Algoritmos e Estruturas de Dados II

Capítulo da Aula 25: Aplicações de Fluxo Máximo

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

Junho de 2026

Introdução e Revisão do Teorema Max-Flow Min-Cut

1. O Coração de Tudo: Redes de Fluxo

Nas aulas anteriores, aprendemos a calcular o **Fluxo Máximo** em um grafo direcionado usando algoritmos como Ford-Fulkerson, Edmonds-Karp e Dinic. Modelamos uma rede com uma Fonte (Source - S) e um Sorvedouro (Sink - T), respeitando a conservação de fluxo em todos os nós intermediários e as capacidades máximas de cada aresta.

No entanto, calcular o fluxo máximo é apenas 10% do trabalho em problemas reais e competições de programação. Os 90% restantes são perceber que o seu problema específico pode ser **reduzido a uma rede de fluxo**! O segredo não é programar o Dinic mais rápido, mas construir o grafo corretamente.

2. O Teorema Max-Flow Min-Cut (1956)

Para entendermos aplicações avançadas de fluxo, precisamos dominar o teorema fundamental estabelecido por L. R. Ford Jr. e D. R. Fulkerson:

Teorema Max-Flow Min-Cut

O valor do **Fluxo Máximo** de S a T é exatamente igual à capacidade do **Corte Mínimo** que separa S de T.

O que é um Corte (Cut)?

Um corte (S_{cut}, T_{cut}) particiona os vértices do grafo em dois conjuntos disjuntos, de forma que a fonte $S \in S_{cut}$ e o sorvedouro $T \in T_{cut}$. A **Capacidade do Corte** é a soma das capacidades de todas as arestas originais que vão de vértices em S_{cut} para vértices em T_{cut}. (Arestas no sentido inverso não contam).

O **Corte Mínimo** é aquele cuja capacidade tem o menor valor possível. Em termos práticos, ele identifica o "gargalo" da rede: se essas arestas forem removidas, a comunicação entre S e T é totalmente interrompida da forma mais "barata" possível.

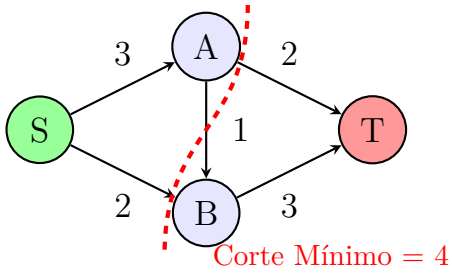

Figure 1: Exemplo de Corte Mínimo. As arestas interceptadas do conjunto S para o T são $A \to T$ (cap 2) e $S \to B$ (cap 2). A capacidade total é 4, logo o Fluxo Máximo também é 4.

Técnicas de Modelagem e Adaptações de Redes

3. Múltiplas Fontes e Múltiplos Sorvedouros

Muitas vezes você terá várias fábricas enviando produtos (várias fontes) e várias cidades consumidoras (vários sorvedouros). Algoritmos padrão aceitam apenas 1 fonte e 1 sorvedouro.

Solução: Super-Nós

1. Crie um nó virtual chamado **Super-Fonte** (SS).
2. Ligue SS a todas as fontes reais do problema. A capacidade desta aresta pode ser a **oferta máxima** daquela fábrica. Se for ilimitado, ponha ∞.
3. Crie um nó virtual chamado **Super-Sorvedouro** (TT).
4. Ligue todos os sorvedouros reais a TT. A capacidade é a **demanda** de cada cidade (ou ∞).

4. Capacidades em Vértices (Vertex Capacities)

Imagine que as rodovias (arestas) têm grande capacidade, mas a fiscalização de uma cidade (vértice) só consegue processar 50 caminhões por dia. Redes clássicas limitam fluxo nas **arestas**. Como limitar no vértice?

Solução: Divisão do Vértice (Node Splitting) Dividimos o vértice V em dois vértices virtuais: V_{in} e V_{out}.

