Aplicação de Teoria dos Grafos ao Problema do Caixeiro Viajante Métrico

Combinando

Sabrina Teodoro

Agosto 2020

Motivação

Motivação

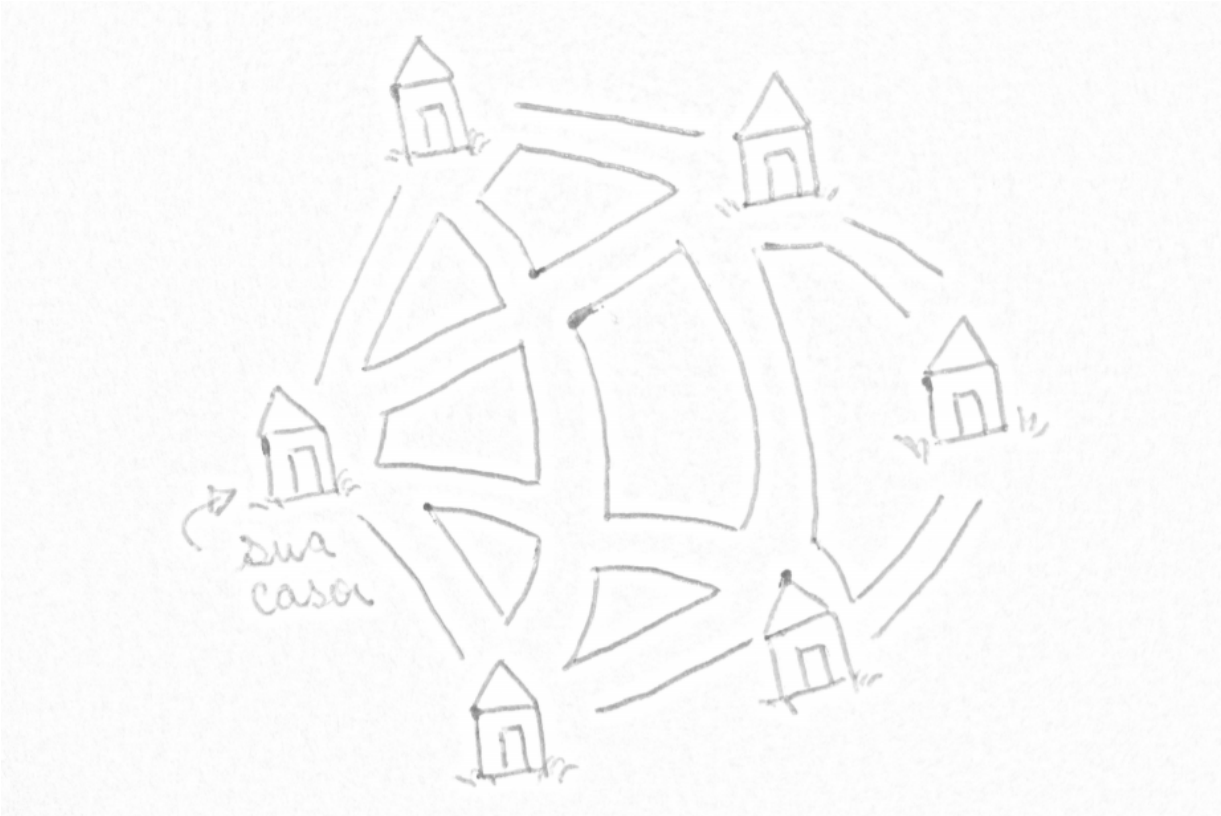
Suponha que você queira visitar todos os seus amigos no pós-pandemia.

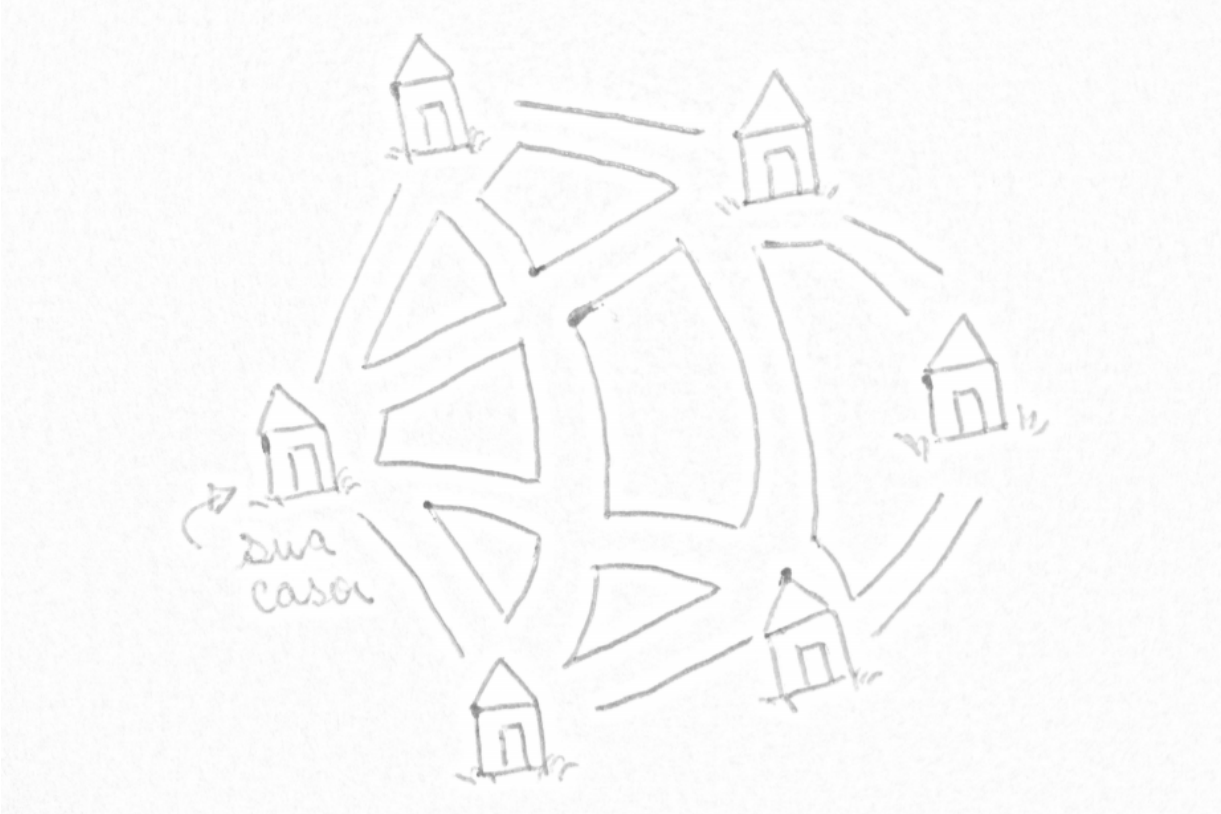
Motivação

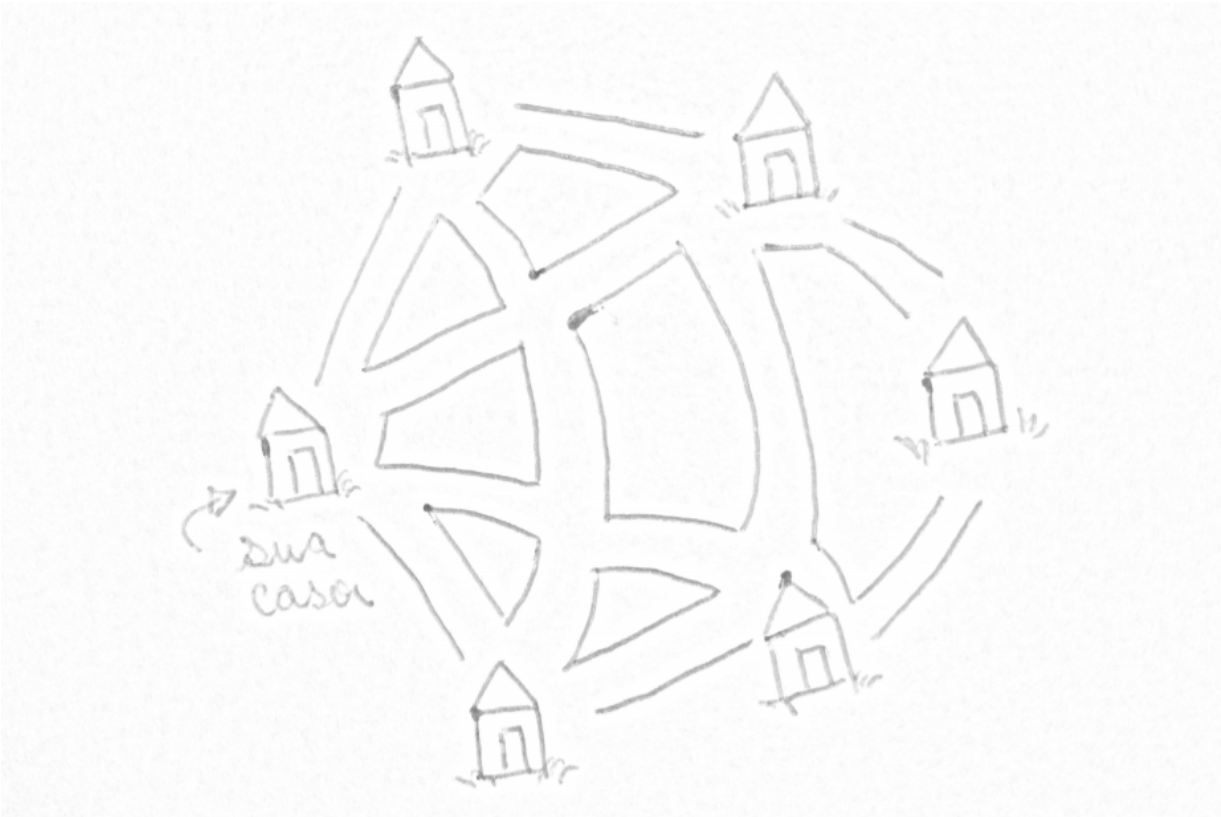
Suponha que você queira visitar todos os seus amigos no pós-pandemia. Eis um trecho do mapa com as casas deles e a sua, e os trajetos não-direcionados entre elas:

Motivação

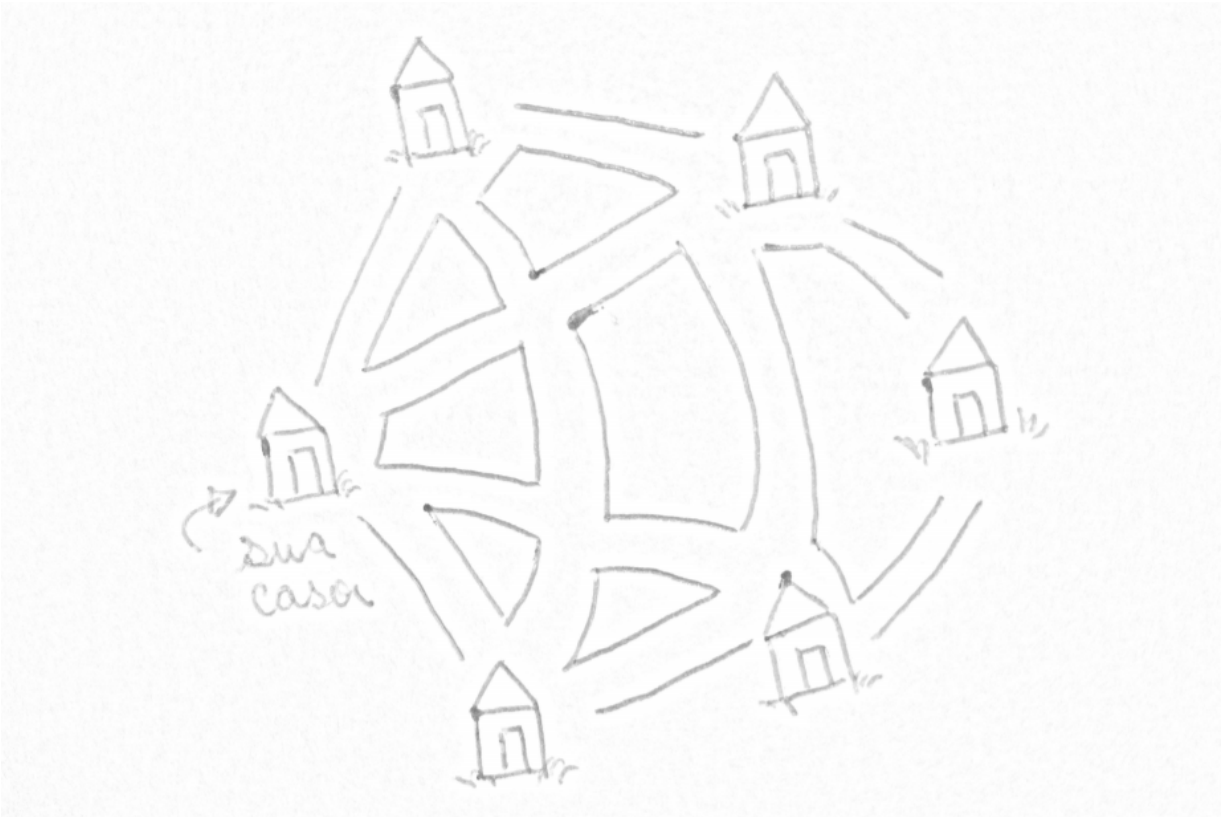
Suponha que você queira visitar todos os seus amigos no pós-pandemia. Eis um trecho do mapa com as casas deles e a sua, e os trajetos não-direcionados entre elas:




sua
casa



Cada trajeto consome determinado tempo para ser percorrido. Digamos que você não queira repetir trajetos e deseje ver cada amigo uma única vez. Você começa e termina a visitação em sua própria casa.



Cada trajeto consome determinado tempo para ser percorrido. Digamos que você não queira repetir trajetos e deseje ver cada amigo uma única vez. Você começa e termina a visitação em sua própria casa.

Como fazer isso o mais rápido possível?

Otimização

A motivação anterior é um problema clássico da **Otimização Combinatória**.

Otimização

A motivação anterior é um problema clássico da **Otimização Combinatória**.

Combinatória porque combinamos os trajetos possíveis para obter o roteiro completo,

Otimização

A motivação anterior é um problema clássico da **Otimização Combinatória**.

Combinatória porque combinamos os trajetos possíveis para obter o roteiro completo, e otimização porque procuramos a visitação mais rápida.

Otimização é sobre isso: dentre possíveis soluções para um mesmo problema, buscar as melhores, e "melhor" depende.

Otimização

A motivação anterior é um problema clássico da **Otimização Combinatória**.

Combinatória porque combinamos os trajetos possíveis para obter o roteiro completo, e otimização porque procuramos a visitação mais rápida.

Otimização é sobre isso: dentre possíveis soluções para um mesmo problema, buscar as melhores, e "melhor" depende.

Em geral, um problema de otimização propõe que se minimize o valor de uma **função objetivo** satisfazendo certas **restrições**.

Otimização

A motivação anterior é um problema clássico da **Otimização Combinatória**.

Combinatória porque combinamos os trajetos possíveis para obter o roteiro completo, e otimização porque procuramos a visitação mais rápida.

Otimização é sobre isso: dentre possíveis soluções para um mesmo problema, buscar as melhores, e "melhor" depende.

Em geral, um problema de otimização propõe que se minimize o valor de uma **função objetivo** satisfazendo certas **restrições**.
Em nosso problema, a função é o tempo total da visitação e uma das restrições é visitar cada amigo uma única vez.

O valor da função objetivo varia de acordo com os **custos** e com as **variáveis** do problema.

O valor da função objetivo varia de acordo com os **custos** e com as **variáveis** do problema.
Em nosso caso, a função objetivo consiste no somatório dos produtos entre custo e valor das variáveis, sendo que cada variável é um trajeto, e os custos, os respectivos tempos consumidos.

O valor da função objetivo varia de acordo com os **custos** e com as **variáveis** do problema.
Em nosso caso, a função objetivo consiste no somatório dos produtos entre custo e valor das variáveis, sendo que cada variável é um trajeto, e os custos, os respectivos tempos consumidos.

Uma **solução** para um dado problema consiste num conjunto de escolhas acerca das variáveis.

