Writing a grep like program in Zig

Seminar Programming Languages

Tobias Schmitz

University of Siegen

Contents


ZIG

Compiled

General Purpose

Systems Programming Language

Successor to C

Focus on readability

and maintainability

Appeared 2016

Written by Andrew Kelley

Zig Software Foundation (ZSF)

Error Handling

No exceptions

Errors as values

Error sets and unions

Error Handling


```
// inferred error set
pub fn main() !void {
    const num = try parseInt(u32, "324", 10);
}

// named error set
const IntError = error{
    IsNull,
    Invalid,
};
fn checkNull(num: usize) IntError!void {
    if (num == 0) {
        return error.IsNull;
    }
}
```

Error Handling


```
// inferred error set
pub fn main() !void {
    const num = try parseInt(u32, "324", 10);
}

// named error set
const IntError = error{
    IsNull,
    Invalid,
};
fn checkNull(num: usize) IntError!void {
    if (num == 0) {
        return error.IsNull;
    }
}
```

Error Handling


```
// inferred error set
pub fn main() !void {
    const num = try parseInt(u32, "324", 10);
}

// named error set
const IntError = error{
    IsNull,
    Invalid,
};
fn checkNull(num: usize) IntError!void {
    if (num == 0) {
        return error.IsNull;
    }
}
```

Error Handling


```
// inferred error set
pub fn main() !void {
    const num = try parseInt(u32, "324", 10);
}

// named error set
const IntError = error{
    IsNull,
    Invalid,
};
fn checkNull(num: usize) IntError!void {
    if (num == 0) {
        return error.IsNull;
    }
}
```

Error Handling


```
// inferred error set
pub fn main() !void {
    const num = try parseInt(u32, "324", 10);
}

// named error set
const IntError = error{
    IsNull,
    Invalid,
};
fn checkNull(num: usize) IntError!void {
    if (num == 0) {
        return error.IsNull;
    }
}
```

Error Handling


```
// named error set
const IntError = error{
    IsNull,
    Invalid,
};
fn checkNull(num: usize) IntError!void {
    if (num == 0) {
        return error.IsNull;
    }
}

switch (checkNull(value))
  .IsNull ⇒ doOneThing(),
  .Invalid ⇒ doAnotherThing(),
}
```

Error Handling


```
// named error set
const IntError = error{
    IsNull,
    Invalid,
};
fn checkNull(num: usize) IntError!void {
    if (num == 0) {
        return error.IsNull;
    }
}

switch (checkNull(value))
  .IsNull ⇒ doOneThing(),
  .Invalid ⇒ doAnotherThing(),
}
```

Error Handling


```
// named error set
const IntError = error{
    IsNull,
    Invalid,
};
fn checkNull(num: usize) IntError!void {
    if (num == 0) {
        return error.IsNull;
    }
}

switch (checkNull(value))
  .IsNull => doOneThing(),
  .Invalid ⟹ doAnotherThing(),
}
```

Error Handling


```
// named error set
const IntError = error{
    IsNull,
    Invalid,
};
fn checkNull(num: usize) IntError!void {
    if (num == 0) {
        return error.IsNull;
    }
}

switch (checkNull(value))
  .IsNull ⇒ doOneThing(),
  .Invalid => doAnotherThing(),
}
```

Comptime

Language

```
class Container<T>(
    var items: ArrayList<T>,
)

fn Container(comptime T: type) type {
    return struct {
        items: ArrayList(T),
    }
}
```

Comptime


```
class Container<T>(
    var items: ArrayList<T>,
)

fn Container(comptime T: type) type {
    return struct {
        items: ArrayList(T),
    }
}
```

Comptime


```
class Container<T>(
    var items: ArrayList<T>,
)

fn Container(comptime T: type) type {
    return struct {
        items: ArrayList(T),
    }
}
```

Comptime


```
class Container<T>(
    var items: ArrayList<T>,
)

fn Container(comptime T: type) type {
    return struct {
        items: ArrayList(T),
    }
}
```

```
pub fn ArrayList(comptime T: type) type {
    return ArrayListAligned(T, null);
}
```

