Programação Concorrente

Deadlocks (Impasses)

Professor: **Jeremias Moreira Gomes**
E-mail: jeremiasmg@gmail.com

Introdução

Introdução

- Em ambientes monoprogramados, geralmente o único processo ativo aloca os recurso que necessita e faz o seu respectivo trabalho.
- Em ambientes multiprogramados, há vários processos competindo por recursos.

Introdução

- Além disso, alguns recursos do computador devem ser alocados de maneira exclusiva a um processo, durante o seu uso.

Introdução

- Situação simples com uso de um recurso:

```
// Processo 01
down(recursoA);
usaRecursoA();
up(recursoA);
```

```
// Processo 02
down(recursoA);
usaRecursoA();
up(recursoA);
```

Introdução

- Situação uso de dois recursos:

```
// Processo 01
down(recursoA);
down(recursoB);
usaRecursoAeB();
up(recursoB);
up(recursoA);
```

```
// Processo 02
down(recursoA);
down(recursoB);
usaRecursoAeB();
up(recursoB);
up(recursoA);
```

Introdução

- Situação uso de dois recursos com problema:

```
// Processo 01
down(recursoA);
down(recursoB);
usaRecursoAeB();
up(recursoB);
up(recursoA);
```

```
// Processo 02
down(recursoB);
down(recursoA);
usaRecursoAeB();
up(recursoA);
up(recursoB);
```

- Possibilidade de bloqueio eterno, na espera por recursos que nunca serão liberados.

Recursos

- São objetos que um processo pode adquirir de maneira exclusiva ou não.
    a. Exclusivo: só é alocado a um processo por vez, em um instante.
- Sequência de operações ao manipular recursos:
    a. Requisitar
    b. Utilizar
    c. Liberar

Tipos de Recursos

- Preemptíveis
  - Podem ser retirados do processo por uma entidade externa.
    - Exemplos: memória, processador, etc.
- Não-preemptíveis
  - Não são retirados de um processo por entidades externas.
  - A liberação só ocorre pela liberação de espontânea vontade.
  - É o tipo de recurso onde *deadlocks* ocorrem.

Deadlocks

Deadlock

Um conjunto de processos está em *deadlock* se cada processo pertencente ao conjunto estiver esperando por um evento que somente um outro processo pertencente ao mesmo conjunto pode fazer ocorrer.

- Como todos os processos a espera por um recurso, nenhum poderá provocar a ocorrência de qualquer evento.
- Eventos nesse contexto são a alocação e liberação de recursos.

Deadlocks - Condições de Ocorrência

1. **Quatro condições devem acontecer ao mesmo tempo**, para a ocorrência de *deadlocks*.
    1.1. Exclusão Mútua: Cada recurso ser alocado de maneira exclusiva
    1.2. Posse e Espera: Um processo que detém recursos pode solicitar novos recursos.

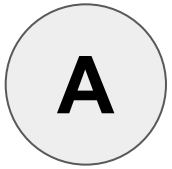
Deadlocks - Condições de Ocorrência

1. **Quatro condições devem acontecer ao mesmo tempo**, para a ocorrência de *deadlocks*.

   1.3. Não-preempção: Cada recurso só pode ser liberado pelo processo que o detém.

   1.4. Espera Circular: Deve existir um ciclo de processos onde cada processo está esperando por um recurso de posse de um próximo processo membro de uma cadeia.

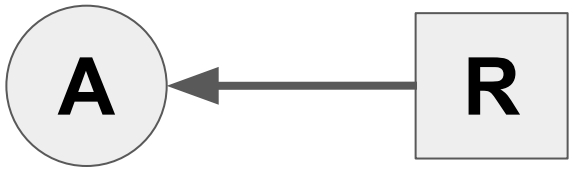
Modelo de *Deadlock*

- Em 1972, Richard C. Holt apresentou uma modelagem dessas quatro condições para a detecção de *deadlocks*, por meio de grafos dirigidos.
- Nesses grafos, há dois tipos de nós:
  - Processos: 

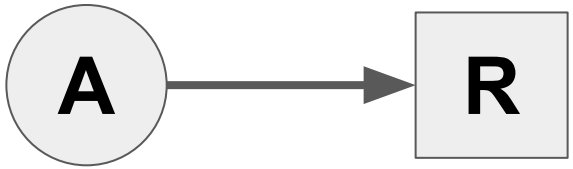
  - Recursos: 

Modelo de *Deadlock*

- Recurso R alocado ao processo A:

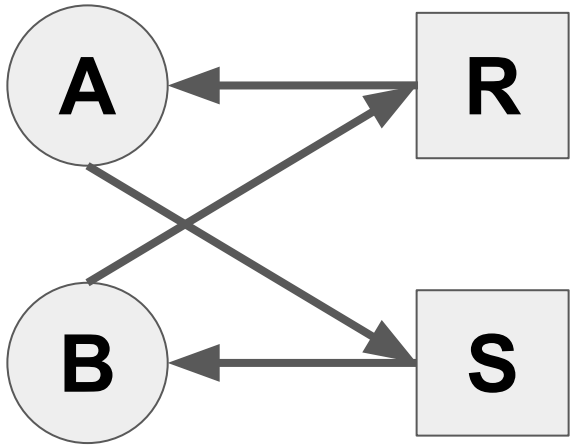


- Processo A deseja obter o recurso R e está bloqueado esperando sua liberação:



Modelo de *Deadlock*

- Deadlock envolvendo dois processos:



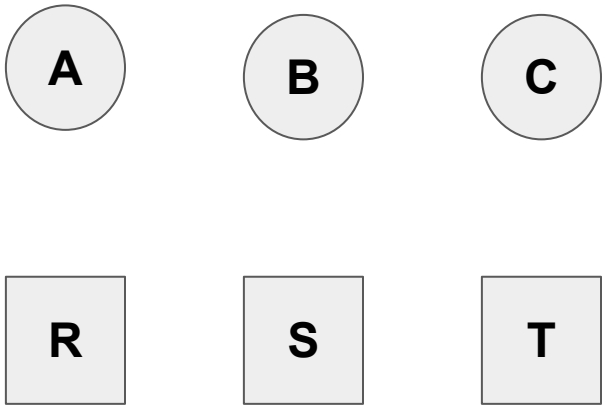
Deadlock envolvendo 03 processos - Sem *deadlock*

```
// Processo A
requisita(R);
requisita(S);
libera(R);
libera(S);
```

```
// Processo B
requisita(S);
requisita(T);
libera(S);
libera(T);
```

```
// Processo C
requisita(T);
requisita(R);
libera(T);
libera(R);
```

```
// Exemplo de execução sem deadlock:
1 - A requisita R
2 - C requisita T
3 - A requisita S
4 - C requisita R
5 - A libera R
6 - A libera S
...
```



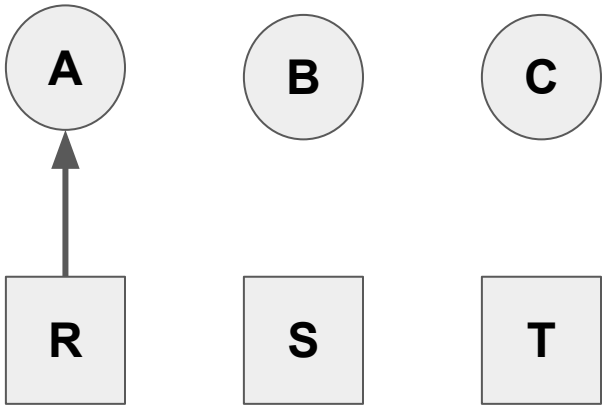
Deadlock envolvendo 03 processos - Sem *deadlock*

```
// Processo A
requisita(R);
requisita(S);
libera(R);
libera(S);
```

```
// Processo B
requisita(S);
requisita(T);
libera(S);
libera(T);
```

```
// Processo C
requisita(T);
requisita(R);
libera(T);
libera(R);
```

```
// Exemplo de execução sem deadlock:
1 - A requisita R
2 - C requisita T
3 - A requisita S
4 - C requisita R
5 - A libera R
6 - A libera S
...
```



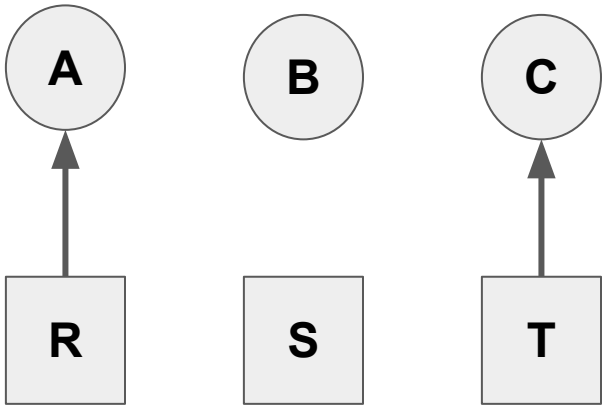
Deadlock envolvendo 03 processos - Sem *deadlock*

```
// Processo A
requisita(R);
requisita(S);
libera(R);
libera(S);
```

```
// Processo B
requisita(S);
requisita(T);
libera(S);
libera(T);
```

```
// Processo C
requisita(T);
requisita(R);
libera(T);
libera(R);
```

```
// Exemplo de execução sem deadlock:
1 - A requisita R
2 - C requisita T
3 - A requisita S
4 - C requisita R
5 - A libera R
6 - A libera S
...
```



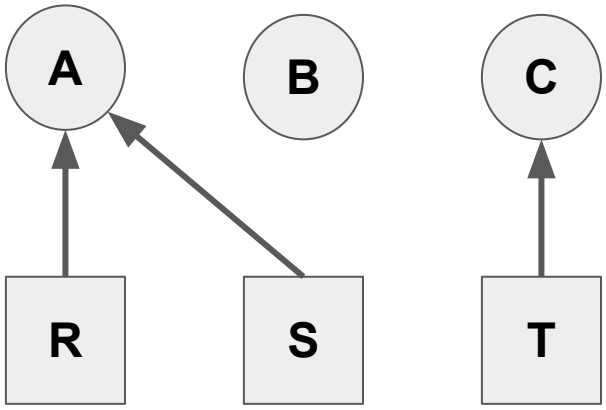
Deadlock envolvendo 03 processos - Sem *deadlock*

```
// Processo A
requisita(R);
requisita(S);
libera(R);
libera(S);
```

```
// Processo B
requisita(S);
requisita(T);
libera(S);
libera(T);
```

```
// Processo C
requisita(T);
requisita(R);
libera(T);
libera(R);
```

```
// Exemplo de execução sem deadlock:
1 - A requisita R
2 - C requisita T
3 - A requisita S
4 - C requisita R
5 - A libera R
6 - A libera S
...
```



