# Jantar dos Filósofos

Algoritmos Distribuídos

Professora: Lúcia Drummond

# Algoritmo Dining_Philosophers

- **Variáveis**

  $hungry_i$ = **false**;
  $Holds\_fork_i^j$ = **false** para todo $n_j$;
  $Holds\_turn_i^j$ = **false** para todo $n_j$;
  $owes\_fork_i^j$ = **false** para todo $n_j$;

# Algoritmo Dining_Philosophers

- **Algoritmo**

**(1)Input:**

$msg_i$ = **nil**;

**Ação quando not** *hungry* e necessita-se do acesso ao recurso compartilhado:

$hungry_i$ := **true**;

Envie *request* para todo $n_j$ tal que $holds\_fork_i^j$= false;

# Algoritmo Dining_Philosophers

- **Algoritmo (cont)**

**(2)Input:**

$msg_i$ = *request* tal que origem$_i$(msg$_i$)=n$_j$;

**Ação:**

**if not** $hungry_i$ **or not** $holds\_turn_i^j$ **then**

**Begin**

$holds\_fork_i^j$:= **false**;

**if not** $hungry_i$ **then** Envie *fork***(nil)** para $n_j$;

**else** Envie *fork***(***request***)** para $n_j$;

**end**

**else**

$owes\_fork_i^j$:= true;

# Algoritmo Dining_Philosophers

- **Algoritmo (cont)**

**(3)Input:**

$msg_i$ = *fork(t)* tal que $origem_i(msg_i)=n_j$;

**Ação:**

$holds\_fork_i^j$:= **true**;

**if** *t* = *turn* **then**

$holds\_turn_i^j$:= **true**;

**if** *t* = *request* **then**

$owes\_fork_i^j$:= **true**;

# Algoritmo Dining_Philosophers

- **Algoritmo (cont)**

```
if holds_fork_i^j para todo n_k ∈ Neig_i then
    begin
        acessa recursos compartilhados;
    hungry_i := false;
    for all n_k ∈ Neig_i do
        if holds_turn_i^k then
        begin
            holds_turn_i^k := false;
            if owes_fork_i^k then
```

# Algoritmo Dining_Philosophers

- **Algoritmo (cont)**

```
                    begin
                        owes_fork_i^k := false;
                        holds_fork_i^k := false;
                        Envie fork(turn) para n_k;
                    end
                else
                    Envie turn para n_k;
            end
        end
```

# Algoritmo Dining_Philosophers

- **Algoritmo (cont)**

**(4)Input:**

$msg_i$ = *turn* tal que $origem_i(msg_i)=n_j$;

**Ação:**

$holds\_turn_i^j$:= **true**;

# Teoremas

**Teorema:**

O algoritmo assegura exclusão mútua no acesso ao recurso compartilhado e além disso está livre de *deadlock* e livre de *starvation*.

**Prova:**

Por 3, um nó somente acessa os recursos compartilhados se este possui os garfos correspondentes a todas as tarefas incidentes a ele.

Por 2 e 3, somente um entre dois vizinhos pode pegar o garfo que eles compartilham em qualquer estado global, e por isso dois vizinhos nunca acessam recursos compartilhados concorrentemente.

# Teoremas

**Prova (cont):**

Como nós que não são vizinhos nunca compartilham recursos, a exclusão mútua é garantida.

A orientação de G é sempre acíclica, e então G sempre tem no mínimo um *sink*. *Sinks* são nós que possuem os *turns* correspondentes a todas as arestas incidentes a eles, e então por 1 e 2 devem adquirir todos os garfos que eles não possuem dentro de um tempo finito.

Assim, o *deadlock* não é possível.

# Teoremas

**Prova (cont):**

A um nó que não é um *sink* mas executa 1 a fim de acessar os recursos compartilhados também é assegurado acessar todos os garfos necessários dentro de um tempo finito e então *starvation* não ocorre.

Esta conclusão se baseia no fato de que ou um nó adquire todos os garfos porque seus vizinhos que possuem a vez não precisam acessar os recursos compartilhados (por 2 e 3), ou porque ele eventualmente adquire todos os *turns* (por 3 e 4) e então os garfos como uma consequência da aciclicidade das orientações de G.

# Complexidades

O número de mensagens que precisa ser trocado por acesso aos recursos compartilhados pode ser calculado da seguinte forma:

- O nó deve enviar mensagens de *request* para todos os vizinhos.
- Pode acontecer que $n_i$ não possua nenhum *turn* e os pedidos que este envia encontram nós que não precisam acessar os recursos compartilhados e então enviam $n_i$ garfos.
- A complexidade de mensagem é:

**O(número máximo de vizinhos)**

# Complexidades

A complexidade de tempo por acesso a recurso compartilhado está relacionada a maior cadeia de mensagens iniciando com o envio de *requests* por um nó e terminando com a recepção por esse por esse nó da última mensagem de *fork* que ele espera.

Tal cadeia ocorre para um nó que é uma fonte na orientação acíclica, quando todos os nós requerem acesso a recursos compartilhados.

# Complexidades

Neste caso, a distância direta deste nó aos *sinks* pode ser tão grande quanto n-1 e então:

**a complexidade de tempo é O(n).**