Comptime


```
fn fibonacci(n: usize) u64 {
    if (n == 0 or n == 1) {
        return 1;
    }
    return fibonacci(n - 1) + fibonacci(n - 2);
}

const FIB_8 = fibonacci(8);

comptime {
    // won't compile
    std.debug.assert(fibonacci(3) == 1);
}
```

Comptime


```
fn fibonacci(n: usize) u64 {
    if (n == 0 or n == 1) {
        return 1;
    }
    return fibonacci(n - 1) + fibonacci(n - 2);
}

const FIB_8 = fibonacci(8);

comptime {
    // won't compile
    std.debug.assert(fibonacci(3) == 1);
}
```

Comptime


```
fn fibonacci(n: usize) u64 {
    if (n == 0 or n == 1) {
        return 1;
    }
    return fibonacci(n - 1) + fibonacci(n - 2);
}

const FIB_8 = fibonacci(8);

comptime {
    // won't compile
    std.debug.assert(fibonacci(3) == 1);
}
```

```
Build Summary: 0/3 steps succeeded; 1 failed (disable with --summary none)
install transitive failure
└─ install zig-demo transitive failure
   └─ zig build-exe zig-demo Debug native 1 errors
zig/0.11.0/files/lib/std/debug.zig:343:14: error: reached unreachable code
    if (!ok) unreachable; // assertion failure
             ^~~~~~~~~~~
src/main.zig:13:21: note: called from here
    std.debug.assert(fibonacci(3) == 1);
    ~~~~~~~~~~~~~~~~^~~~~~~~~~~~~~~~~~~
```

```
Build Summary: 0/3 steps succeeded; 1 failed (disable with --summary none)
install transitive failure
└─ install zig-demo transitive failure
   └─ zig build-exe zig-demo Debug native 1 errors
zig/0.11.0/files/lib/std/debug.zig:343:14: error: reached unreachable code
    if (!ok) unreachable; // assertion failure
             ^~~~~~~~~~~
src/main.zig:13:21: note: called from here
    std.debug.assert(fibonacci(3) == 1);
    ~~~~~~~~~~~~~~~~~^~~~~~~~~~~~~~~~~~
```

```
Build Summary: 0/3 steps succeeded; 1 failed (disable with --summary none)
install transitive failure
└─ install zig-demo transitive failure
   └─ zig build-exe zig-demo Debug native 1 errors
zig/0.11.0/files/lib/std/debug.zig:343:14: error: reached unreachable code
    if (!ok) unreachable; // assertion failure
             ^~~~~~~~~~
src/main.zig:13:21: note: called from here
    std.debug.assert(fibonacci(3) == 1);
    ~~~~~~~~~~~~~~~~~^~~~~~~~~~~~~~~~~~
```

Comptime


```
fn fibonacci(n: usize) u64 {
    if (n == 0 or n == 1) {
        return 1;
    }
    return fibonacci(n - 1) + fibonacci(n - 2);
}

const FIB_8 = fibonacci(8);

comptime {
    // won't compile
    std.debug.assert(fibonacci(3) == 1);
}
```

Comptime


```
fn fibonacci(n: usize) u64 {
    if (n == 0 or n == 1) {
        return 1;
    }
    return fibonacci(n - 1) + fibonacci(n - 2);
}

const FIB_8 = fibonacci(8);

comptime {
    // but this will
    std.debug.assert(fibonacci(3) == 3);
}
```

Defer

Execute code at scope exit

Clean up resources

Defer


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

    var input_paths = ArrayList([]const u8).init(allocator);
    defer input_paths.deinit();

    const pattern = try parseArgs(&input_paths) orelse {
        return;
    };

    // 1. input_paths.deinit();
    // 2. _ = gpa.deinit();
}
```

Defer


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

    var input_paths = ArrayList([]const u8).init(allocator);
    defer input_paths.deinit();

    const pattern = try parseArgs(&input_paths) orelse {
        return;
    };

    // 1. input_paths.deinit();
    // 2. _ = gpa.deinit();
}
```

Defer


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

    var input_paths = ArrayList([]const u8).init(allocator);
    defer input_paths.deinit();

    const pattern = try parseArgs(&input_paths) orelse {
        return;
    };

