

# Capítulo XII

## Redes de Fluxos

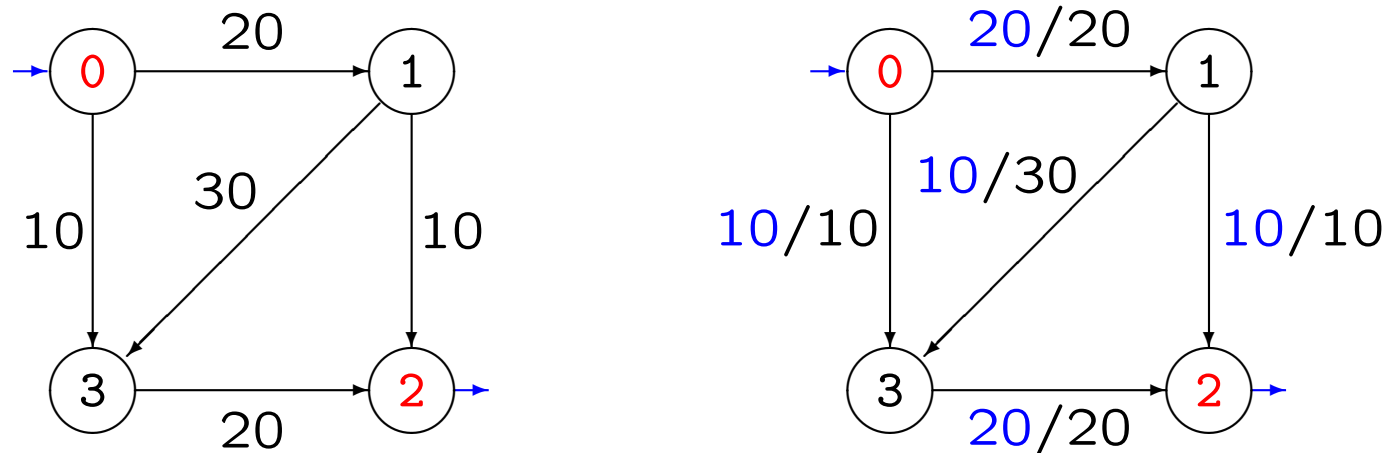
# Fluxo Máximo entre Dois Vértices

---

## Algoritmo de Edmonds-Karp

# Problema

Dado um grafo **orientado** e **pesado** (com pesos não negativos) e dois vértices  $f$  e  $d$ , como encontrar um **fluxo máximo de  $f$  para  $d$** ?

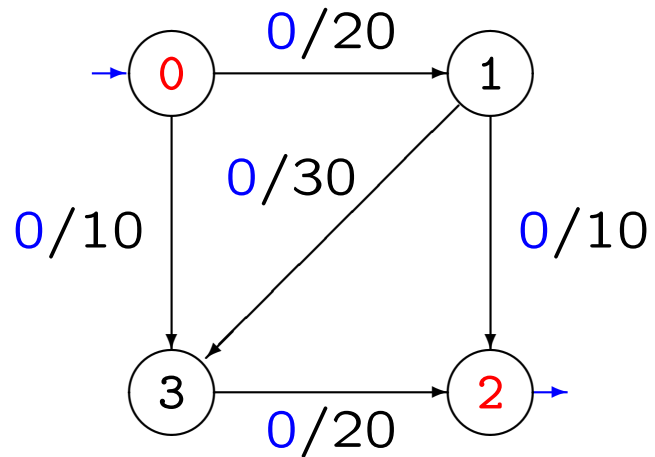


**Valor de um Fluxo Máximo de 0 para 2: 30**

**Assume-se** que qualquer vértice pertence a um caminho de  $f$  para  $d$ .

Portanto, o grafo é **simplesmente conexo** e  $|A| \geq |V| - 1$ .

## Ideia Geral (de 0 para 2)



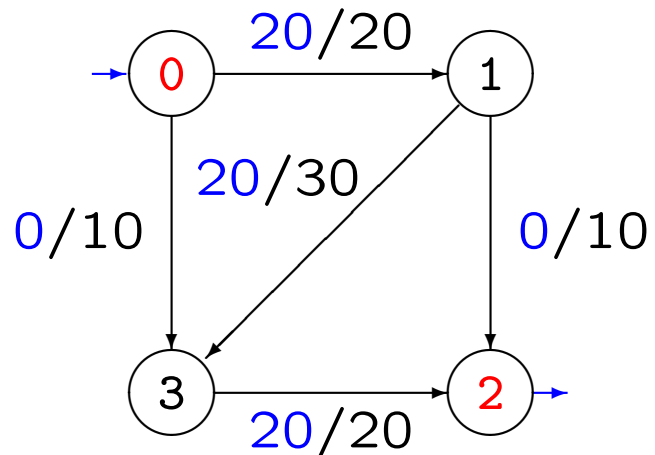
Descobrir um caminho para drenar:

0, 1, 3, 2.

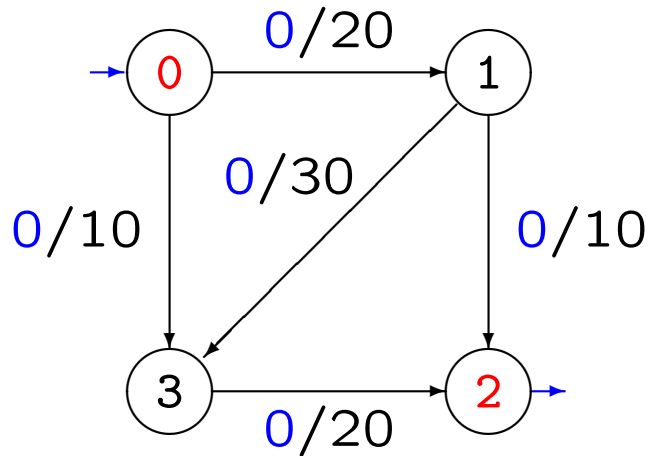
Incremento máximo permitido:

20.

Atualizar o fluxo.



## Ideia Geral (de 0 para 2)



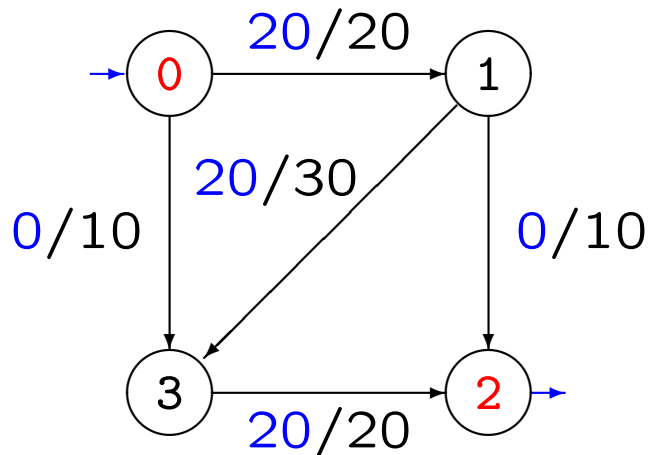
Descobrir um caminho para drenar:

0, 1, 3, 2.

Incremento máximo permitido:

20.

Atualizar o fluxo.



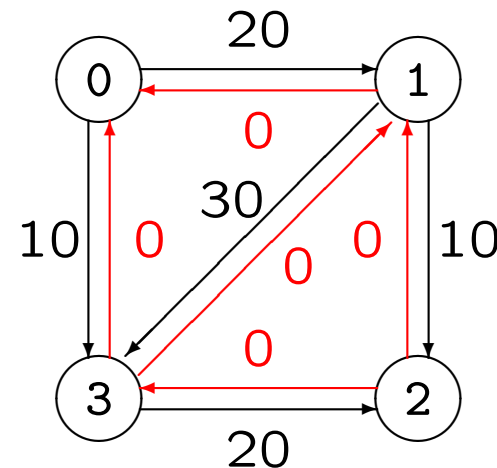
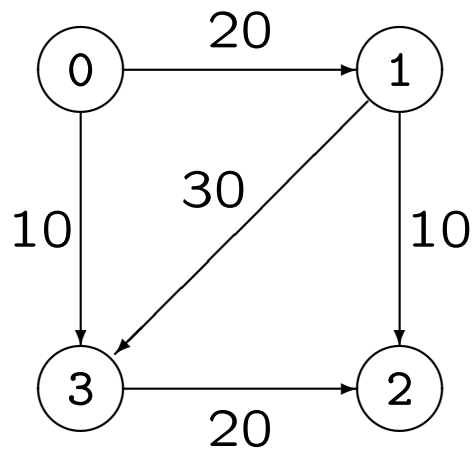
### PROBLEMA:

- já não há caminho para drenar; e
- o valor de um fluxo máximo é superior a 20.

