# Algoritmos e Estruturas de Dados II

Capítulo da Aula 24: Fluxo Máximo

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br
CEFET-MG - Campus Timóteo

Fevereiro de 2026

## 1. O Problema do Encanamento

Temos uma rede de tubulações conectando uma represa (**Fonte** $s$) a uma cidade (**Sorvedouro** $t$). Cada cano tem um diâmetro (**Capacidade**) diferente. Qual é a quantidade máxima de água que consegue chegar na cidade por segundo?

Este é o **Problema do Fluxo Máximo** — um dos problemas mais fundamentais em grafos, com aplicações que vão desde redes de transporte e telecomunicações até segmentação de imagens e bioinformática.

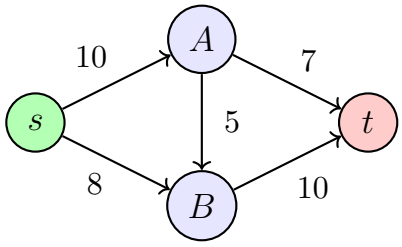



Figure 1: Rede de fluxo: fonte $s$, sorvedouro $t$, capacidades nas arestas. Qual a vazão máxima de $s$ a $t$?

## 2. Definição Formal: Rede de Fluxo

### Rede de Fluxo

**• Teoria**

Uma **Rede de Fluxo** é um grafo dirigido $G = (V, E)$ com:

- Uma **fonte** $s \in V$: vértice onde o fluxo é produzido (origem).
- Um **sorvedouro** $t \in V$: vértice onde o fluxo é consumido (destino).
- Uma função de **capacidade** $c : V \times V \to \mathbb{R}_{\geq 0}$, onde $c(u, v) = 0$ se $(u, v) \notin E$.
- **Sem arestas anti-paralelas:** para cada par $(u, v)$, se $(u, v) \in E$, então $(v, u) \notin E$.

**◇ Informação**

**Arestas anti-paralelas:** Se o grafo original possui arestas $(u, v)$ e $(v, u)$ simultaneamente, podemos eliminar o conflito inserindo um vértice auxiliar $w$ e substituindo $(u, v)$ por $(u, w)$ e $(w, v)$, ambas com a mesma capacidade original. Isso preserva a semântica sem alterar o fluxo máximo.

### Fluxo: Definição e Propriedades

**• Teoria**

Um **fluxo** em uma rede de fluxo $G$ é uma função $f : V \times V \to \mathbb{R}$ que satisfaz:

1 **Restrição de capacidade:** Para todo par $(u, v) \in V \times V$:

$$0 \leq f(u, v) \leq c(u, v)$$

2 **Conservação de fluxo:** Para todo vértice $u \in V \setminus \{s, t\}$:

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

(o que entra em $u$ é igual ao que sai de $u$)

3 **Valor do fluxo:** O valor de um fluxo $f$, denotado $|f|$, é o fluxo líquido saindo da fonte:

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

O **Problema do Fluxo Máximo** consiste em encontrar um fluxo $f$ que maximize $|f|$.

**Exemplo: Fluxo Válido vs Inválido**

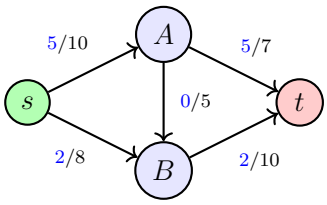



Figure 2: *
**Fluxo Válido** ($|f| = 7$)

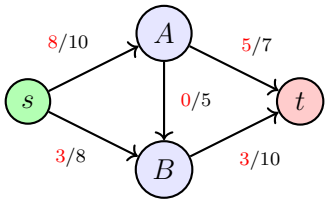



Figure 3: *
**Fluxo Inválido!**

No fluxo da esquerda: $A$ recebe 5, envia $5+0=5$ (conservação OK). $B$ recebe $2+0=2$, envia 2 (OK). No fluxo da direita: $A$ recebe **8**, mas envia apenas $5+0=\mathbf{5}$ — viola conservação no vértice $A$.