    // 1. input_paths.deinit();
    // 2. _ = gpa.deinit();
}
```

Defer


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

    var input_paths = ArrayList([]const u8).init(allocator);
    defer input_paths.deinit();

    const pattern = try parseArgs(&input_paths) orelse {
        return;
    };

    // 1. input_paths.deinit();
    // 2. _ = gpa.deinit();
}
```

Defer


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

    var input_paths = ArrayList([]const u8).init(allocator);
    defer input_paths.deinit();

    const pattern = try parseArgs(&input_paths) orelse {
        return;
    };

    // 1. input_paths.deinit();
    // 2. _ = gpa.deinit();
}
```

Defer


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

    var input_paths = ArrayList([]const u8).init(allocator);
    defer input_paths.deinit();

    const pattern = try parseArgs(&input_paths) orelse {
        return;
    };

    // 1. input_paths.deinit();
    // 2. _ = gpa.deinit();
}
```

Program

```
const AtomicQueue = struct {
    mutex: std.Thread.Mutex,
    state: std.atomic.Atomic(State),
    buf: []T,
}
```

Futex: fast userspace mutex

“A futex consists of a kernel-space wait queue that is attached to an atomic integer in userspace”

Synchronization


```
const State = enum(u32) {
    Empty,
    NonEmpty,
    Full,
};

const AtomicQueue = struct {
    mutex: std.Thread.Mutex,
    state: std.atomic.Atomic(State),
    buf: []T,
}
```

Synchronization

Program

```
pub fn append(self *AtomicQueue, item: T) void {
    self.mutex.lock();
    defer self.mutex.unlock();

    if (self.len ⩾ self.buf.len) {
        self.mutex.unlock();
        Futex.wait(&self.state, State.Full);
        self.mutex.lock();
    }

    self.buf.append(item)

    const new_state: State = .NonEmpty;
    self.state.store(new_state, Ordering.SeqCst);
    Futex.wake(&self.state, 1);
}
```

Synchronization

Program

```
pub fn append(self *AtomicQueue, item: T) void {
    self.mutex.lock();
    defer self.mutex.unlock();

    if (self.len ⩾ self.buf.len) {
        self.mutex.unlock();
        Futex.wait(&self.state, State.Full);
        self.mutex.lock();
    }

    self.buf.append(item)

    const new_state: State = .NonEmpty;
    self.state.store(new_state, Ordering.SeqCst);
    Futex.wake(&self.state, 1);
}
```

Synchronization


```
pub fn append(self *AtomicQueue, item: T) void {
    self.mutex.lock();
    defer self.mutex.unlock();

    if (self.len ⩾ self.buf.len) {
        self.mutex.unlock();
        Futex.wait(&self.state, State.Full);
        self.mutex.lock();
    }

    self.buf.append(item)

    const new_state: State = .NonEmpty;
    self.state.store(new_state, Ordering.SeqCst);
    Futex.wake(&self.state, 1);
}
```

Synchronization

Program

```
pub fn append(self *AtomicQueue, item: T) void {
    self.mutex.lock();
    defer self.mutex.unlock();

    if (self.len ≥ self.buf.len) {
        self.mutex.unlock();
        Futex.wait(&self.state, State.Full);
        self.mutex.lock();
    }

    self.buf.append(item)

    const new_state: State = .NonEmpty;
    self.state.store(new_state, Ordering.SeqCst);
    Futex.wake(&self.state, 1);
}
```

Synchronization

Program

```
pub fn append(self *AtomicQueue, item: T) void {
    self.mutex.lock();
    defer self.mutex.unlock();

    if (self.len ⩾ self.buf.len) {
        self.mutex.unlock();
        Futex.wait(&self.state, State.Full);
        self.mutex.lock();
    }

    self.buf.append(item)

    const new_state: State = .NonEmpty;
    self.state.store(new_state, Ordering.SeqCst);
    Futex.wake(&self.state, 1);
}
```

Synchronization

Program

```
pub fn append(self *AtomicQueue, item: T) void {
    self.mutex.lock();
    defer self.mutex.unlock();

    if (self.len ⩾ self.buf.len) {
        self.mutex.unlock();
        Futex.wait(&self.state, State.Full);
        self.mutex.lock();
    }

    self.buf.append(item)

    const new_state: State = .NonEmpty;
    self.state.store(new_state, Ordering.SeqCst);
    Futex.wake(&self.state, 1);
}
```

