Ciclos Hamiltonianos e o Problema do Caixeiro Viajante 5189-32

Rodrigo Calvo
rcalvo@uem.br

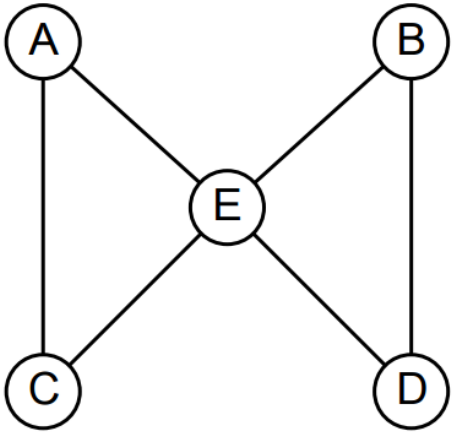
Departamento de Informática – DIN
Universidade Estadual de Maringá – UEM

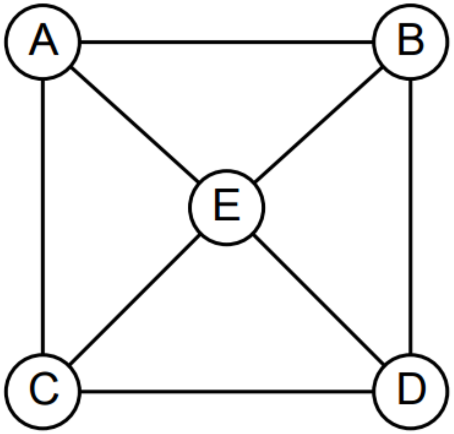
1° semestre de 2016

Introdução

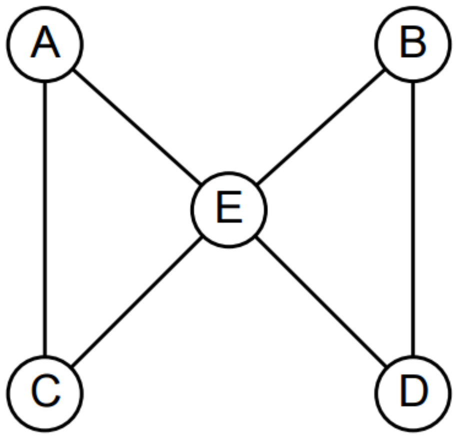
- Um **ciclo hamiltoniano** (caminho hamiltoniano) é um ciclo (caminho) que usa cada vértice do grafo exatamente uma vez (exceto o primeiro vértice que é visitado duas vezes)
- Um grafo que contém um ciclo hamiltoniano é chamado de **grafo hamiltoniano**

Exemplos

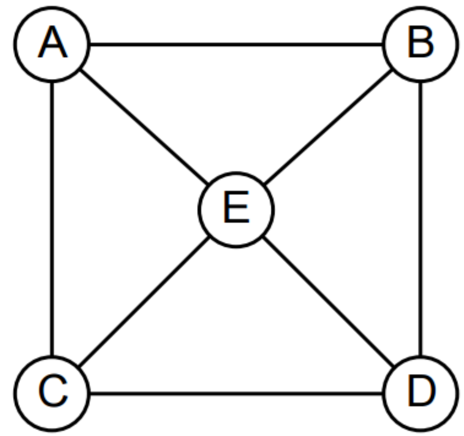




Exemplos



Este grafo não é hamiltoniano



Este grafo é hamiltoniano:
A, C, E, D, B, A

Propriedades

- Teorema de Ore
  - Uma condilção suficiente (mas não necessária) para que um grafo $G = (V, A)$ seja hamiltoniano é que a soma dos graus de cada par de vértices não adjacentes seja no mínimo $|V|$
- Teorema de Dirac
  - Uma condição suficiente (mas não necessária) para que um grafo $G = (V, A)$ possua um ciclo hamiltoniano, é que o grau de cada vértice em G seja pelo menos igual a $|V|/ 2$

Algoritmos

- Não são conhecidos algoritmos de tempo polinomial para determinar se um grafo é hamiltoniano ou não.