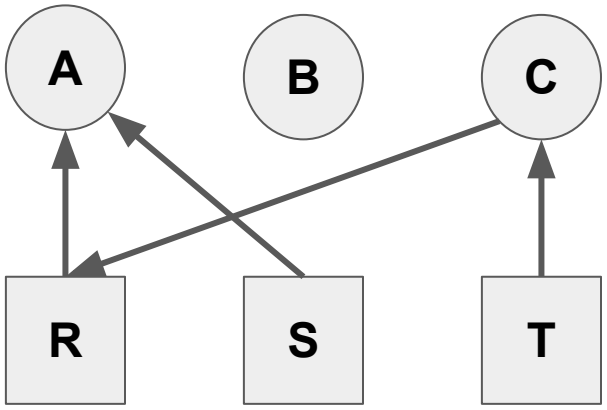
Deadlock envolvendo 03 processos - Sem *deadlock*

```
// Processo A
requisita(R);
requisita(S);
libera(R);
libera(S);
```

```
// Processo B
requisita(S);
requisita(T);
libera(S);
libera(T);
```

```
// Processo C
requisita(T);
requisita(R);
libera(T);
libera(R);
```

```
// Exemplo de execução sem deadlock:
1 - A requisita R
2 - C requisita T
3 - A requisita S
4 - C requisita R
5 - A libera R
6 - A libera S
...
```



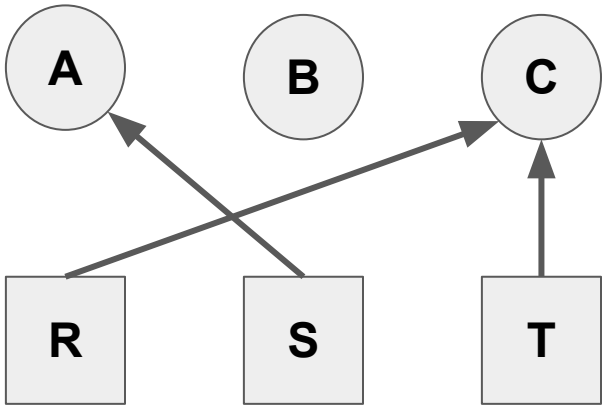
Deadlock envolvendo 03 processos - Sem *deadlock*

```
// Processo A
requisita(R);
requisita(S);
libera(R);
libera(S);
```

```
// Processo B
requisita(S);
requisita(T);
libera(S);
libera(T);
```

```
// Processo C
requisita(T);
requisita(R);
libera(T);
libera(R);
```

```
// Exemplo de execução sem deadlock:
1 - A requisita R
2 - C requisita T
3 - A requisita S
4 - C requisita R
5 - A libera R
6 - A libera S
...
```



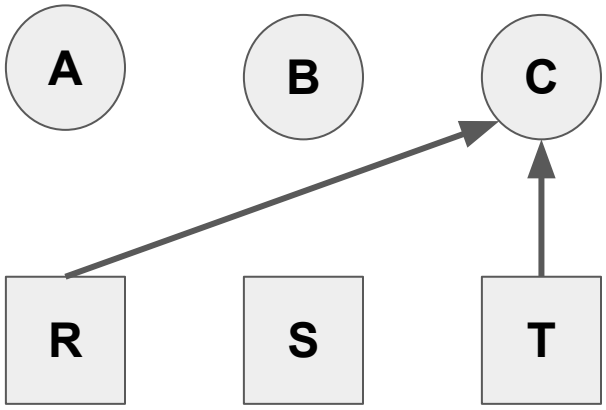
Deadlock envolvendo 03 processos - Sem *deadlock*

```
// Processo A
requisita(R);
requisita(S);
libera(R);
libera(S);
```

```
// Processo B
requisita(S);
requisita(T);
libera(S);
libera(T);
```

```
// Processo C
requisita(T);
requisita(R);
libera(T);
libera(R);
```

```
// Exemplo de execução sem deadlock:
1 - A requisita R
2 - C requisita T
3 - A requisita S
4 - C requisita R
5 - A libera R
6 - A libera S
...
```



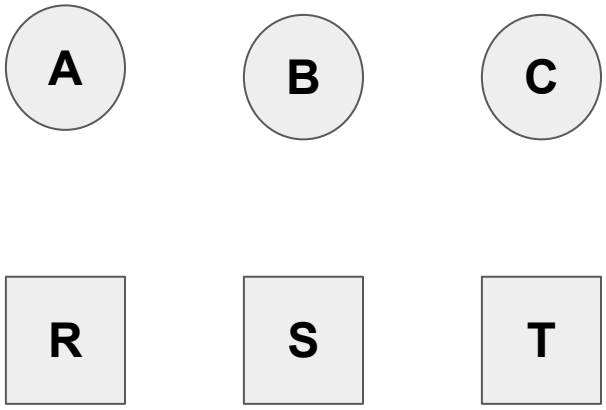
Deadlock envolvendo 03 processos - Com *deadlock*

```
// Processo A
requisita(R);
requisita(S);
libera(R);
libera(S);
```

```
// Processo B
requisita(S);
requisita(T);
libera(S);
libera(T);
```

```
// Processo C
requisita(T);
requisita(R);
libera(T);
libera(R);
```

```
// Exemplo de execução com deadlock:
1 - A requisita R
2 - B requisita S
3 - C requisita T
4 - A requisita S
5 - B requisita T
6 - C requisita R
...
```



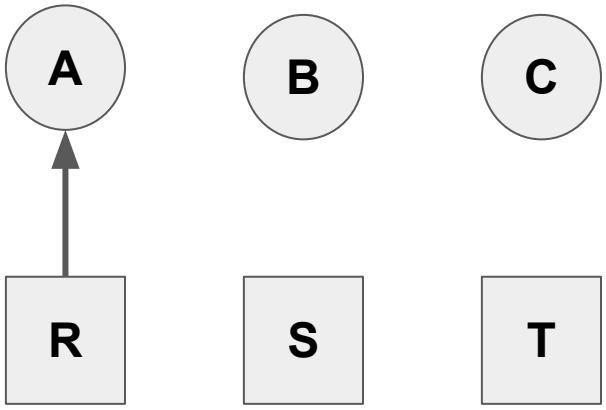
Deadlock envolvendo 03 processos - Com *deadlock*

```
// Processo A
requisita(R);
requisita(S);
libera(R);
libera(S);
```

```
// Processo B
requisita(S);
requisita(T);
libera(S);
libera(T);
```

```
// Processo C
requisita(T);
requisita(R);
libera(T);
libera(R);
```

```
// Exemplo de execução com deadlock:
1 - A requisita R
2 - B requisita S
3 - C requisita T
4 - A requisita S
5 - B requisita T
6 - C requisita R
...
```



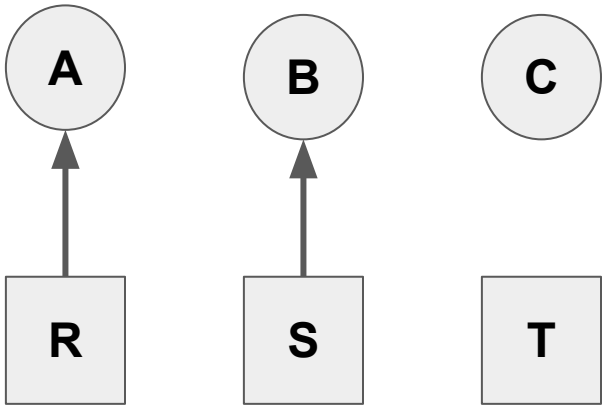
Deadlock envolvendo 03 processos - Com *deadlock*

```
// Processo A
requisita(R);
requisita(S);
libera(R);
libera(S);
```

```
// Processo B
requisita(S);
requisita(T);
libera(S);
libera(T);
```

```
// Processo C
requisita(T);
requisita(R);
libera(T);
libera(R);
```

```
// Exemplo de execução com deadlock:
1 - A requisita R
2 - B requisita S
3 - C requisita T
4 - A requisita S
5 - B requisita T
6 - C requisita R
...
```



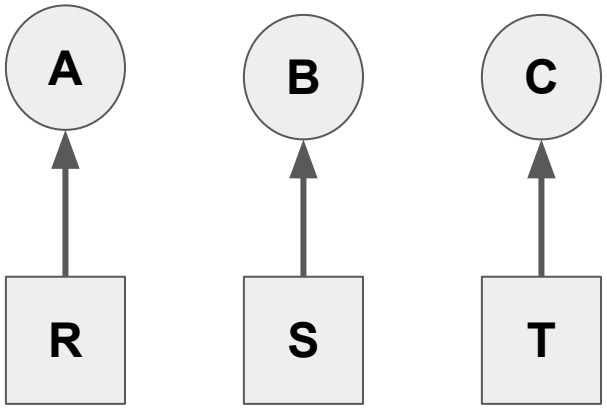
Deadlock envolvendo 03 processos - Com *deadlock*

```
// Processo A
requisita(R);
requisita(S);
libera(R);
libera(S);
```

```
// Processo B
requisita(S);
requisita(T);
libera(S);
libera(T);
```

```
// Processo C
requisita(T);
requisita(R);
libera(T);
libera(R);
```

```
// Exemplo de execução com deadlock:
1 - A requisita R
2 - B requisita S
3 - C requisita T
4 - A requisita S
5 - B requisita T
6 - C requisita R
...
```