Synchronization


```
pub fn append(self *AtomicQueue, item: T) void {
    self.mutex.lock();
    defer self.mutex.unlock();

    if (self.len ⩾ self.buf.len) {
        self.mutex.unlock();
        Futex.wait(&self.state, State.Full);
        self.mutex.lock();
    }

    self.buf.append(item)

    const new_state: State = .NonEmpty;
    self.state.store(new_state, Ordering.SeqCst);
    Futex.wake(&self.state, 1);
}
```

Compiler Bug

Program

```
const UserArgFlag = enum {
    Hidden,
    FollowLinks,
    Color,
    NoHeading,
    IgnoreCase,
    Debug,
    NoUnicode,
    Help,
};
```

Compiler Bug

Program

```
const UserArgFlag = enum {
    Hidden,
    FollowLinks,
    Color,
    NoHeading,
    IgnoreCase,
    Debug,
    NoUnicode,
    Help,
};
```

Compiler Bug

Program

```
@@ -27,12 +27,12 @@ const UserArgKind = union(enum) {
     flag: UserArgFlag,
 };

-const UserArgValue = enum {
+const UserArgValue = enum(u8) {
     Context,
     AfterContext,
     BeforeContext,
 };
-const UserArgFlag = enum {
+const UserArgFlag = enum(u8) {
     Hidden,
     FollowLinks,
     Color,
```

Compiler Bug

Program

```
const UserArgFlag = enum {
    Hidden,
    FollowLinks,
    Color,
    NoHeading,
    IgnoreCase,
    Debug,
    NoUnicode,
    Help,
};
```