- Todas as arestas originais que **entravam** em V agora entram em V_{in}.
- Todas as arestas originais que **saíam** de V agora saem de V_{out}.
- Criamos uma **única aresta** de V_{in} para V_{out} com a capacidade interna do vértice!

5. Arestas Não-Direcionadas e Mão Dupla

E se uma tubulação permite 10 litros de passagem total entre A e B (não importa a direção, se passar 6 de A para B só podem passar 4 de B para A ao mesmo tempo)?

Em algoritmos que usam rede residual (onde a aresta reversa ganha capacidade para cancelar o fluxo), basta adicionar duas arestas:

- Adicionar no grafo: `A -> B` com capacidade 10 (e `B -> A` começa com 0 na residual).
- Adicionar no grafo: `B -> A` com capacidade 10 (e `A -> B` começa com 0 na residual).

Isso se traduz apenas em passar a mesma capacidade para o parâmetro `rev_cap` no seu código de estrutura de dados.

Aplicações Clássicas de Max-Flow

6. Emparelhamento Bipartido Máximo (Maximum Bipartite Matching)

O Emparelhamento Bipartido é o ato de conectar elementos do grupo U a elementos do grupo V, de forma que ninguém seja designado a mais de um par. Exemplo: Trabalhadores e Máquinas, Alunos e TCCs, Garotos e Garotas para um baile.

Em vez de usar algoritmos específicos $O(V \cdot E)$ como Hopcroft-Karp, podemos modelar de forma extremamente flexível via Fluxo:

1. Conecte $S \to U_i$ com capacidade 1. (Cada trabalhador só trabalha 1 vez).
2. Conecte $V_j \to T$ com capacidade 1. (Cada máquina só precisa de 1 pessoa).
3. Aresta direcional $U_i \to V_j$ com capacidade 1, se U_i sabe operar V_j.
4. Fluxo Máximo! O valor do fluxo é o tamanho do emparelhamento máximo.

▷ A grande vantagem de usar Fluxo aqui

Se o Hopcroft-Karp for usado, restrições limitam a 1 para 1. Mas com Max-Flow, e se a máquina 3 puder ser usada por **3 operários simultaneamente**? É só mudar a capacidade de $V_3 \to T$ para 3! E se o operário 2 for um sênior que aguenta rodar **2 máquinas**? Mude a capacidade de $S \to U_2$ para 2. A flexibilidade do Max-Flow torna ele a ferramenta preferida para emparelhamentos complexos.

7. Problema de Cobertura de Caminhos em DAGs (Path Cover)

Imagine escalonar táxis, trens ou aviões. Cada voo tem um horário e origem/destino (ex: V_1: SP $\to$ RJ, 10h-11h). Se houver um voo V_2: RJ $\to$ BA às 12h-14h, o **mesmo** avião pode fazer V_1 seguido de V_2 (já que 11h $<$ 12h). Qual o **mínimo de aviões** necessários para cobrir todos os voos previstos para hoje?

Trata-se de encontrar o menor número de caminhos disjuntos que cobrem todos os vértices de um Grafo Acíclico Direcionado (DAG). A resposta matemática pelo Teorema de Dilworth é:

Aviões = (Total de Voos) - (Emparelhamento Bipartido Máximo)

Se fizermos uma aresta Bipartida entre $V_{1,out}$ e $V_{2,in}$ sempre que um avião de 1 consegue chegar a tempo para fazer 2, e rodarmos o emparelhamento máximo, cada emparelhamento feito significa "uma união" (um voo sendo reaproveitado do avião anterior), economizando a compra de um novo avião.

Aplicações Avançadas de Min-Cut

8. O Problema da Seleção de Projetos

Sua empresa de software quer pegar projetos freelancers.

- O **Projeto A** paga \$50 e depende da Ferramenta de Modelagem (custo \$30).
- O **Projeto B** paga \$40 e depende da Ferramenta de Modelagem (\$30) e Banco de Dados (\$20).
- O **Projeto C** paga \$20 e depende do Banco de Dados (\$20).