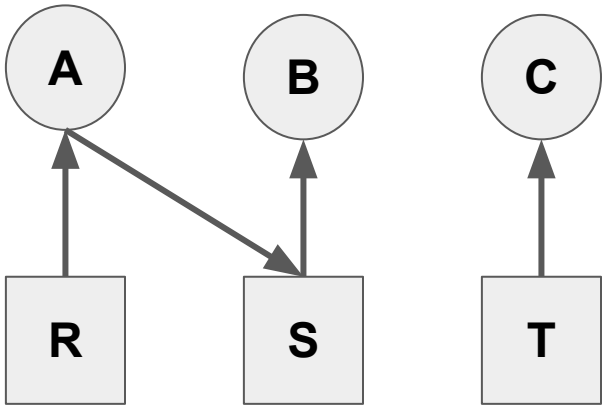
Deadlock envolvendo 03 processos - Com *deadlock*

```
// Processo A
requisita(R);
requisita(S);
libera(R);
libera(S);
```

```
// Processo B
requisita(S);
requisita(T);
libera(S);
libera(T);
```

```
// Processo C
requisita(T);
requisita(R);
libera(T);
libera(R);
```

```
// Exemplo de execução com deadlock:
1 - A requisita R
2 - B requisita S
3 - C requisita T
4 - A requisita S
5 - B requisita T
6 - C requisita R
...
```



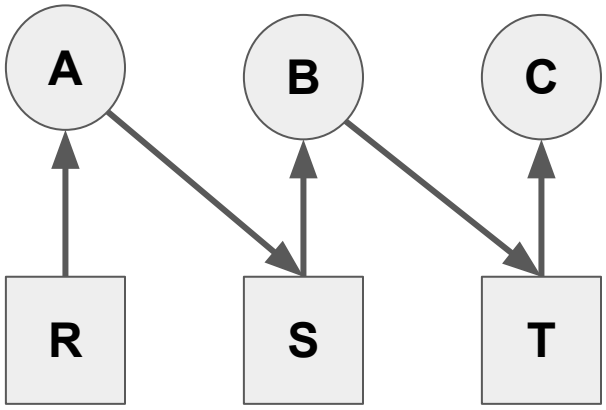
Deadlock envolvendo 03 processos - Com *deadlock*

```
// Processo A
requisita(R);
requisita(S);
libera(R);
libera(S);
```

```
// Processo B
requisita(S);
requisita(T);
libera(S);
libera(T);
```

```
// Processo C
requisita(T);
requisita(R);
libera(T);
libera(R);
```

```
// Exemplo de execução com deadlock:
1 - A requisita R
2 - B requisita S
3 - C requisita T
4 - A requisita S
5 - B requisita T
6 - C requisita R
...
```



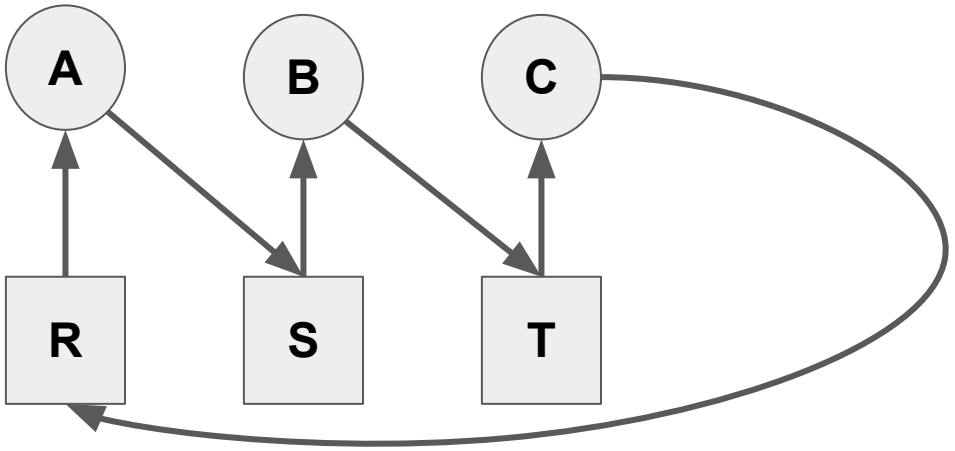
Deadlock envolvendo 03 processos - Com *deadlock*

```
// Processo A
requisita(R);
requisita(S);
libera(R);
libera(S);
```

```
// Processo B
requisita(S);
requisita(T);
libera(S);
libera(T);
```

```
// Processo C
requisita(T);
requisita(R);
libera(T);
libera(R);
```

```
// Exemplo de execução com deadlock:
1 - A requisita R
2 - B requisita S
3 - C requisita T
4 - A requisita S
5 - B requisita T
6 - C requisita R
...
```



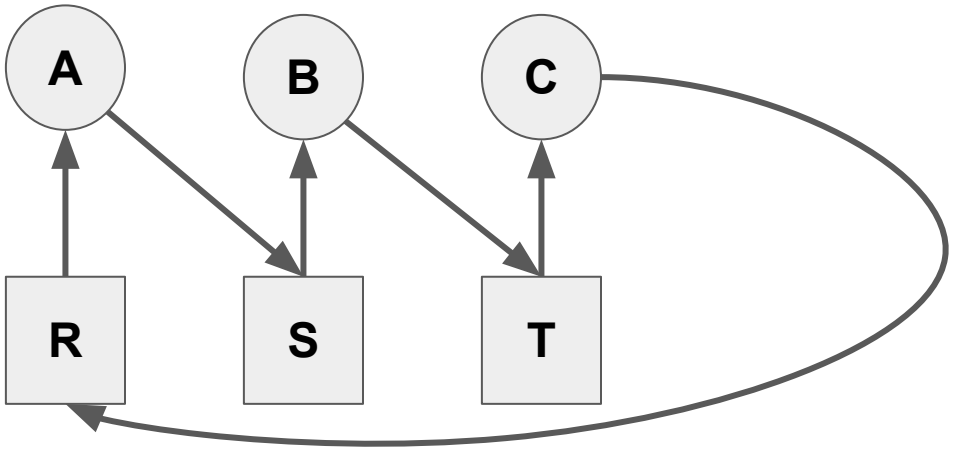
Deadlock envolvendo 03 processos - Com *deadlock*

```
// Processo A
requisita(R);
requisita(S);
libera(R);
libera(S);
```

```
// Processo B
requisita(S);
requisita(T);
libera(S);
libera(T);
```

```
// Processo C
requisita(T);
requisita(R);
libera(T);
libera(R);
```

```
// Exemplo de execução com deadlock:
1 - A requisita R
2 - B requisita S
3 - C requisita T
4 - A requisita S
5 - B requisita T
6 - C requisita R
// Deadlock
```



Estratégias para Tratar *Deadlocks*

- A literatura indica quatro formas de se lidar com *deadlocks*:

  a. Ignorar o problema.

  b. Detectar e recuperar o *deadlock.*

  c. Evitar o *deadlock*.

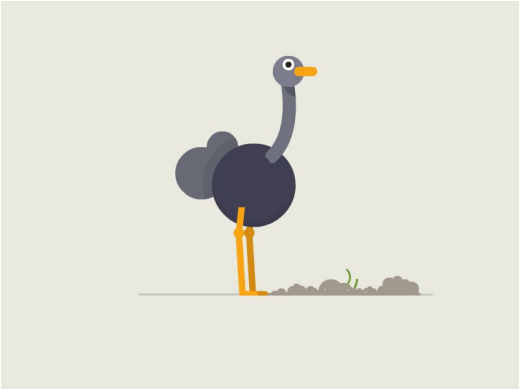
  d. Prevenir o *deadlock*.

1 - Ignorar o Problema

- Justificativa: Não vale a pena degradar a performance do sistema para tratar uma situação que ocorre com pouca frequência.
- O tratamento do *deadlock* fica a cargo do programador.

1 - Ignorar o Problema

- Algoritmo do Avestruz
  - Enfie a cabeça na areia e finja que não há um problema!
    - Inaceitável para alguns (matemáticos), mas útil quando se considera o esforço necessário em relação a quantidade de vezes que o deadlock ocorre (engenheiros).



2 - Detectar e Recuperar o *Deadlock*

- Justificativa: Se existe a possibilidade de ocorrer o problema, então o Sistema Operacional deve tratar.
- Composta por duas etapas:
  - Detecção de *Deadlock*
  - Recuperação de *Deadlock*

2 - Detectar e Recuperar o *Deadlock*

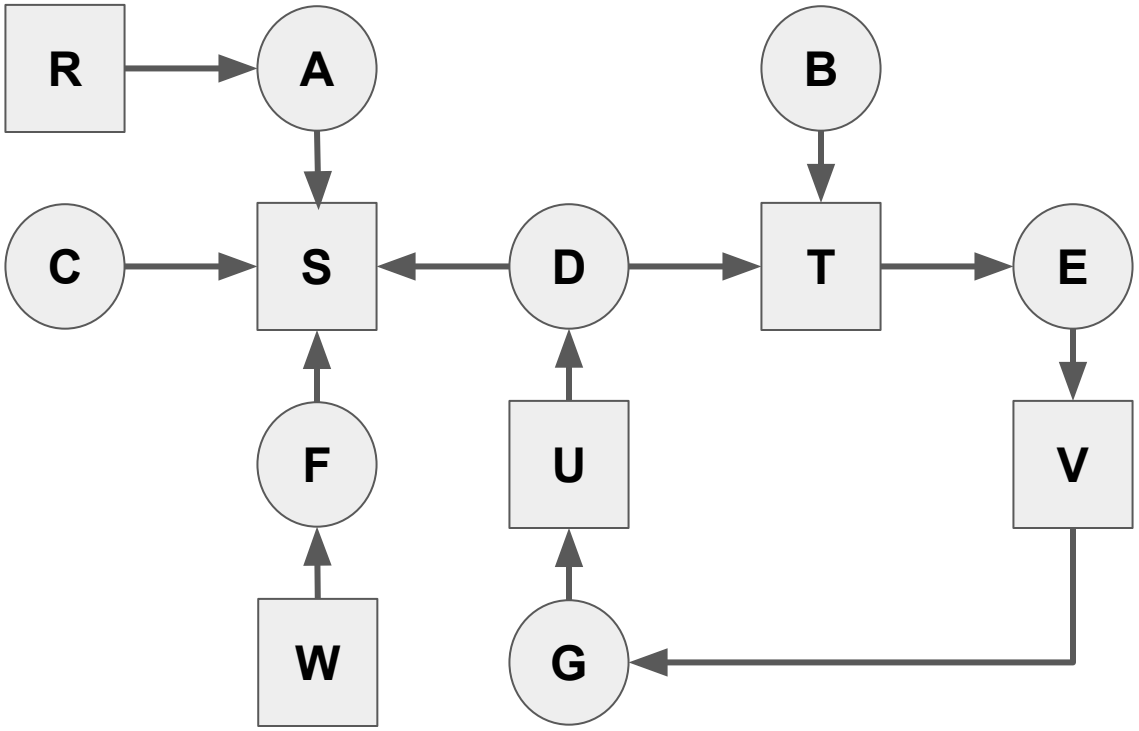
- Detecção de *Deadlock* - Um recurso de cada tipo
  - Inicialmente, constrói-se o grafo de alocação de recursos.
  - Se existir um ciclo, todos os processos que fazem parte deste ciclo estão situação de impasse (*deadlock*).
  - A detecção é feita por um algoritmo de detecção de ciclo em grafos dirigidos.