`0b100` truncated to `0b00`

already fixed on master

Benchmarks

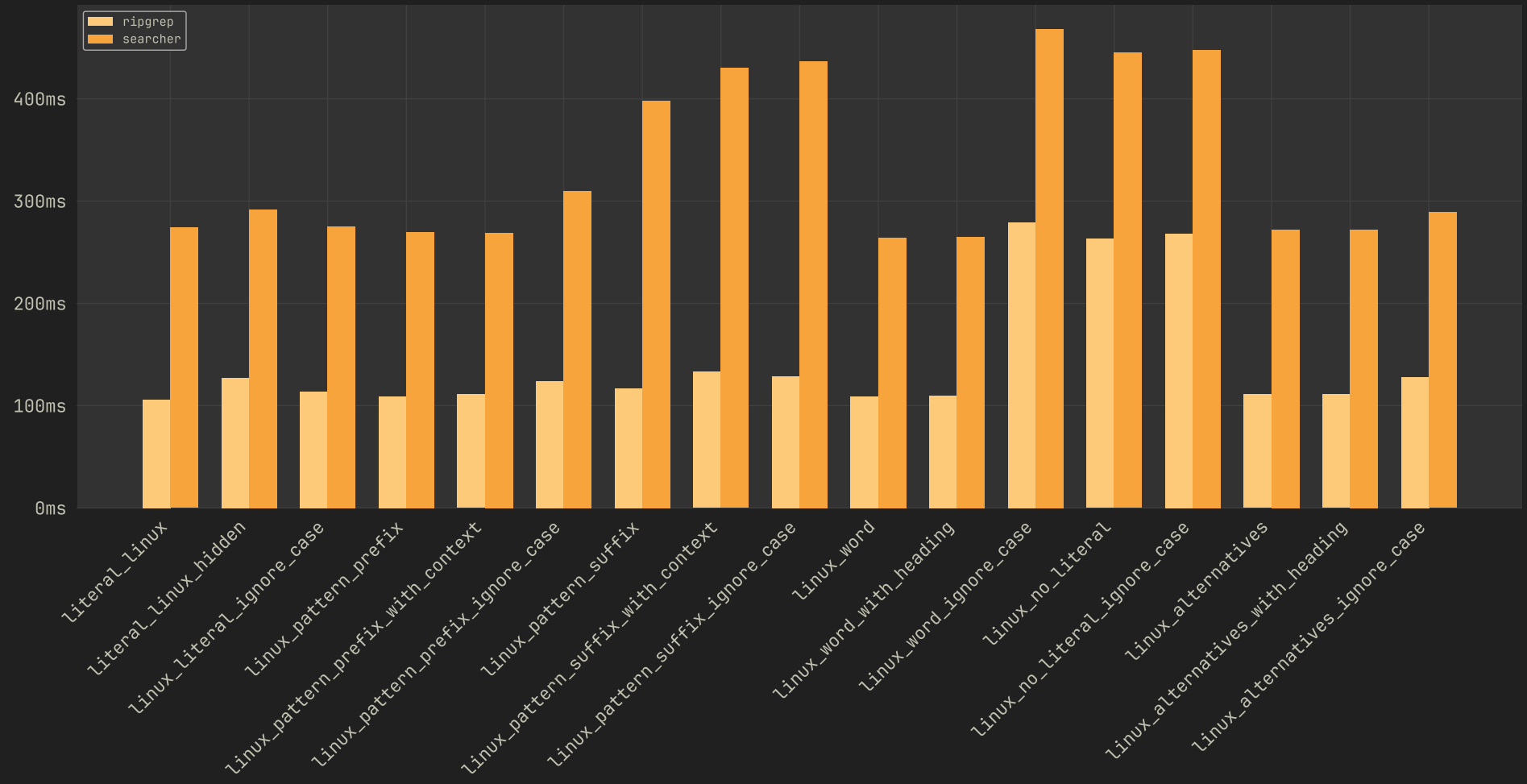
Run using hyperfine

On testsuite