O valor da função objetivo varia de acordo com os **custos** e com as **variáveis** do problema.
Em nosso caso, a função objetivo consiste no somatório dos produtos entre custo e valor das variáveis, sendo que cada variável é um trajeto, e os custos, os respectivos tempos consumidos.

Uma **solução** para um dado problema consiste num conjunto de escolhas acerca das variáveis.
Aqui nossa escolha é binária: percorrer ou não um trajeto.

O valor da função objetivo varia de acordo com os **custos** e com as **variáveis** do problema.
Em nosso caso, a função objetivo consiste no somatório dos produtos entre custo e valor das variáveis, sendo que cada variável é um trajeto, e os custos, os respectivos tempos consumidos.

Uma **solução** para um dado problema consiste num conjunto de escolhas acerca das variáveis.
Aqui nossa escolha é binária: percorrer ou não um trajeto.

Quando (e se) obtemos um conjunto de escolhas que resulta no custo mínimo da função objetivo, temos uma **solução ótima** com **custo ótimo**.

O valor da função objetivo varia de acordo com os **custos** e com as **variáveis** do problema.
Em nosso caso, a função objetivo consiste no somatório dos produtos entre custo e valor das variáveis, sendo que cada variável é um trajeto, e os custos, os respectivos tempos consumidos.

Uma **solução** para um dado problema consiste num conjunto de escolhas acerca das variáveis.
Aqui nossa escolha é binária: percorrer ou não um trajeto.

Quando (e se) obtemos um conjunto de escolhas que resulta no custo mínimo da função objetivo, temos uma **solução ótima** com **custo ótimo**.
Note que o custo ótimo é único, mas podem existir várias soluções que o gerem.

Teoria dos grafos

Teoria dos grafos

Podemos modelar (e resolver) nosso problema com um pouco de Teoria dos Grafos.

Teoria dos grafos

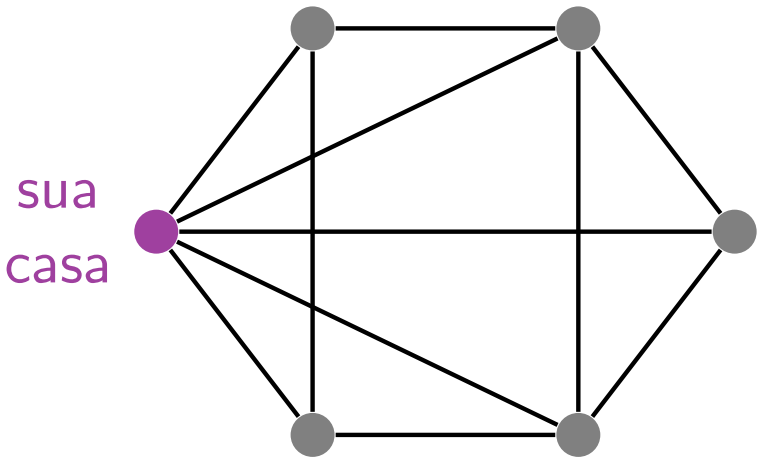
Podemos modelar (e resolver) nosso problema com um pouco de Teoria dos Grafos.

Basta considerar um grafo $G = (V, A)$, em que os vértices em V são casas e as arestas em A são trajetos possíveis entre duas casas,

Teoria dos grafos

Podemos modelar (e resolver) nosso problema com um pouco de Teoria dos Grafos.

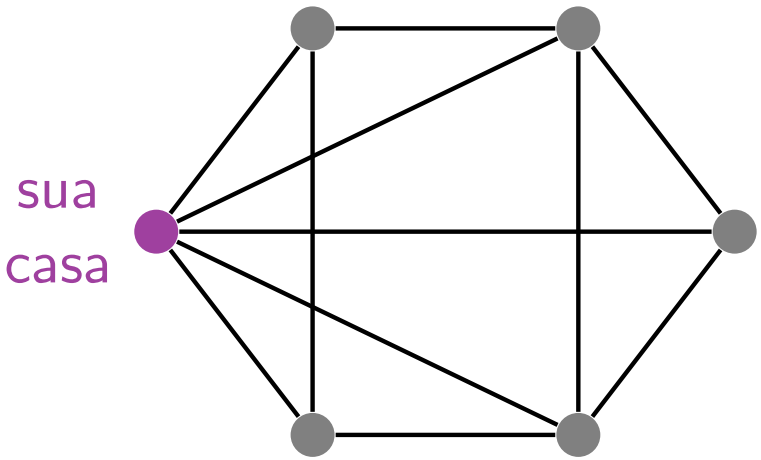
Basta considerar um grafo $G = (V, A)$, em que os vértices em V são casas e as arestas em A são trajetos possíveis entre duas casas, e um custo c_a para cada aresta $a \in A$, correspondente ao tempo demandado pelo trajeto.



Teoria dos grafos

Podemos modelar (e resolver) nosso problema com um pouco de Teoria dos Grafos.

Basta considerar um grafo $G = (V, A)$, em que os vértices em V são casas e as arestas em A são trajetos possíveis entre duas casas, e um custo c_a para cada aresta $a \in A$, correspondente ao tempo demandado pelo trajeto.



Buscamos um circuito (não há repetição de vértices nem de arestas) no grafo que inclua todos vértices e acumule o menor custo.

Mais teoria dos grafos

Mais teoria dos grafos

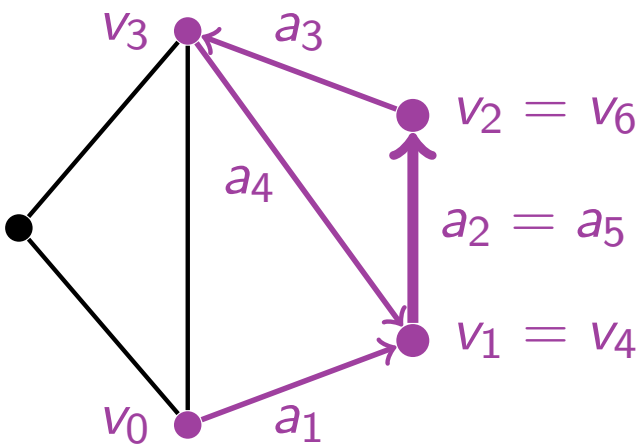
Um **passeio** é uma sequência $(v_0, a_1, v_1, a_2, v_2, \cdots, a_k, v_k)$ em que v_i são vértices e cada a_i é uma aresta com extremos v_{i-1} e v_i.

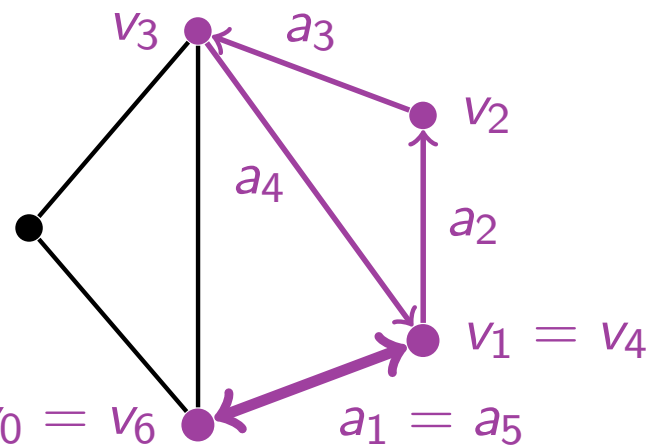
Mais teoria dos grafos

Um **passeio** é uma sequência $(v_0, a_1, v_1, a_2, v_2, \cdots, a_k, v_k)$ em que v_i são vértices e cada a_i é uma aresta com extremos v_{i-1} e v_i.
Pode ser representado pela sequência de vértices $(v_0, v_1, \cdots, v_k)$.

Mais teoria dos grafos

Um **passeio** é uma sequência $(v_0, a_1, v_1, a_2, v_2, \cdots, a_k, v_k)$ em que v_i são vértices e cada a_i é uma aresta com extremos v_{i-1} e v_i.
Pode ser representado pela sequência de vértices $(v_0, v_1, \cdots, v_k)$.
O passeio é **fechado** se $v_k = v_0$.

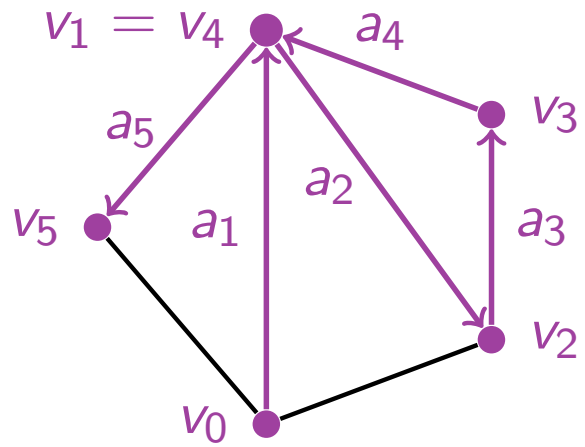




Mais teoria dos grafos

Um **passeio** é uma sequência $(v_0, a_1, v_1, a_2, v_2, \cdots, a_k, v_k)$ em que v_i são vértices e cada a_i é uma aresta com extremos v_{i-1} e v_i.
Pode ser representado pela sequência de vértices $(v_0, v_1, \cdots, v_k)$.
O passeio é **fechado** se $v_k = v_0$.

Uma **trilha** é um passeio cujas arestas são distintas duas a duas.

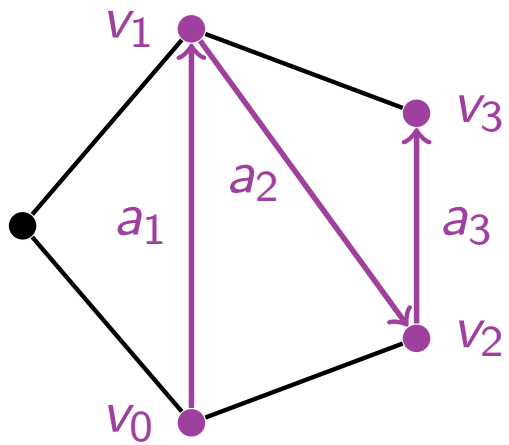


Mais teoria dos grafos

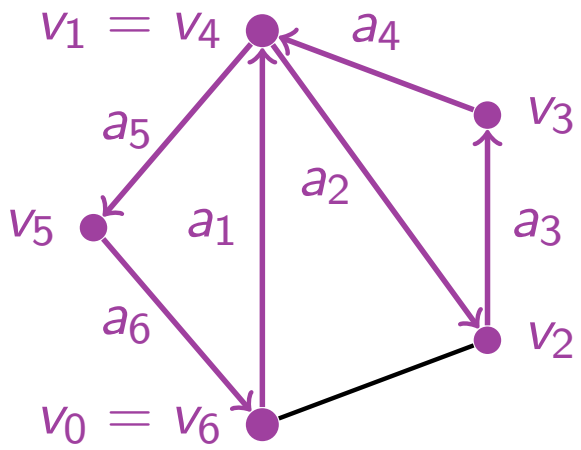
Um **passeio** é uma sequência $(v_0, a_1, v_1, a_2, v_2, \cdots, a_k, v_k)$ em que v_i são vértices e cada a_i é uma aresta com extremos v_{i-1} e v_i.
Pode ser representado pela sequência de vértices $(v_0, v_1, \cdots, v_k)$.
O passeio é **fechado** se $v_k = v_0$.

Uma **trilha** é um passeio cujas arestas são distintas duas a duas.

Um **caminho** é um passeio cujos vértices são distintos dois a dois.

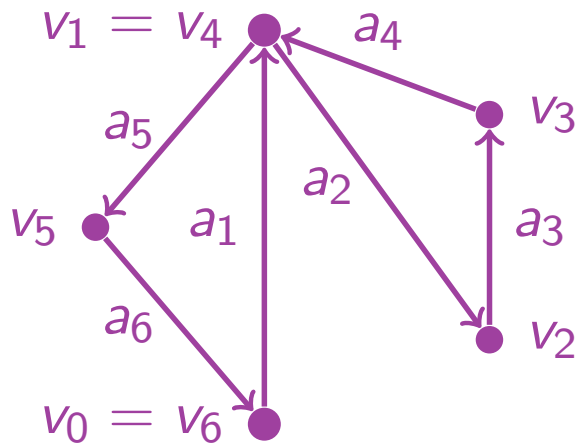


Um **ciclo** é uma trilha fechada com pelo menos três arestas.



Um **ciclo** é uma trilha fechada com pelo menos três arestas.

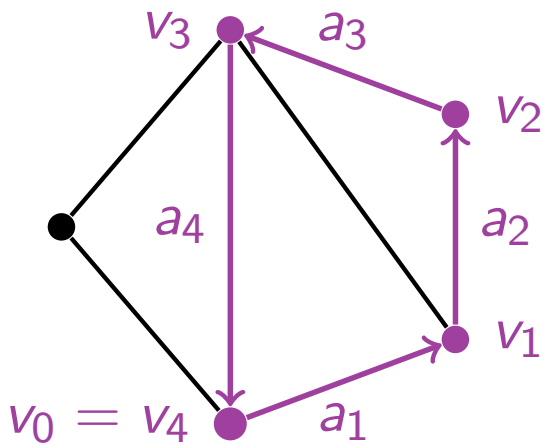
Um ciclo é **euleriano** se contém todas as arestas do grafo.



Um **ciclo** é uma trilha fechada com pelo menos três arestas.

Um ciclo é **euleriano** se contém todas as arestas do grafo.

Um **circuito** é um ciclo $(v_0, v_1, \cdots, v_k)$ tal que $v_1 \neq v_j$ se $i \neq j$ exceto se $i = 0$ e $j = k$.

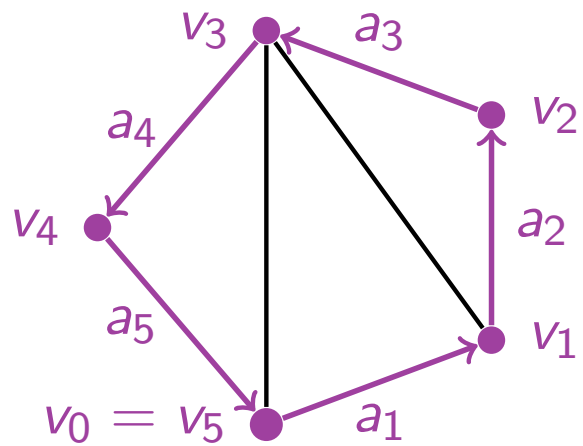


Um **ciclo** é uma trilha fechada com pelo menos três arestas.

Um ciclo é **euleriano** se contém todas as arestas do grafo.

Um **circuito** é um ciclo $(v_0, v_1, \cdots, v_k)$ tal que $v_1 \neq v_j$ se $i \neq j$ exceto se $i = 0$ e $j = k$.

Um circuito é **hamiltoniano** se contém todos os vértices do grafo.



Caixeiro viajante

O problema inicial pode ser generalizado e formalizado como

Caixeiro viajante

O problema inicial pode ser generalizado e formalizado como

TSP (Traveling Salesperson Problem)

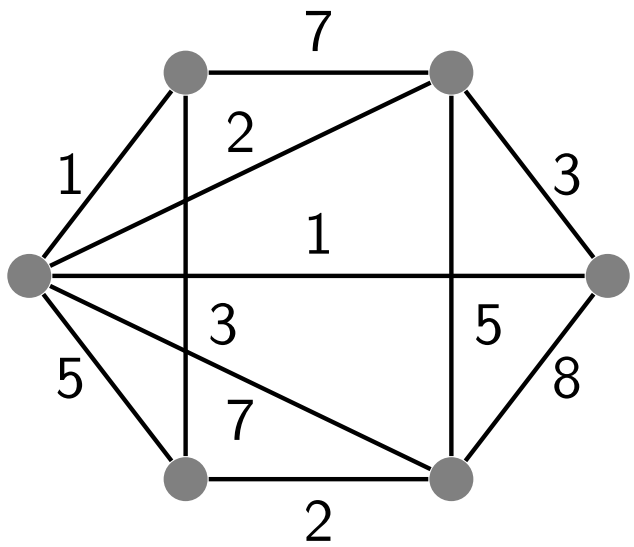
Dados um grafo G e um custo c_a em $\mathbb{Q}_{\geq 0}$ para cada aresta a, determinar um circuito hamiltoniano H que minimize $c(H)$.

