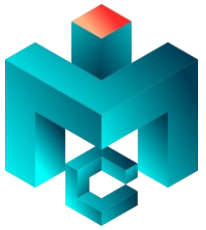
UNIVERSIDADE FEDERAL DE ITAJUBÁ
Instituto de Matemática e Computação



SMAC03 – Grafos

6. Fluxo em Redes

Rafael Frinhani
frinhani@unifei.edu.br

1º Semestre de 2025

OBJETIVOS

Apresentar os conceitos e algoritmos para o problema de fluxo máximo em redes e suas aplicações.

AGENDA

6. Fluxo em Redes

Introdução, Descrição, Definição, Aplicações, Problemas do Algoritmo Guloso para o Fluxo Máximo.

6.1. Algoritmo de Ford-Fulkerson

Contexto Histórico, Capacidade Residual da Rede, Princípio de funcionamento, Algoritmo, Exemplos, Casos Especiais.

6.2. Emparelhamento em grafos Bipartidos

Redes de Fluxo com múltiplas origens e sorvedouros, emparelhamento máximo em grafos bipartidos.

Introdução

Uma rede é a representação da estrutura de conexões entre componentes de um sistema. Em certas situações o relacionamento entre os componentes caracteriza-se pelo movimento de materiais (ex. veículos, água, eletricidade, dados etc).

fluxo = escoamento ou movimento contínuo de algo que segue em curso.

Os problemas de Fluxo em Redes tratam do movimento de materiais de um ponto de origem (oferta) até um ponto de destino (demanda) em um dado sistema.

Introdução

Uma rede é a representação da estrutura de conexões entre componentes de um sistema. Em certas situações o relacionamento entre os componentes caracteriza-se pelo movimento de materiais (ex. veículos, água, eletricidade, dados etc).

fluxo = escoamento ou movimento contínuo de algo que segue em curso.

Os problemas de Fluxo em Redes tratam do movimento de materiais de um ponto de origem (oferta) até um ponto de destino (demanda) em um dado sistema.

O grafo que representa a rede normalmente é direcionado e ponderado, mas o modelo pode considerar grafos simples e não-ponderados. Dois vértices especiais:

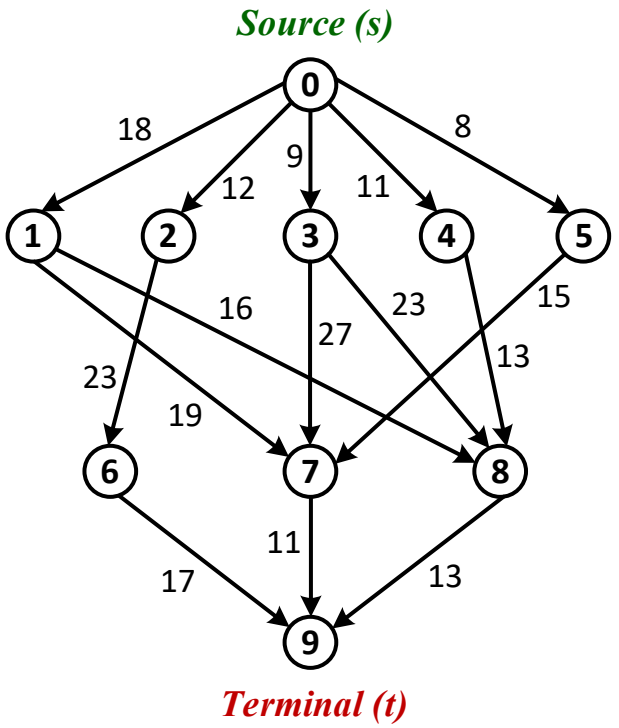
Fonte: ou origem (*source*) é o vértice sem arestas de entrada (apenas de saída) a partir do qual o fluxo tem início.

Destino: terminal ou sumidouro (*sink*) é o vértice sem arestas de saída (apenas de entrada) para onde o fluxo se destina.

Descrição

Nos problemas de Fluxo Máximo o objetivo é enviar da fonte até o sumidouro o maior fluxo possível sem violar restrições de capacidade.

Um grafo neste tipo de problema é denominado rede de fluxo (*flow network*), tendo como características:

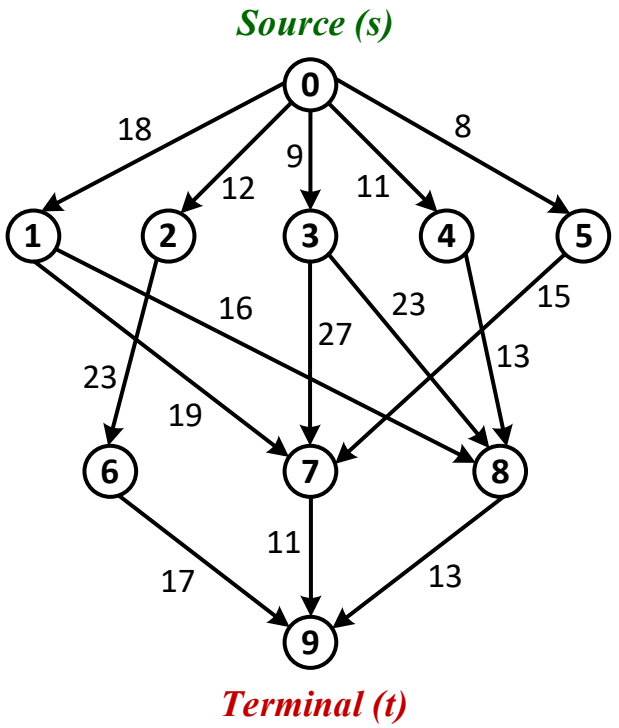


Descrição

Nos problemas de Fluxo Máximo o objetivo é enviar da fonte até o sumidouro o maior fluxo possível sem violar restrições de capacidade.

Um grafo neste tipo de problema é denominado rede de fluxo (*flow network*), tendo como características:

- Arcos representam canais por onde o material pode ser movimentado (ex. tubulação, estrada). Cada canal possui uma capacidade pré-estabelecida (peso do arco) dada como uma taxa máxima na qual o material pode fluir por ele.



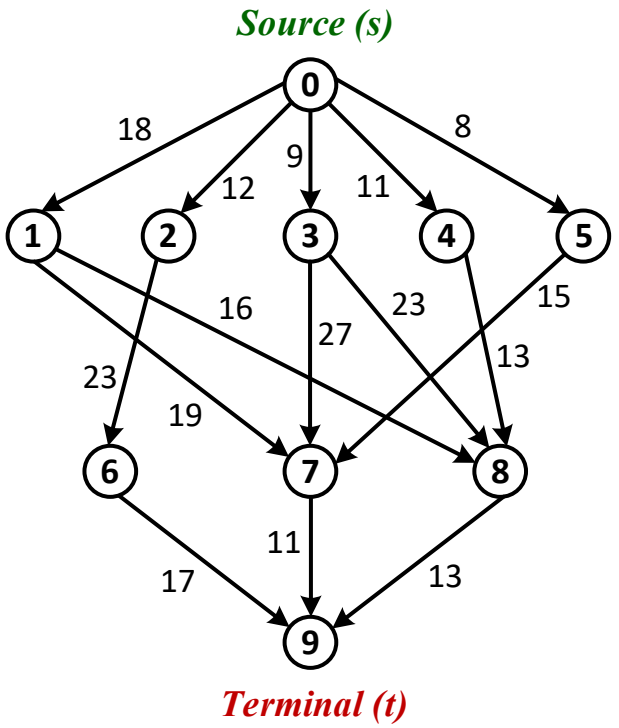
Descrição

Nos problemas de Fluxo Máximo o objetivo é enviar da fonte até o sumidouro o maior fluxo possível sem violar restrições de capacidade.

Um grafo neste tipo de problema é denominado rede de fluxo (*flow network*), tendo como características:

- Arcos representam canais por onde o material pode ser movimentado (ex. tubulação, estrada). Cada canal possui uma capacidade pré-estabelecida (peso do arco) dada como uma taxa máxima na qual o material pode fluir por ele.

- Vértices representam junções de canais (ex. válvula, cruzamento). Vértices intermediários ao fonte e sumidouro não possuem capacidade de armazenamento (propriedade da conservação do fluxo → fluxo que entra = fluxo que sai).



Definição

Um fluxo em rede $G = (V, E)$ é um grafo orientado em que cada arco $(v, u) \in E$ tem capacidade positiva $c(i, j) \geq 0$. Caso não exista uma aresta entre v e u $c(i, j) = 0$.

Distingue-se dois vértices em um fluxo em rede: um vértice origem s considerado o emissor do fluxo, e um vértice terminal t considerado um consumidor do fluxo.

Definição

Um fluxo em rede $G = (V, E)$ é um grafo orientado em que cada arco $(v, u) \in E$ tem capacidade positiva $c(i, j) \geq 0$. Caso não exista uma aresta entre v e u $c(i, j) = 0$.

Distingue-se dois vértices em um fluxo em rede: um vértice origem s considerado o emissor do fluxo, e um vértice terminal t considerado um consumidor do fluxo.

Um fluxo em G é uma função de valor $f : V \times V \rightarrow R$ com as três **Propriedades**:

- **Restrição de Capacidade:** o fluxo de um vértice até outro não deve exceder a capacidade dada.

$$\forall\ v, u \in V, \text{ exige-se que } f(v, u) \leq c(v, u)$$

- **Anti-simetria oblíqua:** o fluxo de um vértice v até um vértice u é o valor negativo do fluxo no sentido inverso.

$$\forall\ v, u \in V, \text{ exige-se que } f(v, u) = -f(u, v)$$

Definição

Um fluxo em rede $G = (V, E)$ é um grafo orientado em que cada arco $(v, u) \in E$ tem capacidade positiva $c(i, j) \geq 0$. Caso não exista uma aresta entre v e u $c(i, j) = 0$.

Distingue-se dois vértices em um fluxo em rede: um vértice origem s considerado o emissor do fluxo, e um vértice terminal t considerado um consumidor do fluxo.

Um fluxo em G é uma função de valor $f : V \text{ x } V \longrightarrow R$ com as três **Propriedades**:

- **Conservação de fluxo:** "fluxo que entra é igual o fluxo que sai". O fluxo total para fora de um vértice que não seja origem ou terminal é 0.

$$\forall\ v, u \in V, \text{ exige-se que } \sum_{v \in V} f(v, u) = 0$$

O valor de um fluxo f é definido como

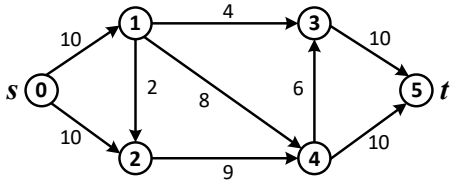
$$|f| = \sum_{v \in V} f(s, v)$$

Aplicações

- Em projetos de redes hidráulicas (água, esgoto), de produtos químicos, de combustível (petróleo, gás)
- Dimensionamento de redes de transmissão de energia elétrica (baixa, média e alta tensão)
- Redes de computadores, de comunicação (ex. telefonia celular, rádio)
- Sistemas viários (estradas, rodovias), ferrovias, rotas aéreas, hidrovias
- Problemas de manufatura em indústrias, planejamento de produção
- Logística, cadeia de suprimentos, transporte de produtos.
- Estudos de dinâmica dos fluidos, escoamento, drenagem.

Algoritmo Guloso

Diferente de outros problemas modelados em grafos (ex. árvore geradora) os algoritmos gulosos não são adequados para fluxo máximo.

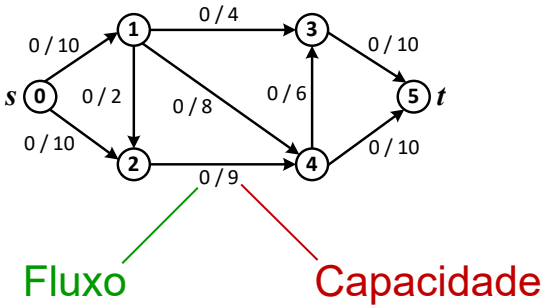


Estratégia Gulosa:

- Iniciar cada aresta com fluxo 0.
- Buscar um caminho $s \rightarrow t$ onde o fluxo das arestas é menor ou igual a sua capacidade.
- Aumentar o fluxo do caminho.
- Repetir enquanto possível.

Algoritmo Guloso

Diferente de outros problemas modelados em grafos (ex. árvore geradora) os algoritmos gulosos não são adequados para fluxo máximo.

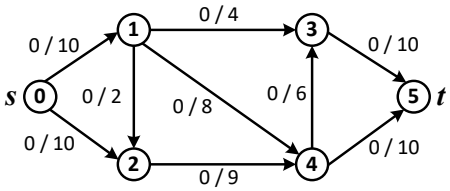


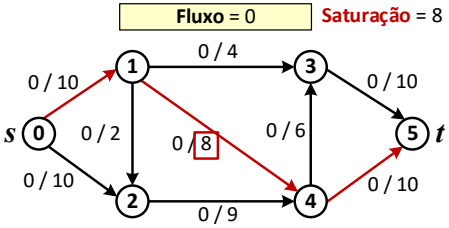
Estratégia Gulosa:

- Iniciar cada aresta com fluxo 0.
- Buscar um caminho $s \rightarrow t$ onde o fluxo das arestas é menor ou igual a sua capacidade.
- Aumentar o fluxo do caminho.
- Repetir enquanto possível.

