

Trabalho 2 - Problema dos Caminhos Mínimos

Igor J. Rodrigues; Leonardo S. Coradeli; Lucas V. Ikeda; Marco V. M. Faria

Universidade Estadual Paulista “Júlio de Mesquita Filho”
Faculdade de Ciência e Tecnologia

9 de novembro de 2025

1 Introdução e Definição do Problema

2 Modelo Matemático (PL)

3 Métodos de Resolução

4 Referências Bibliográficas

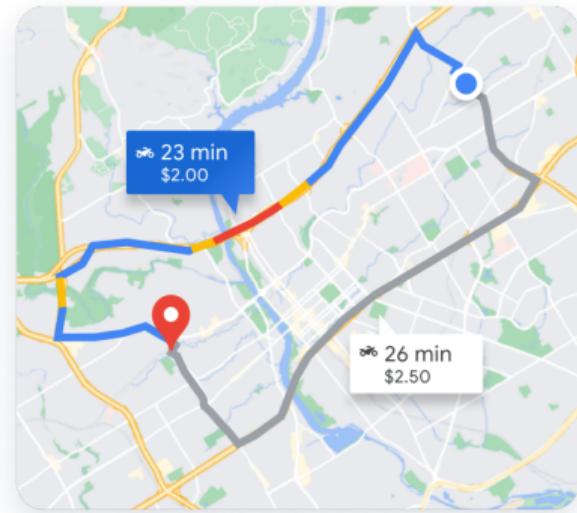
Qual é o melhor caminho?

A Pergunta Intuitiva

Como o Google Maps sabe a rota mais rápida para chegar na FCT?

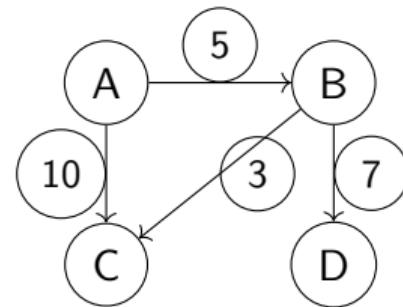
- É o caminho mais rápido (menor tempo)?
- É o caminho mais curto (menor distância)?
- É o caminho mais barato (sem pedágios)?

A ideia central é modelar o mapa como um **grafo** e atribuir **custos** às ruas.



- **Redes de Computadores:** Roteamento de pacotes na internet (Protocolo OSPF)
— o custo é a latência.
- **Logística e Transporte:** Otimização de rotas de entrega (Correios, Mercado Livre) — o custo é a distância ou tempo.
- **Finanças (Avançado):** Detecção de oportunidades de arbitragem em mercados (encontrar ciclos de custo negativo).
- **Análise de Redes Sociais:** Medir o "grau de separação" entre duas pessoas.

- **Grafo (ou Dígrafo):** $G = (V, E)$
- **V (Vértices):** O conjunto de "nós" (ex: cidades, roteadores).
- **E (Arestas):** O conjunto de "conexões" (ex: ruas, cabos).
- **Grafo Ponderado:** Cada aresta $(i, j) \in E$ possui um **peso** (ou custo) w_{ij} associado.



O Problema de Caminhos Mínimos

- **Caminho:** Uma sequência de arestas que conecta um nó origem s a um nó destino t .
 - Ex: $P = (A \rightarrow B \rightarrow D)$
- **Custo do Caminho:** A soma dos pesos w_{ij} das arestas que formam o caminho.
 - Ex: $Custo(P) = w_{AB} + w_{BD}$

O Grande Objetivo

Dado um grafo ponderado G , um nó origem s e um nó destino t , encontrar o caminho P de s para t que **minimiza o custo total**.

$$P^* = \arg \min_{P: s \rightarrow t} \sum_{(i,j) \in P} w_{ij}$$

Observação Importante

Os algoritmos que vamos estudar (Dijkstra, Bellman-Ford) resolvem o problema de **Origem Única (SSSP)**: encontram o caminho mínimo de s para **TODOS** os outros



Variáveis de Decisão

Definimos x_{ij} como a quantidade de fluxo enviado pela aresta (i, j) .

- $x_{ij} = 1$ se a aresta (i, j) faz parte do caminho mínimo.
- $x_{ij} = 0$ caso contrário.

Função Objetivo (Minimizar o Custo Total)

Queremos minimizar a soma dos custos de todas as arestas usadas. O custo de usar a aresta (i, j) é w_{ij} .

$$\min Z = \sum_{(i,j) \in E} w_{ij} \cdot x_{ij}$$

Modelo PL: Restrições de Conservação de Fluxo

Para cada nó k no grafo, o fluxo que "entra" deve ser igual ao fluxo que "sai", com exceção da origem e do destino.

1. Nó Origem (s):

- O nó s **envia 1** unidade de fluxo.
- Fluxo que Sai - Fluxo que Entra = 1

$$\sum_{j:(s,j) \in E} x_{sj} - \sum_{i:(i,s) \in E} x_{is} = 1$$

2. Nó Destino (t):

- O nó t **recebe 1** unidade de fluxo.
- Fluxo que Sai - Fluxo que Entra = -1

$$\sum_{j:(t,j) \in E} x_{tj} - \sum_{i:(i,t) \in E} x_{it} = -1$$

3. Nós Intermediários ($k \neq s, t$):

- Tudo que entra deve sair (conservação pura).
- Fluxo que Sai - Fluxo que Entra = 0

$$\sum_{j:(k,j) \in E} x_{kj} - \sum_{i:(i,k) \in E} x_{ik} = 0$$

4. Restrição de Não-Negatividade

Não podemos enviar fluxo negativo.

$$x_{ij} \geq 0 \quad \forall (i,j) \in E$$

- **Pergunta:** Nós não deveríamos forçar x_{ij} a ser um número inteiro (0 ou 1)?

$$x_{ij} \in \{0, 1\} \quad (\text{Programação Inteira})$$

- **Resposta:** Não precisa!
- Este tipo de problema de rede (Fluxo de Custo Mínimo) tem uma propriedade especial chamada **Total Unimodularidade**.
- **O que importa:** Essa propriedade **garante** que, se a oferta (1) e a demanda (-1) são inteiras, a solução ótima do problema de Programação Linear (com $x_{ij} \geq 0$) **já será inteira**.
- Por isso, podemos resolvê-lo eficientemente como um PL comum.

