AED2 - Algoritmos e Estr. de Dados II

Aula 24: Fluxo Máximo

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

Fevereiro de 2026

Sumário

1. O Problema do Encanamento

Uma rede de tubulações conecta uma represa (**Fonte** s) a uma cidade (**Sorvedouro** t). Cada cano tem uma **capacidade** máxima.

Pergunta Central

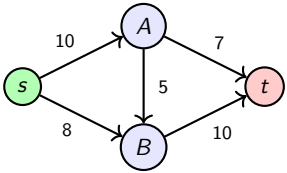
Qual a **vazão máxima** que conseguimos enviar de s a t, respeitando as capacidades?

Uma rede de tubulações conecta uma represa (**Fonte** s) a uma cidade (**Sorvedouro** t). Cada cano tem uma **capacidade** máxima.

Pergunta Central

Qual a **vazão máxima** que conseguimos enviar de s a t, respeitando as capacidades?

- **Fonte** s**:** produz fluxo (só sai).



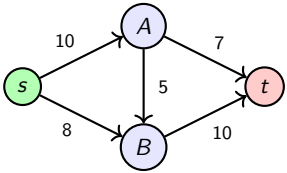
Números = capacidade máxima de cada aresta.

Uma rede de tubulações conecta uma represa (**Fonte** s) a uma cidade (**Sorvedouro** t). Cada cano tem uma **capacidade** máxima.

Pergunta Central

Qual a **vazão máxima** que conseguimos enviar de s a t, respeitando as capacidades?

- **Fonte** s**:** produz fluxo (só sai).
- **Sorvedouro** t**:** consome fluxo (só entra).



Números = capacidade máxima de cada aresta.

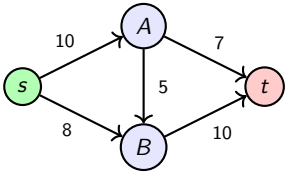
1. O Problema do Encanamento

Uma rede de tubulações conecta uma represa (**Fonte** s) a uma cidade (**Sorvedouro** t). Cada cano tem uma **capacidade** máxima.

Pergunta Central

Qual a **vazão máxima** que conseguimos enviar de s a t, respeitando as capacidades?

- **Fonte** s**:** produz fluxo (só sai).
- **Sorvedouro** t**:** consome fluxo (só entra).
- **Vértices internos:** o que entra = o que sai.



Números = capacidade máxima de cada aresta.

2. Definição Formal: Rede de Fluxo

Rede de Fluxo

Um grafo dirigido $G = (V, E)$ com:

Rede de Fluxo

Um grafo dirigido $G = (V, E)$ com:

- **Função capacidade:** $c : V \times V \to \mathbb{R}_{\geq 0}$, onde $c(u, v) = 0$ se $(u, v) \notin E$.

Rede de Fluxo

Um grafo dirigido $G = (V, E)$ com:

- **Função capacidade:** $c : V \times V \to \mathbb{R}_{\geq 0}$, onde $c(u, v) = 0$ se $(u, v) \notin E$.
- **Fonte** $s \in V$**:** onde o fluxo é produzido.

Rede de Fluxo

Um grafo dirigido $G = (V, E)$ com:

- **Função capacidade:** $c : V \times V \to \mathbb{R}_{\geq 0}$, onde $c(u, v) = 0$ se $(u, v) \notin E$.
- **Fonte** $s \in V$**:** onde o fluxo é produzido.
- **Sorvedouro** $t \in V$**:** onde o fluxo é consumido.

2. Definição Formal: Rede de Fluxo

Rede de Fluxo

Um grafo dirigido $G = (V, E)$ com:

- **Função capacidade:** $c : V \times V \to \mathbb{R}_{\geq 0}$, onde $c(u, v) = 0$ se $(u, v) \notin E$.
- **Fonte** $s \in V$**:** onde o fluxo é produzido.
- **Sorvedouro** $t \in V$**:** onde o fluxo é consumido.
- Para cada par (u, v): se $(u, v) \in E$, então $(v, u) \notin E$ (sem arestas anti-paralelas).

2. Definição Formal: Rede de Fluxo

Rede de Fluxo

Um grafo dirigido $G = (V, E)$ com:

- **Função capacidade:** $c : V \times V \to \mathbb{R}_{\geq 0}$, onde $c(u, v) = 0$ se $(u, v) \notin E$.
- **Fonte** $s \in V$**:** onde o fluxo é produzido.
- **Sorvedouro** $t \in V$**:** onde o fluxo é consumido.
- Para cada par (u, v): se $(u, v) \in E$, então $(v, u) \notin E$ (sem arestas anti-paralelas).

Observação Importante

Se o grafo original tem arestas anti-paralelas (u, v) e (v, u), basta inserir um vértice auxiliar w e trocar (u, v) por (u, w) e (w, v), ambas com a mesma capacidade.

Fluxo em uma Rede

Um **fluxo** é uma função $f : V \times V \to \mathbb{R}$ satisfazendo:

Fluxo em uma Rede

Um **fluxo** é uma função $f : V \times V \to \mathbb{R}$ satisfazendo:

1. **Restrição de capacidade:** $0 \leq f(u, v) \leq c(u, v) \quad \forall (u, v) \in V \times V$

Fluxo em uma Rede

Um **fluxo** é uma função $f : V \times V \to \mathbb{R}$ satisfazendo:

1. **Restrição de capacidade:** $0 \leq f(u, v) \leq c(u, v) \quad \forall (u, v) \in V \times V$
2. **Conservação de fluxo:** $\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v) \quad \forall u \in V \setminus \{s, t\}$

Fluxo em uma Rede

Um **fluxo** é uma função $f : V \times V \to \mathbb{R}$ satisfazendo:

1. **Restrição de capacidade:** $0 \leq f(u, v) \leq c(u, v) \quad \forall (u, v) \in V \times V$
2. **Conservação de fluxo:** $\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v) \quad \forall u \in V \setminus \{s, t\}$
3. **Valor do fluxo:** $|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s) =$ fluxo líquido saindo de s

Fluxo em uma Rede

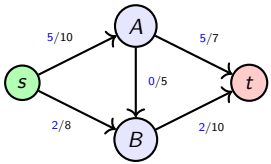
Um **fluxo** é uma função $f : V \times V \to \mathbb{R}$ satisfazendo:

1. **Restrição de capacidade:** $0 \leq f(u, v) \leq c(u, v) \quad \forall (u, v) \in V \times V$
2. **Conservação de fluxo:** $\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v) \quad \forall u \in V \setminus \{s, t\}$
3. **Valor do fluxo:** $|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s) =$ fluxo líquido saindo de s

Objetivo

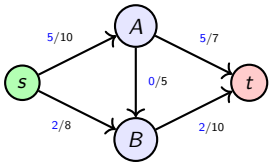
Encontrar um fluxo f que **maximize** $|f|$.

Fluxo Válido $(|f| = 7)$



4. Fluxo Válido vs Inválido

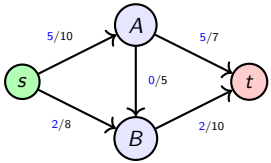
Fluxo Válido ($|f| = 7$)



- A: entra 5, sai $5 + 0 = 5$ ✓
- B: entra $2 + 0 = 2$, sai 2 ✓
- $f \leq c$ em todas as arestas ✓

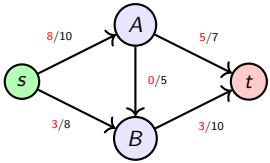
4. Fluxo Válido vs Inválido

Fluxo Válido ($|f| = 7$)



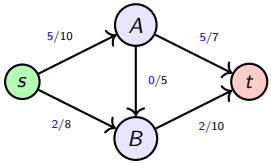
- A: entra 5, sai $5 + 0 = 5$ ✓
- B: entra $2 + 0 = 2$, sai 2 ✓
- $f \leq c$ em todas as arestas ✓

Fluxo Inválido!



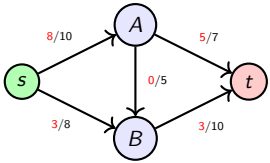
4. Fluxo Válido vs Inválido

Fluxo Válido ($|f| = 7$)



- A: entra 5, sai $5 + 0 = 5$ ✓
- B: entra $2 + 0 = 2$, sai 2 ✓
- $f \leq c$ em todas as arestas ✓

Fluxo Inválido!



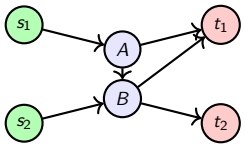
- A: entra **8**, sai $5 + 0 = \mathbf{5}$ $\quad 8 \neq 5$**!**
- Viola **conservação** no vértice A.
- Faltam 3 unidades — para onde foram?

5. Múltiplas Fontes e Sorvedouros

Muitos problemas reais têm **várias fontes** $s_1, s_2, \ldots$ e **vários sorvedouros** $t_1, t_2, \ldots$.

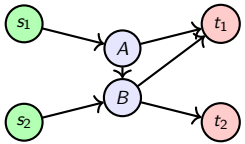
Muitos problemas reais têm **várias fontes** $s_1, s_2, \ldots$ e **vários sorvedouros** $t_1, t_2, \ldots$.

Antes

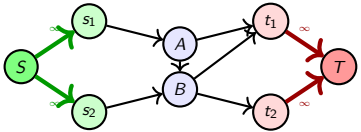


Muitos problemas reais têm **várias fontes** $s_1, s_2, \ldots$ e **vários sorvedouros** $t_1, t_2, \ldots$.

Antes

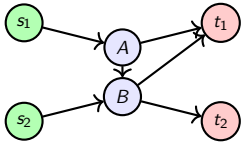


Depois (Super-S e Super-T)

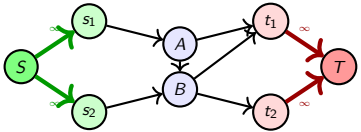


Muitos problemas reais têm **várias fontes** $s_1, s_2, \ldots$ e **vários sorvedouros** $t_1, t_2, \ldots$.

