Trabalho 2 - Problema dos Caminhos Mínimos

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

Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciência e Tecnologia

12 de novembro de 2025

unesp

Sumário

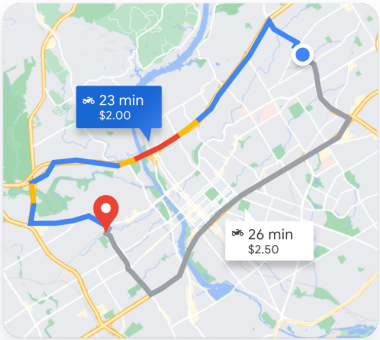
unesp

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.

unesp

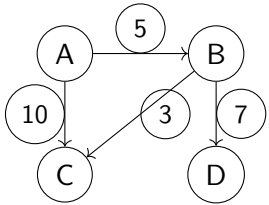
Onde o PCM se Esconde?

- **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.

unesp

Traduzindo o Mapa para a Matemática

- **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.



unesp

O Problema de Caminhos Mínimos

- **Caminho:** Uma sequência de arestas que conecta um nó origem s a um nó destino t.
  - Ex: $P = (A \to B \to D)$
- **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**.

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

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

Modelo PL: A Abordagem Genérica

- **Ideia:** Modelar como um **Problema de Fluxo** para enviar 1 unidade da origem s ao destino t.

Variáveis de Decisão

$x_{ij} = 1$ se o arco (i, j) for usado no caminho, 0 caso contrário.

Função Objetivo (Minimizar o Custo Total)

Minimizar a soma dos custos de todas as arestas usadas:

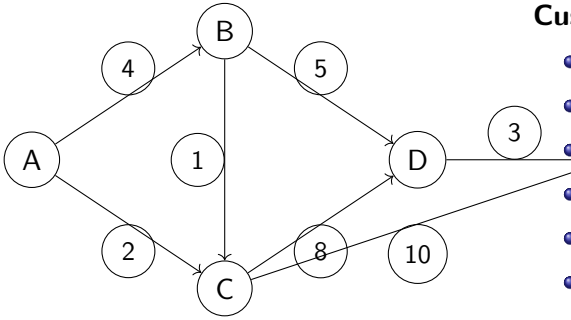
$$\min Z = \sum_{(i,j) \in E} w_{ij} \cdot x_{ij}$$

Restrições (Conservação de Fluxo)

- **Origem (s):** Fluxo que Sai $= 1$
- **Intermediário (k):** Fluxo que Entra $=$ Fluxo que Sai
- **Destino (t):** Fluxo que Entra $= 1$

Exemplo de Modelagem: Grafo Logístico

- Vamos modelar o problema de achar a rota de custo mínimo da Origem **A** para o Destino **E**.



Custos (w_{ij}):

- $w_{AB} = 4$
- $w_{AC} = 2$
- $w_{BC} = 1$
- $w_{BD} = 5$
- $w_{CD} = 8$
- $w_{CE} = 10$
- $w_{DE} = 3$

unesp

Exemplo de Modelagem: O PPL Completo

Função Objetivo (Minimizar Z):

$$\min Z = 4x_{AB} + 2x_{AC} + 1x_{BC} + 5x_{BD} + 8x_{CD} + 10x_{CE} + 3x_{DE}$$

Sujeito a (Conservação de Fluxo):

$$\begin{aligned}
(x_{AB} + x_{AC}) - 0 &= 1 && \text{(Nó A: Origem)} \\
(x_{BC} + x_{BD}) - x_{AB} &= 0 && \text{(Nó B: Intermediário)} \\
(x_{CD} + x_{CE}) - (x_{AC} + x_{BC}) &= 0 && \text{(Nó C: Intermediário)} \\
(x_{DE}) - (x_{BD} + x_{CD}) &= 0 && \text{(Nó D: Intermediário)} \\
0 - (x_{CE} + x_{DE}) &= -1 && \text{(Nó E: Destino)} \\
x_{ij} &\geq 0
\end{aligned}$$

unesp

Modelo PL: A Mágica da Integridade

- **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**.

unesp

- 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**

unesp

Algoritmo de Dijkstra

O Conceito

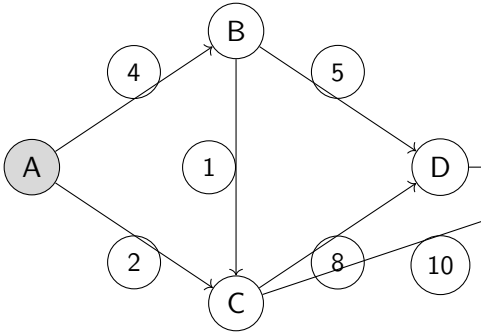
Uma abordagem "Gulosa"(Greedy). A cada passo, ele escolhe o caminho que *parece* ser o melhor (o mais curto) e expande a partir dele.

