---

## 📌 O que é uma Rede de Fluxo?

Uma **rede de fluxo** é um **grafo direcionado** onde as arestas têm **capacidades**. Isso significa que cada aresta tem um limite de **quanto fluxo ela pode carregar**.

Imagine que você tem um conjunto de canos conectando várias torneiras e pias em uma casa. A água pode fluir por esses canos, mas cada um tem um limite de quanta água pode passar por ele. Uma **rede de fluxo** funciona de maneira parecida!

**Componentes principais de uma rede de fluxo:**

1. **Vértices**: São os pontos da rede (como torneiras e pias — os pontos onde a água pode chegar ou sair).

2. **Arestas**: São as conexões entre os pontos, representando os canais por onde o fluxo passa (como os canos — os caminhos por onde a água pode passar).

3. **Capacidades**: Cada aresta tem uma capacidade que limita a quantidade de fluxo que pode passar por ela (cada cano tem um limite de água que pode transportar).

4. **Fonte (s)**: É o ponto de onde o fluxo começa (como uma caixa d'água ou uma grande torneira).

5. **Sorvedouro (t)**: É o ponto onde o fluxo termina, ou seja, o "destino" (como um ralo ou um reservatório que recebe a água).

Em um **grafo direcionado**, se você tem uma aresta de u para v, **não pode haver uma aresta** de v para u na direção contrária (a menos que você defina um fluxo reverso de alguma maneira, mas isso é uma complexidade adicional que veremos mais à frente).

Imagine que a água só pode correr em um sentido dentro dos canos. Isso significa que se há um cano levando água da torneira A para a pia B, não dá para a água simplesmente voltar de B para A sem um outro caminho especial.

💡 **Importante**:
🚫**Sem laços**: Ou seja, **não há aresta que conecta um vértice a ele mesmo**. (não existe um cano que liga um ponto a ele mesmo.

🔗**Conectividade**: A rede de fluxo é conectada, o que significa que **existe um caminho de fluxo** entre a fonte s e o sorvedouro t para **todos os vértices** (sempre tem um jeito de a água sair da fonte e chegar ao destino).

---

---

## 📌 Fluxo em uma Rede

Agora, vamos ver o que é um **fluxo** em uma rede de fluxo.

- **Fluxo**: O fluxo é basicamente a quantidade de "recursos" (por exemplo, água, dados) que você pode enviar da **fonte** para o **sorvedouro** respeitando as **capacidades das arestas**.

- Vamos pensar na água que realmente está passando pelos canos — isso é chamado de **fluxo**. Mas tem algumas regras que ele precisa seguir:

**Propriedades de um Fluxo:**

1️⃣**Restrição de capacidade** 🚧:
Para cada aresta (u,v), a quantidade de fluxo f(u,v) que passa pela aresta deve ser **menor ou igual à capacidade** c(u,v) desta aresta. Ou seja: $0 \leq f(u,v) \leq c(u,v)$. Isso significa que você não pode enviar mais fluxo do que a capacidade da aresta permite.

Em outras palavras, a quantidade de água que passa por um cano não pode ser maior do que o que ele aguenta. Ou seja, se um cano suporta no máximo 5 litros por segundo, não dá para tentar empurrar 10 litros nele!

2️⃣**Conservação de Fluxo** 🔄
Para **todo vértice u** que não seja a fonte **nem o sorvedouro**, a quantidade de fluxo que entra no vértice deve ser **igual** à quantidade de fluxo que sai dele. Isso é o que chamamos de **conservação de fluxo**.

  - Para a fonte s, o fluxo só pode **sair** dela.
  - Para o sorvedouro t, o fluxo só pode **entrar** nele.

  o **fluxo entrando** em u deve ser igual ao **fluxo saindo** de u.

  Em outras palavras, se a água entra por um cano em um ponto, ela também tem que sair de lá por outro cano. Exceto na fonte (onde a água só sai) e no sorvedouro (onde a água só entra).

3️⃣**Fluxo Nulo Reverso** ↩️
Caso você tenha fluxo indo de u para v, você não pode ter fluxo indo de v para u ao mesmo tempo, a não ser que você trate isso como um fluxo reverso (ou seja, o fluxo é enviado de v para u, e você diminui o fluxo de u para v).

Em outras palavras, se a água está fluindo em um sentido, ela não pode simplesmente voltar pelo mesmo cano ao mesmo tempo. Se precisar devolver um pouco, seria como "abrir outro caminho" para a água voltar.

---

---

## 📌 Exemplo Simplificado de Rede de Fluxo

Imagina que temos uma rede simples com 4 vértices e algumas arestas:

```
Unset
Fonte (s) → A → B → Sorvedouro (t)
             ↓           ↑
            C    →      D
```

Com capacidades nas arestas:

- s→A com capacidade 10
- A→B com capacidade 5
- B→t com capacidade 10
- A→C com capacidade 15
- C→D com capacidade 5
- D→t com capacidade 10

Aqui está o que acontece:

1. O fluxo começa da **fonte s**.
2. O fluxo vai para A, depois para B, e finalmente para o **sorvedouro t**.
3. O fluxo pode seguir **outros caminhos** também, como A→C→D→t, mas sempre respeitando as capacidades.