2 - Detectar e Recuperar o *Deadlock*

- Detecção de *Deadlock* - Um recurso de cada tipo
  - Como detectar se há *deadlock*?
    - O processo A possui R e solicita S.
    - O processo B não possui nada, mas solicita T.
    - O processo C não possui nada, mas solicita S.
    - O processo D possui U e solicita S e T.
    - O processo E possui T e solicita V.
    - O processo F possui W e solicita S.
    - O processo G possui V e solicita U.

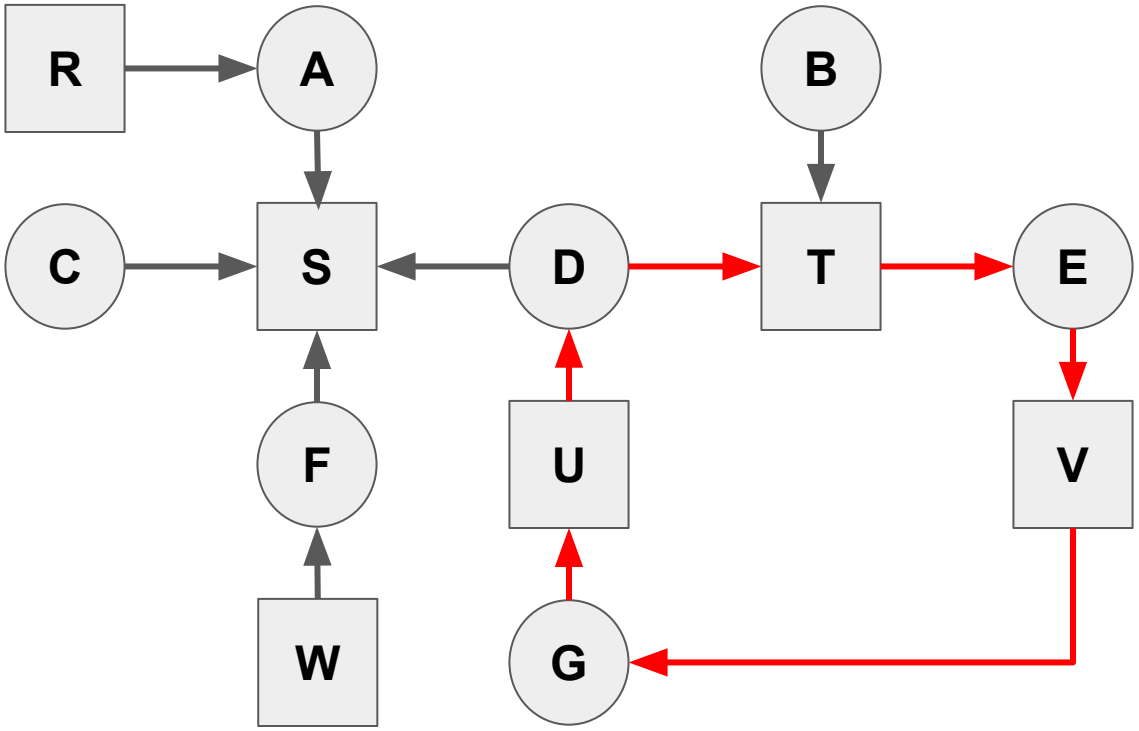
2 - Detectar e Recuperar o *Deadlock*

- Detecção de *Deadlock* - Um recurso de cada tipo



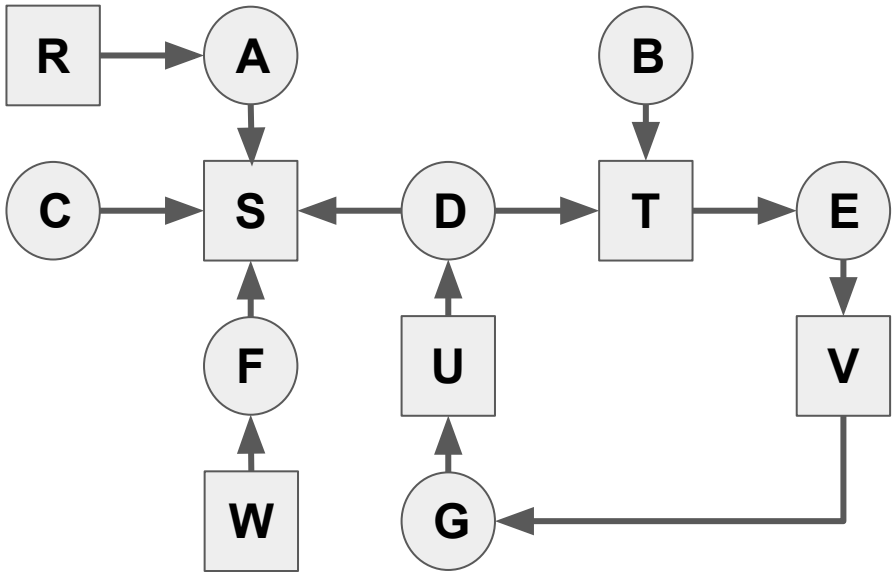
2 - Detectar e Recuperar o *Deadlock*

- Detecção de *Deadlock* - Um recurso de cada tipo



2 - Detectar e Recuperar o *Deadlock*

- Detecção de *Deadlock* - Um recurso de cada tipo
  - O algoritmo de DFS pode ser utilizado para detectar ciclos.



Lista de Nós não visitados (branca):
R, A, B, C, S, D, T, E, F, U, V, W, G

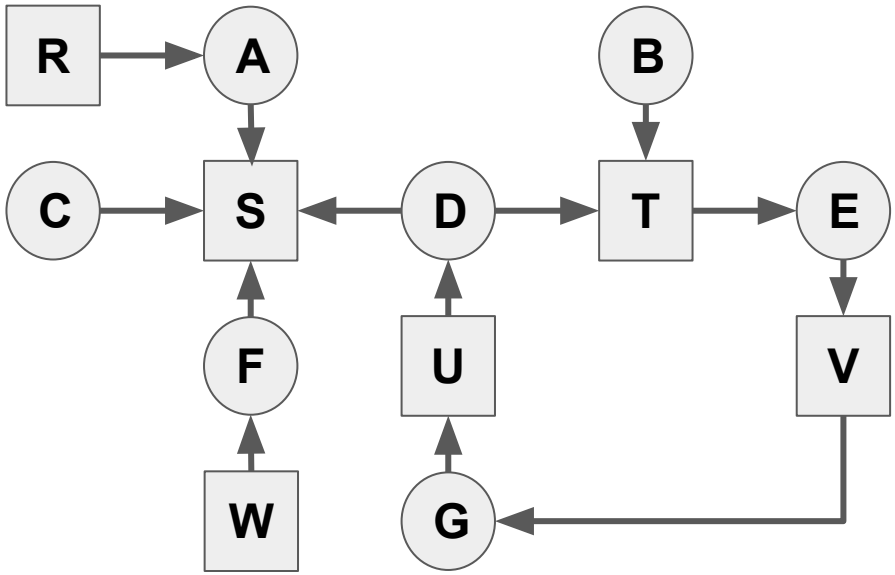
Lista de Nós Vistando no Momento (cinza):

Lista de Nós Visitados (preta):

Relação de Pais:

2 - Detectar e Recuperar o *Deadlock*

- Detecção de *Deadlock* - Um recurso de cada tipo
  - O algoritmo de DFS pode ser utilizado para detectar ciclos.



Lista de Nós não visitados (branca):
A, B, C, S, D, T, E, F, U, V, W, G

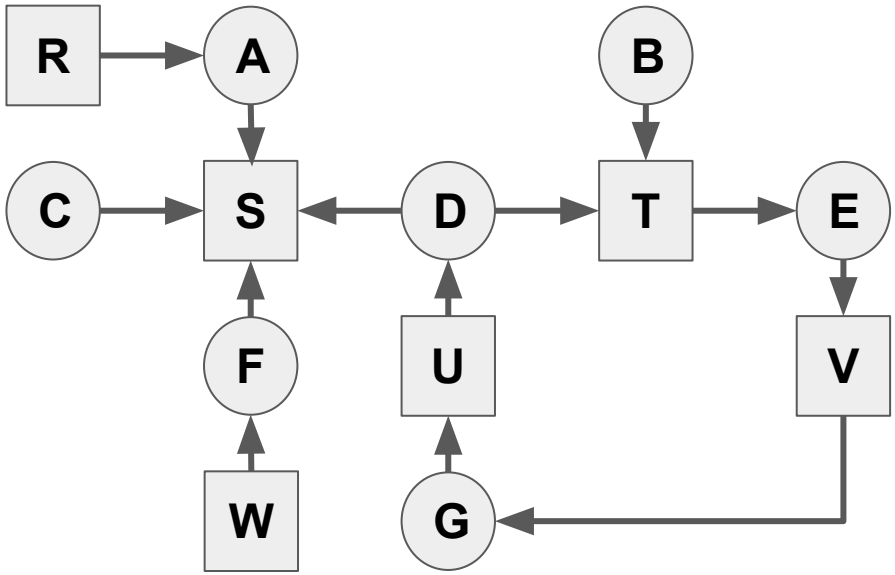
Lista de Nós Vistando no Momento (cinza):
R

Lista de Nós Visitados (preta):

Relação de Pais:
R <- ninguém

2 - Detectar e Recuperar o *Deadlock*

- Detecção de *Deadlock* - Um recurso de cada tipo
  - O algoritmo de DFS pode ser utilizado para detectar ciclos.



