MAC0422

# SISTEMAS OPERACIONAIS

## EP2

Thainara de Assis Goulart
13874413

# Como foi feito o controle de acesso à pista? Abordagem Eficiente

Na abordagem eficiente, em vez de proteger toda a matriz da pista com um único mutex global, criamos um mutex por coluna da matriz, ou seja, um vetor de mutex. Assim, ciclistas em colunas diferentes avançam concorrentemente, sem bloquearem uns aos outros desnecessariamente, já que protegemos apenas os dois segmentos envolvidos na movimentação, não toda a pista.

## Estrutura de Dados

- **pthread_mutex_t *mutexPE**; é um vetor de **d** mutexes, um para cada coluna da pista.
- Cada **mutexPE[i]** protege todas as células pista[i][0..MAXFAIXA-1] do segmento i.

**Observação:** na implementação da matriz, a pista é alocada como pista[d][MAXFAIXA], ou seja, o primeiro índice percorre os segmentos/ colunas ao longo da volta (0...d–1) e o segundo índice percorre as faixas na pista (0...MAXFAIXA–1).

# Como foi feito o controle de acesso à pista? Abordagem Eficiente

## Protocolo de travamento em cada passo

Sempre que uma thread-ciclista vai tentar se mover da posição (oi,oj) , para a posição seguinte (ni,nj), ou seja, da posição original para a nova, fazemos:

1. **Travamos o mutex** da coluna de origem **oi**, depois o da coluna de destino **ni**. Como todas as threads seguem essa mesma ordem, evitamos deadlocks.
2. **Atualização da pista**, de acordo com a movimentação. Removemos o ID do ciclista de pista[oi][oj], escrevemos o ID em pista[ni][nj] e atualizamos c->i = ni, c->j = nj.
3. **Liberação dos mutex**. Primeiro liberamos a coluna de destino, depois o de origem.

## Integração com Barreiras

Mesmo usando mutexes por coluna, continuamos a sincronizar cada “tick” usando duas barreiras (partida e chegada), garantindo que nenhuma thread avance até que todas tenham completado seu movimento anterior.