O problema do caixeiro viajante

- Dado um grafo conexo com peso nas arestas, o problema do caixeiro viajante (do inglês *travelling salesman problem* - TSP) consiste em encontrar um ciclo de peso mínimo que passe por cada vértice exatamente uma vez (retornando ao vértice de origem)

- Aplicações
  - Planejamento de entregas
  - Perfuração de placas de circuito impresso
  - Sequenciamento de DNA (subproblema)

O problema do caixeiro viajante

- O TSP é NP-Difícil, o que implica que, a menos que P = NP, não existem algoritmos exatos de tempo polinomial para resolver este problema. Desta forma, é comum estudar algoritmos que não são exatos (não garantem encontrar o ótimo) mas executam em tempo polinomial

- Veremos os seguintes algoritmos
  - Algoritmo aproximativo baseado na árvore geradora mínima
  - Algoritmo heurístico construtivo vizinho mais próximo
  - Algoritmo heurístico melhorativo 2-opt e 3opt

Algoritmo baseado na árvore geradora mínima

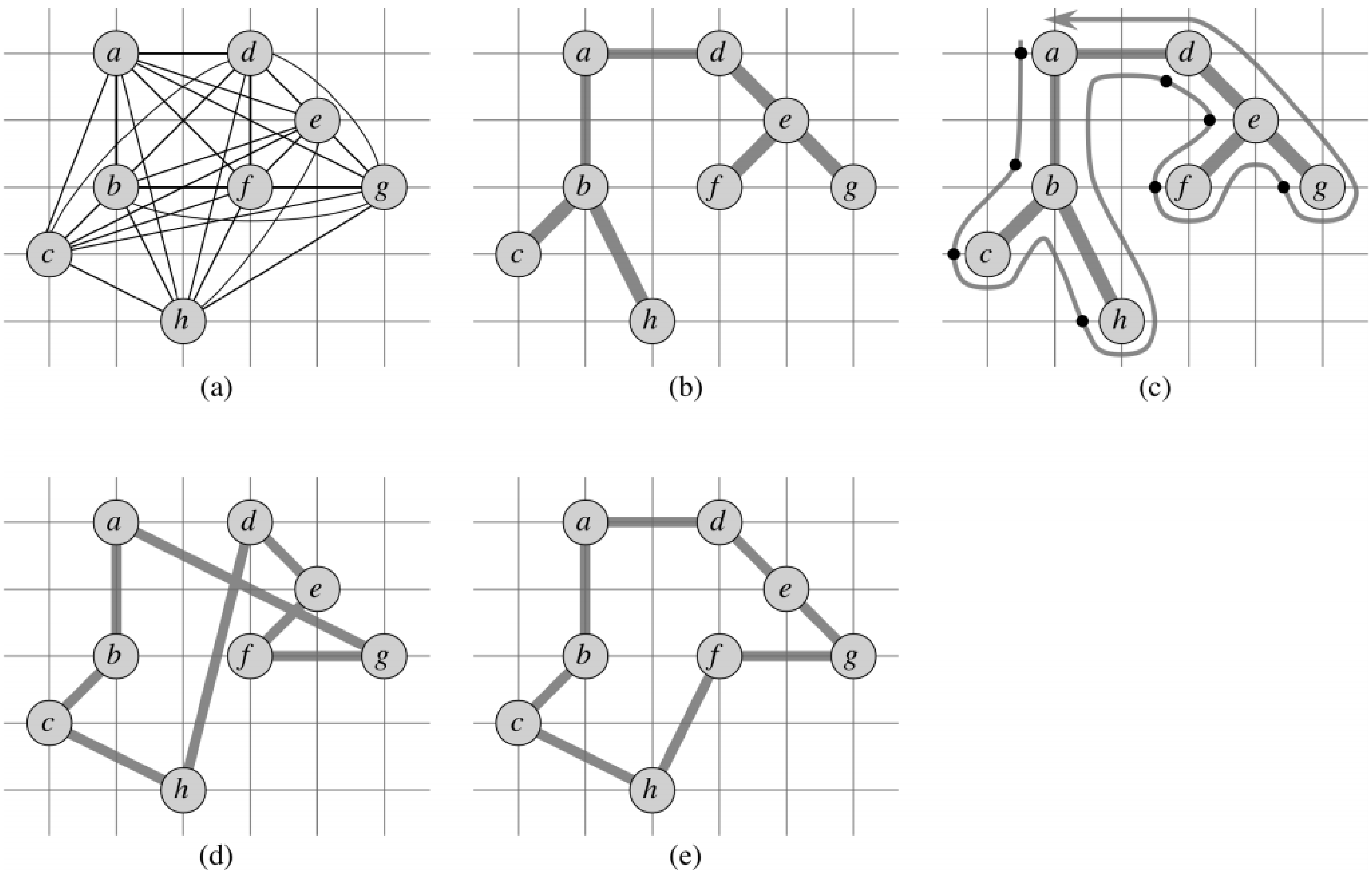
- Ideia
  - Construir uma árvore geradora mínima e fazer um percurso pré-ordem
- Característica
  - Algoritmo aproximativo com fator de aproximação 2
- Requisito
  - A função de custo *c* tem que respeitar a desigualdade de triângulos, isto é, para todo vértice $u, v, w \in V$