Antes



Depois (Super-S e Super-T)



Técnica

Crie um **super-source** S ligado a cada fonte com capacidade ∞ (ou a produção da fonte), e um **super-sink** T ligado a cada sorvedouro com capacidade ∞ (ou a demanda).

6. Caminho Aumentante: Intuição

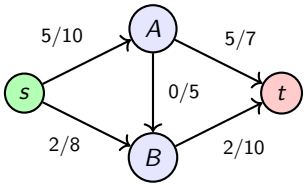
Definição

Um **caminho aumentante** é um caminho de s a t no **grafo residual** onde todas as arestas têm capacidade residual > 0.

6. Caminho Aumentante: Intuição

Definição

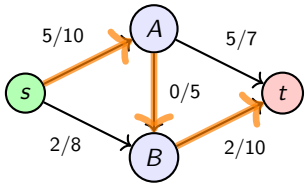
Um **caminho aumentante** é um caminho de s a t no **grafo residual** onde todas as arestas têm capacidade residual > 0.



6. Caminho Aumentante: Intuição

Definição

Um **caminho aumentante** é um caminho de s a t no **grafo residual** onde todas as arestas têm capacidade residual > 0.

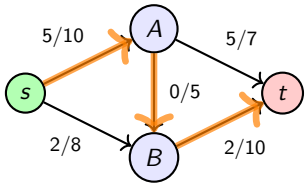


Caminho: $s \to A \to B \to T$, gargalo $= \min(5, 5, 8) = 5$

6. Caminho Aumentante: Intuição

Definição

Um **caminho aumentante** é um caminho de s a t no **grafo residual** onde todas as arestas têm capacidade residual > 0.



Caminho: $s \to A \to B \to T$, gargalo $= \min(5, 5, 8) = 5$

7. Grafo Residual: Para que Serve?

O Problema

Imagine que você enviou fluxo por um caminho e depois descobre que **tomou uma decisão ruim**. O fluxo foi por uma aresta que **deveria ter ficado livre** para outro caminho mais importante. Como consertar?

7. Grafo Residual: Para que Serve?

O Problema

Imagine que você enviou fluxo por um caminho e depois descobre que **tomou uma decisão ruim**. O fluxo foi por uma aresta que **deveria ter ficado livre** para outro caminho mais importante. Como consertar?

Solução: O Grafo Residual

O **grafo residual** G_f é uma **visão alternativa** da rede que, a cada momento, mostra:

7. Grafo Residual: Para que Serve?

O Problema

Imagine que você enviou fluxo por um caminho e depois descobre que **tomou uma decisão ruim**. O fluxo foi por uma aresta que **deveria ter ficado livre** para outro caminho mais importante. Como consertar?

Solução: O Grafo Residual

O **grafo residual** G_f é uma **visão alternativa** da rede que, a cada momento, mostra:

1. **Aresta de ida** (u, v): quanto **ainda cabe** enviar $\Rightarrow c_f(u, v) = c - f$

7. Grafo Residual: Para que Serve?

O Problema

Imagine que você enviou fluxo por um caminho e depois descobre que **tomou uma decisão ruim**. O fluxo foi por uma aresta que **deveria ter ficado livre** para outro caminho mais importante. Como consertar?

Solução: O Grafo Residual

O **grafo residual** G_f é uma **visão alternativa** da rede que, a cada momento, mostra:

1. **Aresta de ida** (u, v): quanto **ainda cabe** enviar $\Rightarrow c_f(u, v) = c - f$
2. **Aresta reversa** (v, u): quanto podemos **"devolver"** $\Rightarrow c_f(v, u) = f$

7. Grafo Residual: Para que Serve?

O Problema

Imagine que você enviou fluxo por um caminho e depois descobre que **tomou uma decisão ruim**. O fluxo foi por uma aresta que **deveria ter ficado livre** para outro caminho mais importante. Como consertar?

Solução: O Grafo Residual

O **grafo residual** G_f é uma **visão alternativa** da rede que, a cada momento, mostra:

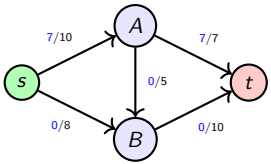
1. **Aresta de ida** (u, v): quanto **ainda cabe** enviar $\Rightarrow c_f(u, v) = c - f$
2. **Aresta reversa** (v, u): quanto podemos **"devolver"** $\Rightarrow c_f(v, u) = f$

Exemplo: aresta original $A \to B$ com cap $= 10$, fluxo $= 6$

ida: $10 - 6 = 4$

8. Grafo Residual: Exemplo Completo

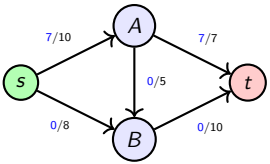
Grafo com fluxo atual ($|f| = 7$)



Fluxo depois do caminho $s \to A \to t$ com gargalo 7.

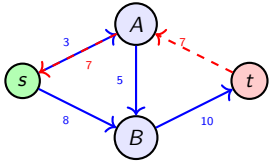
8. Grafo Residual: Exemplo Completo

Grafo com fluxo atual ($|f| = 7$)



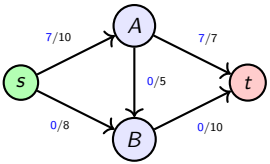
Fluxo depois do caminho $s \to A \to t$ com gargalo 7.

Grafo Residual G_f



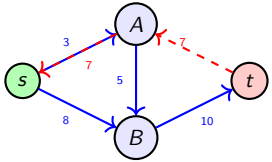
8. Grafo Residual: Exemplo Completo

Grafo com fluxo atual ($|f| = 7$)



Fluxo depois do caminho $s \to A \to t$ com gargalo 7.

Grafo Residual G_f



- $s \to A$: ida=$10 - 7 = 3$, volta=7
- $A \to t$: ida=$7 - 7 = \mathbf{0}$ (não aparece!), volta=7
- $s \to B$: ida=8, volta=0 (não aparece)
- $A \to B$: ida=5, volta=0 (não aparece)
- $B \to t$: ida=10, volta=0 (não aparece)

9. O Grafo "Diamante": O Problema de Decisões Ruins

Motivação

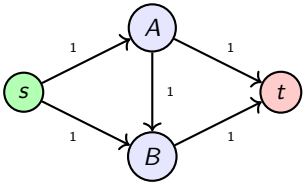
O grafo diamante é o **menor exemplo possível** que mostra por que um algoritmo guloso **sem arestas reversas** pode falhar. É a **prova de conceito** de que precisamos do grafo residual.

9. O Grafo "Diamante": O Problema de Decisões Ruins

Motivação

O grafo diamante é o **menor exemplo possível** que mostra por que um algoritmo guloso **sem arestas reversas** pode falhar. É a **prova de conceito** de que precisamos do grafo residual.

O Grafo (todas caps = 1)



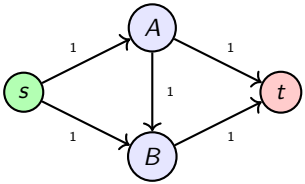
A olho: dois caminhos independentes $s \to A \to t$ e $s \to B \to t$. Fluxo máximo $= 2$.

9. O Grafo "Diamante": O Problema de Decisões Ruins

Motivação

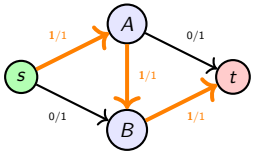
O grafo diamante é o **menor exemplo possível** que mostra por que um algoritmo guloso **sem arestas reversas** pode falhar. É a **prova de conceito** de que precisamos do grafo residual.

O Grafo (todas caps = 1)



A olho: dois caminhos independentes $s \to A \to t$ e $s \to B \to t$. Fluxo máximo $= 2$.

Passo 1 (decisão ruim):
$s \to A \to B \to t$ (gargalo 1)

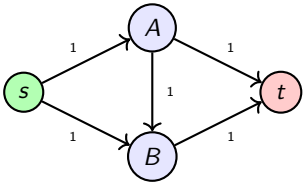


9. O Grafo "Diamante": O Problema de Decisões Ruins

Motivação

O grafo diamante é o **menor exemplo possível** que mostra por que um algoritmo guloso **sem arestas reversas** pode falhar. É a **prova de conceito** de que precisamos do grafo residual.

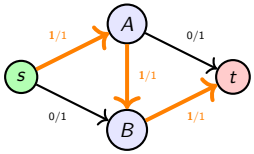
O Grafo (todas caps = 1)



A olho: dois caminhos independentes $s \to A \to t$ e $s \to B \to t$. Fluxo máximo = **2**.

Passo 1 (decisão ruim):

$s \to A \to B \to t$ (gargalo 1)

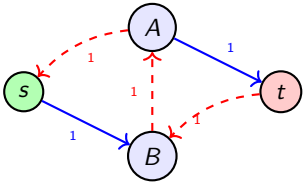


$|f| = 1$. Fluxo passou por $A \to B$, **"desperdiçando"** A que poderia ir direto para t, e **bloqueando** B que poderia receber de s.

Sem aresta reversa: $s \to A$ saturada, $B \to t$ saturada $\Rightarrow$ sem caminho. Parou com $|f| = 1$!

9b. O Grafo “Diamante”: Aresta Reversa Salva!

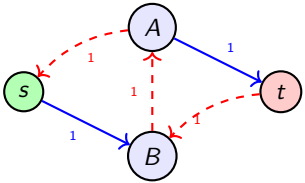
Grafo Residual após Passo 1



A aresta $B \to A$ (reversa de $A \to B$) é a chave!

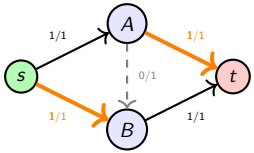
9b. O Grafo "Diamante": Aresta Reversa Salva!

Grafo Residual após Passo 1



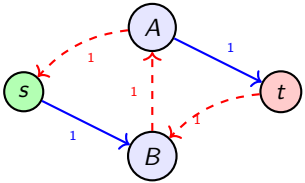
A aresta $B \to A$ (reversa de $A \to B$) é a chave!