A quantidade de fluxo que passa de s para t é o **fluxo máximo** dessa rede, e você precisa garantir que respeite as restrições de **capacidade** e **conservação**.

---

## 📌 Resumo Final

- **Rede de fluxo**: Grafo dirigido com capacidades nas arestas e dois vértices especiais: **fonte** e **sorvedouro**.
- **Fluxo**: A quantidade de "recurso" que pode ser enviado da fonte para o sorvedouro respeitando as capacidades das arestas e a conservação do fluxo.

Esse conceito de fluxo em grafos é central em vários problemas de otimização, como no transporte de recursos, redes de comunicação e distribuição de energia.

---

---

---

---

## 📌 O que são Arestas Antiparalelas?

Arestas **antiparalelas** são um par de **arestas direcionadas** entre dois vértices, onde uma vai de v1 para v2 e a outra vai de v2 para v1. Em outras palavras, temos duas arestas opostas que se conectam aos mesmos vértices, mas em direções opostas.

Imagine que você tem duas cidades, **Cidade A** e **Cidade B**, e duas estradas entre elas:

🚗 **Estrada 1**: Liga a **Cidade A** até a **Cidade B**.
🚙 **Estrada 2**: Liga a **Cidade B** até a **Cidade A**.

Se existe uma estrada indo de A para B e outra voltando de B para A, dizemos que essas estradas são **antiparalelas**.

Exemplo:

- Aresta 1: (v1,v2)
- Aresta 2: (v2,v1)

Ou seja, se existe uma aresta de v1 para v2, também existe uma aresta de v2 para v1.

---

## 📌 O Problema com Arestas Antiparalelas

Na modelagem de **fluxo em redes**, há uma **restrição importante**: **não podemos ter arestas antiparalelas**. Isso ocorre porque em muitas abordagens de algoritmos de fluxo, como o **Ford-Fulkerson** ou o **Edmonds-Karp**, tratamos o fluxo em uma direção e, para permitir o fluxo reverso, utilizamos capacidades residuais, que não são facilmente modeladas quando há arestas antiparalelas.

Imagine que estamos tentando controlar o tráfego dessas estradas usando um sistema inteligente. O problema é que muitos sistemas de controle só conseguem entender o tráfego indo em **uma única direção** de cada vez.

Se tivermos duas estradas entre os mesmos lugares, uma indo e outra voltando, isso pode confundir os cálculos. Alguns algoritmos que organizam esse tráfego, como **Ford-Fulkerson** e **Edmonds-Karp**, não funcionam bem com estradas antiparalelas porque eles precisam controlar o fluxo de forma mais simples.

Por isso, precisamos **transformar a rede** de forma que **não haja arestas antiparalelas**, garantindo que a rede esteja em um formato adequado para resolver o problema de fluxo.

---

---

## 📌 Como Resolver o Problema de Arestas Antiparalelas?

A solução para **remover as arestas antiparalelas** da rede é **transformá-la** de modo a representar as mesmas capacidades de fluxo, mas sem a presença de arestas bidirecionais. Para resolver isso, podemos fazer uma pequena reforma no nosso sistema de estradas:

**Passos para Transformar a Rede:**

🛠️ **Passo 1: Identificar as estradas antiparalelas**: O primeiro passo é identificar as arestas que formam um par antiparalelo, ou seja, as arestas (v1,v2) e (v2,v1). Ou seja, encontramos todas as duplas de estradas que ligam os mesmos pontos em direções opostas.

🛠️ **Passo 2: Criar um novo ponto intermediário (adicionar um novo vértice v′)**: Para a aresta (v1,v2), criamos um novo vértice v′ e dividimos a aresta original em **duas arestas.** É como se colocássemos um novo ponto no meio do caminho para dividir uma das estradas em duas partes. Por exemplo, em vez de ter uma estrada direta de Cidade A para Cidade B, criamos um novo ponto P no meio:

- 🚗 **Nova Estrada 1**: De **Cidade A** para **P**.
  A aresta (v1,v′) com capacidade igual à da aresta (v1,v2).

- 🚙 **Nova Estrada 2**: De **P** para **Cidade B**.
  A aresta (v′,v2) com capacidade igual à da aresta (v1,v2).

🛠️ **Passo 3: Remover a estrada que estava na direção oposta** (**eliminar a aresta antiparalela)**: A aresta (v2,v1) original é removida da rede, pois ela foi "substituída" pelas novas arestas que mantêm o mesmo valor de capacidade. Agora que reorganizamos o caminho, não precisamos mais da estrada direta de **Cidade B** para **Cidade A**. O fluxo pode ser controlado de maneira mais clara.

---

---

## 📌 Por que a Rede Resultante é Equivalente à Original?

A rede resultante, após a transformação, **é equivalente** à rede original no sentido de que ela mantém as **mesmas capacidades de fluxo** entre os vértices. Ou seja, **não alteramos o problema de fluxo**, apenas o reestruturamos de uma forma que facilita a aplicação de algoritmos de fluxo.

