

Grafos IV

Joaquim Madeira

17/06/2021

Sumário

- Recap
- Determinação da **Árvore Geradora de Custo Mínimo** (MST – “Minimum Spanning Tree”)
- O Algoritmo de Kruskal
- O Algoritmo de Prim
- Determinação de **Circuitos Eulerianos** (“Euler Tour”)
- O Algoritmo de Fleury
- Sugestões de leitura

Let's
RECAP

Recapitulação

Caminho mais curto entre s e t

- Problema de **otimização combinatória**
- De todas as **soluções possíveis**, determinar a de **menor custo/distância**
- Podem existir **soluções ótimas alternativas**
 - Caminhos distintos com o mesmo custo/distância total
- Grafo / Grafo Orientado : contar o **nº de arestas** do caminho
- Rede : somar o valor de **distância** associado a cada aresta do caminho

Árvore dos caminhos mais curtos de s para t

- Associar um **rótulo** (“label”) a cada vértice : $(\text{dist}[v], \text{pred}[v])$
- No final do algoritmo o que representa ?
- $\text{pred}[v]$: o predecessor no caminho mais curto a partir de s
- $\text{dist}[v]$: o custo/distância associado ao caminho mais curto a partir de s
- Fazer o “**traceback**” do caminho mais curto !!

Inicialização dos rótulos

- Para cada vértice $v \neq s$

$$\text{dist}[v] = +\infty \qquad \text{pred}[v] = -1$$

- Para o vértice s

$$\text{dist}[s] = 0 \qquad \text{pred}[s] = -1$$

Quando **não há custos** associados às arestas

queue = Criar FILA vazia

Enqueue(queue, **s**)

Enquanto **NãoVazia**(queue) fazer

v = **Dequeue**(queue)

 Para cada vértice **w** adjacente a **v**

 Se **dist[w] == $+\infty$**

 Então **Enqueue**(queue, **w**)

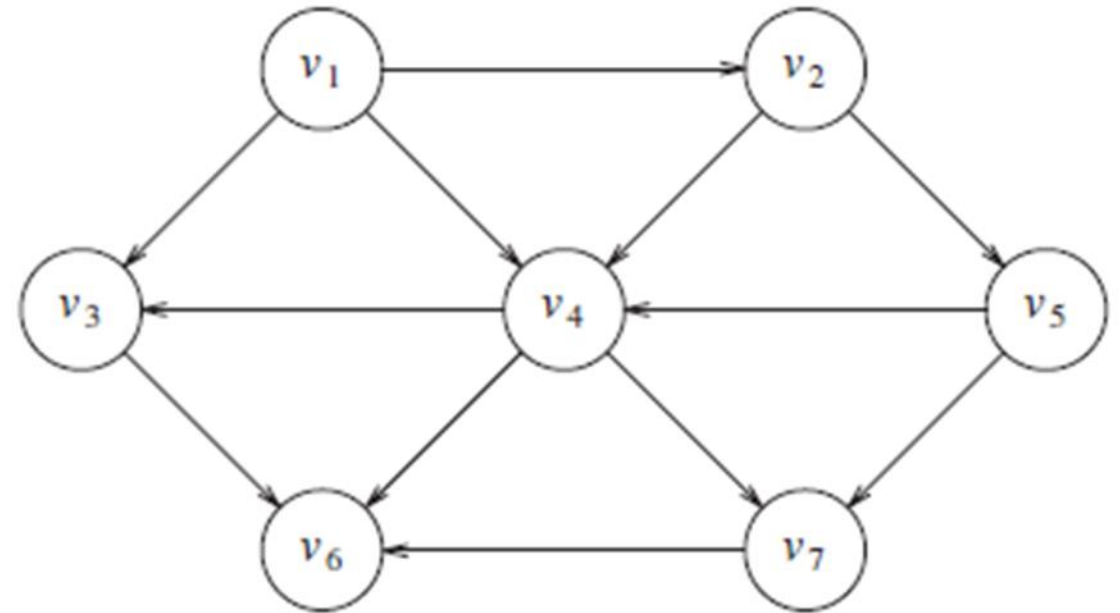
dist[w] = dist[v] + 1

pred[w] = v



Fizeram ?

- Qual é o caminho mais curto entre **v1** e **v5** ?
- E o caminho mais curto entre **v1** e **v6** ?
- Há caminhos ótimos **alternativos** ?
- De que depende a sua escolha ?



[Weiss]

Algoritmo de Bellman-Ford

Inicializar os rótulos dos vértices

Para $i = 1$ até ($\text{numVértices} - 1$) fazer // (V-1) vezes

Para cada aresta (u, v) fazer

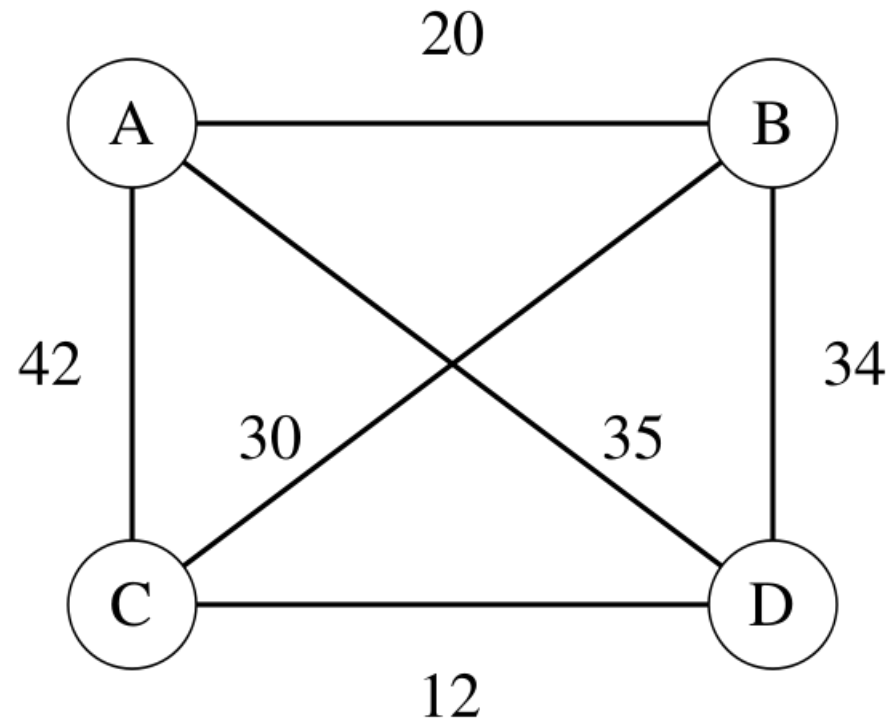
Se $\text{dist}[u] + \text{peso}(u, v) < \text{dist}[v]$ // Alternativa

Então // Atualizar

$\text{dist}[v] = \text{dist}[u] + \text{peso}(u, v)$

$\text{pred}[v] = u$

Fizeram ?



[Wikipedia]

Como melhorar ?

- Em que condição pode o **ciclo externo** ser executado menos vezes ?
- A partir do instante em que há a **certeza** de que **nenhum rótulo** poderá vir a ser **melhorado** !!
- Como verificar ? -> Usar uma **flag** !!

Ordem de complexidade

- Nº de comparações
- Pior Caso: $O(V \times E)$
- Melhor Caso: $O(V)$
- Como melhorar ?

Algoritmo usando QUEUE ou STACK

Inicializar os rótulos dos vértices

→ conjCandidatos = { s };

Enquanto conjCandidatos \neq { } fazer

→ u = próximoElemento(conjCandidatos);

// Depende da EDados

→ conjCandidatos = conjCandidatos - {v};

Para cada vértice v adjacente a u

Se $\text{dist}[u] + \text{peso}(u,v) < \text{dist}[v]$

// Alternativa

Então $\text{dist}[v] = \text{dist}[u] + \text{peso}(u,v);$

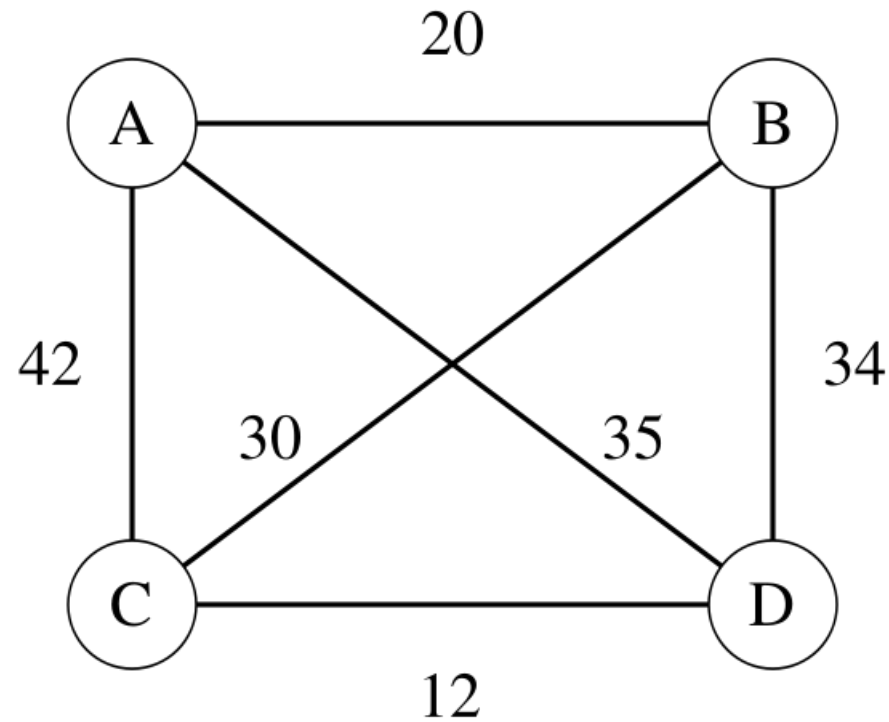
// Atualizar

$\text{pred}[v] = u;$

→ Se v não pertence conjCandidatos

→ Então conjCandidatos = conjCandidatos \cup { v };

Fizeram ?