$$c(u,w) \leq c(u, v) + c(v,w)$$

Algoritmo baseado na árvore geradora mínima

```
tsp-mst(G)
1 selecionar um vértice r ∈ G.V para ser a raiz
2 computar a árvore geradora mínima T usando mst-prim(G,r)
3 seja H a lista de vértices ordenados de acordo com uma
visitação em pré-ordem da árvore T
4 return o ciclo hamiltoniano H
```

Algoritmo baseado na árvore geradora mínima



Algoritmo baseado na árvore geradora mínima

	a	b	c	d	e	f	g	h
a	-	2.0	3.2	2.0	3.2	2.8	4.5	4.1
b	2.0	-	1.4	2.8	3.2	2.0	4.0	2.2
c	3.2	1.4	-	4.2	4.5	3.2	5.1	2.2
d	2.0	2.8	4.2	-	1.4	2.0	2.8	4.1
e	3.1	3.2	4.5	1.4	-	1.4	1.4	3.6
f	2.8	2.0	3.1	2.0	1.4	-	2.0	2.2
g	4.5	4.0	5.1	2.8	1.4	2.0	-	3.6
h	4.1	2.2	2.2	4.1	3.6	2.2	3.6	-

- Caminho encontrado por **`tsp-mst`** $\langle a, b, c, h, d, e, f, g, a \rangle$, custo = 19,1
- Melhor caminho $\langle a, b, c, h, f, g, e, d, a \rangle$, custo = 14,7

Algoritmo baseado na árvore geradora mínima

- Tempo de execução
  - Com uma implementação do **`mst-prim`** usando busca linear, o tempo de execução é $\Theta(V^2)$
- Prova do fator de aproximação 2
  - Discutido em sala (veja a seção 35.2.1 do livro do Cormen)

Heurística do vizinho mais próximo

- Ideia
  - Inicializar com um vértice qualquer
  - A cada passo, escolher o vértice mais próximo do último vértice visitado
  - Voltar para o primeiro vértice