# Rede de Fluxos

Seja  $G = (V, A)$  um grafo orientado e pesado, cujos arcos têm um peso não negativo (que representa a **capacidade** do arco). A **rede de fluxos** de  $G$  é o grafo  $R = (V, A')$ , orientado e pesado, tal que:

- $A \subseteq A'$  (a rede tem todos os arcos de  $G$ );
- Se  $(v, w) \in A$  e  $(w, v) \notin A$ , a rede tem o arco  $(w, v)$  com peso zero.



# Fluxo

Sejam:

- $R = (V, A')$  uma rede de fluxos;
- $f \in V$  o vértice **fonte**;
- $d \in V$  o vértice **dreno**; e
- $c(v, w)$  a capacidade do arco  $(v, w) \in A'$ .

Um **fluxo** em  $R$  é uma função  $\phi : A' \rightarrow \mathbb{R}$  que satisfaz as seguintes propriedades.

- (Capacidade)  $\phi(v, w) \leq c(v, w)$ , para qualquer  $(v, w) \in A'$ .
- (Simetria)  $\phi(v, w) = -\phi(w, v)$ , para qualquer  $(v, w) \in A'$ .
- (Conservação)  $\sum_{\{w | (v, w) \in A'\}} \phi(v, w) = 0$ , para qualquer  $v \in V \setminus \{f, d\}$ .

O **valor do fluxo**  $\phi$  da fonte  $f$  para o dreno  $d$  é:

$$\sum_{\{v | (f, v) \in A'\}} \phi(f, v).$$

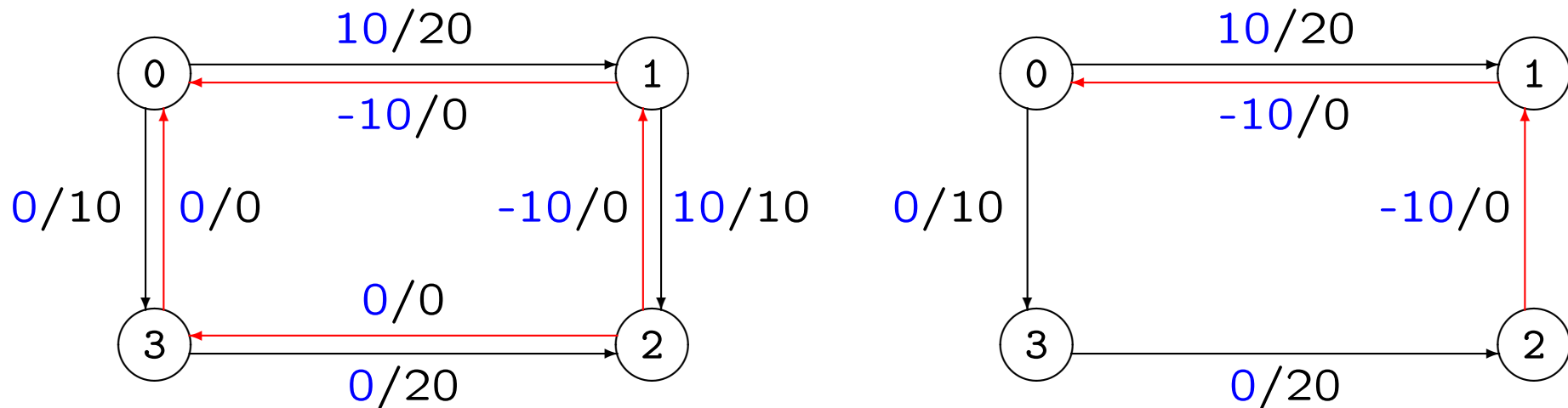
# Rede Residual

Sejam:

- $R = (V, A')$  uma rede de fluxos;
- $c(v, w)$  a capacidade do arco  $(v, w) \in A'$ ; e
- $\phi$  um fluxo em  $R$ .

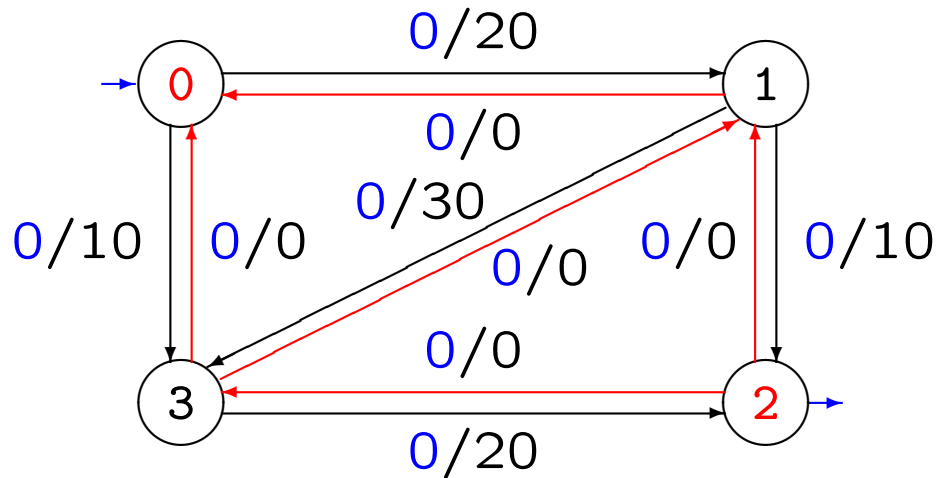
A **rede residual** de  $R$  induzida por  $\phi$  é o grafo orientado  $R_\phi = (V, A'')$ , onde:

$$A'' = \{(v, w) \in A' \mid c(v, w) - \phi(v, w) > 0\}.$$





# Método de Ford-Fulkerson [1962]



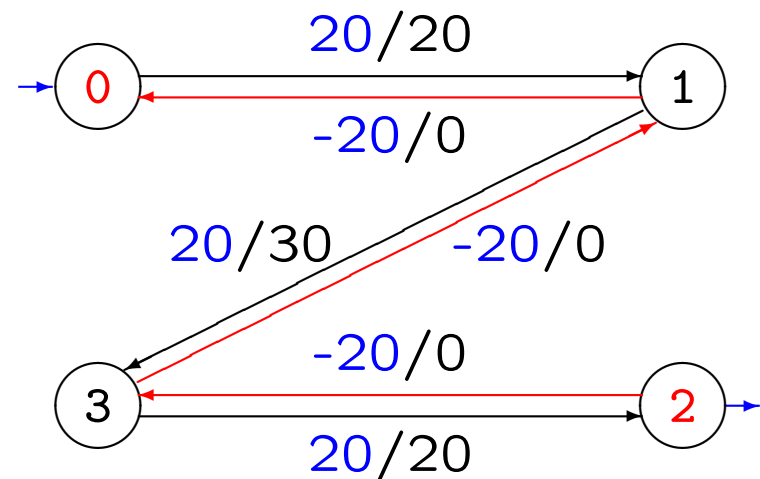
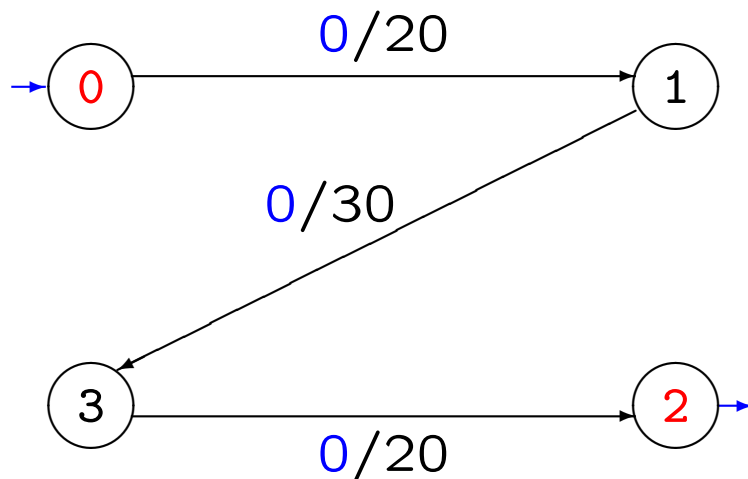
Caminho na rede residual:

0, 1, 3, 2.