[Wikipedia]

Algoritmo de Dijkstra – Estratégia “greedy”

Inicializar os rótulos dos vértices

→ conjCandidatos = { s };

→ Enquanto conjCandidatos $\neq \{ \}$ fazer

→ u = removerMenor(conjCandidatos); // Reordenação implícita

Para cada vértice v adjacente a u que ainda não pertence à solução

Se $\text{dist}[u] + \text{peso}(u,v) < \text{dist}[v]$

Então $\text{dist}[v] = \text{dist}[u] + \text{peso}(u,v)$;

$\text{pred}[v] = u$;

→ Se v não pertence conjCandidatos

→ Então conjCandidatos = conjCandidatos $\cup \{ v \}$;

→ Senão reposicionar v no conjunto ordenado de candidatos

Estrutura de dados

- Como manter ordenado o conjunto dos vértices candidatos ?
- Ordem parcial vs Ordem total
- Usar um MIN-HEAP / PRIORITY QUEUE
- Obter o próximo vértice candidato sem grande esforço computacional
- Há outras estruturas de dados que se podem usar
- A ordem de complexidade do algoritmo depende da estrutura de dados escolhida

Priority Queue

```
// CREATE/DESTROY
```

```
PriorityQueue* PriorityQueueCreate(int capacity, compFunc compF, printFunc printF) ;
```

```
void PriorityQueueDestroy(PriorityQueue** pph) ;
```

```
// MODIFY
```

```
void PriorityQueueInsert(PriorityQueue* ph, void* item) ;
```

```
void PriorityQueueRemoveMin(PriorityQueue* ph) ;
```

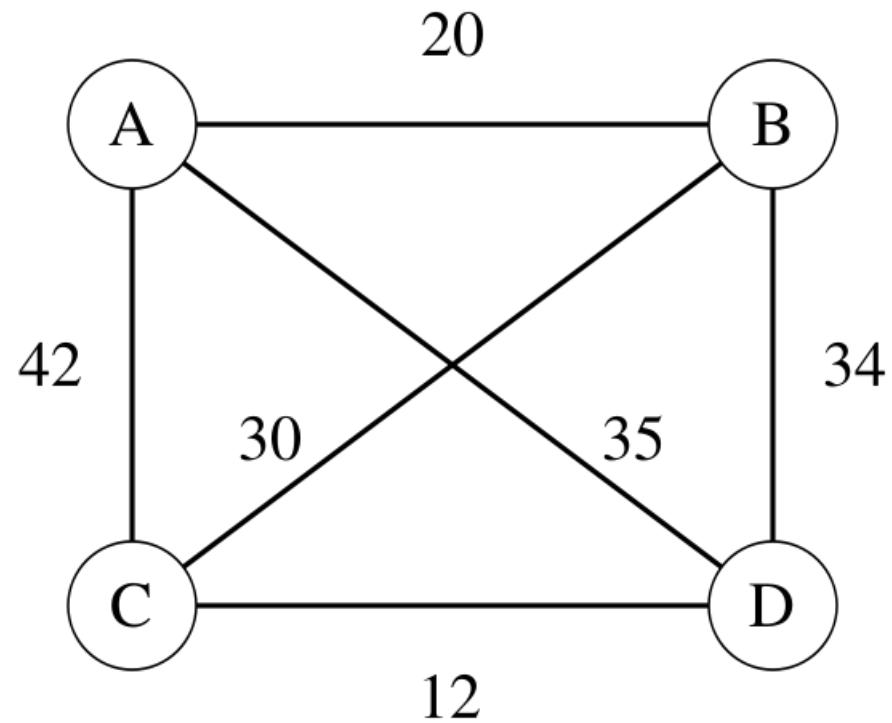
```
void PriorityQueueDecreasePriority(PriorityQueue* ph, void* item) ;
```



Ordem de complexidade – MIN-HEAP binária

- Nº de comparações
- Pior Caso : $O(E \log V)$
- Casos típicos : $O(E \log V)$

Fizeram ?

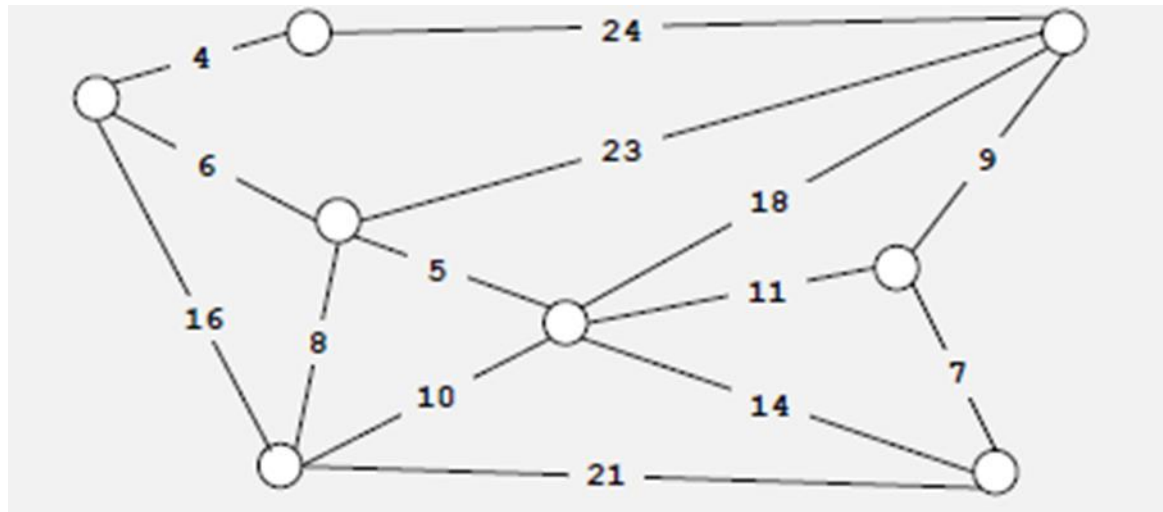


[Wikipedia]

MST - Minimum Spanning Tree

Árvore Geradora

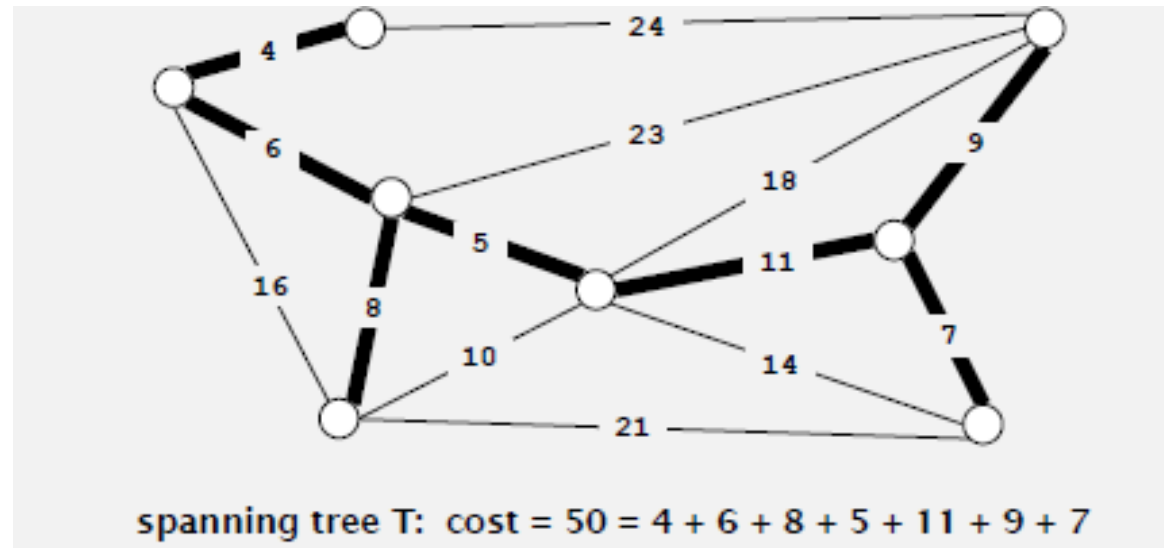
- Dado : um **grafo não orientado** e **conexo** $G(V,E)$, com **custos positivos** associados às arestas
- Definição : uma **árvore geradora** de G é um seu **subgrafo conexo** e **acíclico**, com **V vértices** e **$(V - 1)$ arestas**



[Sedgewick & Wayne]

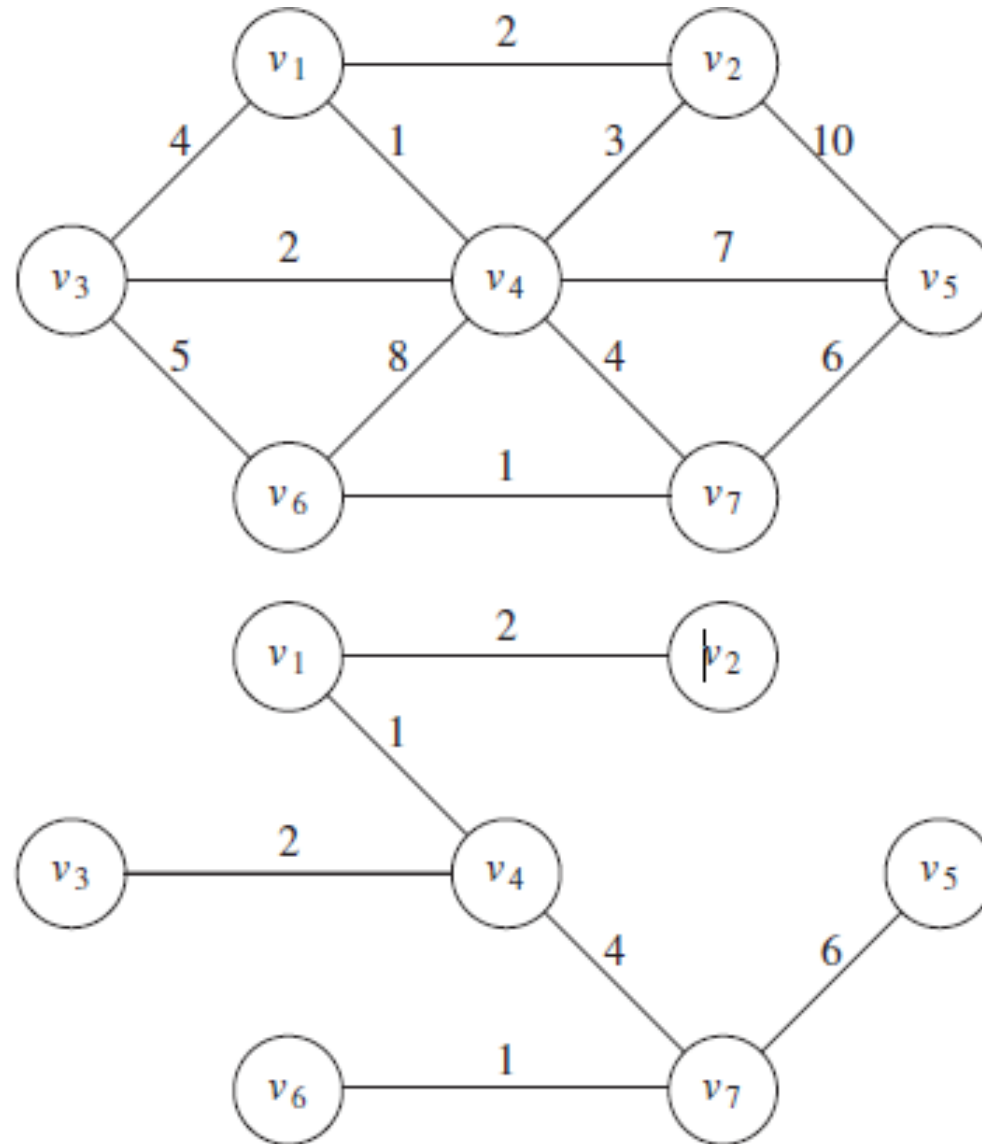
Árvore Geradora de Custo Mínimo

- Problema de **otimização combinatória**
- Determinar a (uma) **árvore geradora** de **custo total mínimo**
 - Soma dos pesos associados às arestas da árvore
 - Assegurar a **conectividade** entre qualquer par de nós com o **menor custo**