Passo 2: $s \to B \to A \to t$ (gargalo 1)



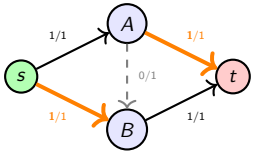
9b. O Grafo "Diamante": Aresta Reversa Salva!

Grafo Residual após Passo 1



A aresta $B \to A$ (reversa de $A \to B$) é a chave!

Passo 2: $s \to B \to A \to t$ (gargalo 1)



O que aconteceu de fato?

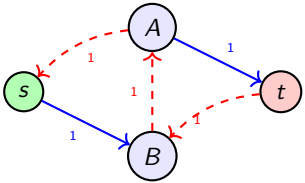
A aresta reversa **cancelou** o fluxo errado em $A \to B$:

- **Antes:** $s \to A \to B \to t$ (ruim)
- **Depois:** $s \to A \to t$ e $s \to B \to t$ (ótimo!)

Fluxo máximo = 2!

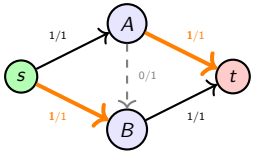
9b. O Grafo "Diamante": Aresta Reversa Salva!

Grafo Residual após Passo 1



A aresta $B \to A$ (reversa de $A \to B$) é a chave!

Passo 2: $s \to B \to A \to t$ (gargalo 1)



O que aconteceu de fato?

A aresta reversa **cancelou** o fluxo errado em $A \to B$:

- **Antes:** $s \to A \to B \to t$ (ruim)
- **Depois:** $s \to A \to t$ e $s \to B \to t$ (ótimo!)

Fluxo máximo = 2!

Conclusão: Por que o Grafo Residual Existe

Arestas reversas são o **mecanismo de correção** do algoritmo. Elas permitem "desfazer" fluxo que passou por um caminho ruim e **redirecioná-lo** por caminhos melhores. Sem elas, qualquer decisão é **irreversível** e pode levar a resultados subótimos.

10. Resumo: O Grafo Residual na Estratégia Algorítmica

O Grafo Residual serve para duas coisas:

10. Resumo: O Grafo Residual na Estratégia Algorítmica

O Grafo Residual serve para duas coisas:

1. **Encontrar oportunidades:** caminhos de s a t indicam que dá para enviar **mais fluxo**.

10. Resumo: O Grafo Residual na Estratégia Algorítmica

O Grafo Residual serve para duas coisas:

1. **Encontrar oportunidades:** caminhos de s a t indicam que dá para enviar **mais fluxo**.
2. **Corrigir erros:** arestas reversas permitem **redirecionar** fluxo de caminhos ruins para melhores.

10. Resumo: O Grafo Residual na Estratégia Algorítmica

O Grafo Residual serve para duas coisas:

1. **Encontrar oportunidades:** caminhos de s a t indicam que dá para enviar **mais fluxo**.
2. **Corrigir erros:** arestas reversas permitem **redirecionar** fluxo de caminhos ruins para melhores.

Lema Fundamental

Um fluxo f é **máximo** se e somente se **não existe** caminho aumentante de s a t no grafo residual G_f.

10. Resumo: O Grafo Residual na Estratégia Algorítmica

O Grafo Residual serve para duas coisas:

1. **Encontrar oportunidades:** caminhos de s a t indicam que dá para enviar **mais fluxo**.
2. **Corrigir erros:** arestas reversas permitem **redirecionar** fluxo de caminhos ruins para melhores.

Lema Fundamental

Um fluxo f é **máximo** se e somente se **não existe** caminho aumentante de s a t no grafo residual G_f.

Sem caminho ⇒ máximo

Se t é inalcançável em G_f, os vértices alcançáveis formam um **corte**. Todas as arestas do corte estão **saturadas** ⇒ impossível enviar mais.

Máximo ⇒ sem caminho

Se houvesse caminho com cap. residual $c > 0$, poderíamos enviar mais c unidades — **contradição**.

10. Resumo: O Grafo Residual na Estratégia Algorítmica

O Grafo Residual serve para duas coisas:

1. **Encontrar oportunidades:** caminhos de s a t indicam que dá para enviar **mais fluxo**.
2. **Corrigir erros:** arestas reversas permitem **redirecionar** fluxo de caminhos ruins para melhores.

Lema Fundamental

Um fluxo f é **máximo** se e somente se **não existe** caminho aumentante de s a t no grafo residual G_f.

Sem caminho ⇒ máximo

Se t é inalcançável em G_f, os vértices alcançáveis formam um **corte**. Todas as arestas do corte estão **saturadas** ⇒ impossível enviar mais.

Máximo ⇒ sem caminho

Se houvesse caminho com cap. residual $c > 0$, poderíamos enviar mais c unidades — **contradição**.

Pseudocódigo

Pseudocódigo

1. Inicialize $f(u, v) = 0$ para todas as arestas.

Pseudocódigo

1. Inicialize $f(u, v) = 0$ para todas as arestas.
2. **Enquanto** existir caminho aumentante P de s a t em G_f:
   - $c_f(P) = \min\{c_f(u, v) : (u, v) \in P\}$ (gargalo)
   - Para cada aresta $(u, v) \in P$:
     - $f(u, v) \leftarrow f(u, v) + c_f(P)$ (aumenta na ida)
     - $f(v, u) \leftarrow f(v, u) - c_f(P)$ ("devolve" na volta)

Pseudocódigo

1. Inicialize $f(u, v) = 0$ para todas as arestas.
2. **Enquanto** existir caminho aumentante P de s a t em G_f:
   - $c_f(P) = \min\{c_f(u, v) : (u, v) \in P\}$ (gargalo)
   - Para cada aresta $(u, v) \in P$:
     - $f(u, v) \leftarrow f(u, v) + c_f(P)$ (aumenta na ida)
     - $f(v, u) \leftarrow f(v, u) - c_f(P)$ ("devolve" na volta)
3. Retorne $|f|$.

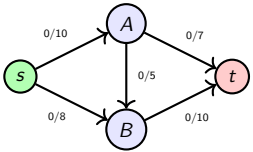
Pseudocódigo

1. Inicialize $f(u, v) = 0$ para todas as arestas.
2. **Enquanto** existir caminho aumentante P de s a t em G_f:
   - $c_f(P) = \min\{c_f(u, v) : (u, v) \in P\}$ (gargalo)
   - Para cada aresta $(u, v) \in P$:
     - $f(u, v) \leftarrow f(u, v) + c_f(P)$ (aumenta na ida)
     - $f(v, u) \leftarrow f(v, u) - c_f(P)$ ("devolve" na volta)
3. Retorne $|f|$.

Atenção: Complexidade depende da busca!

- Se usa **DFS (qualquer caminho):** $O(E \cdot |f^*|)$ — pode ser **exponencial**!
- Se usa **BFS (caminho mais curto):** $O(V \cdot E^2)$ — **polinomial** (Edmonds-Karp).

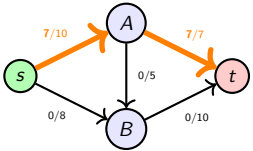
12. Ford-Fulkerson: Trace Animado



Estado inicial: $|f| = 0$

It.	Caminho	Garg.	$\lvert f \rvert$

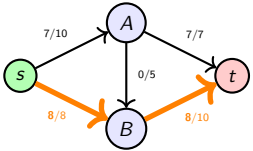
12. Ford-Fulkerson: Trace Animado



Iter 1: $s \to A \to t$, garg. $= 7$

It.	Caminho	Garg.	$\lvert f \rvert$
1	$s \to A \to t$	7	7

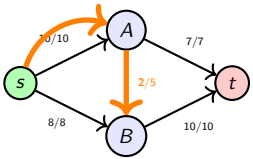
12. Ford-Fulkerson: Trace Animado



Iter 2: $s \to B \to t$, garg. $= 8$

It.	Caminho	Garg.	$\lvert f \rvert$
1	$s \to A \to t$	7	7
2	$s \to B \to t$	8	15

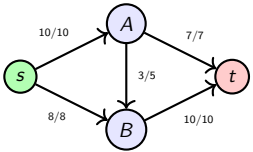
12. Ford-Fulkerson: Trace Animado



Iter 3: $s \to A \to B \to t$, garg. $= 2$

It.	Caminho	Garg.	$\lvert f \rvert$
1	$s \to A \to t$	7	7
2	$s \to B \to t$	8	15
3	$s \to A \to B \to t$	2	**17**

12. Ford-Fulkerson: Trace Animado

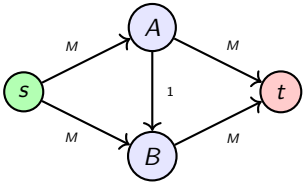


Fluxo máximo = 17!

It.	Caminho	Garg.	$\|f\|$
1	$s \to A \to t$	7	7
2	$s \to B \to t$	8	15
3	$s \to A \to B \to t$	2	**17**
—	Sem caminho	—	**17**

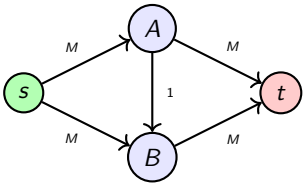
Sem caminho aumentante no residual $\Rightarrow$ **ótimo!**

13. Ford-Fulkerson com DFS: O Caso Patológico



Fluxo máximo $= 2M$, mas DFS pode demorar $2M$ iterações!

13. Ford-Fulkerson com DFS: O Caso Patológico



Fluxo máximo $= 2M$, mas DFS pode demorar $2M$ iterações!

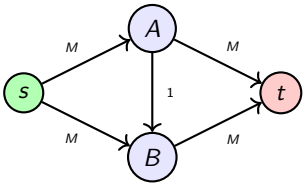
O Problema

Se DFS escolhe alternadamente:

1. $s \to A \to B \to t$ (gargalo 1)
2. $s \to B \to A \to t$ (gargalo 1, via reversa)

Cada iteração aumenta o fluxo em apenas **1**!
Com $M = 10^6$: 2×10^6 **iterações!**

13. Ford-Fulkerson com DFS: O Caso Patológico



Fluxo máximo $= 2M$, mas DFS pode demorar $2M$ iterações!