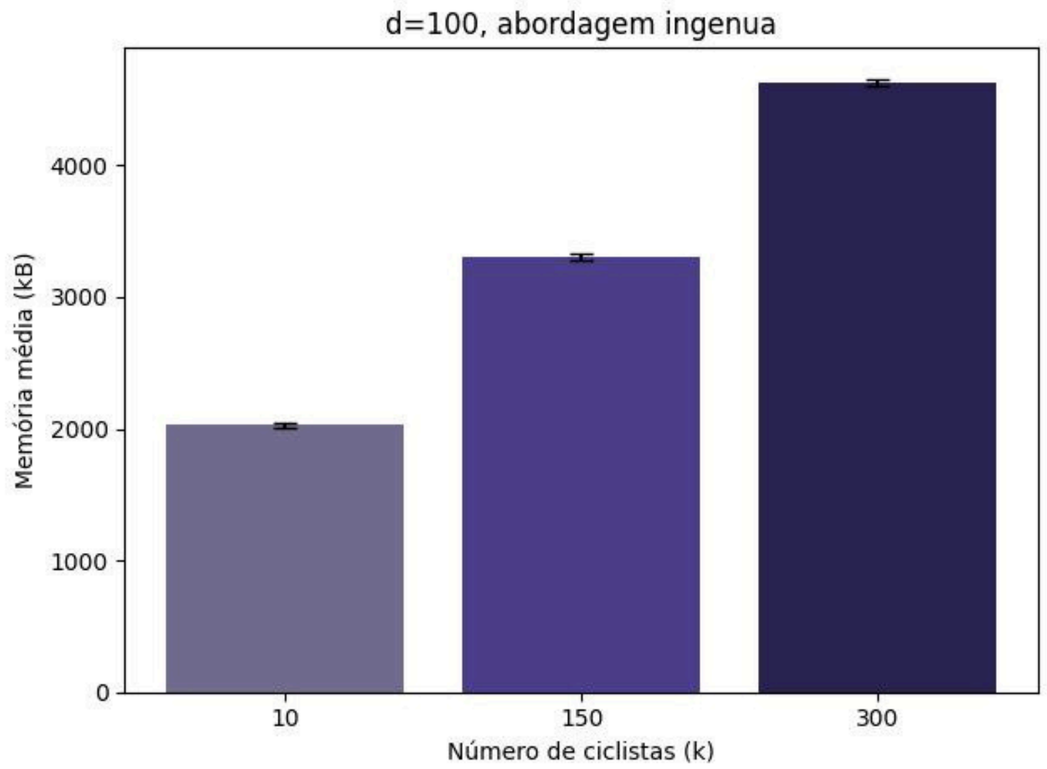
# Resultados

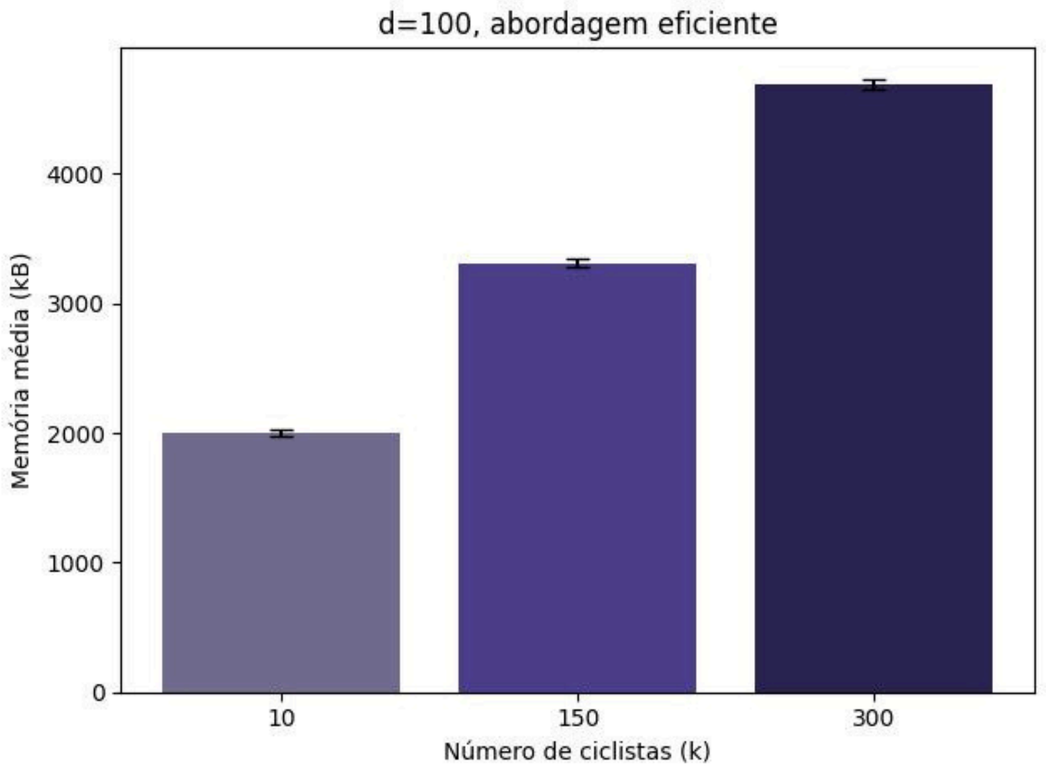
## Configuração da Máquina

| | |
|---|---|
| **Número de CPUs** | • 4 |
| **Modelo do Processador** | • Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz |
| **Arquitetura** | • x86_64 |
| **RAM** | • 8 GB |
| **Sistema Operacional** | • Ubuntu 22.04.1 |