## 3. Múltiplas Fontes e Sorvedouros

Muitos problemas reais têm **várias fontes** $s_1, s_2, \ldots, s_k$ e **vários sorvedouros** $t_1, t_2, \ldots, t_m$. Para converter para o formato padrão (uma fonte, um sorvedouro):

**▶ Prática**

**Técnica: Super-Source e Super-Sink**

1. Crie um vértice **super-source** $S$ com arestas $S \to s_i$ de capacidade $\infty$ (ou a produção máxima de cada fonte $s_i$).
2. Crie um vértice **super-sink** $T$ com arestas $t_j \to T$ de capacidade $\infty$ (ou a demanda máxima de cada sorvedouro $t_j$).
3. Resolva o fluxo máximo de $S$ a $T$ na rede estendida.

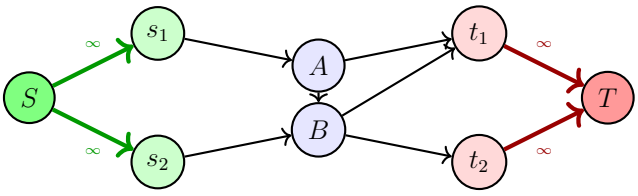



Figure 4: Transformação de múltiplas fontes/sorvedouros em fonte/sorvedouro únicos.

## 4. Caminho Aumentante e Grafo Residual

### Caminho Aumentante

**• Teoria**

Um **caminho aumentante** é um caminho de $s$ a $t$ no **grafo residual** $G_f$ onde todas as arestas têm capacidade residual estritamente positiva.
O **gargalo** (bottleneck) de um caminho aumentante $P$ é a menor capacidade residual ao longo dele:

$$c_f(P) = \min_{(u,v)\in P} c_f(u,v)$$

**• Teoria**

**Teorema (Ford-Fulkerson):** O fluxo $f$ é máximo se e somente se não existe caminho aumentante de $s$ a $t$ no grafo residual $G_f$.

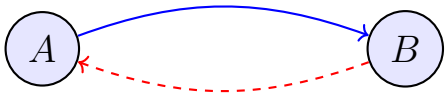
### Grafo Residual: Construção

A grande sacada do método **Ford-Fulkerson** é a capacidade de **desfazer** decisões. Dado um fluxo $f$ na rede $G$, o **grafo residual** $G_f$ tem os mesmos vértices de $G$, mas suas arestas têm capacidades residuais:

- **Aresta de ida** $(u,v)$: se $f(u,v) < c(u,v)$, existe aresta $(u,v)$ em $G_f$ com capacidade residual:
$$c_f(u,v) = c(u,v) - f(u,v) \quad \text{(quanto ainda cabe)}$$
- **Aresta reversa** $(v,u)$: se $f(u,v) > 0$, existe aresta $(v,u)$ em $G_f$ com capacidade residual:
$$c_f(v,u) = f(u,v) \quad \text{(quanto podemos “devolver”)}$$

Ida: $10 - 6 = 4$

**Original:** $c(A, B) = 10$**,** $f(A, B) = 6$




Volta: 6

Figure 5: Grafo residual: aresta de ida com capacidade restante; aresta reversa com fluxo "devolvível".

### Exemplo Completo de Grafo Residual

Considere a rede da Figura 1 após enviar 7 unidades pelo caminho $s \to A \to t$:

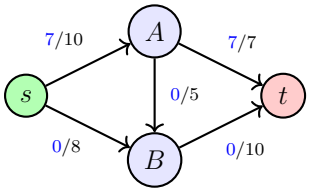



Figure 6: *
**Grafo com fluxo** ($|f| = 7$)

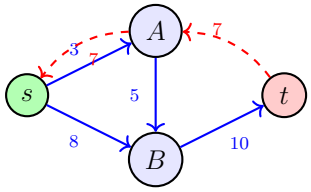



Figure 7: *
**Grafo Residual** $G_f$

Figure 8: Construção do grafo residual a partir do fluxo. A aresta $A \to t$ não aparece no residual (capacidade restante = 0).

Detalhamento aresta por aresta:

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

### Por que Arestas Reversas são Essenciais?

△ Importante

**Sem arestas reversas, o algoritmo pode ficar preso em soluções subótimas.** As arestas reversas permitem "redirecionar" fluxo para caminhos mais eficientes, efetivamente "desfazendo" decisões anteriores.