O Problema

Se DFS escolhe alternadamente:

1. $s \to A \to B \to t$ (gargalo 1)
2. $s \to B \to A \to t$ (gargalo 1, via reversa)

Cada iteração aumenta o fluxo em apenas **1**!
Com $M = 10^6$: 2×10^6 **iterações!**

Solução

Edmonds-Karp: BFS escolhe caminhos mais curtos.
Primeira iteração já manda M de uma vez:

- $s \to A \to t$ (gargalo M)
- $s \to B \to t$ (gargalo M)

Apenas **2 iterações!**

Complexidade: $O(E \cdot |f^*|)$

- Cada iteração encontra um caminho em $O(E)$ (DFS/BFS).
- Cada iteração aumenta o fluxo em ≥ 1 (se capacidades inteiras).
- No máximo $|f^*|$ iterações.

14. Ford-Fulkerson: Análise de Complexidade

Complexidade: $O(E \cdot |f^*|)$

- Cada iteração encontra um caminho em $O(E)$ (DFS/BFS).
- Cada iteração aumenta o fluxo em ≥ 1 (se capacidades inteiras).
- No máximo $|f^*|$ iterações.

Problemas

- Se $|f^*|$ é grande, é muito lento.
- Com capacidades **irracionais**: pode **não convergir**!
- Com capacidades **reais**: pode convergir para valor errado.

Quando é OK?

- Capacidades **inteiras pequenas** ($|f^*|$ pequeno).
- Grafos com caps unitárias (matching).
- Na prática, quase sempre use **Edmonds-Karp** ou **Dinic**.

15. Edmonds-Karp: BFS Garante Polinomial

Edmonds-Karp = Ford-Fulkerson + BFS

Em vez de qualquer caminho, sempre escolha o **caminho mais curto** (menor número de arestas) via BFS.

15. Edmonds-Karp: BFS Garante Polinomial

Edmonds-Karp = Ford-Fulkerson + BFS

Em vez de qualquer caminho, sempre escolha o **caminho mais curto** (menor número de arestas) via BFS.

Lema-Chave (sem prova)

Para todo vértice v, a distância $d(s, v)$ no grafo residual **nunca diminui** ao longo das iterações.

Consequência: cada aresta pode ser "crítica" (gargalo) no máximo $O(V/2)$ vezes.

15. Edmonds-Karp: BFS Garante Polinomial

Edmonds-Karp = Ford-Fulkerson + BFS

Em vez de qualquer caminho, sempre escolha o **caminho mais curto** (menor número de arestas) via BFS.

Lema-Chave (sem prova)

Para todo vértice v, a distância $d(s, v)$ no grafo residual **nunca diminui** ao longo das iterações.

Consequência: cada aresta pode ser "crítica" (gargalo) no máximo $O(V/2)$ vezes.

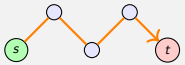
Complexidade: $O(V \cdot E^2)$

- Há $O(VE)$ aumentações no total.
- Cada BFS custa $O(E)$.
- Total: $O(V \cdot E^2)$ — **independente** do valor do fluxo!

16. Edmonds-Karp: Por que BFS?

DFS (Ford-Fulkerson)

- Pode escolher caminhos longos e "tortuosos".
- Pode enviar fluxo mínimo (gargalo $= 1$).
- Número de iterações $\propto |f^*|$.

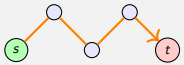


Caminho longo

16. Edmonds-Karp: Por que BFS?

DFS (Ford-Fulkerson)

- Pode escolher caminhos longos e "tortuosos".
- Pode enviar fluxo mínimo (gargalo $= 1$).
- Número de iterações $\propto |f^*|$.



Caminho longo

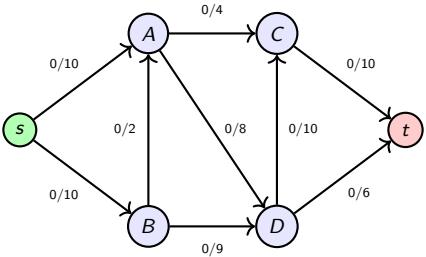
BFS (Edmonds-Karp)

- Sempre escolhe o caminho **mais curto**.
- Distâncias só crescem $\Rightarrow$ progresso garantido.
- Número de iterações $\leq O(VE)$.

Caminho curto

17. Edmonds-Karp: Trace Completo (Parte 1/3)

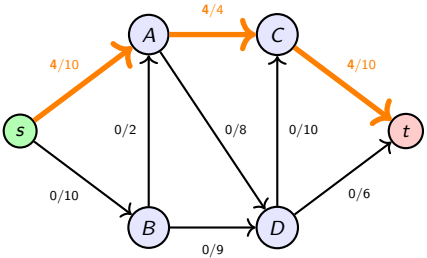
Rede com 6 vértices:



Estado inicial: $|f| = 0$

17. Edmonds-Karp: Trace Completo (Parte 1/3)

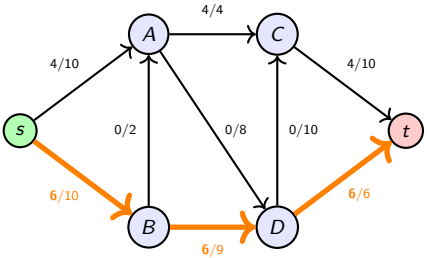
Rede com 6 vértices:



Iter 1: $s \to A \to C \to t$, gargalo=$\min(10, 4, 10) = 4$. $|f| = 4$

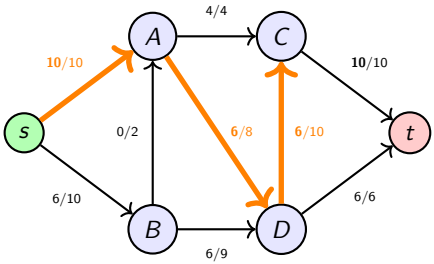
17. Edmonds-Karp: Trace Completo (Parte 1/3)

Rede com 6 vértices:



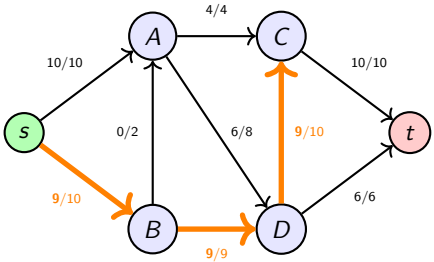
Iter 2: $s \to B \to D \to t$, gargalo=$\min(10, 9, 6) = 6$. $|f| = 10$

18. Edmonds-Karp: Trace Completo (Parte 2/3)



Iter 3: $s \to A \to D \to C \to t$, gargalo=$\min(6, 8, 10, 6) = 6$. $|f| = 16$

18. Edmonds-Karp: Trace Completo (Parte 2/3)

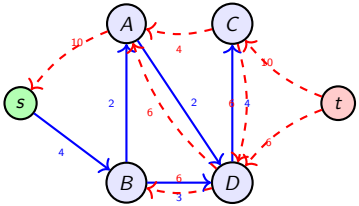


Iter 4: $s \to B \to D \to C \to t$, garg.=min$(4, 3, 4, 0)$... Vejamos o residual!

19. Edmonds-Karp: Trace Completo (Parte 3/3)

Grafo Residual após Iter 3

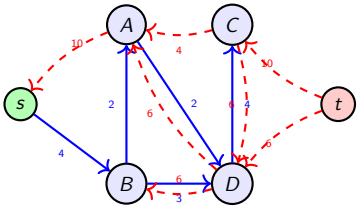
$(|f| = 16)$



19. Edmonds-Karp: Trace Completo (Parte 3/3)

Grafo Residual após Iter 3

($|f| = 16$)



BFS no Residual

BFS a partir de s:

$s \to B$ (cap 4) $\to D$ (cap 3) $\to C$ (cap 4) $\to \ldots$

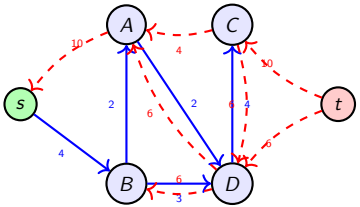
Mas $C \to t$? Não existe no residual (cap=0)!

BFS encontra que t é **inalcançável**.

19. Edmonds-Karp: Trace Completo (Parte 3/3)

Grafo Residual após Iter 3

($|f| = 16$)



BFS no Residual

BFS a partir de s:

$s \to B$ (cap 4) $\to D$ (cap 3) $\to C$ (cap 4) $\to \ldots$

Mas $C \to t$? Não existe no residual (cap=0)!

BFS encontra que t é **inalcançável**.

Resultado Final

Fluxo máximo = 16

Verificação:

Saída de s: $10 + 6 = 16$ ✓

Entrada em t: $10 + 6 = 16$ ✓

20. Edmonds-Karp: Implementação Java

```java
public static int edmondsKarp(int N, int[][] cap, int s, int t) {
    int flow = 0;
    int[] parent = new int[N];
    while (true) {
        // BFS para encontrar caminho aumentante
        Arrays.fill(parent, -1);
        Queue<Integer> q = new LinkedList<>();
        q.add(s); parent[s] = s;
        while (!q.isEmpty()) {
            int u = q.poll();
            if (u == t) break;
            for (int v = 0; v < N; v++)
                if (parent[v] == -1 && cap[u][v] > 0)
                    { parent[v] = u; q.add(v); }
        }
        if (parent[t] == -1) break; // Sem caminho

        // Encontrar gargalo
        int push = Integer.MAX_VALUE;
        for (int v = t; v != s; v = parent[v])
            push = Math.min(push, cap[parent[v]][v]);

        // Atualizar residual
        flow += push;
        for (int v = t; v != s; v = parent[v]) {
            cap[parent[v]][v] -= push;  // Diminui ida
            cap[v][parent[v]] += push;  // Aumenta volta
        }
    }
    return flow;
```

21. O que é um Corte s-t?

Definição

Um **corte** s-t é uma partição de V em dois conjuntos S e $T = V \setminus S$ tal que $s \in S$ e $t \in T$.