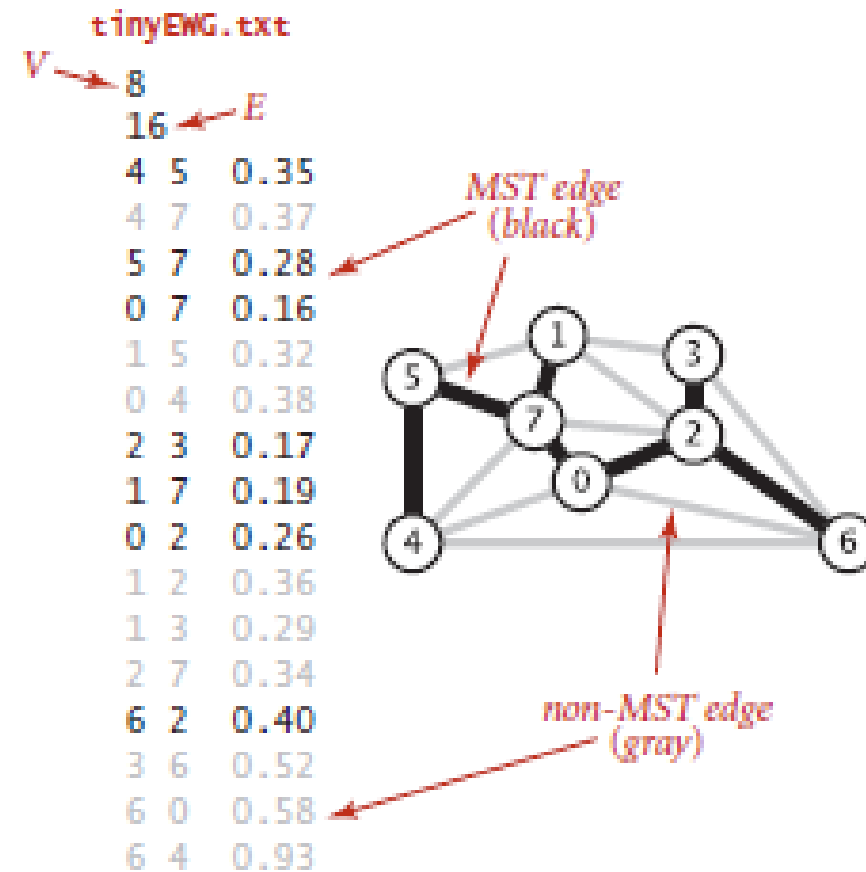
[Sedgewick & Wayne]

Exemplo



[Weiss]

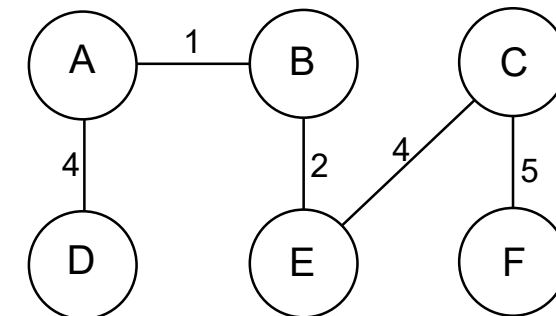
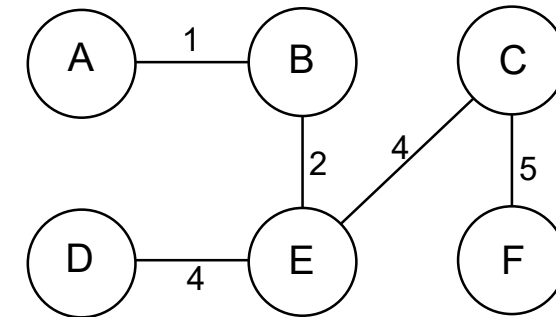
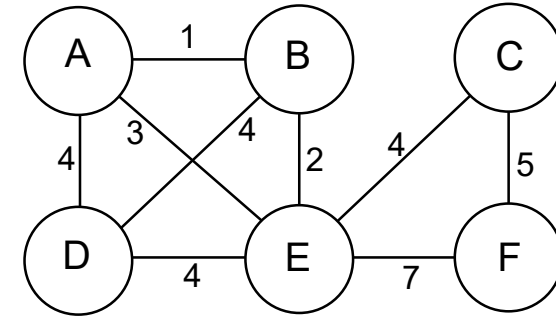
Outro exemplo



[Sedgewick & Wayne]

Soluções ótimas alternativas

- A solução é **única** se cada um das arestas de G tiver um **custo distinto**
- Se todas as arestas tiverem o **mesmo custo**, cada uma das árvores geradoras é ótima
- **Soluções ótimas alternativas**: mais do que uma árvore geradora de custo mínimo



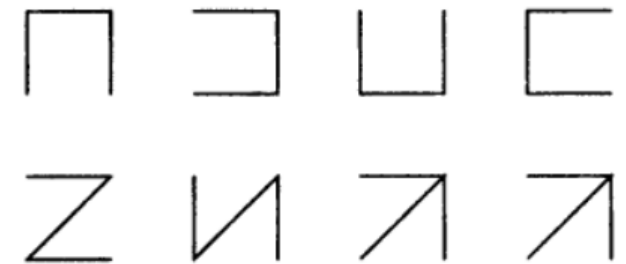
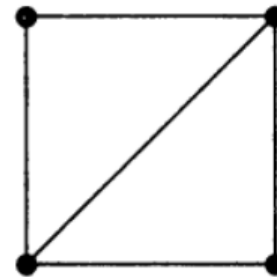
[Wikipedia]

Aplicações

- Projeto de redes de comunicações, elétricas, etc.
- Encaminhamento em redes de computadores
- Detecção de redes viárias em imagens de satélite
- Registo e segmentação de imagens
- Verificação facial em tempo-real
- Modelação de mercados financeiros
- ...

Como fazer ?

- **Estratégia exaustiva** : gerar todas as possíveis árvores geradoras
escolher a (uma) árvore de custo mínimo
- Para um dado grafo $G(V, E)$, quantas árvores geradoras existem ?
- Ordem de complexidade ?
- **Alternativas** ?
- Usar uma estratégia voraz (“greedy”) !!



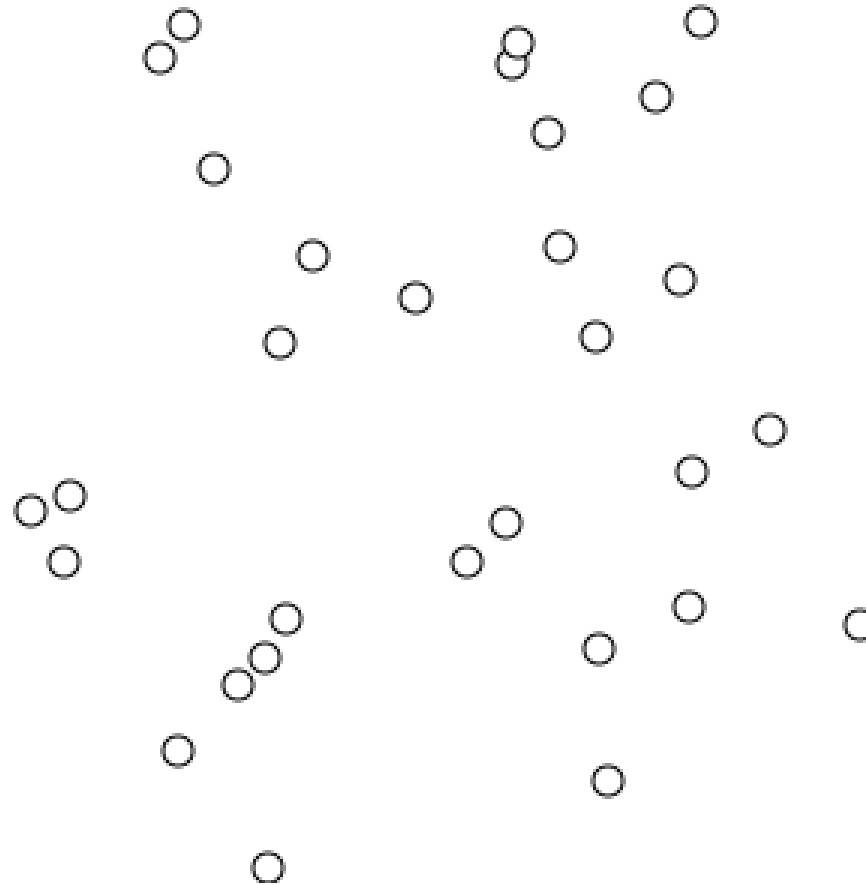
[Adam Sheffer]