Restrição Crítica

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

unesp

Exercício: Algoritmo de Dijkstra (Inicialização)



extbfNós Visitados: {}

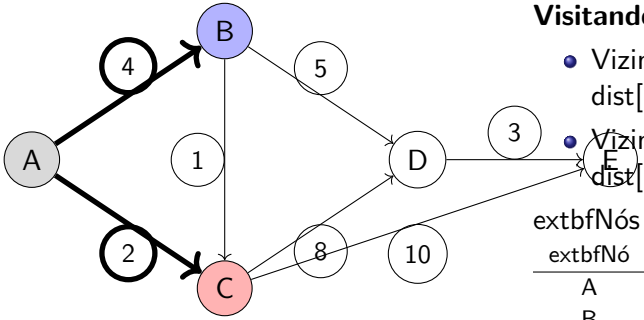
extbfNó	**dist[i]**	**prev[i]**
A	**0**	-
B	∞	-
C	∞	-
D	∞	-
E	∞	-

(custo 0)

Próximo nó: A

unesp

Exercício: Dijkstra - Iteração 1 (Nó A)



Visitando Nó: A (dist=0)

- Vizinho B: $0 + 4 = 4$. (Atualiza dist[B]=4, prev[B]=A)
- Vizinho C: $0 + 2 = 2$. (Atualiza dist[C]=2, prev[C]=A)

extbfNós Visitados: {A}

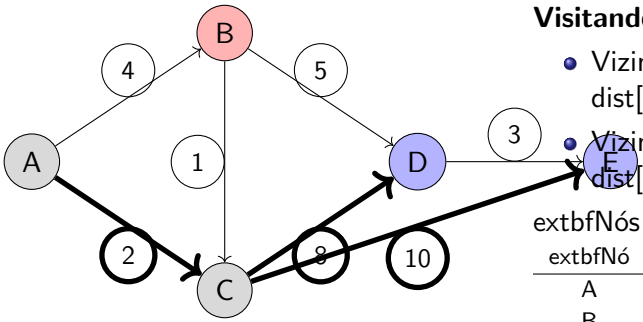
extbfNó	**dist[i]**	**prev[i]**
A	0	-
B	**4**	A
C	**2**	A
D	∞	-
E	∞	-

Próximo nó: C

(menor dist=2)

unesp

Exercício: Dijkstra - Iteração 2 (Nó C)



Visitando Nó: C (dist=2)

- Vizinho D: $2 + 8 = 10$. (Atualiza dist[D]=10, prev[D]=C)
- Vizinho E: $2 + 10 = 12$. (Atualiza dist[E]=12, prev[E]=C)

extbfNós Visitados: {A, C}

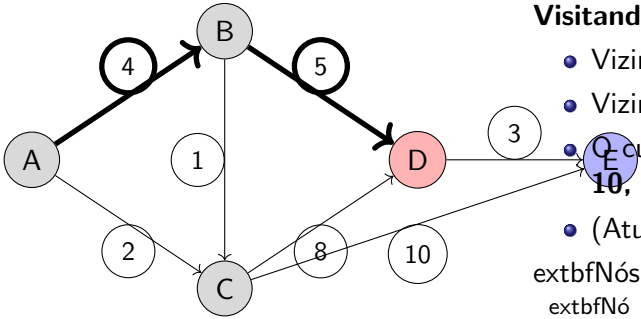
extbfNó	**dist[i]**	**prev[i]**
A	0	-
B	4	A
C	2	A
D	**10**	C
E	**12**	C

Próximo nó: B

(menor dist=4)

unesp

Exercício: Dijkstra - Iteração 3 (Nó B) - A "Relaxação"



Visitando Nó: B (dist=4)

- Vizinho C: Já visitado.
- Vizinho D: $4 + 5 = 9$.
- O custo atual de D é 10. **Como 9 ¡ 10, atualizamos!**
- (Atualiza dist[D]=9, prev[D]=B)

extbfNós Visitados: {A, C, B}

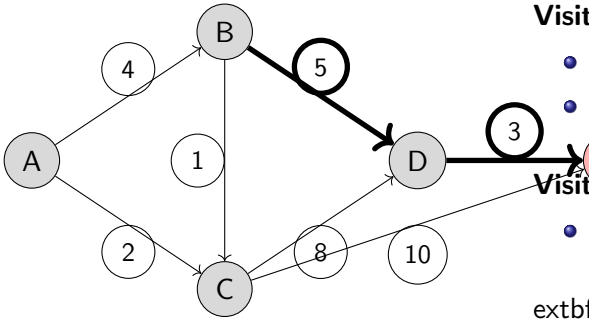
extbfNó	**dist[i]**	**prev[i]**
A	0	-
B	4	A
C	2	A
D	**9**	**B**
E	12	C

Próximo nó: D

(menor dist=9)

unesp

Exercício: Dijkstra - Iterações 4 e 5 (Nós D, E)



Visitando Nó: D (dist=9)

- Vizinho E: $9 + 3 = 12$.
- O custo atual de E é 12. Como $12 \not< 12$, **não há atualização**.

Visitando Nó: E (dist=12)

- Fim do algoritmo (destino alcançado / fila vazia).

extbfNós Visitados: {A, C, B, D, E}

extbfNó	**dist[i]**	**prev[i]**
A	0	-
B	4	A
C	2	A
D	9	B
E	12	C

unesp

Exercício: Dijkstra - Conclusão e Caminho

Resultado Final (Tabela)

A tabela 'dist' nos dá o custo mínimo de A para *todos* os outros nós:

- Custo para B: 4
- Custo para C: 2
- Custo para D: 9
- Custo para E: 12

Encontrando o Caminho (Backtracking)

Mas qual é o caminho de custo 12? Usamos a tabela 'prev' de trás para frente:

1. Começamos em **E**.
2. Quem é o anterior (prev) de E? É o nó **C**. (Caminho: ... $\rightarrow C \rightarrow E$)
3. Quem é o anterior (prev) de C? É o nó **A**. (Caminho: ... $\rightarrow A \rightarrow C \rightarrow E$)
4. Chegamos na Origem (A). Fim.

Algoritmo de Bellman-Ford

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).

unesp

- **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 $\rightarrow$ Moeda B $\rightarrow$ Moeda C $\rightarrow$ Moeda A e sair com mais dinheiro do que começou.

unesp

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.

unesp

Referências Bibliográficas