Mesmo depois dessa mudança, **nada foi perdido!** O tráfego continua fluindo da mesma forma entre as cidades, só que agora de uma maneira que o sistema entende melhor.

Isso significa que ainda podemos transportar a mesma quantidade de carros (ou água, eletricidade, dados...), mas agora sem confusão! 🚀

Vejamos por que isso funciona:

- O que antes era um único caminho de **A para B**, agora passa por **A → P → B**, mas ainda suporta a mesma quantidade de carros.

  O fluxo que passaria de v1 para v2 agora passa de v1 para v′ e de v′ para v2, respeitando a capacidade original da aresta (v1,v2).

- O fluxo que antes ia de **B para A** continua existindo, só que agora é reorganizado em um formato mais fácil de entender.

  O fluxo que passaria de v2 para v1 continua existindo da mesma forma, só que a rede agora representa isso com as arestas **não antiparalelas**.

Portanto, **não estamos alterando** a quantidade de fluxo que pode passar entre v1 e v2, nem a direção do fluxo, apenas reorganizamos a rede para eliminar as antiparalelas.

---

## 📌 Exemplo

Suponha que temos o seguinte par de arestas antiparalelas em um grafo:

- (v1,v2) com capacidade 10
- (v2,v1) com capacidade 10

A transformação seria a seguinte:

1. Adicionamos um novo vértice v'.
2. Substituímos a aresta (v1,v2) por duas arestas:
   - (v1,v′) com capacidade 10
   - (v′,v2) com capacidade 10
3. Removemos a aresta (v2,v1).

Agora, a rede resultante não contém mais arestas antiparalelas, mas ainda mantém as mesmas capacidades de fluxo entre v1 e v2.

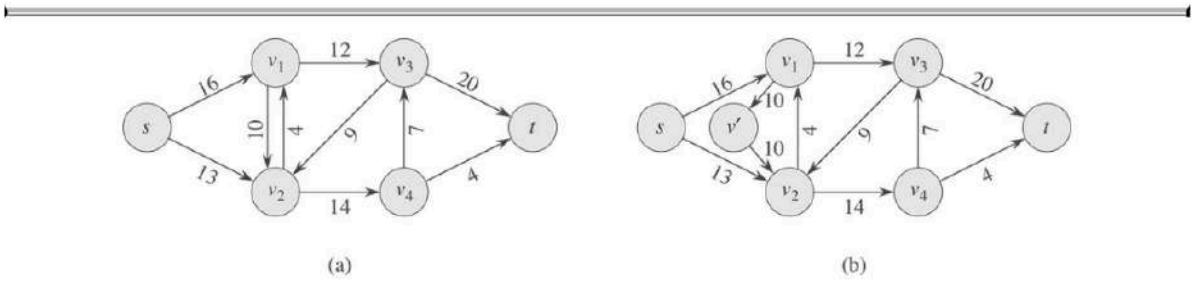


**Figura 26.2** Conversão de uma rede com arestas antiparalelas em uma rede equivalente sem nenhuma aresta antiparalela. **(a)** Rede de fluxo que contém as arestas $(v_1, v_2)$ e $(v_2, v_1)$. **(b)** Uma rede equivalente sem nenhuma aresta antiparalela. Adicionamos um novo vértice $v'$ e substituímos a aresta $(v_1, v_2)$ pelo par de arestas $(v_1, v')$ e $(v', v_2)$, ambas com a mesma capacidade de $(v_1, v_2)$.

---

### 📌 Fluxo Máximo com Várias Fontes e Vários Sorvedouros

O conceito de **fluxo máximo em redes com várias fontes e vários sorvedouros** é uma extensão do problema clássico de fluxo máximo em redes com apenas uma fonte e um sorvedouro. Neste cenário, temos múltiplos pontos de origem (fontes) e múltiplos destinos (sorvedouros), e o desafio é determinar o fluxo máximo que pode ser enviado de todas as fontes para todos os sorvedouros.

Imagine que você tem várias fábricas que produzem brinquedos e vários armazéns que precisam recebê-los. O problema é descobrir qual a **quantidade máxima de brinquedos** que pode ser transportada das fábricas para os armazéns, respeitando os limites de cada estrada.

Felizmente, podemos **reduzir** esse problema a um problema de fluxo máximo comum com **uma única fonte** e **um único sorvedouro**, facilitando a aplicação de algoritmos padrão, como o **algoritmo de Ford-Fulkerson**.

### 📌 O Problema

Temos:
🏭 **Fábricas (Fontes)**: São os pontos de onde os brinquedos saem - **m fábricas**: s1,s2,…,, sm.

📦 **Armazéns (Sorvedouros)**: São os lugares para onde os brinquedos devem ir - **n armazéns**: t1,t2,…,tn

🛣️ **Caminhos (Arestas)**: São as estradas que ligam as fábricas aos armazéns, cada uma com uma capacidade máxima de transporte.