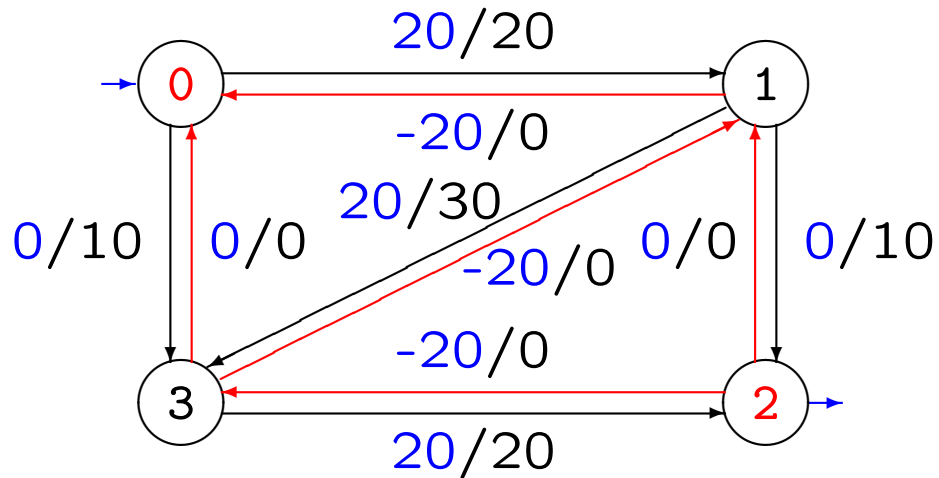
Incremento máximo permitido:

20.

Atualizar o fluxo.



# Método de Ford-Fulkerson (2)



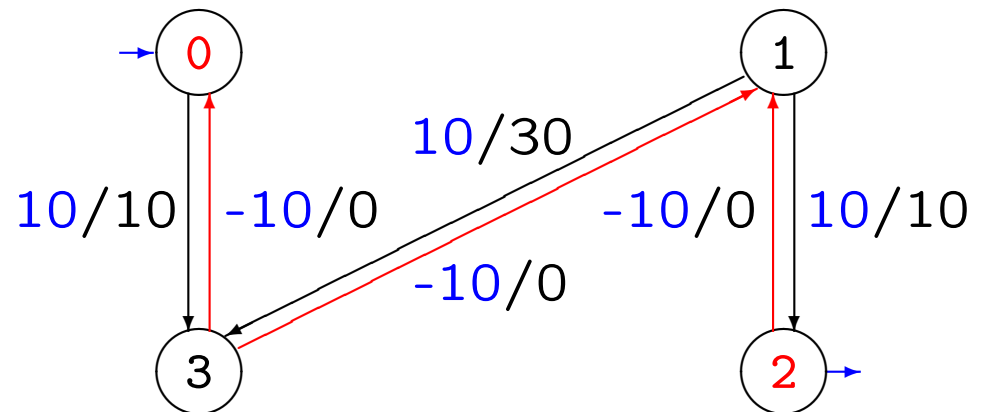
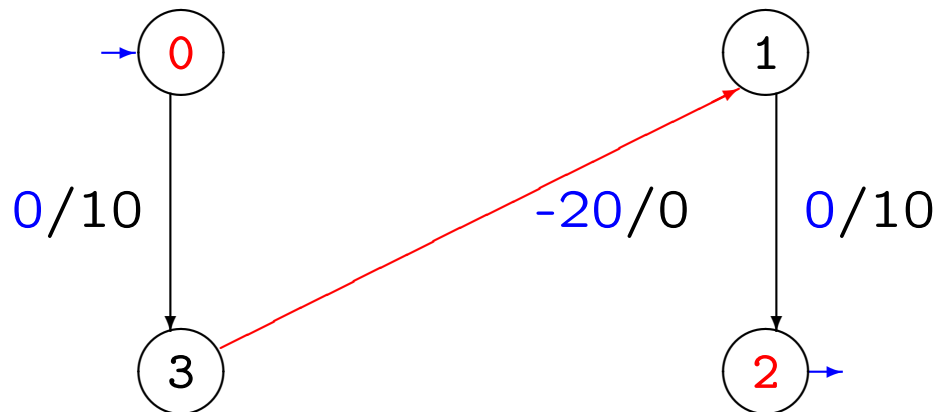
Caminho na rede residual:

0, 3, 1, 2.

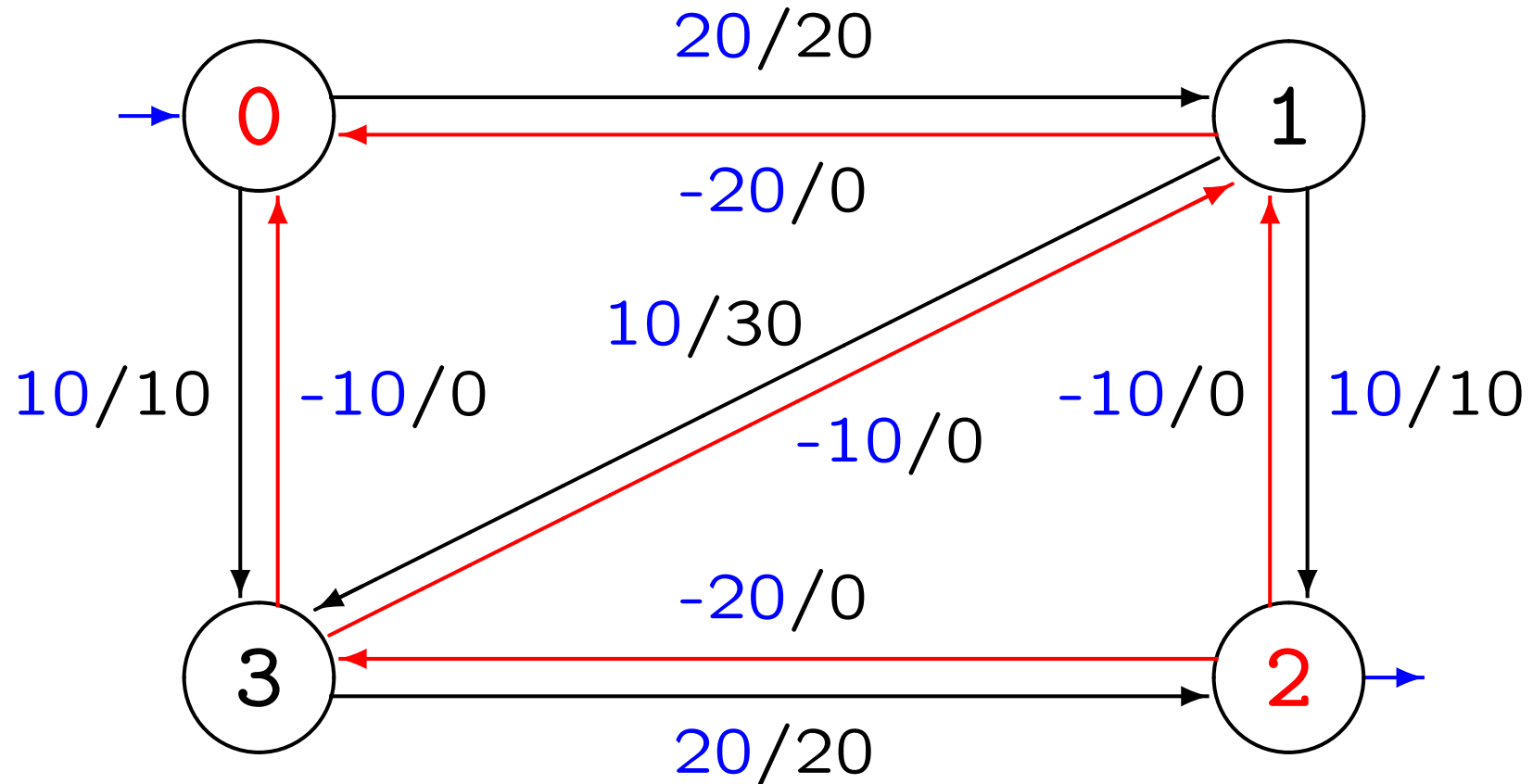
Incremento máximo permitido:

10.