Caixeiro viajante

O problema inicial pode ser generalizado e formalizado como

TSP (Traveling Salesperson Problem)

Dados um grafo G e um custo c_a em $\mathbb{Q}_{\geq 0}$ para cada aresta a, determinar um circuito hamiltoniano H que minimize $c(H)$.

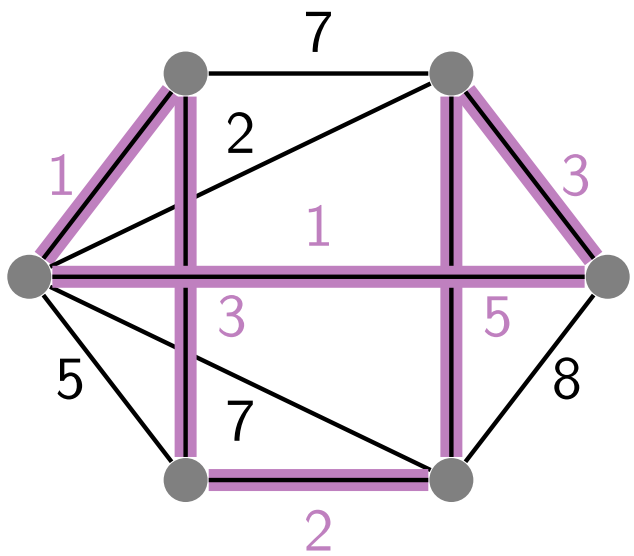


Caixeiro viajante

O problema inicial pode ser generalizado e formalizado como

TSP (Traveling Salesperson Problem)

Dados um grafo G e um custo c_a em $\mathbb{Q}_{\geq 0}$ para cada aresta a, determinar um circuito hamiltoniano H que minimize $c(H)$.

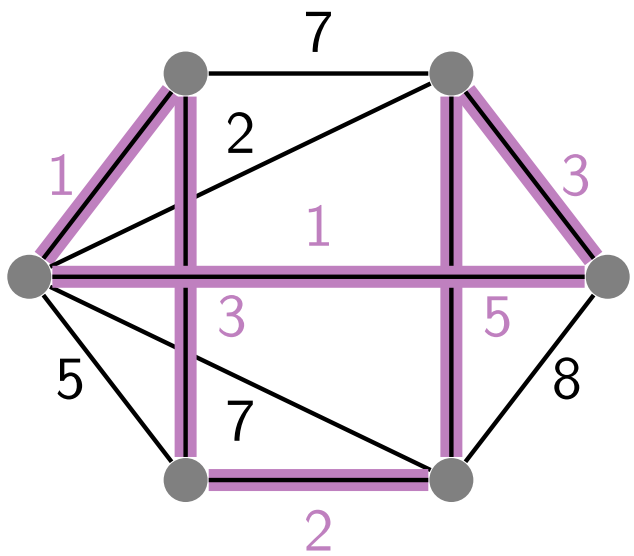


Caixeiro viajante

O problema inicial pode ser generalizado e formalizado como

TSP (Traveling Salesperson Problem)

Dados um grafo G e um custo c_a em $\mathbb{Q}_{\geq 0}$ para cada aresta a, determinar um circuito hamiltoniano H que minimize $c(H)$.



O TSP é muito conhecido devido a suas várias aplicações em problemas práticos e por ser **NP-díficil**.

Solução quase ótima

Na busca de uma solução para um problema de otimização, existem três categorias principais de algoritmos para implementar

Solução quase ótima

Na busca de uma solução para um problema de otimização, existem três categorias principais de algoritmos para implementar

- **exato:** encontra uma solução ótima;

Solução quase ótima

Na busca de uma solução para um problema de otimização, existem três categorias principais de algoritmos para implementar

- **exato:** encontra uma solução ótima;
- **de aproximação:** encontra uma solução sub-ótima com um limite no grau de sub-otimalidade e leva tempo polinomial;

Solução quase ótima

Na busca de uma solução para um problema de otimização, existem três categorias principais de algoritmos para implementar

- **exato:** encontra uma solução ótima;
- **de aproximação:** encontra uma solução sub-ótima com um limite no grau de sub-otimalidade e leva tempo polinomial;
- **heurístico:** encontra uma solução sub-ótima sem garantia de qualidade.

Solução quase ótima

Na busca de uma solução para um problema de otimização, existem três categorias principais de algoritmos para implementar

- **exato:** encontra uma solução ótima;
- **de aproximação:** encontra uma solução sub-ótima com um limite no grau de sub-otimalidade e leva tempo polinomial;
- **heurístico:** encontra uma solução sub-ótima sem garantia de qualidade.

Há casos, como o do TSP, em que o algoritmo exato demora demais, então optamos pelo algoritmo de aproximação, que garante certa proximidade entre o ótimo e a solução obtida.

Solução quase ótima

Na busca de uma solução para um problema de otimização, existem três categorias principais de algoritmos para implementar

- **exato:** encontra uma solução ótima;
- **de aproximação:** encontra uma solução sub-ótima com um limite no grau de sub-otimalidade e leva tempo polinomial;
- **heurístico:** encontra uma solução sub-ótima sem garantia de qualidade.

Há casos, como o do TSP, em que o algoritmo exato demora demais, então optamos pelo algoritmo de aproximação, que garante certa proximidade entre o ótimo e a solução obtida.
Para o TSP, contudo, ainda não é conhecido um algoritmo de aproximação com **razão constante**. Mas tal algoritmo existe para um caso particular do problema.

Grafo métrico

Um grafo é **completo** se quaisquer dois de seus vértices são adjacentes.

Grafo métrico

Um grafo é **completo** se quaisquer dois de seus vértices são adjacentes.

Desigualdade triangular: "Mais fácil ir reto do que dar a volta"

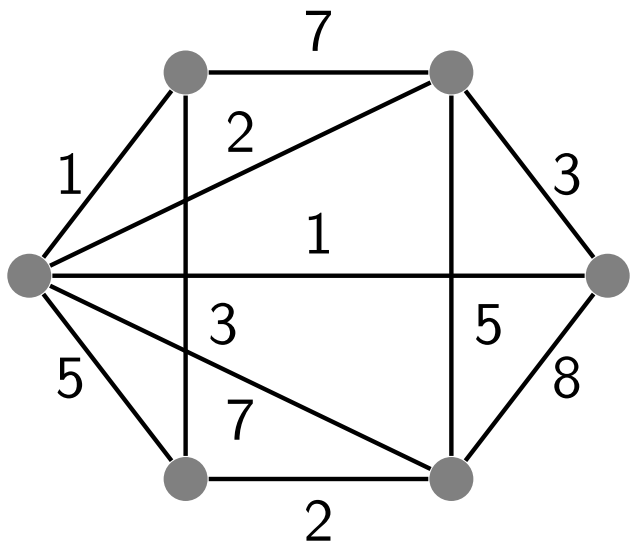
$$c_{ik} \leq c_{ij} + c_{jk} \text{ para quaisquer } i, j, k$$

Grafo métrico

Um grafo é **completo** se quaisquer dois de seus vértices são adjacentes.

Desigualdade triangular: "Mais fácil ir reto do que dar a volta"

$$c_{ik} \leq c_{ij} + c_{jk} \text{ para quaisquer } i, j, k$$

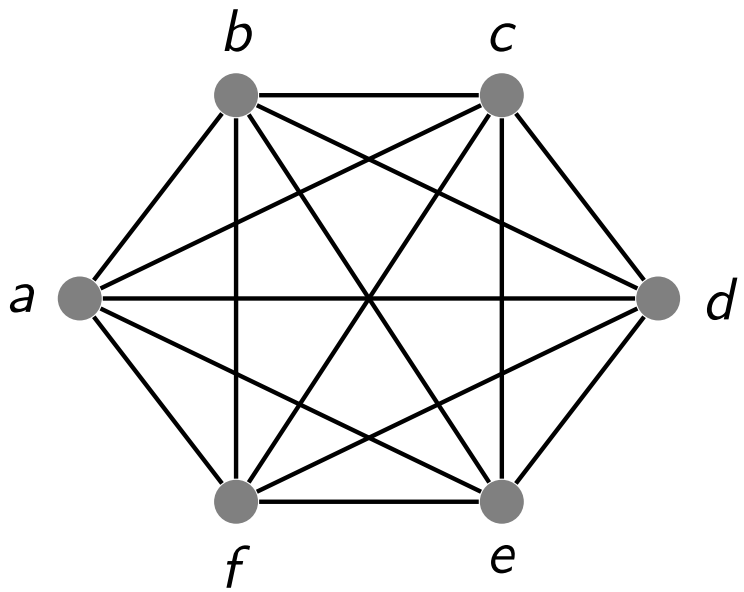


Grafo métrico

Um grafo é **completo** se quaisquer dois de seus vértices são adjacentes.

Desigualdade triangular: "Mais fácil ir reto do que dar a volta"

$$c_{ik} \leq c_{ij} + c_{jk} \text{ para quaisquer } i, j, k$$



	a	b	c	d	e	f
a	-	1	4	7	8	5
b	1	-	3	6	8	6
c	4	3	-	3	5	8
d	7	6	3	-	2	7
e	8	8	5	2	-	5
f	5	6	8	7	5	-

Caixeiro viajante métrico

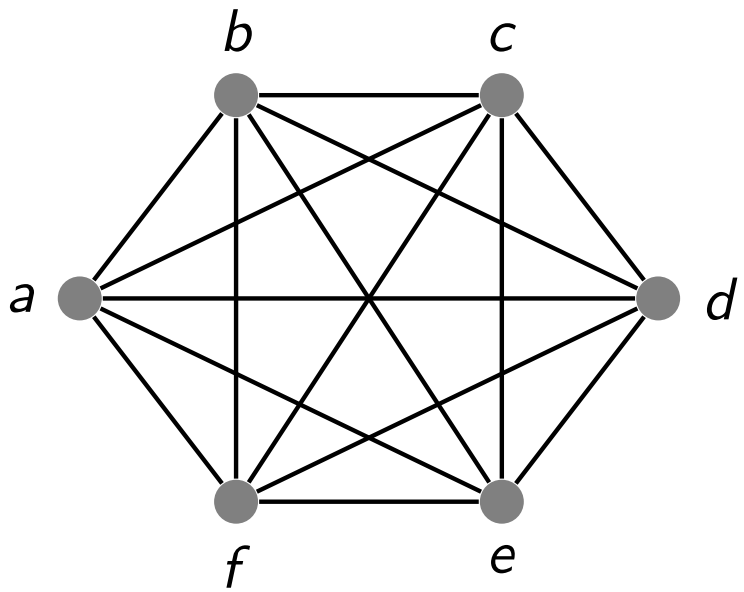
TSPM, *metric* TSP, delta-TSP

TSP restrito ao conjunto de instâncias (G, c) em que o grafo G é completo e os custos c satisfazem a desigualdade triangular.

Caixeiro viajante métrico

TSPM, *metric* TSP, delta-TSP

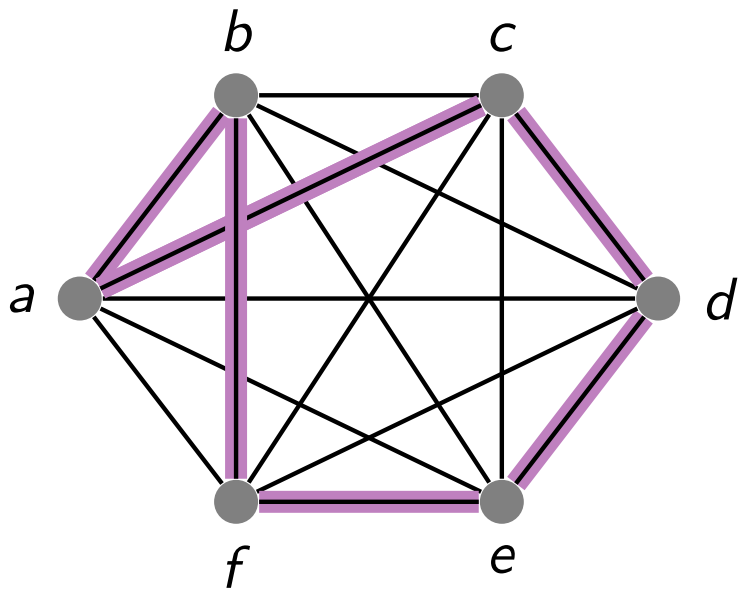
TSP restrito ao conjunto de instâncias (G, c) em que o grafo G é completo e os custos c satisfazem a desigualdade triangular.



	a	b	c	d	e	f
a	-	1	3	7	8	5
b	1	-	4	6	8	6
c	3	4	-	4	5	8
d	7	6	4	-	2	7
e	8	8	5	2	-	5
f	5	6	8	7	5	-

TSPM, *metric* TSP, delta-TSP

TSP restrito ao conjunto de instâncias (G, c) em que o grafo G é completo e os custos c satisfazem a desigualdade triangular.

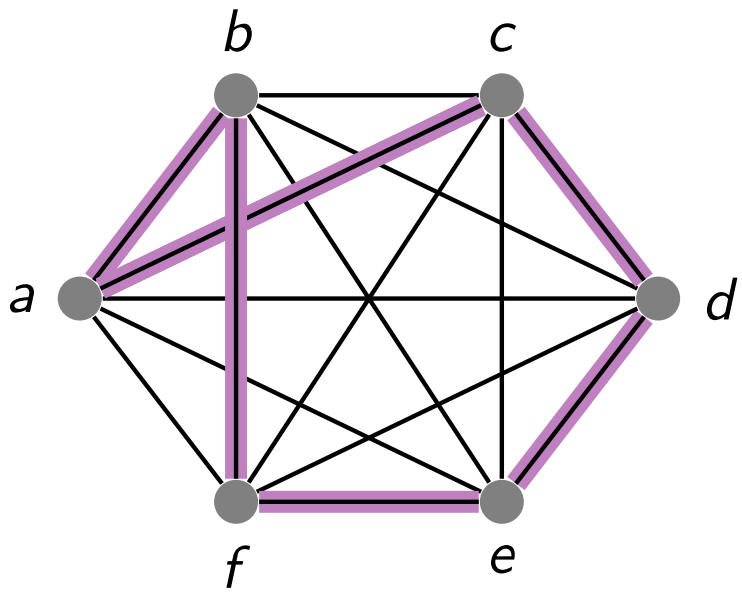


	a	b	c	d	e	f
a	-	1	3	7	8	5
b	1	-	4	6	8	6
c	3	4	-	4	5	8
d	7	6	4	-	2	7
e	8	8	5	2	-	5
f	5	6	8	7	5	-

Caixeiro viajante métrico

TSPM, *metric* TSP, delta-TSP

TSP restrito ao conjunto de instâncias (G, c) em que o grafo G é completo e os custos c satisfazem a desigualdade triangular.



	a	b	c	d	e	f
a	-	1	3	7	8	5
b	1	-	4	6	8	6
c	3	4	-	4	5	8
d	7	6	4	-	2	7
e	8	8	5	2	-	5
f	5	6	8	7	5	-

O TSPM também é NP-difícil.

Roteiro

Algoritmo de Christofides

Roteiro

Algoritmo de Christofides

- Algoritmo de Kruskal
  - Árvore geradora mínima

Roteiro

Algoritmo de Christofides

- Algoritmo de Kruskal
  - Árvore geradora mínima
- Algoritmo de Edmonds
  - Emparelhamento perfeito mínimo

Roteiro

Algoritmo de Christofides

- Algoritmo de Kruskal
  - Árvore geradora mínima
- Algoritmo de Edmonds
  - Emparelhamento perfeito mínimo
- Algoritmo de Hierholzer
  - Ciclo euleriano

Roteiro

Algoritmo de Christofides

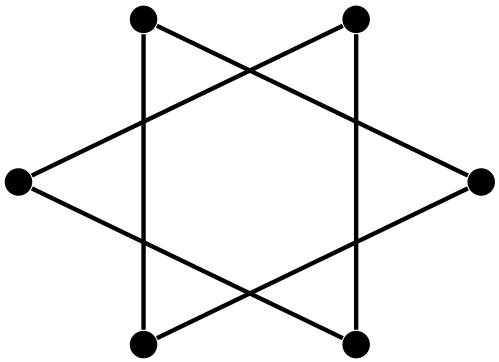
- Algoritmo de Kruskal
  - Árvore geradora mínima
- Algoritmo de Edmonds
  - Emparelhamento perfeito mínimo
- Algoritmo de Hierholzer
  - Ciclo euleriano
- Algoritmo atalho

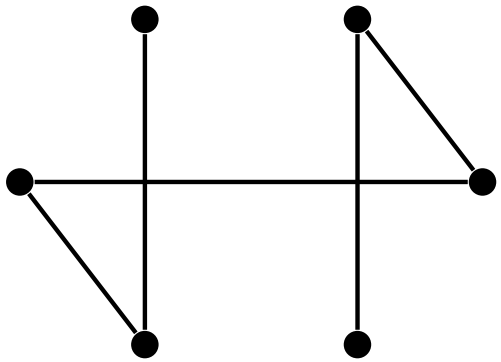
Árvore geradora mínima

Uma grafo é **conexo** se para qualquer par u, v de seus vértices existe um caminhos com extremos u e v. Todo grafo completo é conexo.