21. O que é um Corte s-t?

Definição

Um **corte** s-t é uma partição de V em dois conjuntos S e $T = V \setminus S$ tal que $s \in S$ e $t \in T$.

Capacidade do Corte

$$c(S, T) = \sum_{\substack{u \in S \\ v \in T}} c(u, v)$$

Somamos apenas as capacidades das arestas que **cruzam de S para T** (não de T para S!).

21. O que é um Corte s-t?

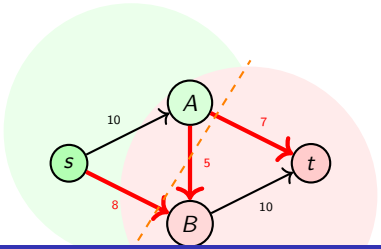
Definição

Um **corte** s-t é uma partição de V em dois conjuntos S e $T = V \setminus S$ tal que $s \in S$ e $t \in T$.

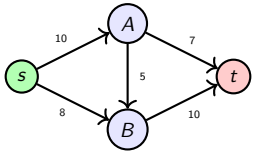
Capacidade do Corte

$$c(S, T) = \sum_{\substack{u \in S \\ v \in T}} c(u, v)$$

Somamos apenas as capacidades das arestas que **cruzam de S para T** (não de T para S!).

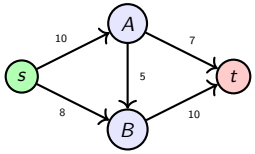


22. Exemplo: Todos os Cortes de um Grafo



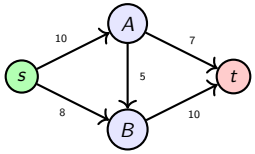
#	S	**Arestas** $S \to T$	**Capacidade**

22. Exemplo: Todos os Cortes de um Grafo



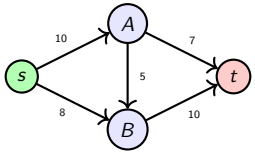
#	S	**Arestas $S \to T$**	**Capacidade**
1	$\{s\}$	$s \to A$ (10), $s \to B$ (8)	18

22. Exemplo: Todos os Cortes de um Grafo



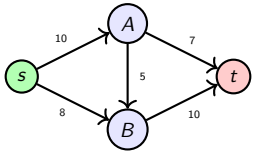
#	S	**Arestas** $S \to T$	**Capacidade**
1	$\{s\}$	$s \to A$ (10), $s \to B$ (8)	18
2	$\{s, A\}$	$s \to B$ (8), $A \to B$ (5), $A \to t$ (7)	20

22. Exemplo: Todos os Cortes de um Grafo



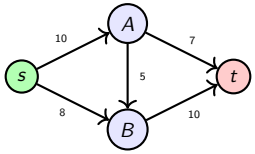
#	S	**Arestas** $S \to T$	**Capacidade**
1	$\{s\}$	$s \to A$ (10), $s \to B$ (8)	18
2	$\{s, A\}$	$s \to B$ (8), $A \to B$ (5), $A \to t$ (7)	20
3	$\{s, B\}$	$s \to A$ (10), $B \to t$ (10)	20

22. Exemplo: Todos os Cortes de um Grafo



#	S	**Arestas** $S \to T$	**Capacidade**
1	$\{s\}$	$s \to A$ (10), $s \to B$ (8)	18
2	$\{s, A\}$	$s \to B$ (8), $A \to B$ (5), $A \to t$ (7)	20
3	$\{s, B\}$	$s \to A$ (10), $B \to t$ (10)	20
4	$\{s, A, B\}$	$A \to t$ (7), $B \to t$ (10)	**17**

22. Exemplo: Todos os Cortes de um Grafo



#	S	**Arestas** $S \to T$	**Capacidade**
1	$\{s\}$	$s \to A$ (10), $s \to B$ (8)	18
2	$\{s, A\}$	$s \to B$ (8), $A \to B$ (5), $A \to t$ (7)	20
3	$\{s, B\}$	$s \to A$ (10), $B \to t$ (10)	20
4	$\{s, A, B\}$	$A \to t$ (7), $B \to t$ (10)	**17**

Corte Mínimo

O corte $\{s, A, B\}|\{t\}$ tem capacidade **17** — é o **corte mínimo!**
E o fluxo máximo também é 17. Coincidência? **Não!**

23. Teorema Max-Flow = Min-Cut

Teorema (Ford & Fulkerson, 1956)

Em qualquer rede de fluxo, o **valor do fluxo máximo** é igual à **capacidade do corte mínimo**.

23. Teorema Max-Flow = Min-Cut

Teorema (Ford & Fulkerson, 1956)

Em qualquer rede de fluxo, o **valor do fluxo máximo** é igual à **capacidade do corte mínimo**.

Parte 1: $|f| \leq c(S, T)$ para qualquer corte

Todo o fluxo de s a t **tem que atravessar** qualquer corte.
Como $f(u, v) \leq c(u, v)$, temos $|f| \leq c(S, T)$.
$\Rightarrow$ **Nenhum fluxo pode exceder nenhum corte.**

23. Teorema Max-Flow = Min-Cut

Teorema (Ford & Fulkerson, 1956)

Em qualquer rede de fluxo, o **valor do fluxo máximo** é igual à **capacidade do corte mínimo**.

Parte 1: $|f| \leq c(S, T)$ para qualquer corte

Todo o fluxo de s a t **tem que atravessar** qualquer corte.
Como $f(u, v) \leq c(u, v)$, temos $|f| \leq c(S, T)$.
$\Rightarrow$ **Nenhum fluxo pode exceder nenhum corte.**

Parte 2: Existe um corte onde $|f^*| = c(S^*, T^*)$

Quando Edmonds-Karp termina (BFS falha), defina:
$S^* = \{v : v \text{ é alcançável de } s \text{ no grafo residual}\}$ e $T^* = V \setminus S^*$.
Todas as arestas originais de S^* para T^* estão **saturadas** $\Rightarrow |f^*| = c(S^*, T^*)$.

23. Teorema Max-Flow = Min-Cut

Teorema (Ford & Fulkerson, 1956)

Em qualquer rede de fluxo, o **valor do fluxo máximo** é igual à **capacidade do corte mínimo**.

Parte 1: $|f| \leq c(S, T)$ para qualquer corte

Todo o fluxo de s a t **tem que atravessar** qualquer corte.
Como $f(u, v) \leq c(u, v)$, temos $|f| \leq c(S, T)$.
$\Rightarrow$ **Nenhum fluxo pode exceder nenhum corte.**

Parte 2: Existe um corte onde $|f^*| = c(S^*, T^*)$

Quando Edmonds-Karp termina (BFS falha), defina:
$S^* = \{v : v \text{ é alcançável de } s \text{ no grafo residual}\}$ e $T^* = V \setminus S^*$.
Todas as arestas originais de S^* para T^* estão **saturadas** $\Rightarrow |f^*| = c(S^*, T^*)$.

Consequência Prática

Fluxo máximo e corte mínimo são **duais** — resolver um resolve o outro!

24. Extraindo o Corte Mínimo na Prática

Após executar Edmonds-Karp e obter o fluxo máximo:

Algoritmo

24. Extraindo o Corte Mínimo na Prática

Após executar Edmonds-Karp e obter o fluxo máximo:

Algoritmo

1. Execute **BFS** no grafo residual a partir de s.

24. Extraindo o Corte Mínimo na Prática

Após executar Edmonds-Karp e obter o fluxo máximo:

Algoritmo

1. Execute **BFS** no grafo residual a partir de s.
2. $S = \{\text{vértices alcançáveis}\}$, $T = \{\text{vértices } \textbf{não} \text{ alcançáveis}\}$.

24. Extraindo o Corte Mínimo na Prática

Após executar Edmonds-Karp e obter o fluxo máximo:

Algoritmo

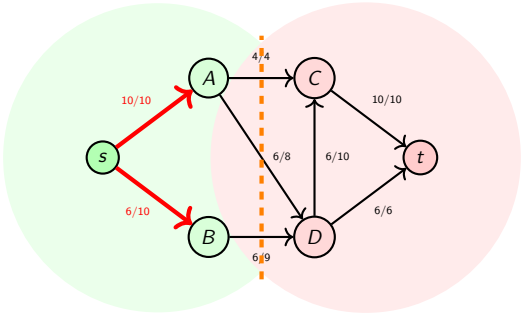
1. Execute **BFS** no grafo residual a partir de s.
2. $S = \{\text{vértices alcançáveis}\}$, $T = \{\text{vértices } \textbf{não} \text{ alcançáveis}\}$.
3. **Arestas do corte** = arestas originais de S para T (estão todas **saturadas**).

24. Extraindo o Corte Mínimo na Prática

Após executar Edmonds-Karp e obter o fluxo máximo:

Algoritmo

1. Execute **BFS** no grafo residual a partir de s.
2. $S = \{\text{vértices alcançáveis}\}$, $T = \{\text{vértices } \textbf{não} \text{ alcançáveis}\}$.
3. **Arestas do corte** = arestas originais de S para T (estão todas **saturadas**).

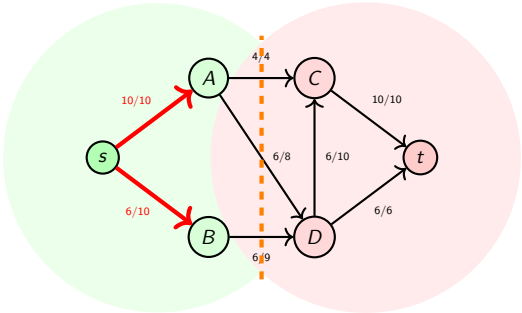


24. Extraindo o Corte Mínimo na Prática

Após executar Edmonds-Karp e obter o fluxo máximo:

Algoritmo

1. Execute **BFS** no grafo residual a partir de s.
2. $S = \{\text{vértices alcançáveis}\}$, $T = \{\text{vértices } \textbf{não} \text{ alcançáveis}\}$.
3. **Arestas do corte** = arestas originais de S para T (estão todas **saturadas**).