O Algoritmo de Kruskal

Algoritmo de Kruskal

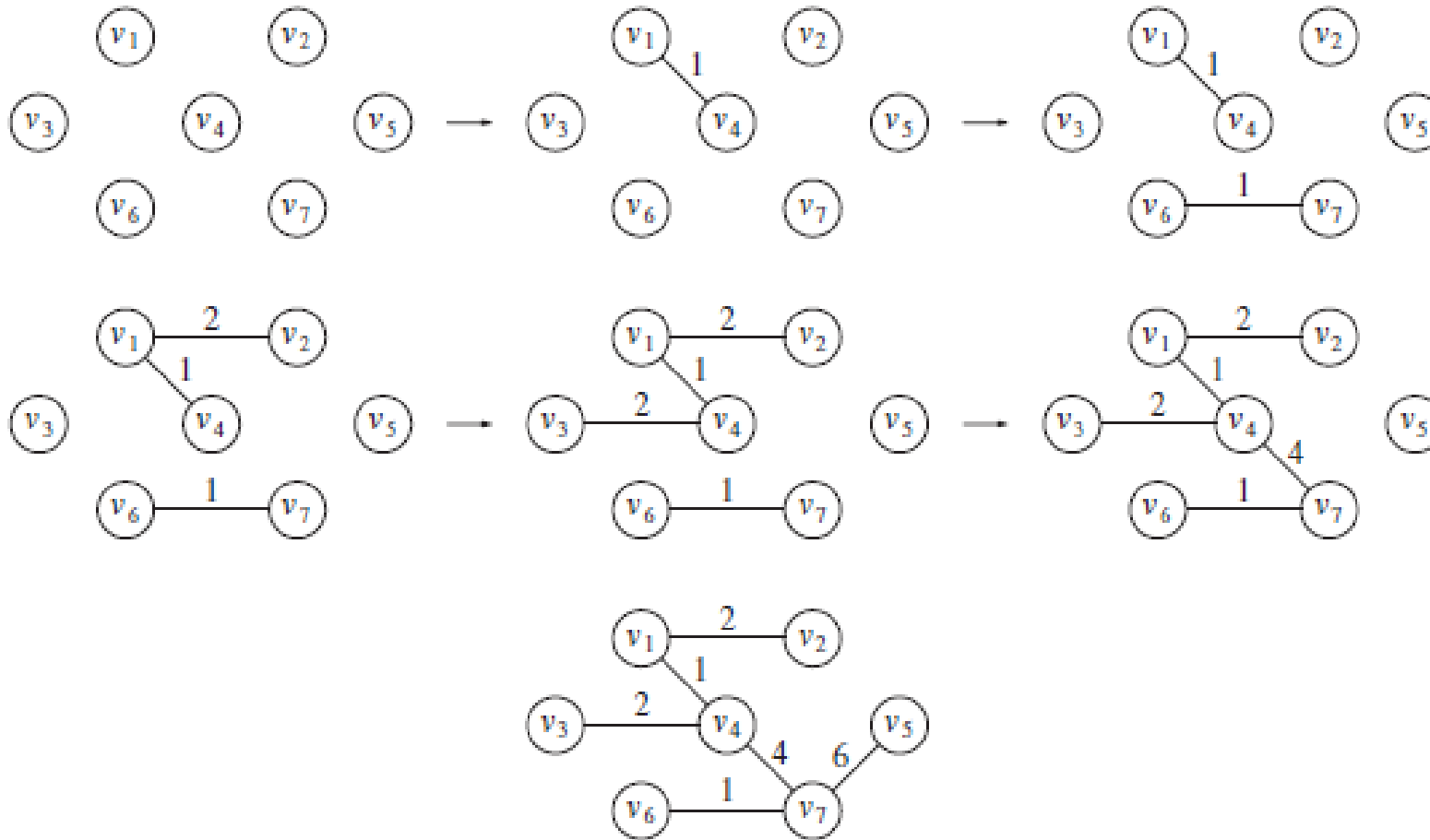
- Começar com uma **floresta** de **árvores**, cada uma com **um só vértice**
- Sucessivamente adicionar uma **aresta de menor custo** que **não** origina um **ciclo**
 - **Reunião** de duas **árvores**
 - Como verificar que não se forma um ciclo ?
- Pré-processamento : construir a **lista ordenada de arestas**
 - **$O(E \log E)$**
- Ordem de complexidade : $O(E \log E) + ? = ?$

Exemplo – Distância Euclideana



[Wikipedia]

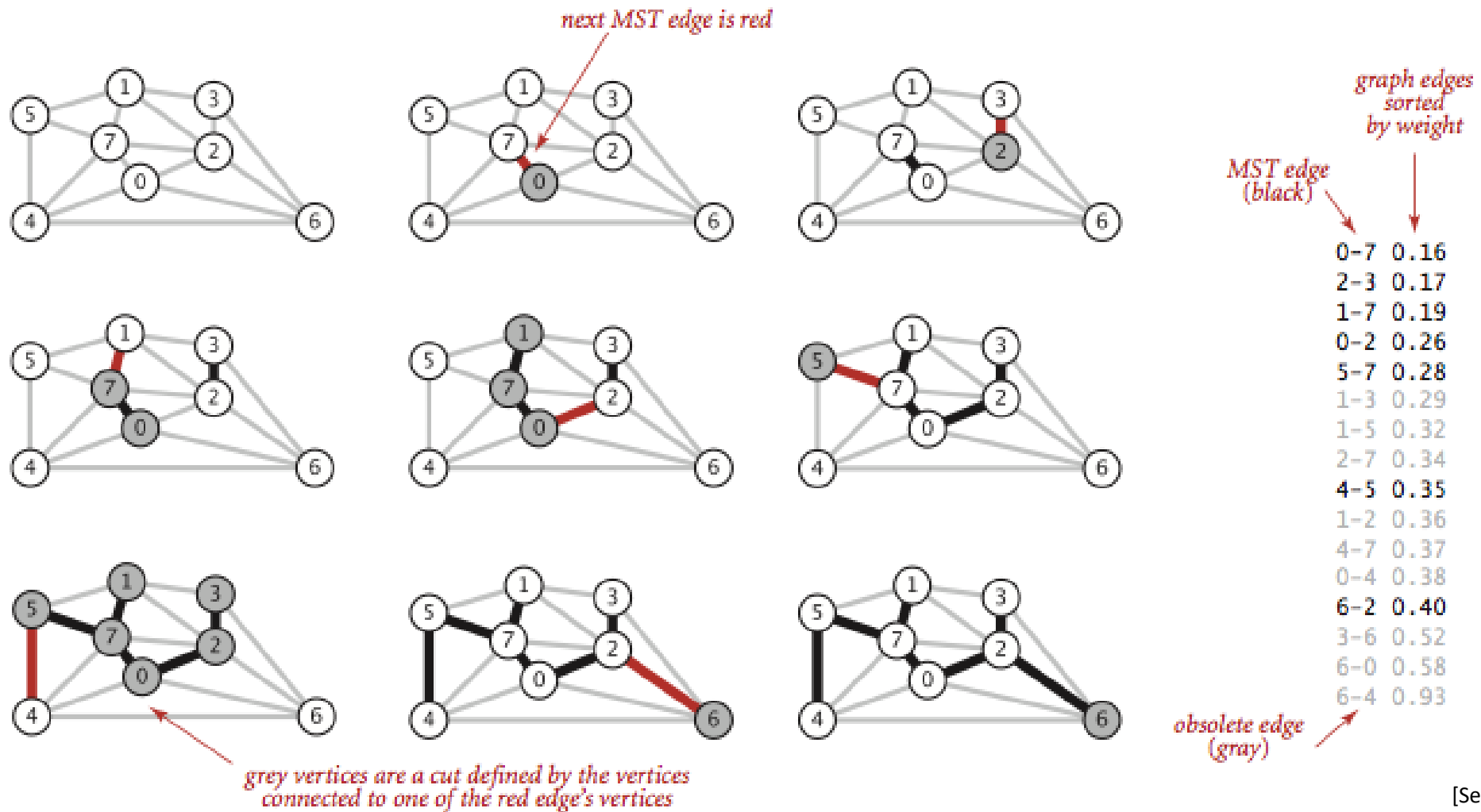
Exemplo



Edge	Weight	Action
(v_1, v_4)	1	Accepted
(v_6, v_7)	1	Accepted
(v_1, v_2)	2	Accepted
(v_3, v_4)	2	Accepted
(v_2, v_4)	3	Rejected
(v_1, v_3)	4	Rejected
(v_4, v_7)	4	Accepted
(v_3, v_6)	5	Rejected
(v_5, v_7)	6	Accepted

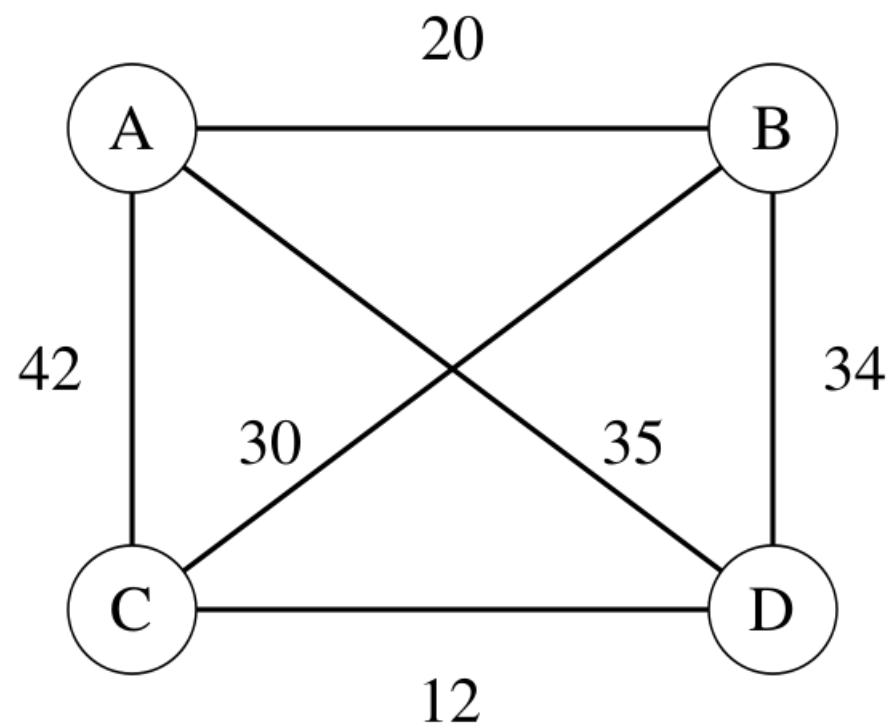
[Weiss]

Outro exemplo



[Sedgewick & Wayne]

Tarefa : aplicar o algoritmo de Kruskal

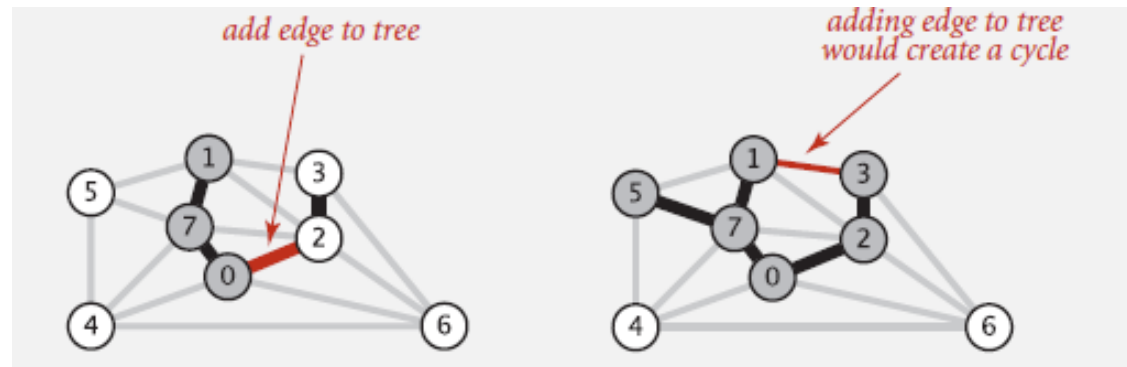


[Wikipedia]

- Qual é a solução ?