Lista de Nós não visitados (branca):
B, C, S, D, T, E, F, U, V, W, G

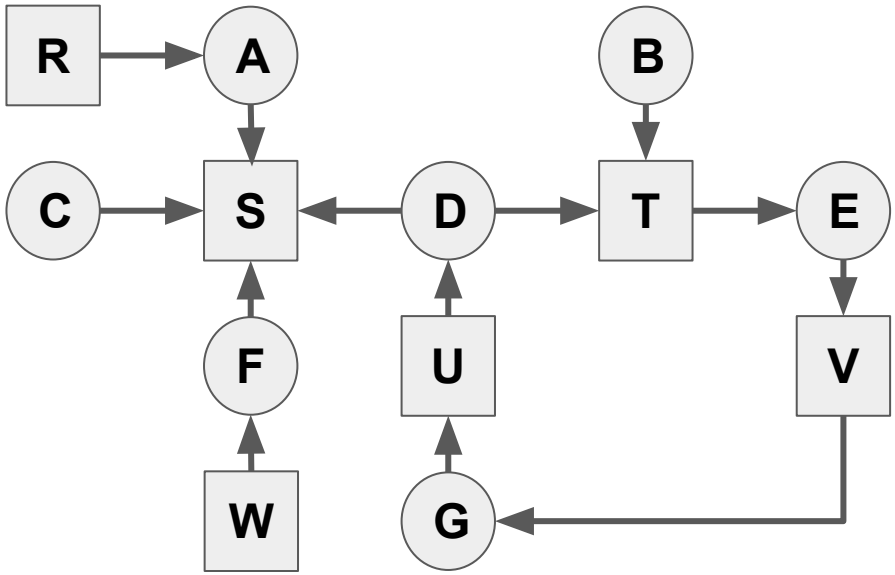
Lista de Nós Vistando no Momento (cinza):
R, A

Lista de Nós Visitados (preta):

Relação de Pais:
R <- ninguém, A <- R

2 - Detectar e Recuperar o *Deadlock*

- Detecção de *Deadlock* - Um recurso de cada tipo
  - O algoritmo de DFS pode ser utilizado para detectar ciclos.



Lista de Nós não visitados (branca):
B, C, D, T, E, F, U, V, W, G

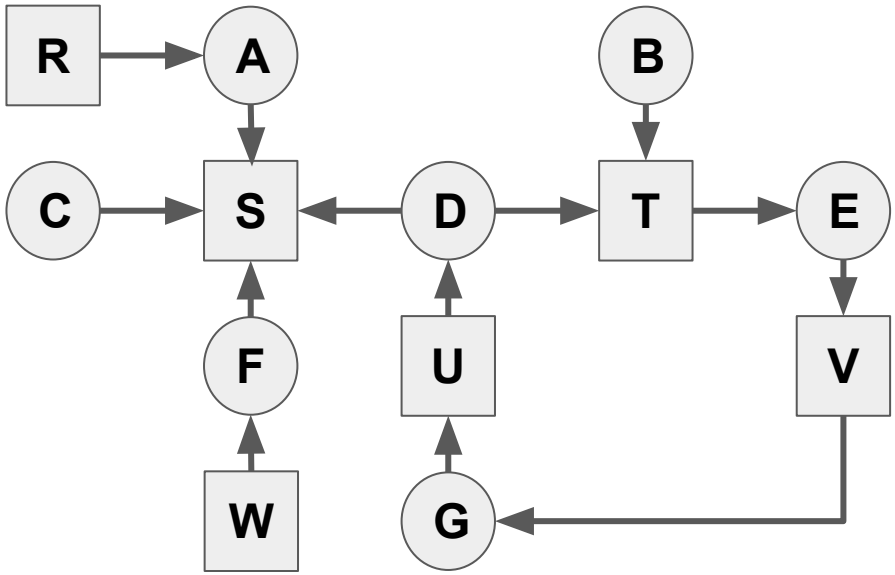
Lista de Nós Vistando no Momento (cinza):
R, A, S

Lista de Nós Visitados (preta):

Relação de Pais:
R <- ninguém, A <- R, S <- A

2 - Detectar e Recuperar o *Deadlock*

- Detecção de *Deadlock* - Um recurso de cada tipo
  - O algoritmo de DFS pode ser utilizado para detectar ciclos.



Lista de Nós não visitados (branca):
B, C, D, T, E, F, U, V, W, G

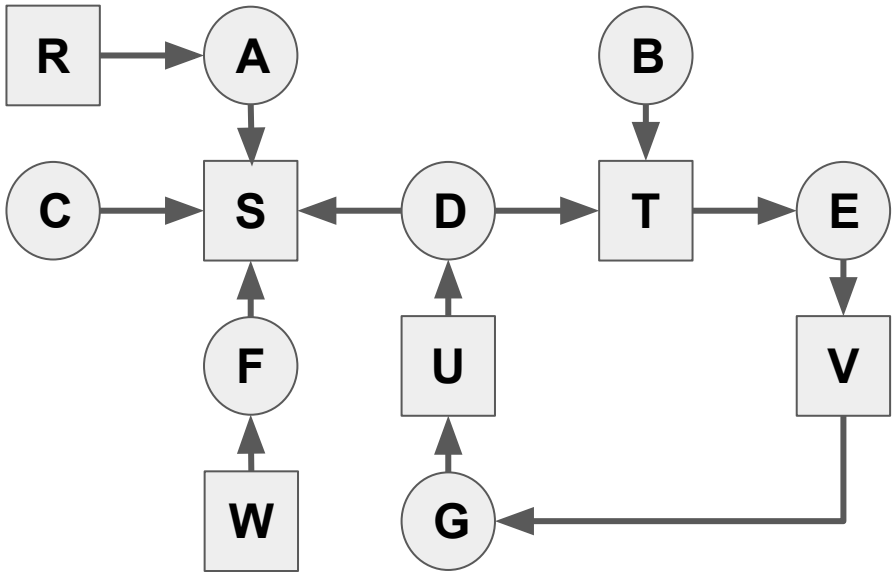
Lista de Nós Vistando no Momento (cinza):
R, A

Lista de Nós Visitados (preta):
S

Relação de Pais:
R <- ninguém, A <- R, S <- A

2 - Detectar e Recuperar o *Deadlock*

- Detecção de *Deadlock* - Um recurso de cada tipo
  - O algoritmo de DFS pode ser utilizado para detectar ciclos.



Lista de Nós não visitados (branca):
B, C, D, T, E, F, U, V, W, G

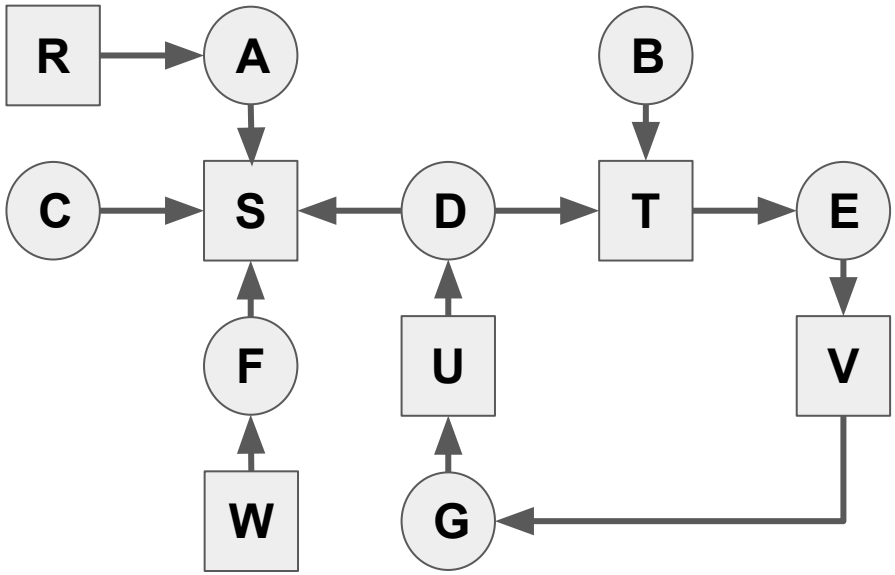
Lista de Nós Vistando no Momento (cinza):
R

Lista de Nós Visitados (preta):
S, A

Relação de Pais:
R <- ninguém, A <- R, S <- A

2 - Detectar e Recuperar o *Deadlock*

- Detecção de *Deadlock* - Um recurso de cada tipo
  - O algoritmo de DFS pode ser utilizado para detectar ciclos.



Lista de Nós não visitados (branca):
B, C, D, T, E, F, U, V, W, G

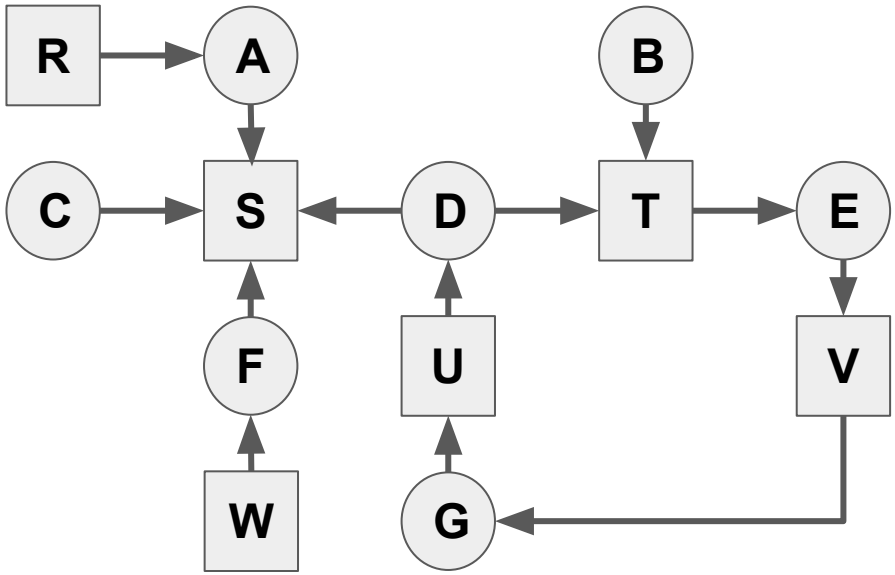
Lista de Nós Vistando no Momento (cinza):

Lista de Nós Visitados (preta):
S, A, R

Relação de Pais:
R <- ninguém, A <- R, S <- A

2 - Detectar e Recuperar o *Deadlock*

- Detecção de *Deadlock* - Um recurso de cada tipo
  - O algoritmo de DFS pode ser utilizado para detectar ciclos.