A rede é composta por essas fontes e sorvedouros, além das arestas entre eles. A tarefa é calcular o fluxo máximo do conjunto **de fábricas** para o **conjunto de armazéns**, respeitando as capacidades de cada aresta.

Como há **muitas fábricas e muitos armazéns**, fica complicado calcular o fluxo máximo diretamente. Mas existe um truque inteligente para simplificar o problema!

Essa rede pode ser mais difícil de lidar diretamente, já que temos **m fontes e n sorvedouros**.

## 📌 Redução para um Problema de Fluxo Máximo Comum - Transformando em um Problema Mais Simples

Felizmente, podemos **transformar** essa rede em uma rede equivalente, mas com apenas **uma fonte** e **um sorvedouro**, Para facilitar, criamos dois novos pontos mágicos, utilizando um truque simples:

1. **Adicionar uma superfonte s**:
    - Criamos uma **nova fonte**, chamada de **superfonte** s, que **conecta** à cada uma das **fontes s1,s2,…,sm** (um novo ponto que está ligado a todas as fábricas e pode fornecer brinquedos sem limite)
      .
    - Para cada aresta (s,si) que conecta s à fonte si, definimos uma **capacidade infinita** c(s,si)=∞. Isso significa que a superfonte pode fornecer fluxo ilimitado para as fontes s1,s2,…,sm, já que estamos interessados no **fluxo máximo global**.

2. **Adicionar um supersorvedouro t**:
    - Criamos um **novo sorvedouro**, chamado de **supersorvedouro** t, que **recebe fluxo** de cada um dos **sorvedouros t1,t2,…,tn** (um novo ponto que recebe brinquedos de todos os armazéns e pode armazenar quantos forem necessários).

    - Para cada aresta (ti,t) que conecta ti ao supersorvedouro t, definimos uma **capacidade infinita** c(ti,t)=∞. Isso implica que o supersorvedouro pode "absorver" fluxo ilimitado de todos os sorvedouros t1,t2,…,tn.

Agora, a única coisa que precisamos calcular é o fluxo máximo que vai de **S até T**. Como essa nova rede representa exatamente o mesmo problema de antes, podemos usar **algoritmos já conhecidos**, como o de **Ford-Fulkerson**, para encontrar a resposta de forma muito mais fácil! 🚀

## 📌 Como Isso Funciona?

- Agora, você tem **uma única fonte s** que fornece fluxo para as fontes s1,s2,…,sm, e um **único sorvedouro t** que recebe fluxo dos sorvedouros t1,t2,…,tn.
- O **fluxo máximo** entre a superfonte s e o supersorvedouro t nesta nova rede será **equivalente** ao fluxo máximo que pode ser enviado de todas as fontes s1,s2,…,sm para todos os sorvedouros t1,t2,…,tn na rede original.
- O algoritmo de **fluxo máximo comum** pode agora ser aplicado à rede reduzida com uma única fonte e um único sorvedouro.

## 📌 Visualização do Processo

**Antes (com várias fontes e vários sorvedouros):**

```
Unset
Fonte(s)   →    Fábricas (s1, s2, ..., sm)
          ↖        |         |   |
           ↘     Armazéns (t1, t2, ..., tn)
                   |   |    |   |
                  Sorvedouros
```

**Depois (com superfonte e supersorvedouro):**

```
Unset
Superfonte (s) → Fábricas (s1, s2, ..., sm)
              ↖     |         |   |
               ↘    ↓       Armazéns (t1, t2, ..., tn)
                           ↓
                    Supersorvedouro (t)
```

## 📌 Prova Formal da Equivalência

Para provar formalmente que os dois problemas são **equivalentes** (ou seja, que o fluxo máximo calculado na rede transformada é o mesmo que o fluxo máximo na rede original), podemos fazer o seguinte:

- **Fluxo da superfonte**: O fluxo da superfonte s é distribuído para as várias fontes s1,s2,...,sm, e a quantidade de fluxo enviada a cada uma delas é limitada pelas suas respectivas capacidades.

- **Fluxo do supersorvedouro**: O fluxo que chega ao supersorvedouro t é oriundo de todos os sorvedouros t1,t2,...,tn. O fluxo máximo que pode ser absorvido por t é o fluxo máximo total da rede original.

Assim, a **quantidade total de fluxo** da superfonte para o supersorvedouro na rede transformada é exatamente a **quantidade total de fluxo** que seria enviada pelas várias fontes para os vários sorvedouros na rede original. Portanto, o problema de fluxo máximo é **preservado** pela transformação.

---

---

---

## 📌 O que é uma rede residual?

Uma **rede residual** é uma maneira de representar as possibilidades de aumentar ou diminuir o fluxo em uma rede de fluxo. Em outras palavras, ela mostra onde e quanto fluxo pode ser **adicionado ou removido** de uma rede existente.

Quando falamos de fluxo em redes, como as **redes de fluxo** que vimos anteriormente, você tem um **fluxo inicial** em cada aresta, e a **capacidade** máxima dessas arestas que limita o fluxo. A **rede residual** ajuda a visualizar as mudanças possíveis no fluxo.