O objetivo é escolher os Projetos a realizar de modo a **Maximizar o Lucro Líquido** (Soma dos pagamentos - Soma do custo de ferramentas). Ferramentas só precisam ser compradas **uma vez** para todos. Heurísticas gulosas não funcionam porque um projeto individual pode dar prejuízo (A: \$50-\$30 = \$20 lucro, ok. B: \$40-\$50 = -\$10, não faço), mas ao combinar projetos que dividem a mesma ferramenta, o cenário muda ($A+B$ = \$90, Ferramentas = \$50, Lucro = \$40. Excelente!).

Modelagem como Corte Mínimo:

1. $S \rightarrow$ Projeto com capacidade = Pagamento.
2. Ferramenta $\rightarrow T$ com capacidade = Custo.
3. Projeto $\rightarrow$ Ferramenta(s) que ele usa com capacidade ∞ (nunca será cortada).

Calculamos o Min-Cut entre S e T. O corte divide a rede. Se cortou a aresta $S \rightarrow P_A$, significa: "Eu escolho NÃO fazer o projeto A" (o corte contabiliza o "dinheiro que deixei de ganhar" como um prejuízo na capacidade do corte). Se cortou $F_1 \rightarrow T$, significa: "Eu escolho comprar a ferramenta 1" (o corte contabiliza o custo real no prejuízo).

$$\text{Lucro Máximo} = (\sum \text{Pagamentos de Todos os Projetos}) - \text{Valor do Min-Cut}$$

9. Segmentação de Imagem e Min-Cut

Outra aplicação brilhante do Corte Mínimo é em **Visão Computacional** (como ferramentas "Remover Fundo" e "Laço Magnético"). Cada **pixel** é um vértice.

- S representa o objeto central (Gato).

- T representa o fundo (Background).
- As arestas de $S \to p_i$ e $p_i \to T$ têm pesos baseados em probabilidade (se o pixel for vermelho como a roupa do objeto, a aresta para S é forte).
- Colocamos arestas de penalidade entre $p_i \leftrightarrow p_j$ vizinhos, que são fracas quando as cores são muito diferentes, e fortes quando são iguais.

Quando rodamos o Min-Cut, o algoritmo "quebra" os laços entre pixels exatamente na borda do objeto, porque é onde as cores mudam drasticamente, tornando as arestas de vizinhos mais baratas de serem cortadas. O resultado é o isolamento perfeito do Gato em relação ao Fundo!

Exercícios

1 **(Básico)** Em uma rede clássica de água, a cidade A produz até 100L e a B até 200L. O objetivo é enviar para X (demanda 150L) e Y (demanda 150L). Como construir as capacidades para as Fontes S_A, S_B e Sorvedouros T_X, T_Y com os nós virtuais SS e TT?

2 **(Corte e Teorema)** Se todos os caminhos do Sorvedouro à Fonte passam necessariamente por um viaduto de capacidade 15 e uma ponte de capacidade 25, qual é a capacidade máxima que o Min-Cut pode ter?

3 **(Divisão de Vértice)** Considere que o vértice central C de um grafo de fluxo possui capacidade interna 5, e tem arestas chegando de X (cap 10) e Y (cap 10), e saindo para Z (cap 20). Desenhe o grafo resultante da aplicação do Node Splitting.

4 **(Problema do Banco de Sangue)** Existem estoques de Sangue O-, O+, A+, B+ e pacientes necessitando de doações de tipos específicos. Sabendo das regras de quem pode receber de quem (ex: A+ pode receber de O-, O+, A-, A+), explique a modelagem de fluxo para saber se **todos** os pacientes podem ser salvos simultaneamente. (Quais capacidades usar para doadores, para pacientes e para o Super Source/Sink?)

5 **(Projetos)** O Projeto A rende \$100 e precisa das ferramentas F1(\$60) e F2(\$40). O Projeto B rende \$100 e precisa da ferramenta F2(\$40) e F3(\$30). O Projeto C rende \$100 e precisa da F1(\$60). Construa o grafo, desenhe as capacidades ∞ e resolva o Min-Cut (visualmente) para encontrar o lucro máximo possível.