Lista de Nós não visitados (branca):
C, D, T, E, F, U, V, W, G

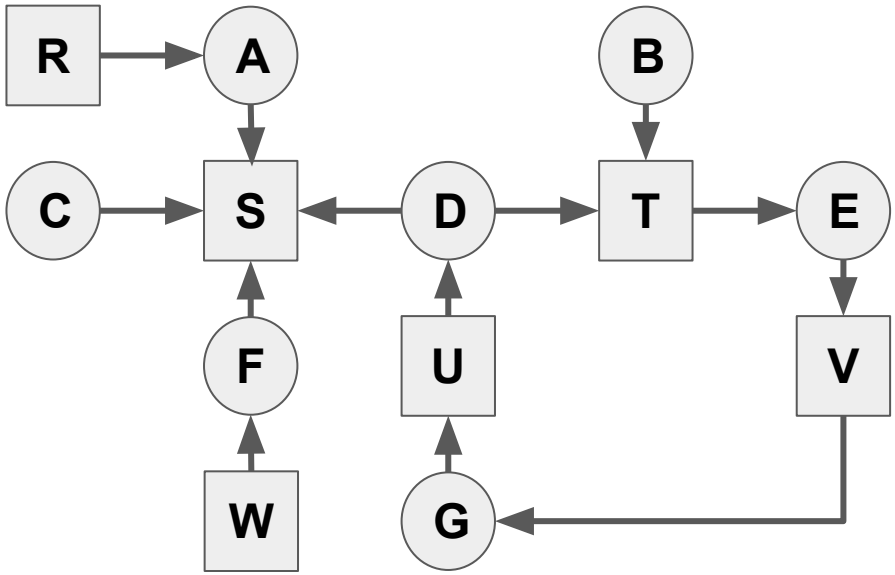
Lista de Nós Vistando no Momento (cinza):
B

Lista de Nós Visitados (preta):
S, A, R

Relação de Pais:
B <- ninguém,

2 - Detectar e Recuperar o *Deadlock*

- Detecção de *Deadlock* - Um recurso de cada tipo
  - O algoritmo de DFS pode ser utilizado para detectar ciclos.



Lista de Nós não visitados (branca):
C, D, E, F, U, V, W, G

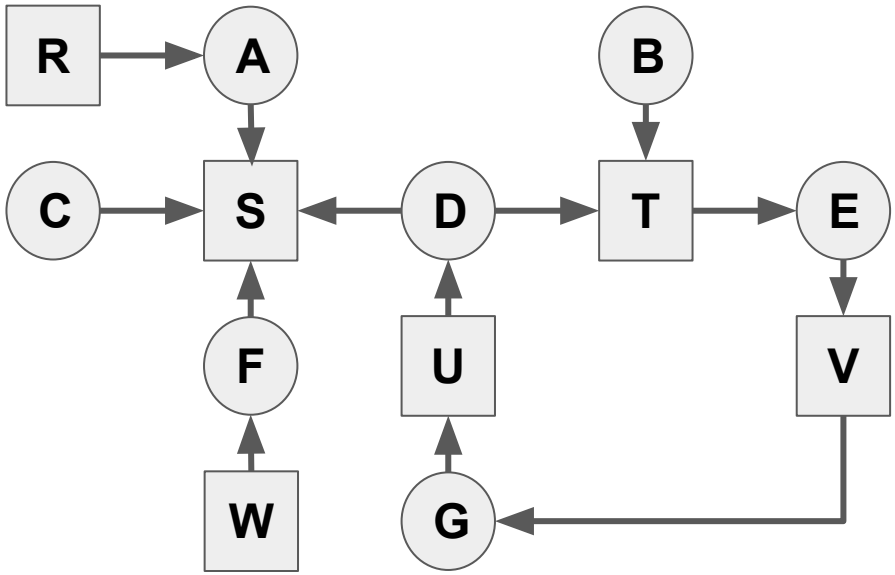
Lista de Nós Vistando no Momento (cinza):
B, T

Lista de Nós Visitados (preta):
S, A, R

Relação de Pais:
B <- ninguém, T <- B

2 - Detectar e Recuperar o *Deadlock*

- Detecção de *Deadlock* - Um recurso de cada tipo
  - O algoritmo de DFS pode ser utilizado para detectar ciclos.



Lista de Nós não visitados (branca):
C, D, F, U, V, W, G

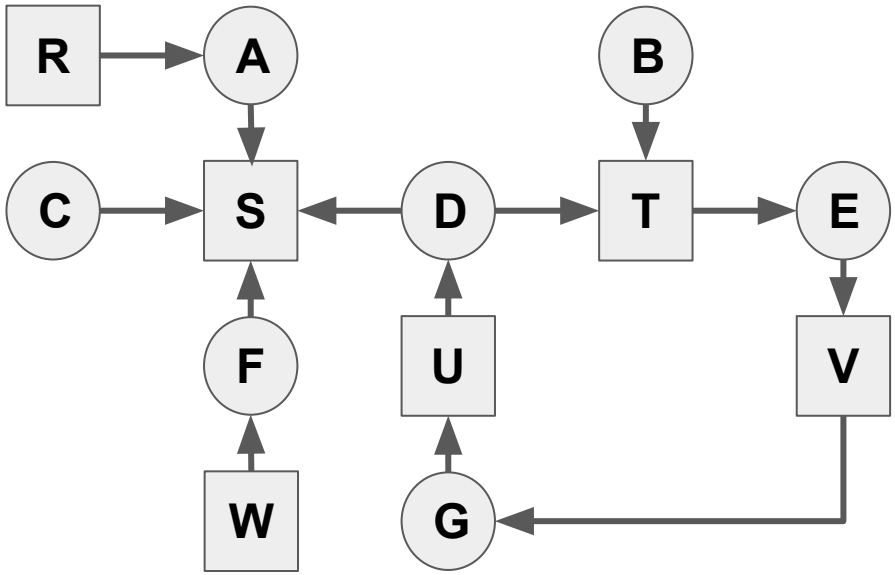
Lista de Nós Vistando no Momento (cinza):
B, T, E

Lista de Nós Visitados (preta):
S, A, R

Relação de Pais:
B <- ninguém, T <- B, E <- T

2 - Detectar e Recuperar o *Deadlock*

- Detecção de *Deadlock* - Um recurso de cada tipo
  - O algoritmo de DFS pode ser utilizado para detectar ciclos.



Lista de Nós não visitados (branca):
C, D, F, U, W, G

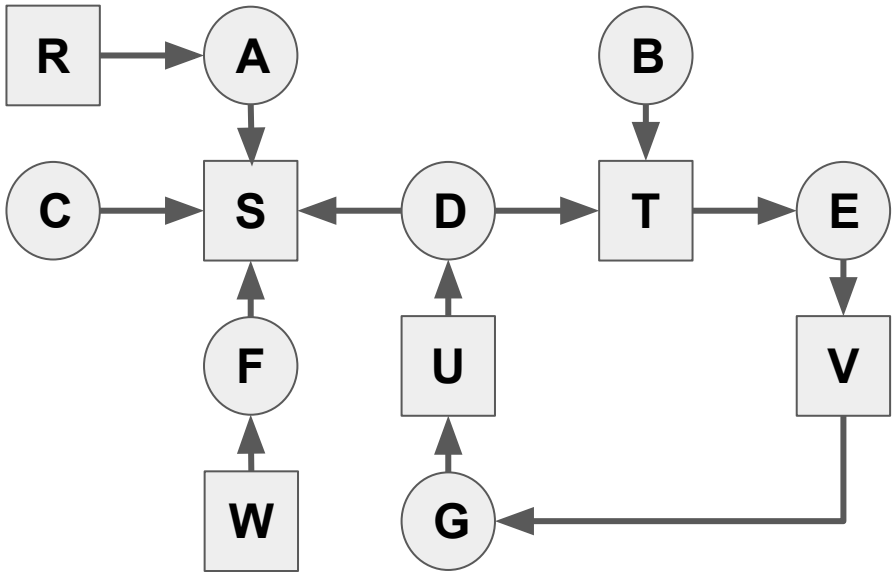
Lista de Nós Vistando no Momento (cinza):
B, T, E, V

Lista de Nós Visitados (preta):
S, A, R

Relação de Pais:
B <- ninguém, T <- B, E <- T, V <- E

2 - Detectar e Recuperar o *Deadlock*

- Detecção de *Deadlock* - Um recurso de cada tipo
  - O algoritmo de DFS pode ser utilizado para detectar ciclos.



Lista de Nós não visitados (branca):
C, D, F, U, W

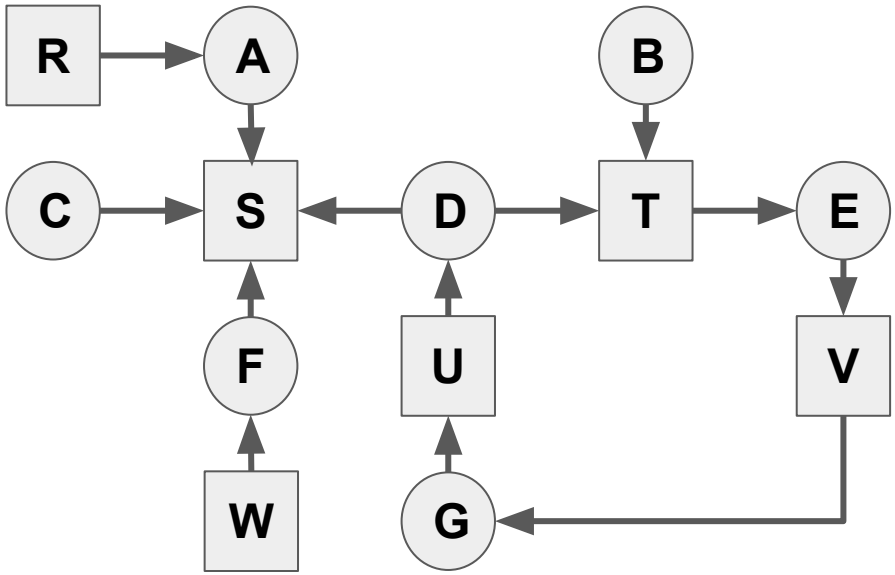
Lista de Nós Vistando no Momento (cinza):
B, T, E, V, G

Lista de Nós Visitados (preta):
S, A, R

Relação de Pais:
B <- ninguém, T <- B, E <- T, V <- E,
G <- V

2 - Detectar e Recuperar o *Deadlock*

- Detecção de *Deadlock* - Um recurso de cada tipo
  - O algoritmo de DFS pode ser utilizado para detectar ciclos.