A rede residual é como um mapa especial que mostra **quanta água ainda pode passar por cada cano** e, além disso, nos diz se **podemos mandar a água de volta**! Se descobrirmos que mandamos água para o lugar errado ou precisamos reorganizar, podemos devolver parte da água.

Por exemplo, se enviamos **7 litros** por um cano de **10 litros**, podemos ainda mandar **3 litros** a mais. Mas, e se quisermos diminuir o fluxo? Nesse caso, adicionamos um **cano invisível no sentido oposto**, que permite "devolver" até **7 litros**! Isso ajuda a corrigir o fluxo caso tenhamos tomado uma decisão ruim.

## Como ela funciona?

1. **Arestas com capacidade residual positiva**: Quando você tem uma aresta (u,v) com fluxo f(u,v) já em andamento, a **capacidade residual** cf(u,v) é dada por:

   $$cf(u,v) = c(u,v) - f(u,v)$$

   Ou seja, é o quanto ainda pode ser enviado de u para v sem ultrapassar a capacidade máxima da aresta. Se o valor for positivo, significa que ainda há capacidade para adicionar mais fluxo nesta direção.

2. **Arestas inversas**: Para permitir que o fluxo seja **retornado** ou **diminuído**, também é adicionada uma aresta **invertida** (v,u) na rede residual. Isso representa a capacidade de **cancelar** parte do fluxo já enviado pela aresta (u, v). A **capacidade residual** da aresta invertida será:

   $cf(v,u)=f(u,v)$

   Ou seja, a capacidade da aresta invertida é igual ao fluxo atual f(u,v), pois podemos **reduzir** o fluxo de u para v e de v para u.

## Exemplo de como a rede residual funciona:

Imagina que você tem uma aresta (u,v) com capacidade $c(u,v)=16$ e já tem um fluxo $f(u,v)=11$. Isso significa que:

- A capacidade residual para (u,v) é: $cf(u,v)=c(u,v)-f(u,v)=16-11=5$. Ou seja, ainda podemos enviar **5 unidades de fluxo** da origem u para o destino v.
- Além disso, a aresta inversa (v,u) terá a capacidade residual: $cf(v,u)=f(u,v)=11$. Isso permite que até **11 unidades de fluxo** possam ser enviadas de volta de v para u, ou seja, podemos "cancelar" até 11 unidades de fluxo.

## Lema 26.1

Imagine que temos um sistema de **tubos de água** (a rede de fluxo) e que a água já está fluindo com certa intensidade f. Agora, olhamos para o **mapa da rede residual**, que nos mostra onde ainda há espaço para aumentar o fluxo ou onde podemos devolver fluxo se for necessário.

Se conseguimos empurrar mais água usando esse **mapa da rede residual** (ou seja, aplicamos um fluxo $f'$ nela), o fluxo resultante $f \uparrow f'$ na rede original **continua respeitando todas as regras**:

1. **Nenhum cano recebe mais água do que sua capacidade permite.**
2. **A água que entra e sai de cada ponto (exceto na fonte e no destino) continua equilibrada.**

Ou seja, $f \uparrow f'$ **é um fluxo válido** e seu valor total é simplesmente a soma dos fluxos originais.

## O que a prova está garantindo?

A prova segue três etapas principais:

1. **Respeito à capacidade dos tubos:** Se estamos adicionando fluxo, ele só pode crescer dentro dos limites que a rede residual nos permite.
2. **Conservação do fluxo:** O novo fluxo f ↑ f′ mantém a mesma lógica de equilíbrio do fluxo original, pois estamos apenas redistribuindo água de maneira válida.
3. **Cálculo do novo fluxo total:** Ao somarmos todas as entradas e saídas na fonte s, provamos matematicamente que o novo fluxo ∣f ↑ f′∣ é simplesmente ∣ f ∣+∣ f′ ∣, ou seja, o fluxo original somado ao fluxo extra que conseguimos empurrar na rede residual.

## Por que isso é útil?

Esse lema nos diz que podemos **sempre aumentar um fluxo de forma segura** usando a rede residual, o que é a base de algoritmos como **Ford-Fulkerson** para encontrar o fluxo máximo. Ele garante que, sempre que encontrarmos um caminho com capacidade disponível na rede residual, podemos aumentar o fluxo total sem quebrar nenhuma regra!

---

## 📌 O que é um caminho aumentador?

Os **caminhos aumentadores** ajudam a encontrar novas formas de aumentar o fluxo em uma rede.

🔹 **Definição:** Um **caminho aumentador** é um caminho de sss (origem) até ttt (destino) dentro da **rede residual** Gf
🔹 **Por que ele é útil?** Ele mostra **por onde ainda podemos aumentar o fluxo** sem ultrapassar as capacidades das arestas.

Na **rede residual**, as arestas têm capacidades residuais cf(u,v), que representam a quantidade de fluxo adicional que pode ser enviado através daquela aresta sem ultrapassar sua capacidade máxima. Um **caminho aumentador** é um caminho simples de **s** até **t** na rede residual, onde todas as arestas do caminho têm capacidade residual maior que 0. Isso significa que é possível **aumentar** o fluxo ao longo desse caminho.