Considere o grafo "diamante" onde todas as capacidades são 1:

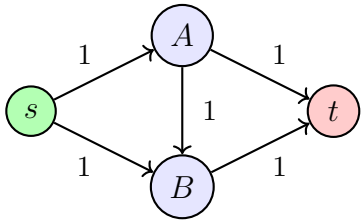



Figure 9: Grafo diamante: fluxo máximo real = 2.

Se a primeira iteração escolhe o caminho $s \to A \to B \to t$ (gargalo 1, $|f| = 1$), sem arestas reversas **não haveria mais caminhos** (pois $A \to B$ estaria saturada), resultando em $|f| = 1$ (subótimo!).

Com arestas reversas, a segunda iteração encontra $s \to B \to A \to t$, usando a aresta reversa $B \to A$ (capacidade 1). Isso efetivamente **redireciona**: $A$ passa a enviar diretamente para $t$ em vez de via $B$, e $B$ recebe diretamente de $s$. Resultado: $|f| = 2$ (ótimo!).

### Três Condições Equivalentes

• Teoria

As seguintes condições são equivalentes:

1. $f$ é um fluxo **máximo**.
2. **Não existe** caminho aumentante de $s$ a $t$ no grafo residual $G_f$.
3. $|f| = c(S, T)$ para algum corte $(S, T)$ (i.e., o fluxo atinge a capacidade de algum corte).

## 5. Ford-Fulkerson: O Método Genérico

O método de Ford-Fulkerson é um *método* (não um algoritmo específico), pois não especifica **como** encontrar o caminho aumentante. A escolha do método de busca determina a complexidade.

### Pseudocódigo

**▶ Prática**

**Método Ford-Fulkerson:**

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

2 **Enquanto** existir caminho aumentante $P$ de $s$ a $t$ no grafo residual $G_f$:

- Calcule o gargalo: $c_f(P) = \min\{c_f(u, v) : (u, v) \in P\}$.
- Para cada aresta $(u, v)$ no caminho $P$:
  - $f(u, v) \leftarrow f(u, v) + c_f(P)$ (aumenta o fluxo na ida)
  - $f(v, u) \leftarrow f(v, u) - c_f(P)$ ("devolve" na volta)

3 Retorne $|f|$.

### Trace Passo a Passo

Usando a rede da Figura 1 ($s \xrightarrow{10} A \xrightarrow{7} t$, $s \xrightarrow{8} B \xrightarrow{10} t$, $A \xrightarrow{5} B$):

| Iteração | Caminho Encontrado | Gargalo | $\lvert f \rvert$ |
|---|---|---|---|
| 1 | $s \to A \to t$ | $\min(10, 7) = 7$ | 7 |
| 2 | $s \to B \to t$ | $\min(8, 10) = 8$ | 15 |
| 3 | $s \to A \to B \to t$ | $\min(3, 5, 2) = 2$ | 17 |
| 4 | Sem caminho | — | **17 (máximo)** |

Na iteração 3, as capacidades residuais são: $s \to A$ tem $10 - 7 = 3$, $A \to B$ tem $5 - 0 = 5$, $B \to t$ tem $10 - 8 = 2$. O gargalo é $\min(3, 5, 2) = 2$.

### Complexidade e o Caso Patológico

**△ Importante**

**Complexidade:** $O(E \cdot |f^*|)$, onde $|f^*|$ é o valor do fluxo máximo.
Com DFS, Ford-Fulkerson pode ser **exponencialmente lento**:

- Em um grafo com arestas de capacidade $M$ muito grande, DFS pode escolher caminhos que passam por uma aresta de capacidade 1, adicionando apenas 1 unidade de fluxo por iteração.
- Com $M = 10^6$: até $2 \times 10^6$ iterações!
- Com capacidades **irracionais**: pode **não convergir** para o valor correto!

▷ **Exemplo**

**Caso Patológico:** Considere a rede com $s \xrightarrow{M} A$, $s \xrightarrow{M} B$, $A \xrightarrow{1} B$, $A \xrightarrow{M} t$, $B \xrightarrow{M} t$. O fluxo máximo é $2M$, mas DFS pode alternar entre $s \to A \to B \to t$ e $s \to B \to A \to t$ (via reversa), enviando apenas 1 unidade por iteração, totalizando $2M$ iterações.