Como verificar que não se forma um ciclo ?

- Adicionar a aresta (v, w) forma um **ciclo** ?



[Sedgewick & Wayne]

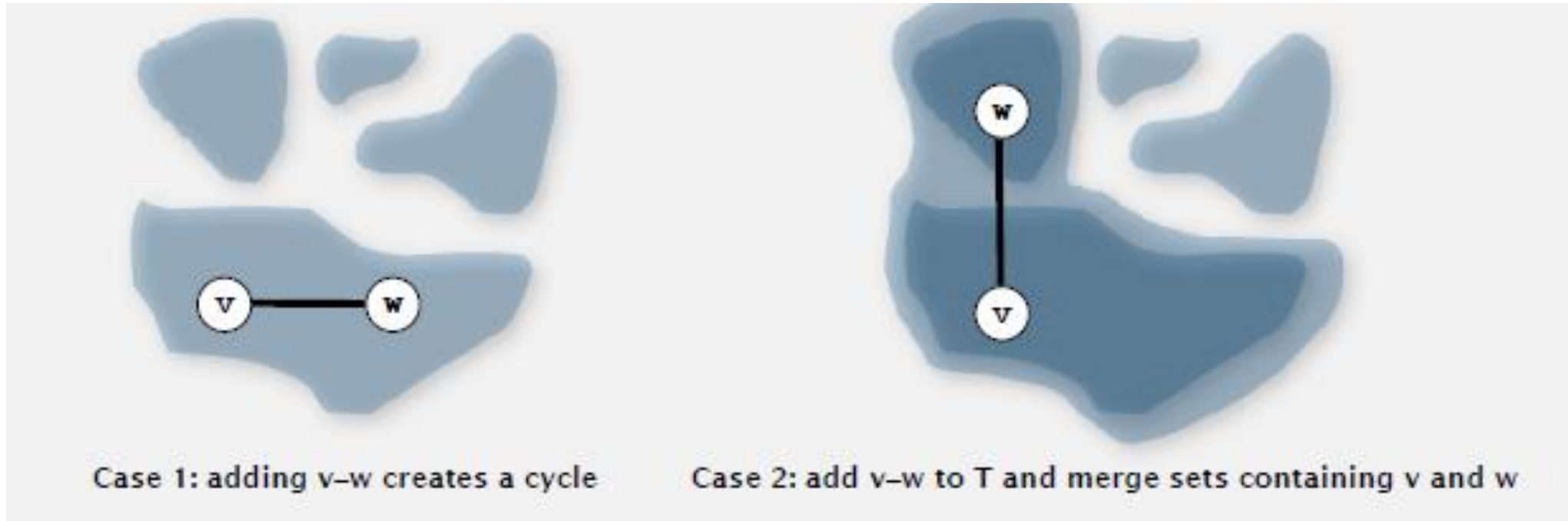
- Ver se w é alcançável a partir de v , na árvore T que contém v
- Executar a travessia em profundidade !!
- T tem, quando muito, V vértices

-> **$O(V)$**

A estrutura de dados UNION-FIND

- Alternativa mais eficiente !!
- Usar a estrutura de dados UNION-FIND
 - Rápida reunião de conjuntos de elementos
 - Fácil verificação da pertença
- Manter um conjunto de vértices para cada árvore da floresta
- Se v e w pertencem ao mesmo conjunto, não considerar a aresta !!
- Se pertencem a conjuntos distintos, efetuar a sua reunião !!

A estrutura de dados UNION-FIND




[Sedgewick & Wayne]

Esforço computacional

operation	frequency	time per op
build pq	1	E
del min	E	$\log E$
union	V	$\log^* V \dagger$
find	E	$\log^* V \dagger$

\dagger amortized bound using weighted quick union with path compression



[Sedgewick & Wayne]

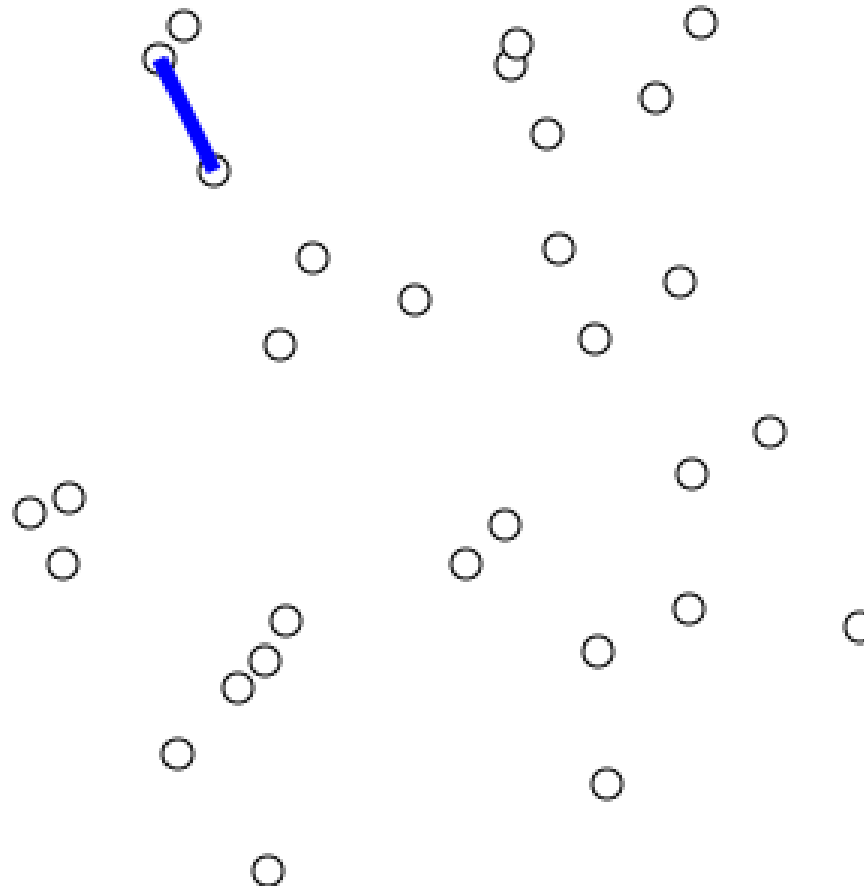
- Usar uma PRIORITY-QUEUE para manter o conjunto de arestas
- O algoritmo de Kruskal determina a MST em $O(E \log E)$
- Se as arestas já estiverem ordenadas, obtém-se $O(E \log V)$

O Algoritmo de Prim

Algoritmo de Prim

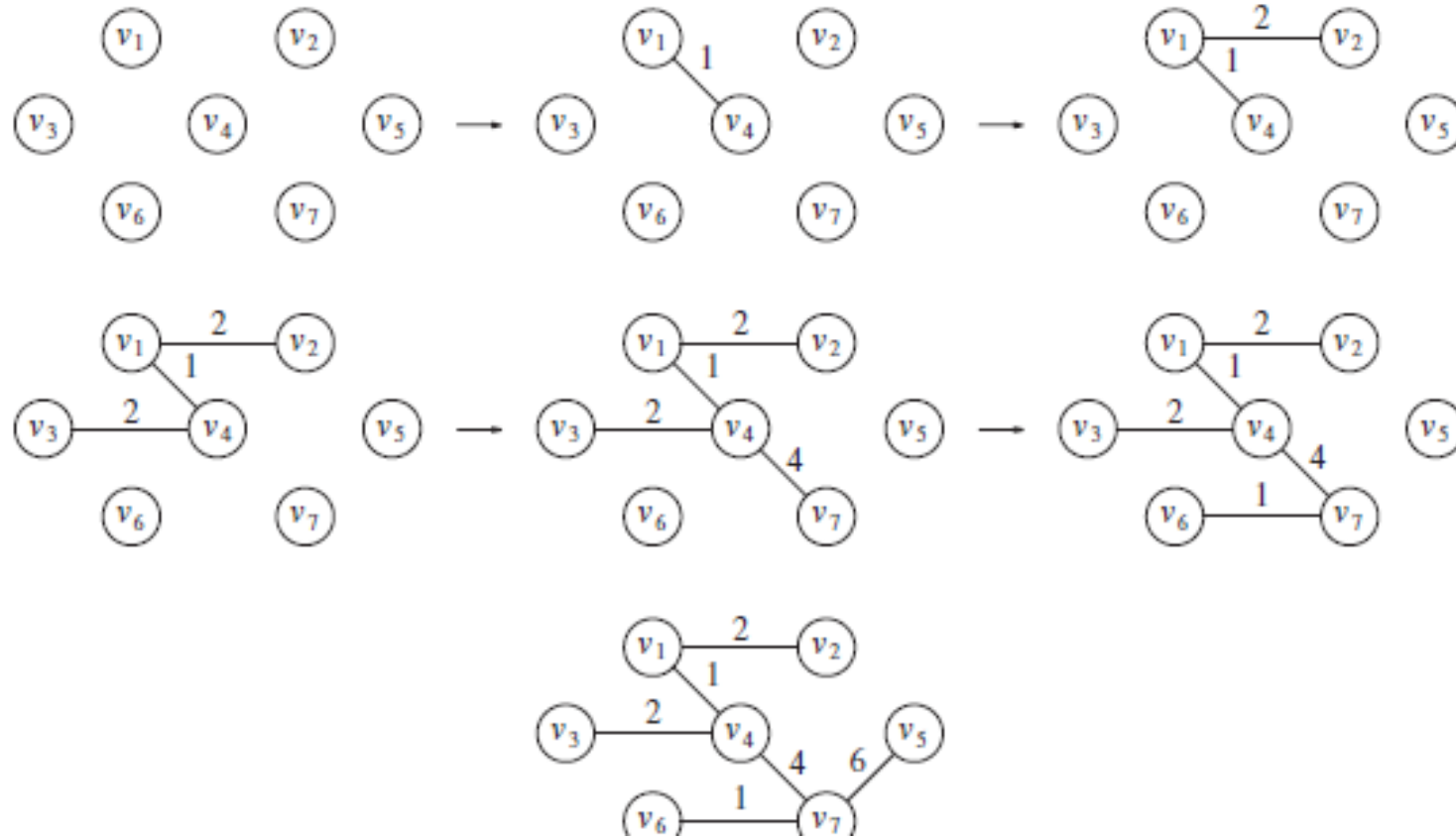
- Começar com **uma árvore** T , com **um só vértice** de G
- Sucessivamente **adicionar** uma aresta a T : a **aresta mais curta** com (apenas) um dos seus vértices em T
 - Não se cria um ciclo **!!**
- Manter o conjunto de **arestas candidatas**
- Usar uma **PRIORITY-QUEUE**
- Semelhanças com o Algoritmo de Dijkstra **?**