# Resultados

## d = 100
## memória

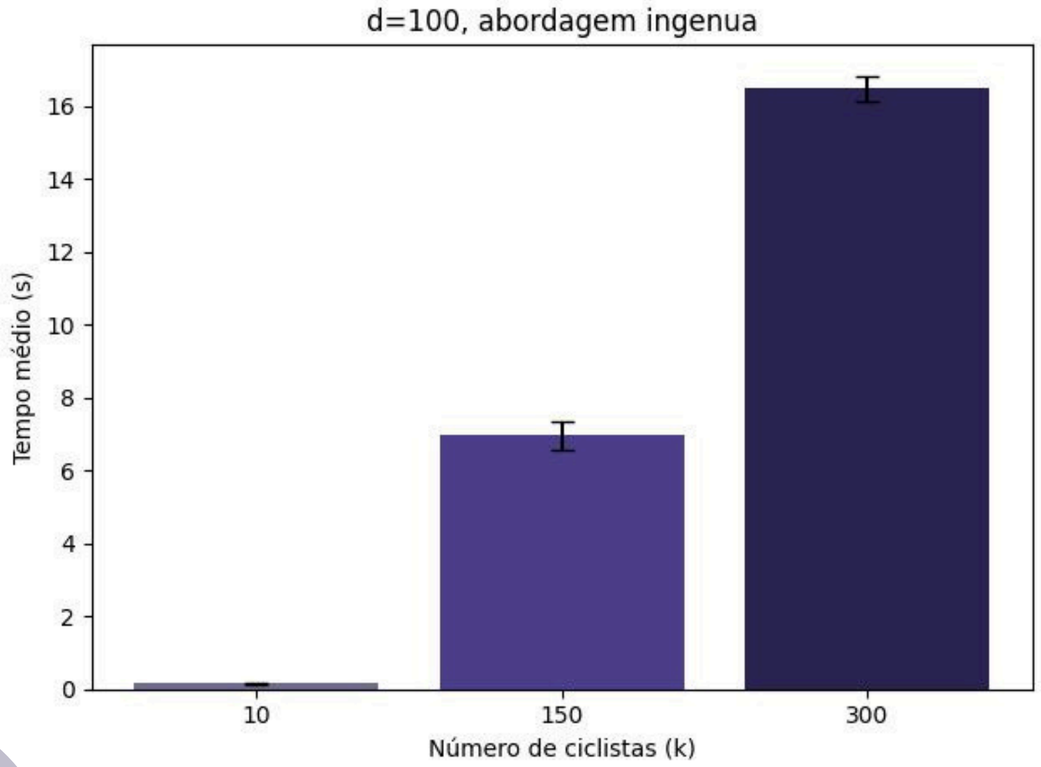


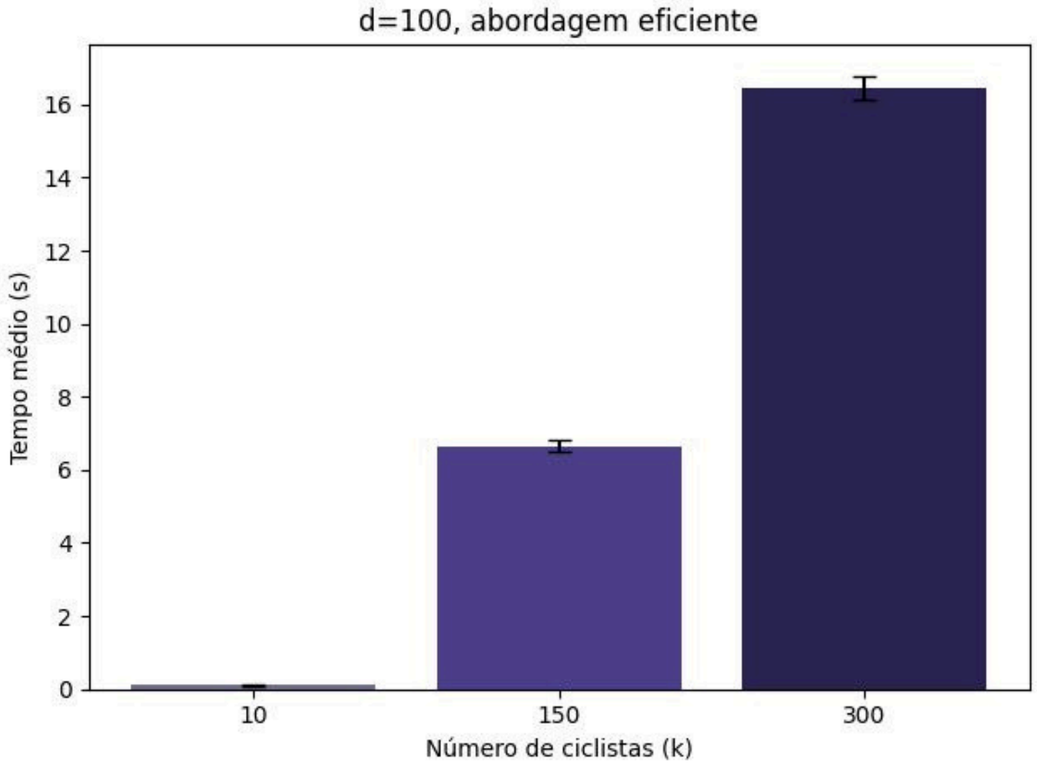



# Resultados

## d = 100

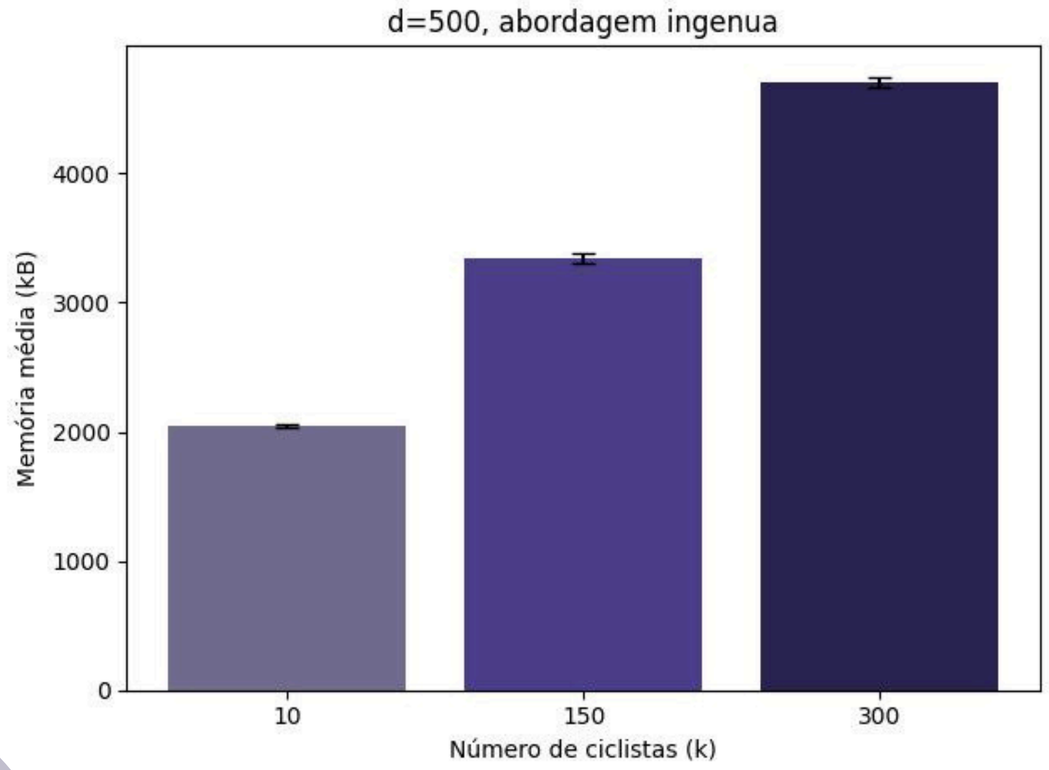
## tempo de execução

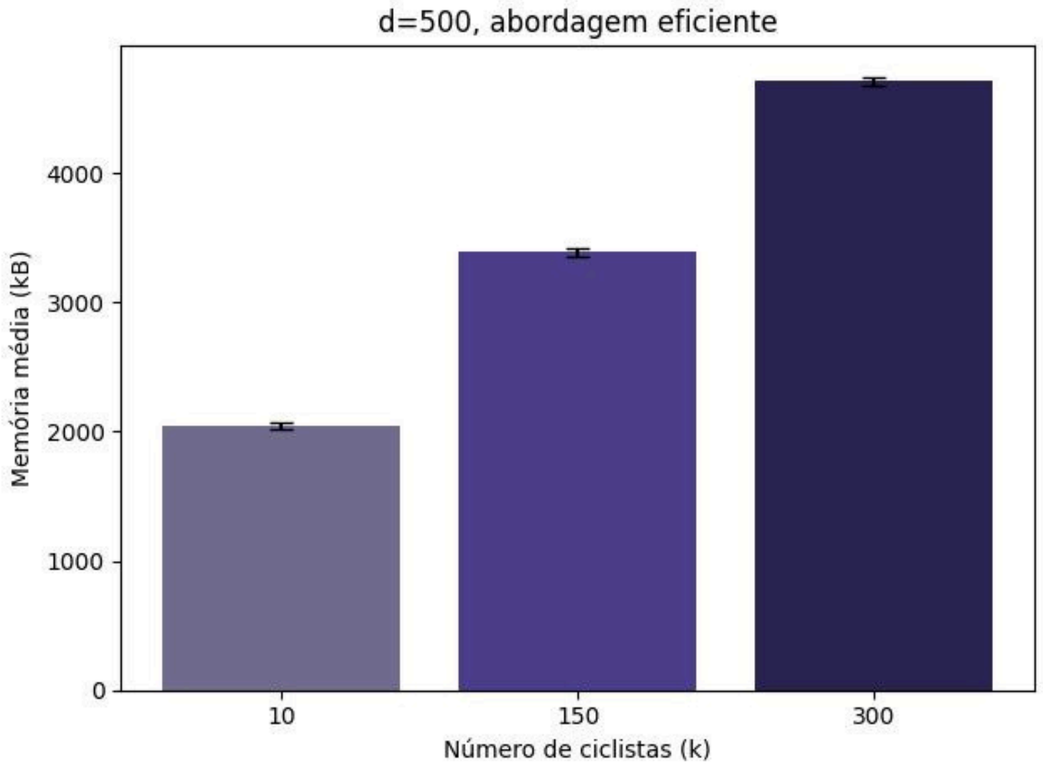


# Resultados

## d = 500
## memória

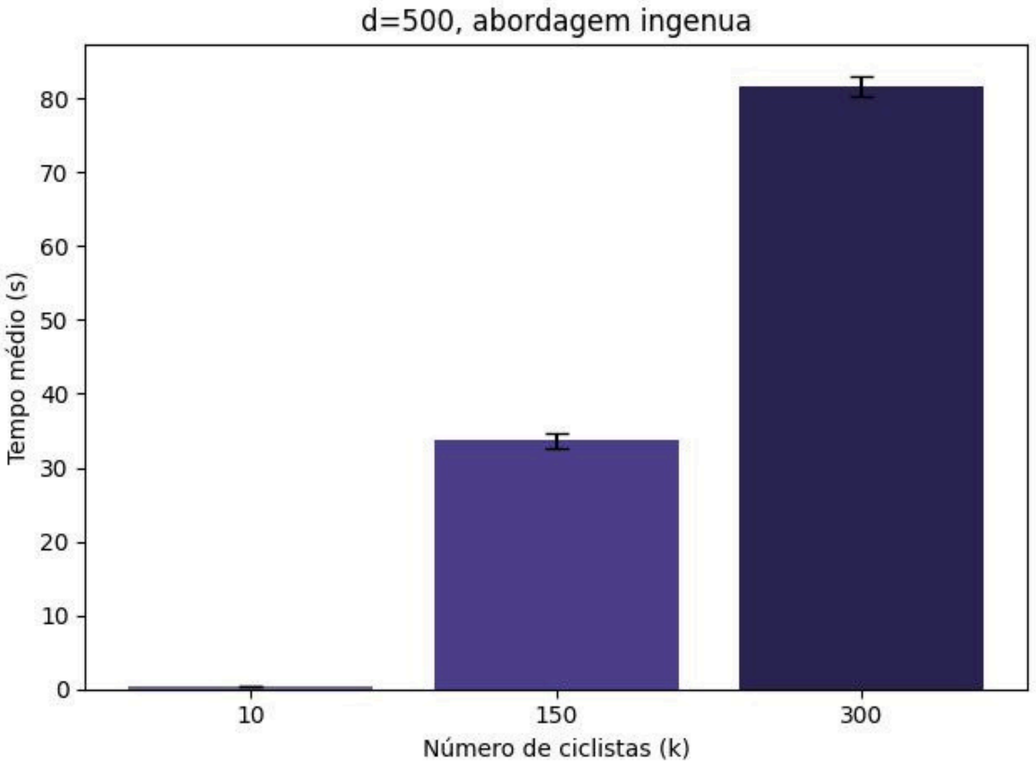


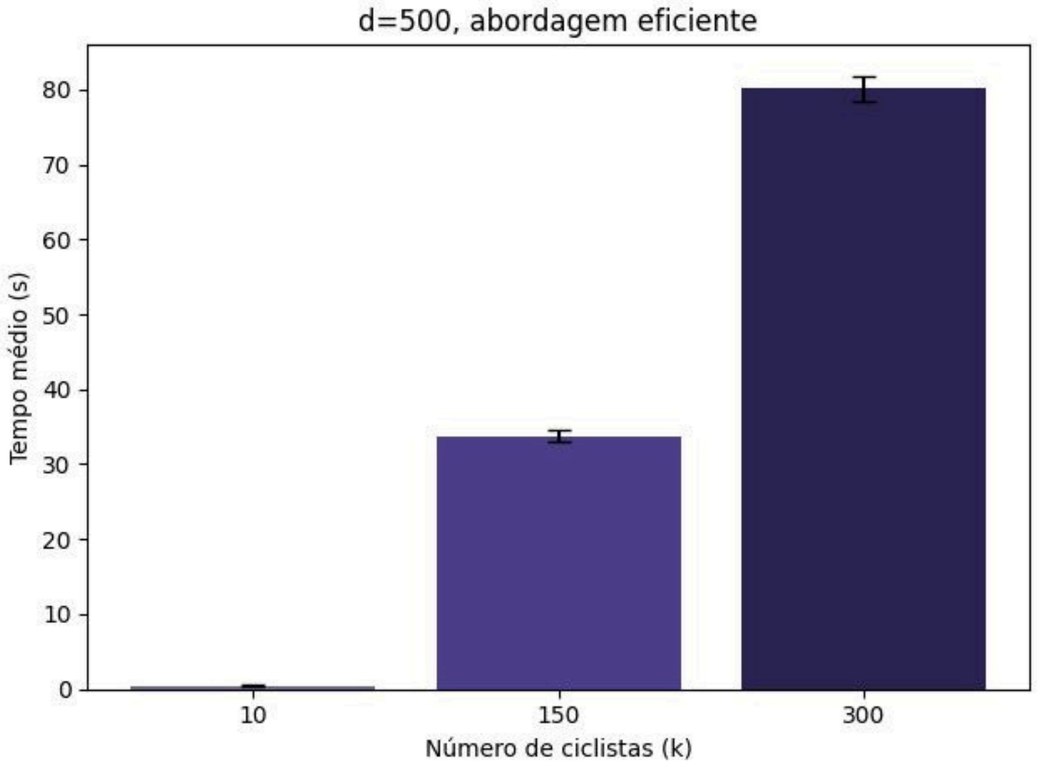



# Resultados

## d = 500

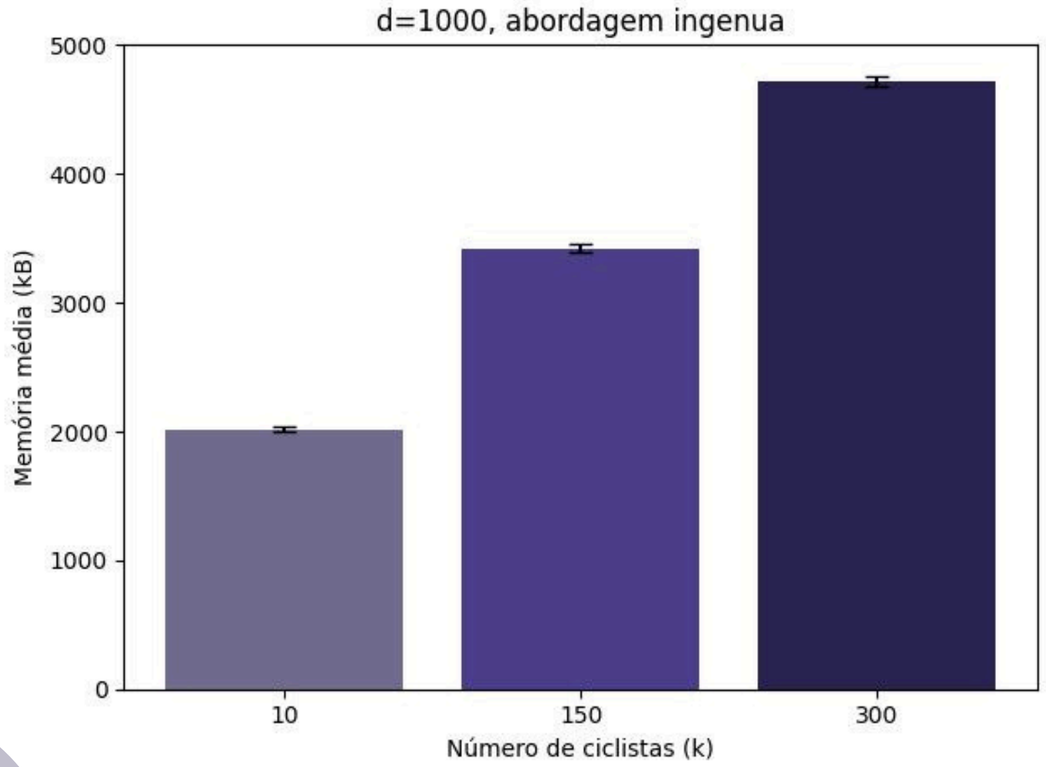