2-3x slower than ripgrep

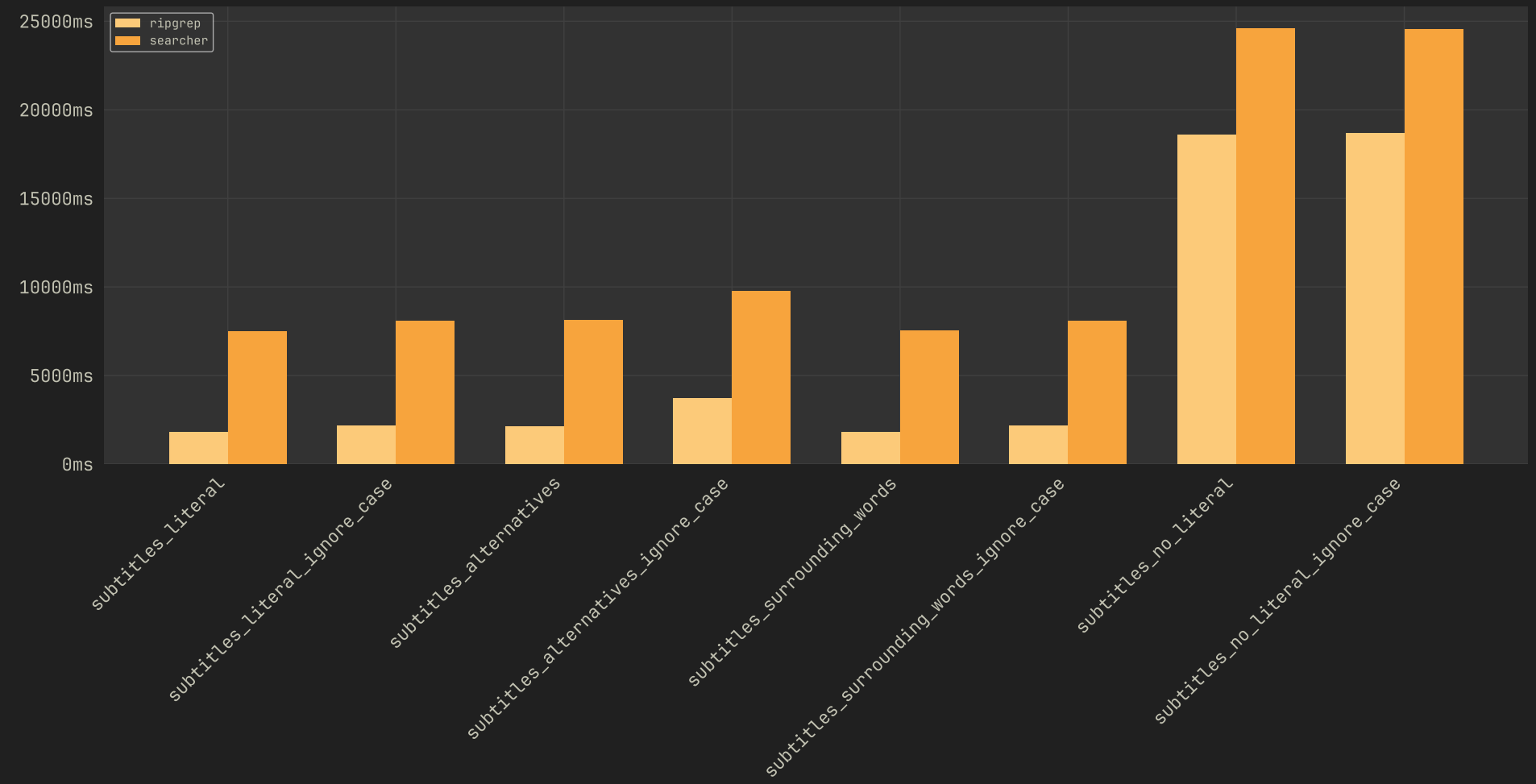
Results R5 5600x & 32Gb

Program



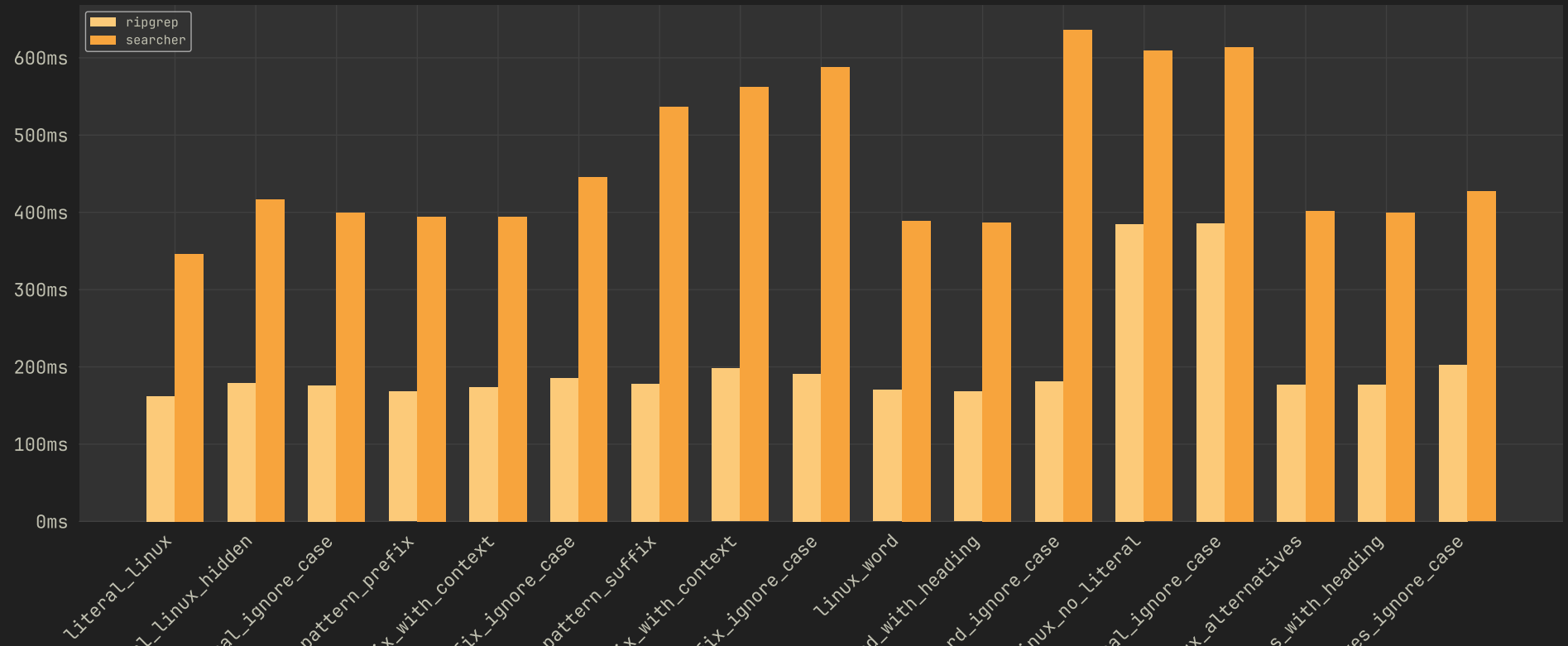