Árvore geradora mínima

Uma grafo é **conexo** se para qualquer par u, v de seus vértices existe um caminhos com extremos u e v. Todo grafo completo é conexo.





Árvore geradora mínima

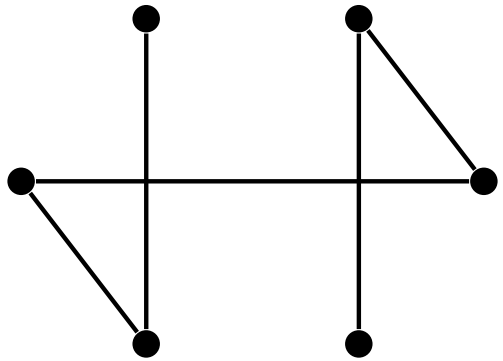
Uma grafo é **conexo** se para qualquer par u, v de seus vértices existe um caminhos com extremos u e v. Todo grafo completo é conexo.

Uma **árvore** é um grafo conexo sem circuitos.
Um grafo conexo com n vértice é uma árvore se, e somente se, tem $n - 1$ arestas.

Árvore geradora mínima

Uma grafo é **conexo** se para qualquer par u, v de seus vértices existe um caminhos com extremos u e v. Todo grafo completo é conexo.

Uma **árvore** é um grafo conexo sem circuitos.
Um grafo conexo com n vértice é uma árvore se, e somente se, tem $n - 1$ arestas.



Árvore geradora mínima

Uma grafo é **conexo** se para qualquer par u, v de seus vértices existe um caminhos com extremos u e v. Todo grafo completo é conexo.

Uma **árvore** é um grafo conexo sem circuitos.
Um grafo conexo com n vértice é uma árvore se, e somente se, tem $n-1$ arestas.

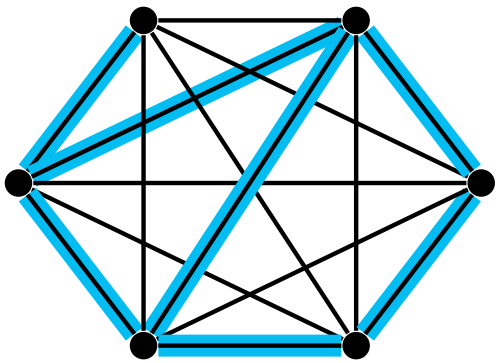
Um subgrafo $G' = (V', A') \subseteq G = (V, A)$ é **gerador** de G se $V' = V$.

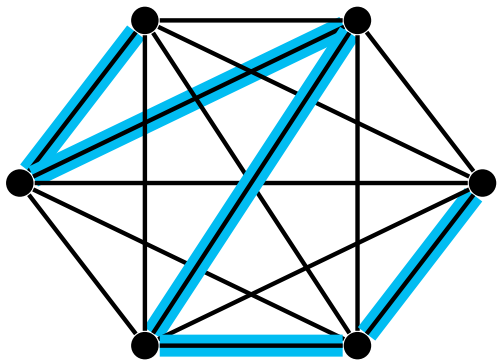
Árvore geradora mínima

Uma grafo é **conexo** se para qualquer par u, v de seus vértices existe um caminhos com extremos u e v. Todo grafo completo é conexo.

Uma **árvore** é um grafo conexo sem circuitos.
Um grafo conexo com n vértice é uma árvore se, e somente se, tem $n-1$ arestas.

Um subgrafo $G' = (V', A') \subseteq G = (V, A)$ é **gerador** de G se $V' = V$.





Árvore geradora mínima

Uma grafo é **conexo** se para qualquer par u, v de seus vértices existe um caminhos com extremos u e v. Todo grafo completo é conexo.

Uma **árvore** é um grafo conexo sem circuitos.
Um grafo conexo com n vértice é uma árvore se, e somente se, tem $n-1$ arestas.

Um subgrafo $G' = (V', A') \subseteq G = (V, A)$ é **gerador** de G se $V' = V$.

MST (Minimmum Spanning Tree)

Dado um grafo G e um custo c_a em $\mathbb{Q}_{\geq 0}$ para cada aresta, determinar uma árvore T geradora de G que minimize $c(T)$.

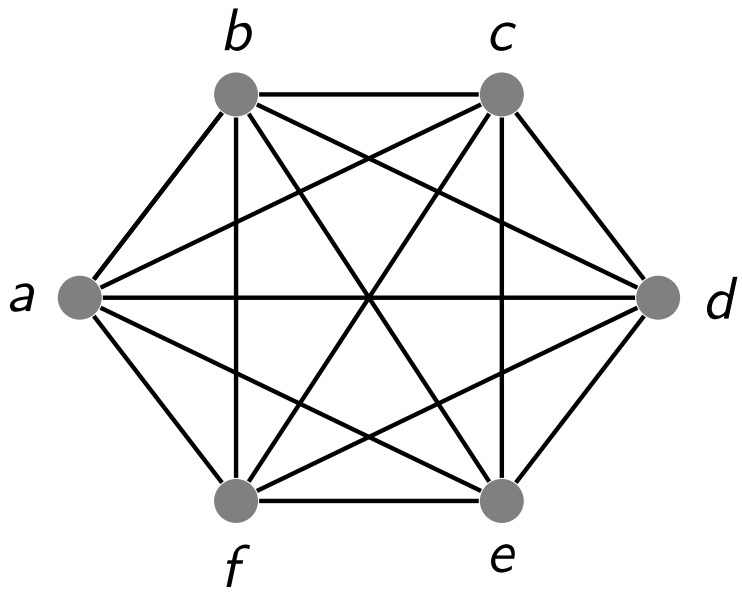
Algoritmo de Kruskal

Publicado em 1956 pelo matemático Joseph Kruskal.

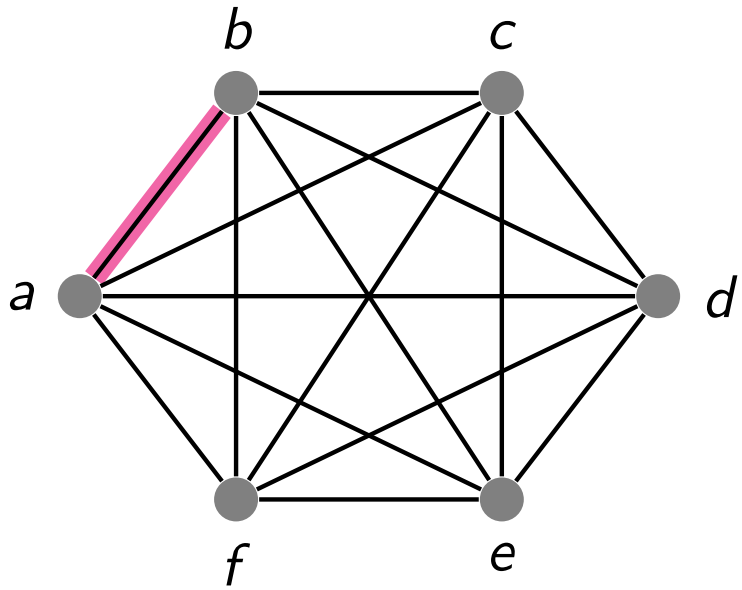
Algoritmo 1: KRUSKAL(G, c)

```
1  faça T ← ∅;
2  faça A' ← A;
3  faça cont ← 0;
4  enquanto A' ≠ ∅ ou cont ≠ n − 1, faça:
5     se a ∈ A' é tal que c_a ≤ c_a' para todo a' ∈ A', então:
6        A' ← A'\{a}
7        se T ∪ {a} não contém ciclos, então:
8           T ← T ∪ {a};
9           cont ← cont + 1;
10 devolva T;
```

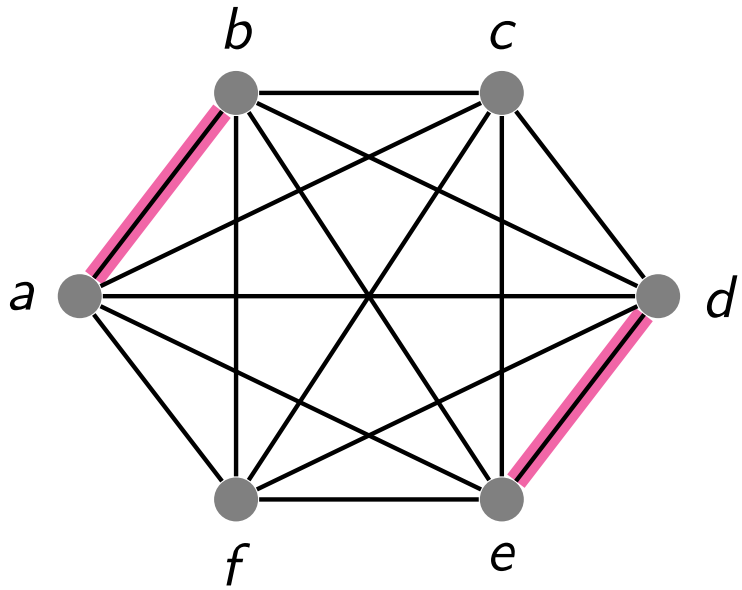
Ideia geral: escolher as arestas da árvore geradora seguindo a ordem crescente dos custos, exceto se a aresta em questão formar um ciclo.



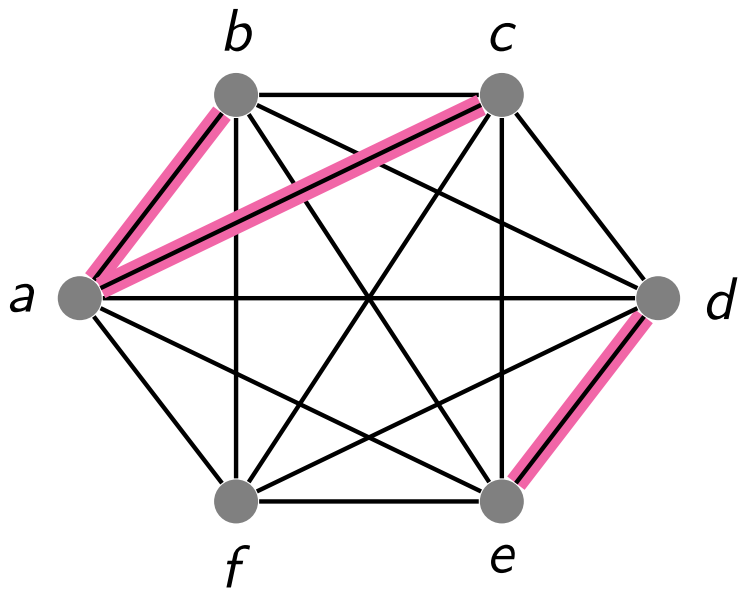
	a	b	c	d	e	f
a	-	1	3	7	8	5
b	1	-	4	6	8	6
c	3	4	-	4	5	8
d	7	6	4	-	2	7
e	8	8	5	2	-	5
f	5	6	8	7	5	-



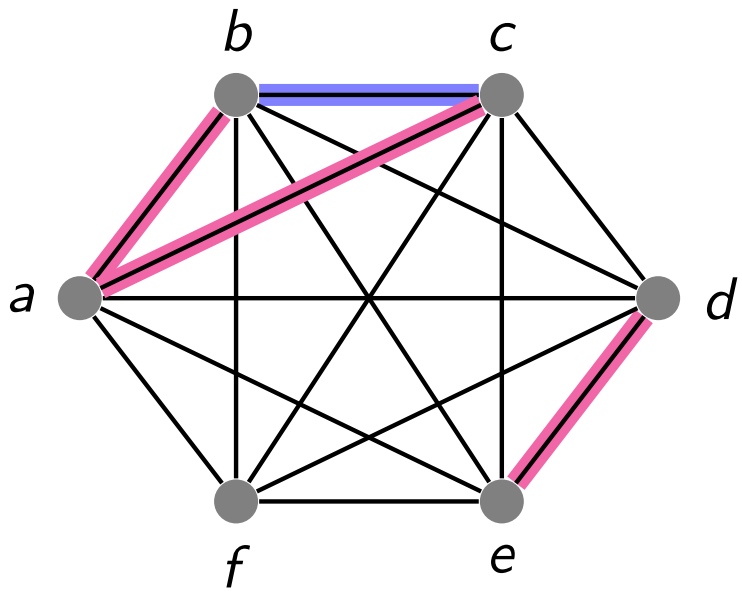
	a	b	c	d	e	f
a	-	1	3	7	8	5
b	1	-	4	6	8	6
c	3	4	-	4	5	8
d	7	6	4	-	2	7
e	8	8	5	2	-	5
f	5	6	8	7	5	-



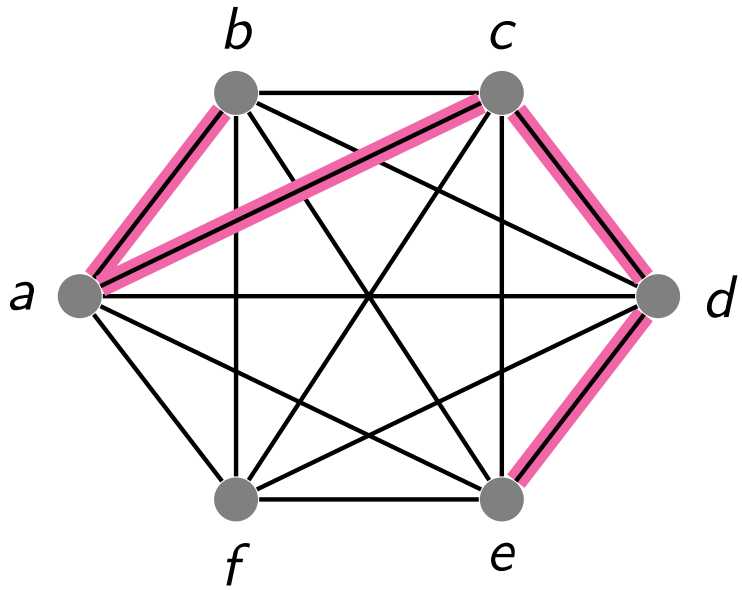
	a	b	c	d	e	f
a	-	1	3	7	8	5
b	1	-	4	6	8	6
c	3	4	-	4	5	8
d	7	6	4	-	2	7
e	8	8	5	2	-	5
f	5	6	8	7	5	-



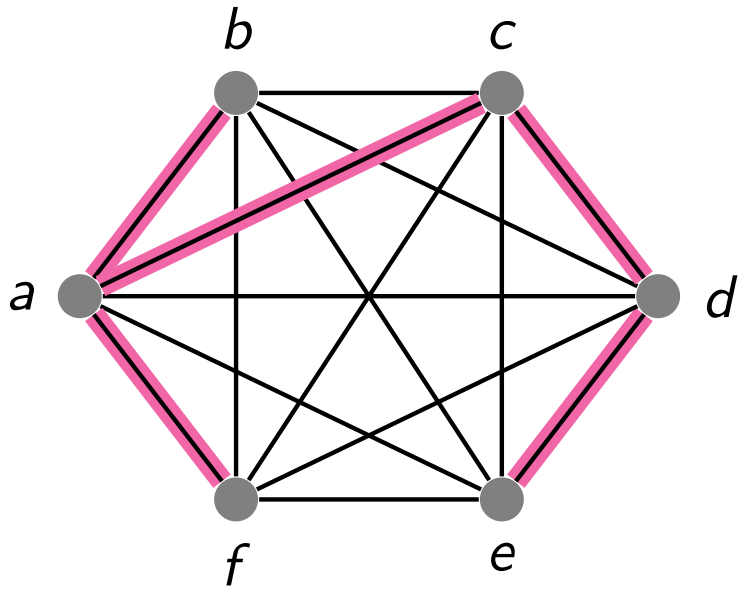
	a	b	c	d	e	f
a	-	1	3	7	8	5
b	1	-	4	6	8	6
c	3	4	-	4	5	8
d	7	6	4	-	2	7
e	8	8	5	2	-	5
f	5	6	8	7	5	-