**Como encontramos um caminho aumentador?**

1️⃣ Pegamos a rede residual Gf (onde as arestas mostram os fluxos ainda possíveis).
2️⃣ Encontramos um caminho de sss até ttt que só usa arestas com **capacidade residual positiva**.
3️⃣ O fluxo pode ser aumentado **até o menor valor de capacidade residual** encontrado nesse caminho.

## Capacidade residual de um caminho aumentador:

A **capacidade residual de um caminho aumentador** p, denotada cf(p), é o valor máximo que podemos aumentar o fluxo ao longo de todas as arestas desse caminho. É dada por:

cf(p) = min {cf(u,v) para todas as arestas (u,v) no caminho p}

Ou seja, a **capacidade residual de um caminho aumentador** é limitada pela **menor capacidade residual** ao longo das arestas desse caminho. Isso é importante porque, se uma aresta tem capacidade residual pequena, ela vai limitar quanto fluxo pode ser aumentado através desse caminho.

## Exemplo:

Imaginando que temos um caminho p entre s e t na rede residual, e as capacidades residuais das arestas ao longo desse caminho são:

- cf(u1,u2)=5
- cf(u2,u3)=4
- cf(u3,u4)=7

A capacidade residual do caminho p será cf(p)=min(5,4,7)=4. Ou seja, podemos aumentar o fluxo ao longo desse caminho em até **4 unidades** sem violar a capacidade das arestas.

## Lema 26.2:

O **Lema 26.2** diz que, se temos um **fluxo** f em uma rede G, e encontramos um caminho aumentador pp na rede residual Gf, podemos aumentar o fluxo f ao longo desse caminho. Isso resulta em um novo fluxo f′ que tem o valor total aumentado em cf(p), ou seja, o fluxo f foi aumentado em cf(p).

Formalmente, temos que o **fluxo aumentado** é um fluxo válido, e seu valor será:

$|f \uparrow fp| = |f| + cf(p)$

Ou seja, o valor do fluxo aumentou de f para f+cf(p), o que aproxima o fluxo do valor máximo possível (se ainda houver mais caminhos aumentadores disponíveis).

## Corolário 26.3:

O **Corolário 26.3** segue diretamente do Lema 26.2 e afirma que, ao aumentar o fluxo f adicionando o fluxo fp, obtemos um novo fluxo com valor maior. Em outras palavras, a cada vez que encontramos um caminho aumentador, podemos aumentar o fluxo, e isso nos aproxima da solução de fluxo máximo.

Se antes tínhamos um fluxo com valor $|f|$, após aumentar esse fluxo ao longo de um caminho aumentador, o novo valor do fluxo será:

$|f \uparrow fp| = |f| + cf(p) > |f|$

Isso significa que o valor do fluxo foi **aumentado** e, portanto, está mais próximo de atingir o valor máximo possível de fluxo na rede.

Resumindo:
🔹 Se encontramos um caminho aumentador p, podemos usá-lo para aumentar o fluxo em G.
🔹 O novo fluxo resultante será **maior do que o fluxo original**.

## Conclusão:

- **Caminhos aumentadores** são fundamentais para o processo de **aumento do fluxo** em redes de fluxo.
- A **capacidade residual** do caminho aumenta o fluxo sem violar as restrições da rede original.
- Cada vez que encontramos um caminho aumentador e aumentamos o fluxo, o valor do fluxo total na rede cresce e nos aproxima do valor máximo.

Esse processo é essencial para os **algoritmos de fluxo máximo**, como o **Edmonds-Karp**, que buscam aumentar o fluxo até que não seja mais possível encontrar caminhos aumentadores na rede residual.

---

## 📌 Cortes em Redes de Fluxo

Quando trabalhamos com **fluxo máximo**, um conceito importante é o de **corte da rede**.

🛑 **O que é um corte?**
Um **corte (S, T)** divide os vértices V da rede em dois grupos:

- S contém a **fonte** s,
- T contém o **sorvedouro** t,
- Não há interseção entre S e T, ou seja, $S \cap T = \varnothing$.

🔹 **Capacidade de um corte**: É a soma das capacidades das arestas que vão de S para T.
🔹 **Fluxo líquido do corte**: É o fluxo que passa por essas arestas, subtraindo qualquer fluxo no sentido inverso.

## Qual a relação entre cortes e fluxo máximo?

O **Lema 26.4** mostra que, para qualquer corte na rede, o fluxo líquido pelo corte é sempre igual ao fluxo total da rede.

Já o **Corolário 26.5** afirma que o valor de qualquer fluxo não pode ser maior que a capacidade de um corte.

🔹 Isso significa que **o fluxo máximo da rede está limitado pelo menor corte possível**.

💡 Ou seja, se encontrarmos um corte com capacidade mínima, sabemos que ele impõe um limite superior para o fluxo máximo!

