ, گ).

Chapter 3 (Tolerant Retrieval): I implemented a B-Tree dictionary structure to support efficient term lookup and range-based queries for wildcard search support.

Chapter 4 (Index Construction): I implemented a dynamic B-Tree structure (Order 4) to demonstrate scalable dictionary maintenance and deterministic node splitting.

System Architecture

The system architecture distinguishes between the Logical Layer (Algorithms) and the Presentation Layer (Visualization), adhering to ISO/IEC 25010 software quality standards.

1. Algorithmic Core

The implementation adheres to the strict definition of B-Trees suitable for database indexing (Knuth, Vol. 3):

Order $m = 4$: Each node holds a maximum of 3 keys and 4 children.

Split Strategy: To ensure deterministic behavior, I utilized the Upper Median Promotion strategy (Pivot = Keys[$\lfloor$N/2$\rfloor$]).

2. The Persian Normalization Challenge

Standard Unicode sorting fails for Persian IR because it treats characters based on their binary code point rather than their linguistic weight. I engineered a NaturalString comparator to map characters to weighted tables, ensuring پ follows ب.

Implementation Engineering

The system is constructed in Rust (2021 Edition). The choice of language is a deliberate engineering decision to ensure System Reliability (ISO/IEC 25010).

Why Rust? (Engineering Justification)

B-Trees are recursive data structures heavily reliant on pointer manipulation.

Memory Safety without GC: Unlike Java or Python, Rust guarantees memory safety at compile time via the Borrow Checker. This prevents “Null Pointer Exceptions” and “Dangling Pointers” without the runtime latency spikes caused by Garbage Collection.

Strict Type System: I utilize Rust's `Option<Box<Node>>` to strictly enforce the presence or absence of child nodes, making invalid tree states unrepresentable.

WebAssembly (WASM): Rust compiles directly to WASM, allowing this high-performance backend to run natively in the client's browser (Zero-Server Architecture).

Core Logic Code

Below is the implementation showing the strict typing and natural sort logic.

```rust
// The compiler guarantees memory safety for these pointers
#[derive(Clone, Debug, PartialEq)]
struct Node<K: Ord + Clone + Debug + PartialEq + 'static> {
keys: Vec<K>,
children: Vec<Node<K>>, // Recursive strict ownership
}

text
// 1. SMART KEY (Persian & Natural Sort)
impl NaturalString {
    // Helper to assign alphabetical weight to Persian characters
    fn get_char_weight(c: char) -> u32 {
        match c {
            'آ' => 1, 'ا' => 2, 'ب' => 3,
            'پ' => 4,  // Pe comes strictly after Be
            'ت' => 5,  // Te comes strictly after Pe
            'چ' => 8,  // Che comes after Jim
            'ژ' => 15, // Zhe comes after Ze
            'گ' => 27, // Gaf comes after Kaf
            _ => c as u32 + 1000, // Non-Persian: keep Unicode order
        }
    }
}
```

Listing 1: Strictly Typed B-Tree Node in Rust with Persian Collation

Standards & Compliance

This implementation is compliant with the following international standards:

ISO/IEC 10646 (Unicode)

Character encoding and collation weights for Persian script. The NaturalString type maps Persian characters to linguistic weights (1–33 for the Persian alphabet), ensuring correct sorting order independent of Unicode code points.

ISO/IEC 25010 (Software Quality)

The system prioritizes:

Reliability: Memory safety guaranteed by Rust's ownership model.

Maintainability: Strict type system prevents invalid tree states.

Performance: WebAssembly compilation for near-native execution in browsers.

Knuth Vol. 3 (B-Tree Definition)

Strict adherence to B-Tree structure with Order $m = 4$, ensuring balanced trees and deterministic splitting behavior.

IETF BCP 47 (Language Tags)

Persian language content is tagged with lang=“fa” for accessibility and proper text rendering in Dioxus components.

Verification & Results

The system was verified against complex sorting scenarios.

Test Case: Cultural Sorting

Input: [“پدر”, “برادر”, “تراکتور”] Standard Unicode Sort: [“برادر”, “تراکتور”, “پدر”] (Incorrect — Pe appears at end) Engine Sort (Correct): [“برادر”, “پدر”, “تراکتور”] (Be $\to$ Pe $\to$ Te, linguistically correct)

References

C. D. Manning, P. Raghavan, H. Schütze, Introduction to Information Retrieval, Cambridge University Press, 2008.

H. Saboohi, Lecture Notes: Foundations of Information Retrieval, Islamic Azad University, Karaj Branch, 2024. Available at http://www.hadisaboohi.com/IR/

D. E. Knuth, The Art of Computer Programming, Vol. 3: Sorting and Searching, Addison-Wesley, 1973.

International Organization for Standardization, ISO/IEC 10646: Information Technology — Universal Coded Character Set (UCS), 2023.

International Organization for Standardization, ISO/IEC 25010: Systems and Software Engineering — System and Software Quality Models, 2023.

Internet Engineering Task Force, BCP 47: Tags for Identifying Languages, 2009. RFC 5646.