- Embora o modelo de PL seja academicamente correto, ele não é a forma mais eficiente de resolver o Problema de Caminhos Mínimos na prática.
- Na prática, usamos algoritmos especializados em grafos que são muito mais rápidos.
- Vamos focar em dois algoritmos fundamentais:
 - ① **Algoritmo de Dijkstra**
 - ② **Algoritmo de Bellman-Ford**

O Conceito

Uma abordagem "Gulosa" (Greedy). A cada passo, ele escolhe o caminho que *parece* ser o melhor (o mais curto) e expande a partir dele.

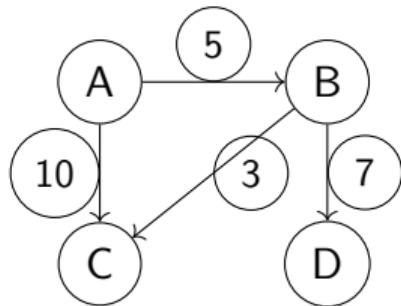
Restrição Crítica

O Algoritmo de Dijkstra **NÃO FUNCIONA** se o grafo tiver arestas com **pesos negativos**.

• Ideia Principal:

- Mantém um conjunto de nós "visitados" e as distâncias até eles.
- Começa na origem s (distância 0) e todos os outros com distância ∞ .
- Em cada passo, visita o nó **não-visitado** com a **menor distância** atual.
- Ao visitar um nó, "relaxa" as arestas para seus vizinhos (tenta encontrar um caminho mais curto para eles).

Exemplo Prático: Algoritmo de Dijkstra



Origem: A

Passo a Passo (Distâncias):

Nó Visitado	$d(A)$	$d(B)$	$d(C)$	$d(D)$
Inicial	0	∞	∞	∞
A	0	5	10	∞
B	0	5	8	12
C	0	5	8	12
D	0	5	8	12

- **Caminho Mínimo $A \rightarrow C$: $A \rightarrow B \rightarrow C$**
(Custo 8)
- **Caminho Mínimo $A \rightarrow D$: $A \rightarrow B \rightarrow D$**
(Custo 12)

O Conceito

Uma abordagem de **Programação Dinâmica**. Em vez de ser "guloso", ele é "pessimista" e re-calcula tudo várias vezes.

Principais Vantagens

- **Funciona com pesos negativos.**
 - Consegue **detectar ciclos negativos** (um loop no grafo que, se percorrido, reduz o custo infinitamente).
-
- **Ideia Principal (Parte 1):**
 - Repete o processo de "relaxar" **TODAS** as arestas do grafo, um total de $|V| - 1$ vezes (onde $|V|$ é o número de nós).

- **Ideia Principal (Parte 2):**

- **Por que $|V| - 1$ vezes?** Porque o maior caminho mínimo possível *sem* um ciclo só pode ter, no máximo, $|V| - 1$ arestas.
- Após as $|V| - 1$ repetições, o algoritmo faz uma **última checagem** (uma $|V|$ -ésima passagem).
- Se ainda for possível "relaxar" alguma aresta nessa última passagem, o algoritmo para e avisa: "**Ciclo Negativo Detectado!**"

Aplicação de Ciclo Negativo

Isso é usado em finanças para detectar **arbitragem**: trocar Moeda A → Moeda B → Moeda C → Moeda A e sair com mais dinheiro do que começou.

Comparativo: Dijkstra vs. Bellman-Ford

Característica	Algoritmo de Dijkstra	Algoritmo de Bellman-Ford
Abordagem	Gulosa (Greedy)	Programação Dinâmica
Pesos Negativos?	Não (falha)	Sim
Ciclos Negativos?	Não (não detecta)	Sim (detecta)
Complexidade	$O(E \log V)$ ou $O(V^2)$	$O(V \cdot E)$
Quando usar?	Grafos sem pesos negativos (Ex: GPS, redes OSPF)	Grafos com pesos negativos (Ex: Detecção de arbitragem)

Conclusão

A escolha do algoritmo depende da natureza do problema. Dijkstra é mais rápido, mas Bellman-Ford é mais robusto e "seguro" para grafos que podem ter custos negativos.

- BAZARAA, M.S., JARVIS, J.J. e SHERALI, H.D. *Linear Programming and Network Flows*, John Wiley & Sons, 1990.
- GOLDBARG, M. C. e LUNA, H. P. L., *Otimização Combinatória e Programação Linear: Modelos e Algoritmos*, Campus, 2000.
- ARENALES, M. N.; et al. *Pesquisa operacional*. 2. ed. Rio de Janeiro: Elsevier, c2015.
- BELFIORE, Patrícia; FÁVERO, Luiz Paulo. *Pesquisa operacional para cursos de engenharia*. Rio de Janeiro: Elsevier Brasil, 2013.