## 6. Algoritmo de Edmonds-Karp

### Ideia Central: BFS em vez de DFS

O algoritmo de Edmonds-Karp é a implementação do método Ford-Fulkerson usando **BFS** para encontrar o caminho aumentante **mais curto** (em número de arestas) no grafo residual a cada iteração. Essa escolha garante complexidade **polinomial**, independente do valor do fluxo.

• **Teoria**

**Lema-Chave:** Para todo vértice $v$, a distância $d(s, v)$ no grafo residual **nunca diminui** ao longo das iterações de Edmonds-Karp.
**Consequência:** Cada aresta pode ser "crítica" (gargalo) no máximo $O(V/2)$ vezes. Como há $O(E)$ arestas, há no máximo $O(VE)$ aumentações.

### Por que BFS e não DFS?

| Aspecto | DFS (Ford-Fulkerson) | BFS (Edmonds-Karp) |
|---|---|---|
| Tipo de caminho | Qualquer (pode ser longo) | Mais curto (menos arestas) |
| Iterações | Até $\lvert f^* \rvert$ | No máximo $O(VE)$ |
| Complexidade total | $O(E \cdot \lvert f^* \rvert)$ | $O(V \cdot E^2)$ |
| Capacidades irracionais | Pode não convergir | Sempre termina |

No caso patológico anterior, BFS encontra diretamente os caminhos $s \to A \to t$ e $s \to B \to t$ (ambos com 2 arestas, enviando $M$ cada), finalizando em apenas 2 iterações.

### Trace Completo: Grafo com 6 Vértices

Considere a seguinte rede:

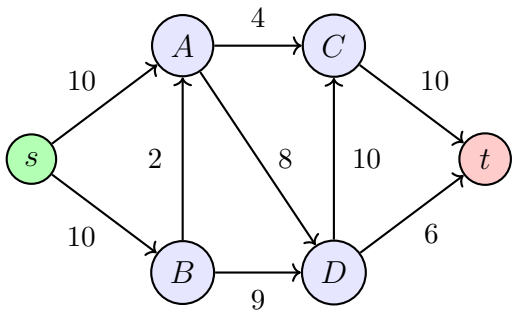



Figure 10: Rede de 6 vértices para trace de Edmonds-Karp.

| Iter | Caminho BFS | Gargalo | $\|f\|$ |
|---|---|---|---|
| 1 | $s \to A \to C \to t$ | $\min(10, 4, 10) = 4$ | 4 |
| 2 | $s \to B \to D \to t$ | $\min(10, 9, 6) = 6$ | 10 |
| 3 | $s \to A \to D \to C \to t$ | $\min(6, 8, 10, 6) = 6$ | 16 |
| 4 | BFS: $t$ inalcançável | — | **16 (máximo)** |

Após a iteração 3, o grafo residual não possui mais caminho de $s$ a $t$. Fluxo máximo $= 16$.

## 7. Implementação Java: Edmonds-Karp

Para Fluxo Máximo, a **Matriz de Adjacência** (`cap[u][v]`) é mais prática que lista de adjacência, desde que $V \leq 500$.

```java
import java.util.*;

public class MaxFlow {
    static final int INF = Integer.MAX_VALUE;

    public static int edmondsKarp(int N, int[][] cap, int s,
       int t) {
        int flow = 0;
        int[] parent = new int[N];

        while (true) {
            // 1. BFS no grafo residual
            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

            // 2. Gargalo do caminho
            int push = INF;
            for (int v = t; v != s; v = parent[v])
                push = Math.min(push, cap[parent[v]][v]);

            // 3. 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;
    }
}
```

◇ Informação

**Observação sobre a implementação:** A matriz `cap[u][v]` serve simultaneamente como grafo e grafo residual. Ao subtrair na ida e somar na volta, automaticamente criamos as arestas reversas. Essa elegância é possível porque $c(u, v) = 0$ para arestas inexistentes, e a soma na volta cria a capacidade de "devolução".