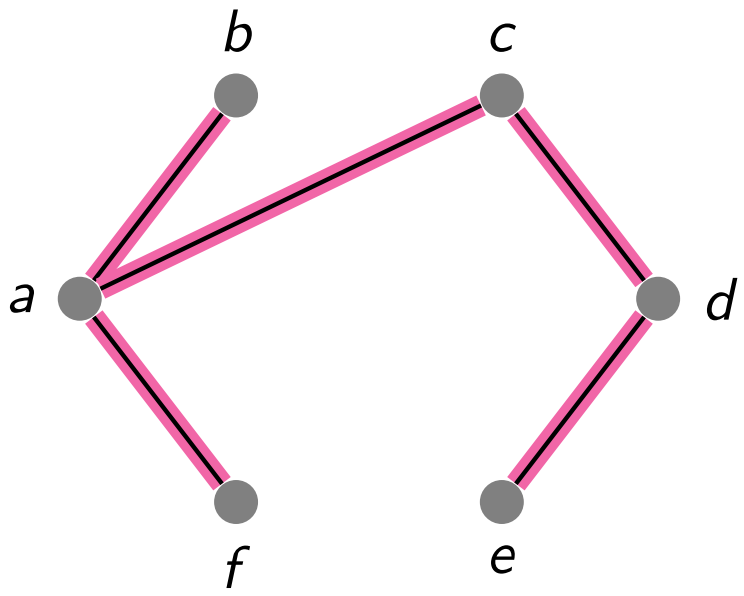
	a	b	c	d	e	f
a	-	1	3	7	8	5
b	1	-	4	6	8	6
c	3	4	-	4	5	8
d	7	6	4	-	2	7
e	8	8	5	2	-	5
f	5	6	8	7	5	-



	a	b	c	d	e	f
a	-	1	3	7	8	5
b	1	-	4	6	8	6
c	3	4	-	4	5	8
d	7	6	4	-	2	7
e	8	8	5	2	-	5
f	5	6	8	7	5	-

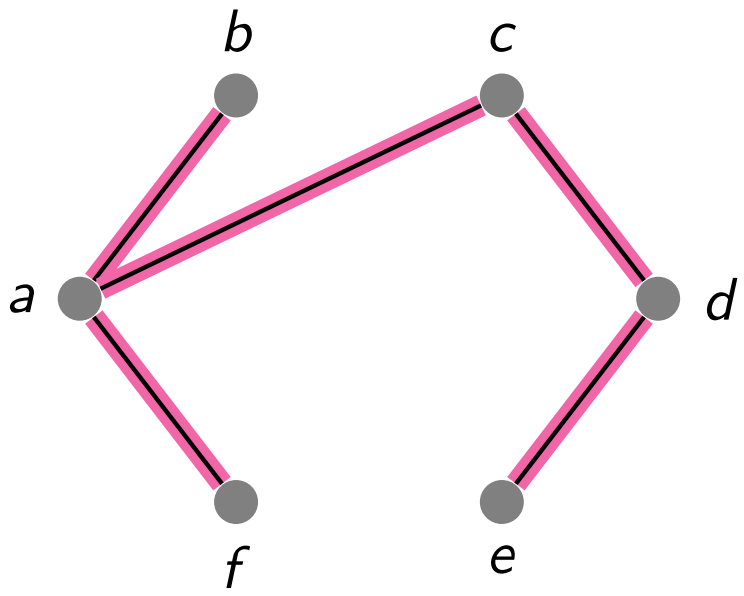


	a	b	c	d	e	f
a	-	1	3	7	8	5
b	1	-	4	6	8	6
c	3	4	-	4	5	8
d	7	6	4	-	2	7
e	8	8	5	2	-	5
f	5	6	8	7	5	-



	a	b	c	d	e	f
a		1	3			5
b	1					
c	3			4		
d			4		2	
e				2		
f	5					

$$c(T) = 15$$

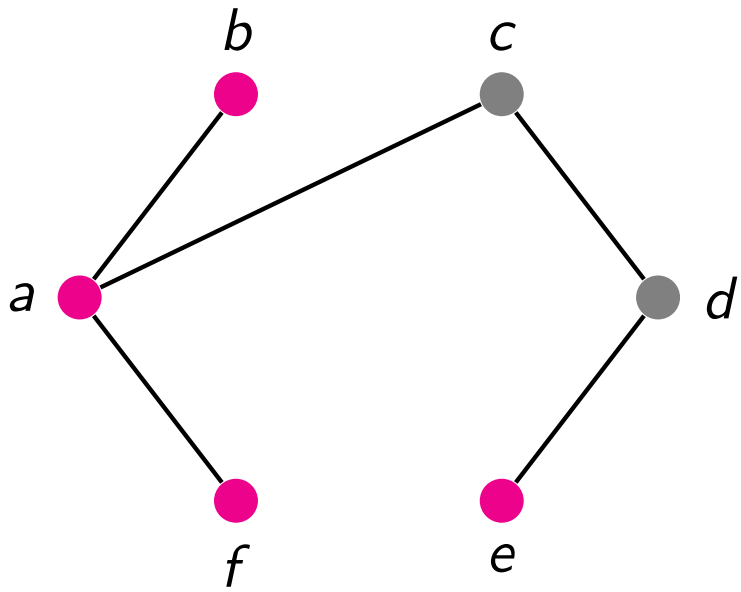


	a	b	c	d	e	f
a		1	3			5
b	1					
c	3			4		
d			4		2	
e				2		
f	5					

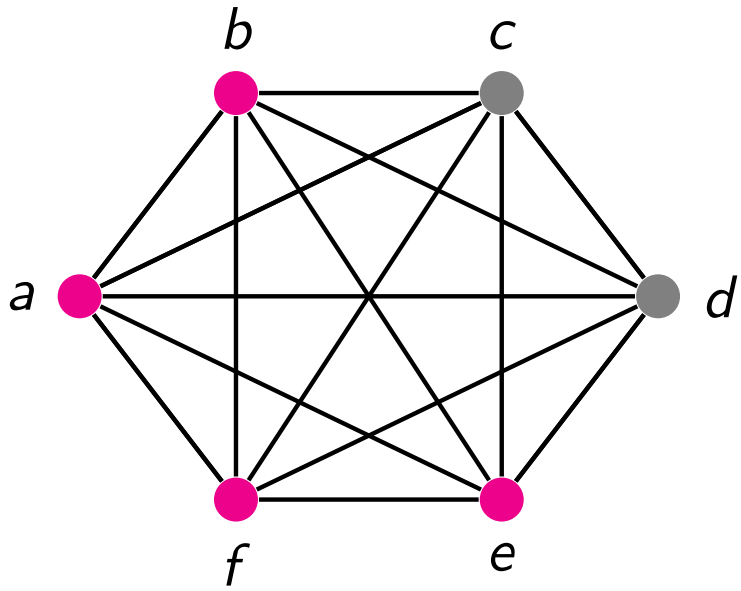
$$c(T) = 15$$

Kruskal's Algorithm Animation - How does it progress?

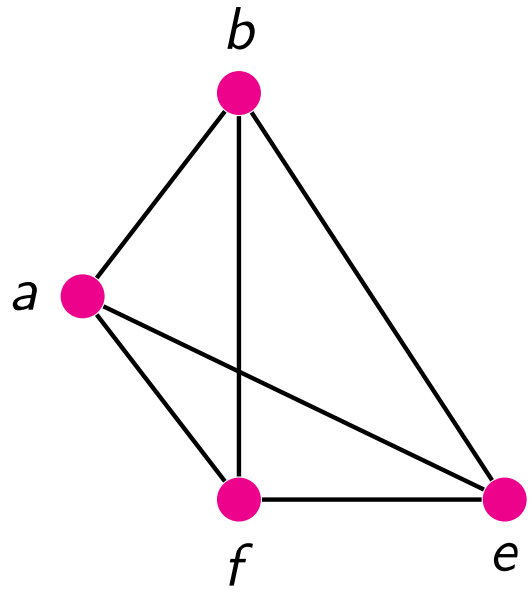
Seja I o conjunto dos vértices de grau ímpar em T.



Considere o grafo induzido $G[I]$, isto é, o grafo (I, B) em que B é o conjunto de todas as arestas de G com ambos os extremos em I.



Considere o grafo induzido $G[I]$, isto é, o grafo (I, B) em que B é o conjunto de todas as arestas de G com ambos os extremos em I.

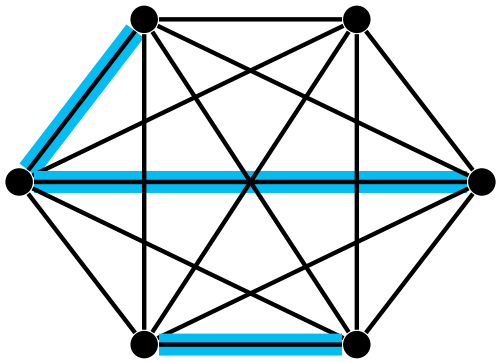


Emparelhamento perfeito mínimo

Um **emparelhamento** (=*matching*) é um conjunto M de arestas duas a duas não adjacentes.

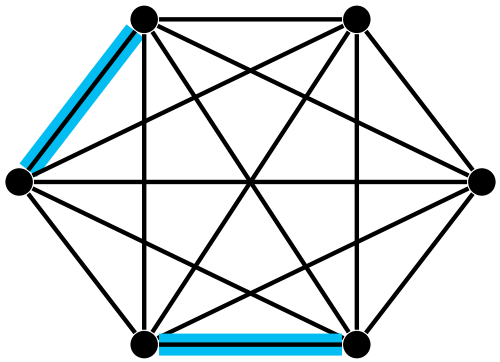
Emparelhamento perfeito mínimo

Um **emparelhamento** (=*matching*) é um conjunto M de arestas duas a duas não adjacentes.



Emparelhamento perfeito mínimo

Um **emparelhamento** (=*matching*) é um conjunto M de arestas duas a duas não adjacentes.



Emparelhamento perfeito mínimo

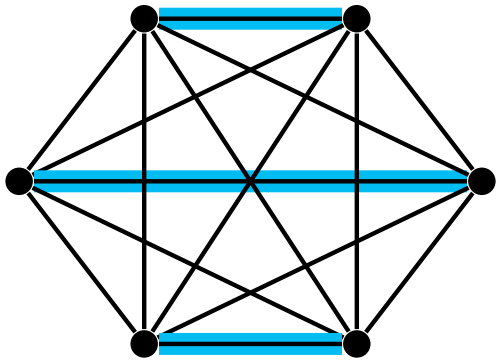
Um **emparelhamento** (=*matching*) é um conjunto M de arestas duas a duas não adjacentes.

Um emparelhamento M é **perfeito** se todo vértice do grafo é extremo de uma (única) aresta de M.

Emparelhamento perfeito mínimo

Um **emparelhamento** (=*matching*) é um conjunto M de arestas duas a duas não adjacentes.

Um emparelhamento M é **perfeito** se todo vértice do grafo é extremo de uma (única) aresta de M.



Emparelhamento perfeito mínimo

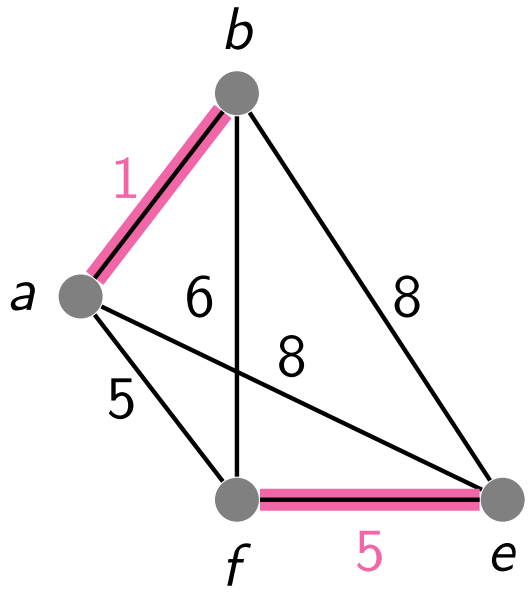
Um **emparelhamento** (=*matching*) é um conjunto M de arestas duas a duas não adjacentes.

Um emparelhamento M é **perfeito** se todo vértice do grafo é extremo de uma (única) aresta de M.

O algoritmo de Edmonds encontra um emparelhamento perfeito M que **minimiza** $c(M)$ em grafo G com um custo c_a em $\mathbb{Q}_{\geq 0}$ para cada aresta.

Um emparelhamento perfeito mínimo M em $G[I]$ é

Um emparelhamento perfeito mínimo M em $G[I]$ é



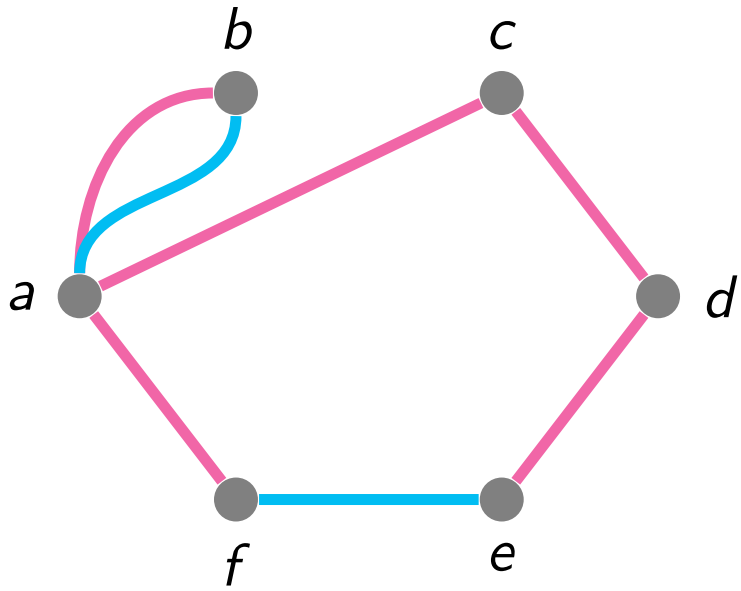
Multigrafo

Em alguns casos, convém admitir a existência várias "cópias" de uma mesma aresta em um grafo. Trata-se de um **multigrafo**.

Multigrafo

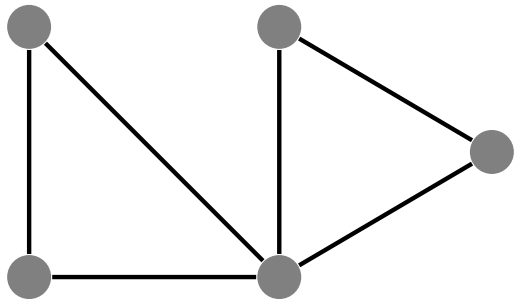
Em alguns casos, convém admitir a existência várias "cópias" de uma mesma aresta em um grafo. Trata-se de um **multigrafo**.

Considere o multigrafo $T' = (V(T), A(T) \cup M)$.



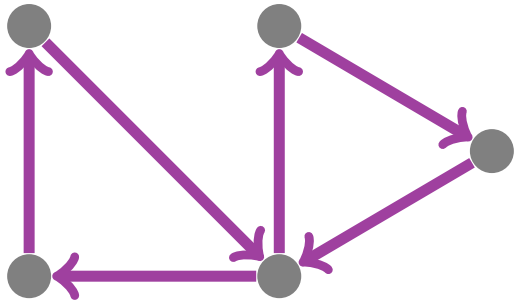
Ciclo euleriano

Um ciclo é **euleriano** se visita cada aresta do grafo exatamente uma vez.