Results R5 5600x & 32Gb

Program



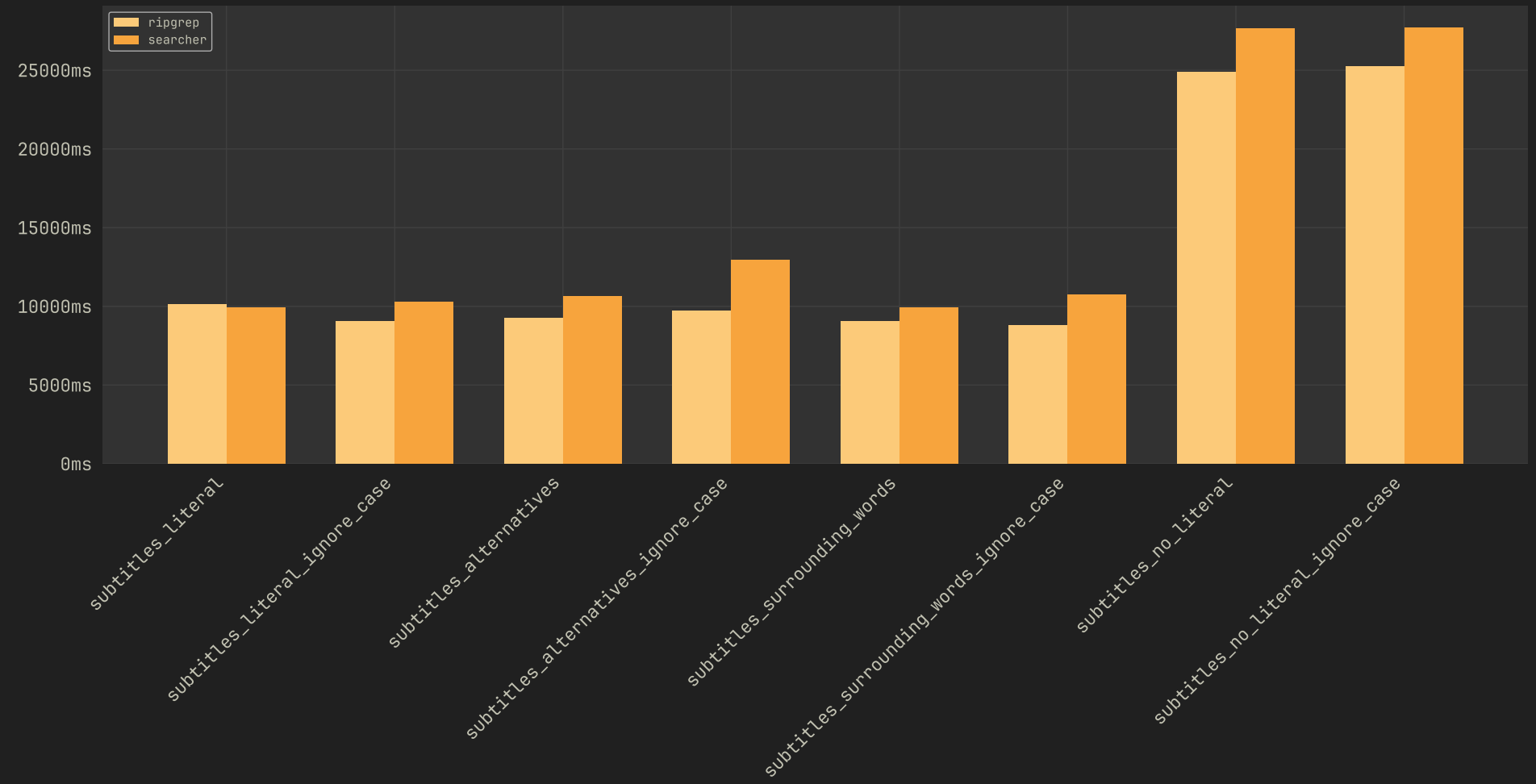
Results R7 5800u & 16Gb

Program



Results R7 5800u & 16Gb

Program



Questions?