Lista de Nós não visitados (branca):
C, D, F, W

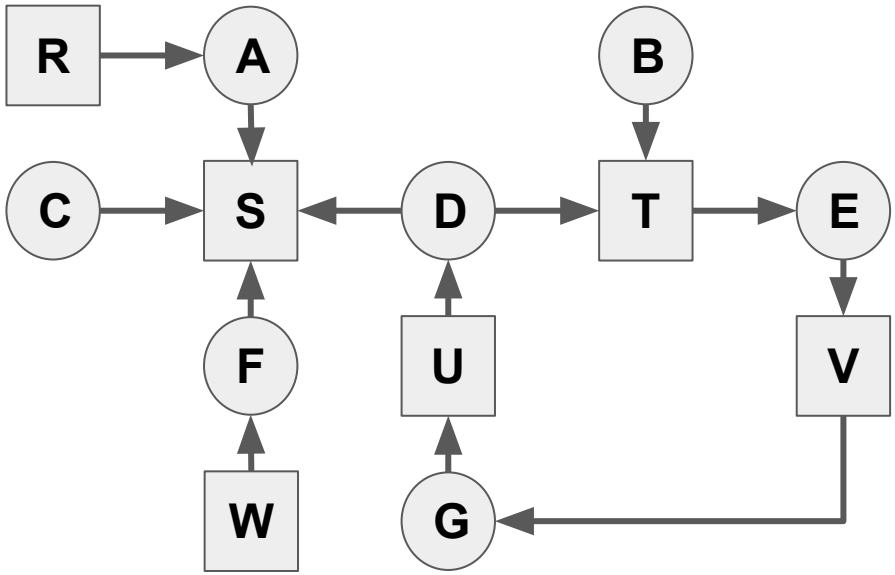
Lista de Nós Vistando no Momento (cinza):
B, T, E, V, G, U

Lista de Nós Visitados (preta):
S, A, R

Relação de Pais:
B <- ninguém, T <- B, E <- T, V <- E,
G <- V, U <- G

2 - Detectar e Recuperar o *Deadlock*

- Detecção de *Deadlock* - Um recurso de cada tipo
  - O algoritmo de DFS pode ser utilizado para detectar ciclos.



Lista de Nós não visitados (branca):
C, F, W

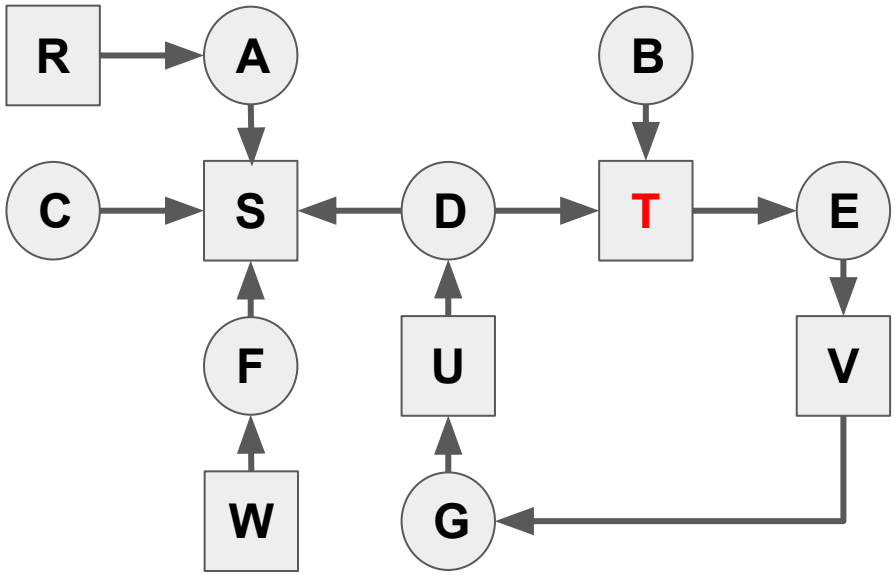
Lista de Nós Vistando no Momento (cinza):
B, T, E, V, G, U, D

Lista de Nós Visitados (preta):
S, A, R

Relação de Pais:
B <- ninguém, T <- B, E <- T, V <- E,
G <- V, U <- G, D <- U

2 - Detectar e Recuperar o *Deadlock*

- Detecção de *Deadlock* - Um recurso de cada tipo
  - O algoritmo de DFS pode ser utilizado para detectar ciclos.



Lista de Nós não visitados (branca):
C, F, W

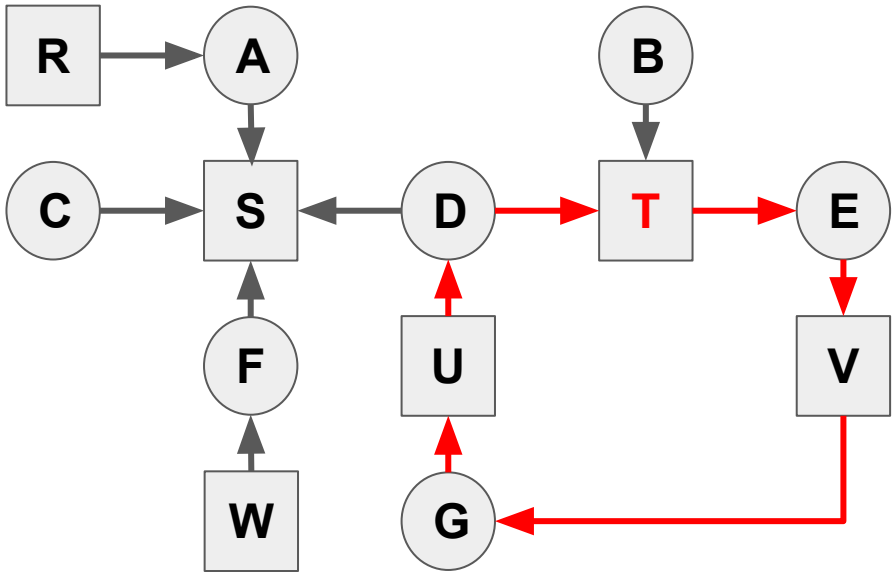
Lista de Nós Vistando no Momento (cinza):
B, T, E, V, G, U, D, T

Lista de Nós Visitados (preta):
S, A, R

Relação de Pais:
B <- ninguém, T <- B, E <- T, V <- E,
G <- V, U <- G, D <- U, T <- D

2 - Detectar e Recuperar o *Deadlock*

- Detecção de *Deadlock* - Um recurso de cada tipo
  - O algoritmo de DFS pode ser utilizado para detectar ciclos.



Lista de Nós não visitados (branca):
C, F, W

Lista de Nós Vistando no Momento (cinza):
B, T, E, V, G, U, D, T

Lista de Nós Visitados (preta):
S, A, R

Relação de Pais:
B <- ninguém, T <- B, E <- T, V <- E,
G <- V, U <- G, D <- U, T <- D

2 - Detectar e Recuperar o *Deadlock*

- Detecção de *Deadlock* - Vários recursos de cada tipo
  - Necessita de uma abordagem diferente da de grafos direcionados.
  - Utiliza uma estratégia de vetores de recursos existentes e disponíveis, combinados com duas matrizes de alocação atual e de requisições.

2 - Detectar e Recuperar o *Deadlock*

- Quando detectar um *deadlock*?
  - A cada nova solicitação de recurso
  - Periodicamente
  - Quando a utilização de CPU estiver baixa
    - Estar abaixo de um limiar, é indicativo de impasse por a CPU estar ociosa.

2 - Detectar e Recuperar o *Deadlock*

- Recuperação de *Deadlock*
  - Após detectado, um deadlock pode ser eliminado das seguintes formas:
    - Preempção
    - *Rollback*
    - Eliminação de Processos

2 - Detectar e Recuperar o *Deadlock*

- Recuperação de *Deadlock*
  - Preempção
    - Consiste em tomar o recurso do processo à força.
    - Depende do tipo de recurso
    - É uma operação complexa
    - A escolha de quem tomar o recurso pode ser direcionada.

2 - Detectar e Recuperar o *Deadlock*

- Recuperação de *Deadlock*
  - *Rollback*
    - Gravação de uma imagem da memória e estado atual.
    - Ao detectar um *deadlock*, faz-se *rollback* a um estado anterior à ocorrência do *deadlock*.
      - Os recursos envolvidos são entregues ao processo gerador do *deadlock*.

2 - Detectar e Recuperar o *Deadlock*

- Recuperação de *Deadlock*
  - Eliminação de Processos
    - Processos envolvidos no *deadlock* são eliminados gradativamente até que o ciclo seja quebrado.
    - A escolha da eliminação é normalmente direcionada a quem tem menos dependências.

3 - Evitar o *Deadlock*

- Algoritmos baseados no conceito de Estados Seguros.
  - Um estado é seguro se há uma maneira de satisfazer todas as requisições pendentes, partindo dos processos em execução.
  - Um estado é inseguro se não pode garantir que as requisições serão satisfeitas.