Algoritmo Guloso

Diferente de outros problemas modelados em grafos (ex. árvore geradora) os algoritmos gulosos não são adequados para fluxo máximo.



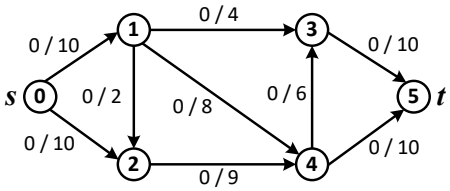


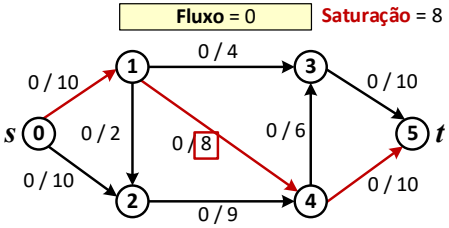
Estratégia Gulosa:

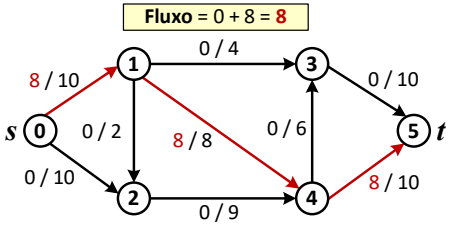
- Iniciar cada aresta com fluxo 0.
- Buscar um caminho $s \rightarrow t$ onde o fluxo das arestas é menor ou igual a sua capacidade.
- Aumentar o fluxo do caminho.
- Repetir enquanto possível.

Algoritmo Guloso

Diferente de outros problemas modelados em grafos (ex. árvore geradora) os algoritmos gulosos não são adequados para fluxo máximo.





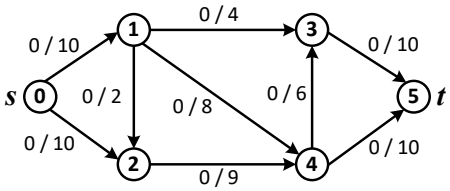


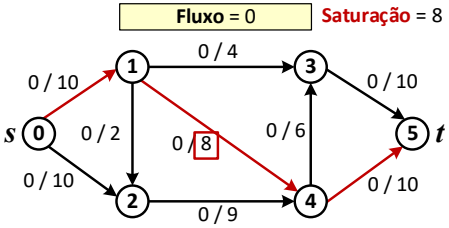
Estratégia Gulosa:

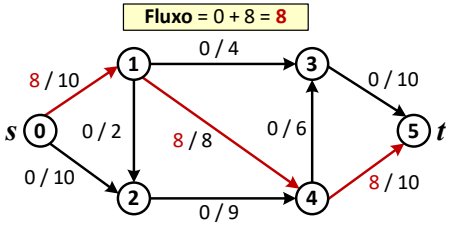
- Iniciar cada aresta com fluxo 0.
- Buscar um caminho $s \rightarrow t$ onde o fluxo das arestas é menor ou igual a sua capacidade.
- Aumentar o fluxo do caminho.
- Repetir enquanto possível.

Algoritmo Guloso

Diferente de outros problemas modelados em grafos (ex. árvore geradora) os algoritmos gulosos não são adequados para fluxo máximo.





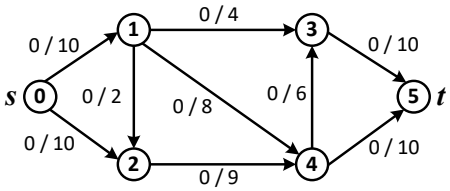


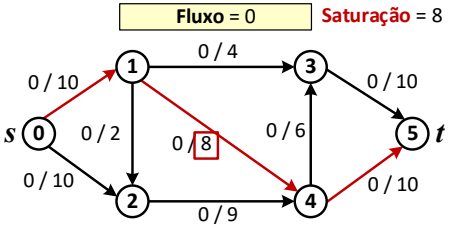
Estratégia Gulosa:

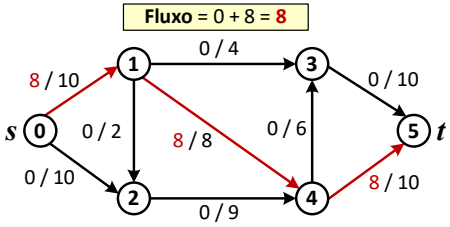
- Iniciar cada aresta com fluxo 0.
- Buscar um caminho $s \rightarrow t$ onde o fluxo das arestas é menor ou igual a sua capacidade.
- Aumentar o fluxo do caminho.
- Repetir enquanto possível.

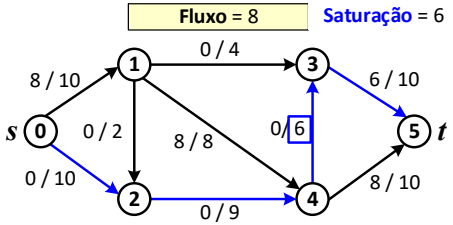
Algoritmo Guloso

Diferente de outros problemas modelados em grafos (ex. árvore geradora) os algoritmos gulosos não são adequados para fluxo máximo.









Estratégia Gulosa:

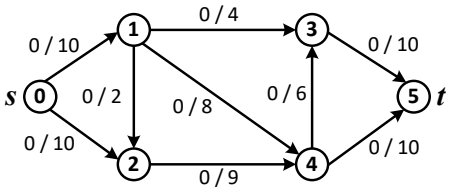
- Iniciar cada aresta com fluxo 0.
- Buscar um caminho $s \rightarrow t$ onde o fluxo das arestas é menor ou igual a sua capacidade.
- Aumentar o fluxo do caminho.
- Repetir enquanto possível.

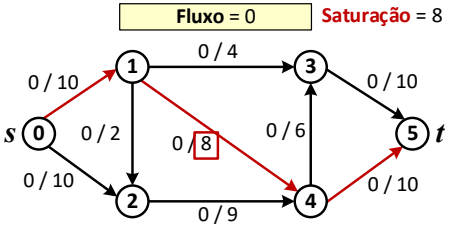
Algoritmo Guloso

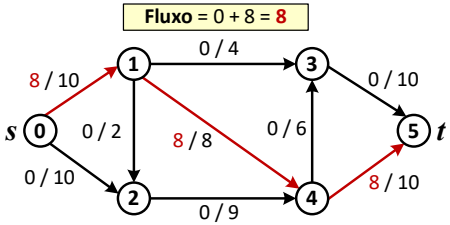
Diferente de outros problemas modelados em grafos (ex. árvore geradora) os algoritmos gulosos não são adequados para fluxo máximo.

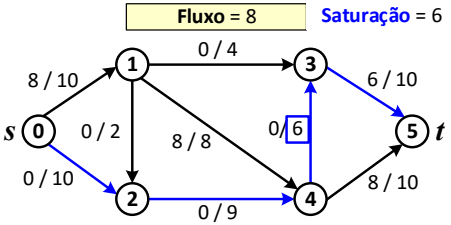
Estratégia Gulosa:

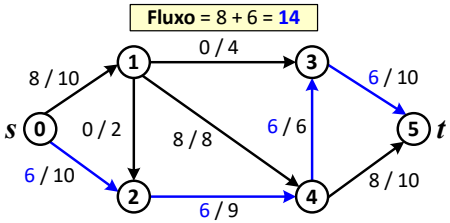
- Iniciar cada aresta com fluxo 0.
- Buscar um caminho $s \rightarrow t$ onde o fluxo das arestas é menor ou igual a sua capacidade.
- Aumentar o fluxo do caminho.
- Repetir enquanto possível.









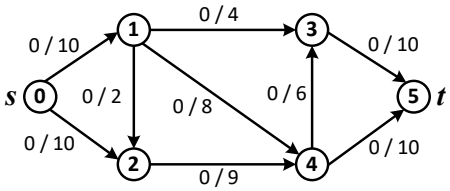


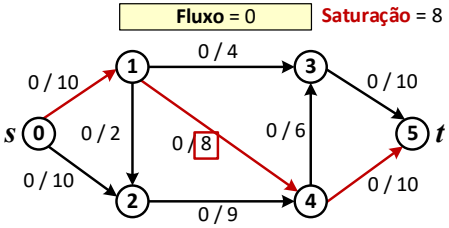
Algoritmo Guloso

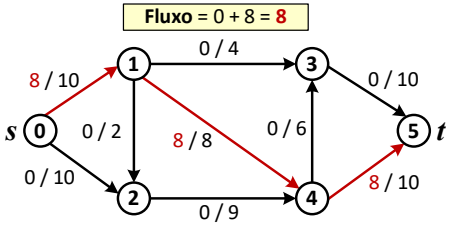
Diferente de outros problemas modelados em grafos (ex. árvore geradora) os algoritmos gulosos não são adequados para fluxo máximo.

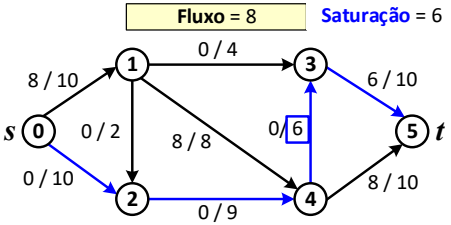
Estratégia Gulosa:

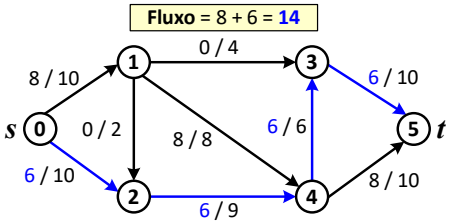
- Iniciar cada aresta com fluxo 0.
- Buscar um caminho $s \rightarrow t$ onde o fluxo das arestas é menor ou igual a sua capacidade.
- Aumentar o fluxo do caminho.
- Repetir enquanto possível.









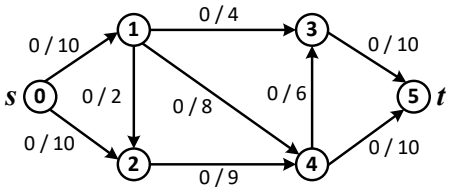


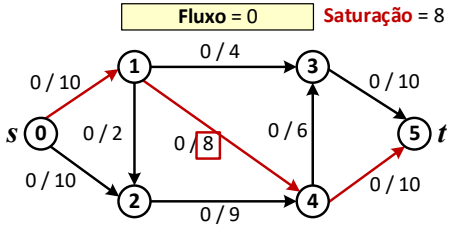
Algoritmo Guloso

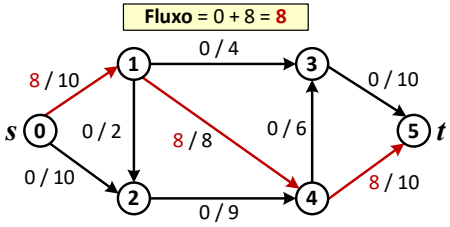
Diferente de outros problemas modelados em grafos (ex. árvore geradora) os algoritmos gulosos não são adequados para fluxo máximo.

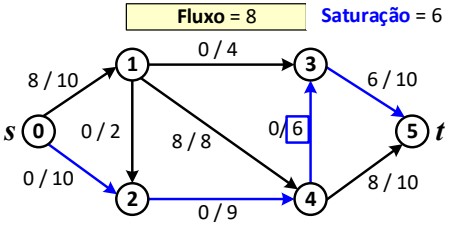
Estratégia Gulosa:

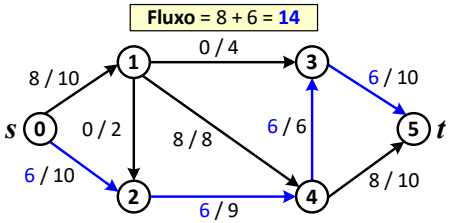
- Iniciar cada aresta com fluxo 0.
- Buscar um caminho $s \rightarrow t$ onde o fluxo das arestas é menor ou igual a sua capacidade.
- Aumentar o fluxo do caminho.
- Repetir enquanto possível.

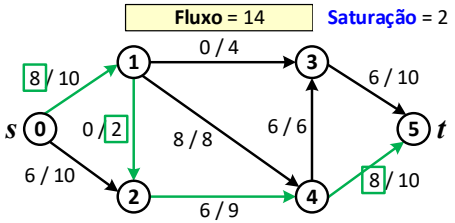


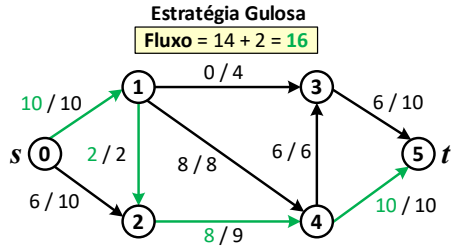












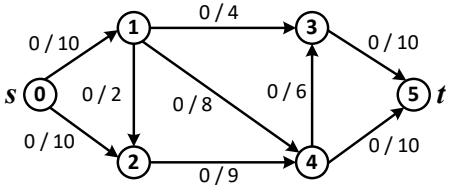
Algoritmo Guloso

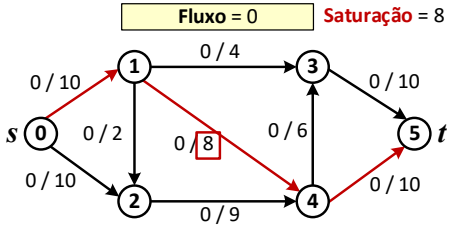
Diferente de outros problemas modelados em grafos (ex. árvore geradora) os algoritmos gulosos não são adequados para fluxo máximo.