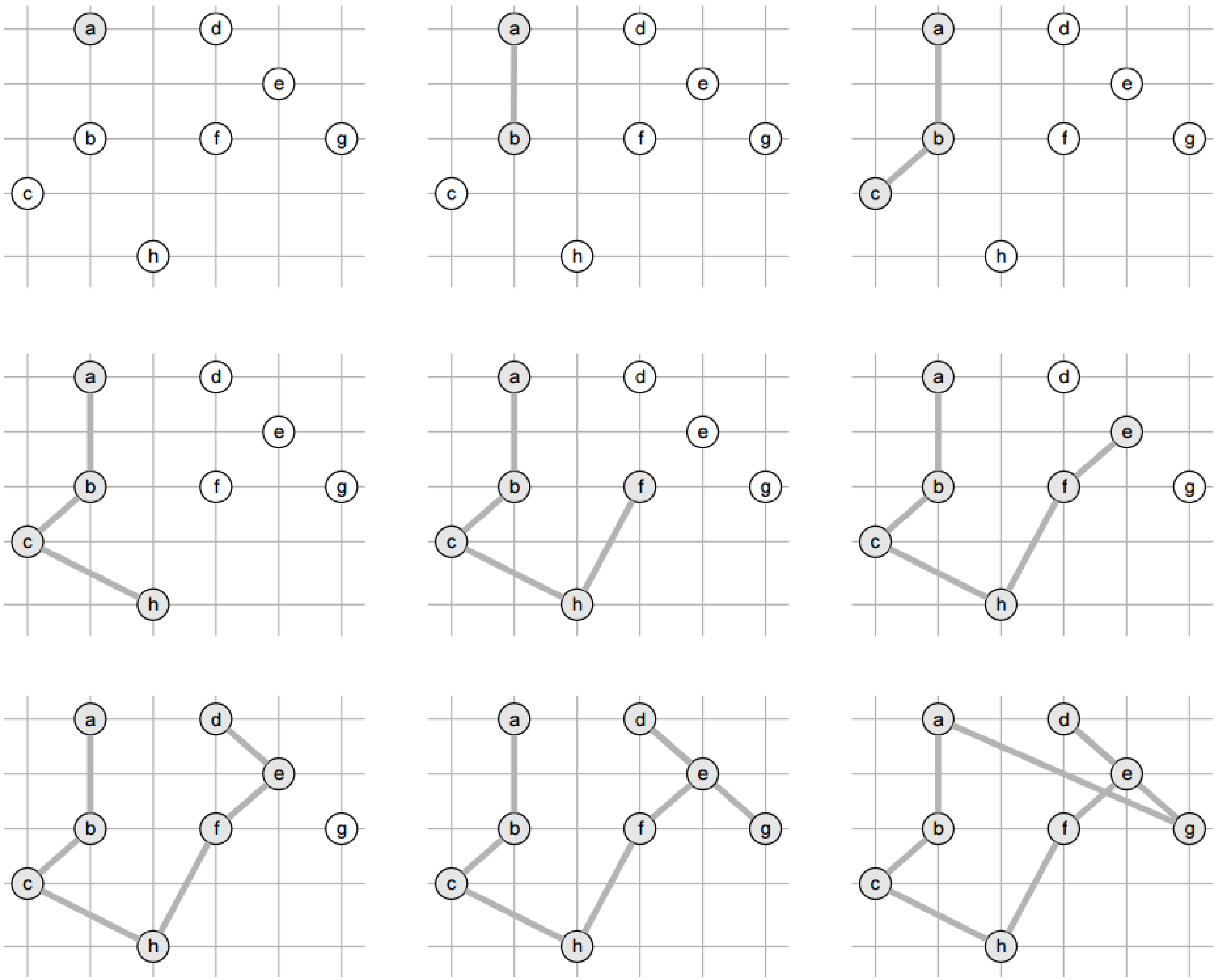
Heurística do vizinho mais próximo

```
tsp-nn(G)
selecionar um vértice inicial v0 ∈ G.V
C = {v0}                // caminho contendo v0
v = v0
while C não inclui todos os vértices
  u = vértice mais próximo de v que não está em C
  C = C ∪ {u}           // adiciona u ao caminho C
  v = u
C = C ∪ {v0}            // fecha o ciclo
return C
```

Heurística do vizinho mais próximo

- Análise do tempo de execução
  - Encontrar o vértice mais próximo que não foi visitado (linha 5) usando uma busca linear custa O(V)
  - Como cada vértice (a menos do primeiro) é colocado uma vez no ciclo, o tempo total de execução do algoritmo é O(V^2)