25. Dinic: Ideia Central

Melhoria sobre Edmonds-Karp

Edmonds-Karp envia **um** caminho aumentante por BFS. Dinic envia **múltiplos** caminhos por fase, usando DFS no *Level Graph*.

25. Dinic: Ideia Central

Melhoria sobre Edmonds-Karp

Edmonds-Karp envia **um** caminho aumentante por BFS. Dinic envia **múltiplos** caminhos por fase, usando DFS no *Level Graph*.

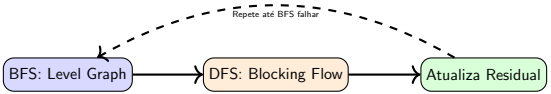
1. **BFS** a partir de $s \Rightarrow$ constrói o **Level Graph** (grafo de níveis).
2. **DFS** no Level Graph $\Rightarrow$ encontra o **Blocking Flow** (fluxo de bloqueio).
3. Repete até BFS não alcançar t.

25. Dinic: Ideia Central

Melhoria sobre Edmonds-Karp

Edmonds-Karp envia **um** caminho aumentante por BFS. Dinic envia **múltiplos** caminhos por fase, usando DFS no *Level Graph*.

1. **BFS** a partir de $s \Rightarrow$ constrói o **Level Graph** (grafo de níveis).
2. **DFS** no Level Graph $\Rightarrow$ encontra o **Blocking Flow** (fluxo de bloqueio).
3. Repete até BFS não alcançar t.

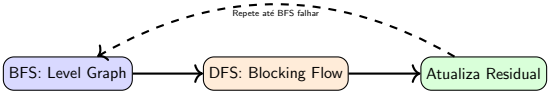


25. Dinic: Ideia Central

Melhoria sobre Edmonds-Karp

Edmonds-Karp envia **um** caminho aumentante por BFS. Dinic envia **múltiplos** caminhos por fase, usando DFS no *Level Graph*.

1. **BFS** a partir de $s \Rightarrow$ constrói o **Level Graph** (grafo de níveis).
2. **DFS** no Level Graph $\Rightarrow$ encontra o **Blocking Flow** (fluxo de bloqueio).
3. Repete até BFS não alcançar t.



Complexidade: $O(V^2 \cdot E)$

No máximo $O(V)$ fases (nível de t cresce a cada fase). Cada fase: $O(VE)$.

26. Level Graph: Construção

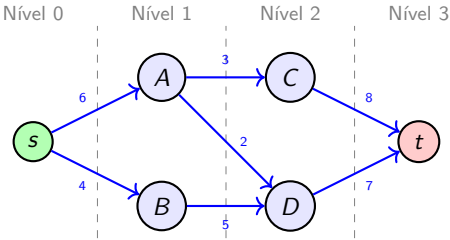
Definição

O **Level Graph** L_G contém apenas as arestas (u, v) do grafo residual onde $\text{dist}(v) = \text{dist}(u) + 1$ (arestas que "avançam" um nível em direção a t).

26. Level Graph: Construção

Definição

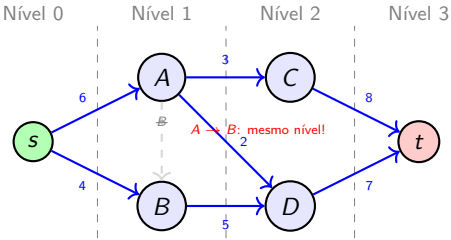
O **Level Graph** L_G contém apenas as arestas (u, v) do grafo residual onde $\text{dist}(v) = \text{dist}(u) + 1$ (arestas que "avançam" um nível em direção a t).



26. Level Graph: Construção

Definição

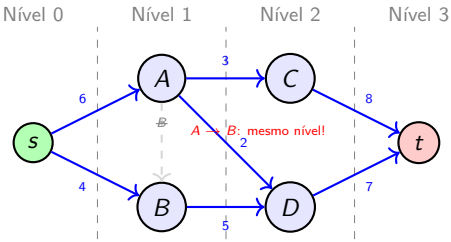
O **Level Graph** L_G contém apenas as arestas (u, v) do grafo residual onde $\text{dist}(v) = \text{dist}(u) + 1$ (arestas que "avançam" um nível em direção a t).



26. Level Graph: Construção

Definição

O **Level Graph** L_G contém apenas as arestas (u, v) do grafo residual onde $\text{dist}(v) = \text{dist}(u) + 1$ (arestas que "avançam" um nível em direção a t).



Regra

Arestas entre vértices do **mesmo nível** ou que "voltam" para níveis anteriores são **removidas**.

27. Blocking Flow: DFS com Poda

Definição

Um **Blocking Flow** (fluxo de bloqueio) satura **pelo menos uma aresta** em cada caminho de s a t no Level Graph.

27. Blocking Flow: DFS com Poda

Definição

Um **Blocking Flow** (fluxo de bloqueio) satura **pelo menos uma aresta** em cada caminho de s a t no Level Graph.

Como encontrar (DFS)

1. DFS de s buscando t.
2. Ao encontrar t: sature o caminho (gargalo).
3. **Dead-end** (beco sem saída): poda o vértice e **retrocede**.
4. Continue DFS para encontrar **mais caminhos**.
5. Para quando DFS não alcançar t.

27. Blocking Flow: DFS com Poda

Definição

Um **Blocking Flow** (fluxo de bloqueio) satura **pelo menos uma aresta** em cada caminho de s a t no Level Graph.

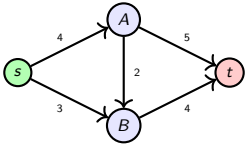
Como encontrar (DFS)

1. DFS de s buscando t.
2. Ao encontrar t: sature o caminho (gargalo).
3. **Dead-end** (beco sem saída): poda o vértice e **retrocede**.
4. Continue DFS para encontrar **mais caminhos**.
5. Para quando DFS não alcançar t.

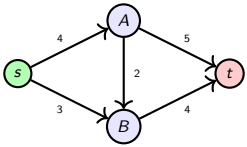
Vantagem sobre Edmonds-Karp

Em uma **única fase**, Dinic encontra **vários** caminhos aumentantes (todos de mesmo comprimento). Edmonds-Karp encontra apenas **um**.

28. Dinic: Trace Animado



28. Dinic: Trace Animado



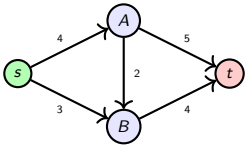
Fase 1: Level Graph (BFS)
Níveis: s=0, A=1, B=1, t=2.
$A \to B$: removida (mesmo nível).

Blocking Flow (DFS):

- $s \to A \to t$: gargalo $= \min(4, 5) = 4$
- $s \to B \to t$: gargalo $= \min(3, 4) = 3$

Fluxo da fase: $4 + 3 = 7$.

28. Dinic: Trace Animado



Fase 1: Level Graph (BFS)
Níveis: s=0, A=1, B=1, t=2.
$A \to B$: removida (mesmo nível).

Blocking Flow (DFS):
- $s \to A \to t$: gargalo $= \min(4, 5) = 4$
- $s \to B \to t$: gargalo $= \min(3, 4) = 3$

Fluxo da fase: $4 + 3 = 7$.

Fase 2: Novo Level Graph no Residual.
BFS de s: t inalcançável!

Fluxo máximo = 7

(Edmonds-Karp usaria 2 iterações de BFS.
Dinic fez em **1 fase** com 2 caminhos simultâneos.)

29. Comparação de Algoritmos

Algoritmo	Complexidade	Quando usar

29. Comparação de Algoritmos

Algoritmo	Complexidade	Quando usar
Ford-Fulkerson (DFS)	$O(E \cdot \lvert f^* \rvert)$	**Evitar! Pode ser exponencial.**

29. Comparação de Algoritmos

Algoritmo	**Complexidade**	**Quando usar**
Ford-Fulkerson (DFS)	$O(E \cdot \lvert f^* \rvert)$	**Evitar! Pode ser exponencial.**
Edmonds-Karp (BFS)	$O(V \cdot E^2)$	Implementação simples, caso geral.

29. Comparação de Algoritmos

Algoritmo	**Complexidade**	**Quando usar**
Ford-Fulkerson (DFS)	$O(E \cdot \lvert f^* \rvert)$	**Evitar! Pode ser exponencial.**
Edmonds-Karp (BFS)	$O(V \cdot E^2)$	Implementação simples, caso geral.
Dinic	$O(V^2 \cdot E)$	Grafos grandes, competições.

29. Comparação de Algoritmos

Algoritmo	**Complexidade**	**Quando usar**
Ford-Fulkerson (DFS)	$O(E \cdot \lvert f^* \rvert)$	**Evitar! Pode ser exponencial.**
Edmonds-Karp (BFS)	$O(V \cdot E^2)$	Implementação simples, caso geral.
Dinic	$O(V^2 \cdot E)$	Grafos grandes, competições.
Dinic (caps unitárias)	$O(E\sqrt{V})$	**Matching bipartido!**

29. Comparação de Algoritmos

Algoritmo	**Complexidade**	**Quando usar**
Ford-Fulkerson (DFS)	$O(E \cdot \lvert f^* \rvert)$	**Evitar! Pode ser exponencial.**
Edmonds-Karp (BFS)	$O(V \cdot E^2)$	Implementação simples, caso geral.
Dinic	$O(V^2 \cdot E)$	Grafos grandes, competições.
Dinic (caps unitárias)	$O(E\sqrt{V})$	**Matching bipartido!**
Push-Relabel	$O(V^2 \cdot E)$	Alternativa ao Dinic.
Push-Relabel (FIFO)	$O(V^3)$	Grafos muito densos.

29. Comparação de Algoritmos

Algoritmo	Complexidade	Quando usar
Ford-Fulkerson (DFS)	$O(E \cdot \lvert f^* \rvert)$	**Evitar! Pode ser exponencial.**
Edmonds-Karp (BFS)	$O(V \cdot E^2)$	Implementação simples, caso geral.
Dinic	$O(V^2 \cdot E)$	Grafos grandes, competições.
Dinic (caps unitárias)	$O(E\sqrt{V})$	**Matching bipartido!**
Push-Relabel	$O(V^2 \cdot E)$	Alternativa ao Dinic.
Push-Relabel (FIFO)	$O(V^3)$	Grafos muito densos.

