



**UNIVERSIDADE ESTADUAL DO CEARÁ**  
**CENTRO DE CIÊNCIAS E TECNOLOGIA - CCT**  
**CIÊNCIA DA COMPUTAÇÃO**  
**PROGRAMAÇÃO PARALELA E CONCORRENTE**

**RELATÓRIO DO PROBLEMA DO BAR DOS FILÓSOFOS**

**FRANCISCO MATHEUS FERNANDES FREITAS, 1607881**

**FORTALEZA**

**2024**

## **Introdução**

O objetivo deste relatório é descrever sobre uma solução para o problema “Bar dos Filósofos”, uma variação do Jantar dos Filósofos [1], um clássico problema de concorrência. O objetivo foi exercitar os conhecimentos adquiridos na disciplina de Programação Paralela e Concorrente, ao implementar técnicas de sincronização, e comentar as estruturas adotadas, assim como os resultados e experiências obtidas com isso.

## **Problema do Bar dos Filósofos**

O problema do Jantar dos Filósofos é um clássico da computação, pois deixa claro desafios de concorrência e sincronização. Se trata de cinco filósofos que alternam entre pensar e comer, sentados em uma mesa circular com um garfo entre cada par de filósofos. Para comer, um filósofo precisa pegar os dois garfos, da esquerda e da direita. O problema surge ao tentar garantir que todos possam comer sem conflitos (deadlock), onde nenhum filósofo consegue comer porque estão todos esperando por garfos que não são liberados.

O Bar dos Filósofos é uma variação desse problema clássico, adicionando características que necessitam de ainda mais atenção na sincronização dos filósofos. Neste problema, temos  $n$  filósofos, com 1 ou  $n - 1$  arestas com outros filósofos, e as arestas são as garrafas compartilhadas entre eles. Não há restrições quanto à estrutura do grafo ou à conectividade, permitindo maior flexibilidade. Cada filósofo escolhe, a cada rodada, um subconjunto de garrafas adjacentes para beber, o que pode variar em cada sessão. Esse problema generaliza o original ao permitir que os processos selecionem diferentes números de recursos e, ao limitar os estados dos filósofos a tranquilo, com sede e bebendo, análogo ao filosofando, com fome e comendo no Jantar dos Filósofos.

## Implementação

O projeto foi desenvolvido usando a linguagem de programação C#, usando o framework dotnet 8.0.401, e o ambiente usado para o desenvolvimento foi o VsCode. Como mecanismo de sincronização das Threads, foi decidido por usar a estrutura de dados Mutex [ 2 ], nativo da linguagem, bem como a estrutura de dados Task, que encapsula as funcionalidades de Threads.

O projeto foi arquitetado da seguinte forma:

1. Modelos:

- Diretório que guarda as entidades que serão usadas para performar o problema.

- a. Matriz.cs:

- Responsável por tratar os arquivos contendo os grafos usados para representar os filósofos e as garrafas.

- b. Estoque.cs:

- Responsável por guardar os mutex de cada garrafa. É por elas que os filósofos vão verificar a disponibilidade do recurso compartilhado.

- c. Filósofo.cs:

- Classe que representa o filósofo, com seus estados de Pensando, com sede e bebendo.

2. Grafos:

- Diretório para guardar os grafos exemplos do problema.

- a. G1.txt

- b. G2.txt

- c. G3.txt

### 3. Program.cs

Possui o fluxo principal de execução da solução para o problema.

#### Mecanismo de Sincronização


A classe Estoque possui o mecanismo usado para sincronização dos filósofos para com os recursos compartilhados, representados pelas garrafas. Como apenas o acesso era o foco para conseguir as garrafas, foi usado a estrutura de dados Mutex, nativa do C#. Se trata de uma primitiva de sincronização que também pode ser usada para sincronização entre processos.

```
1  class Estoque
2  {
3      public Mutex[,] Garrafas { get; set; }
4
5      public Estoque(int[,] matrixAdjacencia)
6      {
7          int qtdeFilosofos = matrixAdjacencia.GetLength(0);
8          Garrafas = new Mutex[qtdeFilosofos, qtdeFilosofos];
9          for (int i = 0; i < qtdeFilosofos; i++)
10         {
11             for (int j = 0; j < qtdeFilosofos; j++)
12             {
13                 if ((matrixAdjacencia[i, j] is 1) && (Garrafas[i,j] is null))
14                 {
15                     var temp = new Mutex();
16                     Garrafas[i,j] = temp;
17                     Garrafas[j,i] = temp;
18                 }
19             }
20         }
21     }
22 }
```