Heurística do vizinho mais próximo



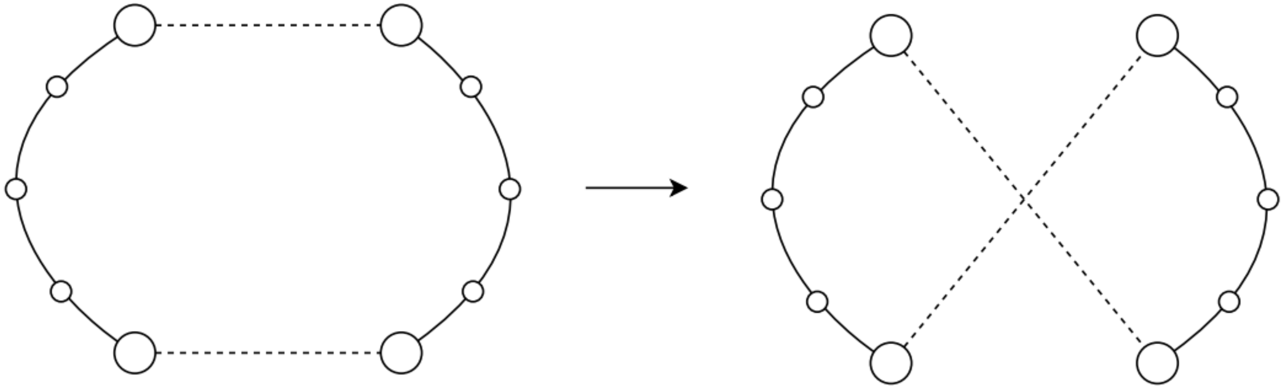
Heurística do vizinho mais próximo

	a	b	c	d	e	f	g	h
a	-	2.0	3.2	2.0	3.2	2.8	4.5	4.1
b	2.0	-	1.4	2.8	3.2	2.0	4.0	2.2
c	3.2	1.4	-	4.2	4.5	3.2	5.1	2.2
d	2.0	2.8	4.2	-	1.4	2.0	2.8	4.1
e	3.1	3.2	4.5	1.4	-	1.4	1.4	3.6
f	2.8	2.0	3.1	2.0	1.4	-	2.0	2.2
g	4.5	4.0	5.1	2.8	1.4	2.0	-	3.6
h	4.1	2.2	2.2	4.1	3.6	2.2	3.6	-

- Caminho encontrado por **`tsp-nn`** $\langle a, b, c, h, f, e, d, g, a \rangle$, custo = 17,9
- Melhor caminho $\langle a, b, c, h, f, g, e, d, a \rangle$, custo = 14,7

Heurística 2-opt

- Usada para melhorar uma solução (obtida por algum algoritmo construtivo)
- Ideia
  - Elimina 2 arestas não adjacentes e reconecta o caminho usando duas outras arestas, verifica se houve melhora
  - Repete este processo para todos os pares de arestas e realiza a melhor troca