Na Prática

- **Maratona/Competição:** Dinic é o mais usado (rápido, $\sim$40 linhas).
- **Didático:** Edmonds-Karp (mais fácil de entender e implementar).
- **Grafos densos muito grandes:** Push-Relabel FIFO.

30. Capacidade nos Vértices: Vertex Splitting

Às vezes cada vértice v tem uma capacidade máxima c_v de fluxo que pode **passar por ele**.

Técnica: Vertex Splitting

30. Capacidade nos Vértices: Vertex Splitting

Às vezes cada vértice v tem uma capacidade máxima c_v de fluxo que pode **passar por ele**.

Técnica: Vertex Splitting

1. Divida v em v_{in} e v_{out}.

30. Capacidade nos Vértices: Vertex Splitting

Às vezes cada vértice v tem uma capacidade máxima c_v de fluxo que pode **passar por ele**.

Técnica: Vertex Splitting

1. Divida v em v_{in} e v_{out}.
2. Todas as arestas que **entram** em $v \rightarrow$ entram em v_{in}.

30. Capacidade nos Vértices: Vertex Splitting

Às vezes cada vértice v tem uma capacidade máxima c_v de fluxo que pode **passar por ele**.

Técnica: Vertex Splitting

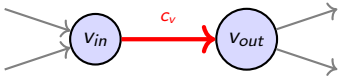
1. Divida v em v_{in} e v_{out}.
2. Todas as arestas que **entram** em $v \rightarrow$ entram em v_{in}.
3. Todas as arestas que **saem** de $v \rightarrow$ saem de v_{out}.

30. Capacidade nos Vértices: Vertex Splitting

Às vezes cada vértice v tem uma capacidade máxima c_v de fluxo que pode **passar por ele**.

Técnica: Vertex Splitting

1. Divida v em v_{in} e v_{out}.
2. Todas as arestas que **entram** em $v \rightarrow$ entram em v_{in}.
3. Todas as arestas que **saem** de $v \rightarrow$ saem de v_{out}.
4. Aresta interna: $v_{\text{in}} \rightarrow v_{\text{out}}$ com capacidade c_v.

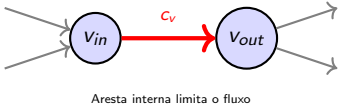


30. Capacidade nos Vértices: Vertex Splitting

Às vezes cada vértice v tem uma capacidade máxima c_v de fluxo que pode **passar por ele**.

Técnica: Vertex Splitting

1. Divida v em v_{in} e v_{out}.
2. Todas as arestas que **entram** em $v \rightarrow$ entram em v_{in}.
3. Todas as arestas que **saem** de $v \rightarrow$ saem de v_{out}.
4. Aresta interna: $v_{\text{in}} \rightarrow v_{\text{out}}$ com capacidade c_v.



Exemplo de Aplicação

"Cada roteador suporta no máximo K pacotes por segundo." $\Rightarrow$ Vertex splitting com $c_v = K$.

31. Matching Bipartido via Fluxo Máximo

Problema

Dado um grafo bipartido $G = (L \cup R, E)$, encontrar o maior número de arestas sem vértice repetido.

31. Matching Bipartido via Fluxo Máximo

Problema

Dado um grafo bipartido $G = (L \cup R, E)$, encontrar o maior número de arestas sem vértice repetido.

Modelagem

1. Super-source $S \rightarrow$ cada $l \in L$ (cap 1).
2. Cada $r \in R \rightarrow$ Super-sink T (cap 1).
3. Arestas $l \rightarrow r$ com cap 1 (se $(l, r) \in E$).
4. Fluxo máximo = tamanho do matching!

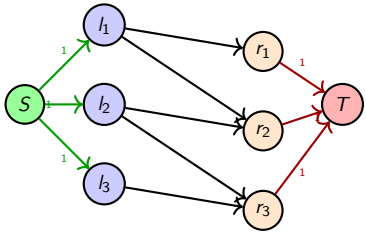
31. Matching Bipartido via Fluxo Máximo

Problema

Dado um grafo bipartido $G = (L \cup R, E)$, encontrar o maior número de arestas sem vértice repetido.

Modelagem

1. Super-source $S \to$ cada $l \in L$ (cap 1).
2. Cada $r \in R \to$ Super-sink T (cap 1).
3. Arestas $l \to r$ com cap 1 (se $(l, r) \in E$).
4. Fluxo máximo = tamanho do matching!



Fluxo máximo = 3 = matching perfeito!

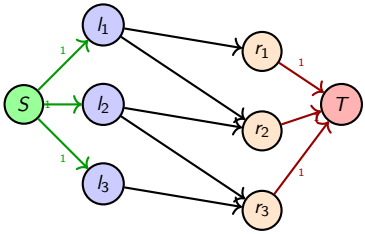
31. Matching Bipartido via Fluxo Máximo

Problema

Dado um grafo bipartido $G = (L \cup R, E)$, encontrar o maior número de arestas sem vértice repetido.

Modelagem

1. Super-source $S \to$ cada $l \in L$ (cap 1).
2. Cada $r \in R \to$ Super-sink T (cap 1).
3. Arestas $l \to r$ com cap 1 (se $(l, r) \in E$).
4. Fluxo máximo = tamanho do matching!



Fluxo máximo = 3 = matching perfeito!

Complexidade com Dinic

Grafos com caps unitárias: $O(E\sqrt{V})$ — mais rápido que Hopcroft-Karp genérico!

32. Circulação com Demandas (Lower Bounds)

Problema

Cada aresta (u, v) tem um fluxo **mínimo** $d(u, v)$ além da capacidade $c(u, v)$.
Queremos: $d(u, v) \leq f(u, v) \leq c(u, v)$.

32. Circulação com Demandas (Lower Bounds)

Problema

Cada aresta (u, v) tem um fluxo **mínimo** $d(u, v)$ além da capacidade $c(u, v)$.
Queremos: $d(u, v) \leq f(u, v) \leq c(u, v)$.

Transformação para Fluxo Padrão

1. Para cada aresta (u, v) com demanda d, capacidade c:
   - Nova capacidade: $c' = c - d$.
   - Ajuste: u "deve" d unidades, v "recebe" d.
2. Crie super-source S' e super-sink T'.
3. $S' \to v$ com cap = excesso de v (total recebido obrigatório).
4. $u \to T'$ com cap = déficit de u (total a enviar obrigatório).
5. Aresta $t \to s$ com cap ∞ (fecha o ciclo).
6. Se fluxo máximo satura todas as arestas de S': **circulação existe!**

33. Project Selection (Closure Problem)

Problema

Projetos com **lucro** (positivo ou negativo) e **dependências**. Selecionar subconjunto de lucro máximo respeitando: se projeto A depende de B, então B também deve ser selecionado.

33. Project Selection (Closure Problem)

Problema

Projetos com **lucro** (positivo ou negativo) e **dependências**. Selecionar subconjunto de lucro máximo respeitando: se projeto A depende de B, então B também deve ser selecionado.

Modelagem como Min-Cut

1. **Source** $s \rightarrow$ projetos com lucro > 0 (cap = lucro).
2. Projetos com lucro $< 0 \rightarrow$ **Sink** t (cap $= |\text{custo}|$).
3. Dependência $A \rightarrow B$: aresta com cap ∞.
4. **Lucro máximo** $= \sum$ lucros positivos $-$ Min-Cut.

33. Project Selection (Closure Problem)

Problema

Projetos com **lucro** (positivo ou negativo) e **dependências**. Selecionar subconjunto de lucro máximo respeitando: se projeto A depende de B, então B também deve ser selecionado.

Modelagem como Min-Cut

1. **Source** $s \to$ projetos com lucro > 0 (cap = lucro).
2. Projetos com lucro $< 0 \to$ **Sink** t (cap = $|$custo$|$).
3. Dependência $A \to B$: aresta com cap ∞.
4. **Lucro máximo** $= \sum$ lucros positivos $-$ Min-Cut.

Intuição

O min-cut "corta" o mínimo necessário: ou desistimos de um lucro (cortamos de s), ou pagamos um custo (cortamos para t). Dependências com ∞ garantem consistência.

34. Dicas de Modelagem e Padrões Comuns

Checklist de Modelagem

Checklist de Modelagem

1. **Identifique** fonte e sorvedouro.
   - Múltiplos? $\Rightarrow$ Super-source / super-sink.

Checklist de Modelagem

1. **Identifique** fonte e sorvedouro.
   - Múltiplos? $\Rightarrow$ Super-source / super-sink.
2. **Defina** capacidades nas arestas.

34. Dicas de Modelagem e Padrões Comuns

Checklist de Modelagem

1. **Identifique** fonte e sorvedouro.
   - Múltiplos? $\Rightarrow$ Super-source / super-sink.
2. **Defina** capacidades nas arestas.
3. Capacidade nos **vértices**? $\Rightarrow$ Vertex splitting.

34. Dicas de Modelagem e Padrões Comuns

Checklist de Modelagem

1. **Identifique** fonte e sorvedouro.
   - Múltiplos? $\Rightarrow$ Super-source / super-sink.
2. **Defina** capacidades nas arestas.
3. Capacidade nos **vértices**? $\Rightarrow$ Vertex splitting.
4. **Execute** Edmonds-Karp ou Dinic.

34. Dicas de Modelagem e Padrões Comuns

Checklist de Modelagem

1. **Identifique** fonte e sorvedouro.
   - Múltiplos? $\Rightarrow$ Super-source / super-sink.
2. **Defina** capacidades nas arestas.
3. Capacidade nos **vértices**? $\Rightarrow$ Vertex splitting.
4. **Execute** Edmonds-Karp ou Dinic.
5. Precisa do **corte**? $\Rightarrow$ BFS no residual.