**Complexidade:** $O(V \cdot E^2)$. Na prática, é muito mais rápido que o pior caso sugere.

## 8. Teorema Max-Flow Min-Cut

### O que é um Corte $s$-$t$?

• Teoria

Um **corte** $s$-$t$ é uma partição dos vértices de $G$ em dois conjuntos $S$ e $T = V \setminus S$, com $s \in S$ e $t \in T$.
A **capacidade do corte** $(S, T)$ é:

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

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

### Exemplo: Todos os Cortes de um Grafo

Na rede da Figura 1, existem $2^{|V|-2} = 2^2 = 4$ possíveis cortes $s$-$t$:

| # | $S$ | **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** |

O corte mínimo é $\{s, A, B\}|\{t\}$ com capacidade **17**. Isso é exatamente igual ao fluxo máximo — **não é coincidência!**

### O Teorema

**• Teoria**

**Teorema Max-Flow Min-Cut (Ford & Fulkerson, 1956):**
Em qualquer rede de fluxo, o **valor do fluxo máximo** é exatamente igual à **capacidade do corte mínimo** que separa $s$ de $t$:

$$\max_f |f| = \min_{(S,T)} c(S, T)$$

### Intuição da Prova

A prova se divide em duas partes:

1. $|f| \leq c(S, T)$ **para qualquer corte:** Todo o fluxo de $s$ a $t$ tem que "atravessar" qualquer corte. Como $f(u, v) \leq c(u, v)$, o fluxo não pode exceder a capacidade do corte.
2. **Existe um corte com $|f^*| = c(S^*, T^*)$:** 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** (capacidade residual $= 0$), logo $|f^*| = c(S^*, T^*)$.

### Extraindo o Corte Mínimo na Prática

Após executar Edmonds-Karp:

**▶ Prática**

**Algoritmo para encontrar o Corte Mínimo:**

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

```java
// Apos Edmonds-Karp: BFS no residual a partir de s
boolean[] reachable = new boolean[N];
Queue<Integer> q = new LinkedList<>();
q.add(s); reachable[s] = true;
while (!q.isEmpty()) {
    int u = q.poll();
    for (int v = 0; v < N; v++)
        if (!reachable[v] && cap[u][v] > 0) {
            reachable[v] = true; q.add(v);
        }
}
// Arestas do corte: originalCap[u][v] > 0 && reachable[u] && !
    reachable[v]
```

▷ Exemplo

**Aplicação Prática:** "Qual o menor número de conexões que, se cortadas, desconecta duas redes?" ⇒ Corte mínimo! Defina cada aresta com capacidade 1 e encontre o corte mínimo.

## 9. Algoritmo de Dinic

O algoritmo de Dinic melhora o Edmonds-Karp usando dois conceitos-chave: **Level Graph** e **Blocking Flow**.

### Level Graph (Grafo de Níveis)

• Teoria

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

A construção do Level Graph é simples: execute BFS a partir de $s$ no grafo residual. Arestas entre vértices do **mesmo nível** ou que "voltam" para níveis anteriores são **removidas**.

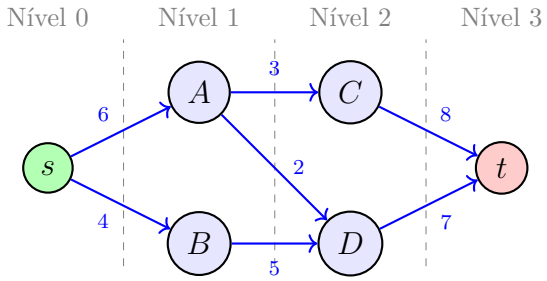



Figure 11: Level Graph: apenas arestas que avançam um nível são mantidas. Arestas entre vértices do mesmo nível (e.g., $A \to B$) são removidas.

### Blocking Flow (Fluxo de Bloqueio)

**• Teoria**

Um **Blocking Flow** é um fluxo no Level Graph que **satura pelo menos uma aresta** em cada caminho de $s$ a $t$. Após um blocking flow, nenhum caminho de $s$ a $t$ sobrevive no Level Graph atual.

O blocking flow é encontrado via **DFS** com poda:

1. DFS de $s$ buscando $t$ no Level Graph.
2. Ao encontrar $t$: sature o caminho (envie o gargalo).
3. **Dead-end** (beco sem saída): poda o vértice e retrocede.
4. Continue DFS para encontrar mais caminhos.
5. Pare quando DFS não alcançar $t$.

### Fluxo do Algoritmo

**▶ Prática**

**Algoritmo de Dinic:**

1. Construa o Level Graph via BFS no grafo residual.
2. Se $t$ não é alcançável: **pare** (fluxo é máximo).
3. Encontre um Blocking Flow via DFS no Level Graph.
4. Atualize o grafo residual com o fluxo encontrado.
5. Volte ao passo 1.