Ciclo euleriano

Um ciclo é **euleriano** se visita cada aresta do grafo exatamente uma vez.



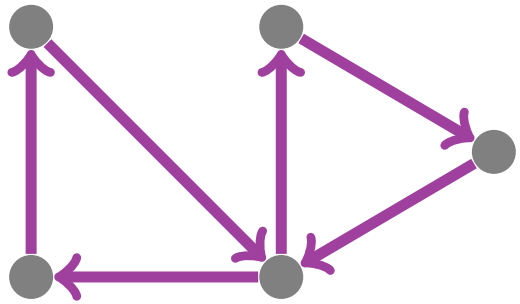
Teorema (Euler 1736)

Um grafo conexo admite um ciclo euleriano se, e somente se, todos os seus vértices têm grau par.

Teorema (Euler 1736)

Um grafo conexo admite um ciclo euleriano se, e somente se, todos os seus vértices têm grau par.

($\Rightarrow$) De fato, um vértice que aparece k vezes em um ciclo euleriano deve ter grau $2k$.



Teorema (Euler 1736)

Um grafo conexo admite um ciclo euleriano se, e somente se, todos os seus vértices têm grau par.

($\Rightarrow$) De fato, um vértice que aparece k vezes em um ciclo euleriano deve ter grau $2k$.
($\Leftarrow$) Seja G um grafo conexo com apenas vértices de grau par. Seja $W = (v_0, a_0, \cdots, a_{t-1}, v_t)$ a trilha mais longa em G.

Teorema (Euler 1736)

Um grafo conexo admite um ciclo euleriano se, e somente se, todos os seus vértices têm grau par.

($\Rightarrow$) De fato, um vértice que aparece k vezes em um ciclo euleriano deve ter grau $2k$.
($\Leftarrow$) Seja G um grafo conexo com apenas vértices de grau par. Seja $W = (v_0, a_0, \cdots, a_{t-1}, v_t)$ a trilha mais longa em G. Como W não pode ser aumentada, contém todas as arestas que incidem em v_t.

Teorema (Euler 1736)

Um grafo conexo admite um ciclo euleriano se, e somente se, todos os seus vértices têm grau par.

($\Rightarrow$) De fato, um vértice que aparece k vezes em um ciclo euleriano deve ter grau $2k$.
($\Leftarrow$) Seja G um grafo conexo com apenas vértices de grau par. Seja $W = (v_0, a_0, \cdots, a_{t-1}, v_t)$ a trilha mais longa em G. Como W não pode ser aumentada, contém todas as arestas que incidem em v_t. Por hipótese, a quantidade de tais arestas é par.

Teorema (Euler 1736)

Um grafo conexo admite um ciclo euleriano se, e somente se, todos os seus vértices têm grau par.

($\Rightarrow$) De fato, um vértice que aparece k vezes em um ciclo euleriano deve ter grau $2k$.
($\Leftarrow$) Seja G um grafo conexo com apenas vértices de grau par. Seja $W = (v_0, a_0, \cdots, a_{t-1}, v_t)$ a trilha mais longa em G. Como W não pode ser aumentada, contém todas as arestas que incidem em v_t. Por hipótese, a quantidade de tais arestas é par. Disso $v_0 = v_t$ e W é um ciclo.

Teorema (Euler 1736)

Um grafo conexo admite um ciclo euleriano se, e somente se, todos os seus vértices têm grau par.

($\Rightarrow$) De fato, um vértice que aparece k vezes em um ciclo euleriano deve ter grau $2k$.
($\Leftarrow$) Seja G um grafo conexo com apenas vértices de grau par. Seja $W = (v_0, a_0, \cdots, a_{t-1}, v_t)$ a trilha mais longa em G. Como W não pode ser aumentada, contém todas as arestas que incidem em v_t. Por hipótese, a quantidade de tais arestas é par. Disso $v_0 = v_t$ e W é um ciclo.
Suponha que W não seja euleriano. Então existe uma aresta $a = vv_i \notin W$ com $v_i \in W$. Logo existe uma trilha $(v, a, v_i, \cdots, a_{t-1}, v_t, a_0, \cdots, a_{i-1}, v_i)$ mais longa do que W, uma contradição.

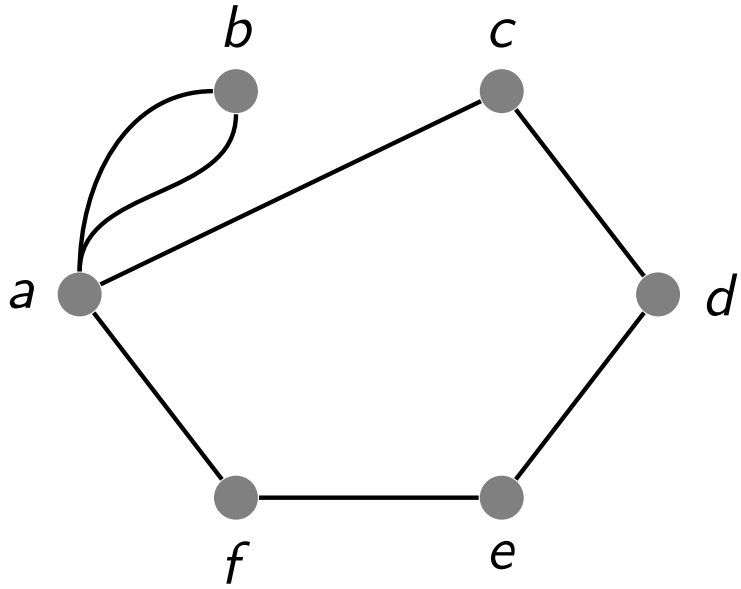
Algoritmo de Hierholzer

Proposto em 1873 pelo matemático Carl Hierholzer.

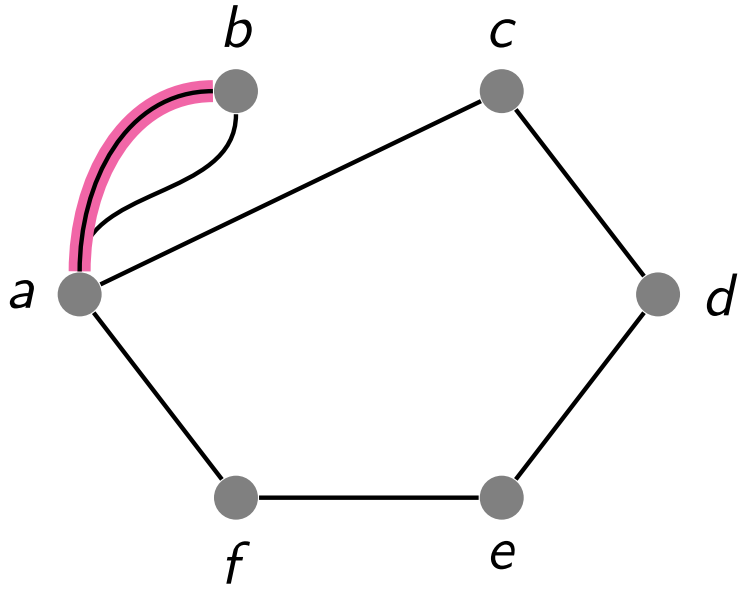
Algoritmo 2: HIERHOLZER(G)

1 faça $E \leftarrow \emptyset$;
2 faça $A' \leftarrow A$;
3 faça $C \leftarrow \emptyset$;
4 enquanto $A' \neq \emptyset$, faça:
5 $C \leftarrow \{v\}$ para $v \in G$ tal que $\exists w \in G$ com $vw \in A'$;
6 $A' \leftarrow A' \backslash \{vw\}$;
7 $v \leftarrow w$;
8 enquanto existir w tal que $vw \in A'$, faça:
9 $C \leftarrow C \cup \{v\}$;
10 $A' \leftarrow A' \backslash \{vw\}$;
11 $v \leftarrow w$;
12 $E \leftarrow C \cup E$;
13 devolva E;

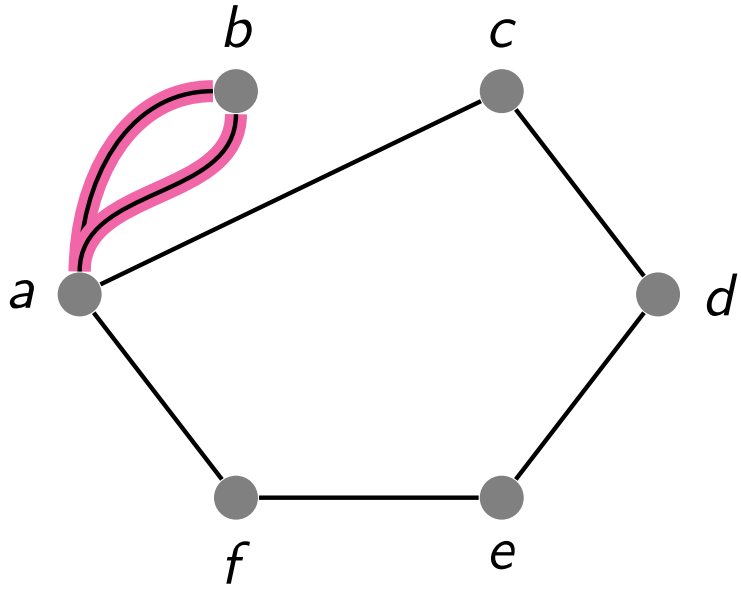
Ideia geral: construir um ciclo euleriano conectando ciclos aresta-disjuntos.
Começamos com um vértice v_1 qualquer e uma aresta incidente em v_1.
Seguimos por uma sequência de arestas adjacentes ainda não visitadas até retornar a v_1.
Se esse ciclo não for euleriano, repetimos os passos a partir de um vértice v_2 no qual incidam arestas não percorridas.
Obtemos um único ciclo pela união dos dois anteriores, tomando v_2 como vértice inicial e final.
Repetimos o processo até esgotar as arestas do grafo.