Atualizar o fluxo.



## Método de Ford-Fulkerson (3)



Como não há caminho de 0 para 2 na rede residual, o fluxo é máximo (e o seu valor é 30).

# Observações

- O **método** de Ford-Fulkerson não especifica como se descobrem os caminhos da fonte  $f$  para o dreno  $d$  na rede residual.
- Se os pesos dos arcos forem números **irracionais**, o método pode não convergir.
- Se os pesos dos arcos forem números **racionais**, podem ser convertidos para números inteiros.
- Se os pesos dos arcos forem números **inteiros**, o número máximo de iterações não excede ...

**Justificação:**

# Observações

- O **método** de Ford-Fulkerson não especifica como se descobrem os caminhos da fonte  $f$  para o dreno  $d$  na rede residual.
- Se os pesos dos arcos forem números **irracionais**, o método pode não convergir.
- Se os pesos dos arcos forem números **racionais**, podem ser convertidos para números inteiros.
- Se os pesos dos arcos forem números **inteiros**, o número máximo de iterações não excede o valor dos fluxos máximos.

**Justificação:** no início, o valor do fluxo é zero e, em cada iteração, o valor do fluxo é incrementado em, pelo menos, uma unidade.

# Algoritmo de Edmonds-Karp [1972]

Aplica o método de Ford-Fulkerson,  
descobrendo os caminhos da fonte  $f$  para o dreno  $d$   
com uma pesquisa em largura na rede residual.

Esses caminhos têm comprimento mínimo,  
considerando que cada arco tem custo (distância) um.

# Construir a Rede de Fluxos

```
Digraph<L> buildNetwork( Digraph<L> graph )
{
    // The network has every vertex and every edge in the graph.
    Digraph<L> network = graph.clone();

    // For every edge  $(v, w)$  in the graph  $(V, E)$ , if  $(w, v) \notin E$ ,
    // add edge  $(w, v)$  with capacity zero into the network.
    for every Edge<L> e in graph.edges()
    {
        Node[] endPoints = e.endNodes();
        if ( !graph.edgeExists(endPoints[1], endPoints[0]) )
            network.addEdge(endPoints[1], endPoints[0], 0);
    }
    return network;
}
```

# Fluxo Máximo

```
Pair<L, L[][]> edmondsKarp( Digraph<L> graph, Node source,  
    Node sink )  
{  
    Digraph<L> network = buildNetwork(graph);  
    int numNodes = network.numNodes();  
    L[][] flow = new L[numNodes][numNodes];  
    for every Edge<L> e in network.edges()  
    {  
        Node[] endPoints = e.endNodes();  
        flow[ endPoints[0] ][ endPoints[1] ] = 0;  
    }  
    Node[] via = new Node[numNodes];  
    L flowValue = 0;  
    L increment;
```



```

while ( ( increment = findPath(network, flow, source, sink, via) )
        != 0 )
{
    flowValue += increment;
    // Update flow.
    Node node = sink;
    while ( node != source )
    {
        Node origin = via[node];
        flow[origin][node] += increment;
        flow[node][origin] -= increment;
        node = origin;
    }
}
return new PairClass<L, L[][]>(flowValue, flow);
}

```

```

L findPath( Digraph<L> network,  L[][] flow,  Node source,
             Node sink,  Node[] via )
{
  int numNodes = network.numNodes();

  Queue<Node> waiting = new QueueIn...<Node>( ? );

  boolean[] found = new boolean[numNodes];
  for every Node v in network.nodes()
    found[v] = false;

  L[] pathIncr = new L[numNodes];

  waiting.enqueue(source);
  found[source] = true;
  via[source] = source;
  pathIncr[source] =  $+\infty$ ;

```

```

do {
    Node origin = waiting.dequeue();
    for every Edge<L> e in network.outIncidentEdges(origin)
    {
        Node destin = e.endNodes()[1];
        L residue = e.label() – flow[origin][destin];
        if ( !found[destin]  &&  residue > 0 )
        {
            via[destin] = origin;
            pathIncr[destin] = Math.min(pathIncr[origin], residue);
            // destin == sink? Tratamento diferente.
        }
    }
}

while ( !waiting.isEmpty() );
return 0;
}

```

```

for every Edge<L> e in network.outIncidentEdges(origin)
{
    Node destin = e.endNodes()[1];
    L residue = e.label() – flow[origin][destin];
    if ( !found[destin]  &&  residue > 0 )
    {
        via[destin] = origin;
        pathIncr[destin] = Math.min(pathIncr[origin], residue);
        if ( destin == sink )
            return pathIncr[destin];

        waiting.enqueue(destin);
        found[destin] = true;
    }
}

```

# Complexidade do Algoritmo de Edmonds-Karp

Grafo  $G = (V, A)$  em Vetor de Listas de Incidências

construir a rede de fluxos  $O(|V| \times |A|)$

inicializar o fluxo  $\Theta(|A|)$

$k \times$  descobrir um caminho  $O(k \times |A|)$

$k \times$  atualizar o fluxo  $O(k \times |V|)$

**TOTAL**  $O((|V| + k) \times |A|)$

# Complexidade do Algoritmo de Edmonds-Karp

Grafo  $G = (V, A)$  em Vetor de Listas de Incidências

construir a rede de fluxos  $O(|V| \times |A|)$

inicializar o fluxo  $\Theta(|A|)$

$k \times$  descobrir um caminho  $O(k \times |A|)$

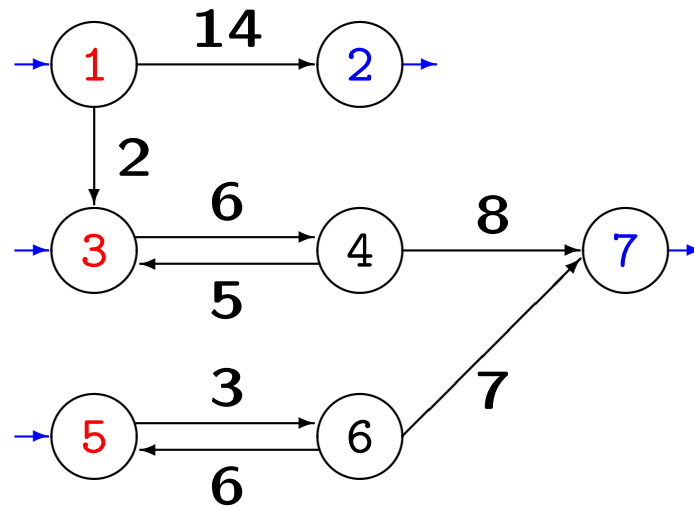
$k \times$  atualizar o fluxo  $O(k \times |V|)$