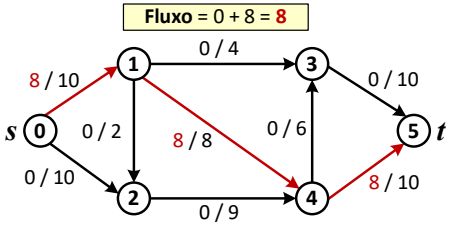
Estratégia Gulosa:

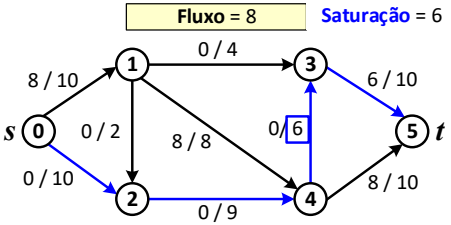
- Iniciar cada aresta com fluxo 0.
- Buscar um caminho $s \rightarrow t$ onde o fluxo das arestas é menor ou igual a sua capacidade.
- Aumentar o fluxo do caminho.
- Repetir enquanto possível.

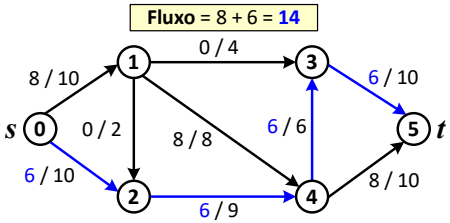
O fluxo máximo da rede é 19 e não 16 que foi obtido pelo algoritmo guloso.

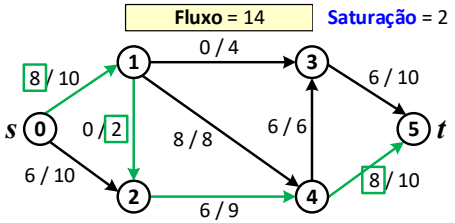


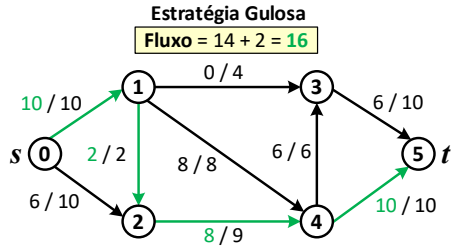


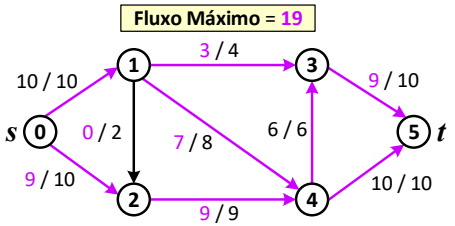






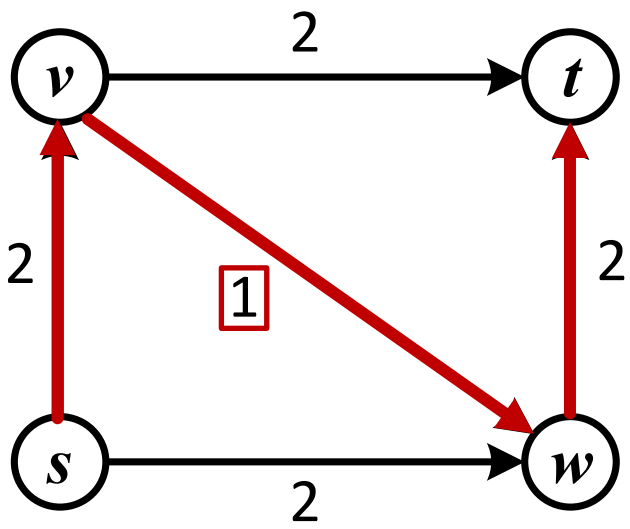






Algoritmo Guloso

O algoritmo guloso falha pois, uma vez que ele escolhe o caminho e aumenta o fluxo de seus arcos, ele nunca diminui seu fluxo novamente para possíveis ajustes.



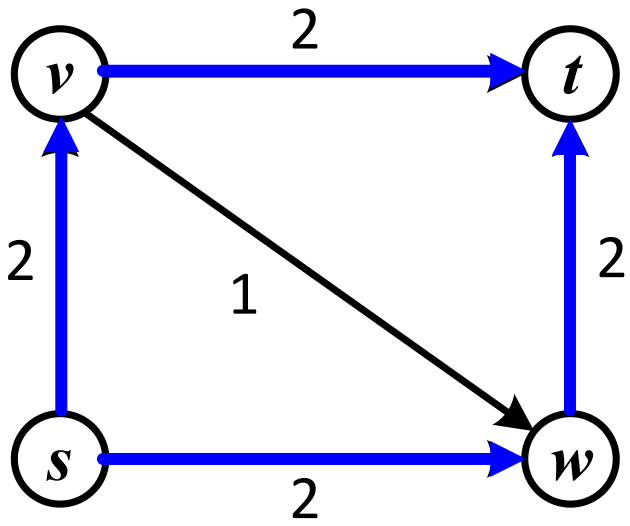
Fluxo Máximo = 3

Exemplo:

- O algoritmo guloso pode optar primeiro pelo caminho $s - v - w - t$, mas esta é uma decisão ruim pois o canal (v, w) é um gargalo da rede.
- Para uma melhor solução, é necessário algum mecanismo para "desfazer" decisões ruins.

Algoritmo Guloso

O algoritmo guloso falha pois, uma vez que ele escolhe o caminho e aumenta o fluxo de seus arcos, ele nunca diminui seu fluxo novamente para possíveis ajustes.



Fluxo Máximo = **4**

Exemplo:

- O algoritmo guloso pode optar primeiro pelo caminho $s - v - w - t$, mas esta é uma decisão ruim pois o canal (v, w) é um gargalo da rede.
- Para uma melhor solução, é necessário algum mecanismo para "desfazer" decisões ruins.
- Os caminhos $s - v - t$ e $s - w - t$ correspondem ao fluxo máximo.
- No fluxo máximo de $s - t$ da rede ao lado, o fluxo $f(v, w) = 0$, pois ele não é armazenado.

Algoritmos

A história dos algoritmos de fluxo está relacionada à análise da rede ferroviária da União Soviética entre as décadas de 1930 e 1950. Schrijver, Alexander, "*On the history of the transportation and maximum flow problems". Mathematical Programming*. **91**: 437-445, 2002.

Ford-Fulkerson: Definiu um algoritmo para solução do fluxo máximo. Não especifica o procedimento de busca.

Algoritmos

A história dos algoritmos de fluxo está relacionada à análise da rede ferroviária da União Soviética entre as décadas de 1930 e 1950. Schrijver, Alexander, "*On the history of the transportation and maximum flow problems*". *Mathematical Programming*. **91**: 437-445, 2002.

Ford-Fulkerson: Definiu um algoritmo para solução do fluxo máximo. Não especifica o procedimento de busca.

Edmonds-Karp: Baseado no algoritmo de Ford-Fulkerson, emprega BFS como método para encontrar os caminhos de aumento de fluxo. Possui complexidade total igual a $O(VE^2)$.

Dinic: Também baseado no Ford-Fulkerson, usa uma combinação de DFS e BFS para encontrar os caminhos de aumento. Complexidade total é $O(V^2E)$.

As complexidades de tempo dos algoritmos acima são pessimistas. Na prática, os algoritmos tendem a executar mais rapidamente, tornando difícil compará-los apenas por sua complexidade assintótica.

Algoritmo de Ford-Fulkerson

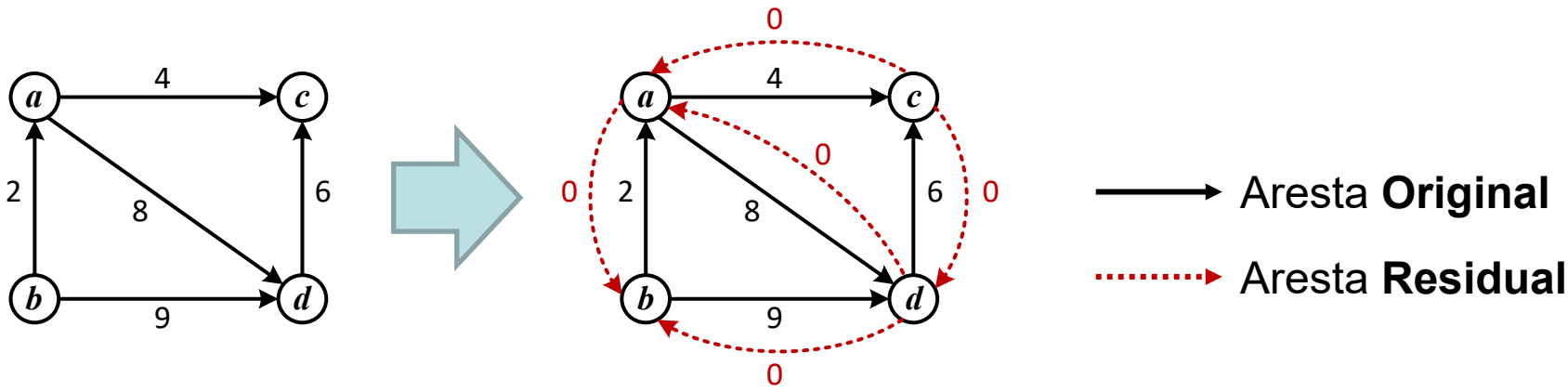
Proposto originalmente por Lester Randolph Jr e Delbert Ray Fulkerson em 1962 para solução do problema de determinação do fluxo máximo.

Princípio de Funcionamento

- O método considera inicialmente um fluxo vazio na rede, ou seja, o fluxo de cada um dos canais é zero.
- Através de um procedimento de busca (ex. BFS, DFS, Dijkistra etc), a cada passo o algoritmo encontra um caminho da origem até o destino, de modo a aumentar o fluxo.
- Quando o algoritmo não é mais capaz de aumentar o fluxo, significa que o fluxo máximo da rede foi atingido.

Capacidade Residual da Rede

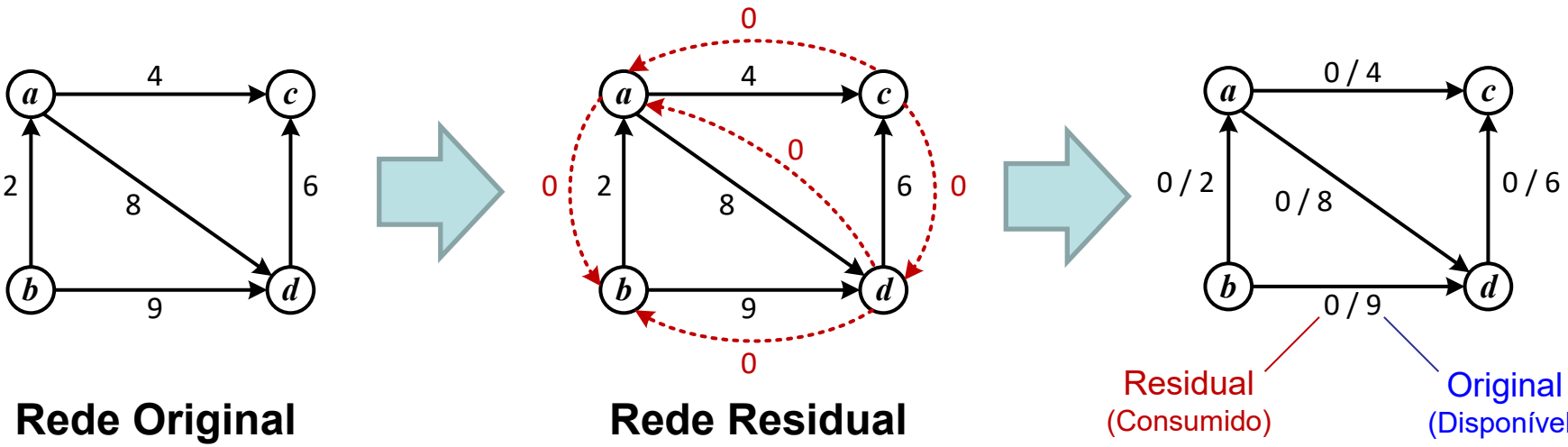
- O método que será mostrado a seguir usa uma representação especial de grafo, na qual cada aresta original da rede possui uma aresta correspondente em sentido contrário, denominada aresta residual.



Rede Original **Rede Residual**

Capacidade Residual da Rede

- O método que será mostrado a seguir usa uma representação especial de grafo, na qual cada aresta original da rede possui uma aresta correspondente em sentido contrário, denominada aresta residual.
- O peso de cada aresta (original ou residual) indica o quanto de fluxo ainda é possível trafegar por ela (capacidade do canal).
- No início do algoritmo, o peso de cada aresta original é igual a sua capacidade, e o peso de cada aresta residual é igual a zero. A soma das arestas original e residual é sempre igual a capacidade total do canal que elas representam.



Rede Original **Rede Residual**

Funcionamento

- O algoritmo executa várias iterações sobre o grafo.
- A cada iteração, ele escolhe um caminho da origem até o destino de forma que o peso das arestas do caminho sejam positivos maior que zero.
  - Se existir mais de um caminho, qualquer um deles pode ser analisado naquela iteração.
  - Qualquer aresta (original ou residual) pode fazer parte do caminho.