Heurística 2-opt

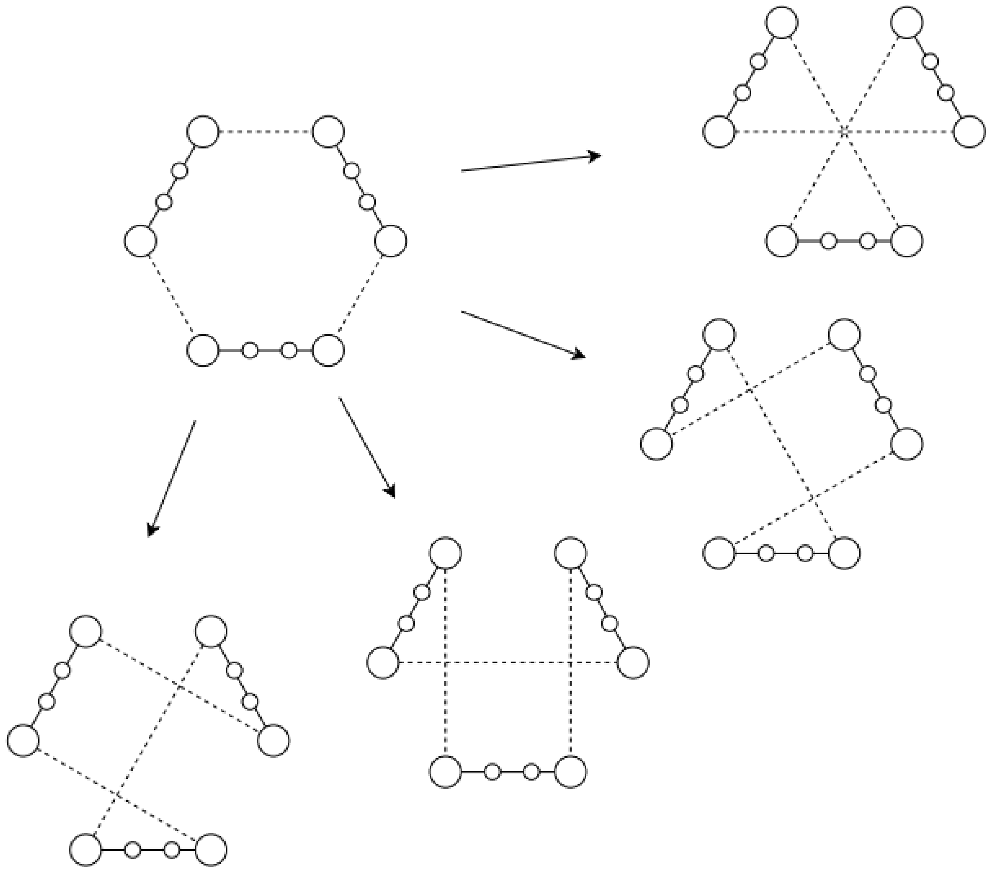
- Seja *w* a função de custo e *t* um percurso
- Vamos supor que o percurso *t* é representado por um vetor (índices 1..|*V*|) de vértices. A aresta $(t_{|V|},t_1)$ está implícita nesta representação
- Seja (t_a,t_b) e (t_c,t_d) duas arestas não adjacentes de *t* (*a*, *b*, *c*, *d* são os índices em *t*)
- Seja *t'* o percurso (válido) obtido a partir de *t* trocando a aresta (t_a,t_b) por (t_a,t_c) e (t_c,t_d) por (t_b,t_d)
- Seja $\delta = w(t_a,t_b) + w(t_c,t_d) - w(t_a,t_c) - w(t_b,t_d)$ a diferença do custo de *t* e *t'* (quanto maior a diferença, maior a melhoria)
- Seja δ_{max} o maior valor entre os δ's de todas as trocas de arestas

Heurística 2-opt

```
Heuristica 2-opt tsp-2opt(t,w)
loop
δmax = 0
best = nil
for a = 1 to (n - 2)
  b = a + 1
  if a ≠ 1
    lim = n
  else lim = n - 1
  for c = (a + 2) to lim
    if c ≠ n
      d = c + 1
    else d = 1
    δ = w(ta,tb) + w(tc ,td) - w(ta,tc) - w(tb,td)
    if δ > δmax
      δmax = δ
      best = a, b, c, d
 if δmax ≠ 0
   a, b, c, d = best
   t = (t - {(ta,tb),(tc ,td)}) U {(ta,tc),(tb,td)}
 else return t
```

Heurística 3-opt

- Mesmo princípio do 2-opt, mas seleciona 3 arestas



Exercício

- Resolva o problema do caixeiro viajante para o grafo representado pela matriz de adjacência utilizando um dos algoritmos vistos em sala.

Cidade	1	2	3	4	5
1	0	12	7	9	8
2	12	0	11	7	6
3	7	11	0	12	13
4	9	7	12	0	8
5	8	6	13	8	0

Bibliografia

- Thomas H. Cormen et al. Introduction to Algorithms. 3rd edition. Capítulo 35.2.1.
- Caminho hamiltoniano. Wikipédia. https: //pt.wikipedia.org/wiki/Caminho_hamiltoniano
- Problema do caminho hamiltoniano. Wikipédia https: //en.wikipedia.org/wiki/Hamiltonian_path_problem
- Problema do caixeiro viajante. Wikipédia. https://en.wikipedia.org/wiki/Travelling_salesman_problem
- Algoritmo vizinho mais próximo. Wikipédia. https://en.wikipedia.org/wiki/Nearest_neighbour_algorithm
- Algoritmo de Christofides. Wikipédia. https: //en.wikipedia.org/wiki/Christofides_algorithm