Exemplo – Distância euclidiana



[Wikipedia]

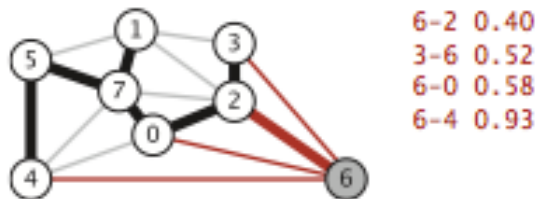
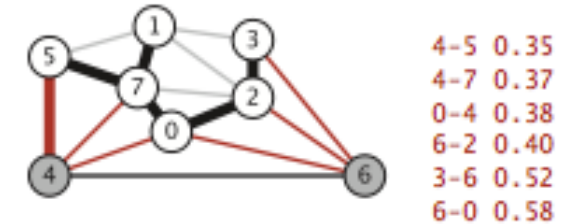
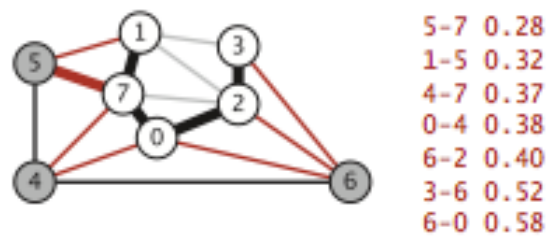
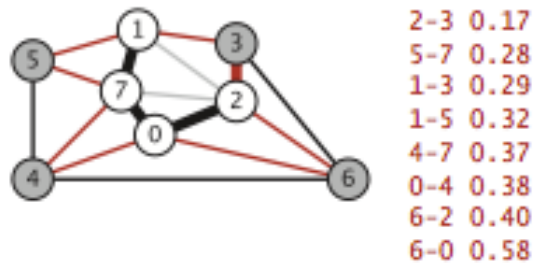
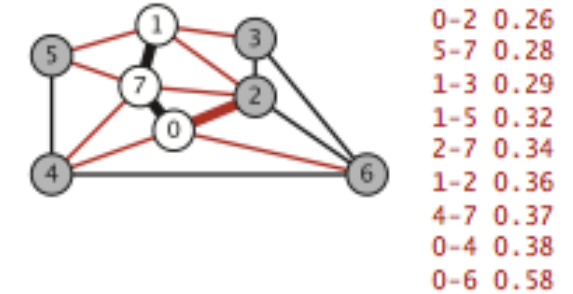
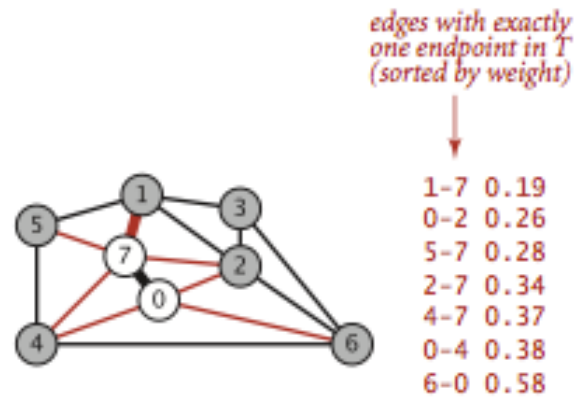
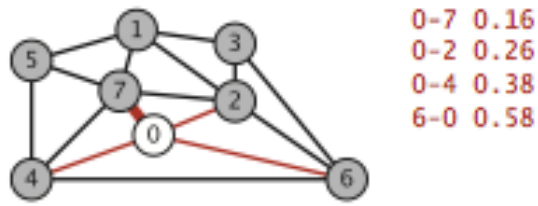
Exemplo



v	$known$	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	T	2	v_4
v_4	T	1	v_1
v_5	T	6	v_7
v_6	T	1	v_7
v_7	T	4	v_4

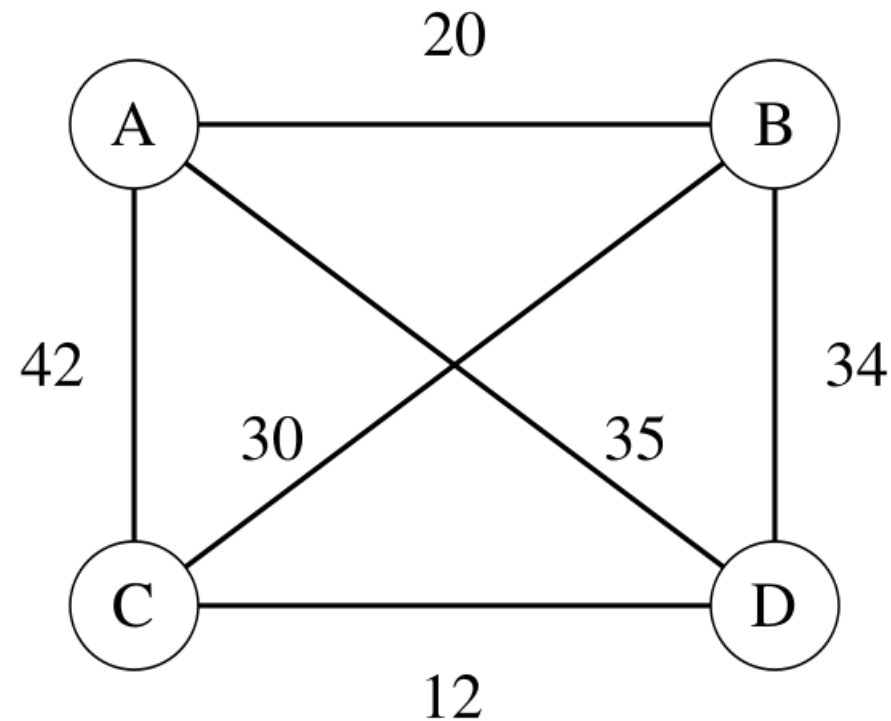
[Weiss]

Outro exemplo



[Sedgewick & Wayne]

Tarefa : aplicar o algoritmo de Prim

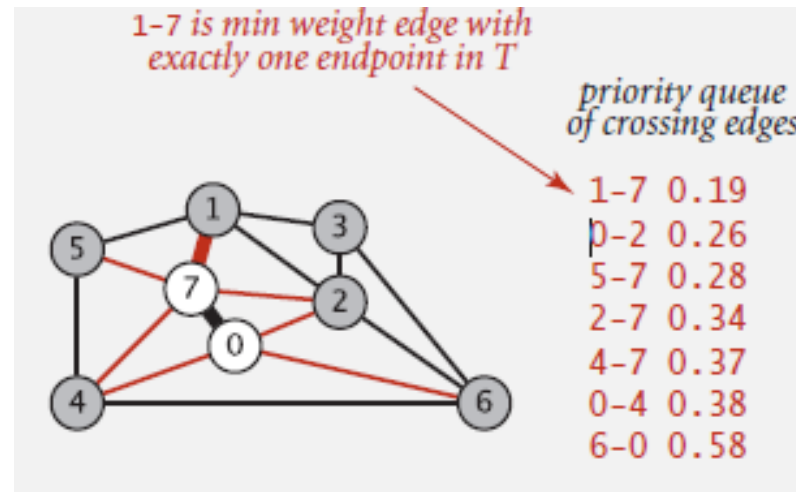


[Wikipedia]

- Qual é a solução ?

Como encontrar a próxima aresta ?

- Aresta (v, w) de menor custo com um vértice em T ?



[Sedgewick & Wayne]