Funcionamento

- O algoritmo executa várias iterações sobre o grafo.
- A cada iteração, ele escolhe um caminho da origem até o destino de forma que o peso das arestas do caminho sejam positivos (maior que zero).
  - Se existir mais de um caminho, qualquer um deles pode ser analisado naquela iteração.
  - Qualquer aresta (original ou residual) pode fazer parte do caminho.
- **Após escolher o caminho:**
  - O fluxo do canal é X unidades, sendo que X é o menor peso de aresta do caminho (canal que caracteriza um gargalo).
  - O peso das arestas originais do caminho diminui em X unidades (representa o valor ainda disponível no canal).
  - O peso das arestas residuais do caminho aumenta em X unidades (valor consumido do canal).

Funcionamento

- A ideia é que, o aumento do fluxo em um caminho diminui a quantidade de fluxo que pode percorrer as respectivas arestas no futuro.

- Ao mesmo tempo, é possível "cancelar" fluxos em passos subsequentes usando as arestas residuais (reversas), se o algoritmo detectar que esta ação pode beneficiar a injeção de fluxo em outro caminho.

Funcionamento

- A ideia é que, o aumento do fluxo em um caminho diminui a quantidade de fluxo que pode percorrer as respectivas arestas no futuro.

- Ao mesmo tempo, é possível "cancelar" fluxos em passos subsequentes usando as arestas residuais (reversas), se o algoritmo detectar que esta ação pode beneficiar a injeção de fluxo em outro caminho.

  Interpretação: não se "caminhou" por uma aresta em seu sentido contrário. Apenas "deixou-se" de injetar uma certa quantia de fluxo por ela, aumentando a capacidade do canal, representado pela aresta original.

Funcionamento

- A ideia é que, o aumento do fluxo em um caminho diminui a quantidade de fluxo que pode percorrer as respectivas arestas no futuro.

- Ao mesmo tempo, é possível "cancelar" fluxos em passos subsequentes usando as arestas residuais (reversas), se o algoritmo detectar que esta ação pode beneficiar a injeção de fluxo em outro caminho.

  Interpretação: não se "caminhou" por uma aresta em seu sentido contrário. Apenas "deixou-se" de injetar uma certa quantia de fluxo por ela, aumentando a capacidade do canal, representado pela aresta original.

- O algoritmo aumenta o fluxo enquanto existirem caminhos do vértice de origem ao vértice de destino que passem apenas por arestas com pesos positivos maior que zero.

- Não havendo mais caminhos com algum fluxo disponível, o algoritmo encerra e o fluxo máximo é retornado.

Algoritmo

```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4  |   c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5  |   for cada arco (v,u) ∈ p do
6  |   |   G_r[v][u] ← G_r[v][u] − c_r(p);
7  |   |   G_r[u][v] ← G_r[v][u] + c_r(p);
8  |   end
9  |   fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

ESTRUTURAS

- ***G*, G_r** : Rede de Fluxo G e respectiva Rede Residual G_r
- **$c_r(p)$** : capacidade residual de um caminho p
- **$G_r[v][u]$**: aresta (v, u) da Rede Residual G_r

Definir a rede residual G_r como uma cópia da rede de fluxo G e inicializar a variável *fluxoMax*, que armazenará o fluxo máximo da rede residual (linhas 1 e 2).

Algoritmo

```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4  |   c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5  |   for cada arco (v,u) ∈ p do
6  |   |   G_r[v][u] ← G_r[v][u] − c_r(p);
7  |   |   G_r[u][v] ← G_r[v][u] + c_r(p);
8  |   end
9  |   fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

ESTRUTURAS

- G, G_r : Rede de Fluxo G e respectiva Rede Residual G_r
- $c_r(p)$: capacidade residual de um caminho p
- $G_r[v][u]$: aresta (v, u) da Rede Residual G_r

Definir a rede residual G_r como uma cópia da rede de fluxo G e inicializar a variável $fluxoMax$, que armazenará o fluxo máximo da rede residual (linhas 1 e 2).

Enquanto existirem caminhos entre $(s - t)$ na rede residual G_r (linhas 3 a 10):

Encontrar o valor mínimo (gargalo) da capacidade residual do caminho p (linha 4).

Para cada aresta do caminho p (linha 5) atualizar a capacidade das arestas originais (linha 6) e das arestas residuais (linha 7) de G_r.

Atualizar o valor do fluxo máximo da rede (linha 9).

Retornar o fluxo máximo da rede G_r.

Algoritmo – Exemplo 1

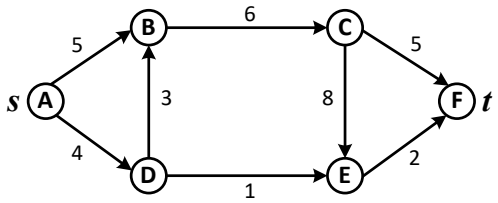
```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4  |   c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5  |   for cada arco (v,u) ∈ p do
6  |   |   G_r[v][u] ← G_r[v][u] − c_r(p);
7  |   |   G_r[u][v] ← G_r[v][u] + c_r(p);
8  |   end
9  |   fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

Uma rede de fluxo G com vértice A como origem e F como destino.

Caminho: –

Fluxo = 0

G



Algoritmo – Exemplo 1

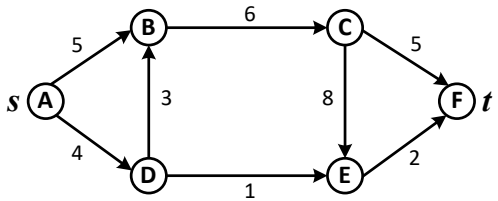
```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4  |   c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5  |   for cada arco (v,u) ∈ p do
6  |   |   G_r[v][u] ← G_r[v][u] − c_r(p);
7  |   |   G_r[u][v] ← G_r[v][u] + c_r(p);
8  |   end
9  |   fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

Uma rede de fluxo G com vértice A como origem e F como destino.

É criada uma rede residual G_r a partir da rede de fluxo G. O fluxo máximo é inicializado com valor 0 (Fluxo = 0).

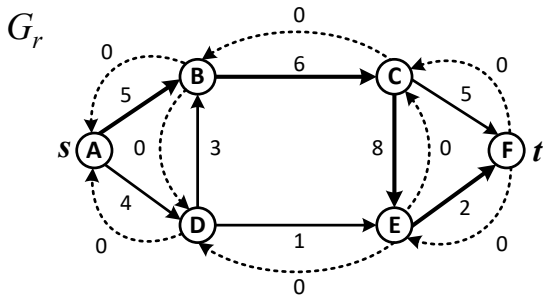
Caminho: – **Fluxo** = 0

G



Caminho: A – B – C – E – F **Fluxo** = 0

G_r



Algoritmo – Exemplo 1

```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4  |   c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5  |   for cada arco (v,u) ∈ p do
6  |   |   G_r[v][u] ← G_r[v][u] − c_r(p);
7  |   |   G_r[u][v] ← G_r[v][u] + c_r(p);
8  |   end
9  |   fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

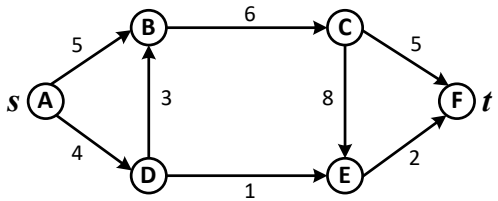
Uma rede de fluxo G com vértice A como origem e F como destino.

É criada uma rede residual G_r a partir da rede de fluxo G. O fluxo máximo é inicializado com valor 0 (Fluxo = 0).

A capacidade residual do caminho destacado na rede é $c_r(p) = 2$.

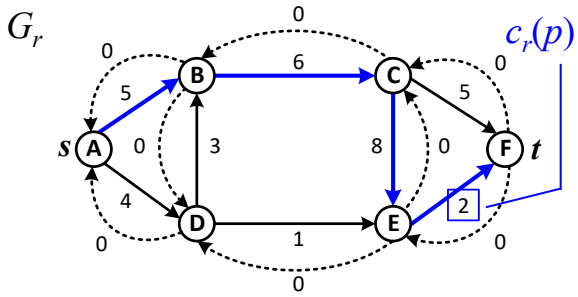
Caminho: – **Fluxo** = 0

G



Caminho: A – B – C – E – F **Fluxo** = 0

G_r



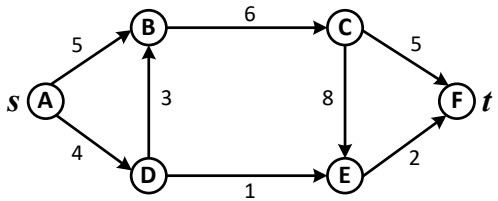
Algoritmo – Exemplo 1

```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4  |   c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5  |   for cada arco (v,u) ∈ p do
6  |   |   G_r[v][u] ← G_r[v][u] − c_r(p);
7  |   |   G_r[u][v] ← G_r[v][u] + c_r(p);
8  |   end
9  |   fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

Atualiza cada aresta original e residual do caminho p conforme a capacidade residual $c_r(p) = 2$. Também atualiza o valor do fluxo.

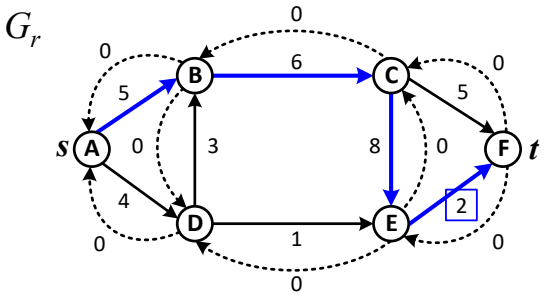
Caminho: – **Fluxo** = 0

G

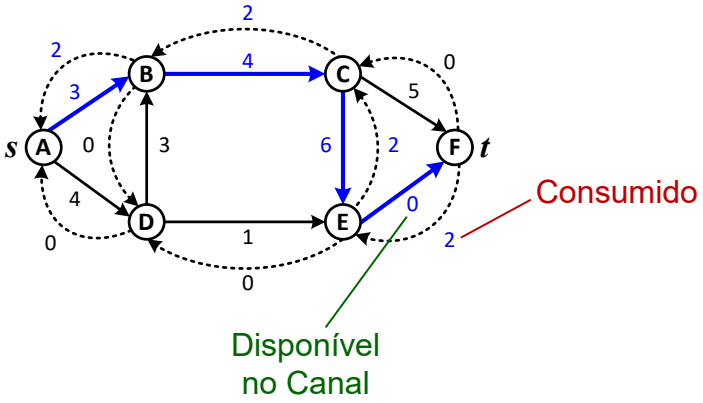


Caminho: A – B – C – E – F **Fluxo** = 0

G_r



Caminho: A – B – C – E – F **Fluxo** = 0 + 2 = **2**



Algoritmo – Exemplo 1

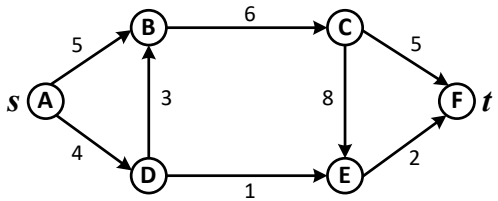
```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4  |   c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5  |   for cada arco (v,u) ∈ p do
6  |   |   G_r[v][u] ← G_r[v][u] − c_r(p);
7  |   |   G_r[u][v] ← G_r[v][u] + c_r(p);
8  |   end
9  |   fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

Atualiza cada aresta original e residual do caminho p conforme a capacidade residual $c_r(p) = 2$. Também atualiza o valor do fluxo.

Reinicia a partir de outro caminho, cujo fluxo está limitado ao canal mais restrito $c_r(p) = 3$.

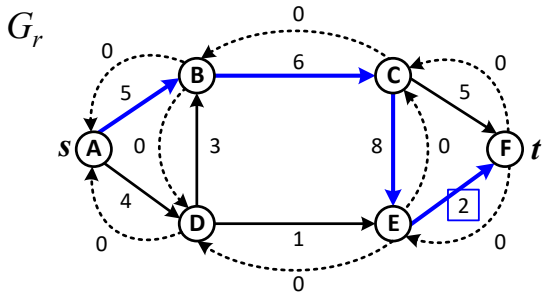
Caminho: – **Fluxo** = 0

G

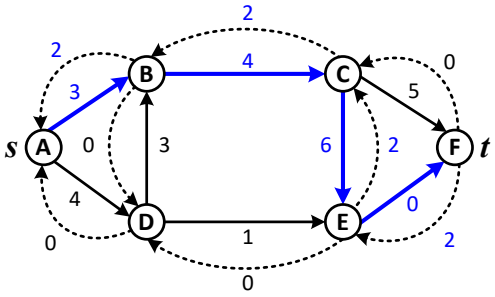


Caminho: A – B – C – E – F **Fluxo** = 0

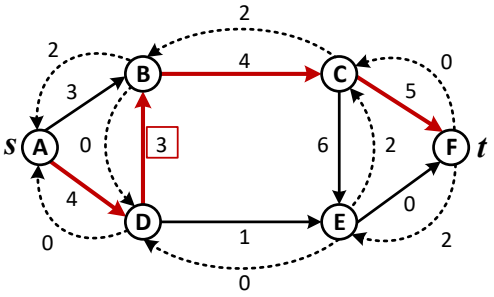
G_r



Caminho: A – B – C – E – F **Fluxo** = 0 + 2 = **2**



Caminho: A – D – B – C – F **Fluxo** = 2



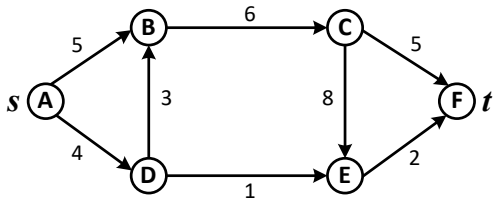
Algoritmo – Exemplo 1

```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4  |   c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5  |   for cada arco (v,u) ∈ p do
6  |   |   G_r[v][u] ← G_r[v][u] − c_r(p);
7  |   |   G_r[u][v] ← G_r[v][u] + c_r(p);
8  |   end
9  |   fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

Atualiza as arestas e o fluxo da rede considerando $c_r(p) = 3$.