### Vantagem sobre Edmonds-Karp

Em uma **única fase**, Dinic encontra **múltiplos** caminhos aumentantes (todos de mesmo comprimento mínimo), enquanto Edmonds-Karp encontra apenas **um** por iteração.

### Complexidade

| Caso | Complexidade | Justificativa |
|---|---|---|
| Geral | $O(V^2 \cdot E)$ | $\leq V$ fases, cada fase $O(VE)$ |
| Caps unitárias | $O(E\sqrt{V})$ | $\leq \sqrt{V}$ fases |

Para **matching bipartido** (todas as capacidades $= 1$), Dinic em $O(E\sqrt{V})$ é o algoritmo mais rápido na prática, equivalente ao algoritmo de Hopcroft-Karp.

## 10. Comparação de Algoritmos

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

◇ Informação

**Na prática:**

- **Maratona/Competição:** Dinic é o mais usado (rápido, ∼40 linhas de código).
- **Didático/Aprendizado:** Edmonds-Karp (mais fácil de entender e implementar).
- **Grafos densos muito grandes:** Push-Relabel FIFO ($O(V^3)$, sem dependência em $E$).

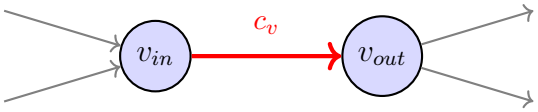
## 11. Modelagem: Capacidade nos Vértices (Vertex Splitting)

Às vezes, cada vértice $v$ tem uma capacidade máxima $c_v$ de fluxo que pode **passar por ele**. Para modelar isso:

▶ **Prática**

**Técnica: Vertex Splitting**

1. Divida cada vértice $v$ em dois: $v_{\text{in}}$ e $v_{\text{out}}$.
2. Todas as arestas que **entram** em $v$ vão para $v_{\text{in}}$.
3. Todas as arestas que **saem** de $v$ partem de $v_{\text{out}}$.
4. Adicione aresta interna $v_{\text{in}} \to v_{\text{out}}$ com capacidade $c_v$.




Aresta interna limita o fluxo total

Figure 12: Vertex splitting: a aresta interna $v_{\text{in}} \to v_{\text{out}}$ com capacidade $c_v$ garante que no máximo $c_v$ unidades passam pelo vértice.

▷ **Exemplo**

**Aplicação:** "Cada roteador suporta no máximo $K$ pacotes por segundo." $\Rightarrow$ Vertex splitting com $c_v = K$ para cada roteador.
**Conectividade de vértices:** Para encontrar o número mínimo de vértices cuja remoção desconecta $s$ de $t$, aplique vertex splitting com $c_v = 1$ para cada vértice (exceto $s$ e $t$) e calcule o fluxo máximo.

## 12. Matching Bipartido via Fluxo Máximo

O **Matching Máximo Bipartido** é um caso especial do fluxo máximo.

### O Problema

Dado um grafo bipartido $G = (L \cup R, E)$, encontrar o maior conjunto de arestas tal que nenhum vértice seja coberto por mais de uma aresta.

### Modelagem como Fluxo

**▶ Prática**

1. Crie um **super-source** $S$ com arestas $S \to l$ de capacidade 1, para cada $l \in L$.
2. Crie um **super-sink** $T$ com arestas $r \to T$ de capacidade 1, para cada $r \in R$.
3. Para cada aresta $(l, r)$ do grafo bipartido, crie aresta $l \to r$ com capacidade 1.
4. **Fluxo máximo = tamanho do matching máximo!**

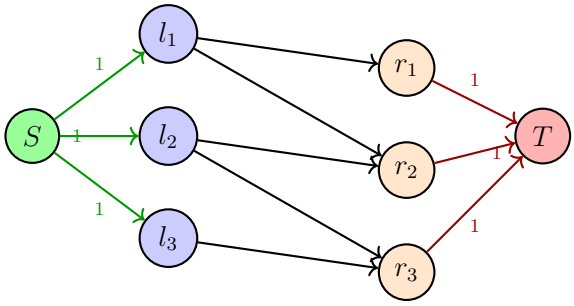



Figure 13: Matching bipartido modelado como fluxo. Fluxo máximo = 3 = matching perfeito: $l_1 \to r_1$, $l_2 \to r_2$, $l_3 \to r_3$.