## tempo de execução

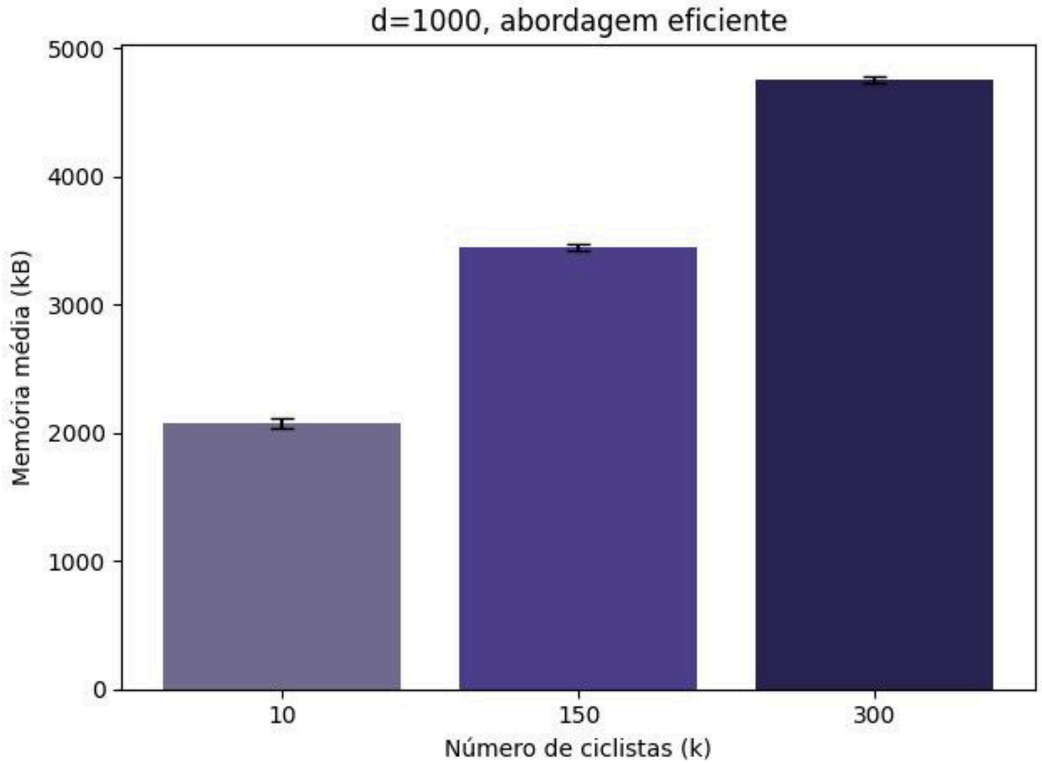


# Resultados

## d = 1000
## memória

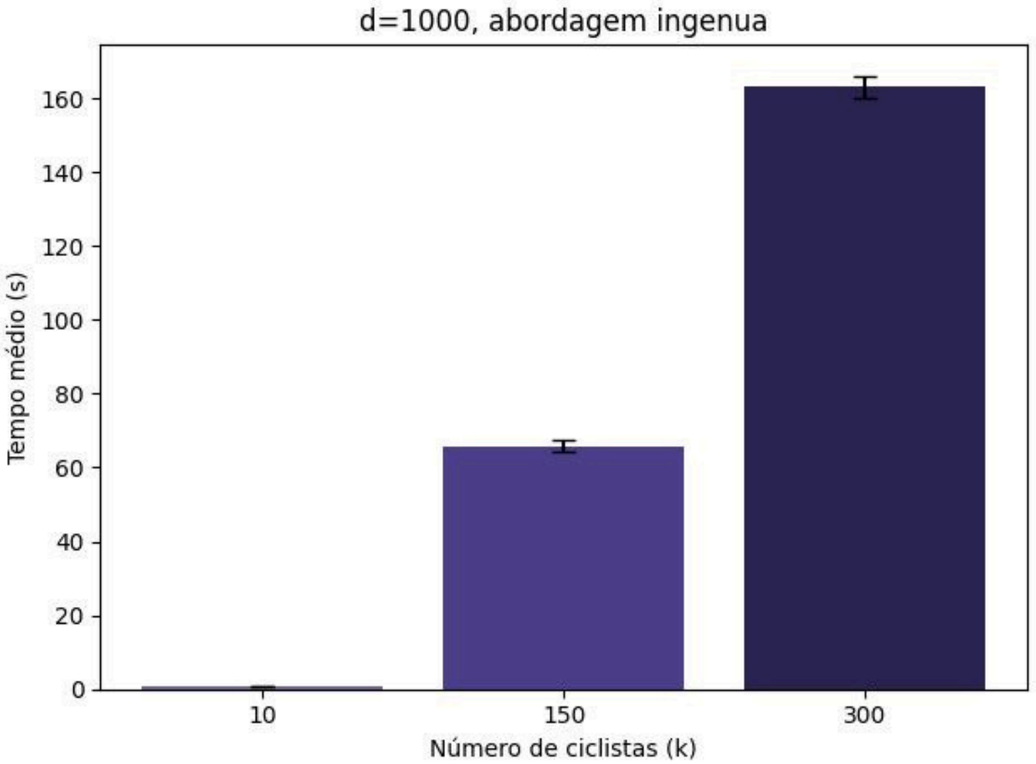


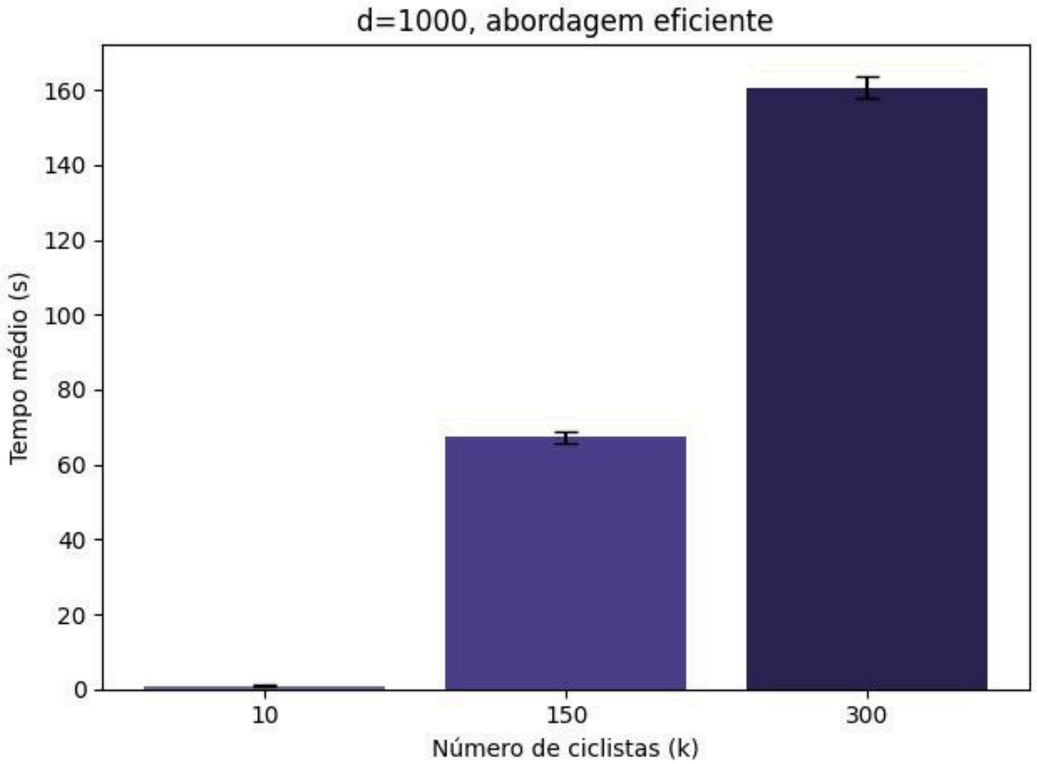



# Resultados

## d = 1000

## tempo de execução



# Análise

## Memória

### Aumenta com o número de threads (k)

A memória cresce conforme aumentamos **k** de 10 -> 150 -> 300. Isso era de se esperar, como cada thread aloca estruturas idênticas, então aumentar o número de ciclistas aumenta o consumo de RAM.