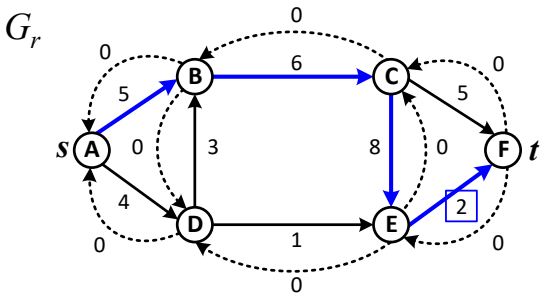
Caminho: – **Fluxo** = 0

G

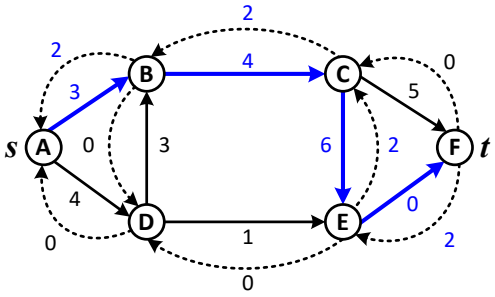


Caminho: A – B – C – E – F **Fluxo** = 0

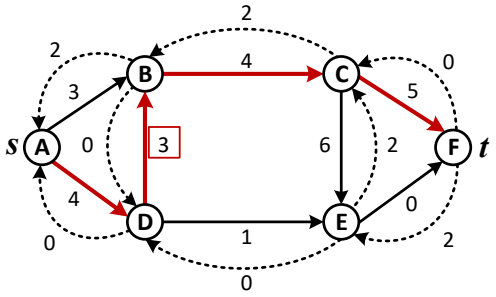
G_r



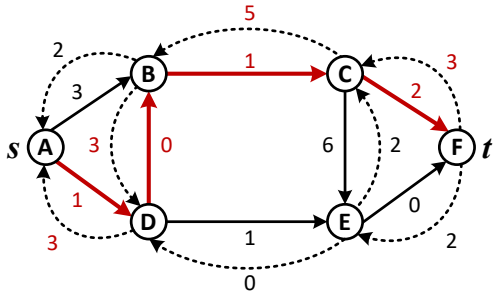
Caminho: A – B – C – E – F **Fluxo** = 0 + 2 = **2**



Caminho: A – D – B – C – F **Fluxo** = 2



Caminho: A – D – B – C – F **Fluxo** = 2 + 3 = **5**



Algoritmo – Exemplo 1

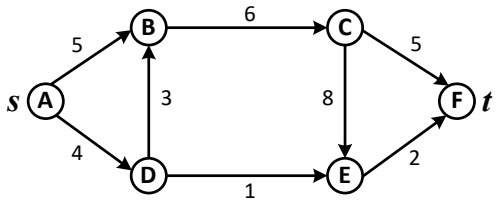
```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4  |   c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5  |   for cada arco (v,u) ∈ p do
6  |   |   G_r[v][u] ← G_r[v][u] − c_r(p);
7  |   |   G_r[u][v] ← G_r[v][u] + c_r(p);
8  |   end
9  |   fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

Atualiza as arestas e o fluxo da rede considerando $c_r(p) = 3$.

O algoritmo continua por outro caminho, o qual possui capacidade residual $c_r(p) = 1$.

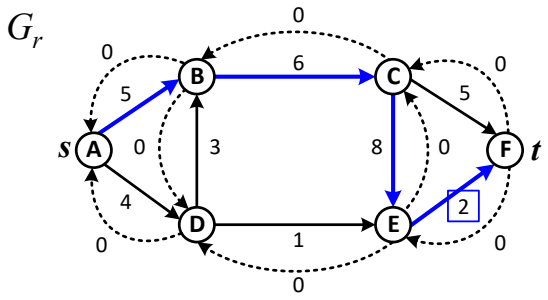
Caminho: – **Fluxo** = 0

G

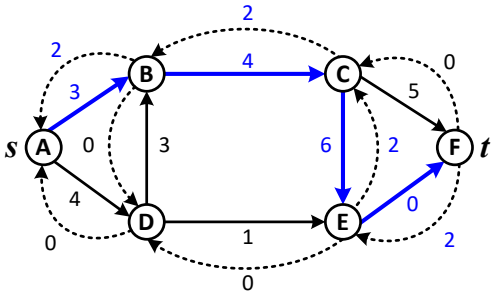


Caminho: A – B – C – E – F **Fluxo** = 0

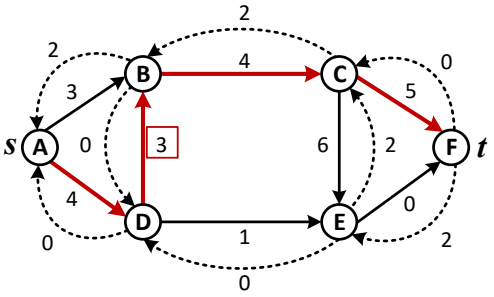
G_r



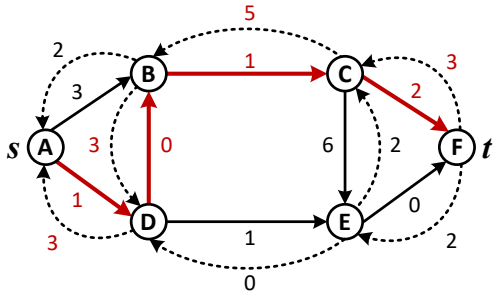
Caminho: A – B – C – E – F **Fluxo** = 0 + 2 = **2**



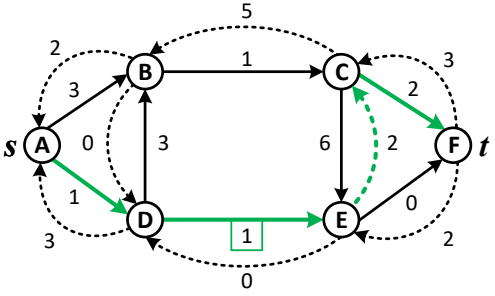
Caminho: A – D – B – C – F **Fluxo** = 2



Caminho: A – D – B – C – F **Fluxo** = 2 + 3 = **5**



Caminho: A – D – E – C – F **Fluxo** = 5



Algoritmo – Exemplo 1

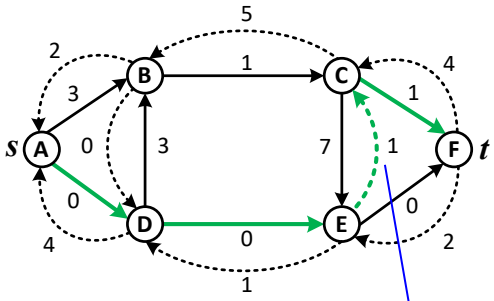
```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4  |   c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5  |   for cada arco (v,u) ∈ p do
6  |   |   G_r[v][u] ← G_r[v][u] − c_r(p);
7  |   |   G_r[u][v] ← G_r[v][u] + c_r(p);
8  |   end
9  |   fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

Atualiza as arestas e o fluxo da rede considerando $c_r(p) = 3$.

O algoritmo continua por outro caminho, o qual possui capacidade residual $c_r(p) = 1$.

Atualiza as arestas e o fluxo da rede considerando $c_r(p) = 1$.

Caminho: A – D – E – C – F **Fluxo** = 5 + 1 = **6**



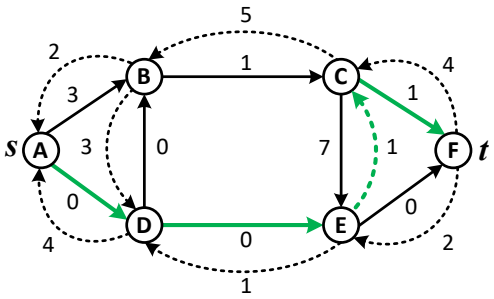
Note que a aresta (E, C) é uma aresta residual.

Isso significa desconsiderar a passagem de fluxo pela aresta (C, E) para maximizar outro caminho.

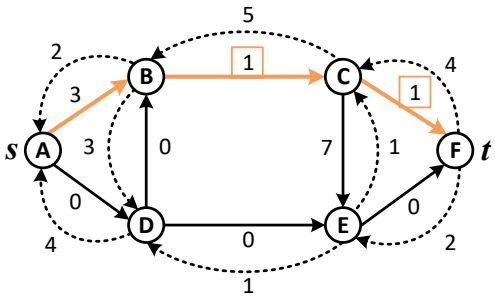
Algoritmo – Exemplo 1

```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4  |   c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5  |   for cada arco (v,u) ∈ p do
6  |   |   G_r[v][u] ← G_r[v][u] − c_r(p);
7  |   |   G_r[u][v] ← G_r[v][u] + c_r(p);
8  |   end
9  |   fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

Caminho: A – D – E – C – F **Fluxo** = 5 + 1 = **6**



Caminho: A – B – C – F **Fluxo** = 6

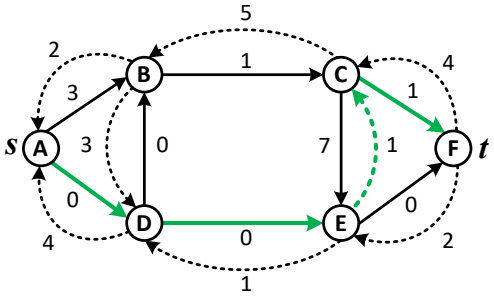


O algoritmo continua por outro caminho, o qual também possui capacidade residual $c_r(p) = 1$.

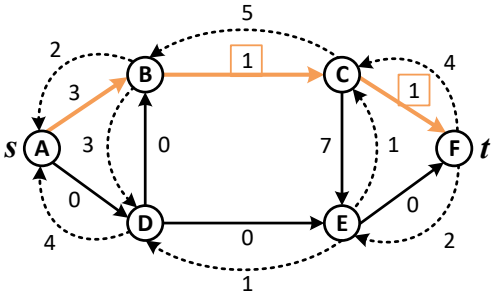
Algoritmo – Exemplo 1

```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4      c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5      for cada arco (v,u) ∈ p do
6          G_r[v][u] ← G_r[v][u] − c_r(p);
7          G_r[u][v] ← G_r[v][u] + c_r(p);
8      end
9      fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

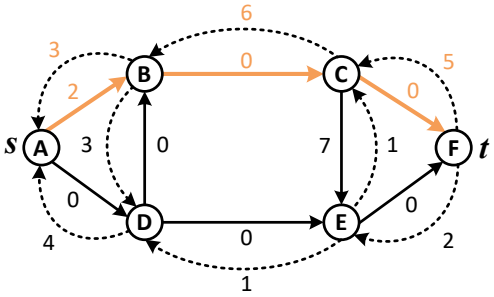
Caminho: A – D – E – C – F **Fluxo** = 5 + 1 = **6**



Caminho: A – B – C – F **Fluxo** = 6



Caminho: A – B – C – F **Fluxo** = 6 + 1 = **7**



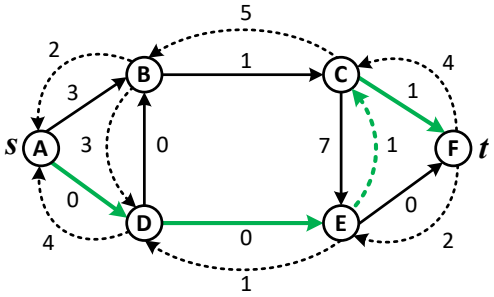
O algoritmo continua por outro caminho, o qual também possui capacidade residual $c_r(p) = 1$.

As arestas e o fluxo são atualizados conforme a capacidade residual do caminho $c_r(p) = 1$.