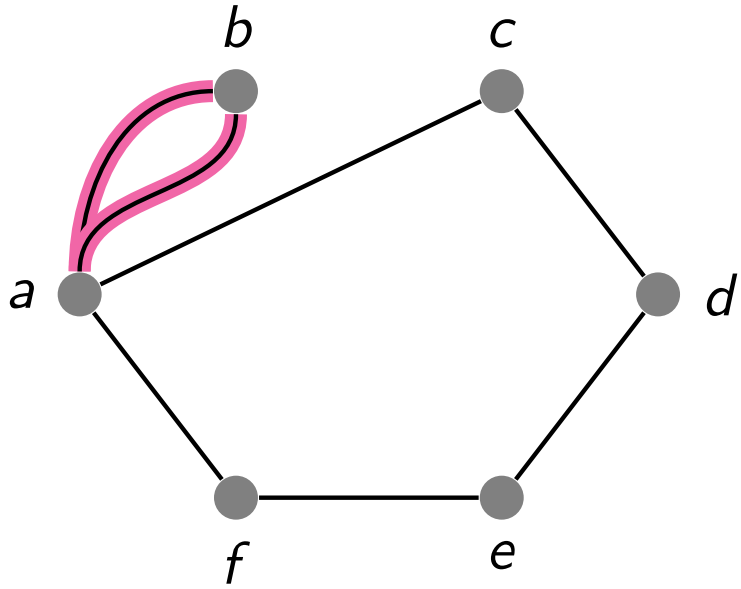
$$C = \{b\}$$



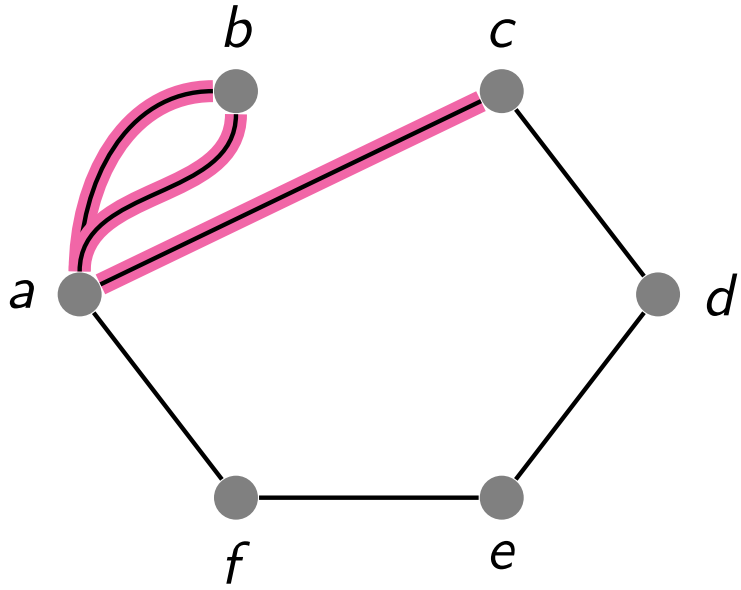
$$C = \{b, a\}$$



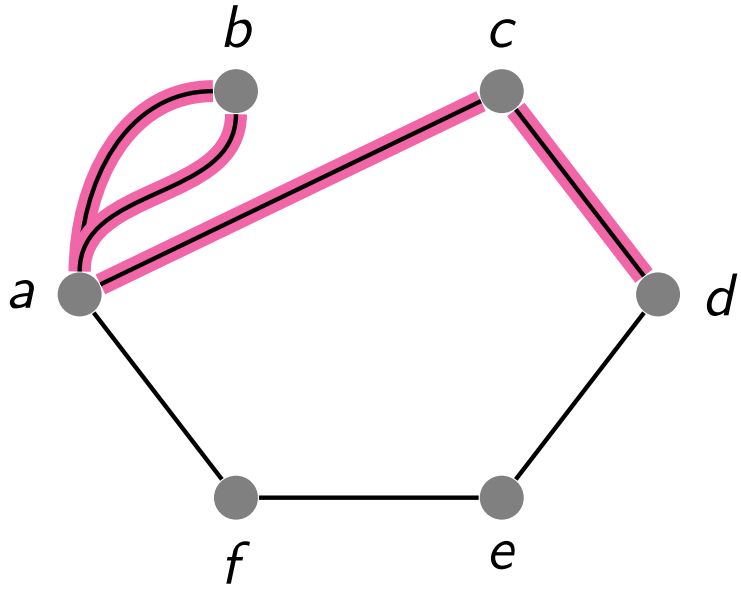
$$C = \{b, a, b\}$$



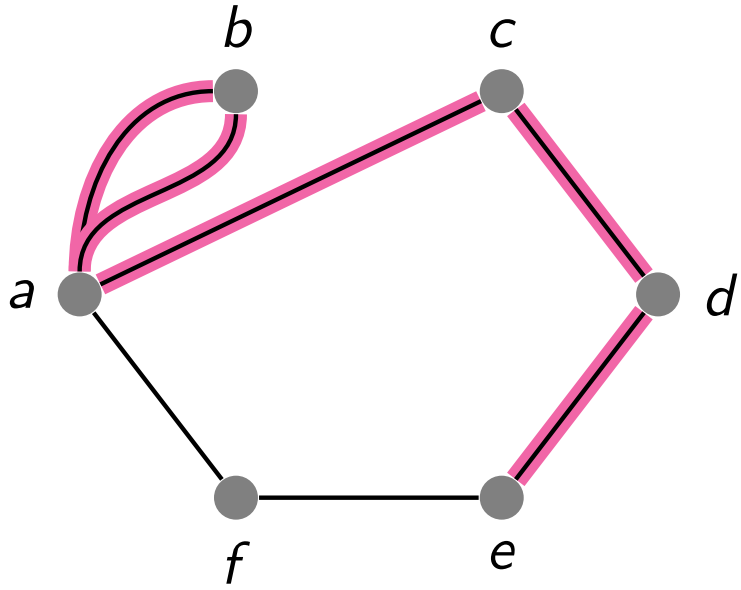
$$C = \{a\}; E = \{b, a, b\}$$



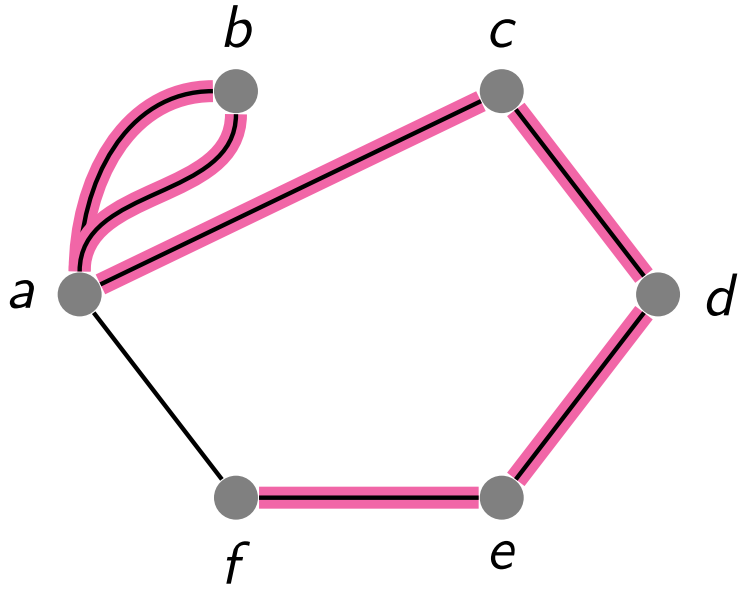
$$C = \{a, c\}; E = \{b, a, b\}$$



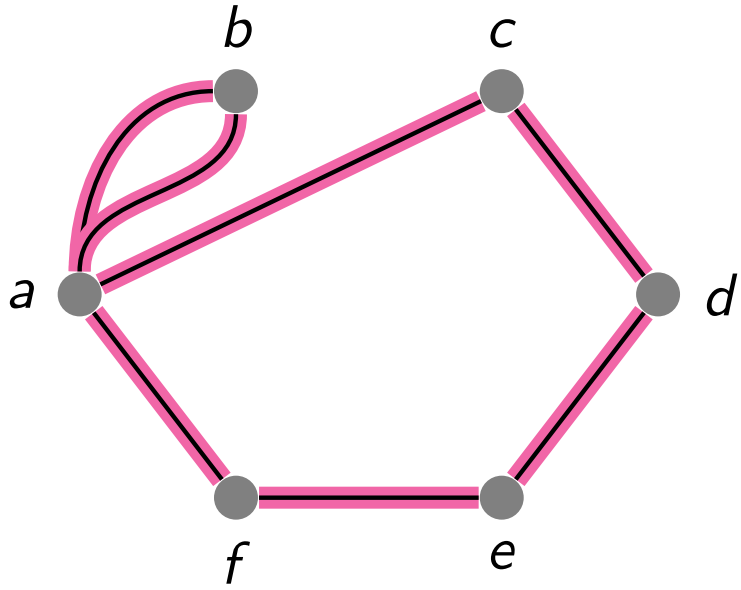
$$C = \{a, c, d\}; E = \{b, a, b\}$$



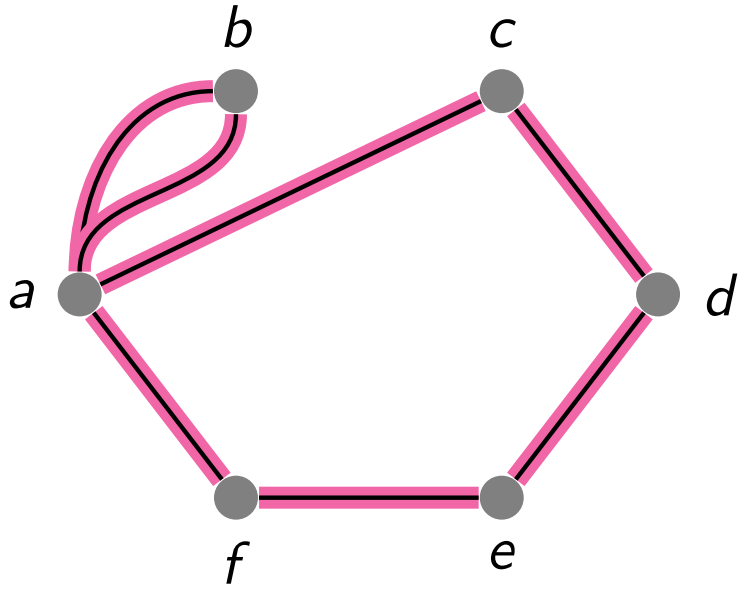
$$C = \{a, c, d, e\}; E = \{b, a, b\}$$



$$C = \{a, c, d, e, f\}; E = \{b, a, b\}$$



$$C = \{a, c, d, e, f, a\}; E = \{b, a, b\}$$



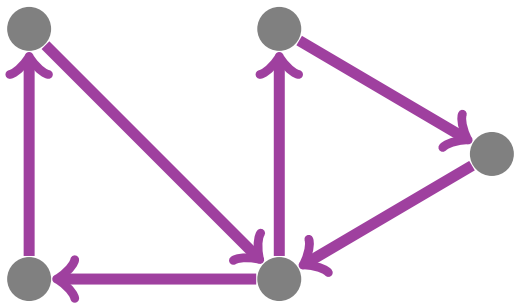
$$E = \{a, c, d, e, f, a, b, a\}$$

Algoritmo atalho

Ideia geral: extrair uma sequência maximal $S_H = (w_0, w_1, \cdots, w_n)$ sem vértices repetidos a partir da sequência $S_E = (v_0, v_1, \cdots, v_m)$ de vértices de um ciclo E gerador de G.

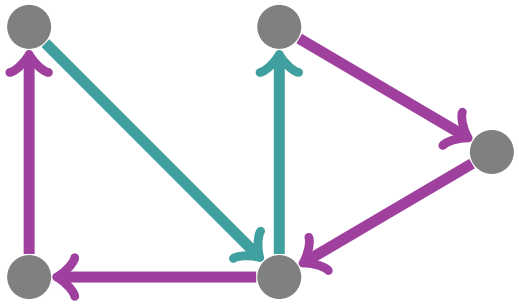
Algoritmo atalho

Ideia geral: extrair uma sequência maximal $S_H = (w_0, w_1, \cdots, w_n)$ sem vértices repetidos a partir da sequência $S_E = (v_0, v_1, \cdots, v_m)$ de vértices de um ciclo E gerador de G.



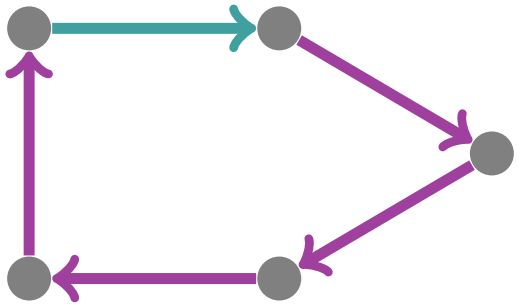
Algoritmo atalho

Ideia geral: extrair uma sequência maximal $S_H = (w_0, w_1, \cdots, w_n)$ sem vértices repetidos a partir da sequência $S_E = (v_0, v_1, \cdots, v_m)$ de vértices de um ciclo E gerador de G.



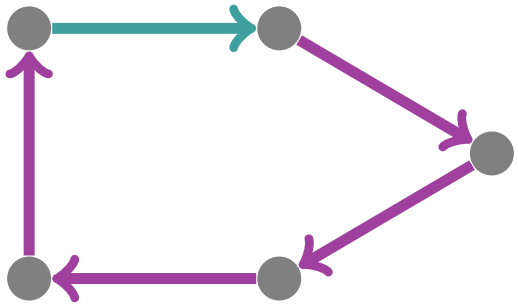
Algoritmo atalho

Ideia geral: extrair uma sequência maximal $S_H = (w_0, w_1, \cdots, w_n)$ sem vértices repetidos a partir da sequência $S_E = (v_0, v_1, \cdots, v_m)$ de vértices de um ciclo E gerador de G.



Algoritmo atalho

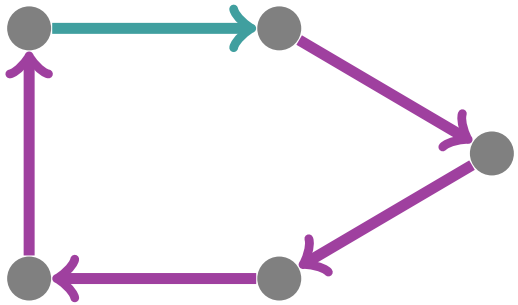
Ideia geral: extrair uma sequência maximal $S_H = (w_0, w_1, \cdots, w_n)$ sem vértices repetidos a partir da sequência $S_E = (v_0, v_1, \cdots, v_m)$ de vértices de um ciclo E gerador de G.



Como G é completo e E é gerador, S_H define um circuito hamiltoniano H.

Algoritmo atalho

Ideia geral: extrair uma sequência maximal $S_H = (w_0, w_1, \cdots, w_n)$ sem vértices repetidos a partir da sequência $S_E = (v_0, v_1, \cdots, v_m)$ de vértices de um ciclo E gerador de G.



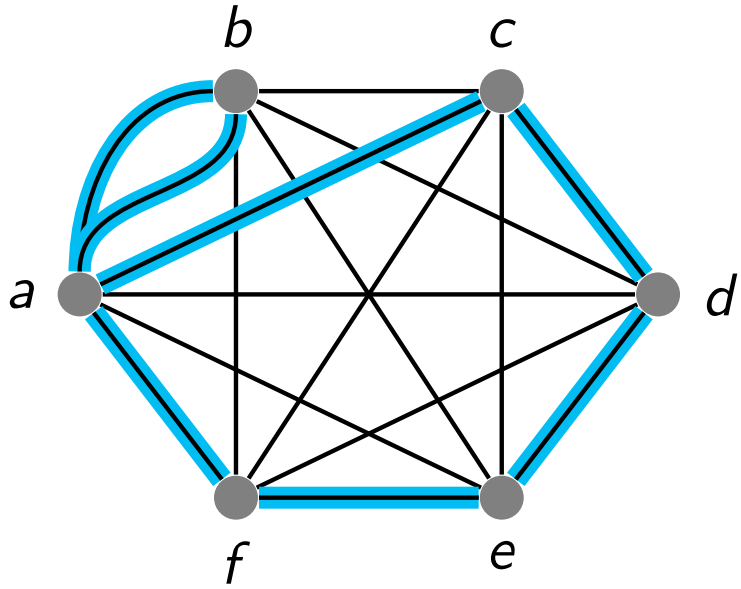
Como G é completo e E é gerador, S_H define um circuito hamiltoniano H. Pela desigualde triangular, temos

$$c((w_i, w_{i+1}), H) \leq c((w_i, w_{i+1}), E)$$

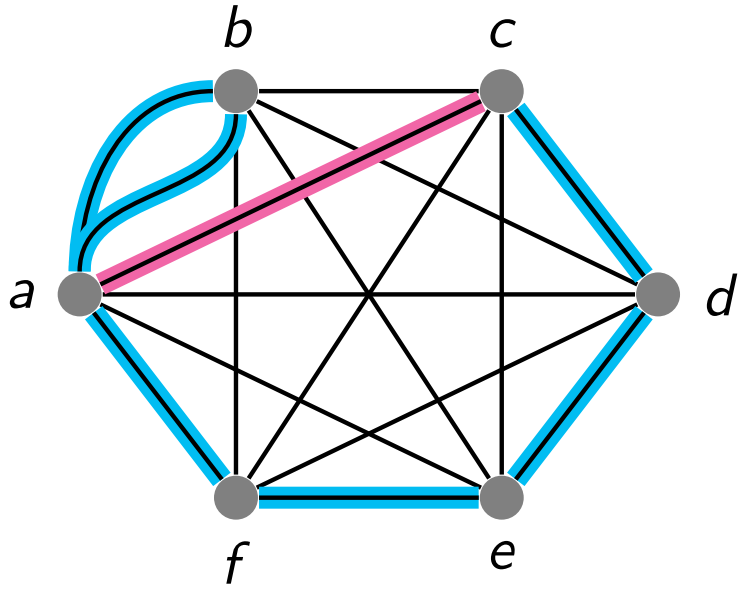
para os trajetos de custo mínimo entre quaisquer dois vértices w_i e w_{i+1} adjacentes em H.

Algoritmo 3: ATALHO(E)

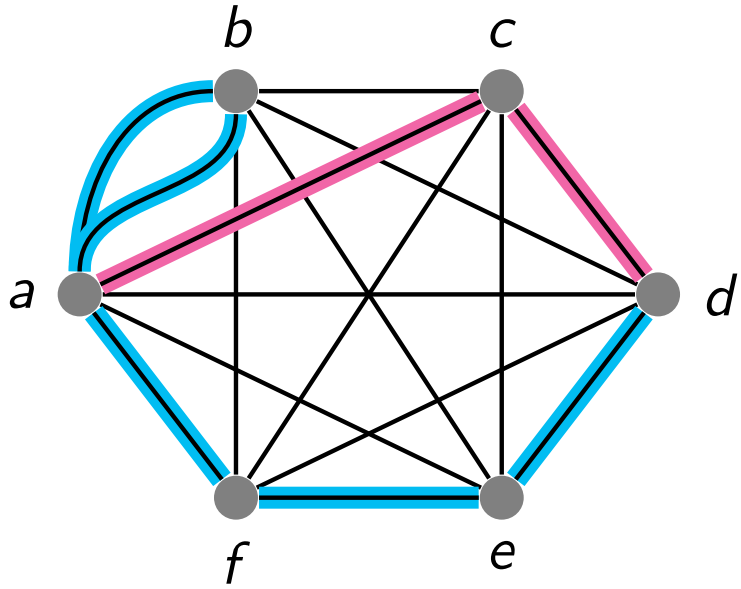
```
1 faça S_E ← (v_0, v_1, ··· , v_m);
2 faça w_0 ← v_0;
3 faça n ← 0;
4 para i de 1 a m, faça:
5    se v_i ∉ {w_0, ··· , w_n}, então:
6       n ← n + 1;
7       w_n ← v_i;
8 faça S_H ← (w_0, w_1, ··· , w_n);
9 devolva S_H;
```



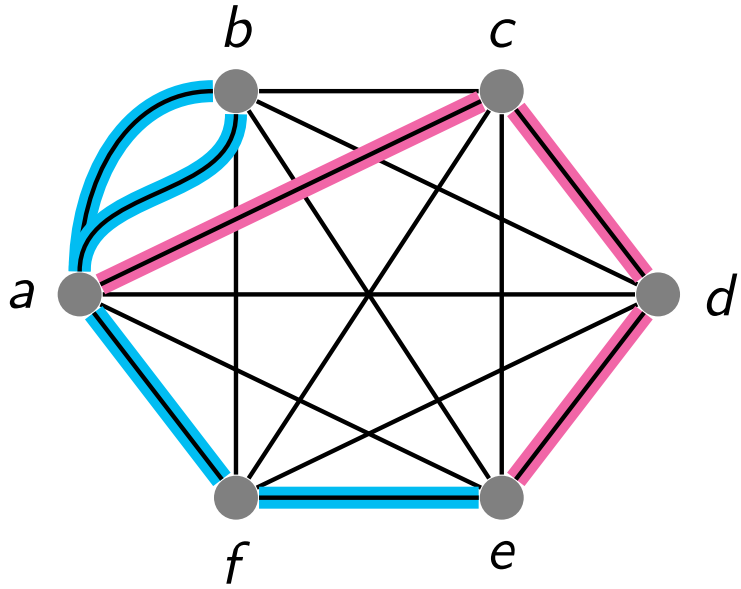
$$S_E = (a, c, d, e, f, a, b, a)$$



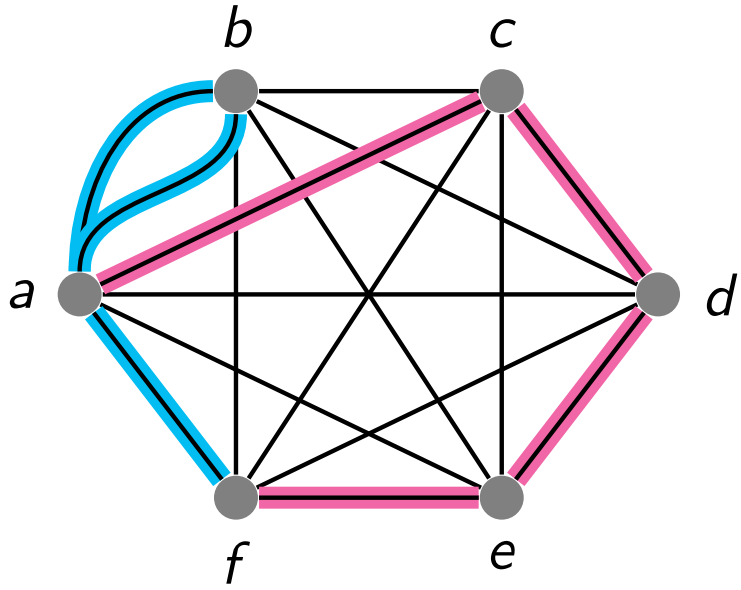
$$S_E = (a, c, d, e, f, a, b, a); S_H = (a, c)$$