- Verificar todas as arestas ?
- Usar uma PRIORITY-QUEUE

-> $O(E)$

-> $O(\log E)$

Esforço computacional

operation	frequency	binary heap
delete min	E	$\log E$
insert	E	$\log E$

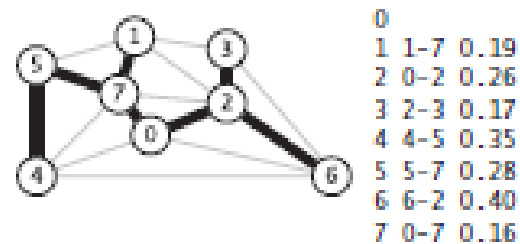
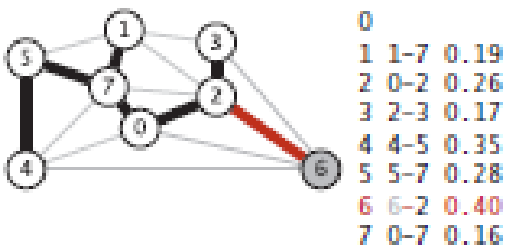
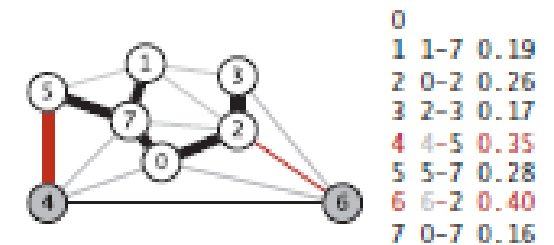
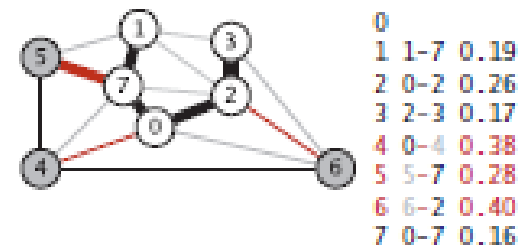
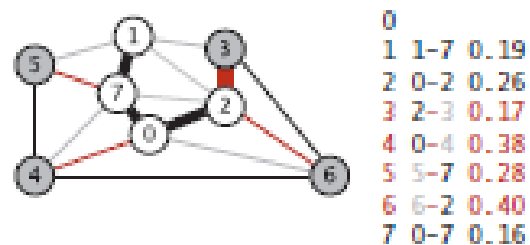
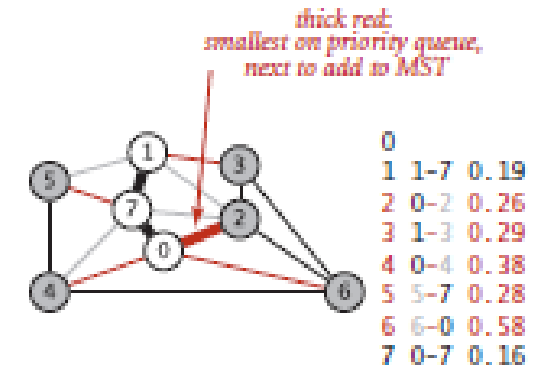
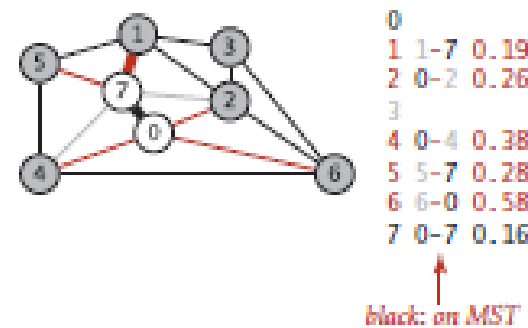
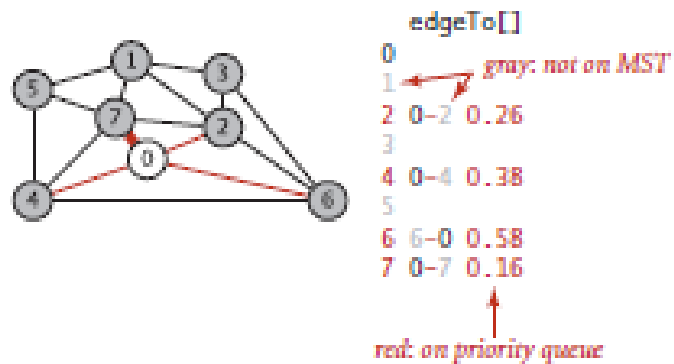
[Sedgewick & Wayne]

- Usar uma **PRIORITY-QUEUE** para manter o conjunto de arestas
- O algoritmo de Prim determina a MST em $O(E \log E)$

Alternativa : conjunto de vértices candidatos

- Começar com uma árvore T , com um só vértice de G
- Sucessivamente adicionar um novo **vértice** e uma nova aresta a T : o **vértice mais próximo** de qualquer um dos vértices em T
 - Não se cria um ciclo !!
- Manter o **conjunto de vértices candidatos**
- Usar uma PRIORITY-QUEUE
 - Que custo associar a cada vértice ?
- Semelhanças com o Algoritmo de Dijkstra ?

Estratégia alternativa



[Sedgewick & Wayne]

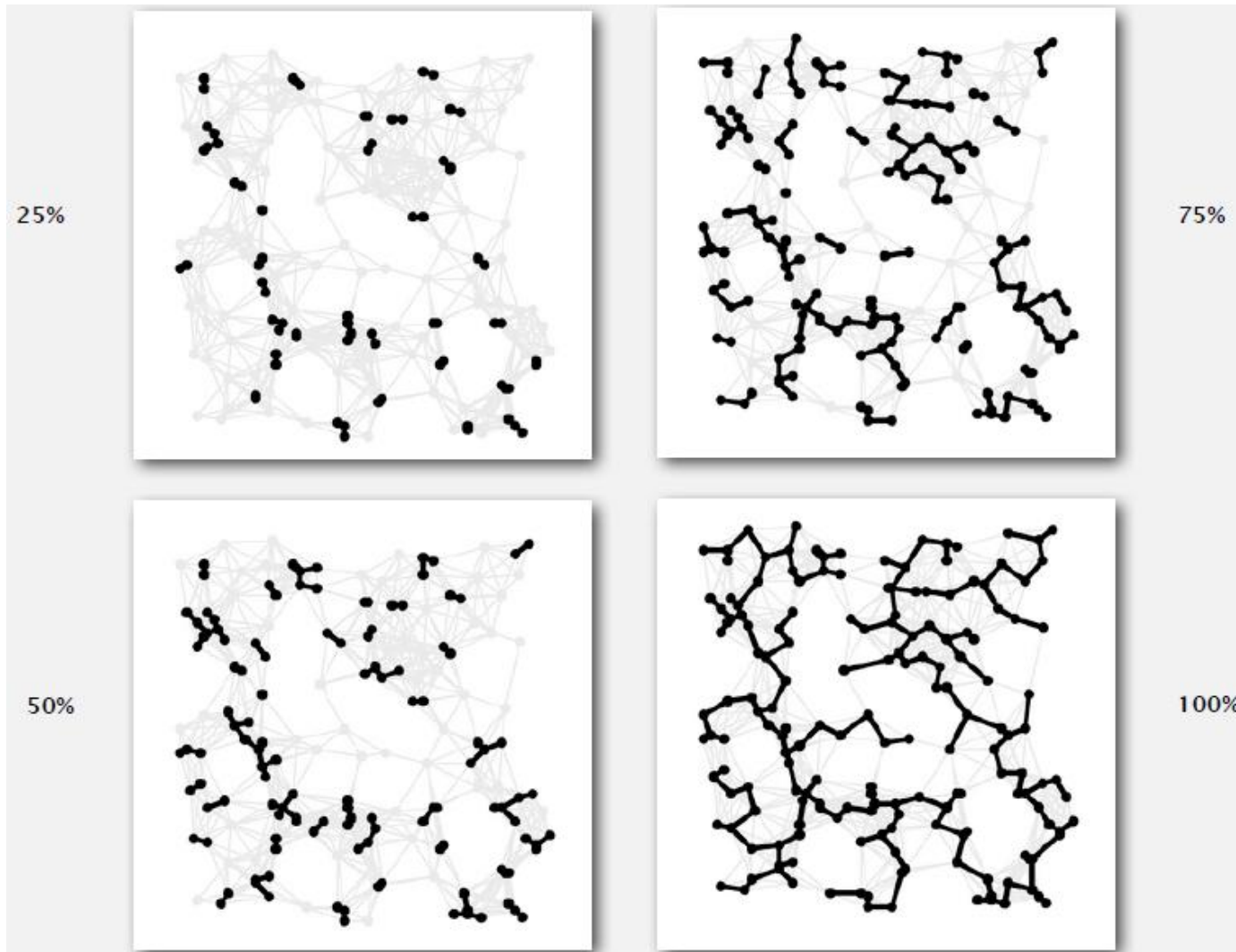
Esforço computacional

PQ implementation	insert	delete-min	decrease-key	total
array	1	V	1	V^2
binary heap	$\log V$	$\log V$	$\log V$	$E \log V$

[Sedgewick & Wayne]

- Grafo **denso** : array
- Grafo **esparso** : heap binária

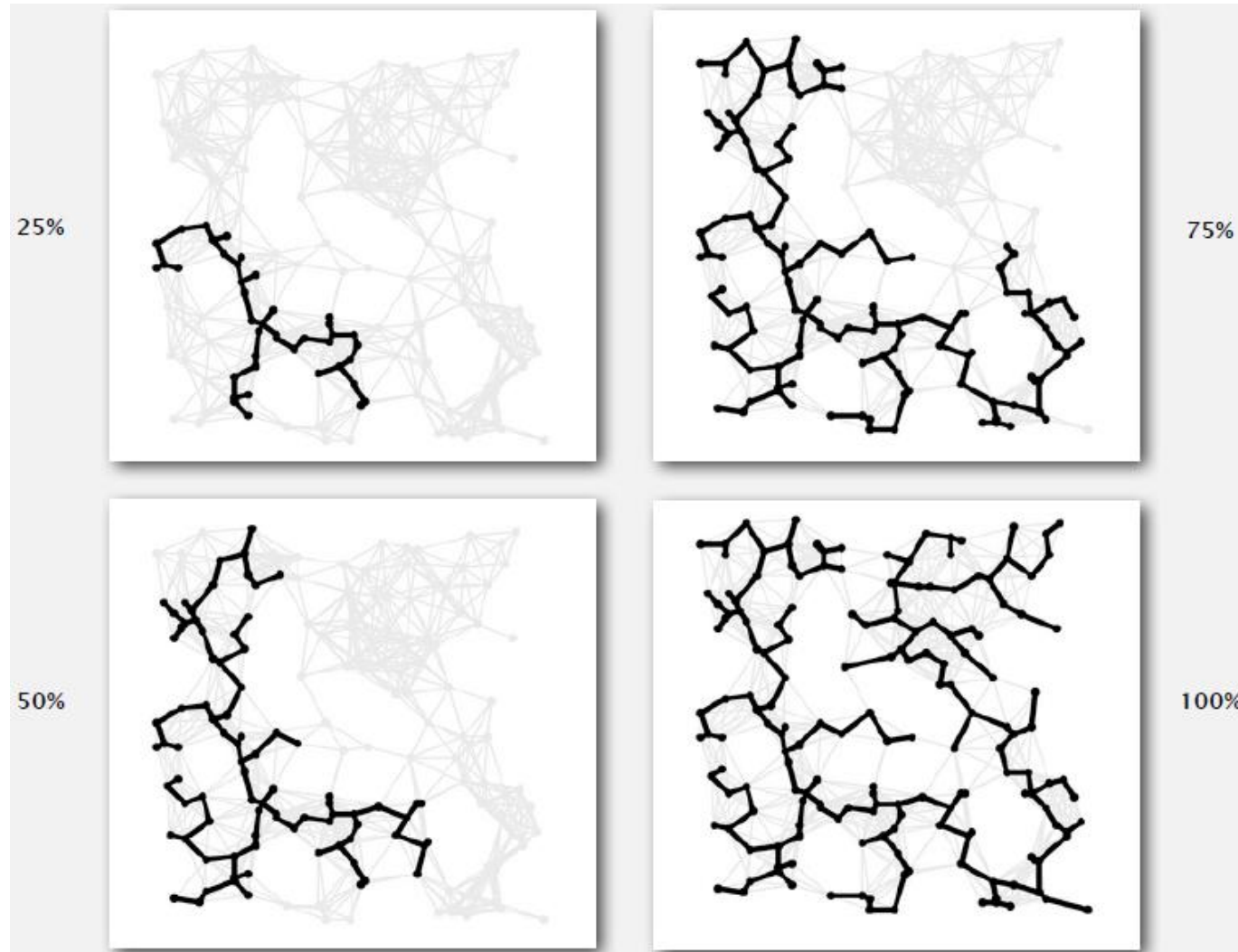
Prim ou Kruskal ?