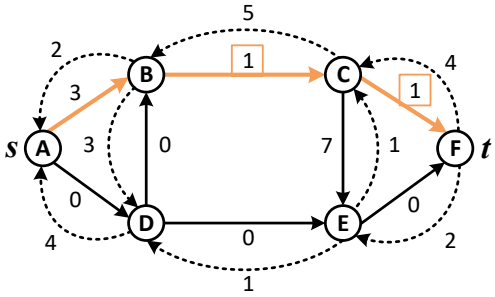
Algoritmo – Exemplo 1

```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4  |   c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5  |   for cada arco (v,u) ∈ p do
6  |   |   G_r[v][u] ← G_r[v][u] − c_r(p);
7  |   |   G_r[u][v] ← G_r[v][u] + c_r(p);
8  |   end
9  |   fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

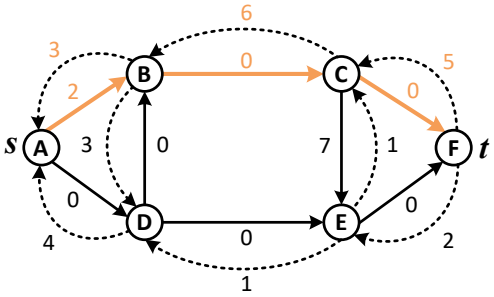
Caminho: A – D – E – C – F **Fluxo** = 5 + 1 = **6**



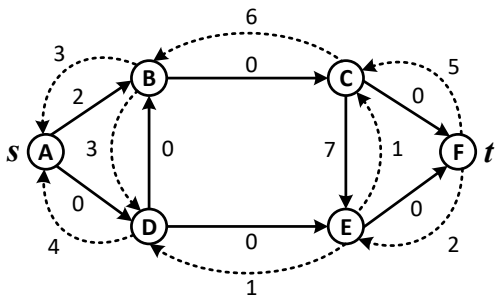
Caminho: A – B – C – F **Fluxo** = 6



Caminho: A – B – C – F **Fluxo** = 6 + 1 = **7**



Fluxo Máximo = 7



O algoritmo continua por outro caminho, o qual também possui capacidade residual $c_r(p) = 1$.

As arestas e o fluxo são atualizados conforme a capacidade residual do caminho $c_r(p) = 1$.

O método termina pois não existem mais caminhos disponíveis (ambos os fluxos de entrada de F estão saturados). O método retorna o fluxo máximo da rede igual a 7.

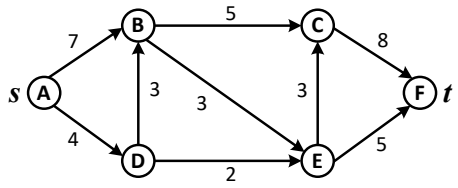
Algoritmo – Exemplo 2

```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4  |   c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5  |   for cada arco (v,u) ∈ p do
6  |   |   G_r[v][u] ← G_r[v][u] − c_r(p);
7  |   |   G_r[u][v] ← G_r[v][u] + c_r(p);
8  |   end
9  |   fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

A partir de uma rede de fluxo G é obtida uma rede residual G_r e inicializada a variável *fluxoMax*.

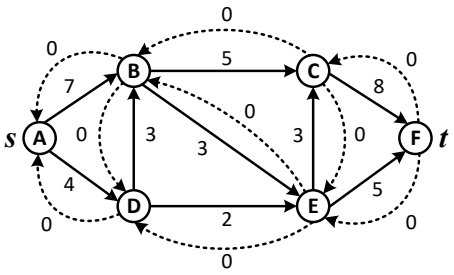
Caminho: – Fluxo = 0

G



Caminho: – Fluxo = 0

G_r

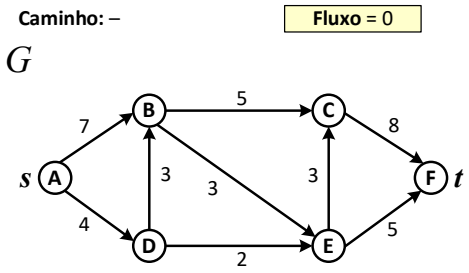


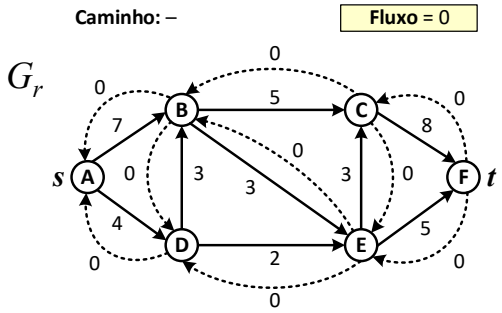
Algoritmo – Exemplo 2

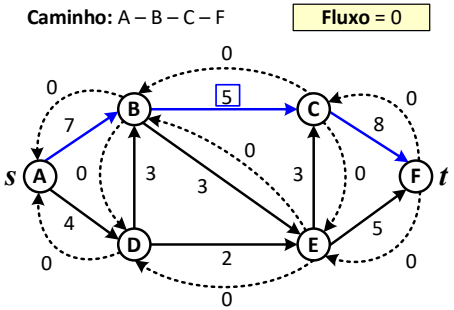
```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4  |   c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5  |   for cada arco (v,u) ∈ p do
6  |   |   G_r[v][u] ← G_r[v][u] − c_r(p);
7  |   |   G_r[u][v] ← G_r[v][u] + c_r(p);
8  |   end
9  |   fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

A partir de uma rede de fluxo G é obtida uma rede residual G_r e inicializada a variável $fluxoMax$.

O método continua identificando um caminho p (A – F), cuja capacidade residual é $c_r(p) = 5$.



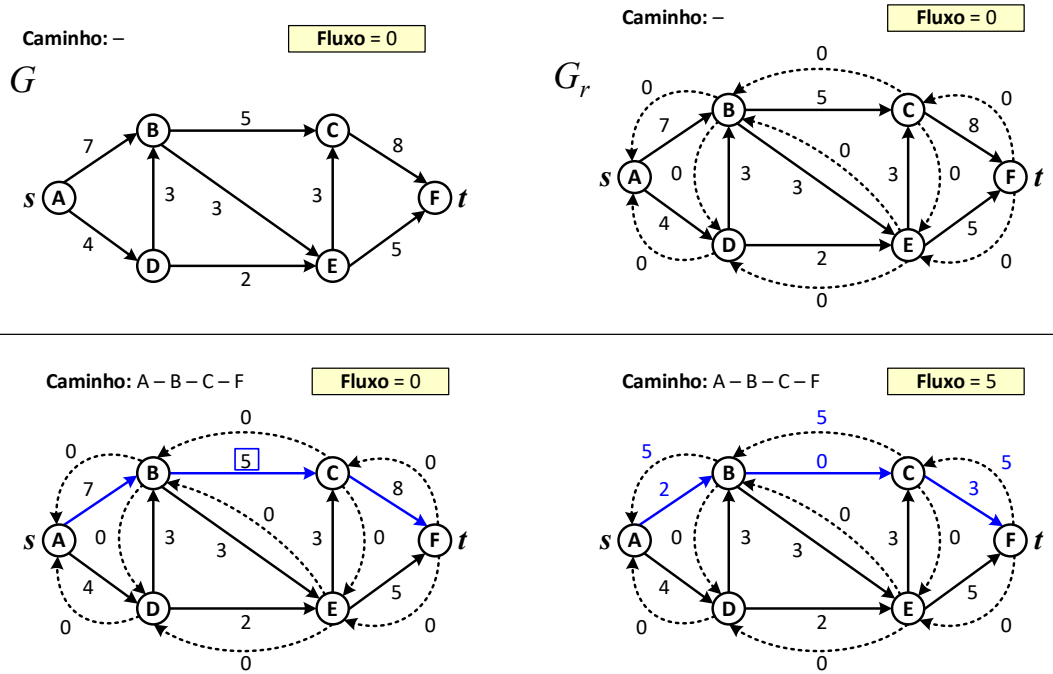




Algoritmo – Exemplo 2

```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4  |   c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5  |   for cada arco (v,u) ∈ p do
6  |   |   G_r[v][u] ← G_r[v][u] − c_r(p);
7  |   |   G_r[u][v] ← G_r[v][u] + c_r(p);
8  |   end
9  |   fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

As arestas originais e residuais do caminho p em G_r são atualizadas conforme $c_r(p) = 5$, além do fluxo.

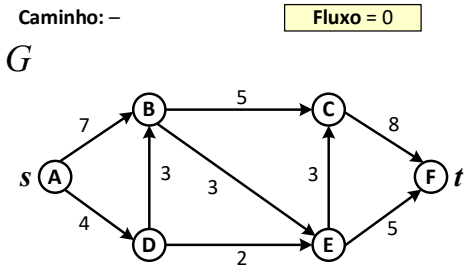


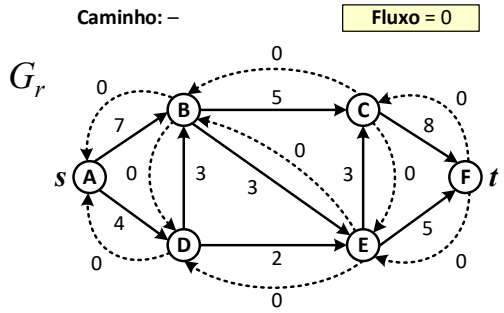
Algoritmo – Exemplo 2

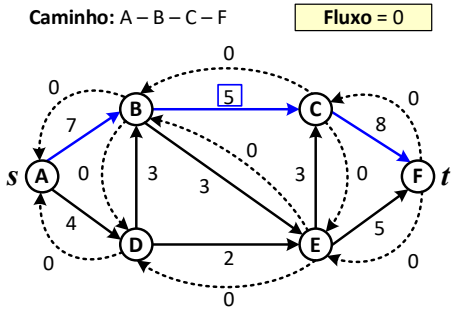
```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4  |   c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5  |   for cada arco (v,u) ∈ p do
6  |   |   G_r[v][u] ← G_r[v][u] − c_r(p);
7  |   |   G_r[u][v] ← G_r[v][u] + c_r(p);
8  |   end
9  |   fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

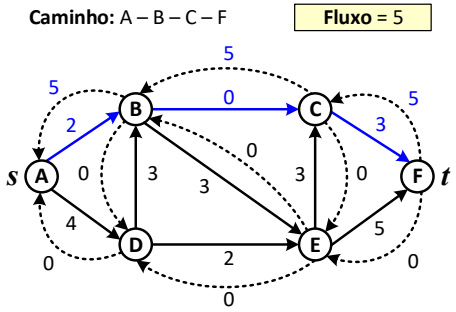
As arestas originais e residuais do caminho p em G_r são atualizadas conforme $c_r(p) = 5$, além do fluxo.

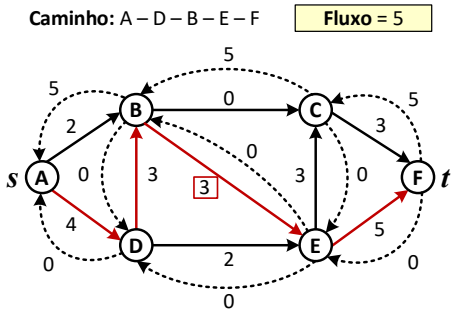
O método considera outro caminho p, cuja capacidade residual é $c_r(p) = 3$.











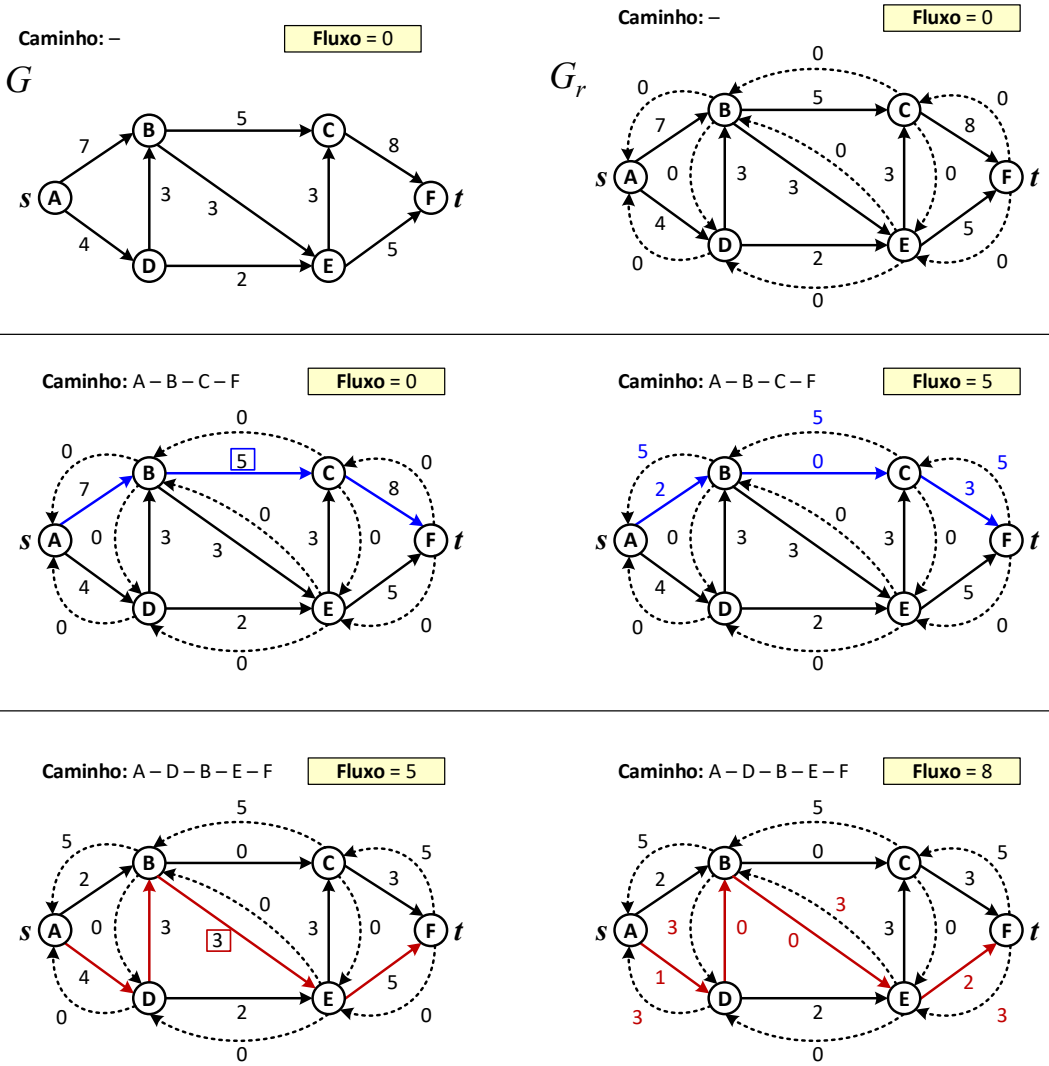
Algoritmo – Exemplo 2

```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4  |   c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5  |   for cada arco (v,u) ∈ p do
6  |   |   G_r[v][u] ← G_r[v][u] − c_r(p);
7  |   |   G_r[u][v] ← G_r[v][u] + c_r(p);
8  |   end
9  |   fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

As arestas originais e residuais do caminho p em G_r são atualizadas conforme $c_r(p) = 5$, além do fluxo.