**TOTAL**  $O((|V| + k) \times |A|)$

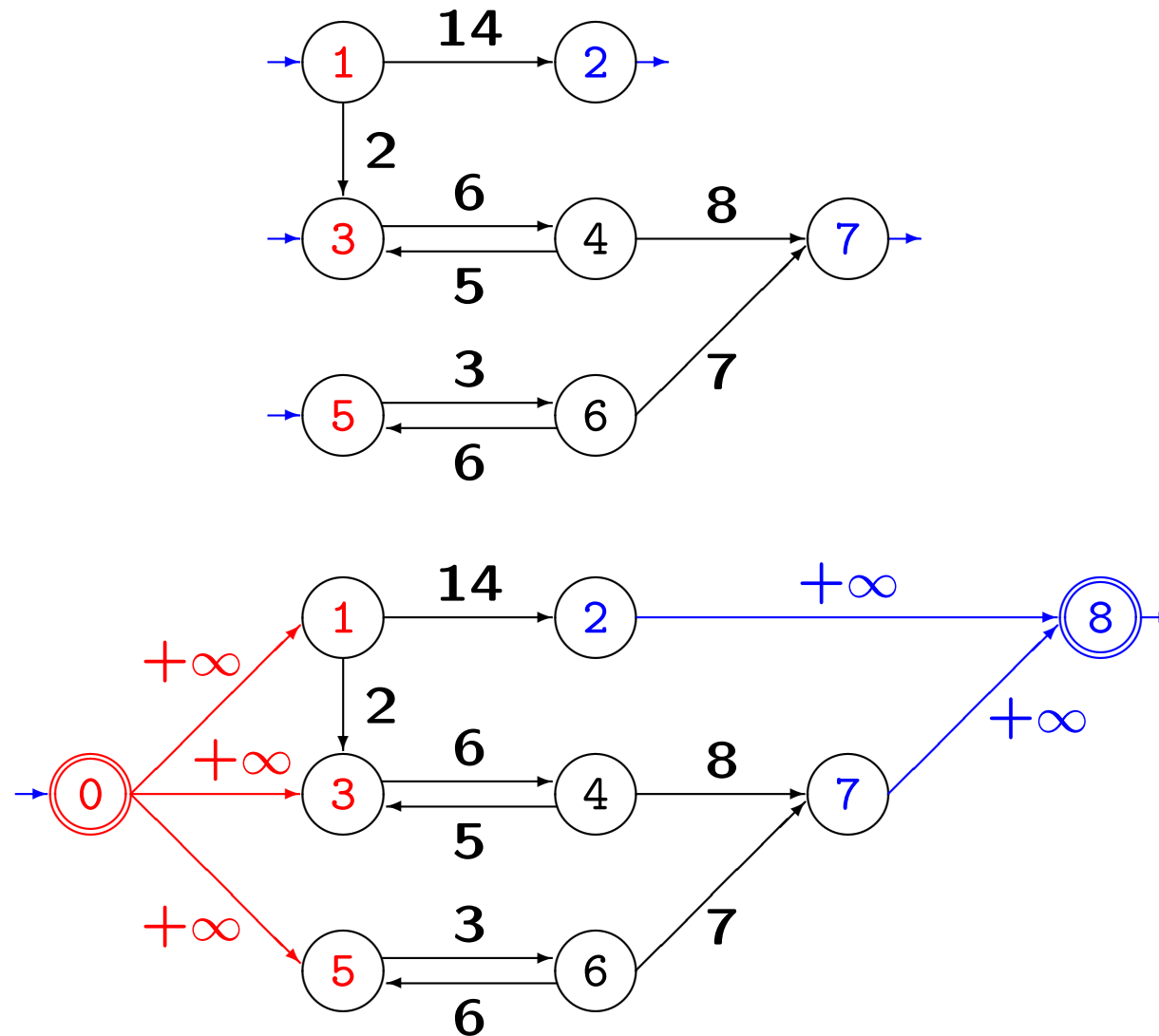
**Resultado:** O número de iterações ( $k$ ) do algoritmo de Edmonds-Karp não excede  $|V| \times |A|$  (nem excede o valor dos fluxos máximos).

Portanto, a complexidade do algoritmo é  $O(|V| \times |A|^2)$ .

# Generalização para Múltiplas Fontes ou Drenos



# Generalização para Múltiplas Fontes ou Drenos



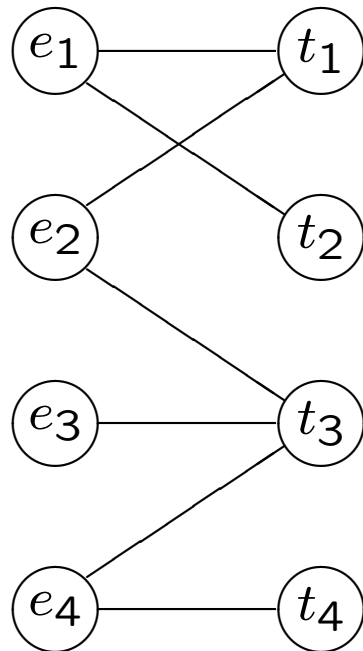


# Emparelhamento Máximo num grafo bipartido — Transformação e Conquista

# Exemplo de Problema

Cada empregado pode executar, no máximo, uma tarefa e cada tarefa só pode ser atribuída a um empregado.

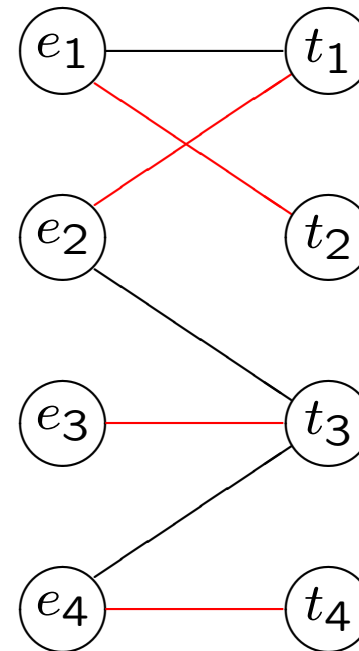
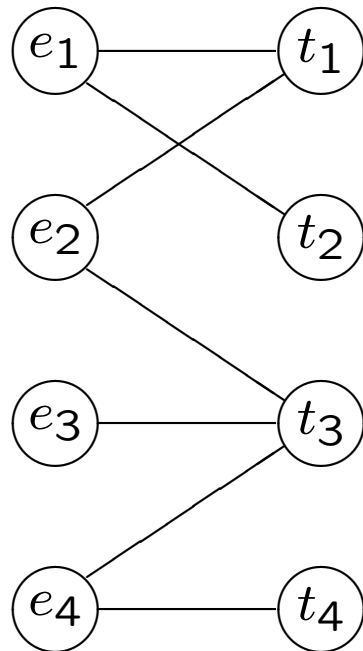
Como distribuir as tarefas pelos empregados, maximizando o número de tarefas atribuídas (ou seja, maximizando o número de empregados com trabalho)?



# Exemplo de Problema

Cada empregado pode executar, no máximo, uma tarefa e cada tarefa só pode ser atribuída a um empregado.

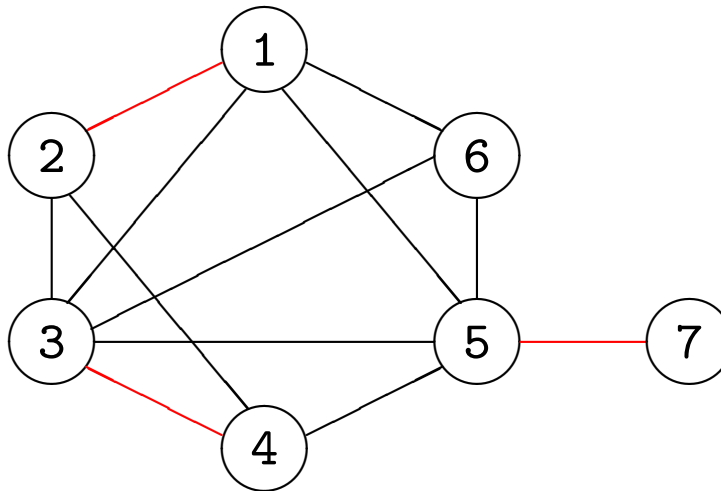
Como distribuir as tarefas pelos empregados, maximizando o número de tarefas atribuídas (ou seja, maximizando o número de empregados com trabalho)?



# Emparelhamento

Seja  $G = (V, A)$  um grafo **não orientado**. Um **emparelhamento**  $E$  em  $G$  é um subconjunto de arcos de  $G$  não adjacentes entre si. Ou seja,

- $E \subseteq A$  e
- para todo o vértice  $v \in V$ , existe no máximo um arco em  $E$  que tem uma extremidade em  $v$ .