## Teorema do Fluxo Máximo / Corte Mínimo (Teorema 26.6)

Esse teorema é fundamental porque **liga os conceitos de fluxo e corte**. Ele diz que as três afirmações abaixo são equivalentes:

1️⃣ O fluxo f é **máximo**.
2️⃣ **Não existe caminho aumentador** na rede residual.
3️⃣ O fluxo $|f|$ é **igual à capacidade de algum corte mínimo**.

📌 **O que isso significa na prática?**

- Se não houver mais **caminhos aumentadores**, então **o fluxo atual já é máximo**.
- O **fluxo máximo** é exatamente **igual** à **capacidade do menor corte possível**.
- O **algoritmo de Ford-Fulkerson** funciona porque, ao aumentar o fluxo sempre que possível, ele eventualmente atinge esse limite.

## Conclusão 🚀

Esse teorema prova que o **fluxo máximo** de uma rede é **sempre igual** à capacidade do **corte mínimo**.
Ou seja, se quisermos saber qual é o fluxo máximo, basta encontrar o **menor corte possível** na rede!

---

---

## Como funciona em um algoritmo de fluxo máximo?

Quando usamos **algoritmos de fluxo máximo** (como o **algoritmo de Edmonds-Karp** ou o **Ford-Fulkerson**), a rede residual é uma parte importante. Esses algoritmos buscam **aumentar o fluxo** de uma rede, encontrando caminhos adicionais onde é possível adicionar mais fluxo ou, em alguns casos, **retornar fluxo** de uma aresta para outra.

O algoritmo tenta **aumentar o fluxo** aproveitando as arestas com **capacidade residual positiva** na rede residual. Cada vez que um aumento de fluxo é realizado, as capacidades residuais nas arestas da rede residual são **atualizadas**.

## Fluxo aumentado

Quando se aplica uma mudança (aumento de fluxo) em uma aresta, isso é chamado de "aumento do fluxo". Formalmente, o aumento do fluxo é a função que **atualiza** o fluxo na rede original com o fluxo na rede residual.

- Se o fluxo na aresta (u,v) na rede residual foi aumentado por f′(u,v), então o fluxo f(u,v) na rede original é atualizado somando esse valor.
- Se o fluxo foi enviado ao longo da aresta inversa (v,u), então o fluxo na aresta (u,v) da rede original será **reduzido**.

## Exemplo de cancelamento de fluxo:

Imagine que, inicialmente, você enviou 5 unidades de fluxo de u para v, mas depois deseja enviar 2 unidades de volta, ou seja, **diminuir o fluxo**. Em termos de rede residual, isso seria representado pela aresta inversa (v,u) com capacidade 5, permitindo que até 5 unidades de fluxo possam ser "retornadas". Se você enviar 2 unidades de volta, o fluxo na aresta (u,v) será **reduzido** em 2, e o fluxo na aresta (v,u) será **aumentado** em 2.

## Conclusão

A rede residual é essencial para entender como os **algoritmos de fluxo máximo** operam, pois ela permite que o fluxo seja tanto **aumentado** quanto **reduzido** ao longo de uma rede de fluxo. Ela representa as possibilidades de **ajuste** do fluxo nas arestas existentes, seja para adicionar mais fluxo ou para retornar parte do fluxo já enviado.

Vamos simplificar o conceito de **caminhos aumentadores** e o que ele implica em termos de **fluxo máximo**.

---

---

# O Algoritmo Ford-Fulkerson 🚀

O **algoritmo Ford-Fulkerson** resolve o problema do **fluxo máximo** encontrando **caminhos aumentadores** e **aumentando o fluxo iterativamente**.

## Passos do algoritmo

1️⃣ Inicializar o fluxo como **zero.**
2️⃣ Enquanto houver um caminho aumentador p na rede residual Gf, **aumentar o fluxo ao longo de p.**
3️⃣ Quando não houver mais caminhos aumentadores, **o fluxo encontrado é o máximo**.

💡 Cada iteração ajusta o fluxo usando a **capacidade residual** das arestas e atualiza a rede residual.

# Execução do Algoritmo 🕑

O **tempo de execução** do Ford-Fulkerson depende da escolha dos **caminhos aumentadores**:

- **Escolha ruim**: O algoritmo pode levar um número muito grande de iterações (ou até não terminar).
- **Escolha eficiente**: Se usarmos **Busca em Largura (BFS)** para encontrar caminhos aumentadores, ele roda em **tempo polinomial**.

✅ **Caso com capacidades inteiras**

- O fluxo sempre aumenta em pelo menos 1 unidade por iteração.
- Se o fluxo máximo **| f* |** for pequeno, o algoritmo é eficiente.
- Tempo de execução: **O(E | f*|)**, onde **E** é o número de arestas.

⚠️ **Problema da convergência lenta**
Se escolhermos caminhos aumentadores de forma ruim, podemos ter um caso extremo onde o fluxo sobe apenas 1 unidade por iteração.
Isso pode levar **milhões de iterações**, tornando o algoritmo **muito ineficiente**.