O método considera outro caminho p, cuja capacidade residual é $c_r(p) = 3$.

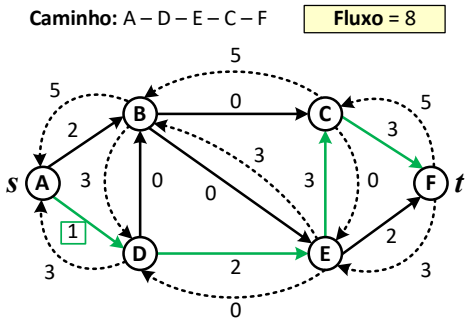
As arestas do caminho p são atualizadas, além do Fluxo Máximo.



Algoritmo – Exemplo 2

```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4  |   c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5  |   for cada arco (v,u) ∈ p do
6  |   |   G_r[v][u] ← G_r[v][u] − c_r(p);
7  |   |   G_r[u][v] ← G_r[v][u] + c_r(p);
8  |   end
9  |   fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

O algoritmo continua por outro caminho, o qual possui capacidade residual $c_r(p) = 1$.



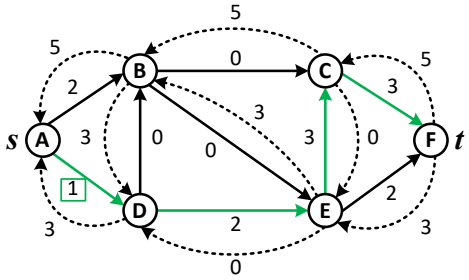
Algoritmo – Exemplo 2

```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4  |   c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5  |   for cada arco (v,u) ∈ p do
6  |   |   G_r[v][u] ← G_r[v][u] − c_r(p);
7  |   |   G_r[u][v] ← G_r[v][u] + c_r(p);
8  |   end
9  |   fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

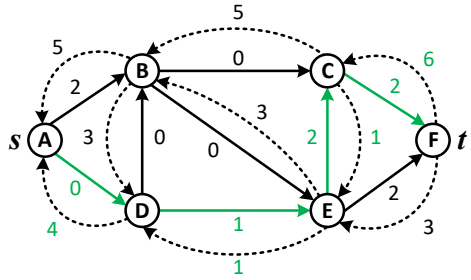
O algoritmo continua por outro caminho, o qual possui capacidade residual $c_r(p) = 1$.

As capacidades das arestas do caminho e o fluxo são atualizados.

Caminho: A – D – E – C – F **Fluxo** = 8



Caminho: A – D – E – C – F **Fluxo** = 9



Algoritmo – Exemplo 2

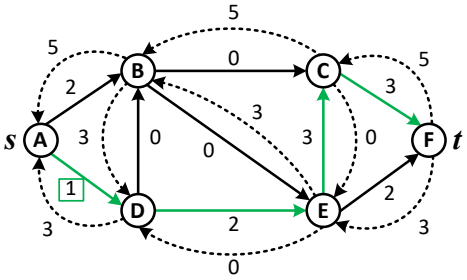
```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4  |   c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5  |   for cada arco (v,u) ∈ p do
6  |   |   G_r[v][u] ← G_r[v][u] − c_r(p);
7  |   |   G_r[u][v] ← G_r[v][u] + c_r(p);
8  |   end
9  |   fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

O algoritmo continua por outro caminho, o qual possui capacidade residual $c_r(p) = 1$.

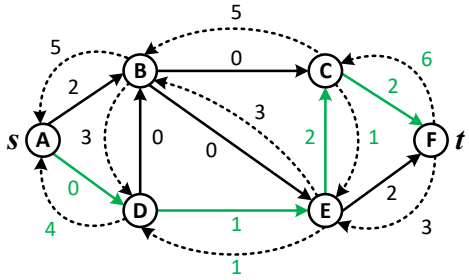
As capacidades das arestas do caminho são atualizadas.

Outro caminho é analisado. Note que ele contém uma aresta residual (B, D) na sua constituição.

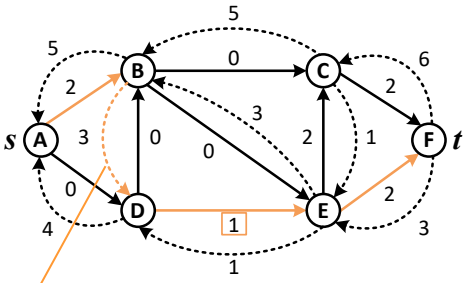
Caminho: A – D – E – C – F **Fluxo** = 8



Caminho: A – D – E – C – F **Fluxo** = 9



Caminho: A – B – D – E – F **Fluxo** = 9

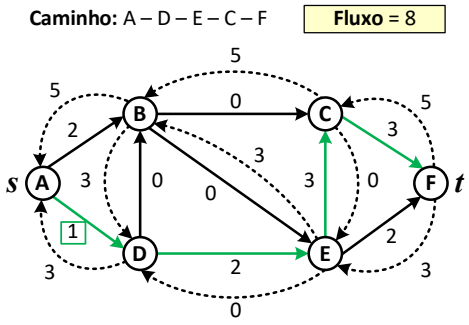


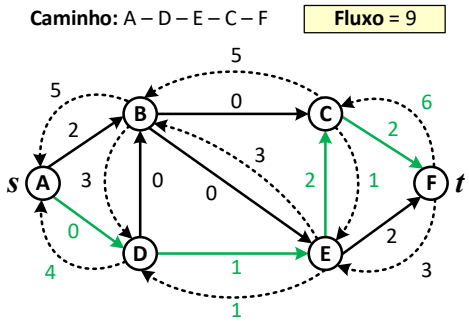
Isso significa desconsiderar a passagem de fluxo pela (D, B) para maximizar outro caminho.

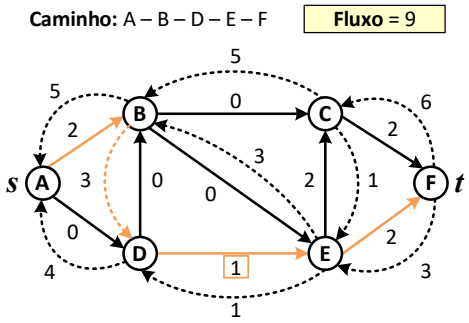
Algoritmo – Exemplo 2

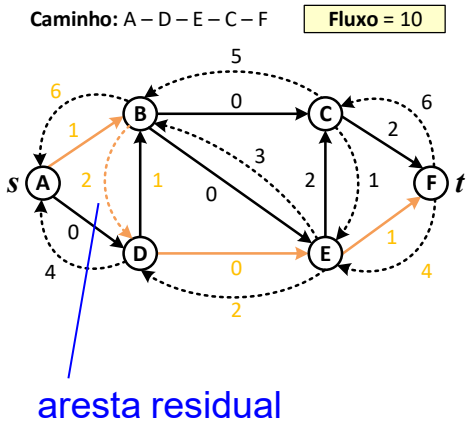
```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4      c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5      for cada arco (v,u) ∈ p do
6          G_r[v][u] ← G_r[v][u] − c_r(p);
7          G_r[u][v] ← G_r[v][u] + c_r(p);
8      end
9      fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

As arestas originais e residuais são atualizadas conforme a capacidade residual $c_r(p) = 1$, além do Fluxo.









Algoritmo – Exemplo 2

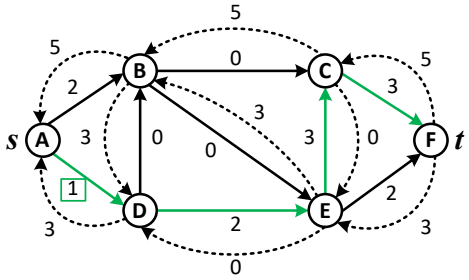
```
fordFulkerson(G, s, t)
1  G_r ← G;
2  fluxoMax ← 0;
3  while ∃ caminho p de s−t em G_r do
4  |   c_r(p) ← min{c_r(v,u) : (v,u) ∈ p};
5  |   for cada arco (v,u) ∈ p do
6  |   |   G_r[v][u] ← G_r[v][u] − c_r(p);
7  |   |   G_r[u][v] ← G_r[v][u] + c_r(p);
8  |   end
9  |   fluxoMax ← fluxoMax + c_r(p);
10 end
11 return fluxoMax
```

As arestas originais e residuais da rede são atualizadas conforme a capacidade residual $c_r(p) = 1$.

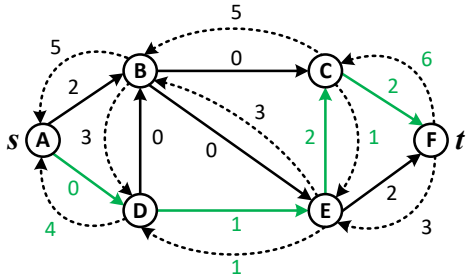
O método termina quando não existirem mais caminhos alternativos de A para F.

O Fluxo Máximo da rede é **10**.

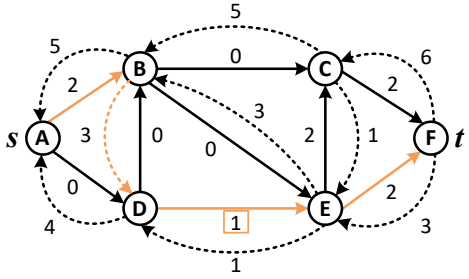
Caminho: A – D – E – C – F **Fluxo = 8**



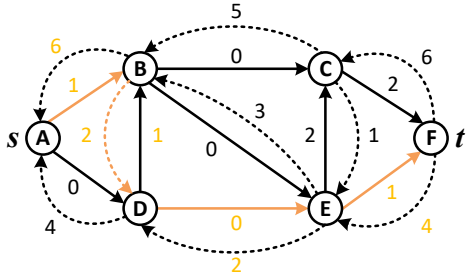
Caminho: A – D – E – C – F **Fluxo = 9**



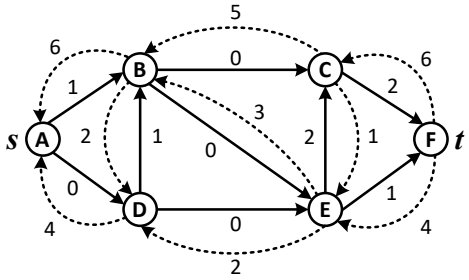
Caminho: A – B – D – E – F **Fluxo = 9**



Caminho: A – D – E – C – F **Fluxo = 10**



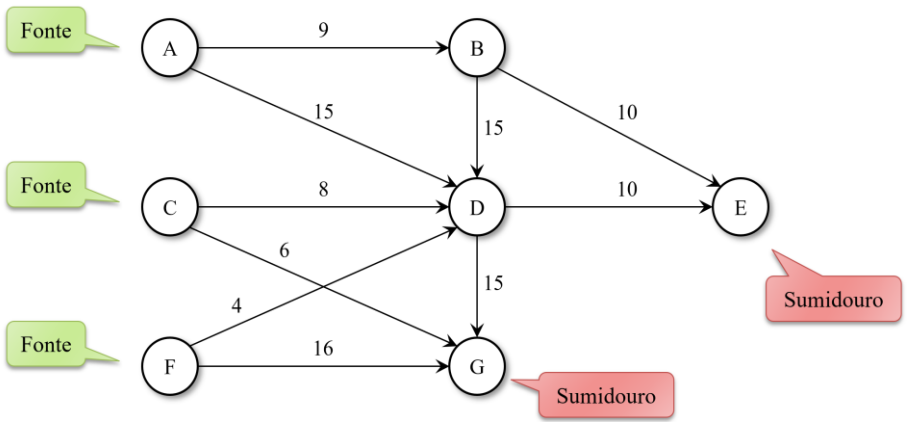
Fluxo Máximo = 10



Casos Especiais

Diversos problemas podem ser resolvidos como um problema de fluxo máximo, mas alguns deles parecem não se adequar a todas as restrições colocadas até então.

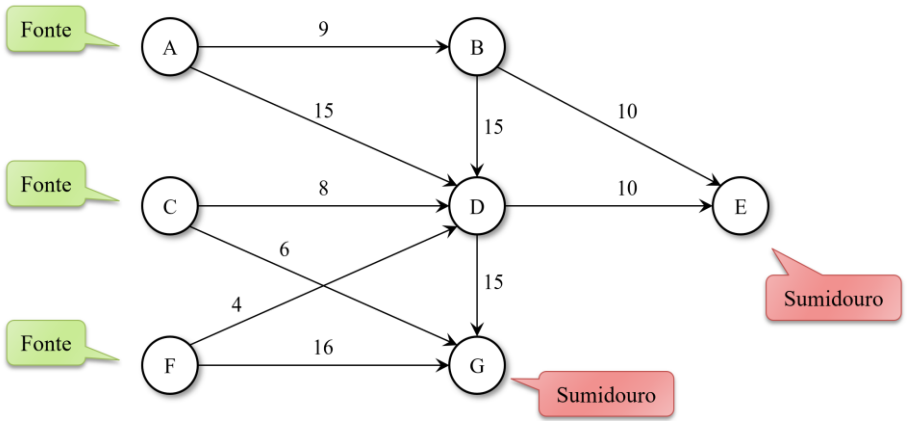
Em diversos destes casos é possível adequar a modelagem conforme um problema de fluxo máximo, apenas representando o grafo de forma conveniente.