3 - Evitar o *Deadlock* - Exemplo 1

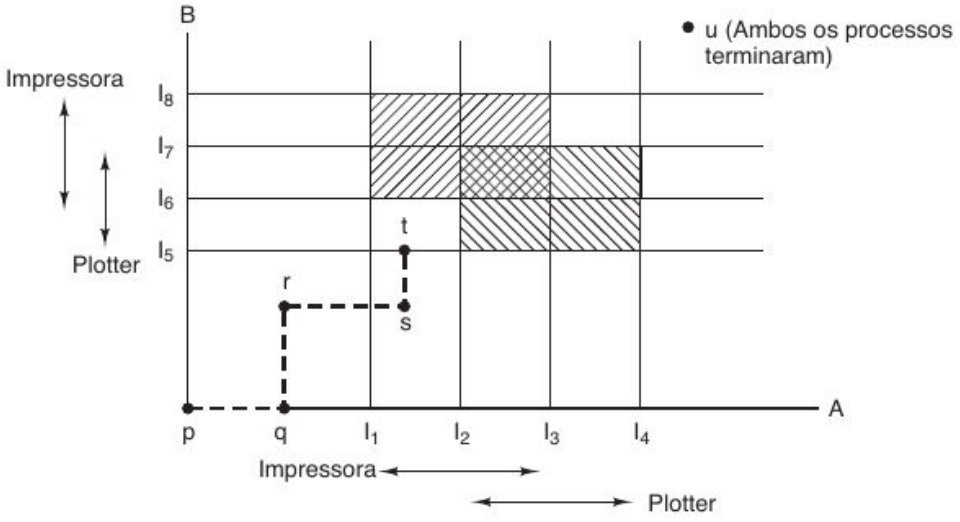
```
// Processo A
requisita(impressora);
requisita(plotter);
libera(impressora);
libera(plotter);
```

```
// Processo B
requisita(plotter);
requisita(impressora);
libera(plotter);
libera(impressora);
```

3 - Evitar o *Deadlock* - Exemplo 1

```
// Processo A
requisita(impressora);
requisita(plotter);
libera(impressora);
libera(plotter);
```

```
// Processo B
requisita(plotter);
requisita(impressora);
libera(plotter);
libera(impressora);
```



3 - Evitar o *Deadlock* - Exemplo 2

- Um estado é seguro se existe uma sequência de alocações que permite que todos os processos sejam concluídos.
  - Exemplo de estado seguro:

Possui máximo		
A	3	9
B	2	4
C	2	7

Disponível: 3
(a)

Possui máximo		
A	3	9
B	4	4
C	2	7

Disponível: 1
(b)

Possui máximo		
A	3	9
B	0	–
C	2	7

Disponível: 5
(c)

Possui máximo		
A	3	9
B	0	–
C	7	7

Disponível: 0
(d)

Possui máximo		
A	3	9
B	0	–
C	0	–

Disponível: 7
(e)

3 - Evitar o *Deadlock* - Exemplo 2

- Um estado é seguro se existe uma sequência de alocações que permite que todos os processos sejam concluídos.
  - Exemplo de estado inseguro (b):

	Possui máximo	
A	3	9
B	2	4
C	2	7

Disponível: 3
(a)

	Possui máximo	
A	4	9
B	2	4
C	2	7

Disponível: 2
(b)

	Possui máximo	
A	4	9
B	4	4
C	2	7

Disponível: 0
(c)

	Possui máximo	
A	4	9
B	–	–
C	2	7

Disponível: 4
(d)

3 - Evitar o *Deadlock*

- Algoritmo do Banqueiro (Dijkstra)
  - Objetivo: garantir linha de crédito aos correntistas do banco
  - Entidades:
    - Clientes (processos)
    - Unidade de Crédito (recursos)
    - Banqueiro (Sistema Operacional)

3 - Evitar o *Deadlock*

- Algoritmo do Banqueiro (Dijkstra)
  - Funcionamento:
    - Se uma solicitação de recursos leva a um estado seguro, ela é atendida.
    - Senão, a solicitação é adiada por algum tempo (processo fica bloqueado)
  - Uma solicitação é segura é o caso onde o banqueiro tem recursos suficientes para atender ao menos um cliente.

3 - Evitar o *Deadlock*

- Algoritmo do Banqueiro (Dijkstra)

	Possui	máximo
A	0	6
B	0	5
C	0	4
D	0	7

Disponível: 10

	Possui	máximo
A	1	6
B	1	5
C	2	4
D	4	7

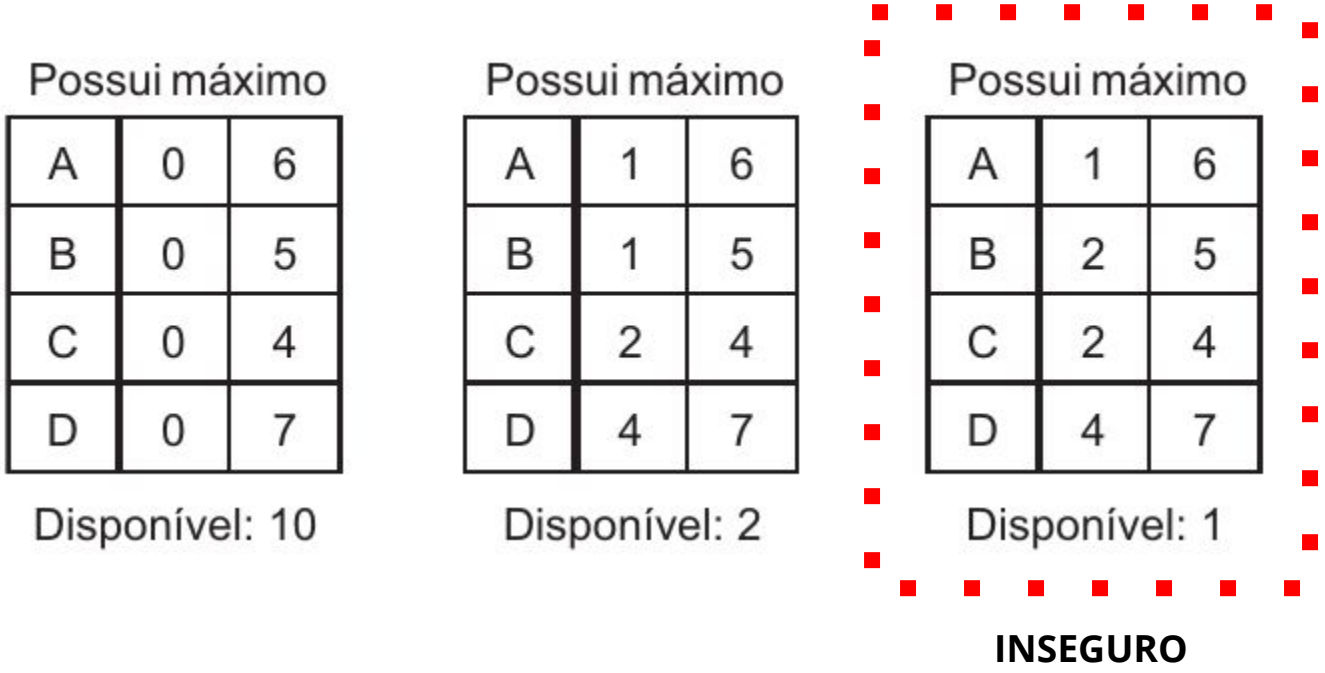
Disponível: 2

	Possui	máximo
A	1	6
B	2	5
C	2	4
D	4	7

Disponível: 1

3 - Evitar o *Deadlock*

- Algoritmo do Banqueiro (Dijkstra)



3 - Evitar o *Deadlock*

- Algoritmo do Banqueiro (Dijkstra)
  - Apesar de apresentado para somente um recurso, esse algoritmo já foi generalizado para vários tipos de recursos.
  - É um ótimo algoritmo na teoria, mas péssimo na prática.
    - Necessidade de conhecimento antecipado de necessidades máximas por recursos.
    - Processos são dinâmicos (seu número varia ao longo do tempo).

4 - Prevenir o *Deadlock*

- A prevenção parte para impedir que uma das quatro condições para ocorrência de *deadlocks* nunca seja verdadeira.
  - Impossibilitar a exclusão mútua.
  - Impossibilitar a posse e espera.
  - Impossibilitar a não-preempção.
  - Impossibilitar a espera circular.

4 - Prevenir o *Deadlock*

- Impossibilitar a exclusão mútua.
  - Restringir ao máximo o número de processos que podem solicitar um recurso.
    - Exemplo:
      - Recurso impressora é centralizado em um gerenciador de impressão (*spool* de impressão).
    - Problema:
      - Nem todos os recursos podem ser gerenciados por um único processo.

4 - Prevenir o *Deadlock*

- Impossibilitar a posse e espera.
  - Técnica 1:
    - Fazer com que os processos solicitem todos os recurso que necessitam antes da sua execução.
    - Problema: necessidade de antecipação
  - Técnica 2:
    - Um processo que solicita um recurso, deve liberar todos os recursos que adquiriu anteriormente e em seguida adquirir todos os recurso que necessita de uma vez só.
    - Problema: complexidade de programação

4 - Prevenir o *Deadlock*

- Impossibilitar a não-preempção.
  - Consiste em garantir que sempre haja uma maneira de retirar o recurso de um processo.
  - Problema: nem sempre é útil e pode conduzir a resultados inesperados.

4 - Prevenir o *Deadlock*

- Impossibilitar a espera circular.
  - Consiste em impedir ciclos na alocação de recursos.
  - Técnica: impor uma numeração global de todos os recursos do sistema e fazer com que os processos sigam esta ordem na requisição de recursos.
  - Problema: Lista muito grande de quantidade de recursos.

Deadlocks - Outras Questões

- *Livelocks*:
  - Processos não estão bloqueados, mas encontram-se presos em algum ponto de sua execução.
  - Caso onde processos, por extrema educação, estão sempre cedendo seus recursos uns aos outros e ninguém, de fato, evolui no seu processamento.

Deadlocks - Outras Questões

- *Starvation* (Inanição):
  - Caso onde a política de alocação não causa *deadlock*, porém é possível que processos nunca sejam atendidos.
    - Também conhecido como postergação indefinida.
  - Exemplo:
    - Política de alocação de impressora:
      - O processo escolhido é aquele com o menor número de páginas.
      - Se o fluxo de impressão for contínuo, arquivos grandes não serão impressos.

Deadlocks - Conclusão

- Deadlock é um problema que ocorre quando:
  - Processos detém recursos de maneira exclusiva.
  - Estando de posse de um recurso, um processo pode pedir por outro recurso.
  - Um recurso só pode ser liberado pelo processo que o detém.
  - Há a presença de um ciclo de alocação dos recursos desejados.

Deadlocks - Conclusão

- Mesmo com todas as soluções existentes, o algoritmo do avestruz ainda é o mais empregado.
  - O tratamento de *deadlocks* é uma operação complexa.
  - A maioria dos Sistemas Operacionais não utiliza nenhuma técnica de tratamento de *deadlocks*.
  - O programador precisa ser competente o suficiente.