Com Dinic (capacidades unitárias), a complexidade é $O(E\sqrt{V})$, que é equivalente ao algoritmo especializado de Hopcroft-Karp.

**◇ Informação**

**Papel das arestas reversas no matching:** Se $l_1$ for inicialmente emparelhado com $r_2$, mas depois $l_2$ (que só pode ir para $r_2$) precisar de $r_2$, a aresta reversa permite "desemparelhar" $l_1$ de $r_2$ e redirecioná-lo para $r_1$. Isso é exatamente o que os "caminhos alternados" do algoritmo de Hopcroft-Karp fazem!

## 13. Circulação com Demandas (Lower Bounds)

### O Problema

Em algumas redes, cada aresta $(u, v)$ tem não apenas uma capacidade máxima $c(u, v)$, mas também um fluxo **mínimo obrigatório** (demanda) $d(u, v)$. Queremos:

$$d(u, v) \leq f(u, v) \leq c(u, v)$$

### Transformação para Fluxo Padrão

**▶ Prática**

1. Para cada aresta $(u, v)$ com demanda $d$ e capacidade $c$:
   - Nova capacidade: $c' = c - d$.
   - Acumule em cada vértice: $v$ "recebe" $d$ obrigatoriamente (excesso positivo), $u$ "deve" enviar $d$ (excesso negativo).
2. Crie super-source $S'$ e super-sink $T'$:
   - Para cada vértice $v$ com excesso positivo $D_v > 0$: aresta $S' \to v$ com capacidade $D_v$.
   - Para cada vértice $u$ com excesso negativo $D_u < 0$: aresta $u \to T'$ com capacidade $|D_u|$.
3. Aresta $t \to s$ com capacidade $\infty$ (fecha o ciclo).
4. Se o fluxo máximo de $S'$ a $T'$ **satura todas** as arestas de $S'$: a circulação é viável!

## 14. Project Selection (Closure Problem)

### O Problema

Temos um conjunto de projetos, cada um com um **lucro** (que pode ser positivo ou negativo — custo). Há **dependências**: se o projeto $A$ depende de $B$, então para fazer $A$ também devemos fazer $B$. Queremos selecionar o subconjunto de projetos que **maximiza o lucro total**.

### Modelagem como Min-Cut

**▶ Prática**

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

**◇ Informação**

**Intuição:** O min-cut decide o que "cortar". Cortar a aresta de $s$ significa **desistir de um lucro**. Cortar a aresta para $t$ significa **pagar um custo**. Dependências com capacidade $\infty$ garantem que nunca são cortadas — se selecionamos $A$, obrigatoriamente selecionamos $B$.

# 15. Dicas de Modelagem e Padrões Comuns

### Checklist de Modelagem

**✓ title=✓ Checklist para Problemas de Fluxo**

1. **Identifique** fonte e sorvedouro.
   - Múltiplas fontes/sorvedouros? ⇒ Super-source / super-sink.
2. **Defina** capacidades nas arestas (representam limites).
3. Capacidade nos **vértices**? ⇒ Vertex splitting.
4. **Execute** Edmonds-Karp ou Dinic.
5. Precisa do **corte mínimo**? ⇒ BFS no residual após o fluxo.

### Padrões Frequentes

| 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) |
| Distribuição de tarefas | Super-source (produção), super-sink (demanda) |

**△ Importante**

**Dica de Maratona:** Se o problema menciona "capacidade", "gargalo", "vazão máxima", "corte mínimo", "matching", ou "distribuição ótima" ⇒ pense em **Fluxo Máximo**!

# 16. Exercícios

### 16.1. Exercícios Conceituais

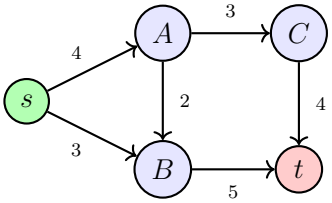
1. Explique com suas palavras o conceito de **grafo residual**. Por que arestas reversas são essenciais para a corretude do algoritmo?
2. Enuncie o **Teorema Max-Flow Min-Cut**. Descreva como identificar o corte mínimo após executar Edmonds-Karp.
3. Por que Ford-Fulkerson com DFS pode ser exponencialmente lento, enquanto Edmonds-Karp (BFS) garante complexidade polinomial? Dê um exemplo concreto.
4. Compare Edmonds-Karp, Dinic e Push-Relabel em termos de complexidade, facilidade de implementação e casos de uso recomendados.