### Não depende do tamanho da pista (d)

O consumo em kB para cada valor de **k** mal varia quando mudamos **d** de 100 -> 500 -> 1000. Embora a matriz pista tenha tamanho O(d), na prática ela é pequena (10 faixas × d posições) frente ao overhead por thread, e portanto não impacta significativamente o uso total de memória.

### Abordagem ingênua × eficiente

Os dois conjuntos de gráficos de memória (ingênua vs. eficiente) são parecidos. Como ambas guardam as mesmas estruturas de dados, não havia razão para uma consumir muito mais que a outra.

# Análise

## Tempo de Execução

### Aumenta com o número de threads (k)

Mantendo a pista fixa, o tempo de execução cresce conforme aumentamos **k** de 10 -> 150 -> 300. Esse comportamento reflete que cada thread-ciclista realiza trabalho por iteração, e aumentar o número de threads também aumenta o custo, já que aumenta a concorrência por recursos do SO e a dificuldade de escolonar os processos.

### Aumenta com o tamanho da pista (d)

Mantendo k fixo, o tempo de execução também cresce conforme aumentamos **d** de 100 -> 500 -> 1000. Isso ocorre pois cada volta exige **d** passos, os quais exigem 60ms cada para serem completados. Então, se **d** aumenta, o tempo de execução também aumentará.

### Abordagem ingênua × eficiente

As barras de tempo para as duas abordagens são praticamente iguais. Ou seja, o controle de acesso por coluna não trouxe ganho perceptível sobre o mutex único, nos cenários testados. Isso pode indicar que o custo da sincronização pelas barreiras e o overhead de gerenciamento de threads dominam o tempo total, e/ou que há tão pouca contenção de mutex, já que os ciclistas ficam  espalhados, que trocar um mutex global por vários não faz diferença prática.