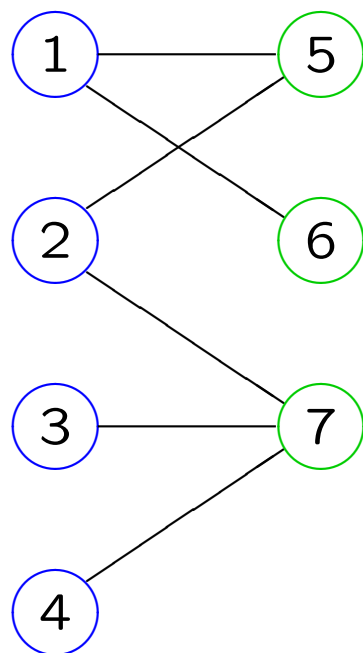
Um **emparelhamento máximo** é um emparelhamento de cardinalidade máxima.

# Grafo Bipartido

Um grafo **não orientado**  $G = (V, A)$  diz-se **bipartido** se existir uma partição  $\{X, Y\}$  do conjunto  $V$

(ou seja,  $X \neq \emptyset$ ,  $Y \neq \emptyset$ ,  $X \cup Y = V$  e  $X \cap Y = \emptyset$ )

tal que todos os arcos de  $G$  têm uma extremidade em  $X$  e outra em  $Y$ .

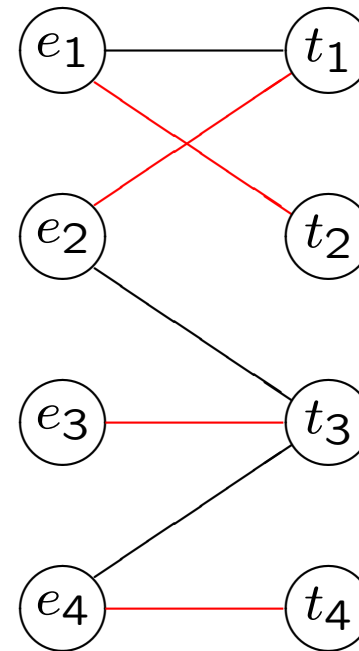
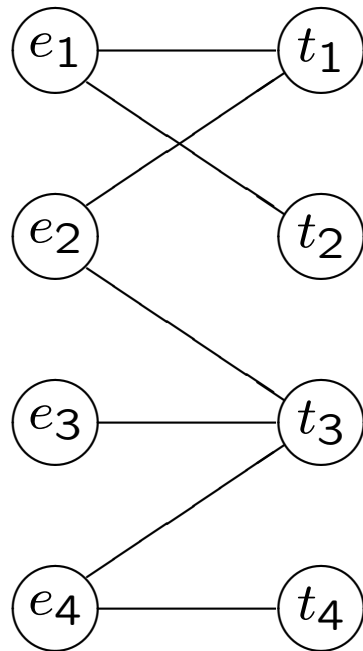


$X = \{1, 2, 3, 4\}$

$Y = \{5, 6, 7\}$

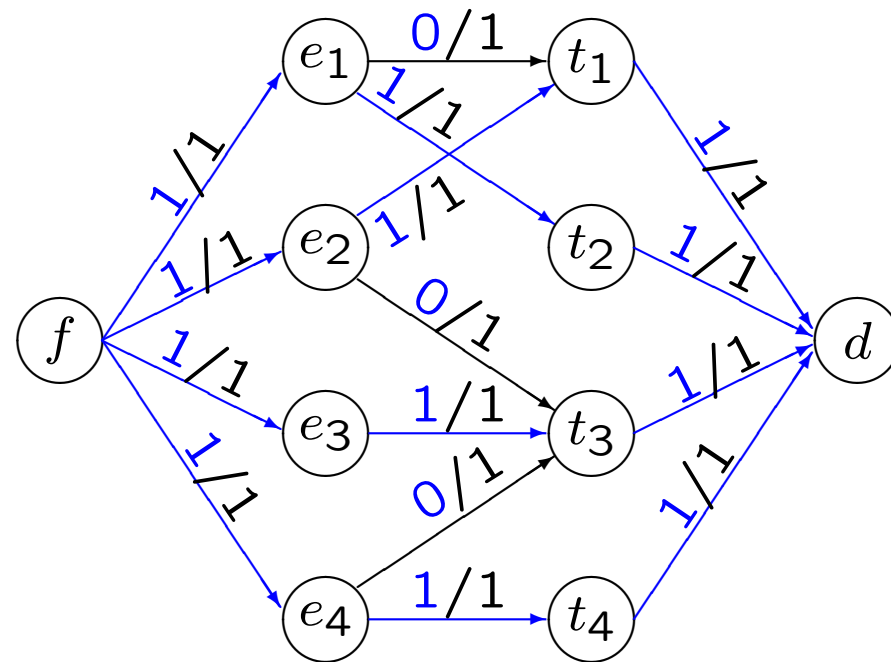
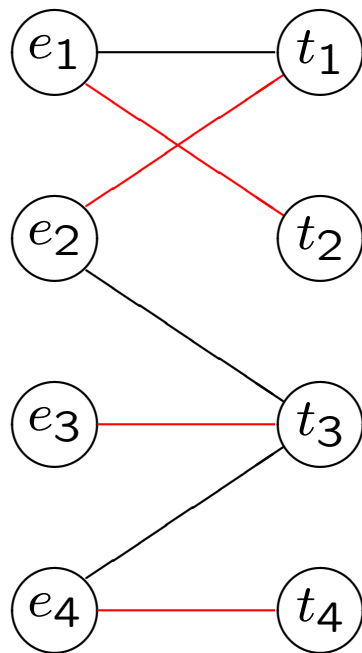
# Exemplo de Problema

Como distribuir as tarefas pelos empregados, maximizando o número de tarefas atribuídas (ou seja, maximizando o número de empregados com trabalho)?



Como encontrar um **emparelhamento máximo** num grafo **não orientado** e **bipartido**?

# Fluxo Máximo num Grafo Orientado e Pesado



Há uma correspondência “perfeita” entre o emparelhamento máximo  $E$  e o fluxo máximo  $\phi$  de  $f$  para  $d$ :

- o número de arcos de  $E$  é o valor de  $\phi$ ;
- para todos os arcos da forma  $(e, t)$ ,  $(e, t) \in E \Leftrightarrow \phi(e, t) = 1$ .

# Emparelhamento Máximo num Grafo Bipartido

Sejam  $G = (V, A)$  um grafo não orientado e bipartido, e  $\{X, Y\}$  uma partição de  $V$ . Seja  $G' = (V', A')$  o grafo orientado e pesado tal que:

$V' = V \cup \{f, d\}$ , onde  $f$  e  $d$  são dois novos vértices ( $f, d \notin V$ );

$A' = \{(f, x) \mid x \in X\} \cup \{(x, y) \mid (x, y) \in A, x \in X, y \in Y\} \cup \{(y, d) \mid y \in Y\}$ ;

e todos os arcos de  $A'$  têm peso 1.

## Proposição

Há uma correspondência biunívoca entre cada emparelhamento máximo  $E$  em  $G$  e cada fluxo máximo  $\phi$  de  $f$  para  $d$  em  $G'$ :

- o número de arcos de  $E$  é o valor de  $\phi$ ;
- para todos os arcos  $(x, y) \in A$ , com  $x \in X$  e  $y \in Y$ ,

$$(x, y) \in E \Leftrightarrow \phi(x, y) = 1.$$



# Complexidade do Emparelhamento Máximo com o Algoritmo de Edmonds-Karp

Grafo  $G = (V, A)$  em Vetor de Listas de Incidências  
cuja partição de vértices  $\{X, Y\}$  verifica  
 $X = \{0, 1, \dots, n - 1\}$  e  $Y = \{n, n + 1, \dots, |V| - 1\}$

construir a rede de fluxos  $\Theta(|V| + |A|)$

inicializar o fluxo  $\Theta(|V| + |A|)$

$k \times$  descobrir um caminho  $O(k \times (|V| + |A|))$

$k \times$  atualizar o fluxo  $O(k \times |V|)$

**TOTAL**  $O(k \times (|V| + |A|))$

Como o número de iterações ( $k$ ) não excede .....,  
a complexidade do algoritmo é .....

