



Guia Rápido de Uso



Início Rápido (3 passos)

1 Compilar o código C

```
make build
```

2 Executar TODOS os testes (12 combinações = 360 execuções)

```
make run-all
```

 **Tempo estimado:** 5-15 minutos

3 Analisar os resultados

```
python3 analyze_results.py
```



Testes Individuais

Executar teste específico

```
# Sintaxe
make run SIZE=<tamanho> CASE=<caso> LANG=<linguagem>

# Exemplos
make run SIZE=small CASE=best LANG=c
make run SIZE=large CASE=worst LANG=python
make run SIZE=medium CASE=best LANG=c
```

Opções:

- SIZE : small (3x3), medium (6x6), large (9x9)
 - CASE : best (poucas células vazias), worst (muitas células vazias)
 - LANG : c , python
-



O que cada comando faz?

make build

- Compila o código C
- Cria diretório `c/bin/` com executável

make run

- Compila C (se necessário)

- Gera um puzzle Sudoku válido
- Executa 30 testes para a combinação escolhida
- Salva resultados em `logs/`

make run-all

- Executa as 12 combinações:
- 3 tamanhos (small, medium, large)
- 2 casos (best, worst)
- 2 linguagens (C, Python)
- Gera 12 arquivos de log

python3 analyze_results.py

- Lê todos os logs em `logs/`
- Extrai estatísticas
- Gera tabelas comparativas
- Calcula speedup C vs Python
- Mostra análise de complexidade

make clean

- Remove arquivos compilados (`c/bin/`)
- Remove todos os logs (`logs/`)

Onde estão os resultados?

Todos os logs são salvos em: `logs/`

Arquivos gerados:

```
logs/
├── c_small_best.log
├── c_small_worst.log
├── c_medium_best.log
├── c_medium_worst.log
├── c_large_best.log
├── c_large_worst.log
├── python_small_best.log
├── python_small_worst.log
├── python_medium_best.log
├── python_medium_worst.log
├── python_large_best.log
└── python_large_worst.log
```

Exemplo de Uso Completo

```
# 1. Limpar tudo (opcional)
make clean

# 2. Compilar
make build

# 3. Executar todos os testes
make run-all

# 4. Analisar resultados
python3 analyze_results.py

# 5. Ver um log específico
cat logs/c_large_worst.log
```

Dicas

✓ **Para relatório:** Use `make run-all` + `analyze_results.py` para obter todos os dados

✓ **Para testes rápidos:** Use `make run` com parâmetros específicos

✓ **Para depuração:** Execute direto:

```
cd c/bin && ./sudoku_solver small best
cd python/src && python3 main.py small best
```

✓ **Ver ajuda completa:** `make help`

Interpretando os Resultados

Arquivo de Log (exemplo: `c_small_best.log`)

```
=== Análise de Complexidade - Backtracking Iterativo para Sudoku ===
Linguagem: C
Tamanho: 3x3
Caso: best
Células vazias alvo: 2

Execução 1:
  Células vazias: 2
  Tempo: 0.000001 segundos
  Iterações: 2
  Resolvido: Sim

[... 30 execuções ...]

=== ESTATÍSTICAS FINAIS ===
Tempo médio: 0.000001 segundos
Iterações médias: 2.00
```

Script de Análise

Mostra:

- ✓ Tabela com todos os resultados
- ✓ Comparação C vs Python (speedup)
- ✓ Análise best case vs worst case
- ✓ Crescimento de iterações

? Problemas Comuns

“command not found: make”

```
# Ubuntu/Debian
sudo apt-get install build-essential

# macOS
xcode-select --install
```

“Python module not found”

```
# O projeto usa apenas biblioteca padrão do Python
# Certifique-se de usar Python 3.x
python3 --version
```

“Permission denied”

```
chmod +x analyze_results.py
```

🎓 Próximos Passos

1. ✓ Execute `make run-all`
2. ✓ Analise com `python3 analyze_results.py`
3. ✓ Use os dados para criar gráficos no relatório
4. ✓ Compare complexidade teórica vs prática
5. ✓ Documente as conclusões

 **Mais informações:** Veja `README.md` completo