34. Dicas de Modelagem e Padrões Comuns

Checklist de Modelagem

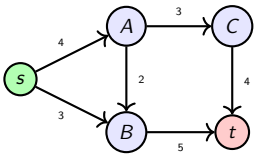
1. **Identifique** fonte e sorvedouro.
   - Múltiplos? $\Rightarrow$ Super-source / super-sink.
2. **Defina** capacidades nas arestas.
3. Capacidade nos **vértices**? $\Rightarrow$ Vertex splitting.
4. **Execute** Edmonds-Karp ou Dinic.
5. Precisa do **corte**? $\Rightarrow$ BFS no residual.

Padrões Frequentes em Competições

Padrão	**Modelagem**
Matching bipartido	Caps 1, super-s/super-t
Alocação com limites	Caps = limites de produção/demanda
Conectividade de arestas	Caps 1 em cada aresta, corte mínimo
Conectividade de vértices	Vertex splitting + caps 1 + corte mínimo
Seleção com dependências	Closure Problem (min-cut)

Exercício 1: Trace Manual Edmonds-Karp

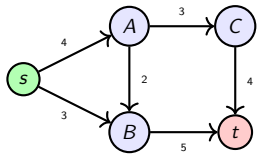
Simule Edmonds-Karp na rede abaixo. Mostre cada caminho aumentante e o fluxo acumulado.



1. Qual o fluxo máximo?
2. Quais os caminhos BFS em cada iteração?
3. Qual o gargalo de cada caminho?
4. Qual o corte mínimo?

Exercício 1: Trace Manual Edmonds-Karp

Simule Edmonds-Karp na rede abaixo. Mostre cada caminho aumentante e o fluxo acumulado.



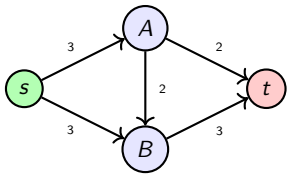
1. Qual o fluxo máximo?
2. Quais os caminhos BFS em cada iteração?
3. Qual o gargalo de cada caminho?
4. Qual o corte mínimo?

Resposta

Iter	Caminho (BFS)	Gargalo	$\|f\|$
1	$s \to A \to C \to t$	$\min(4,3,4) = 3$	3
2	$s \to B \to t$	$\min(3,5) = 3$	6
3	$s \to A \to B \to t$	$\min(1,2,2) = 1$	**7**

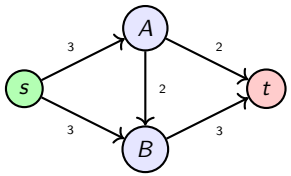
Corte mínimo: $S = \{s\}$, $T = \{A, B, C, t\}$. $c(S,T) = 4 + 3 = 7$ ✓

Exercício 2: A Importância da Aresta Reversa



1. Trace Edmonds-Karp. Em alguma iteração, uma aresta reversa é usada?
2. Qual seria o resultado **sem** arestas reversas?
3. Desenhe o grafo residual final.

Exercício 2: A Importância da Aresta Reversa



1. Trace Edmonds-Karp. Em alguma iteração, uma aresta reversa é usada?
2. Qual seria o resultado **sem** arestas reversas?
3. Desenhe o grafo residual final.

Resposta

Iter 1: $s \to A \to t$, garg=2, $|f| = 2$.
Iter 2: $s \to B \to t$, garg=3, $|f| = 5$.
Iter 3: $s \to A \to B \to t$, garg=1 (usa $A \to B$ de ida, mas sem reversa crucial aqui).
Porém se Iter 1 fosse $s \to A \to B \to t$ (garg=2), Iter 2: $s \to B \to \ldots$ precisa reversa $B \to A$!
Fluxo máximo = 5. Depende do caminho BFS escolhido.

Exercício 3: Modelagem — Fábricas e Lojas

Problema

3 fábricas (F_1: produz 5, F_2: produz 8, F_3: produz 4) e 4 lojas (L_1: demanda 3, L_2: demanda 5, L_3: demanda 4, L_4: demanda 6). Nem toda fábrica atende toda loja. Capacidades de transporte dadas. Qual a distribuição ótima?

1. Como modelar como rede de fluxo?
2. Onde ficam s e t?
3. O que significa "distribuição ótima"?

Exercício 3: Modelagem — Fábricas e Lojas

Problema

3 fábricas (F_1: produz 5, F_2: produz 8, F_3: produz 4) e 4 lojas (L_1: demanda 3, L_2: demanda 5, L_3: demanda 4, L_4: demanda 6). Nem toda fábrica atende toda loja. Capacidades de transporte dadas. Qual a distribuição ótima?

1. Como modelar como rede de fluxo?
2. Onde ficam s e t?
3. O que significa "distribuição ótima"?

Resposta

Super-Source $S \to$ cada fábrica (cap = produção de cada fábrica).
Cada loja $\to$ **Super-Sink** T (cap = demanda de cada loja).
Fábricas $\to$ lojas com caps de transporte.
Fluxo máximo = quantidade máxima distribuída. Se $|f^*| = \sum$ demandas, toda demanda é atendida!

Exercício 4: Matching Bipartido

3 alunos (A_1, A_2, A_3) e 3 projetos (P_1, P_2, P_3).

- A_1: gosta de P_1 e P_2.
- A_2: gosta de P_2 e P_3.
- A_3: gosta de P_1.

Cada aluno faz no máximo 1 projeto. Cada projeto é feito por no máximo 1 aluno.

1. Modele como rede de fluxo.
2. Qual o matching máximo?
3. Qual o corte mínimo correspondente?

Exercício 4: Matching Bipartido

3 alunos (A_1, A_2, A_3) e 3 projetos (P_1, P_2, P_3).

- A_1: gosta de P_1 e P_2.
- A_2: gosta de P_2 e P_3.
- A_3: gosta de P_1.

Cada aluno faz no máximo 1 projeto. Cada projeto é feito por no máximo 1 aluno.

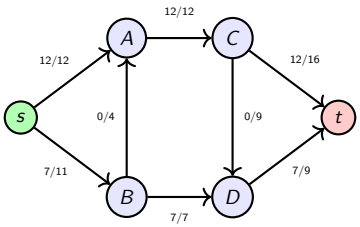
1. Modele como rede de fluxo.
2. Qual o matching máximo?
3. Qual o corte mínimo correspondente?

Resposta

$S \to A_1, A_2, A_3$ (cap 1). $P_1, P_2, P_3 \to T$ (cap 1). Arestas de preferência (cap 1).
Matching máximo = **3**: $A_1 \to P_2$, $A_2 \to P_3$, $A_3 \to P_1$.
(Se A_1 for atribuído a P_1 primeiro, a aresta reversa redireciona: A_3 "rouba" P_1 de A_1.)

Exercício 5: Encontrar o Corte Mínimo

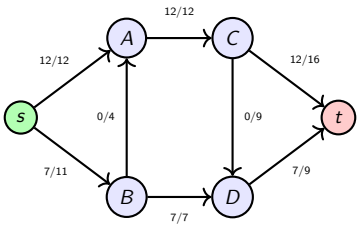
Após executar Edmonds-Karp no grafo abaixo e obter $|f^*| = 19$:



1. Construa o grafo residual.
2. Execute BFS de s no residual. Quais vértices são alcançáveis?
3. Quais são as arestas do corte mínimo?

Exercício 5: Encontrar o Corte Mínimo

Após executar Edmonds-Karp no grafo abaixo e obter $|f^*| = 19$:



1. Construa o grafo residual.
2. Execute BFS de s no residual. Quais vértices são alcançáveis?
3. Quais são as arestas do corte mínimo?

Resposta

Residual: s pode alcançar B (cap=4), B pode alcançar A (cap=4), B pode alcançar D?
$S = \{s, B\}$, $T = \{A, C, D, t\}$. Corte: $s \to A$ (12) + $B \to D$ (7) = **19** ✓

Resumo Final: Fluxo Máximo

Conceito	Complexidade	Nota
Ford-Fulkerson (DFS)	$O(E \cdot \lvert f^* \rvert)$	Evitar! Não é polinomial.
Edmonds-Karp	$O(VE^2)$	BFS. Fácil de implementar.
Dinic	$O(V^2E)$	Level Graph + Blocking Flow.
Min-Cut	Grátis	BFS no residual após o fluxo.
Vertex Splitting	—	Cap. nos vértices.
Matching Bipartido	$O(E\sqrt{V})$	Dinic com caps unitárias.

Resumo Final: Fluxo Máximo

Conceito	Complexidade	Nota
Ford-Fulkerson (DFS)	$O(E \cdot \lvert f^* \rvert)$	Evitar! Não é polinomial.
Edmonds-Karp	$O(VE^2)$	BFS. Fácil de implementar.
Dinic	$O(V^2E)$	Level Graph + Blocking Flow.
Min-Cut	Grátis	BFS no residual após o fluxo.
Vertex Splitting	—	Cap. nos vértices.
Matching Bipartido	$O(E\sqrt{V})$	Dinic com caps unitárias.

Checklist de Modelagem

1. Identifique fonte e sorvedouro (criar super-S/super-T se necessário).
2. Defina capacidades nas arestas.
3. Capacidade em vértices? $\Rightarrow$ Vertex splitting.
4. Execute Edmonds-Karp ou Dinic.
5. Precisa do corte? $\Rightarrow$ BFS no residual.

Resumo Final: Fluxo Máximo

Conceito	Complexidade	Nota
Ford-Fulkerson (DFS)	$O(E \cdot \lvert f^* \rvert)$	Evitar! Não é polinomial.
Edmonds-Karp	$O(VE^2)$	BFS. Fácil de implementar.
Dinic	$O(V^2E)$	Level Graph + Blocking Flow.
Min-Cut	Grátis	BFS no residual após o fluxo.
Vertex Splitting	—	Cap. nos vértices.
Matching Bipartido	$O(E\sqrt{V})$	Dinic com caps unitárias.

Checklist de Modelagem

1. Identifique fonte e sorvedouro (criar super-S/super-T se necessário).
2. Defina capacidades nas arestas.
3. Capacidade em vértices? $\Rightarrow$ Vertex splitting.
4. Execute Edmonds-Karp ou Dinic.
5. Precisa do corte? $\Rightarrow$ BFS no residual.

Dica de Maratona