Kruskal

[Sedgewick & Wayne]

Prim ou Kruskal ?



Prim

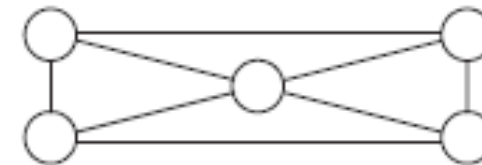
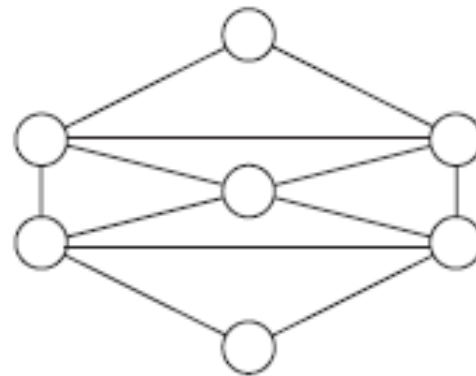
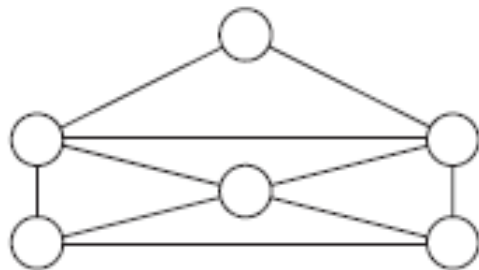
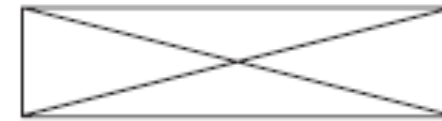
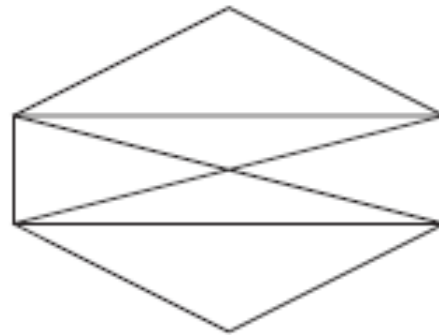
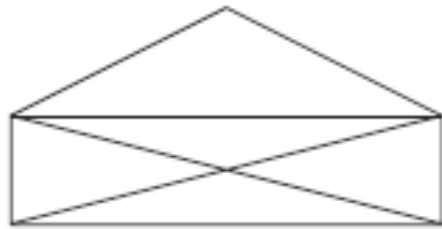
[Sedgewick & Wayne]

Tarefa

- Analisar os exemplos nos ficheiros adicionais

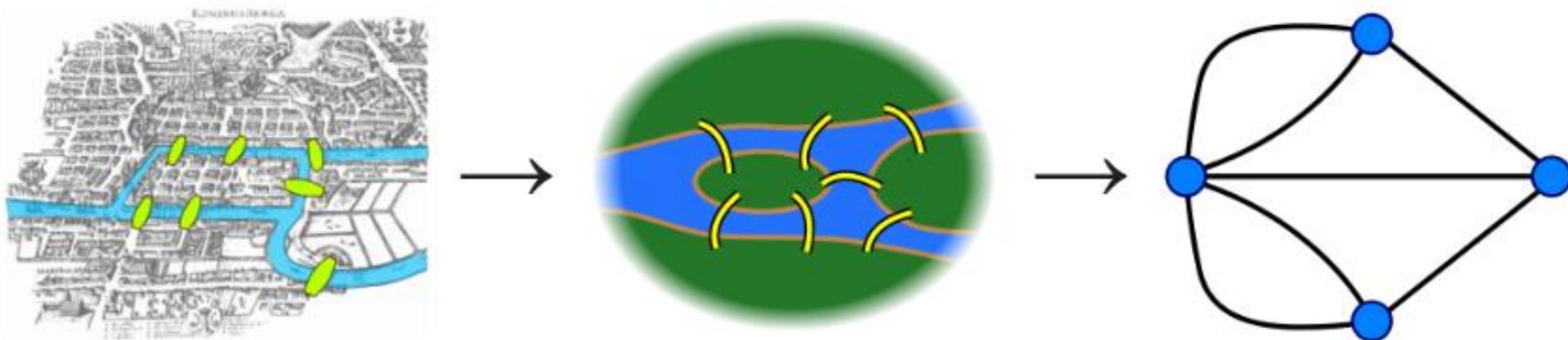
Caminhos e Circuitos Eulerianos

Desenhar sem levantar a ponta do lápis ?



[Weiss]

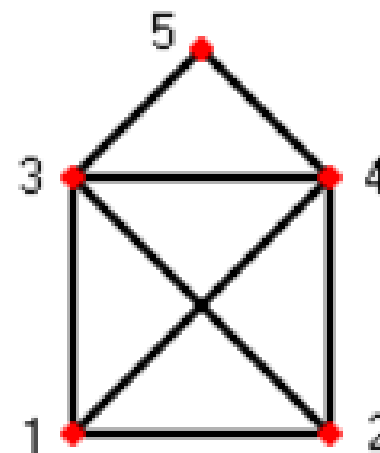
Problema das Pontes de Königsberg (1736)



[Wikipedia]

Caminho Euleriano / Circuito Euleriano

- Grafo / Grafo orientado
 - **Caminho** que contém uma única vez cada uma das arestas de um grafo
 - **Circuito** que contém uma única vez cada uma das arestas de um grafo
 - Qual é a **sequência de arestas** ?
-
- Há algum caminho Euleriano ?
 - Há algum circuito Euleriano ?



[Wikipedia]

Propriedades – Tarefa : arranjar exemplos !

- **Grafo conexo**
- Um grafo tem um **circuito Euleriano** sse cada vértice for um vértice de grau par
- Um grafo tem um **caminho Euleriano** sse exatamente zero ou dois vértices tiverem grau ímpar

Propriedades – Tarefa : arranjar exemplos !

- **Grafo orientado fortemente conexo**
- Um grafo orientado tem um **circuito Euleriano** sse cada vértice tiver o mesmo in-degree e out-degree
- Um grafo orientado tem um **caminho Euleriano** sse quando muito um vértice tem **out-degree – in-degree = 1**, quando muito um vértice tem **in-degree – out-degree = 1**, e cada um dos outros vértices tem o mesmo in-degree e out-degree


Aplicações

- Projeto de circuitos CMOS
- Reconstrução de sequências de ADN
- ...

Circuitos Eulerianos

– O Algoritmo de Fleury

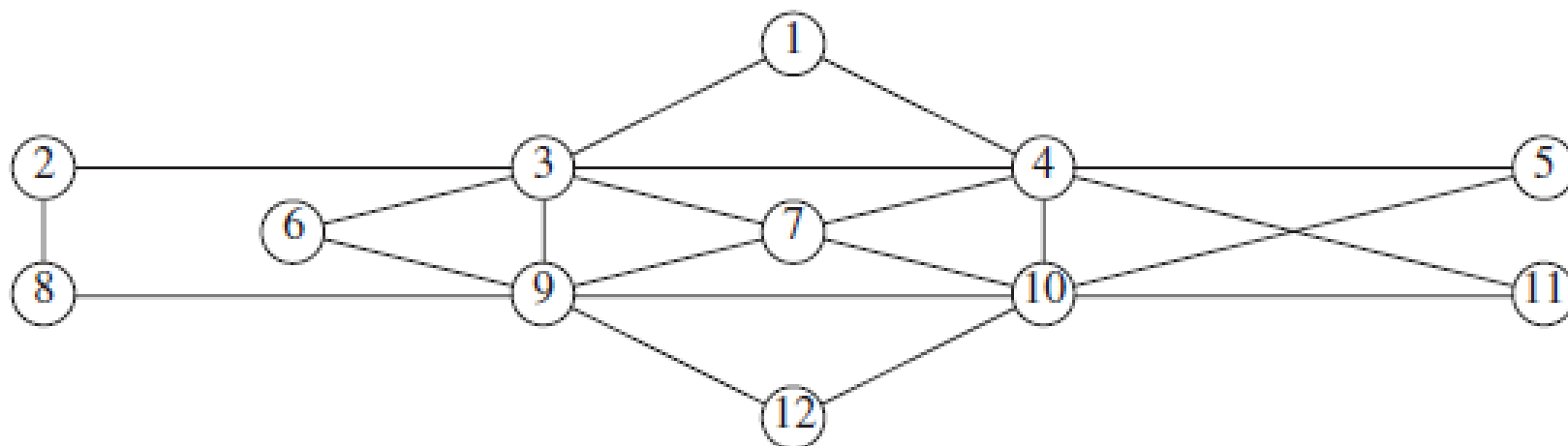
Algoritmo de Fleury (1883)

- Assegurar que cada vértice tem grau par 
- v = escolher um qualquer vértice inicial
- Em v , escolher a próxima aresta (v, w) da solução
Condição : a escolha de (v, w) não torna o grafo desconexo, a menos que seja a única escolha possível – ponte / istmo ?
Adicionar (v, w) à solução
 $v = w$
Apagar (v, w)
Apagar v , se é agora um vértice isolado

Problemas ?

- Trabalhar sobre uma **cópia** do grafo dado
- Como **verificar** se uma aresta é um **istmo / ponte** e não pode ser apagada sem tornar o grafo desconexo ?
- Remover **tentativamente** a aresta e verificar se os outros vértices continuam a ser alcançáveis
 - Sucessivas **travessias em profundidade**, por exemplo
- **$O(E \times E)$**
- Há algoritmos alternativos mais eficientes...

Tarefa : aplicar o algoritmo



[Weiss]

Sugestões de Leitura

Sugestões de leitura

- M. A. Weiss, *“Data Structures and Algorithm Analysis in C++”*, 4th. Ed., Pearson, 2014
 - Chapter 9
- R. Sedgewick and K. Wayne, *“Algorithms”*, 4th. Ed., Addison-Wesley, 2011
 - Chapter 4