5 Explique a técnica de **vertex splitting**. Em que situações ela é necessária?

6 Como o **matching bipartido** se reduz a um problema de fluxo máximo? Qual a complexidade resultante com Dinic?

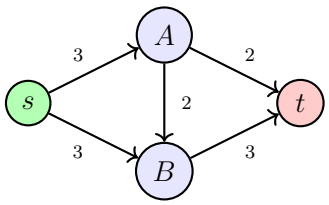
### 16.2. Exercícios Analíticos

1 **Trace manual:** Simule Edmonds-Karp passo a passo na rede abaixo. Mostre cada caminho BFS, o gargalo, o fluxo acumulado e o grafo residual após cada iteração.




**Resposta:** Caminhos: $s \to A \to C \to t$ (garg=3), $s \to B \to t$ (garg=3), $s \to A \to B \to t$ (garg=1). Fluxo máximo $= 7$. Corte mínimo: $\{s\}|\{A, B, C, t\}$, capacidade $4 + 3 = 7$.

2 **Aresta reversa crucial:** Simule Edmonds-Karp na rede abaixo. Em alguma iteração, uma aresta reversa é usada como parte do caminho BFS?




3 **Corte mínimo:** Após executar Edmonds-Karp em uma rede e obter $|f^*| = 19$, o grafo residual permite BFS alcançar apenas $\{s, B\}$ a partir de $s$. Dado que as arestas originais de $\{s, B\}$ para os demais são $s \to A$ (cap 12) e $B \to D$ (cap 7), verifique que $12 + 7 = 19$.

4 **Todos os cortes:** Na rede da Figura 1, enumere todos os $2^2 = 4$ cortes $s$-$t$ possíveis, suas capacidades, e identifique o corte mínimo.

5 **Modelagem — Fábricas e Lojas:** 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). Capacidades de transporte dadas. Como modelar como rede de fluxo? O que $S$, $T$ e o fluxo máximo representam?

**Resposta:** Super-source $S \to$ fábricas (cap = produção). Lojas $\to$ Super-sink $T$ (cap = demanda). Fábricas $\to$ lojas com caps de transporte. Fluxo máximo = distribuição ótima.

6 **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 $P_1$. Modele como fluxo e encontre o matching máximo.

**Resposta:** Matching máximo = 3: $A_1 \to P_2$, $A_2 \to P_3$, $A_3 \to P_1$. A aresta reversa é crucial se $A_1$ for inicialmente atribuído a $P_1$.

### 16.3. Exercícios de Programação

1 Implemente Edmonds-Karp completo com identificação do corte mínimo. Teste na rede da Figura 1.

2 Resolva emparelhamento máximo bipartido usando sua implementação de fluxo máximo. Crie a modelagem com super-source e super-sink.

3 Implemente uma função que, dado o grafo residual final, identifique e imprima todas as arestas do corte mínimo.

4 **Desafio:** Implemente o algoritmo de Dinic com Level Graph (BFS) e Blocking Flow (DFS com poda). Compare o desempenho com Edmonds-Karp em grafos grandes.

5 Resolva problemas de juiz online:

- **UVa 820** — Internet Bandwidth (fluxo direto)
- **UVa 10480** — Sabotage (corte mínimo)
- **UVa 259** — Software Allocation (matching bipartido via fluxo)

## Resumo

### ✓ title=✓ Resumo: 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 | — | Caps nos vértices. |
| Matching Bipartido | $O(E\sqrt{V})$ | Dinic com caps unitárias. |

**Conceitos-chave:**

- O **grafo residual** permite desfazer decisões via arestas reversas.
- O **Teorema Max-Flow Min-Cut** garante que fluxo máximo = capacidade do corte mínimo.
- **BFS** (Edmonds-Karp) garante convergência polinomial.
- **Modelagem** é a parte mais difícil: super-source/sink, vertex splitting, capacidades corretas.