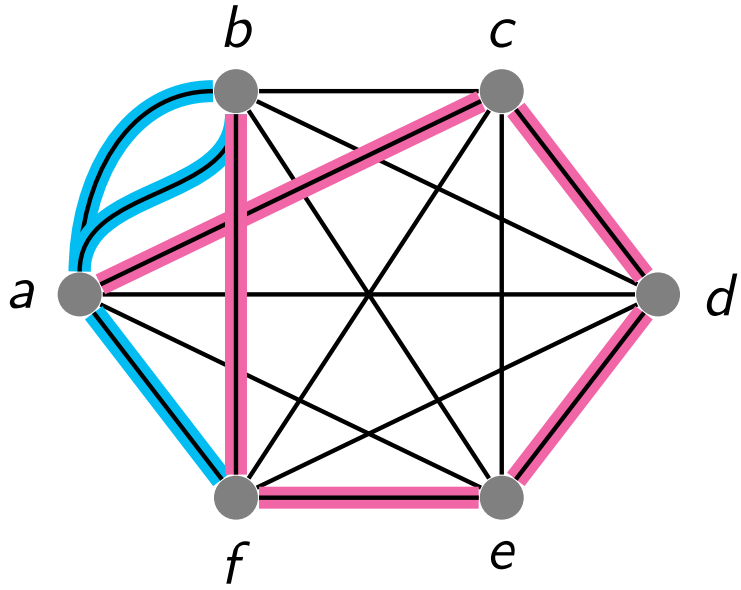
$$S_E = (a, c, d, e, f, a, b, a); S_H = (a, c, d)$$



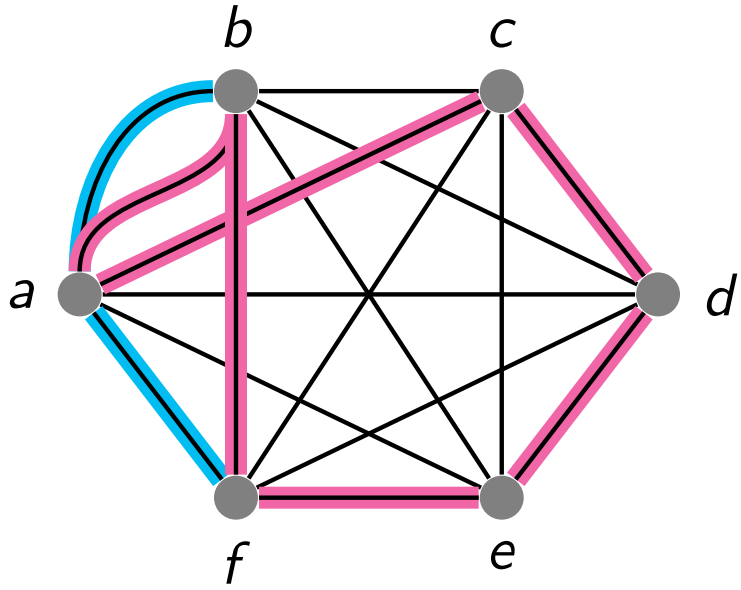
$$S_E = (a, c, d, e, f, a, b, a); S_H = (a, c, d, e)$$



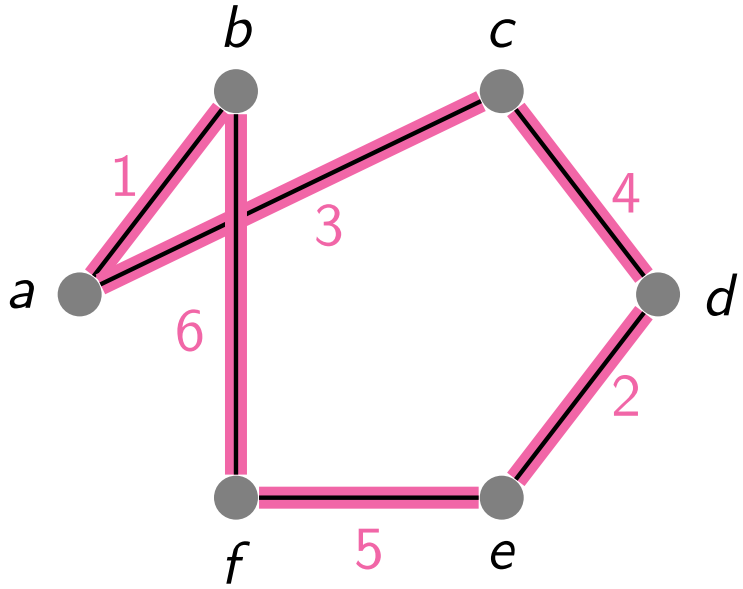
$$S_E = (a, c, d, e, f, a, b, a); S_H = (a, c, d, e, f)$$



$$S_E = (a, c, d, e, f, a, b, a); S_H = (a, c, d, e, f, b)$$



$$S_E = (a, c, d, e, f, a, b, a); S_H = (a, c, d, e, f, b, a)$$



$$c(H) = 21$$

Algoritmo de Christofides

Publicado em 1976 pelo matemático Nico Christofides.

Ideia geral: aplicar os demais algoritmos em sequência.

Algoritmo de Christofides

Publicado em 1976 pelo matemático Nico Christofides.

Ideia geral: aplicar os demais algoritmos em sequência.
Dado o grafo completo G, KRUSKAL encontra uma árvore geradora mínima T.

Algoritmo de Christofides

Publicado em 1976 pelo matemático Nico Christofides.

Ideia geral: aplicar os demais algoritmos em sequência.
Dado o grafo completo G, KRUSKAL encontra uma árvore geradora mínima T.
Considerar o conjunto I dos vértices de grau ímpar de T.
Dado o grafo induzido $G[I]$, EDMONDS encontra um emparelhamento perfeito mínimo M.

Algoritmo de Christofides

Publicado em 1976 pelo matemático Nico Christofides.

Ideia geral: aplicar os demais algoritmos em sequência.
Dado o grafo completo G, KRUSKAL encontra uma árvore geradora mínima T.
Considerar o conjunto I dos vértices de grau ímpar de T.
Dado o grafo induzido $G[I]$, EDMONDS encontra um emparelhamento perfeito mínimo M.
Dado o multigrafo $T' = (V(T), A(T) \cup M)$ tal que todos os vértices têm grau par, HIERHOLZER encontra um ciclo euleriano E.

Algoritmo de Christofides

Publicado em 1976 pelo matemático Nico Christofides.

Ideia geral: aplicar os demais algoritmos em sequência.
Dado o grafo completo G, KRUSKAL encontra uma árvore geradora mínima T.
Considerar o conjunto I dos vértices de grau ímpar de T.
Dado o grafo induzido $G[I]$, EDMONDS encontra um emparelhamento perfeito mínimo M.
Dado o multigrafo $T' = (V(T), A(T) \cup M)$ tal que todos os vértices têm grau par, HIERHOLZER encontra um ciclo euleriano E.
Dado E, ATALHO encontra e devolve um circuito hamiltoniano H.

Algoritmo 4: CHRISTOFIDES(G, c)

```
1 faça T ← KRUSKAL(G, c);
2 faça M ← EDMONDS(G[I], c);
3 faça T' ← T ∪ M;
4 faça E ← HIERHOLZER(T');
5 faça H ← ATALHO(E);
6 devolva H;
```

Cada um dos subalgoritmos KRUSKAL, EDMONDS e HIERHOLZER é polinomial. Portanto, o algoritmo CHRISTOFIDES é polinomial.

Teorema

O algoritmo CHRISTOFIDES é uma $\frac{3}{2}$-aproximação polinomial para o TSPM.

Teorema

O algoritmo CHRISTOFIDES é uma $\frac{3}{2}$-aproximação polinomial para o TSPM.

Precisamos mostrar que $c(H) \leq \frac{3}{2}\mathrm{opt}(G, c)$.

Teorema

O algoritmo CHRISTOFIDES é uma $\frac{3}{2}$-aproximação polinomial para o TSPM.

Precisamos mostrar que $c(H) \leq \frac{3}{2}\text{opt}(G, c)$. Sabemos que $c(H) \leq c(E)$. Além disso $c(E) = c(T') = c(T) + c(M)$ e $c(T) \leq \text{opt}(G, c)$.

Teorema

O algoritmo CHRISTOFIDES é uma $\frac{3}{2}$-aproximação polinomial para o TSPM.

Precisamos mostrar que $c(H) \leq \frac{3}{2}\text{opt}(G, c)$. Sabemos que $c(H) \leq c(E)$. Além disso $c(E) = c(T') = c(T) + c(M)$ e $c(T) \leq \text{opt}(G, c)$. Daí segue $c(H) \leq \text{opt}(G, c) + c(M)$ e basta mostrar que $c(M) \leq \frac{1}{2}\text{opt}(G, c)$.

Teorema

O algoritmo CHRISTOFIDES é uma $\frac{3}{2}$-aproximação polinomial para o TSPM.

Precisamos mostrar que $c(H) \leq \frac{3}{2}\text{opt}(G, c)$. Sabemos que $c(H) \leq c(E)$. Além disso $c(E) = c(T') = c(T) + c(M)$ e $c(T) \leq \text{opt}(G, c)$. Daí segue $c(H) \leq \text{opt}(G, c) + c(M)$ e basta mostrar que $c(M) \leq \frac{1}{2}\text{opt}(G, c)$. Seja H^* uma solução ótima e sejam $w_1, w_2, \cdots, w_{2k}$ os vértices de grau ímpar em T.

Teorema

O algoritmo CHRISTOFIDES é uma $\frac{3}{2}$-aproximação polinomial para o TSPM.

Precisamos mostrar que $c(H) \leq \frac{3}{2}\text{opt}(G, c)$. Sabemos que $c(H) \leq c(E)$. Além disso $c(E) = c(T') = c(T) + c(M)$ e $c(T) \leq \text{opt}(G, c)$. Daí segue $c(H) \leq \text{opt}(G, c) + c(M)$ e basta mostrar que $c(M) \leq \frac{1}{2}\text{opt}(G, c)$. Seja H^* uma solução ótima e sejam $w_1, w_2, \cdots, w_{2k}$ os vértices de grau ímpar em T. Como G é completo, a sequência $C := (w_1, w_2, \cdots, w_{2k})$ é um circuito em $G[I]$. Tome dois vértices w_i e w_{i+1}.

Teorema

O algoritmo CHRISTOFIDES é uma $\frac{3}{2}$-aproximação polinomial para o TSPM.

Precisamos mostrar que $c(H) \leq \frac{3}{2}\text{opt}(G, c)$. Sabemos que $c(H) \leq c(E)$. Além disso $c(E) = c(T') = c(T) + c(M)$ e $c(T) \leq \text{opt}(G, c)$. Daí segue $c(H) \leq \text{opt}(G, c) + c(M)$ e basta mostrar que $c(M) \leq \frac{1}{2}\text{opt}(G, c)$. Seja H^* uma solução ótima e sejam $w_1, w_2, \cdots, w_{2k}$ os vértices de grau ímpar em T. Como G é completo, a sequência $C := (w_1, w_2, \cdots, w_{2k})$ é um circuito em $G[I]$. Tome dois vértices w_i e w_{i+1}. Podemos substituir o caminho de H^* que os une pela aresta $w_i w_{i+1}$ de C. Pela desigualdade triangular, temos que $c(C) \leq c(H^*)$.

Teorema

O algoritmo CHRISTOFIDES é uma $\frac{3}{2}$-aproximação polinomial para o TSPM.

Precisamos mostrar que $c(H) \leq \frac{3}{2}\text{opt}(G, c)$. Sabemos que $c(H) \leq c(E)$. Além disso $c(E) = c(T') = c(T) + c(M)$ e $c(T) \leq \text{opt}(G, c)$. Daí segue $c(H) \leq \text{opt}(G, c) + c(M)$ e basta mostrar que $c(M) \leq \frac{1}{2}\text{opt}(G, c)$. Seja H^* uma solução ótima e sejam $w_1, w_2, \cdots, w_{2k}$ os vértices de grau ímpar em T. Como G é completo, a sequência $C := (w_1, w_2, \cdots, w_{2k})$ é um circuito em $G[I]$. Tome dois vértices w_i e w_{i+1}. Podemos substituir o caminho de H^* que os une pela aresta $w_i w_{i+1}$ de C. Pela desigualdade triangular, temos que $c(C) \leq c(H^*)$. Note que C é um circuito par, então $A(C)$ é união de dois emparelhamentos perfeitos disjuntos M_1 e M_2. Desse modo $c(M_1) + c(M_2) = c(C)$. Como M tem custo mínimo, segue $2c(M) \leq c(M_1) + c(M_2)$.

Teorema

O algoritmo CHRISTOFIDES é uma $\frac{3}{2}$-aproximação polinomial para o TSPM.

Precisamos mostrar que $c(H) \leq \frac{3}{2}\text{opt}(G, c)$. Sabemos que $c(H) \leq c(E)$. Além disso $c(E) = c(T') = c(T) + c(M)$ e $c(T) \leq \text{opt}(G, c)$. Daí segue $c(H) \leq \text{opt}(G, c) + c(M)$ e basta mostrar que $c(M) \leq \frac{1}{2}\text{opt}(G, c)$. Seja H^* uma solução ótima e sejam $w_1, w_2, \cdots, w_{2k}$ os vértices de grau ímpar em T. Como G é completo, a sequência $C := (w_1, w_2, \cdots, w_{2k})$ é um circuito em $G[I]$. Tome dois vértices w_i e w_{i+1}. Podemos substituir o caminho de H^* que os une pela aresta $w_i w_{i+1}$ de C. Pela desigualdade triangular, temos que $c(C) \leq c(H^*)$. Note que C é um circuito par, então $A(C)$ é união de dois emparelhamentos perfeitos disjuntos M_1 e M_2. Desse modo $c(M_1) + c(M_2) = c(C)$. Como M tem custo mínimo, segue $2c(M) \leq c(M_1) + c(M_2)$. Finalmente $2c(M) \leq c(H^*) = \text{opt}(G, c)$, como queríamos.

Referências

Algorithmus von Hierholzer. Disponível em: https://www-m9.ma.tum.de/graph-algorithms/hierholzer/index_en.html

Marina Andretta. *Algoritmos de aproximação - Problema do caixeiro viajante*. Disponível em: https://sites.icmc.usp.br/andretta/ensino/sme0216-5826-2-15.html

Algoritmo de Kruskal. Disponível em: https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/kruskal.html

Paulo Feofiloff. *Curso de Otimização Combinatória*. Disponível em: https://www.ime.usp.br/~pf/otimizacao-combinatoria/

Reinhard Diestel. *Graph Theory*. Springer, New York, 2000.

Dimitris Bertsimas e John N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific e Dynamic Ideas, LLC, Belmont, MA, 1997.

Kruskal's Algorithm Animation - How does it progress? Disponível em: https://www.youtube.com/watch?v=o8Sqm1_3BRo&feature=youtu.be

L. Lovász e M. D. Plummer. *Matching Theory*. Elsevier, Amsterdam, 1986.

M. H. Carvalho, M. R. Cerioli, R. Dahab, P. Feofiloff, C. G. Fernandes, C. E. Ferreira, K. S. Guimarães, F. K. Miyazawa, J. C. Piña Jr., J. A. R. Soares e Y. Wakabayashi. *Uma introdução sucinta a algoritmos de aproximação*. Disponível em: `https://www.ime.usp.br/~cris/aprox/`

P. Feofiloff, Y. Kohayakawa e Y. Wakabayashi. *Uma introdução sucinta a teoria dos grafos*. Disponível em: `https://www.ime.usp.br/~pf/teoriadosgrafos/`