# Complexidade do Emparelhamento Máximo com o Algoritmo de Edmonds-Karp

Grafo  $G = (V, A)$  em Vetor de Listas de Incidências  
cuja partição de vértices  $\{X, Y\}$  verifica  
 $X = \{0, 1, \dots, n - 1\}$  e  $Y = \{n, n + 1, \dots, |V| - 1\}$

construir a rede de fluxos  $\Theta(|V| + |A|)$

inicializar o fluxo  $\Theta(|V| + |A|)$

$k \times$  descobrir um caminho  $O(k \times (|V| + |A|))$

$k \times$  atualizar o fluxo  $O(k \times |V|)$

**TOTAL**  $O(k \times (|V| + |A|))$

Como o número de iterações ( $k$ ) não excede  $\frac{|V|}{2}$ ,

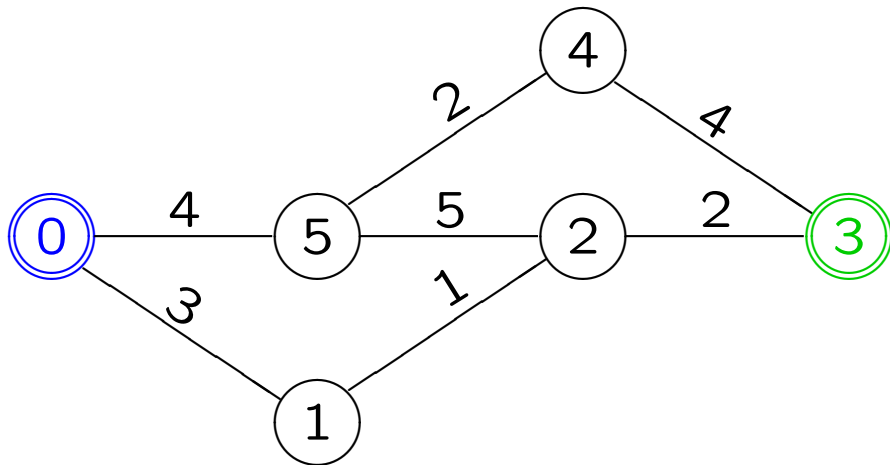
a complexidade do algoritmo é  $O(|V| \times (|V| + |A|))$ .

# Corte Mínimo num grafo não orientado e pesado — Transformação e Conquista

# Problema

Pretende-se **remover** um conjunto de **arcos** do grafo de forma a deixar de haver caminho entre os vértices **0** e **3**.

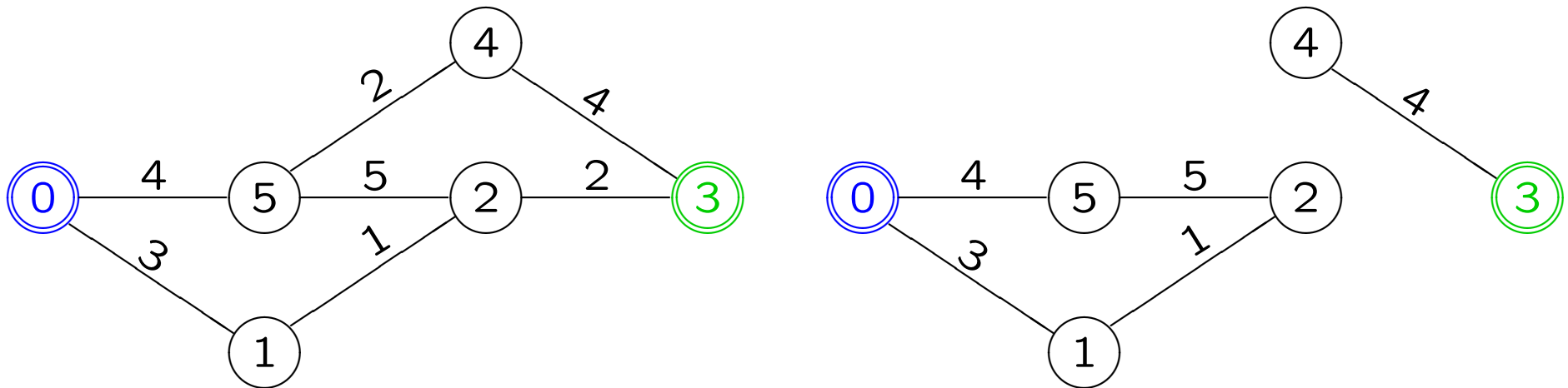
Que conjunto desses **minimiza** a soma dos pesos dos arcos removidos?



# Problema

Pretende-se **remover** um conjunto de **arcos** do grafo de forma a deixar de haver caminho entre os vértices **0** e **3**.

Que conjunto desses **minimiza** a soma dos pesos dos arcos removidos?



Conjunto  $C = \{(5, 4), (2, 3)\}$

Soma dos pesos dos arcos de  $C$ : 4

# Corte

Sejam  $G = (V, A)$  um grafo **não orientado** e **pesado** (com pesos não negativos) e  $f, d \in V$  dois vértices distintos. Assume-se que qualquer vértice pertence a um caminho entre  $f$  e  $d$ .

Um **corte**  $f-d$  é uma partição  $\{F, D\}$  de  $V$  tal que  $f \in F$  e  $d \in D$ .

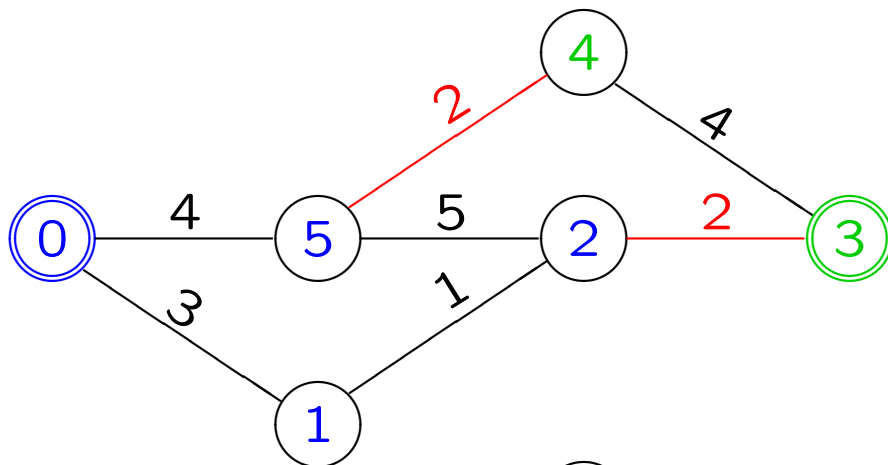
O **conjunto de arcos do corte**  $\{F, D\}$  é o conjunto dos arcos com uma extremidade em  $F$  e outra em  $D$ :

$$C_{F,D} = \{(v, w) \in A \mid v \in F, w \in D\}.$$

A **capacidade do corte**  $\{F, D\}$  é a soma dos pesos dos arcos do corte:

$$\text{capacidade}(C_{F,D}) = \sum_{(v,w) \in C_{F,D}} \text{peso}(v, w).$$

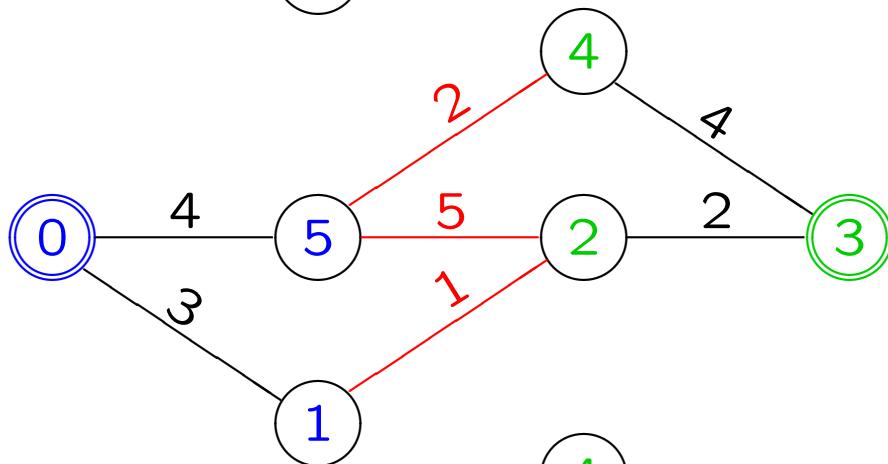
Um **corte mínimo** é um corte cuja capacidade é a menor possível.