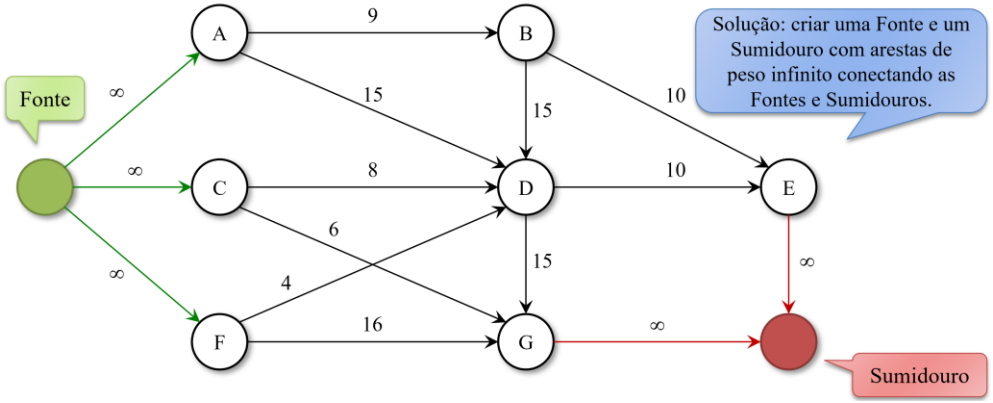
Casos Especiais

Diversos problemas podem ser resolvidos como um problema de fluxo máximo, mas alguns deles parecem não se adequar a todas as restrições colocadas até então.

Em diversos destes casos é possível adequar a modelagem conforme um problema de fluxo máximo, apenas representando o grafo de forma conveniente.



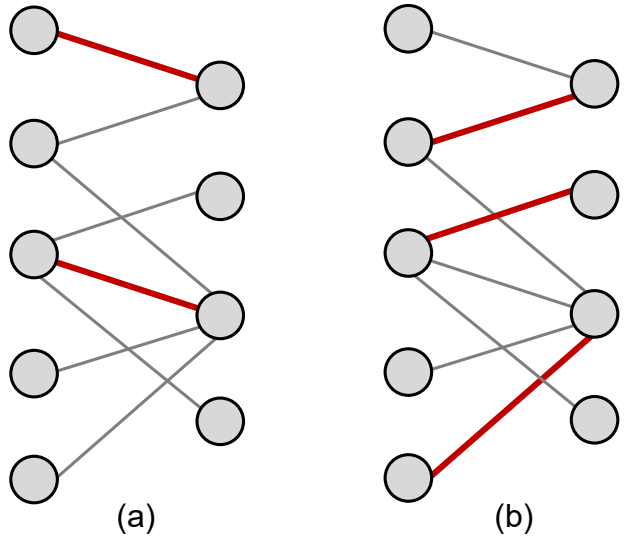
Um caso especial é quando existem múltiplas fontes ou sumidouros. A solução é incluir um vértice fonte e um sumidouro com as arestas de peso infinito conectando as fontes e sumidouros originais.



Redes de Fluxo com Múltiplas Origens e Sorvedouros

Redes de fluxo com múltiplas origens e sorvedouros podem ser utilizadas para solução do problema de Emparelhamento Máximo em Grafo Bipartido.

Dado um grafo não orientado $G = (V, E)$, um emparelhamento (*matching*) é um subconjunto de arestas $M \subseteq E$ tais que, para todos os vértices $v \in V$, no máximo uma aresta de M é incidente sobre v.



Exemplos de emparelhamento com cardinalidade 2 (a) e em (b) com cardinalidade 3 (máximo).

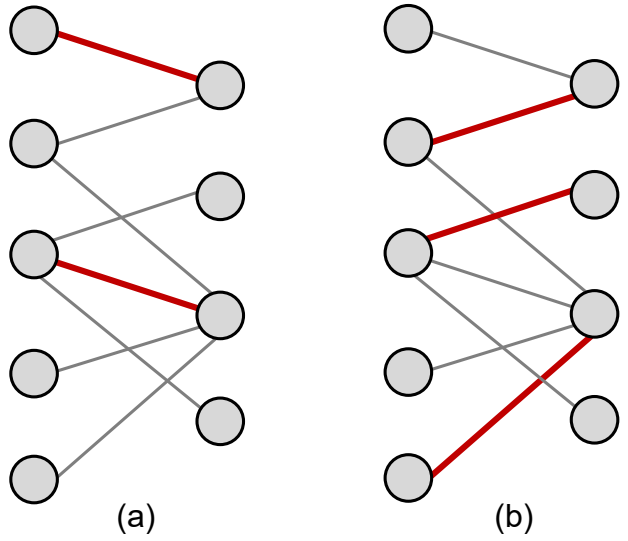
Redes de Fluxo com Múltiplas Origens e Sorvedouros

Redes de fluxo com múltiplas origens e sorvedouros podem ser utilizados para solução do problema de Emparelhamento Máximo em Grafo Bipartido.

Dado um grafo não orientado $G = (V, E)$, um emparelhamento (*matching*) é um subconjunto de arestas $M \subseteq E$ tais que, para todos os vértices $v \in V$, no máximo uma aresta de M é incidente sobre v.

Um vértice $v \in V$ é dito correspondido pelo emparelhamento se incide alguma aresta de M sobre ele, ou não-correspondido caso contrário.

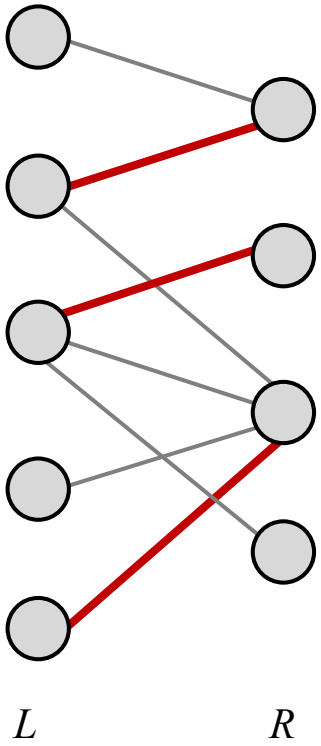
Um emparelhamento máximo é aquele que possui a maior cardinalidade, ou seja, é um emparelhamento M tal que, para qualquer emparelhamento M' temos $|M| \geq |M'|$



Exemplos de emparelhamento com cardinalidade 2 (a) e em (b) com cardinalidade 3 (máximo).

Emparelhamento Máximo em grafo Bipartido

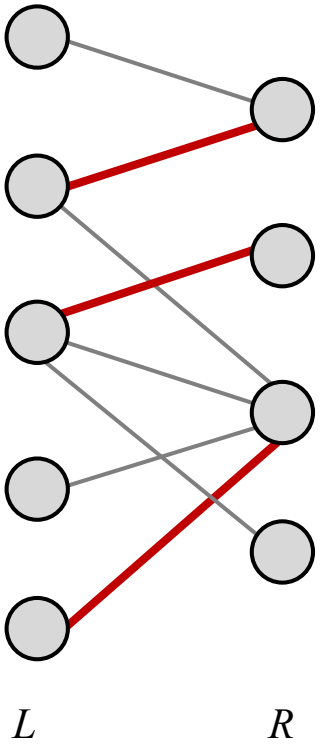
Possui aplicações práticas como, por exemplo, a alocação de tarefas a máquinas. Existe um conjunto de máquinas (L) e um conjunto de tarefas (R) a serem executadas simultaneamente.



Considera-se que uma aresta $(v, u) \in E$ indica que uma determinada máquina $v \in L$ é capaz de executar uma dada tarefa $u \in R$. O emparelhamento máximo fornece trabalho para a maior quantidade de máquinas possível.

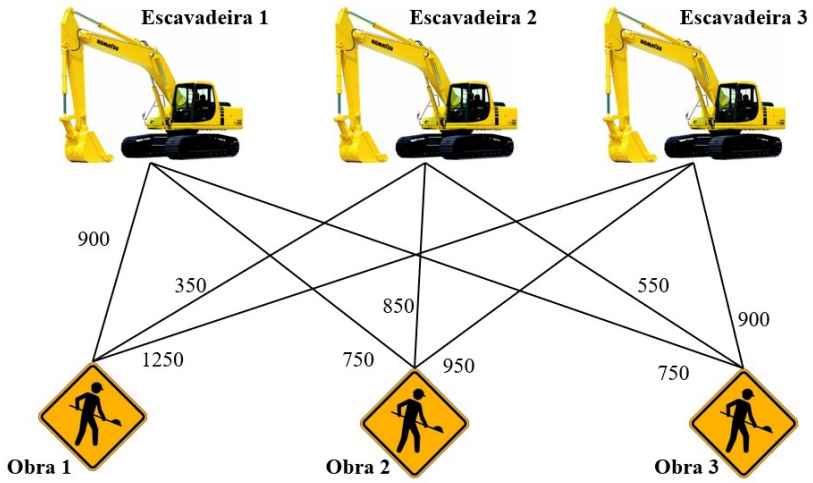
Emparelhamento Máximo em grafo Bipartido

Possui aplicações práticas como, por exemplo, a alocação de tarefas a máquinas. Existe um conjunto de máquinas (L) e um conjunto de tarefas (R) a serem executadas simultaneamente.



Considera-se que uma aresta $(v, u) \in E$ indica que uma determinada máquina $v \in L$ é capaz de executar uma dada tarefa $u \in R$. O emparelhamento máximo fornece trabalho para a maior quantidade de máquinas possível.

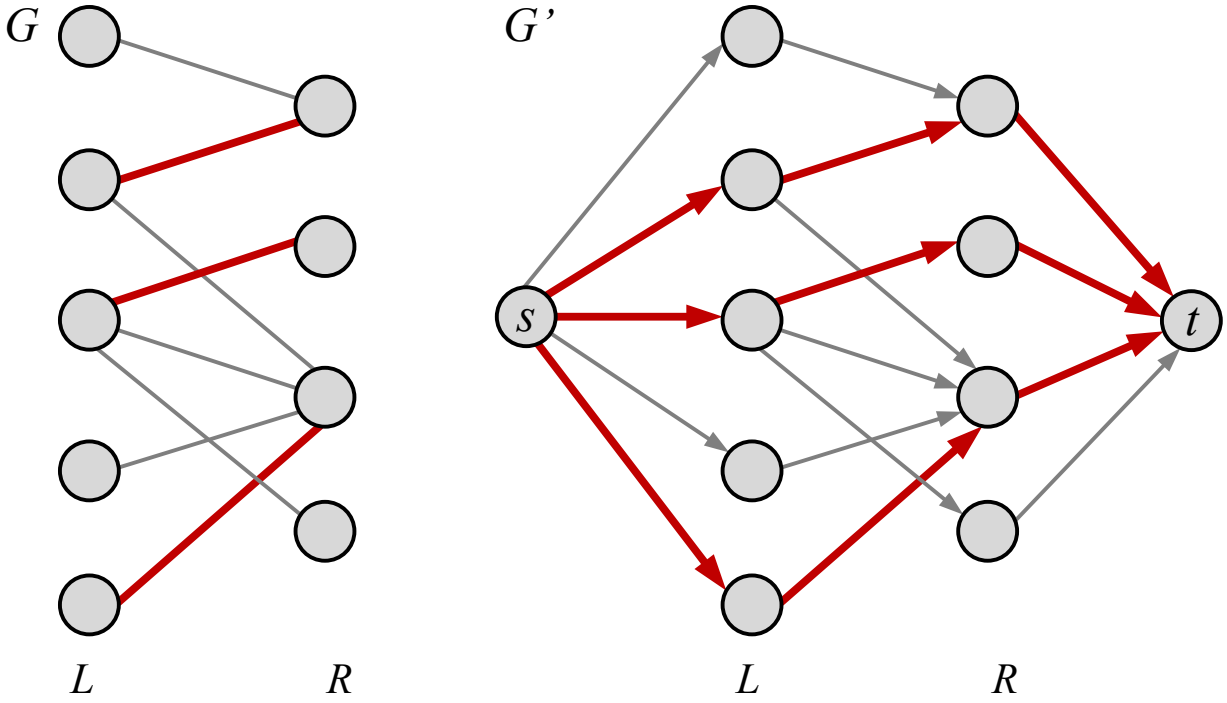
Ex. considere escavadeiras que estão em garagens diferentes precisam ser enviadas até o local de uma obra. O objetivo é alocar o máximo possível de máquinas e minimizar o custo de transporte.



Emparelhamento Máximo em grafo Bipartido

O método de Ford-Fulkerson pode ser utilizado para encontrar um emparelhamento máximo em um grafo bipartido não orientado $G = (V, E)$. A ideia é construir um fluxo em rede na qual os fluxos equivalem a emparelhamentos.

Define-se um fluxo em rede correspondente $G' = (V', E')$ para o grafo G.



São adicionados vértices origem e sorvedouro. As arestas em vermelho têm capacidade unitária, e as demais não conduzem fluxo (igual a 0).

As arestas em vermelho equivalem as arestas de um emparelhamento máximo em grafo bipartido.

Perguntas? Sugestões?