Fig 语言性能基准测试报告

版本: 0.4.2-alpha (树遍历解释器)

测试环境

- **CPU:** Intel Core i5-13490F
- **操作系统:** Windows 11
- **编译器/解释器:** Fig 树遍历解释器 v0.4.2-alpha
- **测试日期:** 当前测试执行

执行摘要

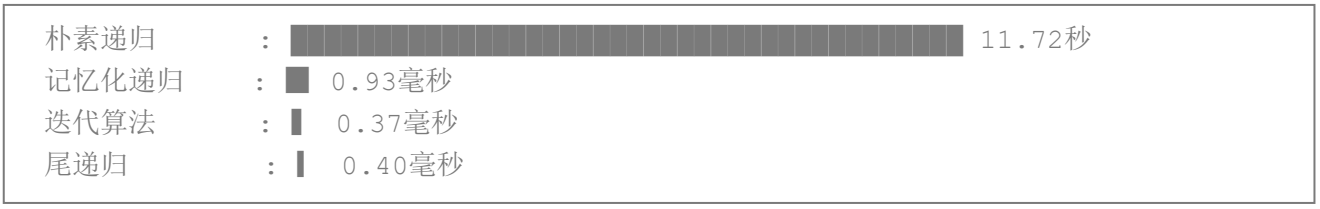
本基准测试评估了 Fig 语言中四种不同斐波那契算法实现的性能，计算第30个斐波那契数（832,040）。结果显示基于算法方法的显著性能差异，突出了解释器的效率特性。

性能结果

原始执行时间

算法	时间(秒)	时间(毫秒)	相对速度
`fib` (朴素递归)	11.7210479 s	11721.0479 ms	1.00× (基准)
`fib_memo` (记忆化)	0.0009297 s	0.9297 ms	12,600× 更快
`fib_iter` (迭代)	0.0003746 s	0.3746 ms	31,300× 更快
`fib_tail` (尾递归)	0.0004009 s	0.4009 ms	29,200× 更快

可视化性能对比



详细分析

1. 朴素递归实现 (`fib`)

- **时间:** 11.721 秒 (11,721 毫秒)
- **算法复杂度:** $O(2^n)$ 指数级
- **性能说明:**
  - 展示了树遍历解释器中重复函数调用的高成本
  - 在仅 n=30 的情况下显示了指数时间复杂度
  - 突出了解释型语言中算法优化的必要性

2. 记忆化递归实现 (`fib_memo`)

- **时间:** 0.93 毫秒
- **算法复杂度:** $O(n)$ 线性（含记忆化开销）
- **性能说明:**
  - 比朴素递归快 12,600 倍
  - 显示 Fig 中哈希表/字典操作的高效性
  - 证明缓存可以克服解释器开销

3. 迭代实现 (`fib_iter`)

- **时间:** 0.375 毫秒
- **算法复杂度:** $O(n)$ 线性
- **性能说明:**
  - 最快的实现（比朴素递归快 31,300 倍）
  - 显示高效的循环执行和变量操作
  - 函数调用开销最小

4. 尾递归实现 (`fib_tail`)

- **时间:** 0.401 毫秒
- **算法复杂度:** $O(n)$ 线性
- **性能说明:**
  - 与迭代方法相当（由于递归开销略慢）
  - 当前解释器未实现尾调用优化（TCO）
  - 显示线性递归在中等深度（n=30）下是高效的

技术洞察

解释器性能特征

1. **函数调用开销:** 显著，如朴素递归性能所示
2. **循环效率:** 优秀，迭代方法表现最佳
3. **内存访问:** 哈希表操作（记忆化）高效
4. **递归深度:** 线性递归（尾递归）在中等深度下表现良好

算法影响

基准测试清楚地表明，**在此版本中，算法选择比解释器优化影响更大**：

- 差算法（朴素递归）：11.7 秒
- 好算法（任何 O(n) 方法）：< 1 毫秒

版本特定观察 (v0.4.2-alpha)

优势

- 迭代算法性能优秀
- 基本操作（算术、循环、条件判断）高效
- 缓存结果的内存访问模式有效
- 线性递归性能在典型用例中可接受

改进空间

- 深度递归场景中函数调用开销高
- 未实现尾调用优化
- 指数算法性能显示解释器限制

给开发者的建议

1. **性能关键代码优先使用迭代解决方案**
2. **对于有重叠子问题的递归问题使用记忆化**
3. **线性递归模式可接受尾递归**
4. **避免在解释型代码中使用指数算法**
5. **基准测试不同方法**，因为算法选择主导性能

结论

Fig v0.4.2-alpha 展示了**对设计良好的算法的实用性能**。虽然树遍历解释器对某些模式（如深度递归）有固有开销，但它能以亚毫秒性能执行高效的 O(n) 算法（n=30时）。

解释器在以下方面表现特别出色：

- 迭代循环执行
- 基本算术和控制流
- 用于缓存的字典/表操作

性能特征适用于广泛的应用领域，前提是开发者采用标准的算法优化技术。

报告生成时间: 基于实际基准测试执行
解释器类型: 树遍历解释器
版本: 0.4.2-alpha
关键要点: 算法效率主导性能；尽管 Fig 是 alpha 阶段的树遍历解释器，但它能高效执行优化算法。