$$F = \{0, 1, 2, 5\}, \quad D = \{3, 4\}$$

$$C_{F,D} = \{(5, 4), (2, 3)\}$$

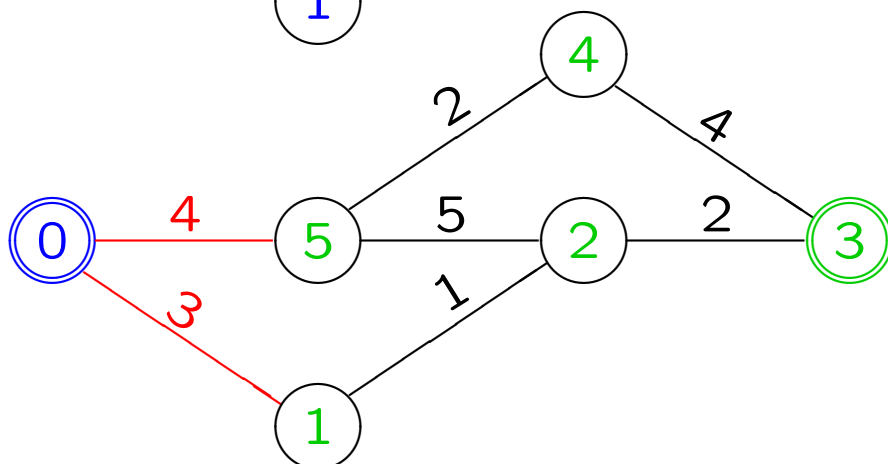
$$\text{capacidade}(C_{F,D}) = 4$$



$$F = \{0, 1, 5\}, \quad D = \{2, 3, 4\}$$

$$C_{F,D} = \{(5, 4), (5, 2), (1, 2)\}$$

$$\text{capacidade}(C_{F,D}) = 8$$



$$F = \{0\}, \quad D = \{1, 2, 3, 4, 5\}$$

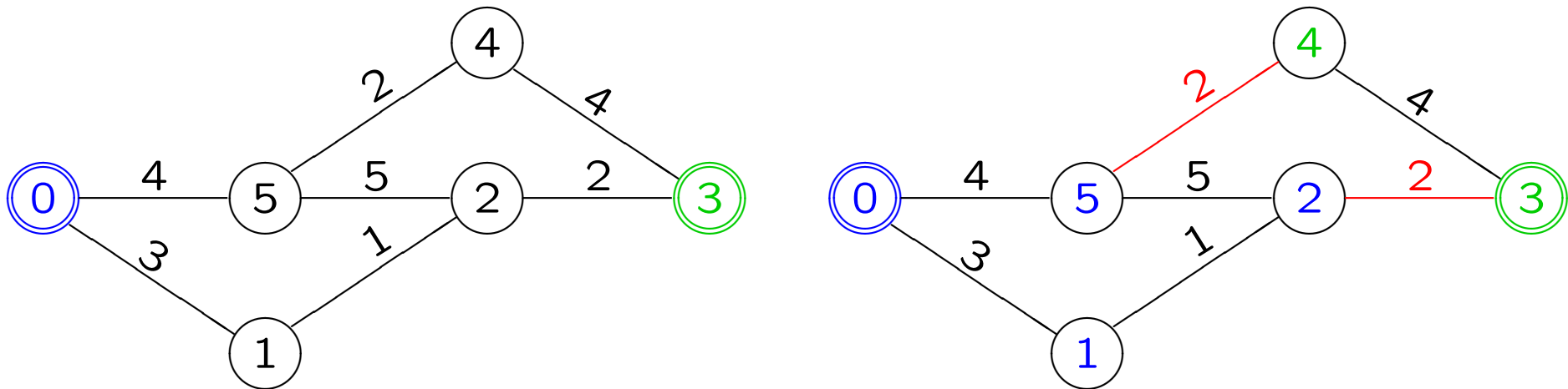
$$C_{F,D} = \{(0, 5), (0, 1)\}$$

$$\text{capacidade}(C_{F,D}) = 7$$

# Problema

Pretende-se **remover** um conjunto de **arcos** do grafo de forma a deixar de haver caminho entre os vértices **0** e **3**.

Que conjunto desses **minimiza** a soma dos pesos dos arcos removidos?



Como encontrar (os arcos de) um **corte 0–3 mínimo** no grafo **não orientado e pesado**?



# Teorema do fluxo-máximo corte-mínimo (*max-flow min-cut theorem*)

Sejam:

- $G = (V, A)$  um grafo não orientado e pesado  
(com pesos não negativos);
- $f, d \in V$  dois vértices distintos.

Assume-se que qualquer vértice pertence a um caminho entre  $f$  e  $d$ .

O valor de um fluxo máximo de  $f$  para  $d$  em  $G$   
**é igual**  
à capacidade dos cortes  $f-d$  mínimos em  $G$ .

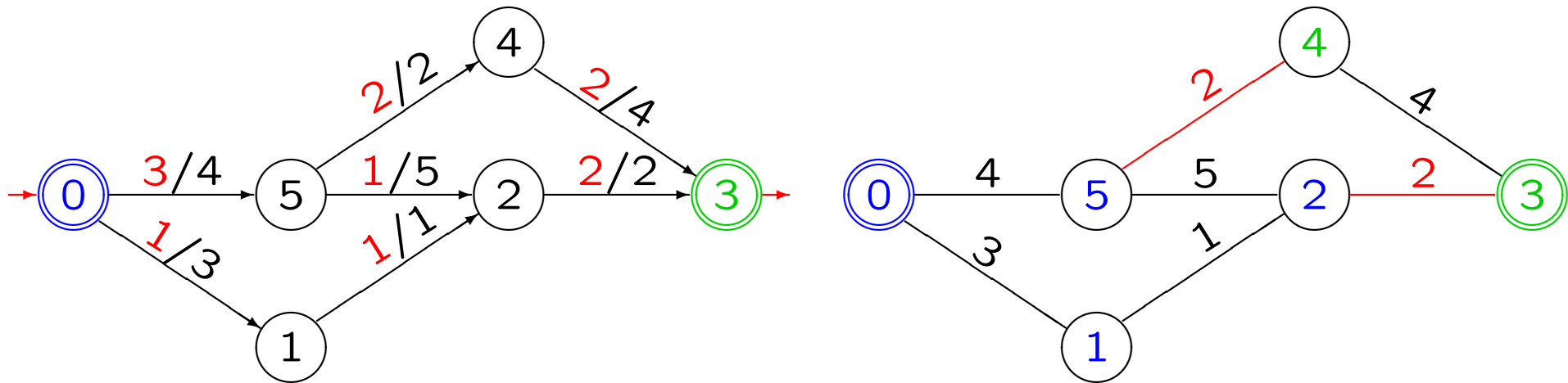
# Construção de um corte-mínimo $f - d$ (a partir de um fluxo-máximo de $f$ para $d$ )

1. Calcula-se um fluxo máximo  $\phi$  de  $f$  para  $d$  no grafo orientado correspondente.

Para cada arco (não orientado) do grafo original,  
há dois arcos com sentidos opostos no grafo orientado,  
ambos com o peso do arco original.

2. Calcula-se o conjunto ( $F$ ) dos vértices para os quais há caminho a partir de  $f$  na rede residual induzida por  $\phi$ .
3. A partição  $\{F, V \setminus F\}$  é um corte  $f - d$  mínimo.

# Fluxo-Máximo / Corte-Mínimo



1. Calcula-se um fluxo máximo  $\phi$  de 0 para 3 no grafo orientado correspondente (figura da esquerda, omitindo os arcos com fluxo  $< 0$ ).
2. Calcula-se o conjunto ( $F$ ) dos vértices para os quais há caminho a partir de 0 na rede residual induzida por  $\phi$ :  $F = \{0, 1, 2, 5\}$
3. A partição  $\{F, V \setminus F\}$  é um corte 0–3 mínimo:

$$F = \{0, 1, 2, 5\}$$

$$D = \{0, 1, 2, 3, 4, 5\} \setminus F = \{3, 4\}$$