Figura 1: Estoque, classe que guarda as garrafas compartilhadas

## Filósofo


A classe Filósofo guarda os dados que o filósofo precisa para finalizar sua linha de execução, como seu ID para identificar as garrafas que ele tem acesso. Outros dados como o número de vezes que ele precisa beber, a lista de garrafas que ele tem acesso e uma propriedade privada que marca os tempo que ele levou para finalizar sua execução.



```
1 public int Id { get; private set; }
2 public int DrinksFaltantes { get; set; }
3 private List<int> GarrafasPossiveis { get; }
4 private readonly Stopwatch stopwatch = new();
```

**Figura 2: Propriedades da classe Filósofo**

O construtor do Filósofo espera seu ID, quantas vezes ele precisará beber (pois esse número é diferente quando o grafo é maior) e a matriz de adjacência para que ele possa identificar quais garrafas ele tem acesso.



```
1 public Filosofo(int id, int drinksTotal, ref int[,] matrixAdjacencia)
2 {
3     Id = id;
4     DrinksFaltantes = drinksTotal;
5     GarrafasPossiveis = PegarGarrafasPossiveis(ref matrixAdjacencia);
6 }
```

**Figura 3: Construtor do Filósofo**

O filósofo inicia com sede, sorteando quantas garrafas ele precisa na rodada atual e começa a tentar pegar as garrafas que tem à sua disposição. Para evitar deadlock, onde o filósofo segura uma garrafa até conseguir todas que precisa, foi implementado um timeout, onde após dois segundos segurando a garrafa sem conseguir as outras necessárias, ele libera todas as garrafas que conseguiu.


No caso em que ele não consiga as garrafas necessárias, ao liberar, sua prioridade é elevada para que ele tenha mais chances de conseguir. Ao pegar todas as garrafas que precisa para aquela rodada, o filósofo as bebe e libera, e tem sua prioridade normalizada, voltando para o estado pensando.

## **Execução**

O arquivo Program.cs possui a solução para o problema “Bar dos Filósofos”, onde primeiro é conseguido a matriz de adjacências dado um arquivo txt que possui a representação do grafo. Segundo, o Estoque com as garrafas compartilhadas é configurada usando a matriz de adjacências.

Os filósofos são colocados em uma lista, já configurados com um ID, o número de vezes que vai precisar beber e a matriz de adjacências, para que possa guardar as garrafas ele tem acesso.

Por fim, os filósofos são executados em paralelo.



```
1  var data = Matriz.ParseFileToMatriz("grafos/63.txt");
2  var qtdeBeber = data.Item1;
3  var matriz = data.Item2;
4
5  // Estoque guarda as garrafas disponiveis
6  Estoque estoque = new(matriz);
7
8  // Preparando os filosofos para serem executados em paralelo
9  List<Task> filosofos = [];
10 for (int i = 0; i < matriz.GetLength(0); i++)
11 {
12     var filosofo = new Filosofo(i, qtdeBeber, ref matriz);
13     filosofos.Add(Task.Run(() => filosofo.Beber(estoque.Garrafas)));
14 }
15
16 Console.WriteLine("Todos já foram preparados e começarão agora...");
17
18 Task.WaitAll(.. filosofos);
19
20 Console.WriteLine("Todos os filósofos terminaram de beber.");
```

**Figura 4: Program.cs, a main da solução**

Dessa forma, não houveram deadlocks por conta dos mecanismos de anti-bloqueio como o timeout na espera por um recurso específico, possibilitando a tentativa de conseguir outro possível recurso. Starvation também foram evitadas a um método de escolha híbrida, com uma prioridade aumentada para aqueles que não conseguem o recurso, mas uma escolha aleatória para assegurar que todos possuem uma chance.

Mais detalhes da execução serão vistos no vídeo demonstrativo.

## **Experiência**

Entender como uma boa sincronização leva a um desempenho melhorado em sistemas concorrentes e quais técnicas estão disponíveis para isso foram afluadas por esse projeto. Muitas aplicações atuais são passíveis de melhorias por meio da programação paralela, e saber como modelar e executar uma estratégia como a realizada neste projeto serão essenciais para projetos que já estou inserido, bem como os futuros também.



### **Referências bibliográficas**

- [ 1 ] [Dining philosophers problem - Wikipedia](#)
- [ 2 ] [Mutex Class \(System.Threading\) | Microsoft Learn](#)