💡 **Solução:**
Usar **BFS** (como no **Algoritmo de Edmonds-Karp**) para garantir tempo **O(VE²)**, que é bem melhor.

```
FORD-FULKERSON(G, s, t)
    Para cada (u, v) ∈ E faça
        f(u, v) ← 0  // Inicializa o fluxo como zero
    Enquanto existir um caminho aumentador p de s até t em G_f faça
        c_f(p) ← min{ c_f(u, v) | (u, v) está em p }  // Capacidade residual do caminho
        Para cada aresta (u, v) em p faça
            f(u, v) ← f(u, v) + c_f(p)  // Aumenta o fluxo na aresta direta
            f(v, u) ← f(v, u) - c_f(p)  // Diminui o fluxo na reversa (rede residual)
    Retorna f  // Fluxo máximo encontrado
```

---

## Como Funciona o Algoritmo Edmonds-Karp?

Imagina que você tem uma cidade (a **fonte**) e várias ruas conectando a cidade a outras áreas (os **vértices** e **arestas**). Você quer mandar o máximo de carros (que representam o **fluxo**) de um ponto da cidade para outro (o **sorvedouro**, que é o destino final). A **capacidade das ruas** é como se fosse a quantidade máxima de carros que uma rua pode suportar de cada vez. O objetivo é encontrar o **fluxo máximo**, ou seja, a quantidade máxima de carros que podem sair da cidade sem que nenhuma rua fique sobrecarregada.

## O Algoritmo de Edmonds-Karp

O **Edmonds-Karp** é uma forma mais inteligente de resolver esse problema de fluxo máximo, melhorando o algoritmo anterior chamado **Ford-Fulkerson**. A diferença principal é que, no **Edmonds-Karp**, em vez de escolher qualquer rua aleatória para enviar os carros, nós sempre escolhemos a **rua mais curta** (ou o caminho mais curto) para enviar os carros da cidade até o destino.

Aqui está o que acontece em cada etapa:

1. **Fluxo Inicial**: A princípio, nenhum carro foi enviado, então o fluxo é **zero** em todas as ruas.

2. **Encontrar o Caminho Aumentador**: A cada vez que tentamos mandar carros, precisamos encontrar um **caminho aumentador**, ou seja, um caminho de ruas que ainda têm capacidade disponível para passar mais carros. Usamos a **busca em largura (BFS)** para garantir que estamos escolhendo o caminho mais curto possível.

3. **Aumentar o Fluxo**: Uma vez que encontramos esse caminho, a quantidade de carros que podemos mandar por esse caminho é determinada pela **capacidade mínima** das ruas nesse caminho. Ou seja, se a rua mais apertada (com menos

espaço) só pode suportar 10 carros, então mandamos 10 carros por esse caminho.

4. **Atualizar o Fluxo nas Ruas**: Depois de mandar os carros, as ruas têm agora menos capacidade para enviar mais carros, então vamos atualizar o fluxo nas ruas: as ruas que foram usadas para mandar os carros ficam mais “ocupadas” e as ruas no caminho inverso (para voltar os carros) ficam mais “desbloqueadas” para possíveis futuras passagens.

5. **Repetir**: Agora, com as ruas atualizadas, repetimos o processo. Buscamos o próximo caminho aumentador, mandamos mais carros, e assim por diante, até que não existam mais caminhos disponíveis.

## Por que Usar a Busca em Largura (BFS)?

A escolha do **caminho mais curto** é importante para garantir que o algoritmo funcione de forma eficiente. Se não fizermos isso, poderíamos acabar escolhendo caminhos muito longos, e isso poderia fazer o algoritmo demorar mais do que o necessário para encontrar o fluxo máximo.

## A Grande Sacada

O que faz o **Edmonds-Karp** ser mais eficiente do que o algoritmo anterior (Ford-Fulkerson) é que a cada vez que fazemos um aumento no fluxo, estamos **garantindo** que o próximo caminho será o mais curto possível, o que **ajuda a evitar que o algoritmo entre em um loop longo**.

## Complexidade do Algoritmo

O algoritmo de Edmonds-Karp tem uma complexidade de tempo **$O(VE^2)$**, ou seja, ele pode ser mais lento do que o algoritmo básico de Ford-Fulkerson, mas ainda assim é muito mais eficiente do que a versão anterior, já que escolhe o caminho mais curto de maneira sistemática.

```
EDMONDS-KARP(G, s, t)
  Para cada (u, v) ∈ E faça

    f(u, v) ← 0  // Inicializa o fluxo como zero

  Enquanto existir um caminho aumentador p de s até t em G_f (encontrado por BFS) faça

    c_f(p) ← min{ c_f(u, v) | (u, v) está em p }  // Capacidade residual do caminho

    Para cada aresta (u, v) em p faça
      f(u, v) ← f(u, v) + c_f(p)  // Aumenta o fluxo na aresta direta

      f(v, u) ← f(v, u) - c_f(p)  // Diminui o fluxo na reversa (rede residual)

  Retorna f  // Fluxo máximo encontrado
```

---